VOLLDAMPF

24 C3 | VORAUS

Für'n Käpt'n.

Volldampf voraus!
24. Chaos Communication Congress

Tagungsband

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http//dnb.d-nb.de/ abrufbar.

24C3 Tagungsband Volldampf voraus!
27. - 30. Dezember 2007, Kongreßhalle am Alexanderplatz, Berlin.

24. Chaos Communication Congress Eine Veranstaltung des Chaos Computer Clubs.
http://events.ccc.de/congress/2007/

Umschlag: **evelyn & hukl** (Cover) sowie **Marten** (Rücken)
Satz: **wetterfrosch**

Schrift: **Yanone Kaffeesatz** von Jan Gerner, lizensiert unter Namensnennung 2.0 Deutschland.

Herausgeber: **Matthias Mehldau**
Verlag: **Art d'Ameublement** Marktstraße 18 in 33602 Bielefeld
Vertrieb: **FoeBuD e.V. Unterstützungsbedarf** Marktstraße 18 in 33602 Bielefeld http://shop.foebud.org/
ISBN-13: 978-3-934636-06-4

Programmierung der Vorträge
unter dem sympathisch herrschendem Schirm der Wau-Holland-Stiftung.

1. Auflage, 400 Stück.
Alle bis zum 17. Dezember 2007 eingereichten Papers. Stand des Fahrplans vom 1. Dezember 2007.
Herstellung: **copy print Kopie & Druck GmbH** Berlin
2. Auflage, on Demand geplant
Herstellung: **Books on Demand GmbH** Norderstedt bod.de-ID: 0005147212

Inhaltsverzeichnis

Volldampf voraus!

24. Chaos Communication Congress

Papers

Science
lecture
Tag 2 12:45
Saal 2
de

Anoushirvan Dehghani

Absurde Mathematik

Paradoxa wider die mathematische Intuition

Ein kleiner Streifzug durch die Abgründe der Mathematik. Eigentlich ist der Mensch mit einer recht gut funktionierenden Intuition ausgerüstet. Dennoch gibt es Paradoxa, welche mathematisch vollkommen korrekt und beweisbar sind, jedoch unserer Intuition widersprechen. Der Vortrag bietet einen Streifzug durch einige dieser Paradoxa, die kurz und anschaulich erklärt werden.

Nicht alles, was mathematisch beweisbar ist, ist auch intuitiv und verständlich zu erfassen. Wie kann beispielsweise ein einfacher Körper wie Gabriels Horn ein begrenztes Volumen, aber eine unendlich große Oberfläche haben? Oder warum ist es bei einem Triell, einem Duell mit drei Schützen, als schlechter Schütze für das eigene Überleben von Vorteil, wenn man als letztes schießen darf? Woher kommt das Braess'sche Paradoxon, bei dem die Verbesserung eines Verkehrsstreckenabschnittes zum Zusammenbruch des gesamten Verkehrsflusses führen kann? Wie kann bei Penney-Ante ein unfaires Spiel entstehen, wo doch eine absolut faire Münze geworfen wird?Und wie lief das genau mit dem bekannten Ziegenproblem, soll man sich nach Öffnen der ersten Tür mit der Niete zwischen den anderen beiden Türen umentscheiden?

Absurde Mathematik

Anoushirvan Dehghani

4. Dezember 2007

Zusammenfassung. *Ein kleiner Streifzug durch die etwas absurderen und paradoxen Seiten der Mathematik. Es werden Beweise gezeigt, die der menschlichen Intuition oder einfach nur sich selbst widersprechen. Wo es möglich ist, sollen die Paradoxa auch aufgelöst werden.*

1 Gabriels Horn

Ein seit der Neuzeit bekanntes mathematisches Paradoxon ist *Gabriels Horn*. Nach seinem Entdecker Evangelista Torricelli [1] wird es auch *Toricellis Trompete* genannt.

Es handelt sich dabei um der in Abb. 1 gezeigten Rotationskörper, der durch eine Drehung des Graphen von $y = \frac{1}{x}$ für alle $x \geq 1$ um die x-Achse erzeugt wird.

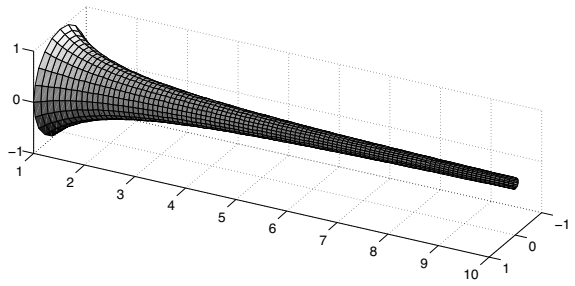


Abbildung 1: Anfangsverlauf von Gabriels Horn

Dieser recht simpel aussehende Körper hat eine seltsame Eigenschaft. Die Berechnung seines Volumens ergibt einen endlichen Wert:

$$V = \int_1^\infty \frac{\pi}{x^2}\,\mathrm{d}x = \pi\left[-\frac{1}{x}\right]_1^\infty = \pi[0-(-1)] = \pi \quad (1)$$

Anders hingegen sieht es aus, wenn die Oberfläche bestimmt werden soll:

$$A = \int_1^\infty 2\pi y \cdot \sqrt{1+y'^2}\,\mathrm{d}x = 2\pi \int_1^\infty \frac{\sqrt{1+\frac{1}{x^4}}}{x}\,\mathrm{d}x$$
$$\geq 2\pi \int_1^\infty \frac{1}{x}\,\mathrm{d}x = 2\pi\,[\ln(x)]_1^\infty = \infty \quad (2)$$

[1] * 15. Oktober 1608 in Faenza, IT; † 25. Oktober 1647 in Florenz, IT.

Dieser Körper hat also eine unendlich große und dennoch glatte Oberfläche[2], jedoch ein nur endlich großes Volumen! Anschaulich gesagt: Entspricht eine Maßeinheit $10\ \mathrm{cm}$, so reichen etwas mehr als drei Liter Farbe aus, um das Horn komplett zu füllen. Jedoch würde sich niemals genug Farbe finden, um die $\infty\ \mathrm{qm}$ große Oberfläche anzustreichen - und dies, obwohl das Horn doch bereits komplett mit Farbe gefüllt ist!

Die Erklärung dieses Paradoxons liegt an den unterschiedlichen Dimensionen der Oberfläche und des Volumens. Die Integration eines Rotationskörpers kann als stückweise Addition kurzer zweidimensionaler Ring- bzw. dreidimensionaler Scheibchensegmente angenähert werden. Deren Radius entspricht dabei jeweils dem momentanen Funktionswert von $y = \frac{1}{x}$.

Werden diese Segmente infinitesimal kurz gehalten, so ergeben sich eindimensionale Ringstreifen bzw. zweidimensionale Kreise. Wächst nun x über alle Grenzen, so gilt:

$$2\pi\frac{\sqrt{1+\frac{1}{x^4}}}{x} \gg \pi\frac{\pi}{x^2} \qquad \text{für} \quad x \to \infty \qquad (3)$$

Das wachsende x geht also nur reziprok linear in die Größe der Ringstreifen ein, während es für die Fläche der Kreise zu einem quadratischen Absinken führt. Dies führt einerseits zu dem existierenden Grenzwert π, andererseits zu dem unbegrenzten Wachstum der Oberfläche.

Die praktische Durchführung eines „Befüll-Experimentes“ scheitert daran, dass die Herstellung eines solchen, unendlich langen Objektes nicht so recht gelingen mag. Unabhängig davon wäre ab einer bestimmten Länge der Horndurchmesser so klein, dass nicht mal mehr ein einziges Molekül oder Atom der verwendeten Füllsubstanz hineinpassen würde.

Merke: Zweidimensionale Oberflächen im dreidimensionalen Raum sind nicht ohne weiteres mit dreidimensionalen Volumina zu vergleichen!

[2] „glatt“ bedeutet hier, dass es nicht um eine fraktale Oberfläche oder ähnliche Taschenspielertricks geht.

2 Efrons intransitive Würfel

Der gesunde Menschenverstand sagt: *Wenn der Porsche meist schneller ist als der Audi, und der Ferrari meist schneller als der Porsche, so wird der Ferrari in der Regel auch den Audi schlagen*. Der Mathematiker spricht hier von einem *transitiven Vorteil*. Dass dies bei einem Glücksspiel mit fairen Würfeln nicht gelten muß, erscheint absurd - und dennoch ist es so!

Die erste Person, die einen Satz solch intransitiver Würfel vorgestellt hat, war Bradley Efron[3]. Die Belegung ist in Abb. 2 dargestellt. *Fair* bedeutet, dass jede Seite eines Würfels die gleiche Auftretenswahrscheinlichkeit von $p = \frac{1}{6}$ besitzt. Seltsam dabei: Spieler 1 darf sich einen beliebigen die-

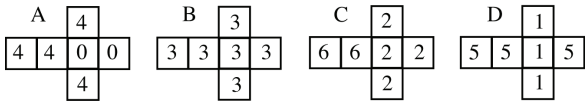


Abbildung 2: Efrons Würfel

ser vier Würfel aussuchen. Spieler 2 kann nun *immer* einen der verbleibenden Würfen so auswählen, dass sein Würfel den von Spieler 1 im statistischen Mittel schlägt. Mathematisch formuliert gilt:

$$P(A > B) = P(B > C) = P(C > D) = P(D > A) = \frac{2}{3} \quad (4)$$

Wird der Wettstreit beispielsweise über zehn Runden gespielt, so gewinnt A über B mit an Sicherheit grenzender Wahrscheinlichkeit. Genauso B über C. Und C über D. Und D über A - womit das Bild eines Treppenhauses im Stile von Escher[4] vor Augen rückt.

Wie kommt dieses Phänomen zustande? Die Betrachtung der Erwartungswerte, also der statistischen Mittelwerte, bringt keinen Hinweis: $E[A] = \frac{16}{6}$, $E[B] = 3$, $E[C] = \frac{20}{6}$, $E[C] = 3$. Aufschlussreicher ist dagegen ein Blick auf die *bedingten Wahrscheinlichkeiten*. Bei diesem direkten Vergleich zeigt sich, dass die Abstufungen der Würfel genau so gewählt sind, dass sie jeweils ihren „Vorgänger“ gerade eben mit $p = \frac{2}{3}$ schlagen - unter minimalem Einsatz der Mittel, also der Augen auf den Seiten. Anders formuliert: Jeder Würfel ist genau so „eingestimmt“, dass er im Vergleich zu seinem unterlegenen Widerpart in 24 von 36 Fällen überlegen ist. Die dazu verwendeten Ziffern sind dabei so gewählt, dass sich der genannte „Kreislauf“ bilden kann - und damit zu jedem Würfel ein überlegener existiert.

Mittlerweile gibt es eine Reihe weitere Sätze intransitiver Würfel. Der Schönheitsfehler von Würfel B, dessen Wurf rasch langweilig wird, konnte beseitigt werden. Auch mit nur drei Würfeln läßt sich ein intransiver Satz erstellen. Als Fazit bleibt: Die Eigenschaft, der wahrscheinliche Gewinner eines Matches zu sein, muß nicht transitiv sein! Was bei „Stein, Schere, Papier“ willkürlich festgelegt wurde, kann auch mit solidem Regelwerk begründet werden.

3 Penney-Ante

Wo wir gerade bei intransitiven Paradoxa sind: Wie wäre es mit einem einfachen Münzwurf? Die Wahrscheinlichkeit p für Zahl, `Z`, sei dabei genauso hoch wie q, die Wahrscheinlichkeit für Kopf `K`: $p = q = \frac{1}{2}$. Es soll sich dabei um gleichermaßen *faire* wie *gedächtnislose* Münzen handeln. Der Ausgang eines Wurfes ist also nicht von den vorhergehenden Würfen beeinflußt.

Die Regeln des Spieles lauten: Spieler 1 sucht sich eine beliebige Reihe von Münzwürfen der Mindestlänge drei aus, beispielsweise `ZKK` oder `KKZK`. Spieler 2 wählt nun ebenfals eine Wurfreihe aus. Sodann wird die Münze so lange geworfen, bis die Reihe eines der beiden Spieler auftaucht. Wenn Spieler 2 alles richtig anstellt, so wird er *immer* eine Kombination finden, deren Gewinnwahrscheinlichkeit höher ist als die von Spieler 1. Für die genannten Beispiele wären das `ZZK` und `ZKKZ`. Wie kann und darf das sein? Die Wahrscheinlichkeiten sind doch $pqq = ppq = \frac{1}{8}$ bzw. $qqpq = pqqp = \frac{1}{16}$. Oder etwa nicht?

Die Taktik, mit der Walter Penney [6] den wahrscheinlichen Ausgang dieses Spieles zu seinen Gunsten beeinflußt, lautet wie folgt: hat Spieler 1 die folgende Münzreihe der Länge n gewählt

$$m_1 m_2 m_3 \ldots m_n, \quad (5)$$

so setzt Spieler 2 auf die Reihe:

$$\overline{m_2} m_1 m_2 \ldots m_{n-1}. \quad (6)$$

Entscheidend ist hierbei $\overline{m_2}$, welches das Gegenteil von m_2 darstellt: `K` anstatt `Z` und `Z` anstatt `K`. Spieler 2 wählt also für seine letzten $n-1$ Plätze genau die Werte, die Spieler 1 auf den ersten $n-1$ Plätzen hat. Der erste Wert von Spieler 2 ist die Negation des zweiten Wertes von Spieler 1: `K` anstatt `Z` bzw. `Z` anstatt `K`, wie auch in den oben genannten Beispielen geschehen.

Zum Verständnis dieses Sachverhaltes ist ein Zustandsdiagramm wie in Abb. 3 hilfreich. Spieler 1 setzt hier auf `ZKK`, Spieler 2 auf `ZZK`. Die Übergänge entsprechen jeweils dem Ausgang eines Münzwurfes, `K` oder `Z`. Wir beginnen im linken Zustand „Start“. Sobald das erste mal ein `Z` landet, entspricht das der Initialisierung beider Reihen (die jeweils mit `Z` beginnen), und der Zustand A wird erreicht. Je nach dem weiteren Verlauf der Münzwürfe wird früher oder später das Gewinnfeld für Spieler 1 oder Spieler 2 erreicht.

[3] * Mai 1938 in Minnesota, USA.

[4] Nach Maurits Cornelis Escher, * 17. Juni 1898 in Leeuwarden; NL; † 27. März 1972 in Hilversum, NL.

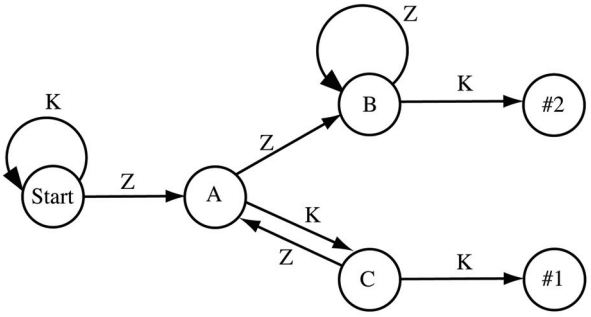


Abbildung 3: Zustandsdiagramm für Zahl-Kopf-Kopf (#1) gegen Zahl-Zahl-Kopf (#2)

Das Zustandsdiagramm erlaubt eine interessante Beobachtung. Mit Erreichen von Zustand B ist das Spiel so gut wie gelaufen, und Spieler 2 der designierte Gewinner. Es gibt nämlich keinen Weg, um von hier aus noch zum Zustand $\#1$ zu gelangen. Aus Zustand C heraus kann hingegen sehr wohl ein Pfad zurück in Richtung Zustand $\#2$ gefunden werden. Das gesamte Spiel wird also in Zustand A schon entschieden! Spieler 2 benötigt hier nur ein einziges Auftreten von `Z`, während Spieler 1 auf ein nur halb so wahrscheinliches `KK` hoffen muß.

Sicher ist es müßig, für jede einzelne Folge von Würfen ein derartiges Zustandsdiagramm zu erstellen. Es läßt sich herleiten, dass die Gewinnwahrscheinlichkeit einer bestimmten Folge A im Vergleich zu einer anderen Folge B wie folgt berechnen läßt:

$$\frac{P(A)}{P(B)} = \frac{B:B - B:A}{A:A - A:B}. \tag{7}$$

Dabei ist $V:W$ definiert als

$$V:W = \sum_{k=1}^{\min l,m} 2^{k-1} \nabla(V_{l-k-1:l} == W_{1:k}). \tag{8}$$

Der $\nabla\cdot$-Operator liefert hier eine eins zurück, falls sein Argument wahr ist, ansonsten eine null. $\nabla(V_{l-k-1:l} == W_{1:k})$ überprüft also, ob die letzten k Symbole von V den letzten k Symbolen von W entsprechen.

Mittlerweile ist dieses Phänomen auch für größere Alphabete, d.h. mehr als nur Kopf und Zahl, bewiesen werden. Ausführlichere Informationen hierzu finden sich in [3], als rasche Einführung leistet [1] gute Dienste.

Als Fazit bleibt zu sagen, dass ein auf den ersten Blick faires Spiel wie Penney-Ante sich bei näherer Betrachtung als ganz und gar nicht fair entpuppt.

4 Das Ziegenproblem

Eine in ihren Grundzügen seit dem späten 19. Jhdt. durch Joseph Bertrand[5] bekannt gewordene mathematische Problemstellung ist das *Ziegenproblem*. Ein größeres Publikuminteresse erlangte es 1990, nachdem Marilyn vos Savant in ihrer Kolumne im amerikanischen *Parade*-Magazin das Thema aufgriff. Auf diesen Artikel hin erhielt sie tausende von Leserbriefen, die ihre mathematischen Fähigkeiten anzweifelten - zu Unrecht, wie sie später belegen konnte. Immerhin hat gut die Hälfte der Leserbriefschreiber den Anstand gehabt, sich einsichtig zu zeigen und ein Entschuldigungsschreiben aufzusetzen. Teile aus diesen Schriftwechseln sind auf ihrer Webseite nachzulesen unter: http://www.marilynvossavant.com/articles/gameshow.html.

Worum es bei dem Ziegenproblem geht: Ein Kandidat wird in einem Quiz vor die Wahl zwischen den drei Türen A, B unc C gestellt. Eine der Türen führt zum Hauptgewinn, hinter den anderen beiden Türen verbirgt sich eine Ziege, mithin also eine Niete. Der Kandidat darf sich für eine der drei Türen entscheiden. Diese Tür bleibt jedoch vorerst verschlossen. Stattdessen wird eine der beiden anderen Türen vom Quizleiter geöffnet und eine der Nieten gezeigt. Nun darf der Kandidat entscheiden, ob er bei seiner Wahl bleibt, oder die Tür wechseln möchte.

Intuitiv antworten die meisten Leute, dass es doch egal sei, ob man wechselt oder nicht. Schließlich ist es doch jetzt eine 50:50 Chance, ob man vorher die Tür mit der Ziege oder dem Hauptgewinn erwischt hat. Ob Wechsel oder nicht, was kann das jetzt für einen Unterschied machen?

Es macht einen Unterschied - und zwar verdoppelt sich die Gewinnchance nach einem Wechsel! Wie kommt es dazu? Angenommen, der Kandidat hat anfangs auf die richtige Tür A gesetzt. Die Wahrscheinlichkeit hierfür liegt bei $\frac{1}{3}$. Nun entfernt der Moderator eine der beiden Nieten. Ein Kandidat, der die Wechsel-Taktik spielt, wird jetzt zur verbleibenden Niete wechseln, und damit leer ausgehen. Der wechselunwillige Kandidat gewinnt hier.

Nun nehmen wir an, der Kandidat hat zu Anfang eine der beiden Nieten-Türen gewählt. Das wird in $\frac{2}{3}$ aller Fälle eintreffen. Die verbleibende Nieten-Tür wird anschließend vom Moderator aus dem Spiel genommen (den Gewinn darf der Moderator ja nicht entfernen). Mit der Wechsel-Taktik landet der Kandidat nun bei der Tür mit dem Hauptgewinn, während der wechselunwillige Kandidat auf seiner Ziege sitzen bleibt. In Abb. 4 ist diese Situation dargestellt.

Es zeigt sich also, dass der Wechsel-Kandidat eine doppelt so hohe Gewinnwahrscheinlichkeit erreicht! Man kann die Begründung auch anders angehen: Es ist wahrscheinlicher, anfangs auf eine Ziegen-Tür anstatt auf den Gewinn zu tippen. Jedoch muß der Moderator danach die verbleibende

[5]* 11. März 1822 in Paris, FR; † 5. April 1900 in Paris, FR

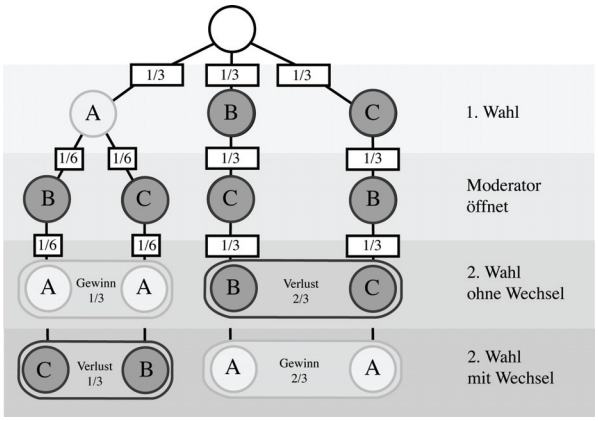


Abbildung 4: Entscheidungsbaum für das Ziegenproblem

Ziegen-Tür entfernen, so dass hinter der noch im Spiel befindlichen und in der ersten Runde ungetippten Tür der Gewinn verbleibt.

5 Das Triell

Eine etwas paradoxe Situation kann bei einem *Triell* entstehen. Die erste bekannte Erwähnung dieses Phänomens fand 1938 in [7] statt, größere Bekanntheit erlangte es u.a. mit [2] 1959 sowie unlängst durch eine Erwähnung in [8].

Die Regeln eines Triells sind schnell erklärt: Drei Schützen, jeder mit einer gewissen Trefferwahrscheinlichkeit, schießen nacheinander so lange aufeinander, bis nur noch einer lebt. Aus Gründen der Fairness darf der schlechteste Schütze anfangen, als zweites schießt der zweitschlechteste, und als letztes der beste, wenn er dann noch lebt. Nennen wir unsere Schützen Anton, Bernd und Claas. Die Trefferwahrscheinlichkeit für Claas liegt bei $p_C = \frac{1}{3}$, Bernd trifft in zwei von drei Fällen ($p_B = \frac{2}{3}$), und Anton ist der perfekte Schütze: $p_A = 1$. Wie soll man sich nun verhalten, wenn man dummerweise die Rolle des Claas einnehmen darf?

Intuitiv mag man versucht sein, Anton ins Visier zu nehmen. Schließlich stellt er ja irgendwie die größte Gefahr dar. Oder doch auf Bernd anlegen? Immerhin ist er direkt der nächste nach Claas.

Sehen wir uns die Optionen etwas genauer an. Wenn wir mit Erfolg auf Bernd schießen, dann hat Anton nur noch uns als Ziel. Bei seiner einhundertprozentigen Trefferwahrscheinlichkeit keine sehr gute Idee. Entscheiden wir uns dagegen auf Anton anzulegen und treffen, so ist unmittelbar nach uns Bernd dran. Auch er hat dann nur noch uns als Ziel, und in 67% der Fälle wären wir erledigt.

Der Ausweg aus diesem Dilemma, so überraschend es erscheint: Wir schießen in die Luft! Bernd wird dann auf Anton anlegen. Sollte Bernd treffen, wären wir wieder dran, und hätten nur noch Bernd als Gegner. Verfehlt Bernd sein Ziel, so wird Anton Bernd als größte Gefahr identifizieren und ausschalten. Auch danach wären wir an der Reihe, und haben immerhin eine Chance, Anton auszuschalten. Egal, welcher der beiden anderen Spieler treffen mag, am Anfang der zweiten Runde steht uns nur noch ein einziger Gegner gegenüber. Das Triell kann somit in ein Duell verwandelt werden, mit erheblich besseren Aussichten für uns, da wir wieder den ersten Schuss in diesem Duell haben!

Der erwähnte Sachverhalt hält auch einer genaueren mathematischen Untersuchung stand. Durch die Taktik des ersten Schusses in die Luft kann Claas eine durchschnittliche Überlebenswahrscheinlichkeit von knapp 40% erreichen. Beispiele dafür finden sich in [4] und [5]. Werden allerdings die Parameter variiert, also die Trefferwahrscheinlichkeiten der Schützen verändert, so kann sich auch die optimale Strategie ändern. Der Schuss in die Luft muß dann nicht der Königsweg sein.

Als Fazit bleibt: So manches Mal kann purer Aktionismus (in diesem Falle einfach drauf loszuschießen) doch die schlechtere Wahl gegenüber einem gelassenen Aussitzen der Situation sein.

Literatur

[1] Andrews, M. W.: *Anyone for a Nontransitive Paradox? The Case of Penney-Ante*, 2004

[2] Gardner, M.: *Mathematical Puzzles and Diversions*, Penguin Books Ltd, Harmondsworth, England, 1959

[3] Graham, R. L., Knuth, D., Patashnik, O.: *Concrete Mathematics: A Foundation for Computer Science*, 2nd edition, Addison-Wesley, 1994

[4] Kilgour, D. M.: *The Sequential Truel*, International Journal of Game Theory, Volume 4, Number 3, Physica / Springer Verlag, 1975

[5] Kilgour, D. M., Brams, S. J.: *The Truel*, Mathematics Magazine 70, 5, S. 315-326, 1997, http://www.econ.nyu.edu/cvstarr/working/1997/RR97-05.PDF

[6] Penney, W: *Problem 95: Penney-Ante*, Journal of Recreational Math. 7 (1974), S. 321.

[7] Phillips, H.: *Question time; an omnibus of problems for a brainy day*, Farrar & Rinehart, LCCN 38-005540, New York, 193

[8] Singh, S.: *Fermats letzter Satz*, Deutscher Taschenbuch Verlag, München, 7. Aufl. 2002

Hacking
lecture
Tag 1 17:15
Saal 2
en

Victor Muñoz

AES: side-channel attacks for the masses

AES (Rijndael) has been proven very secure and resistant to cryptanalysis, there are not known weakness on AES yet. But there are practical ways to break weak security systems that rely on AES.

In this lecture we will see how easy could be retrieve AES keys attacking the implementations, when you have physical access to the box that tries to hide a key you can easily spot it, such kind of security could be just named obfuscation but is widely used in DRM technologies like AACS.This is just a demonstration that using a strong security algorithm like AES is not of much sense when give the key somehow obfuscate to the attacker, remember that the security chain is as strong as the weakest of their components.

http://www.ingenieria-inversa.cl/AES02.pdf AES: side-channel attacks for the masses

AES: side-channel attacks for the masses. (rev 0.2)

Victor Muñoz
vmunoz@ingenieria-inversa.cl

October 2007

Abstract.

AES (Rijndael) has been proven very secure and resistant to cryptanalysis, there are not known weakness on Rijndael algorithm up to day. But there are some practical ways to break weak security systems that rely on AES.

Introduction.

AES has been subject to exhaustive cryptanalysis efforts, but none of them could break the cipher.

The newest attacks can break only short-cut versions of AES, with a reduced number of rounds (up to 9 rounds on AES-192), the most fruitfully techniques used were Collision Attack, Square Attack, Impossible Differential, Truncated Differential and Related Key, you could see a summary of the cipher breaking level of such techniques in [1], and see a briefly description of some of them in [2].

The most practical attacks on AES are side-channel attacks, that don't intend to attack the algorithm itself, but look to reconstruct the key from secret leakage through the physical implementation of the algorithm; such leak of information could be –among others- Power Consumption, Time, Electromagnetic Radiation, and etcetera.

In AES breaking quest Simple Power Analysis and Differential Power Analysis were used roughly on attacks to smart-cards as stated in [x]. Also Cache Timing Attacks are well known, but seem a little hard to use it in real world situations, also they may need clock cycle level accuracy in the timing measurements, and big amounts of sampling, those Cache Timing Attacks do not seems feasible for other scenarios than process-to-process attacks (ie: remote key retrieval).

Suppose you are in a dealing with a process-to-process situation, that means that your offensive process has some access to the overall system, then why to bother to use a complex attack when you could use some other meaning to spot AES keys in no time?.

In this document we will see 2 methods for attack AES that should work with no problem in real world situations and are

not exclusively for neither laboratory experiments nor concept proofs.

Those attacks are intended to retrieve an AES key when you have physical access to the machine you want to attack, one method require you have full access to the system meaning you could install a debugger or exception handler, and full access to the process you want to attack.

The second method is simpler to implement and you only need to have reading access to memory of the victim process, extending this method you could gain access to AES key directly from the RAM IC modules assuming the RAM is not encrypted, the AES implementation is software based, and of course all the key processing is not fit just in the internal CPU data cache.

Why could you be interested to attack machines that you own and not a third party victim? Simple, there exists lot of boxes that come locked (and limited) only to run the software singed for the box vendor, machines like videogame consoles, set-top boxes, cell phones, routers, etc.

Such key retrieving activity has been very useful –for example- in the efforts to circumvent DRM schemes like AACS, that rely strongly on AES, your AACS licensed player software hides you the keys needed for decode a movie, and that simply prevent you to make your own media player or see your movies in any free operating system, moreover you could not see a HD movie at full resolution in a non HDCP licensed (and yet expensive) monitor.

Easy AES key retrieval History.

Let's begin with a little of history, *muslix* (the former hacker of AACS system) [4], has got the keys needed to consider AACS cracked back in December 2006 without the need for tracing or debugging any bit of code, the method he used was simply guess the decrypted header of a video stream block and run a key finder in a memory dump of the process of the AACS enabled player software trying every 16 continuous bit as keys, and that lead him –just in seconds- to a VUK (Volume Unique Key) needed to decrypt the whole movie, and see it in any player, setup or OS that you want.

We are going to refer here to the above attack as known-plaintext/key within process memory (in rigor was guessed-plaintext and not known-plaintext).

This attack was recognized by the same AACS LA on January 24, 2007 [5], and from that moment AACS scheme was in fact full compromised.

Some months after the original attack, more attacks come to

the AACS scheme, all those attacks have something in common: AES key spotting with a little of effort in comparison with the state of art side-channel attacks on AES.

Reference

[1] http://www.iaik.tu-graz.ac.at/research/krypto/AES/ - IAIK Krypto Group - AES Lounge

[2] http://www.iaik.tugraz.at/aboutus/people/oswald/papers/aes_report.pdf - AES - The State of the Art of Rijndael's Security

[x]

[4] http://forum.doom9.org/showthread.php?t=119871

[5] http://www.aacsla.com/press/ January 24, 2007

Science
lecture
2007-12-29 16:00
Saal 2
en

Tomasz Rybak

Analysis of Sputnik Data from 23C3

Attempts to regenerate lost sequences

In December 2006, in BCC 1000 atendees were wearing Sputnik Tags. Data was stored, and then made available for analysis. Unfortunately all IDs of tags were lost. This lecture presents what was stored, what happened to it, and attempts of reconstructing IDs and sequences of movements.

Presentation shows simple statistics of Sputnik data. The main part is description of ways of generating sequences of packets generated by tags. Two methods, local ang global are described, with few variants. Problems with using those methods are presented.

http://www.openbeacon.org/	Main page of Sputnik Project
http://www.bogomips.w.tkb.pl/sputnik.html	My page with some analysis
http://pmeerw.net/23C3_	Page with analysis made by Peter Meerwald
http://wiki.openbeacon.org/wiki/Datamining	Open Beacon Wiki about analysing data

Analysis of 23C3 Sputnik data

Tomasz Rybak
tomasz.rybak@post.pl

This article describes attempts to analyse data coming from Sputnik project gathered during 23rd Chaos Communication Congress. The most significant problem was recovering lost sequence identifiers, and this is main subject of article.

1 Sputnik idea

Sputnik is RFID system intended to trace people in small areas, and buildings. Each person is wearing tag that transmits its identifier in regular time intervals to allow to store this persons position at those moments. System was used during previous, 23rd Congress, and during Chaos Communication Camp 2007. Data from Camp has not yet been released, and this article describes analysis performed on data from 23C3.

After releasing data there were few web pages created describing system and data, and trying to analyse it. The main page of project[1] is maintained by creators of Sputnik system. Wiki of OpenBeacon contains page[2] with discussion about released data. Peter Meerwald came with page[3] presenting come analysis of gathered data. Kaners page[4] contains parser of log files, allowing to get information about only particular ID. My page[5] contains programs and results described in this article.

2 Hardware

Ordinary RFID systems are operating in range of few dozens kHz, and use passive tags. Tag does not contain any power source; it is powered by reader during reading process. So without reader it can do nothing. Sputnik uses active tags; they have own battery and transmit data whatever there is reader listening to it or not. Using own battery allows for having high power and thus high range of transmission. Range in buildings is up to the 10m even through dry walls. Concrete walls tend to block signal. Because transmission occurs at 2.4GHz, human body decreases power by about 50%.

Thanks to own battery tag has control over transmission power and can send signals varying in strength. This allows for estimating distance from reader. During 23C3 25 readers were placed in BCC in such a way that in most cases more than one reader saw tag. This, because of possibility of estimating distance from reader, allows for estimation of position of tag.

First readers were large boxes using Power Ethernet to communicate with the server and to power themselves. During Camp Milosz Meriac presented USB reader[6], small device, powered and transmitting data using USB. It acts like terminal, sending data in text format; computer can receive read packets, and send commands to it. Additionally it can also serve as tag, as it have full transmitter on board. Because it is more sophisticated than tag, user has more control over sent RFID packets. It creates /dev/ttyACM* device and sends text in either “ID,Sequence,strength,flags” or “RX: ID,strength,number” format, depending on version of firmware. It can be reprogrammed directly using USB, without any additional hardware.

[1] http://www.openbeacon.org/
[2] http://wiki.openbeacon.org/wiki/Datamining
[3] http://pmeerw.net/23C3_Sputnik/
[4] http://cakelab.org/ kaner/sputnik_01/
[5] http://www.bogomips.w.tkb.pl/sputnik.html
[6] http://wiki.openbeacon.org/wiki/OpenBeacon_USB

3 Data format

Data gathered during 23C3 was made available as both XML and binary files.

XML file
Consisted of "observation" tags with following attributes:

id ID of tag

time

position position of tag; (0, 0, 0) if unknown

direction always (0, 0, 0)

priority always the same value 24

min-distance always 0.0

max-distance always 255.0

observer URL of aggregating station; only one value present in file

observed-object URL of station together with tag ID

XML file contains very small portion of data that was gathered during 23C3. It has only 357974 entries, where full data set is 11.1 million of observations. It does not contain details of readers used to calculate positions of tags. This omission is important, as about 1/3rd of observations has no meaningful position calculated, probably because in those cases there was not enough data to calculate those positions. Also XML file contains data from only few hours for each day of Congress; probably those are hours when server was active. Number of observations during the Congress stored in XML file is shown in Figure 3.

Because of having no sequence numbers and reading stations used to calculate positions, I did not use XML data in analysis.

Data from binary file was more useful for analysis, although it contained errors. Because of error in server software, identifiers of tags were not saved.

Binary format according to source code

0-4 timestamp

5-8 reader station IP

9 size of frame (0x10)

10 protocol (0x17)

11 flags (0x02 — button pressed)

12 strength of signal

12-16 sequence number

17-20 Tag ID

21-24 check sum

Binary format present in file

0-4 timestamp

5-8 reader station IP

9-12 garbage (used by me to write ID)

13-16 garbage, reversed IP of reader station

17 size of frame (0x10)

18 protocol (0x17)

19 flags (0x02 — button pressed)

20 strength of signal

21-24 sequence number

Missing identifiers made analysis almost impossible. Additional problem were 8 bytes in one of files; information published on OpenBeacon mailing list allowed me to removed those unnecessary bytes and to have full data set. Binary data set had 64K repeated readings — observations that were the same as other observations.

4 Database

Data set so large takes long time to read and parse it. I decided to store it in PostgreSQL database. In the beginning both XML and binary sets were stored in one table, but then it was divided into two tables; then more support tables were added; PostgreSQL table inheritance was used to ease operating on main data tables[7].

Created database can be seen as temporal, and when looking at XML data also as spatial one. Such databases store information about presence of phenomenas in space and time. This database stores information about presence of tags (and probably persons wearing them) at the place at the moment. Also activities done to this tags, like pressing button, are stored. Additional spatial data, like geometry of building and rooms where events were held, and temporal data (schedule of Congress) can be used for more sophisticated analysis.

Created tables

station Describes readers

sputnik base table for storing data; tables with data inherit from it

ccc23 contains binary data from 23C3

ccc23xml contains XML data from 23C3; has additional columns containing values of attributes from XML file

reader table used to store data received by USB reader

adjacency stores count of readings seen by pairs of readers

room describes lecture rooms

event describes events that took place during 23C3; taken from Schedule XML file

[7] Scripts creating database can be downloaded from my web page

Base table for holding data from tags

id

time

sequence value of sequence counter

strength strength of signal

station id of station that received this signal

tags array of data, like pressed button

XML data table
is like raw data table and also contains:

position position of tag

plane position on the floor

direction direction; currently only (0. 0, 0)

observer

observedobject

priority

mindistance

maxdistance

Table of rooms
Describes room in which events (lectures) were taking places.

id identifier of room

name name of room: “Saal 1”, “Shelter foo”, . . .

shape path describing room shape. Currently empty column; data to fill it could be taken from GPS data or from building plans

ymin

ymax

bbox Is it necessary, or better use geometry calculations or PostGIS?

Event table
Describes information about events. Populated using XML schedules published on http://www.ccc.de/

id identifier of event

organizerid

name name of event

place identifier of room event is taking place

description human-readable description

address URL of description of event

start timestamp of beginning moment of event

finish timestamp of end moment of event

Table containing data from 23C3 occupies about 700MB on hard drive. Data types used to store sequence and time values occupy 8 bytes each; index for each of those columns takes 250MB. Sequence identifier is stored as 4 byte integer and its index takes about 130MB. Creation of those indexes is necessary to have database offering good performance. This is not huge database, but is rather large for desktop computer.

Large amounts of rows can be changed when operations on data are performed. To be able to find good query plan, PostgreSQL needs to have accurate statistics of stored data. PostgreSQL does not update rows in place, but creates new row and marks old as deleted; this technique is called MultiVersion Concurrency Control (MVCC). So once in a while database needs to be vacuumed to remove all those deleted rows and to gather statistics. Autovacuum is daemon that takes care of observing all tables and performing vacuum when it is needed. Its default settings are too low for Sputnik data. The more reasonable is to analyse data table after 0.5% rows were changed and vacuum after 10% rows were changed. It makes sense to have more aggressive autovacuum by setting cost limit to 500 and delay to 0.

PostgreSQL client library, libpq, fetches entire result data set into RAM. This can be problem when exporting Sputnik data from database. I was getting "out of memory" error, so I had to use cursor to be able to retrieve data set partially. Solving this problem internally in libpq library (by using internal cursor) to be able to fetch large data set partially is in ToDo list of PostgreSQL.

5 Analysis of data

To understand further operations, one needs to understand how internally tags work. In each transmission tag sends its ID and strength of signal it uses to transmit. Each transmission is encrypted using XXTEA. To avoid replay attacks, it is necessary to change packets. Because adding real time clock would be too complicated, ever-increasing counter was added. Base station discards all packages with counter numbers less that the one seen previously. To avoid problems with reset of tag (removing battery) when counter is again set to 0, counter was divided. Higher word was saved on reset, and lower not. So after reset tag increases higher word, so counter value always grows. This feature means that gaps occur in counter values sequences when tag is reset. To avoid collisions, each tag transmits and sleeps random time, from 2 to 4 seconds.

5.1 Basic graphs

Following pictures present simple characteristics of data. They are based on work done by Peter Meerwald, mostly to make sure that data was correctly imported. Numbers present on following figures are larger than presented by Peter Meerwald. He was using hash tables to store Sputnik data, so he had not seen 64k repeated observations, which become visible in database.

Figure 1 presents how many packets were seen by more than one station. It shows only situations where stations were seeing more than 1000 common packets. It can be used to deduce how people were walking inside Congress Center, and also could be used to deduce positions of readers inside building.

Figure 2 shows number of packets seen in entire system in each minute. It can be seen that during day there is high activity, and during night hours activity is very low, because most of attended left the BCC.

Figure 3 shows activity of all XML data points. It shows both observations containing valid estimated position, and position "0, 0, 0". Activity in the beginning consists of observations with invalid position; almost all later observations contain valid positions.

Following tables show number of packets that each reading station has received and number of received packets with particular strength of signal.

Packets read by each station

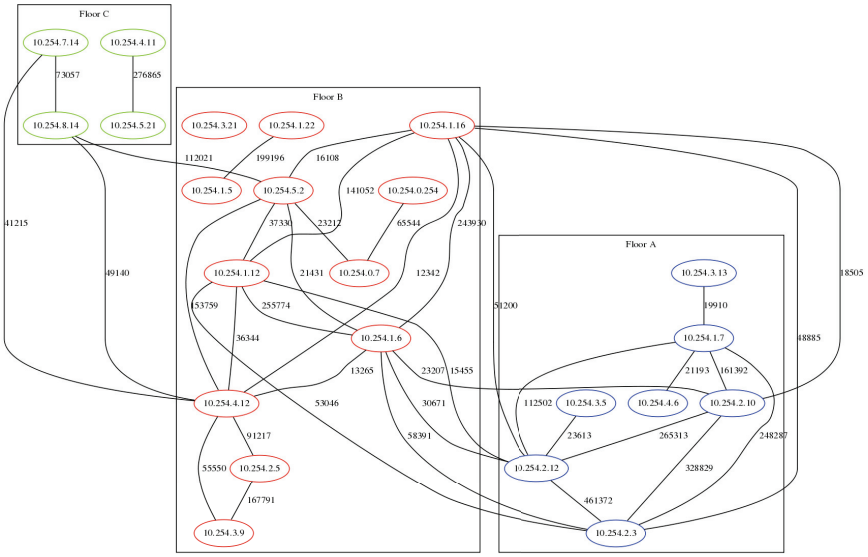


Figure 1: Pings read by more than one station (> 1000)

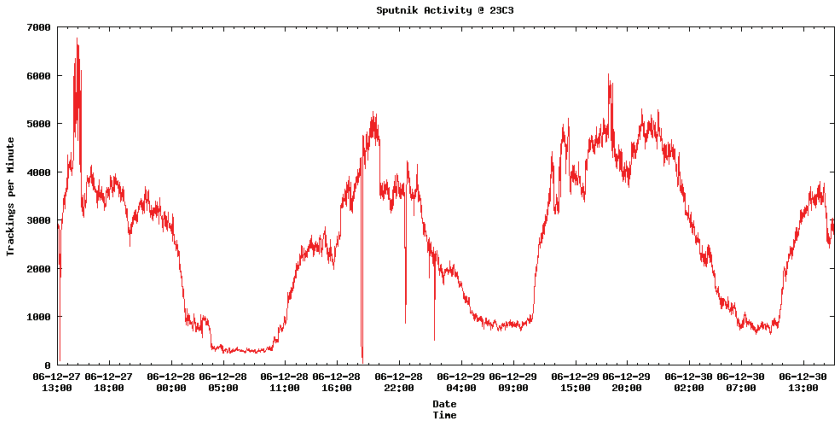


Figure 2: Number of packets read during one minute

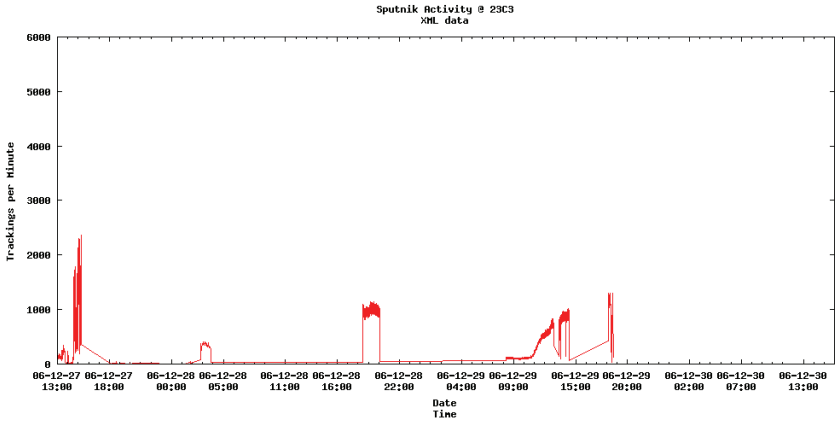


Figure 3: Number of packets read during one minute including unknown points

Id	IP address	count
2	10.254.2.3	1322696
21	10.254.5.21	880833
3	10.254.2.12	760606
15	10.254.1.6	758782
18	10.254.5.2	596466
14	10.254.4.12	589640
20	10.254.8.14	585443
26	10.254.1.16	570525
5	10.254.1.7	568765
4	10.254.2.10	563488
1	10.254.4.6	542657
16	10.254.1.12	532699
22	10.254.4.11	528187
11	10.254.1.22	494524
10	10.254.1.5	448760
9	10.254.2.5	428565
8	10.254.3.9	376396
24	10.254.3.5	231483
23	10.254.7.14	225075
17	10.254.0.254	187078
6	10.254.3.13	130379
13	10.254.0.7	129144
12	10.254.3.21	54863
25	10.254.0.100	8524

Strength of packets

Strength	count
0	182874
85	568413
170	1167287
255	9225658

5.2 Rebuilding sequences

To be able to analyse data and gain some knowledge from it, sequences need to be restored. It requires joining single packets into sequences and then attaching unique number into each found sequence. Unfortunately original tag identifiers are lost and it is impossible to recover them; but even without them restoring sequences will allow for analysis of data.

Global searching requires large amounts of CPU time, RAM and disk resources, so first program was using local search for short sequences.

Following snippet presents ideal situation when building sequences. It takes first packet and then tries to find next one, that has next value of counter, and is 1 or 2 seconds from previous one. It does not take into consideration gaps in sequences because of person leaving BCC, or because one is not in the range of any readers, or when tag is transmitting too weak signal to be received by any of readers. However it presents idea of finding local sequences; following functions are using this idea and add code dealing with gaps and choosing one packet that can be added to sequence when there is more than one.

First attempt of building sequences

```
SELECT time, extract('epoch' from time), sequence
FROM sputnik.sputnik WHERE id IS NULL AND
time BETWEEN %s::TIMESTAMP WITH TIME ZONE
AND %s::TIMESTAMP WITH TIME ZONE+%s::INTERVAL
for i in c.fetchall():
    old_e, old_s = int(i[1]), int(i[2])
    old_major = old_s/65536
    old_minor = old_s%65536
    p = []
```

```
    for j in data:
        e, s = int(j[1], int(j[1])
        major = s/65536
        minor = s%65536
        probable = (major == old_major and minor == old_minor+1)
            or (major == old_major+1 and minor == 0)
        if probable: p.append([e, s])
    if len(p) > 0:
        print old_e, old_s,
        for j in p: print j[0], j[1],
```

Basic idea of algorithm for searching local sequences is enhancements of code above. It takes all points from choosen period of few dozens seconds. To find all sequences of ticks there it assumes that ticks are about 1.5s from one another. Starting from the lowest counter value it tries to find the next value. In case of very close values of counter, difference of time is 1 or 2 seconds. In case of longer time distances, difference should be closer to 1.5s for every tick. It ignores data about strength of signal or stations that were able to receive it.

When more than one packet can be chosen to extend sequence conflict occurs, and this problem must be resolved. Conflict may be because either at the same time there are two different counter values, or the same value occurs at different moments. In case of either conflict we must choose only one packet to include in sequence, and discard another one. It needs to be noticed that not only two, but more packets may be involved in conflict. The general case is presence of more than one sub-sequence that can extend existing sequence. Only one of them must be chosen, as adding all sub-sequences will destroy existing sequence by introducing decreases in either time or counter values.

Sub-sequence may be chosen by taking into consideration length or resemblance to already existing sequence. Using separate function for choosing sequence to add allows for researching on different criteria of choosing and introducing more sophisticated criteria.

Alternative solution is creation of function returning next values of time and counter, basing on sequence that is being rebuilt. This is more complicated, as it requires knowing exact parameters of tag, especially time when it was started or reset, and exact time tag sleeps between transmissions.

Function `GetTickDistance` returns difference between counter values. It tries to take reset into consideration by treating reset as difference of 1. It decides that reset occurred when values passed as arguments have differing high words. However if there is less than about one minute to change of high word, it does not assume reset was involved.

Distance between sequence values

```
# Assumes a <= b
# Will not work when there is more than 1 overflow
def GetTickDistance(a, b):
    majora = a/65536
    minora = a%65536
    majorb = b/65536
    minorb = b%65536
# Inside one minor, or less than minute to overflow
    if majora >= majorb or minora >= 65500:
        return b-a
    else:
        return majorb-majora + minorb+1
```

To be able to recreate sequences it is necessary to create all alternatives and then choose the best ones. Hashes are used to store all counter values that were received at any moment, and all moments when any value of counter was received. All keys of hashes are read in increasing order, and all values stored under every key are considered as extensions of sequences. If considered point can be added to sequence, it is. If not, conflict is detected. Previous value is removed from sequence, and both points are added to special list of alternatives. In such case each subsequent point is treated as extension not of main sequence, but alternative sub-sequences. If it can be added to all of them, alternatives are stored,

and this point is added to main sequence. If it can be added to only some of sub-sequences, conflict still remains. If it cannot be added to any of sub-sequences, it is added as another alternative sub-sequence.

Function `FindBestSequence` takes sequence and all alternative sub-sequences calculated by previous function and builds optimal sequence. It chooses the best possible sub-sequences to add. To choose the best ones it uses slope of sub-sequences, and chooses one with the slope closest to 1.5. Minimal square difference is used to find slope closest to ideal.

Finding best sequences amongst all created

```
# Sequence with len >= 3
def FindBestSequence(a):
    b = max(map(len, a))
    c, a = a, []
    for i in c:
        if len(i) == b: a.append(i)
# Find minimal difference between min and max, in case of many alternative sequences
    best = i = a[0]
    ds = float(i[1][0]-i[0][0])/GetTickDistance(i[0][1], i[1][1])
    mini = maxi = ds
    for j in range(1, len(i)-1):
        ds = float(i[j+1][0]-i[j][0])/GetTickDistance(i[j][1], i[j+1][1])
        mini = min(mini, ds)
        maxi = max(maxi, ds)
    c = (mini-1.5)*(mini-1.5)+(maxi-1.5)*(maxi-1.5)
    for i in a[1:]:
        ds = float(i[1][0]-i[0][0])/GetTickDistance(i[0][1], i[1][1])
        maxi = mini = ds
        for j in range(1, len(i)-1):
            ds = float(i[j+1][0]-i[j][0])/GetTickDistance(i[j][1], i[j+1][1])
            mini = min(mini, ds)
            maxi = max(maxi, ds)
        d = (mini-1.5)*(mini-1.5)+(maxi-1.5)*(maxi-1.5)
        if d < c: best, c = i, d
    return best
```

Described algorithm can be implemented in two ways. Main loop may iterate over time and check all possible counter values, or it can iterate over counter values and check all moments of appearance of this value. Those approaches should be equivalent, but iterating over counter values gives as result more and longer sequences. If using more CPU time is not a problem, both variants can be used and the best results given by any of them are chosen, independently for each considered interval.

First code that was used to use large part of data was implementation of $O(N^3)$ algorithm. For each point it was finding whether any of other points can be added to the sequence by checking if equation $\Delta s = a\Delta t, 1.0 \leq a \leq 2.0$ was met. After finding all possible points it was generating all possible alternatives from this chosen set. As it was checking all other points for every point from given interval, this operation was $O(N^2)$. If any sequence was found, it was removed from data set, and entire process was started from the beginning, thus $O(N^3)$ time cost.

$O(N^3)$ algorithm

```
SELECT DISTINCT time, extract('epoch' from time), sequence
FROM sputnik.sputnik WHERE id IS NULL AND
time BETWEEN %s::TIMESTAMP WITH TIME ZONE
AND %s::TIMESTAMP WITH TIME ZONE+%s::INTERVAL
a, b, again = 0, 0, True
while again:
    again, s = False, []
    for i in data:
        majort, majors = int(i[1]), int(i[2])
        p = [[majort, majors]]
        for j in data:
            minort, minors = int(j[1]), int(j[2])
```

```
            dt = minort-majort
            ds = GetTickDistance(majors, minors)
            if dt > 0 and ds <= dt and dt <= 2*ds:
                p.append([minort, minors])
        if len(p) > 1:
            again = True
            r = CreateAllSequencesSeqs(p)
            s = FindBestSequence(r)
            a += 1
            if len(s) > b: b = len(s)
            break
    if again:
        for i in s:
            UPDATE sputnik.sputnik SET id = %s
            WHERE sequence = %s AND time = to_timestamp(%s)
            for j in data:
                if i[0] == j[1] and i[1] == j[2]:
                    data.remove(j)
                    break
        id += 1
```

Improving speed of this algorithm came from observation that the longest sequences are be made when starting from the lowest time and lowest counter values. Query was changed to return sorted result. Algorithm was changed to take first tuple, and try to find all other tuples that can make sequence with the first one. If sequence was found, it was removed from data set; if not, only the first tuple was removed. So for each tuple all other tuples were considered, which gives $O(N^2)$. Because there is no repetition of this process if sequence is found, but further tuples are processed, this cost remains.

This algorithm gives the same results as previous one; this was proved by comparing sequences generated by both for few intervals. Cost of those algorithms can be slightly higher than $O(N^3)$ and $O(N^2)$ when considering building and comparing alternative sub-sequences. However size of such sub-sequences is small when compared to main sequences. Also size of sub-sequences tend to remain constant even when increasing length of analysed interval, which increases size of generated sequences.

$O(N^2)$ **algorithm**

```
SELECT DISTINCT time, extract('epoch' from time), sequence
FROM sputnik.sputnik WHERE id IS NULL AND
time BETWEEN %s::TIMESTAMP WITH TIME ZONE
AND %s::TIMESTAMP WITH TIME ZONE+%s::INTERVAL
ORDER BY sequence, time
a, b = 0, 0
while len(data) > 0:
    s, i = [], data[0]
    majort, majors = int(i[1]), int(i[2])
    p = [[majort, majors]]
    for j in data[1:]:
        minort, minors = int(j[1]), int(j[2])
        dt = minort-majort
        ds = GetTickDistance(majors, minors)
        if dt >= 0 and ds <= dt and dt <= 2*ds:
            p.append([minort, minors])
    if len(p) > 1:
        r = CreateAllSequencesSeqs(p)
        s = FindBestSequence(r)
        a += 1
        if len(s) > b: b = len(s)
        for j in s:
            UPDATE sputnik.sputnik SET id = %s
            WHERE sequence = %s AND time = to_timestamp(%s)
            for k in data:
```

```
                if j[0] == k[1] and j[1] == k[2]:
                    data.remove(k)
                    break
        id += 1
    else:
        data.remove(i)
```

Function `JoinIDs` computes all sequences for one interval and interval after that, and then tries to join found sequences. For each sequence in main interval it calculates coefficient of line created by its last point and by first point of sequence from the next interval. If any line with coefficient between 1.0 and 2.0 is found it means that those sequences are candidates for joining. However they would also have to have the same coefficients themselves before they could be joined.

Function trying to join found sequences

```
def JoinIDs(c, t, d, period):
    main = GetLines(c, t.strftime("%Y-%m-%d %H:%M:%S+01:00"), period)
    after = GetLines(c, (t+d).strftime("%Y-%m-%d %H:%M:%S+01:00"), period)
    before = GetLines(c, (t-d).strftime("%Y-%m-%d %H:%M:%S+01:00"), period)
    for i in sorted(main.keys()):
        majort = main[i]['max-time']
        majors = main[i]['max-seq']
        for j in sorted(after.keys()):
            minort = after[j]['min-time']
            minors = after[j]['min-seq']
            dt = minort-majort
            ds = GetTickDistance(majors, minors)
            if ds <= dt and dt <= 2*ds:
                print "Can Join"
                print "\t", main[i]['id'], main[i]['length'], main[i]['min-time'], main[i]['min-seq'],
                print main[i]['max-time'], main[i]['max-seq']
                print "with", ds, dt, float(dt)/ds
                print "\t", after[j]['id'], after[j]['length'], after[j]['min-time'], after[j]['min-seq'],
                print after[j]['max-time'], after[j]['max-seq']
```

I think it could be even possible to improve local algorithm to have $O(N)$ time cost. However it was not implemented so I do not know if it is really possible and if it would give good results.

Function calculating distance in counter values was changed, as it was producing strange sequences (65600, 132000, 512000, ...). Reset was ignored, and distance was ordinary difference of counter values. However this was not helpful. Local algorithms were not able to find long enough sequences. Although few found sequences were rather long (up to 20 packets for 1 minute), but most found were only consisting of 2 or 3 packets. This was leading to large gaps between sequences from consecutive intervals, and troubles with joining them.

New distance in sequence counter function

```
# Assumes a <= b
# Will not work when there is more than 1 overflow
def GetTickDistance(a, b):
    majora = a/65536
    minora = a%65536
    majorb = b/65536
    minorb = b%65536
    return b-a
```

Scatter plots drawn for long intervals are revealing straight lines. This lead to the idea to find straight lines (as drawn in geometry) and to treat them as sequences. To avoid problems with reset calculations were done inside 64k blocks of counter values.

The best way to find the longest sequences is to start with point with the lowest values of counter and time. Then try to draw lines through it and all other points from the range. Choosing slope that

results in line going through the most points gives the longest sequence. This is greedy algorithm as in each step the largest sequence is chosen.

To choose the best line coefficient histogram of all slopes is used, with bucket of size 0.1. To be sure that no point is left because of rounding errors, range of slopes is used: all points that are on lines with slopes differing less than ± 0.3 from chosen slope are included into created sequence.

Because for each point all other points are used to calculate slopes and then all points that are in right coefficient range are chosen, time cost is $O(N^2)$.

It finds long sequences. It leaves only about 4000 points (out of 11.1 million) without any sequence. However rather strange line coefficients are found; besides ordinary 2.4, 2.5, it comes with 0.1, 0.4, 0.5, 9.9, 10.0, 8.1, ...

Function `FindIDs` takes range of counter values and tries to find all sequences in this range. It finds all counter values and for each value finds all times it occurs; this is similar to hashes used in local algorithms. Then for each starting point histogram of all coefficients of lines is created and the largest value is used. Query similar to one calculating slopes is used to mark all points as belonging to one sequence. Update is done by one SQL query.

Finding sequences in global manner

```
def FindIDs(connection, sa, sz, ta, tz, id):
    SELECT DISTINCT sequence FROM sputnik.sputnik WHERE id IS NULL
    AND sequence BETWEEN %s AND %s ORDER BY sequence
    for s in c.fetchall():
        s0 = s[0]
        SELECT DISTINCT time FROM sputnik
        WHERE id IS NULL AND sequence = %s
        for t in
            t0, hash = t[0], {}
            SELECT DISTINCT ON (sequence, time) time, sequence,
            (extract('epoch' FROM (time-%s)))::float/(sequence-%s)::float
            FROM sputnik.sputnik WHERE id IS NULL AND time > %s AND
            sequence BETWEEN %s AND %s AND sequence != %s
            ORDER BY sequence, time
            for i in  c.fetchall():
                k = int(i[2]*10)
                if 0 < k and k <= 100:
                    hash[k] = hash.get(k, 0)+1
                i = c.fetchone()
            k = -1.0
            if len(hash) > 0:
                m = max(hash.values())
                for i in sorted(hash.keys()):
                    if m == hash[i]:
                        k = float(i)/10.0
                        break
                UPDATE sputnik.sputnik SET id = %s WHERE id IS NULL
                AND sequence = %s AND time = %s
                UPDATE sputnik.sputnik SET id = %s WHERE id IS NULL AND
                sequence BETWEEN %s AND %s AND sequence != %s AND
                (extract('epoch' FROM (time-%s)))::float/(sequence-%s)::float
                BETWEEN %s AND %s
                id += 1
    return id
```

Following code shows calling of function for creating sequences. First the lowest unused value for identified sequence is found, and then function `FindIDs` is called for each of the values of high word of tag counter. First range was divided into time intervals so program operates on smaller data sets, but because of error in code time interval was not respected and first call calculated all sequences from entire range.

Calling a sequence finder

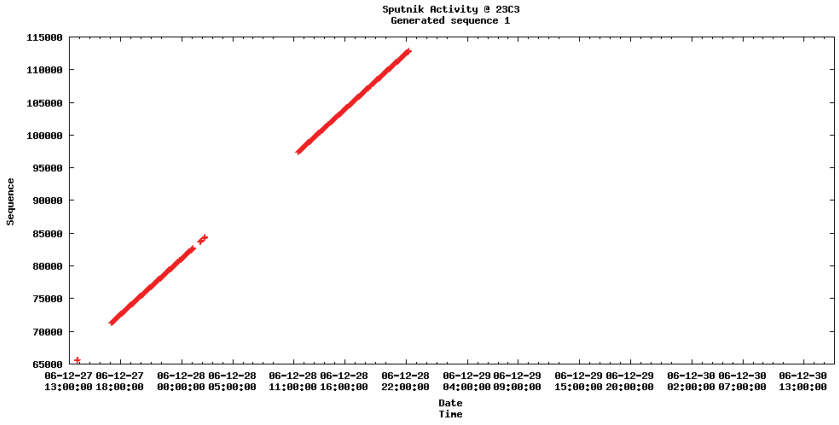


Figure 4: Generated sequence; first set, number 1

```
id = (SELECT MAX(id) FROM sputnik.sputnik WHERE id IS NOT NULL)+1
ta = '2006-12-27 12:59:19+01:00'
tz = '2006-12-30 15:59:59+01:00'
id = FindIDs(connection, 0, 2*65536, ta, tz, id)
# Very large data set, 2924448 rows
id = FindIDs(connection, 131072, 196608, '2006-12-27 12:59:19+01:00', '2006-12-27 18:00:00+01:00', id)
id = FindIDs(connection, 131072, 196608, '2006-12-27 18:00:00+01:00', '2006-12-28 00:00:00+01:00', id)
id = FindIDs(connection, 131072, 196608, '2006-12-28 00:00:00+01:00', '2006-12-28 17:00:00+01:00', id)
id = FindIDs(connection, 131072, 196608, '2006-12-28 17:00:00+01:00', '2006-12-29 00:00:00+01:00', id)
id = FindIDs(connection, 131072, 196608, '2006-12-29 00:00:00+01:00', '2006-12-29 16:00:00+01:00', id)
id = FindIDs(connection, 131072, 196608, '2006-12-29 16:00:00+01:00', '2006-12-30 00:00:00+01:00', id)
id = FindIDs(connection, 131072, 196608, '2006-12-30 00:00:00+01:00', '2006-12-30 15:59:59+01:00', id)
# Very large data set, 2076875 rows
id = FindIDs(connection, 3*65535, 4*65536, ta, tz, id)
# Very large data set, 1277488 rows
id = FindIDs(connection, 4*65535, 5*65536, ta, tz, id)
# Very large data set, 1016195 rows
id = FindIDs(connection, 5*65535, 6*65536, ta, tz, id)
# Very large data set, 620763 rows
id = FindIDs(connection, 6*65535, 7*65536, ta, tz, id)
```

Figures 4 to 9 show sequences generated by this algorithm. Some sequences are the proper ones, but other are wrong; their points really belong to many different sequences.

Figures 5 and 6 show sequences that from the beginning look like collage of many sequences. They show the main problem of algorithm: range of allowed coefficients is too wide, and too many points are added to sequence. The farther away from the first point, the more obvious it is.

Figure 7 shows sequence that in the beginning is correct, and gets wrong only in the end. So first part should be preserved, and after it, somewhere is this gap, sequence should end.

Figure 8 shows sequence that is generated by all variants of global algorithm.

Sequence shown in Figure 9 shows errors that came from integer overflow. Because initially I did not use Python large integers, counter values close to 4 billions were treated as small negative values, and joined with real small values. Column storing counter values was using 64-bit integers, so PostgreSQL was able to update rows with large counter values, and not destroy other sequences.

Figure 10 shows packets that were not used in any sequence. It was only about 4000 points, and it's very good result for data set consisting of 11.1 million of points.

Figure 11 shows size of generated sequences calculated as number of occurrences of pair (time, counter value); event if packet was seen by more than one reader, it was counted only once. In other words it shows number of occurrences of tag, not how many times it was seen.

Figure 12 shows size of sequences calculated as number of tuples that are included into each sequence.

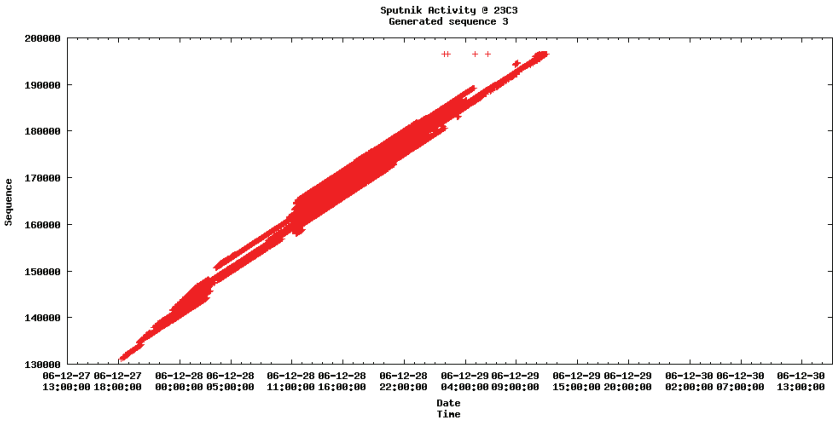


Figure 5: Generated sequence; first set, number 3

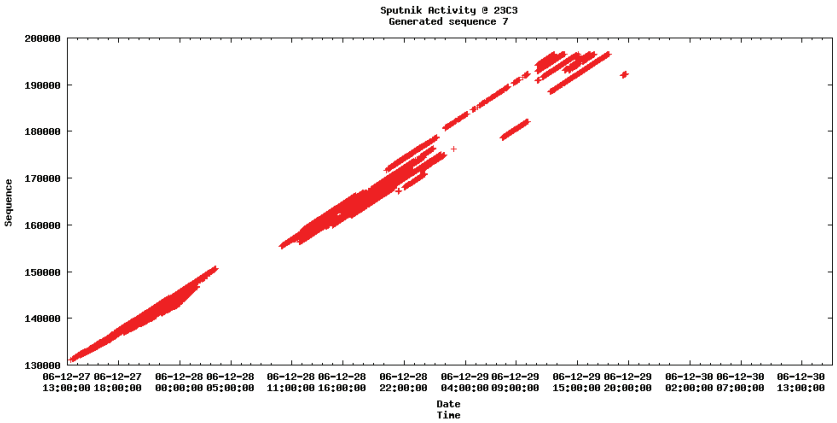


Figure 6: Generated sequence; first set, number 7

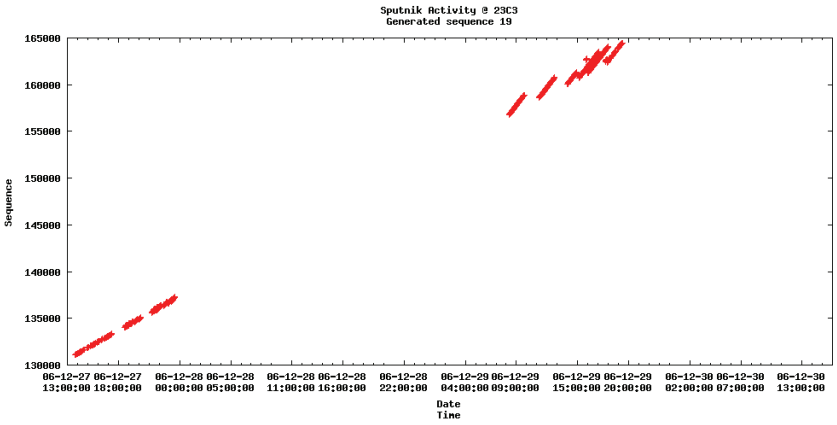


Figure 7: Generated sequence; first set, number 19

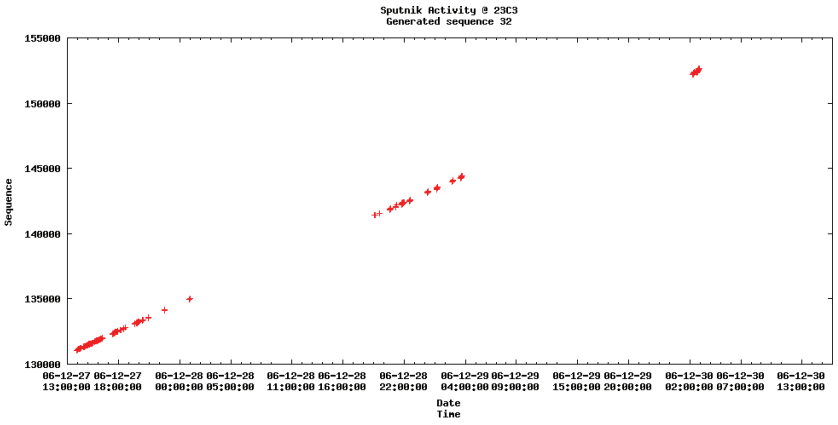


Figure 8: Generated sequence; first set, number 32

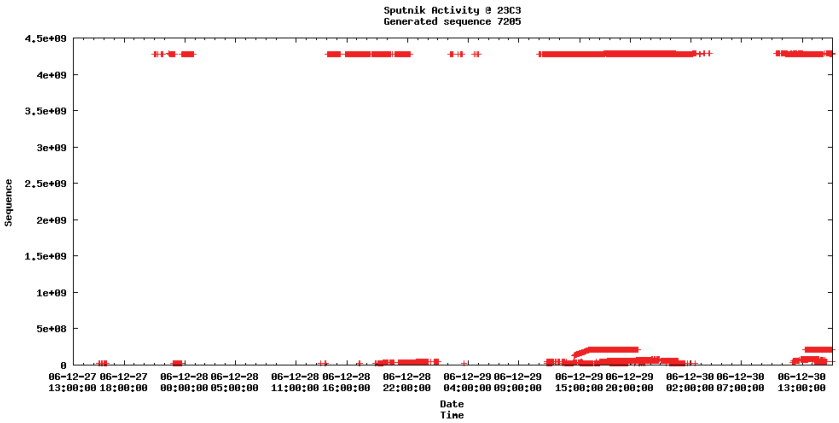


Figure 9: Generated sequence; first set, number 7205

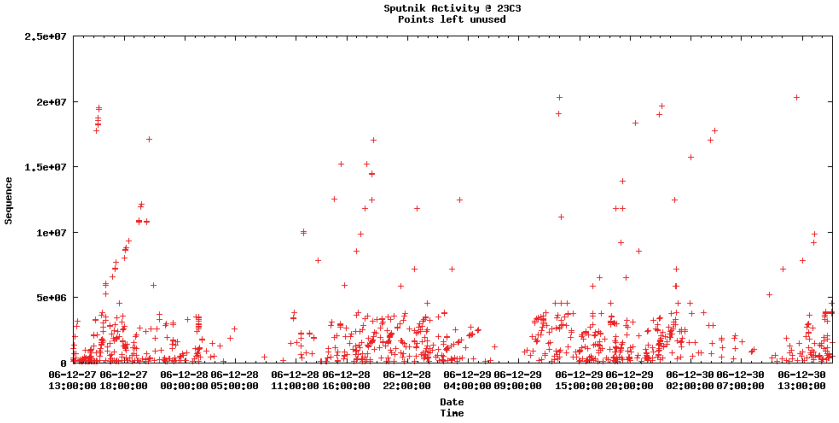


Figure 10: Points left without sequence; first set

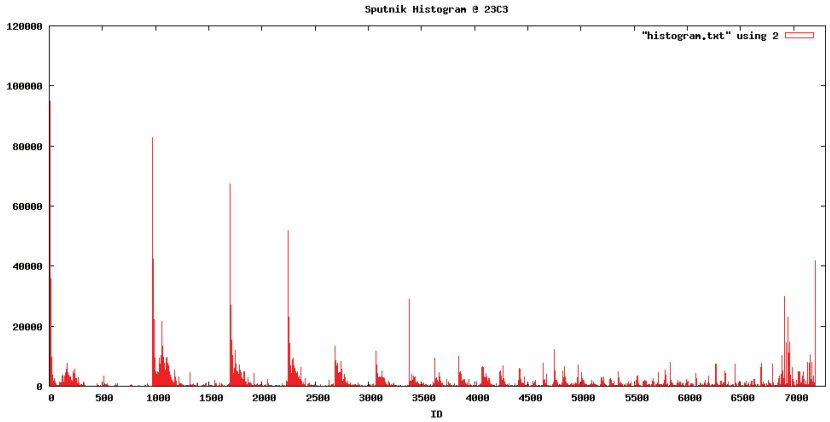


Figure 11: Histogram of sizes of generated sequences for the first set

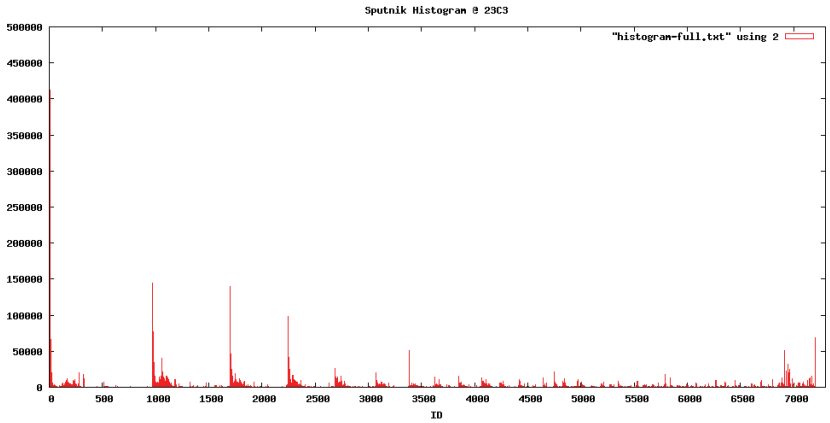


Figure 12: Histogram of sizes of generated sequences for the first set

Program was running for about 72h on AMD Duron 1.3GHz with 768MB RAM and single HDD IDE 7200RPM. It was IO-constraint, probably because of database size larger than available RAM; CPU was not much used. Clustering data table according to counter values could improve performance in the beginning. However PostgreSQL does not try to preserve clustering, so after adding many points to sequences clustering would be lost and Input/Output capacity would again become limiting factor. Also PostgreSQL decides to scan entire table if there is more than 5% rows in result, so in this algorithm entire data table is read.

The main problem with algorithm are sequences that contain point that should belong to many different sequences. This is caused by too wide range of possible coefficient values. The more distant from the initial point, the more visible the problem is.

Figure 13 shows histogram of line coefficients for buckets of size of 0.1. Figure 14 shows histogram of line coefficients for buckets of size of 0.001. As can be seen, first histogram presents false situation; number of points in many lines that consist of small number of points but have close coefficient values is able to outnumber one line with high number of points. So in this situation instead of long one line short one is chosen, and all its neighbours that were able to outnumber the long ones are joined to this improper sequence.

Improvements of algorithm were necessary to get better results. First was refactoring of code; most of activities were moved into functions. Second improvement was creation of SQL aggregate function to choose only one counter value at any given time. This function was used together with grouping with respect to time, and chosen point was the closest one to the chosen slope. To avoid problems with many

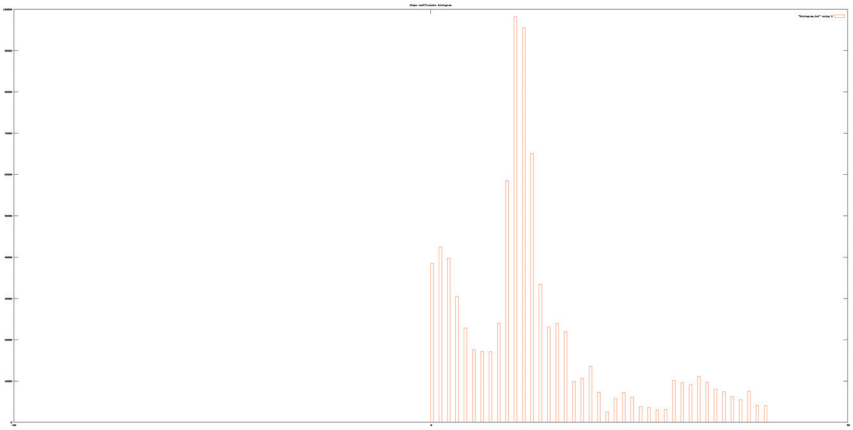


Figure 13: Coefficients histogram for 10 buckets

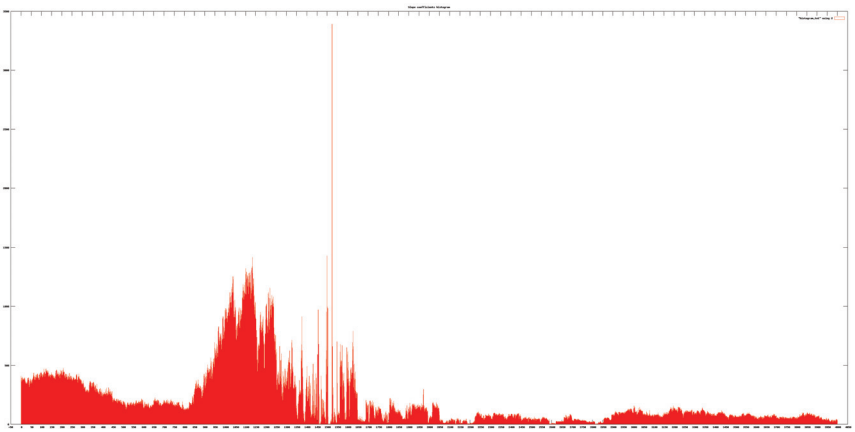


Figure 14: Coefficients histogram for 1000 buckets

lines joining into one width of histogram buckets was changed to 0.001. Histogram was calculated for slopes from range 1.0 to 5.0. Additionally range of allowed coefficients was changed from ± 0.3 to ± 0.001. However this caused gap at the beginning of each sequence; because of rounding errors in the first few minutes slope was not close enough to the ideal to be included in chosen range of slopes.

Function `sputnik_guessbest` is SQL aggregate used to choose one point in case of presence of more than one counter value at the same time. It requires grouping by time in SQL query. It chooses point which distance from the chosen slope is the smallest. To be able to calculate distance from this line it needs to know parameters of line; before using this aggregate function `sputnik_guessinit` must be called. Initialisation function must be called before every query using `sputnik_guessbest`. Both functions are written in pl/Python and use global hash for PostgreSQL Python functions to store line parameters and the best found point.

Currently PostgreSQL in Debian does not offer trusted pl/Python, so untrusted pl/PythonU is used. Creation of functions in untrusted languages requires administrative access to database (usually user "postgres") and SECURITY DEFINER during creation to allow ordinary used to use it.

Grouping function

```
CREATE OR REPLACE FUNCTION sputnik.guessinit(t TIMESTAMP WITH TIME ZONE, sequence BIGINT, slope DOUBLE PRECISIOI
RETURNS VOID
VOLATILE RETURNS NULL ON NULL INPUT SECURITY DEFINER
LANGUAGE 'plpythonu' AS
```

```
$$
GD["time"] = t
GD["sequence"] = sequence
GD["slope"] = slope
$$;

CREATE OR REPLACE FUNCTION sputnik.guessbest(state BIGINT, t TIMESTAMP WITH TIME ZONE, sequence BIGINT)
RETURNS BIGINT
VOLATILE CALLED ON NULL INPUT SECURITY DEFINER
LANGUAGE 'plpythonu' AS
$$
if (not GD.has_key("time")) or (not GD.has_key("sequence")) or (not GD.has_key("slope")):
return None
if (t is None) or (sequence is None):
return None

plan = plpy.prepare("""
SELECT (extract('epoch' FROM ($1::TIMESTAMP WITH TIME ZONE-$2::TIMESTAMP WITH TIME ZONE)))::float/($3::BIGINT-$
""", ["timestamptz", "timestamptz", "int8", "int8"])

result = sequence
if state is not None:
r0 = plpy.execute(plan, [t, GD["time"], sequence, GD["sequence"]], 1)
r1 = plpy.execute(plan, [t, GD["time"], state, GD["sequence"]], 1)
if abs(r0[0]["slope"]-GD["slope"]) >= abs(r1[0]["slope"]-GD["slope"]):
result = sequence
else:
result = state
return result
$$;

CREATE AGGREGATE sputnik.guesser (TIMESTAMP WITH TIME ZONE, BIGINT) (
SFUNC = sputnik.guessbest,
STYPE = BIGINT
);
```

Function `Histogram` calculates histogram of slopes of all lines going through given point. If there is more than one slope with the same maximal number of points, the smallest one is chosen. Function returns slope and number of points in bucket. If it is unable to calculate any slope it returns pair 0, 0.

Histogram function

```
def Histogram(c, time, sequence, sa, sz):
    hash = {}
    c.execute("""SELECT DISTINCT ON (time, sequence) time, sequence,
    (extract('epoch' FROM (time-%s::TIMESTAMP WITH TIME ZONE)))::float/(sequence-%s::BIGINT)::float
    FROM sputnik.sputnik WHERE id IS NULL AND
    sequence BETWEEN %s::BIGINT AND %s::BIGINT AND
    time > %s::TIMESTAMP WITH TIME ZONE AND
    sequence > %s::BIGINT""", (time, sequence, sa, sz, time, sequence))
    i = c.fetchone()
    while i != None:
        k = int(i[2]*1000)
        if 1000 <= k and k <= 5000:
            hash[k] = hash.get(k, 0)+1
        i = c.fetchone()
    if len(hash) > 0:
        m = max(hash.values())
        for i in xrange(1000, 5001):
# Let's take the smallest max
            if m == hash.get(i, 0):
```

```
                result = float(i)/1000.0
                break
        return result, m
    else:
        return 0.0, 0
```

Function `Line` takes as parameters starting point of line, slope of line and allowed range of slopes and finds all points that lie on that line. It initialises global Python hash, as main query uses aggregate `sputnik_guessbest`. It retrieves all matching points from database and returns list holding them.

Function finding points on line with given slope

```
def Line(c, time, sequence, slope, margin, sa, sz):
    result = [[time, sequence]]
    c.execute("""SELECT sputnik.guessinit(%s::TIMESTAMP WITH TIME ZONE,
    %s::BIGINT, %s::DOUBLE PRECISION)""", (time, sequence, slope))
    c.execute("""SELECT time, sputnik.guesser(time, sequence)
    FROM sputnik.sputnik WHERE id IS NULL AND
    sequence BETWEEN %s::BIGINT AND %s::BIGINT AND
    time > %s::TIMESTAMP WITH TIME ZONE AND
    sequence > %s::BIGINT AND
    (extract('epoch' FROM (time-%s::TIMESTAMP WITH TIME ZONE)))::float/(sequence-%s::BIGINT)::float
    BETWEEN %s::float AND %s::float GROUP BY time
    ORDER BY time""", (sa, sz, time, sequence, time, sequence, slope-margin, slope+margin))
    i = c.fetchone()
    while i != None:
        result.append([i[0], i[1]])
        i = c.fetchone()
    return result
```

Function `FindIDs` iterates through all values of counter inside given range, and finds all times when any counter had particular value. Each such pair is treated as potential starting point of line; histogram of slopes is calculated, and if returned bucked holds more than 8 points, new sequence is created. Unlike previous version, this function does not use one update query, but every point is updated by separate SQL command.

Function finding all lines

```
def FindIDs(connection, sa, sz, id):
    c.execute("""SELECT DISTINCT sequence
    FROM sputnik.sputnik WHERE id IS NULL AND
    sequence BETWEEN %s AND %s
    ORDER BY sequence""", (sa, sz))
    start = c.fetchall()
    for s in start:
        s0 = s[0]
        c.execute("""SELECT DISTINCT time FROM sputnik.sputnik
        WHERE id IS NULL AND sequence = %s""", (s0,))
        for t in c.fetchall():
            t0 = t[0]
            slope, count = Histogram(c, t0, s0, sa, sz)
            if slope > 0.0 and count >= 8:
                line = Line(c, t0, s0, slope, 000.1, sa, sz)
                for i in line:
                    UPDATE sputnik.sputnik SET id = %s WHERE id IS NULL AND
                    time = %s::TIMESTAMP WITH TIME ZONE AND
                    sequence = %s::BIGINT
                id += 1
    return id
```

Figures 15 to 19 show sample sequences generated by improved algorithm.

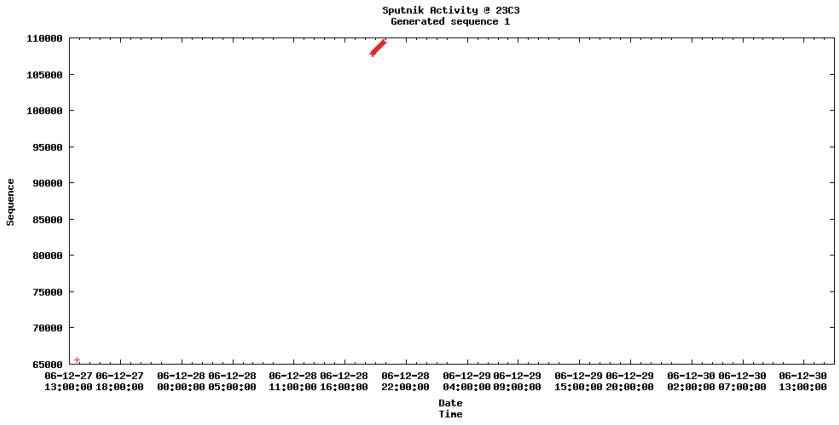


Figure 15: Generated sequence; second set, number 1

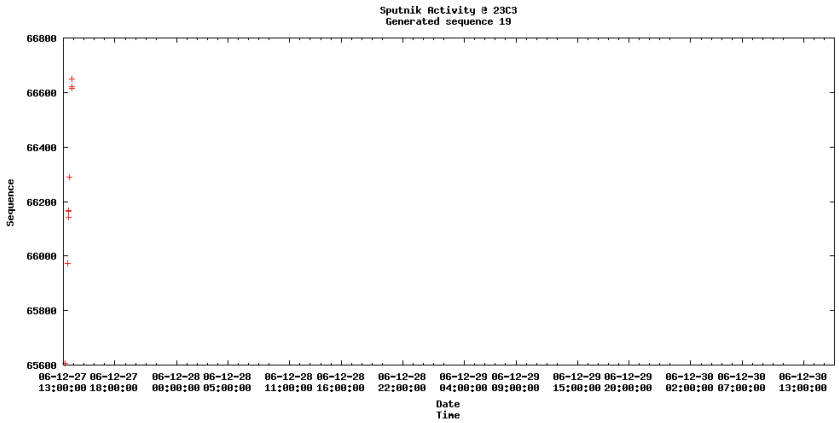


Figure 16: Generated sequence; second set, number 19

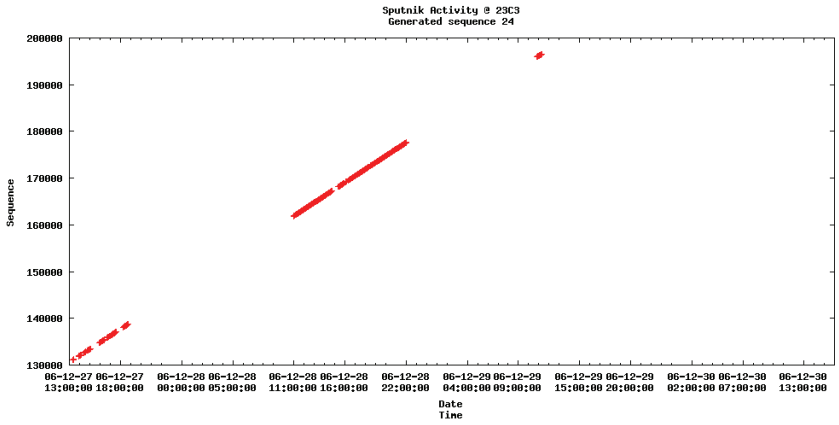


Figure 17: Generated sequence; second set, number 24

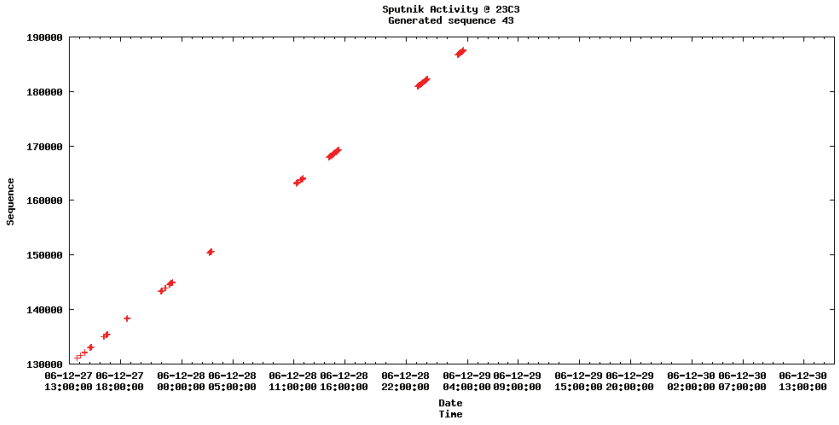


Figure 18: Generated sequence; second set, number 43

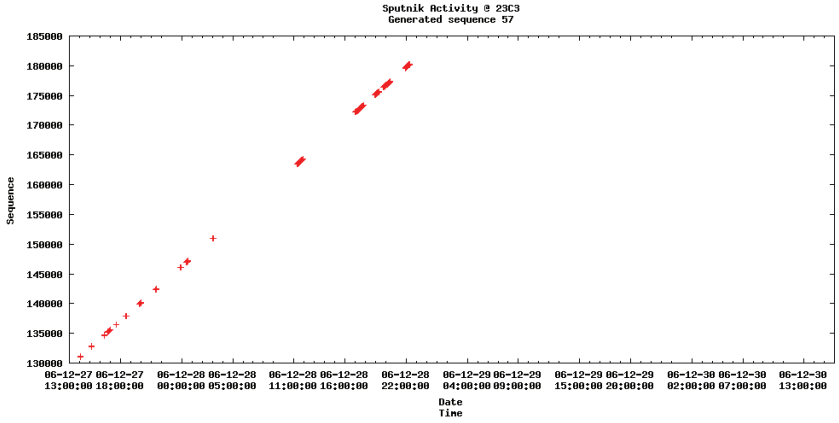


Figure 19: Generated sequence; second set, number 57

Figure 17 shows sequence that is generated by all variants of global algorithm.

Figures 18 and 19 shows generated sequences that have missing some points. Either program did not add some points that should be taken into those sequences or persons wearing those tags was appearing and disappearing from sight of readers.

Figure 20 shows size of generated sequences calculated as number of occurrences of pair (time, counter value); event if packet was seen by more than one reader, it was counted only once. In other words it shows number of occurrences of tag, not how many times it was seen.

Figure 21 shows size of sequences calculated as number of tuples that are included into each sequence.

Program was running very slowly. It was running for almost 2 weeks before I interrupted it. It could not go outside first large data set ($counter \in < 2*65536; 3*65536 >$) so I stopped program and run it for later counter values. It did not leave the next counter values block. It was using IO subsystem and CPU more equally. Its slow speed may come from performing more calculations, using pl/Python function, and updating information about sequences as many individual queries instead of one bulk query.

Generated sequences were initially big, but later they were getting smaller and smaller, down to dozen points.

Algorithm was joining sequences in spite of aggregate function which was used to guard against it. Data analysis was showing that some sequences had errors, but as they were more subtle it was not easily seen on the graphs,

Figure 22 shows two distinct sequences that are joined. Their points are in allowed slope range, and their packets are interlaced, so even aggregate function can not remove one of them.

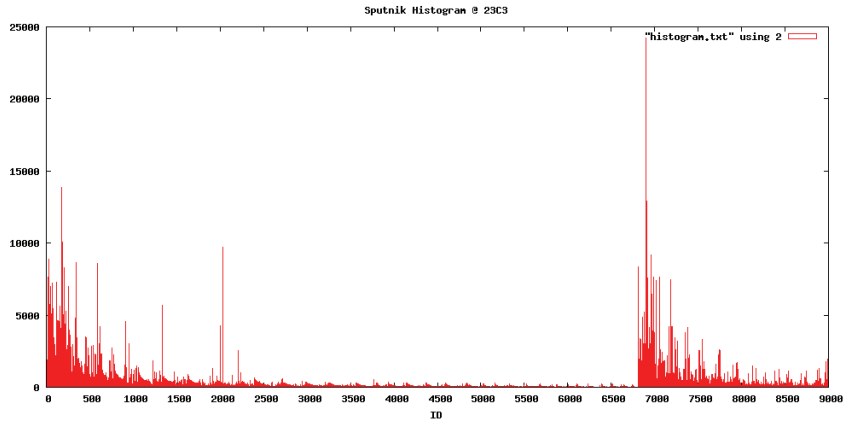


Figure 20: Histogram of sizes of generated sequences for the second set

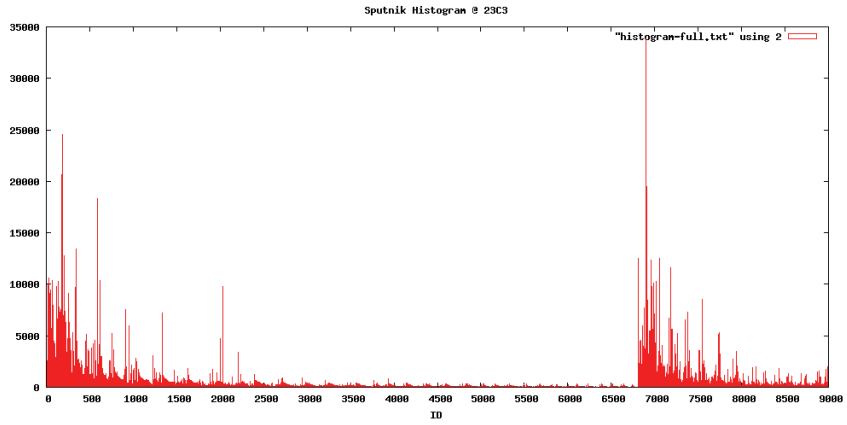


Figure 21: Histogram of sizes of generated sequences for the second set

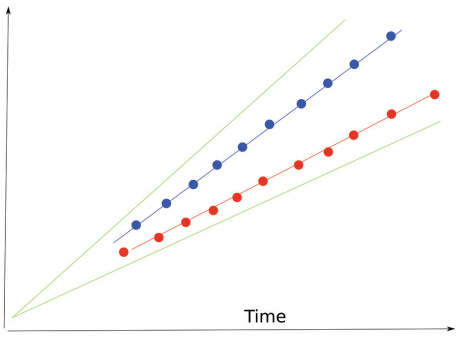


Figure 22: Interlaced sequences

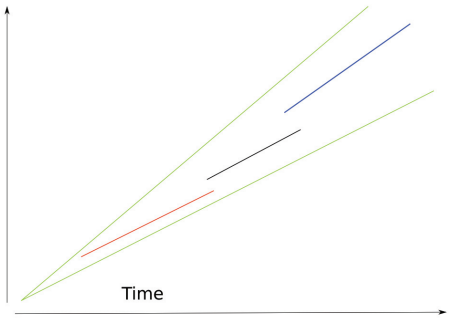


Figure 23: Collinear sequences

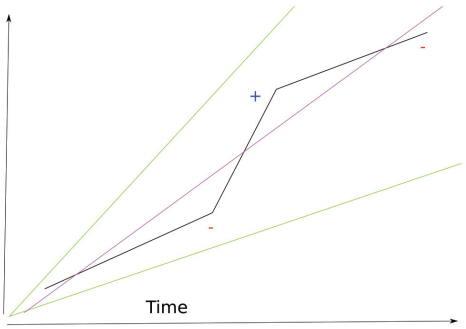


Figure 24: Incorrectly joined sequences

Figure 23 shows three distinct sequences joined into one. They have similar slope and their points lie in allowed range, so they are joined together, even though that points should create distinct sequences.

Figure 24 shows three sequences that have different slopes, but are also joined. This situation can be detected by calculating difference of slopes between consecutive points, similarly to differentiating. The long sequence of differences of the same sign may mean followed by long sequence of differences of another sign suggests join of different sequences.

Figure 25 shows sequence that have points not placed directly on ideal line. It may seem similar to previous situation, but (especially if differences between points and slopes are not large) it is single sequence. The main difference between situation in figures 24 and 25 is number of points that have the same sign of difference between slopes and absolute difference between those slopes. If both of those parameters are small, there is single sequence.

New firmware of tags was released during CCC2007. Transmission was not occurring every few seconds, but about 10 times a second. This, together with USB reader, allowed for analysing if discarding sub-second parts introduces large error in scope of lines. I took few minutes of readings, and calculated two slopes, one taking all data into consideration, and another using floor function to discard milliseconds. Resulted slopes differed on 4th place after comma, so having only seconds when transmission occurred does not result in error disallowing operating on data.

Either having too wide range and having joined sequences, or having too narrow range and leaving some points out, without guarantee that appropriate points are included in sequence meant need for including additional data in searching for good sequences. First of additional variables that could point whether to include tuple into the sequence was signal strength. Each tag changes strength of sent signal, either in sequence of 0x00, 0xff, 0x55, 0xff, 0xaa, 0xff, 0xff, 0xff, or in 0x00, 0x55, 0xaa, 0xff, depending on used firmware version.

First problem would be that in old firmware 5 out of 8 values was 0xff, so it would be difficult to determine where in sequence of signal strengths particular point is. However analysing of source code

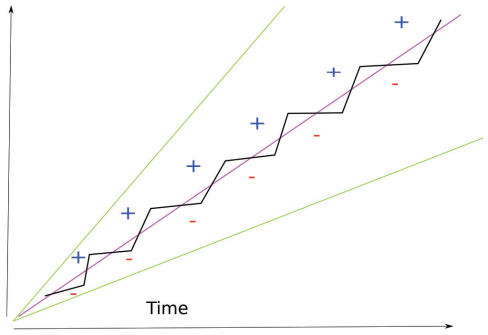


Figure 25: Correctly joined sequence

and Sputnik data revealed that strength of signal was not distinctive between tags. Each tag starts at the same strength sequence point, so there is no variability between sequences. If more than one point has the same counter value, they also have the same strength of signal. It can not be used to distinguish different sequences.

As mentioned earlier, because of rounding errors at the beginning of sequence coefficients do not have the same values as coefficients for further points. It is necessary to have wider allowed range of slopes in the beginning and more narrow near the end. This can be accomplished by sigmoid function[8]. Function $0.01 + \frac{0.09}{1+e^{(x-500)/100}}$ was used in program. At the distance 0 it generated border of 0.1; its value was getting smaller to reach 0.01 for argument of 1000. Because of very large exponential values, FPU exception was generated for arguments greater than about 70000.

Because strength of signal could not be used, stations that received signal from tag were used. The main assumption was that set of seen stations did not change from one point to another if that points were close in time. To keep algorithm simple only list of seen stations was considered, not their distribution in space. Similarity was defined as number of stations in both sets, divided by size of joined sets.

If strengths of signals in both points differ similarity function was slightly changed, and returned number of stations seen using weaker signal divided by number of stations seen with stronger signal. But because most of points in data set had the strongest value of signal, there was not many situations with different signals between points.

To avoid errors shown in Figures 22, 23, and 24, algorithm was changed to retrieve all potential points that could be added to generated sequence and choose the best one itself. This approach is return to the idea of generating alternative sub-sequences used in local algorithm.

Points that are in conflict have condition $\neg(T_1 > T_0 \wedge S_1 > S_0)$ met. Program creates all possible sub-sequence from them and then chooses the best one. To choose the best it locally compares lengths, slopes of sub-sequences and reading stations seen by all sub-sequences and chooses one that is the most similar to main sequence.

Last version of algorithm differs from previous ones, and those changes can be summarised in “take more points and choose the best ones”. Instead of using constant range, sigmoid function was used to include more points in the beginning of sequence. All points are read from database, and program builds alternative sequences from them. Instead of using custom aggregate function to choose only one point, standard function aggregating all seen stations into array is used. This array is then used to choose the best points to include into sequence. The last change is breaking line if it is discovered that created line has high probability of being two different lines.

Function `Similarity` returns number from range $< 0.0; 1.0 >$. This is degree of similarity of two sets of readers that were able to receive signal from tag. Function uses sets introduced in Python 2.4.

Similarity of seen stations

```
def Similarity(a, b):
    result = 0.0
    station0, strength0 = a
```

[8] http://en.wikipedia.org/wiki/Sigmoid_function

```
    station1, strength1 = b
    size0, size1 = len(station0), len(station1)
    if strength0[0] > strength1[0]:
        same = 0.0
        for i in station1:
            if i in station0: same += 1
        result = same/len(station1)
    elif strength0[0] < strength1[0]:
        same = 0.0
        for i in station0:
            if i in station1: same += 1
        result = same/len(station0)
    else:
        result = float(len(set(station0)&set(station1)))/
            float(len(set(station0)|set(station1)))
    return result
```

Function `Fetch` reads all points from database that can be used to create sequence. It takes all packets that were received less than two minutes after first point of sequence, and then returns those which slope lies in range determined by sigmoid function.

Getting all points that can create line

```
def Fetch(c, time, sequence, slope, sa, sz):
    result = [[time, sequence, slope, 0.0]]
    c.execute("""SELECT sputnik.array_accum(station),
    sputnik.array_accum(strength)
    FROM sputnik.ccc23 WHERE id IS NULL AND
    time = %s::TIMESTAMP WITH TIME ZONE AND
    sequence = %s::BIGINT""", (time, sequence))
        i = c.fetchone()
    if i != None:
        result[0].append(i[0])
        result[0].append(i[1])
    i = c.fetchall()
# Union of first 100s and the rest
    c.execute("""SELECT time, sequence,
    (extract('epoch' FROM (time-%s::TIMESTAMP WITH TIME ZONE)))::float/(sequence-%s::BIGINT)::float,
    0.0, sputnik.array_accum(station), sputnik.array_accum(strength)
    FROM sputnik.ccc23 WHERE id IS NULL AND
    sequence > %s::BIGINT AND sequence <= %s::BIGINT+100::BIGINT AND
    time > %s::TIMESTAMP WITH TIME ZONE AND time <= %s::TIMESTAMP WITH TIME ZONE+'100 second'::INTERVAL
    GROUP BY time, sequence
    UNION
    SELECT time, sequence,
    (extract('epoch' FROM (time-%s::TIMESTAMP WITH TIME ZONE)))::float/(sequence-%s::BIGINT)::float,
    0.0, sputnik.array_accum(station), sputnik.array_accum(strength)
    FROM sputnik.ccc23 WHERE id IS NULL AND
    sequence BETWEEN %s::BIGINT AND %s::BIGINT AND
    time > %s::TIMESTAMP WITH TIME ZONE AND
    sequence > %s::BIGINT AND
    (extract('epoch' FROM (time-%s::TIMESTAMP WITH TIME ZONE)))::float/(sequence-%s::BIGINT)::float
    BETWEEN %s::float-sputnik.BorderWidth(sequence-%s) AND %s::float+sputnik.BorderWidth(sequence-%s)
    GROUP BY time, sequence ORDER BY time""", (time, sequence, sequence, sequence, time, time, time, sequence,
    i = c.fetchone()
    while i != None:
        result.append([i[0], i[1], i[2], i[2]-result[-1][2], i[4], i[5]])
        i = c.fetchone()
    return result
```

Function `Lines` takes list of all points that were read from database and creates all possible sequences from them. It is similar to function used in local algorithm.

Calculating all possible sequences from points

```
def Lines(data):
    result = [] candidate = []
    for i in data:
        num = 0
        for j in candidate:
            if i[0] > j[-1][0] and i[1] > j[-1][1]:
                num += 1
        if len(candidate) == num:
            if len(candidate) == 1: result.extend(candidate[0])
            elif len(candidate) > 1: result.append(candidate)
            candidate = [[i]]
        else:
            for j in candidate:
                if i[0] > j[-1][0] and i[1] > j[-1][1]:
                    j.append(i)
            if 0 == num: candidate.append([i])
# Add last alternative
    if len(candidate) == 1: result.extend(candidate[0])
    elif len(candidate) > 1: result.append(candidate)
    return result
```

Function `Line` takes all sub-sequences and chooses the best line from all given alternatives. Each of alternatives has calculated up to five factors that are taken into consideration: length, similarity of slopes in the beginning and in the end, similarity of seen stations in the beginning and in the end. Only the best sub-sequence gets points for each factor, and then only the best one is chosen. If there is more than one best alternative, the first one is chosen.

The very important part of this function if condition $j[0][0] > result[-1][0]\ldots$ which allows only sub-sequences which time and counter values are greater than already existing in sequence to be considered as alternatives. This protects from the problem of having improper sequence in case when one alternative choosing after another.

Choosing the best line from all alternatives

```
def Line(lines):
    result = []
    for i in xrange(len(lines)):
        if type(lines[i][0]) != type([]): result.append(lines[i])
        else: alternatives = []
            if len(result) > 0:
                for j in lines[i]:
                    if j[0][0] > result[-1][0] and j[0][1] > result[-1][1]: alternatives.append(j)
            else: alternatives = lines[i]
            scores = [0] * len(alternatives)
            sizes = map(lambda x: len(x), alternatives)
            best = max(sizes)
            for j in xrange(len(alternatives)):
                if sizes[j] == best: scores[j] += 1
            stationsa = map(lambda x: Similarity((result[-1][4], result[-1][5]), (x[0][4], x[0][5])), alternati
# Find best alternative for stations in the beginning
            if i+1 < len(lines) and type(lines[i+1][0]) != type([]):
                stationsz = map(lambda x: Similarity((x[-1][4], x[-1][5]), (lines[i+1][4], lines[i+1][5])), alt
# Find best alternative for stations in the end
            slopesa = map(lambda x: abs(alternatives[x][0][3]-result[-1][3]), xrange(len(alternatives)))
# Find best alternative for slopes in the beginning
            if i+1 < len(lines) and type(lines[i+1][0]) != type([]):
                slopesz = map(lambda x: abs(alternatives[x][0][3]-lines[i+1][3]), xrange(len(alternatives)))
```

```
# Find best alternative for slopes in the end
# Find the best alternative:
            best = max(scores)
            for j in xrange(len(alternatives)):
                if scores[j] == best:
                    result.extend(alternatives[j])
                    break
# Count slope deltas once more, for final line proposal
    slope = result[0][2]
    for i in result:
        i[3] = i[2]-slope
        slope = i[2]
    return result
```

Function `Break` takes four consecutive points a, b, c, and d and returns number from range $< 0.0; 1.0 >$, the probability that line should be broken between points b and c, because they belong to different lines. It takes six factors into consideration: difference in slopes between lines a-b and b-c, and b-c and c-d, difference in time between following points, similarity of seen stations between points b and c, and absolute changes of slope between local and global value.

Function returning probability of break

```
def Break(a, b, c, d, slope):
    result = 0.0
    SlopeDiff = 10.0
    SlopeTrigger = 0.01
    CounterDiff = 100
    TimeDiff = datetime.timedelta(0, 120)
    StationSimilarity = 0.5
    if abs(c[3]) > SlopeTrigger:
        if abs(c[3]) > abs(b[3])*SlopeDiff: result += 1.0
        if abs(c[3]) > abs(d[3])*SlopeDiff: result += 1.0
# Time is more intuitive that sequence counter
# Also I do not have to think about line coefficient
#    if c[1] - b[1] > CounterDiff: result += 1.0
    if c[0] - b[0] > TimeDiff: result += 1.0
    if Similarity((b[4], b[5]), (c[4], c[5])) < StationSimilarity: result += 1.0
    SlopeAB = float((b[0]-a[0]).seconds)/(b[1]-a[1])
    SlopeBC = float((c[0]-b[0]).seconds)/(c[1]-b[1])
    SlopeCD = float((d[0]-c[0]).seconds)/(d[1]-c[1])
# Slopes should be similar to each other and to the main slope
    if slope-1.0 <= SlopeAB and SlopeAB <= slope+1.0 and (SlopeBC < slope-1.0 or slope+1.0 < SlopeBC):
        result += 1.0
    if slope-1.0 <= SlopeCD and SlopeCD <= slope+1.0 and (SlopeBC < slope-1.0 or slope+1.0 < SlopeBC):
        result += 1.0
    return result/6.0
```

Main function `FindIDs` calls all previous functions and generates sequence. It decides to break line if probability returned by function `Break` is more than 0.5, in such case of iteration of loop creates more than one sequence.

Function creating all lines

```
def FindIDs(connection, sa, sz, id):
    c.execute("""SELECT DISTINCT sequence FROM sputnik.ccc23 WHERE id IS NULL AND
    sequence BETWEEN %s AND %s ORDER BY sequence""", (sa, sz))
    for s in c.fetchall():
        s0 = s[0]
        c.execute("""SELECT DISTINCT time FROM sputnik.ccc23
        WHERE id IS NULL AND sequence = %s""", (s0,))
        for t in c.fetchall():
```

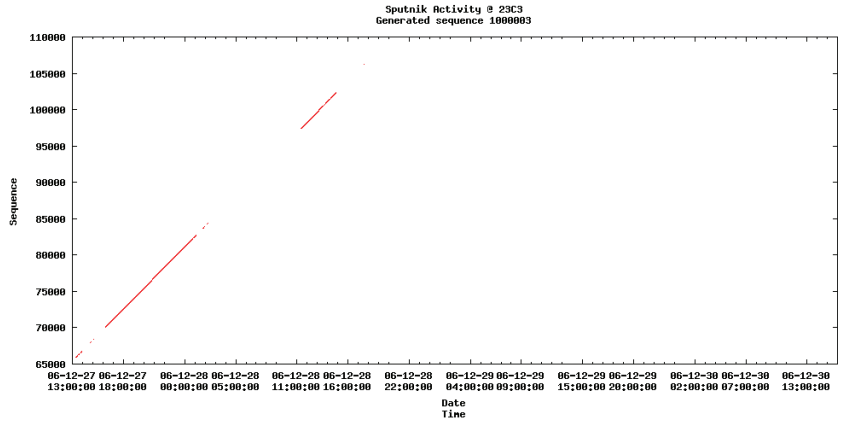


Figure 26: Generated sequence; third set, number 3

```
            t0 = t[0]
            slope, count = Histogram(c, t0, s0, sa, sz)
            if slope > 0.0 and count >= 8:
                data = Fetch(c, t0, s0, slope, sa, sz)
                lines = Lines(data)
                line = Line(lines)
                for i in xrange(len(line)):
                    skip = False
                    if len(line[i][4]) != len(line[i][5]):
                        print "Error in size of ", line[i]
                        skip = True
                    s = line[i][5][0]
                    for j in line[i][5]:
                        if j != s:
                            print "Error in strength of ", line[i]
                            skip = True
                    if skip:
                        break
                    UPDATE sputnik.sputnik SET id = %s WHERE id IS NULL AND
                    time = %s::TIMESTAMP WITH TIME ZONE AND sequence = %s::BIGINT
                    if i > 0 and i < len(line)-2:
                        b = Break(line[i-1], line[i], line[i+1], line[i+2], slope)
                        if b > 0.5:
                            id += 1
                            print "Break here, new id ", id, b
                id += 1
        return id
```

Figures 26 to 30 show some of sequences generated by improved algorithm.

Figure 27 shows sequence that is generated by all variants of global algorithm.

Figure 31 shows size of generated sequences calculated as number of occurrences of pair (time, counter value); event if packet was seen by more than one reader, it was counted only once. In other words it shows number of occurrences of tag, not how many times it was seen.

Figure 32 shows size of sequences calculated as number of tuples that are included into each sequence.

Program was run on different machine than previous ones. It was running 5634 minutes on 64 bit AMD 3400+ with 1GB of RAM and one IDE HDD 7200RPM. It was stopped by FPU error in sigmoid function for large values of counter. 10.6 million rows was used in generated sequences. Over 1600 sequences were made from more than 1000 points.

Because many of generated sequences were short, the next step should be joining of them. One solution is to try to join existing sequences, another could be trying to extend sequences by points not

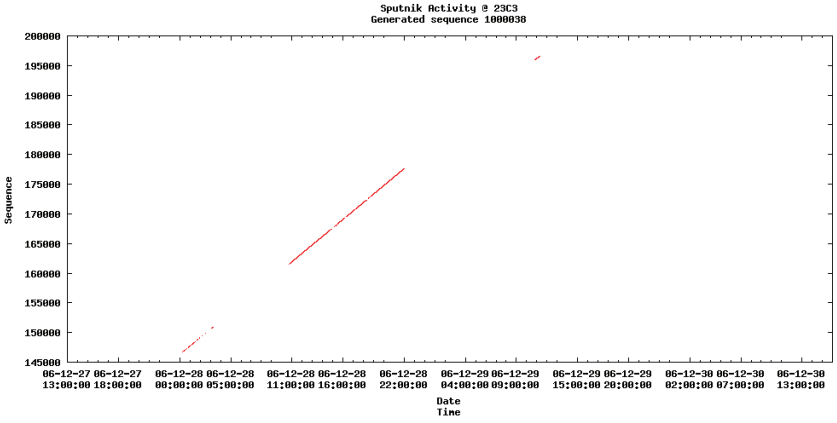


Figure 27: Generated sequence; third set, number 38

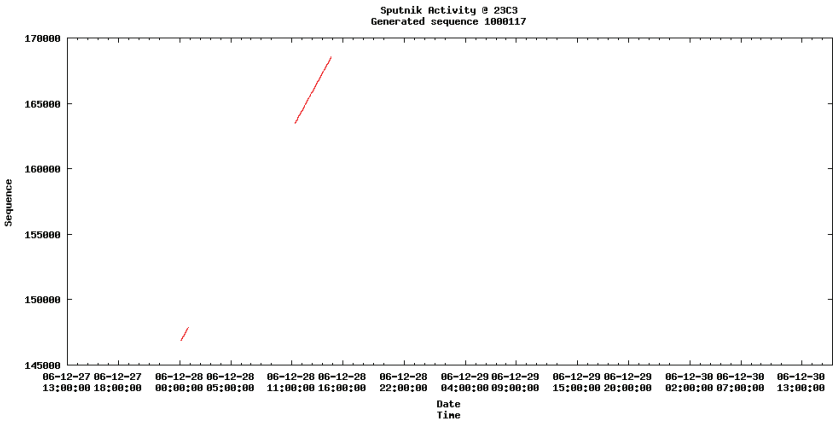


Figure 28: Generated sequence; third set, number 117

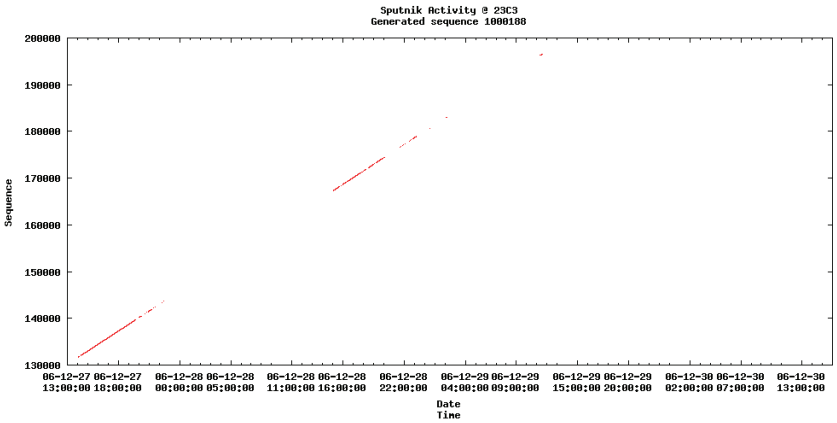


Figure 29: Generated sequence; third set, number 188

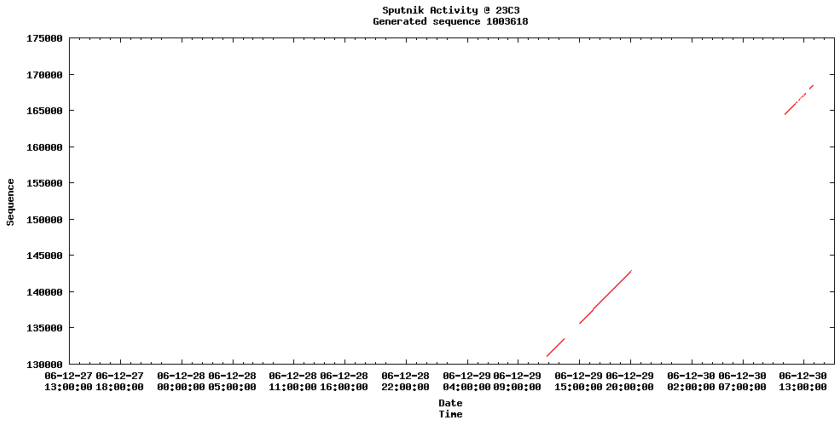


Figure 30: Generated sequence; third set, number 3618

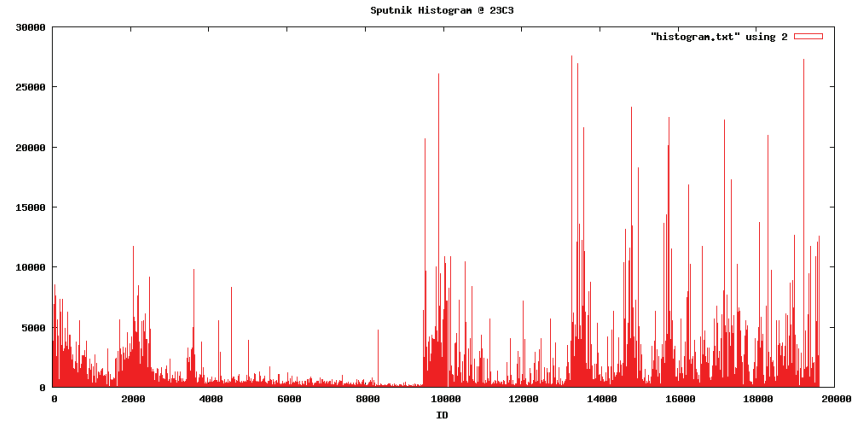


Figure 31: Histogram of sizes of generated sequences for the third set

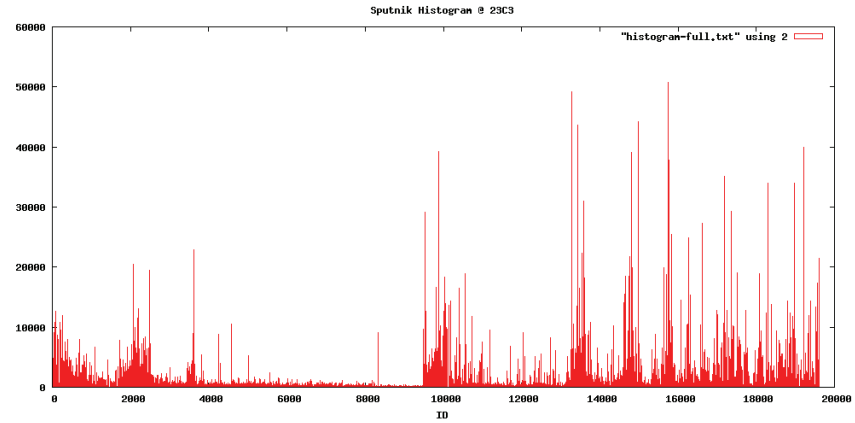


Figure 32: Histogram of sizes of generated sequences for the third set

belonging to any sequence. But problem with joining is choosing which sequence to join with each another. Which sequence from those shown in Figures 26, 27, 28. 29 should be joined to the one shown in Figure 30? It could be different case of `Break` function. If none of the causes for break occurs, there is possibility of join. Another possible solution is manual joining. Program could display few candidates and let user choose which ones look best together. If manual joining is success, this approach could be used to change generating algorithm and allow for manual choosing of alternative sub-sequences.

Knowledge gathered during analysing data and generating sequences leaves some doubts. I started with assumption that each tag sends packet every 1.5s. This lead to setting coefficient range from 1s to 2s. Because this was not giving good results in local algorithms, and by observing scatter plots, global algorithms were using range from 0.0 to 10.0, and later, basing on analysing source code of Sputnik firmware, from 1.0 to 5.0, Source code of firmware contains two calls of sleep function. One sleeps for 2s, and another for random period from 0s to 2s. This gives range of line slopes from 2s to 4s. But because second sleep function parameter is random value, there should be no straight line! However scatter plots reveal many of them. So either Sputnik data contains so many points that one can draw any line, or function rand() returns not very random numbers. Basing on analysing packets generated by single tag, second possibility is true.

Fragment of firmware of tag

```
void main (void)
{// get random seed
  ((unsigned char *) &seq)[0] = EEPROM_READ (4);
  ((unsigned char *) &seq)[1] = EEPROM_READ (5);
  ((unsigned char *) &seq)[2] = EEPROM_READ (6);
  ((unsigned char *) &seq)[3] = EEPROM_READ (7);
// increment code block after power cycle
  ((unsigned char *) &crc)[0] = EEPROM_READ (8);
  ((unsigned char *) &crc)[1] = EEPROM_READ (9);
  store_codeblock (++crc);
  seq ^= crc;
  srand (crc16 ((unsigned char *) &seq, sizeof (seq)));
// increment code blocks to make sure that seq is higher or equal after battery change
  seq = ((u_int32_t) crc) << 16;
  i = 0;
  while (1) {
// update code_block so on next power up the seq will be higher or equal
      crc = seq >> 16;
      if (crc == 0xFFFF) break;
      if (crc == code_block) store_codeblock (++crc);
// encrypt my data
      shuffle_tx_byteorder ();
      xxtea_encode ();
      shuffle_tx_byteorder ();
// send it away
      nRFCMD_Macro ((unsigned char *) &g_MacroBeacon);
      CONFIG_PIN_LED = 1; nRFCMD_Execute (); CONFIG_PIN_LED = 0;
// reset touch sensor pin
      TRISA = CONFIG_CPU_TRISA & ~0x02; CONFIG_PIN_SENSOR = 0;
sleep_jiffies (0xFFFF);
      CONFIG_PIN_SENSOR = 1; TRISA = CONFIG_CPU_TRISA;
// sleep a random time to avoid on-air collosions
sleep_jiffies (rand ());
      i++;
    }
}
```

No physical (or geometrical) model was taken into consideration during generating sequences. No distance between stations or speed of movement was analysed. This could give better results in sequences, by limiting point to only those that are in range to reach from previous point. On the other hand this approach would require calculating position of each tag in every moment.

5.3 Analysis

Following paragraphs describe potential approaches. They base on validity of generated sequences. I did not yet performed any analysis of data using generated sequences, as recovering them was my primary concern.

XML data set proves that it is possible to calculate position of tag. Tags send packets with different signal strength to allow for estimation of distance from reader. This estimation bases on negative knowledge. If reader is unable to read signal with small strength it means that tag is far away from it. So having few packets it is possible to calculate minimal and maximal distance tag is from reader. Power of signal was set so next level of power increases twice radius of range. This gives two spheres with small and large radius; person is between them. When data from few readers is known, it is possible to calculate common fragment of space where all those spheres intersect, and this is position of tag. But this requires knowing exact positions of readers.

Human body decreases strength of signal. This decreases precision of estimating position of tag. But maybe this could be used to calculate direction person has, assuming that tag is worn in the front. Range would not be sphere, but two hemispheres, larger in the front and smaller in the back. This would require performing more calculations (two times for each reader), but as there is no situation when all readers see one tag, it would not be impossible. Direction could be proven when person moves in this direction, again with assumption that person walks forwards, not backwards.

Simple analysis is calculating time of entering BCC and leaving it. Most people leave Center for the night, but some stay. Also when one sequence disappears and another one appears in the same place it means that someone is playing with battery and reset tag.

The most interesting analysis is looking for connections and similarities between attendees. This can be done by looking for people that attended similar talks. Those people may not even know each other but have common interests.

Another research area is looking for friends. Friends can be defined as people that stay together; they tend to be together not only during talks, but also and especially during breaks. If two people are close during most breaks, they are close friends. If they are close for some times, and not close for other moments, they may be colleagues. Or they may just stay in the same queue for pizza. However here the most important is relative position (distance between people), not exact position of tags.

This data set leves many conclusions to be drawn.

Hacking
lecture
Tag 1 21:45
Saal 2
de

Daniel Otte, Sören Heisrath

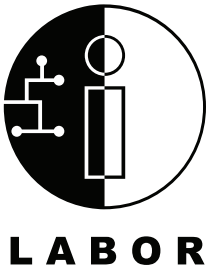
AnonAccess

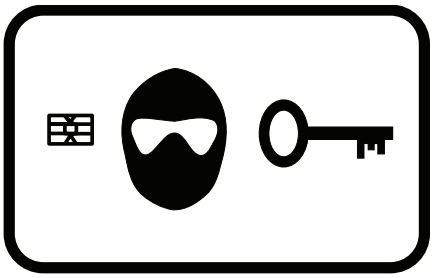
Ein anonymes Zugangskontrollsystem

AnonAccess ist ein elektronisches System, welches anonymen Zugang nicht nur zu Hackerspaces ermöglicht.

Mit Hilfe kryptographischer Verfahren kann das Mikrocontroller-basierende System verblüffend einfach sicheren und anonymen Zugang kontrollieren.Es wird das Zusammenspiel verschiedener Primitiven unter Berücksichtigung der Limitierungen eingebetteter Systeme gezeigt. Angriffsszenarien und Anforderungen an derartige Systeme stellen einen weiteren Beobachtungsgegenstand da.Gezeigt wird das komplette System von der ICC-Speicherkarte über die gesicherte Kommunikation bis zur verschlüsselten Datenbank.

http://www.das-labor.org/wiki/AnonAccess AnonAccess im Labor wiki





AnonAccess

das Labor
http://www.das-labor.org

Daniel Otte
daniel.otte@ruhr-uni-bochum.de

Sören Heisrath
sh@3dots.de

December 3, 2007

Abstract

This paper gives an overview of the AnonAccess-system, which tries to provide access to users which may be known by name, pseudonym or a shared pseudonym, to a given functionality (ex. open a door). The shared pseudonym access feature is tried to be extended and implemented in such a way that it can be claimed to be anonymous.

1 Notations and conventions

$a \leftarrow b$	a is asigned the value of b
$a \oplus b$	a xor b
$a \wedge b$	a bit wise and b
$a \vee b$	a bit wise or b
$a \parallel b$	concatenation of the bit strings a and b
$a_{(base)}$	the constant a is given in base $base$ notation, if not specified the base is 10
$H(a)$	is the value of the hash function SHA-256 of message a
$HMAC_{key}(a)$	is the value of the HMAC-SHA256 MAC function of message a and key key
bit	a bit is the basic unit of information; it can only have one of two values, which we consider to be 1 and 0
byte	a byte is considered to be a group of eight bits throughout this document
Ki, Mi, Gi	prefixes to units, specifying a multiple of $2^{10} = 1,024$, $2^{20} = 1048,576$ and $2^{30} = 1,073,741,824$; see [1] for reasons
K, M, G	prefixes to units, specifying a multiple of $10^3 = 1,000$, $10^6 = 1,000,000$ and $10^9 = 1,000,000,000$

2 Cryptographic algorithms used

We use the following cryptographic primitives:

- SHA-256 hash function as specified in [2]
- HMAC-SHA256 MAC function as specified in [3]
- Shabea with 16 rounds as data encryption algorithm as specified in appendix B
- a PRNG as specified in appendix A

3 Components

The AnonAccess system is divided in *Terminal-Unit* and *Master-Unit*, additionaly there is a chip-card for each user, which stores the user's authentication data.

3.1 Chip-Card

We use simple memory cards with I^2C-Bus[4] and form factor ID-1 as specified in [5][6]. They are quite cheap (less then 1€ per card) and not secure. Their contents might easily be read or modified, so everyone can read and check what we write on his/her card.

The card contains a so called *AuthBlock* embedded in an ASN.1-BER[7] octal-string object. The *AuthBlock* has the following structure:

name	size	description
UID	2 bytes	index to the *TicketDB*
ticket	32 bytes	ticket containing encrypted time-stamp
r_{key}	32 bytes	random key for r_{ID} decryption
r_{ID}	32 bytes	encrypted user pseudonym
HMAC	32 bytes	$HMAC_{absign\_key}(UID \parallel ticket \parallel r_{key} \parallel r_{ID})$

3.2 Terminal-Unit

The *Terminal-Unit* handles user inputs, displays information and reads and writes the user's card. It is equipped with keypad, display, card reader and a hardware random number generator. It'power is supplied by the *Master-Unit* and it should therefore not be reset even in the case of power failure.

3.3 Master-Unit

The *Master-Unit* keeps the databases, does the authentication and executes the secured action (ex. opens a door).

3.4 Power supply

The power supply is designed to power the *Terminal-Unit* and the *Master-Unit*. It uses an accumulator to work as uninterruptible power supply, so that about 60 hours of operation without external power supply should be possible. Therefore under normal circumstances a reset due to a power failure should not happen.

3.5 Real time clock (RTC)

The real time clock is implemented in software by using one of the microcontroller's timers. A timer interrupt function increments a 64bit value each millisecond (this counter will wrap around in about 584.542.046 years, which should be quite enough for us). Additionally the counter's value is periodically[1] written to the microcontroller's EEPROM and read back after reset. On reset we also add the value $3FFFFF_{(16)}$ to the counter to avoid having the same timestamp for more than one time.

The backup storage is implemented in a ring buffer structure with an additional index byte. The index byte indicates which cell of the ring buffer is to be used. After writing a value to a cell it is read back and checked. If the check fails the index byte is incremented by one and the next cell is used. The EEPROM is specified to be written 100,000 times so one cell may work for 116,508.4 hours which is about 13.29 years. So with a ring buffer of 20 cells, we should be able to operate for about 265.82 years which should be sufficient for most applications (if not the ring buffer could be easily made even larger).

It should be known that the timer value does not necessarily correspond to a linear continuous time line or human time, although the time is monotonic increasing.

[1] the value is backed up every $3FFFFF_{(16)}$ milliseconds which is about every 1.165 hours

3.6 Microcontroller

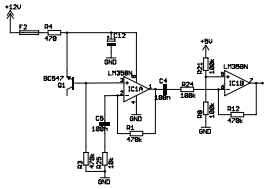
We use microcontrollers from the ATmega family from Atmel[13]for both units. They are relatively cheap and support protection of the internal memories (flash and EEPROM) from being read through their lock-bit feature. There also is a toolchain including GCCs[16] C-compiler and a libc implementation[17] available for these 8 bit microcontrollers which eases the writing of the software.

The *Master-Unit* uses an ATmega644[14] in DIL-Package with 64KiB of program flash, 4KiB of internal SRAM and 2KiB of internal EEPROM (100,000 rewrite cycles guaranteed).

The *Terminal-Unit* uses an ATmega32[15] in DIL-Package with 32KiB of program flash, 2KiB of internal SRAM and 1KiB of internal EEPROM (100,000 rewrite cycles guaranteed).

3.7 Random number generator (RNG)

This circuit utilises the randomness of the transistor diode's breakdown current to generate random voltages in the range from 0 to 5 volts. While this is quite random it does not need to be cryptographically secure, because the RNGs output is used only as input for the cryptographically secure PRNG.



schematic of the hardware random generator

3.8 Pseudo-random number generator (PRNG)

The PRNG is based on the SHA-256 hash function and is specified in appendix A. It has two main functions:

- AddEntropy: this function adds data to the entropy pool, the input can be of arbitrary bit length
- GetRandomBlock: this function fills a 32 byte block of memory with a randomised bit string

Another function (GetRandomByte) uses a buffer and the GetRandomBlock function and returns a random byte. The PRNG is periodically filled with entropy from the hardware RNG using the AddEntropy function.

3.9 Secure serial port (QPort-tiny)

QPort-tiny[11] is a software stack which offers a secure communication channel over an insecure serial line. For that purpose it uses a pre-shared secret key to agree on a set of secret symmetric keys, which are then used for encryption. HMAC-SHA256 is used for session key generation, and XTEA[12] is used in OFB and CFB mode for encryption.

3.10 External serial EEPROM

The external serial EEPROM is used to keep the ticket databases and the flag-modify database, and can be used for key-storage in the migration process.

We use standard I^2C[4] EEPROMs with 512KiBit or 1MiBit (24xx512[8] or 24xx1025[9]) from Microchip[10]. It is possible to extend the storage capabilities by using multiple EEPROMs. That makes it possible to have up to 4MiBit or 512KiBytes of storage space which normally allows more than 10,000 users.

All contents of the EEPROM are encrypted (except the keymigration-area). Shabea-16 is used to encrypt the content. We therefore divide the EEPROM space into 32 byte blocks which are encrypted separately. Every block is encrypted with an individual key which is the result of concatenation of the "mainkey"(*eepromcrypt_key*) and the block address. So we are protected from most attacks against mass storage encryption (ex. watermarking).

3.11 Ticket-Database (TicketDB)

This database is used to store a HMAC of the user's ticket, her/his permissions, and some statistics about the whole system. The first element in the database is the header followed by the entries for the users.
Header structure:

name	size	description
ID	10 bytes	set to the ASCII string "AnonAccess"
majversion	1 byte	major version; set to 1
minversion	1 byte	minor version; set to 0
headersize	1 byte	specifies the size of the header
stat	10 bytes	statistics
reserved	8 bytes	reserved field for future extensions and for alignment; set to 0

The statistics field has the following structure:

name	size	description
max_users	2 bytes	maximum number of users
users	2 bytes	actually active user
admins	2 bytes	actually active admins
locked_users	2 bytes	number of locked users
locked_admins	2 bytes	number of locked admins

The following space of the *TicketDB* is filled with user entries which have the following structure:

name	size	description
flags	1 byte	the flags associated with the user
nickname	7 bytes	the nickname if the user decided to be known by name
ticketmac	32 bytes	HMAC from users ticket

Where the flag field has the following structure:

name	size	description
exists	1 bit	indicates if this entry is used (1: in use; 0: free)
admin	1 bit	set if user has admin privileges, cleared otherwise
locked	1 bit	set if user is locked; cleared otherwise
notify_lostadmin	1 bit	set if user has to be notified about lost admin privileges
anonymous	1 bit	set if the user did not specify user name to be stored
reserved	3 bit	reserved, should be set to 0

3.12 FlagModifying-Database (FLMDB)

The flag-modifying-Database keeps entries which specify how a given user account should be modified.

name	size	description
active	1 byte	set to 1 if this entry is active; set to 0 otherwise
permanent	1 byte	set to 1 if this entry should not be removed if applied; set to 0 otherwise
last	1 bytes	if set to 1 this is the last entry to check; set to 0 otherwise
setflags	1 byte	specifies which bits have to be set in the userflags
clearflags	1 byte	specifies which bits have to be cleared in the userflags
reserved	3 byte	reserved; set to 0
timestamp	8 bytes	timestamp of the creation of this entry
hnick	32 bytes	HMAC of the *user pseudonym*

3.13 Key-Database (Key-DB)

This database stores all the cryptographic keys used in the system.

name	size	description
ticket_key	256 bit	used to generate the HMAC from the ticket which is stored in *TicketDB*
absign_key	256 bit	used to generate the HMAC in the *AuthBlock*
rid_key	256 bit	used to encrypt the *user pseudonym*
nick_key	256 bit	used to generate the HMAC from the user's nickname giving the *user pseudonym*
timestamp_key	256 bit	used to generate a new ticket by encrypting a 24 byte random string and a 8 byte timestamp
eepromcrypt_key	256 bit	used for encrypting the external EEPROM's content

4 Being known by name or shared pseudonym

AnonAccess allows three ways of being known:

- being known by name
- being known by pseudonym
- being known by a shared pseudonym

4.1 Being known by name

If the user selects to be known by name the nickname is stored in the *TicketDB* in a way that is available in plaintext to the *Master-Unit*. It can be searched for and it can be read by an administrator. This allows immediate manipulation of the user's flags.

4.2 Being known by pseudonym

In every mode the user enters his/her nickname at card creation time at the *Terminal-Unit*, and the *Master-Unit* generates a HMAC (with a special key, the

nickkey) from this nickname. This HMAC is referred to as *user pseudonym* in this document. It is neither possible for the *Master-Unit* nor the *Terminal-Unit* to compute the user's nickname from this pseudonym. The *user pseudonym* is not stored in the *Master-Unit* neither in the *Terminal-Unit*, it is stored only in double encrypted form in the *AuthBlock* on the users card.

This pseudonym is used to apply modifications to a given account. A modification is done by adding an entry to the *FLMDB*. As this requires the *user pseudonym*, the nickname of the associated user must be known. Also the modifications can only be applied when the user processes the user authentication process.

4.3 Sharing a pseudonym

It is also possible to have multiple users sharing the same *user pseudonym*. Therefore they simply have to enter the same nickname. It is recommended to use the name of colors for such groups.

To apply modifications to an account in such a group, the modification has to be applied to all members of the group. An exception is the case where the card related to this account is available. In this case the *UID* from the card can be used to modify the flags in the *TicketDB* directly.

5 Usage

This section describes the AnonAccess system from the user's point of view.

5.1 Actions and commands

5.1.1 mainopen

Execute a special action (ex. open a door).

5.1.2 mainclose

Execute a special action (ex. closing/locking a door).

5.1.3 adduser

Add a user to the system. A user nickname must be specified. A user is added by generating a new valid *AuthBlock* which is written to an empty card, and by writing corresponding information to the *TicketDB*.

5.1.4 remuser

Remove a user from the system. A user nickname must be specified. If the nickname is stored in the *TicketDB* the entry in the *TicketDB* is immediately deleted which includes setting the *exists*-flag to 0. If the nickname is not stored in *TicketDB* a new entry in *FLMDB* is generated which leads to removal of the account when a *AuthBlock* is processed whichs *user pseudonym* matches the generated *user pseudonym*.

Table 1: example for minimum permission levels for different tasks

action	requirements
mainopen	1 user
mainclose	1 user
adduser	1 admin
remuser	1 admin
lockuser	1 admin
unlockuser	1 admin
addadmin	2 admins
remadmin	2 admins
keymigrate	3 admins

5.1.5 lockuser

Same as removing a user but instead of deleting the entry only the lock bit is set, which will cause the system to not accept the card as valid user card.

5.1.6 unlockuser

Same as removing a user, but instead of deleting the entry, an eventually set lock bit will be cleared.

5.1.7 addadmin

Same as removing a user, but instead of deleting the entry, the admin bit will be set, granting admin privileges to the user.

5.1.8 remadmin

Same as removing a user, but instead of deleting the entry, an eventually set admin bit will be cleared, so the user will not have admin privileges any more.

5.1.9 keymigrate

Initiate a key-migration, which will write the internal secret keys to the external serial EEPROM. This might not be implemented for security reasons.

5.2 Privileges

The system differentiates between "normal" (non-admin) users and admin users. To execute a given task in a session, special authorisation requirements must be met. These requirements are given as the number of users and admins which have to participate in the session. It might be decided to restrict admin privileges to users which are known by nickname. The given example of minimum permission levels assumes that admin privileges are restricted to users that are known by nickname.

6 Ideal run

1. User inserts card in *Terminal-Unit*

2. *Terminal-Unit* reads *AuthBlock* from card and transmits it in *addAuth*-Packet to *Master-Unit*
3. *Master-Unit* checks *UID* to be in range
4. *Master-Unit* checks *ticket* against the HMAC in *TicketDB* at *UID*
5. *Master-Unit* loads *userflags* from *TicketDB*
6. *Master-Unit* decrypts *ticket* and checks *timestamp* to be in range
7. *Master-Unit* decrypts r_{ID} ($dec_{pseudokey}(dec_{r_{key}}(r_{ID}))$) to get users pseudonym
8. *Master-Unit* searches in *FLMDB* for entries matching users pseudonym; for every matching entry it does:
   (a) modify users flags as indicated by the *setflags* and *clearflags* fields
   (b) delete the entry if the *permanent*-flag is not set
9. *Master-Unit* deletes *TicketDB*-entry
10. *Master-Unit* generates a new *UID* which points to an entry in *TicketDB*
11. *Master-Unit* generates a new *ticket* with a new *timestamp*
12. *Master-Unit* writes new *ticket* at *UID* in *TicketDB*
13. *Master-Unit* generates new r_{key}
14. *Master-Unit* generates new r_{ID}= $enc_{rid\_key}(enc_{r_{key}}(userspseudonym))$
15. *Master-Unit* transmits new *AuthBlock* in *addAuthAck*-Packet to *Terminal-Unit*
16. *Terminal-Unit* writes new *AuthBlock* onto card

7 Attacks and trusted components

This section tries to give an overview of the trust level of components and thereby an overview of the trust level of a complete implementation of AnonAccess.

7.1 Security goals

- access should only be granted to users who have a valid card whichs information and related information in the database state, that access should be granted to this user.
- no valuable information should be retrievable from the card's contents
- no valuable information should be retrievable by an unauthorised user from the AnonAccess system
- no information about the presence of a user who is not known by nickname should be available, even to an user with admin privileges

7.2 Trusted components

We consider a component to be a trusted component if the compromisation of this component leads to compromisation of at least one of the former declared security goals.

7.2.1 Terminal-Unit

The *Terminal-Unit* is considered trusted, especially the connection between the microcontroller and the card must be protected.

7.2.2 Master-Unit

The *Master-Unit* is considered trusted, especially the serial bus between the microcontroller and the external serial EEPROM must be protected. Although the external EEPROM's content is encrypted, an attacker might gather usefull information from the addresses which are accessed.

A The PRNG

The PRNG utilises SHA-256 as hash function. The entropy pool is 64 bytes (512 bits) large, which is the block size of SHA-256. We specify two algorithms which implement the functionality of the PRNG, one to add entropy to the entropy pool and one to get a block (32 bytes) of random data.

Algorithm 1 Add some data to the entropy pool

Require: $pool = pool_0 \parallel pool_1$ where $pool_0$ and $pool_1$ are both 32 bytes large
Require: $data$ of arbitrary length
Require: $offset$ which may be 0 or 1
$temp \leftarrow H(pool \parallel data)$
$pool_{offset} \leftarrow pool_{offset} \oplus temp$
$offset \leftarrow offset \oplus 1$

Algorithm 2 Get a block of random data from the entropy pool

Require: $pool = pool_0 \parallel pool_1$ where $pool_0$ and $pool_1$ are both 32 bytes large
Require: $offset$ which may be 0 or 1
$temp \leftarrow H(pool)$
$pool_{offset} \leftarrow pool_{offset} \oplus temp$
$offset \leftarrow offset \oplus 1$
$temp[temp[0] \wedge 31] \leftarrow temp[temp[0] \wedge 31] + 1$
$OUTPUT \leftarrow H(temp)$

B the Shabea-Cipher

Shabea (SHA based encryption algorithm) is a SHA-256 based Feistel-Cipher. It was designed to securely encrypt data where a SHA-256 implementation is available. It was important to have a small (in program space and memory

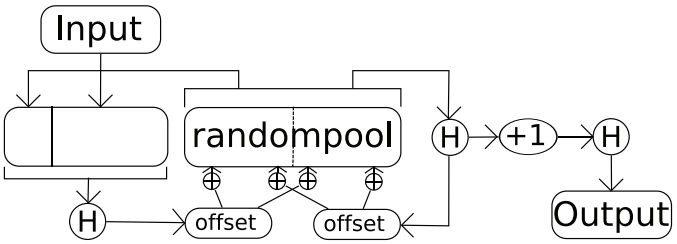


Figure 1: schematic of the PRNG

requirement) and nevertheless secure symmetric cipher, in the case that a SHA-256 implementation is available.

Algorithm 3 Encryption with Shabea

Require: $INPUT = L_0 \parallel R_0$ where L_0 and R_0 are both 16 bytes large
Require: $4 \leq rounds \leq 255$
Require: key which length (in bits) is $keylength$ of any size
  for $i = 0$ to $rounds$ **do**
    $L_{i+1} \leftarrow R_i$
    $R_{i+1} \leftarrow L_i \oplus H(key \parallel 0 \parallel i \parallel R_i)$
  end for
  $OUTPUT = L_{i+1} \parallel R_{i+1}$

Algorithm 4 Decryption with Shabea

Require: $INPUT = L_{rounds} \parallel R_{rounds}$ where L_{rounds} and R_{rounds} are both 16 bytes large
Require: $4 \leq rounds \leq 255$
Require: key which length (in bits) is $keylength$ of any size
  for $i = rounds + 1$ downto 1 **do**
    $R_{i-1} \leftarrow L_i$
    $L_{i-1} \leftarrow R_i \oplus H(key \parallel 0 \parallel i \parallel L_i)$
  end for
  $OUTPUT = L_0 \parallel R_0$

References

[1] When is a kilobyte a kibibyte? And an MB an MiB? (http://www.iec.ch/zone/si/si_bytes.htm)

[2] FIPS 180-2: Secure Hash Standard (SHS) (http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf)

[3] RFC 2104: HMAC: Keyed-Hashing for Message Authentication

[4] The I^2C-Bus Specification, Version 2.1, January 2000, original specification from NXP Semiconductors (http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf)

[5] ISO/IEC 7816-1:1998 Identification cards – Integrated circuit(s) cards with contacts – Part 1: Physical characteristics

[6] ISO/IEC 7816-2:1999 Identification cards – Integrated circuit cards – Part 2: Cards with contacts – Dimensions and location of the contacts

[7] ITU-T Rec. X.690: Information technology ? Abstract Syntax Notation One (ASN.1): Specification of basic notation (http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf)

[8] 24AA512/24LC512/24FC512 1024K I^2C CMOS Serial EEPROM, datasheet by Microchip (http://ww1.microchip.com/downloads/en/DeviceDoc/21754H.pdf)

[9] 24AA1025/24LC1025/24FC1025 1024K I^2C CMOS Serial EEPROM, datasheet by Microchip (http://ww1.microchip.com/downloads/en/DeviceDoc/21941E.pdf)

[10] The Microchip Cooperation web presence (http://www.microchip.com)

[11] QPort-tiny specification, Daniel Otte (http://nerilex.3dots.de/qport-tiny.pdf).

[12] Tea extensions, Roger M. Needham and David J. Wheeler, (Notes October 1996, Revised March 1997, Corrected October 1997) (http://www.cix.co.uk/~klockstone/xtea.pdf)

[13] The Atmel Cooperation web presence (http://www.atmel.com)

[14] ATmega644 Preliminary (revision M, updated 08/07) (http://www.atmel.com/dyn/resources/prod_documents/doc2593.pdf)

[15] ATmega32(L) (revision K, updated 08/07) (http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf)

[16] GCC, the GNU Compiler Collection (http://gcc.gnu.org)

[17] AVR Libc Home Page (http://www.nongnu.org/avr-libc/)

Science
lecture
2007-12-30 14:00
Saal 3
en

Immanuel Scholz

Dining Cryptographers, The Protocol

Even slower than Tor and JAP together!

Imi gives an introduction into the idea behind DC networks, how and why they work. With demonstration!

Back in 1988, David Chaum proposed a protocol for perfect untracable communication. And it was completly different to the (former invented) Mix Cascades. While the Mixes got all the press (heard of "Tor" and "JAP"? Told you!), the idea of DC networks were silently ignored by the majority of the community.This talk is to show how DC networks work, why they are secure and presents an implementation.

http://www.eigenheimstrasse.de/imi/dc	DC Network Client (Java WebStart)
http://www.eigenheimstrasse.de/svn/dc/	Source Code to the DC Network Client
http://www.eigenheimstrasse.de/svn/dc/doc/dcnetwork.pdf	Slides

Dining Cryptographers – The Protocol

Immanuel Scholz

http://www.eigenheimstrasse.de/svn/dc

1 Introduction

What are dining cryptographer networks? How and why do they work? And why is there still no real implementation available?

Back in 1988, David Chaum wrote an article for the first issue of "Journal of Cryptology" about a method to provide sender and receiver anonymity in a closed group network [2]. He called it "The Dining Cryptographers Problem", after his introductory example.

In his example, three cryptographers meet for dinner, which has paid paid beforehand. They are curious, whether one of them has paid the dinner or whether it was sponsored by the government. So they came up with the DC-protocol.

1.1 Original Protocol

Each cryptographer *exchanges a secret key* with both other cryptographers by flipping a coin under cover. Then, all three *announce the sum* of the coin flips[1] except if one really paid for the dinner. Then he *"adds" this message* by stating the inverse of the coin flip. After all cryptographers announced their sums, the *sums are summed up* again. If the final number is 0, nobody said that he paid, so the dinner must have been paid by the government.

To summarize, the following is necessary to set up a DC-network:

1. Exchange symmetric keys with other participants. As example the key between Alice and Bob is k_{ab}, between Alice and Charlie k_{ac} and between Bob and Charlie k_{bc}. One of the participants gets the inverse element of the key, e.g. when Alice gets k_{ab}, Bob remembers $-k_{ab}$.

2. Participants sum up their keys and add their message to the sum. So Alice calculates $k_{ab} + k_{ac} + m_{alice}$, Bob $-k_{ab} + k_{bc} + m_{bob}$ and Charlie $-k_{ac} - k_{bc} + m_{charlie}$. Only one of the participants should send a message at the same time.[2]

3. All sums are summed up again. The final result is the message of the one who had something to say or 0, if nobody said anything.

$$k_{ab} + k_{ac} + m_{alice} - k_{ab} + k_{bc} + m_{bob} - k_{ac} - k_{bc} + m_{charlie} = m_{alice} + m_{bob} + m_{charlie}$$

Although the network can run on a coin-flip-base, using another mathematical group is more appropriate. The formula works for all abelian groups.

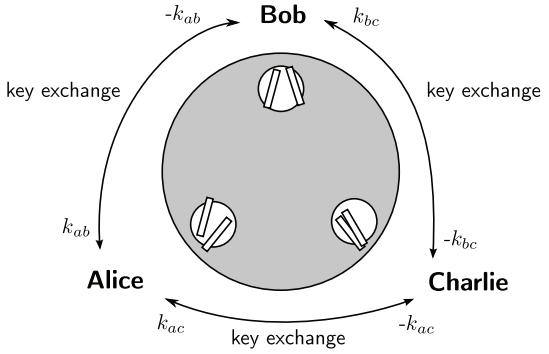


Figure 1: Alice, Bob and Charlie forming a DC – network

[1] David Chaum talked about stating "whether the two coins he can see...fell on the same side or on different sides". Mathematically, this can be expressed as binary addition, where H is head, T is tail, $H+H=T$, $H+T=H$, $T+T=T$, H ="different sites" and T ="same site".

[2] There exists a collision resolving algorithm with no data loss, but this is beyond the scope of this paper. [4, page 188–190]

Also, there may be more than three participants. It is not required that every participant exchanges keys with every other, but it does no harm to do so either.

1.2 Security

So why is it secure? Let's start with a definition of the attacker model.

For every message, every participant in an anonymity network can be in the role of the receiver, sender, none or both. The security in this network is broken, if an attacker learns something about a participant's role from the network. The attacker may observe communication lines[3] and may collude with a number of participants. The strongest possible attacker in this definition is observing *all* lines and collude with all but two participants. Naturally, if only one participant is not cooperating, his anonymity is broken, because he sends / receives if nobody else does. "Colluding" means, the attacker knows all the exchanged keys and the individual messages of the participant.

Providing receiver anonymity is easy. Every message is broadcast to every participant. For sender anonymity, everyone exchanges random keys with everyone else and add them to the value he sends over the wire. So in our example, the strongest attacker knows all the three values $k_{ab}+k_{ac}+m_{alice}$, $-k_{ab}+k_{bc}+m_{bob}$ and $-k_{ac}-k_{bc}+m_{charlie}$ as well as the keys of one participant, e.g. Bob: k_{ab}, k_{bc} and m_{bob}. Even then, he cannot learn m_{alice} or $m_{charlie}$. So he cannot know who of both sent the message.

For example, Bob adds the value observed from Charlie to his key exchanged with Charlie: $-k_{ac}-k_{bc}+m_{charlie}+k_{bc}=-k_{ac}+m_{charlie}$. As $-k_{ac}$ is a random value not known to Bob, he does not learn anything about $m_{charlie}$. $-k_{ac}$ acts as a key for an one-time-pad to "hide" the value of the message. This holds true for any formulas Bob calculates and is proved in [2].

1.3 Performance

Maybe the single most important reason, why there is no widespread DC–network implementation, is the amount of overhead it requires in most scenarios, especially for standard bilateral communications like web surfing. To achieve sender and receiver anonymity, every possible sender and receiver has to transfer a whole block each time, whether he wants to send something or not. This means, for w people in the network, a single byte requires a minimum of $2w$ bytes transferred. As the grade of anonymity is directly related to w, it should be as large as possible. And even if nobody in the network sends a message, all participants have to transfer their message block.

Finally, each byte requires each participant to send and receive one byte, which does not play well together with asymmetric connection speeds found in some countries.[4]

To reduce the amount of overhead, two things should be implemented. First, an anonymous reservation scheme helps to transfer data only, when really someone has something to say. Second, an application that needs message broadcasting is highly preferable, as broadcasting is done by the DC-network anyway. This reduces the overhead to $2w$ bytes transferred for every w bytes information.

2 Time

So far, we thought about one message. To transfer more messages, time is separated by *ticks* into *rounds*. Each participant has to send exactly once each round. A tick occurs, if the last participant sent his message for the current round or after some timeout expired, in which case all pending clients are considered broken and removed from the network. Each round, each client has to do the following things:

- Get the list of all participants and their keys
- Prepare and send the new message
- Get the message after all clients have sent
- Know that all other have received the message

The last item might be a surprise, but the following section explains the reason behind it.

[3] As example by controlling routers between the participant's computers.

[4] In Germany, ADSL connections have several MBit per second download speed but usually no more than one MBit per second upload speed.

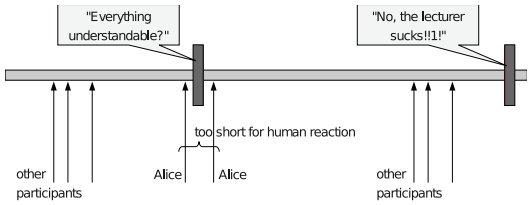


Figure 2: Timing Attack

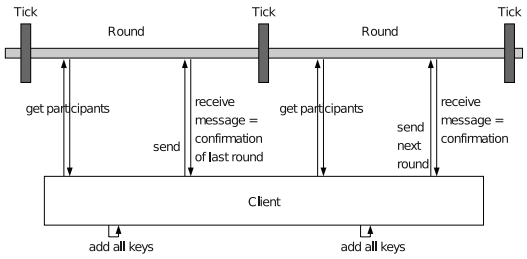


Figure 3: The protocol

2.1 Linkability

When considering a stream of messages, another privacy aspect is introduced: *linkability*. An attacker should not be able to link the role a participant has for different messages. At least, the used technical system should not give him any help in linking. In a chat, it is unlikely that a person is answering its own questions. However, the system should not provide any additional evidence about the origin of the answer, so it should be "as unlikely as without the technical system" to answer a question posted by himself.

As a key requirement, the network must ensure that every participant receives every message (receiver anonymity). Else, e. g. an attacker on a chat application could prevent a specific member from getting the question and look whether the answer is still posted.

Also, all participants have to be treated equally. In every round, everybody can try to send and even try to send a collision with himself. [4, page 187]

2.2 Timing Attack

Figure 2 shows a common attack, using the ability of the attacker to link two messages together. The attacker observes that Alice send her messages just before and just after the new round. Also, the attacker recognizes that the second messages is an answer to the first, so it is very likely that Alice did not send the second message, as she did not have time to read, type and send the answer.

One way to decrease the possibility of a successful timing attack: The server can delay the publication of the message, e. g. for one round. An easy way of doing this is returning the message of the last round within the connection, where the client sends the current message.

2.3 The Protocol

Figure 3 shows an overview of the protocol used in the implementation. The four necessary items in the list above are split to two adjacent rounds, where the return value of the send operation returns the message together with the confirmation. In this scheme, the server could lie about the distribution of the message. A modification of the DC-network protocol [4, page 184–185] has been implemented, so that the key of the new round depends on the message of the last round. This ensures that all participants got the correct message, even in scenarios with untrusted servers.

3 Excluding bad clients

To improve the availability, a fast way of detecting and excluding broken or malicious clients is required. Clients who do not send are excluded by the server at the next tick. After a broken round, all clients do a re-keying by requesting the participants' keys (see section 4) or sending their key to the server again.

Detection of malicious clients is a bit more complex and are beyond the scope of this paper. In short, an anonymous reservation scheme is deployed as described in [2]. Together with trap messages, malicious clients trying to disturb the network can be detected without restrictions on the anonymity. Other social precautions like "joining by invitation" or reputation systems could be deployed as well.

4 Key Exchange

Keys are exchanged using the Diffie–Hellman key agreement protocol [3]. The server publishes a prime p and generator g. Alice generates x_{alice} mod p at random. She calculates $g^{x_{alice}} \bmod p$ and sends this as her "key" to the server. Bob does the same. To generate a secret key between the two, both retrieve the key of the other and Alice calculates $(g^{x_{bob}})^{x_{alice}} = g^{x_{alice}x_{bob}}$, while Bob calculates $(g^{x_{alice}})^{x_{bob}} = g^{x_{alice}x_{bob}}$. The published keys have to be signed by an official signing key. The current implementation uses GnuPG for this purpose [1].

One advantage of using Diffie–Hellman is that no direct client to client communication is needed. Also, everyone exchanges keys with everyone, providing maximum security against attackers controlling even most of the other participants.

Diffie–Hellman is practical for rather short secrets only, typically 1024 bits. So the key is the seed to a pseudo random number generator. Unfortunately, the random sequence becomes provable, which means that the round key can be verified by publishing the seed. As example, if the round message is "Bob sucks", and Bob wants to know whether Alice or Charlie said this, he forces Charlie to publish his seed. Bob can verify that Charlie did not lie about the seed, as either the empty message or the insult shows up when Bob does the number generation with Charlie's seed.

Fortunately, Alice and Charlie could have exchanged an *additional* key or even a true random key for a true one-time-pad. So Charlie can make excuses when asked for the seed. See section 5.1 for more about the additional key.

5 On-demand disclosure

David Chaum proposed a scheme to disclose the origin of a message [2, chapter 2.6]. He suggested that every participant signs the symmetric key exchanged using a digital signature system and gives the signature to the other participant. E. g. the key k_{ab} exchanged between Alice and Bob is signed by both and Alice gets $sign_{bob}(k_{ab})$ while Bob gets $sign_{alice}(k_{ab})$. In case of "disclosure", all participants publish all exchanged keys and Alice can prove that Bob did not lie while Bob proves Alice's key testimony. We propose another scheme

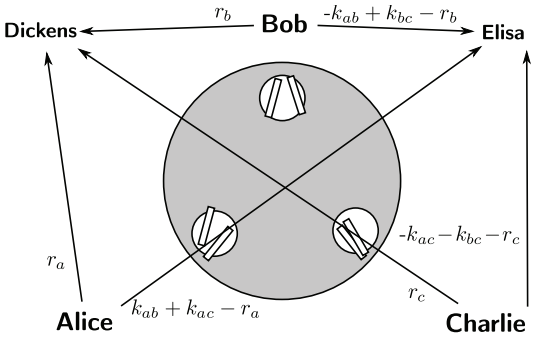


Figure 4: Watchmen on-demand disclosure scheme

which we call "watchmen disclosure scheme".

There exist n predefined watchmen, who may work together to disclose the sender of a specific message. Each client has to split his round key sum into n parts and send one part to each watchman. The split is done by generating $n-1$ random numbers r_1 to r_{n-1}, which are send to the first $n-1$ watchmen. The last watchman gets $k - \sum r_i$. Before a round message is published, all watchmen add their parts from all participants together and publish their sums. The sum of all sums must be 0. This is used to verify that no single participant lies about his round key. In case of "disclosure", the watchmen release all individual parts and so the round keys of all participants can be reconstructed. Figure 4 shows the key parts sent for two watchmen Dickens and Elisa.

5.1 Conspiracy

In a mood of plotting, Alice and Charlie exchange an additional key l_{ab}. They agree to use the key only on their round keys but not on the key parts send to the watchmen. In normal operation, the watchmen do not recognize the additional key – the sum of their sums is still 0. But if a disclosure is required, the keys from Alice *and* Charlie differ from the reconstructed round keys. If one of Alice or Charlie sent the message, nobody knows, who of the both actually sent it. Of course, both could be considered "bad guys".

5.2 Threshold secret sharing scheme

The watchmen scheme can be modified, to decrease the share of watchman required to disclose a message, as example in scenarios where only 5 out of 10 watchmen have to agree to track the messages origin. Adi Shamir proposed the threshold secret sharing scheme based on polynomials [5]. This can be applied if the following restrictions are made:

- All participants must use same set of watchmen, which may be necessary anyway
- The x-coordinates of the threshold scheme is fixed for each watchman

5.3 Comparison between DC-disclosure and Watchmen

So what are the practical differences between the original proposal based on asymmetric signed round keys and the watchmen scheme?

Unfortunately[5], both schemes are insecure against conspirations. However, in the watchmen scheme, a conspiration cannot be formed *after* the message was published, as the additional key has to be added before the parts are sent to the watchmen. In the original scheme, e. g. the participants could agree to cover each other depending on the message content.

For the process of disclosing a sender, the watchmen scheme does not require to contact the participants. Only the watchmen have to be contacted. This also means, the act of disclosing a sender can be kept secret.[6]

6 Serverless DC-network

What do we need a server for? The only things the server does is coordinating the clients, providing web-space for the messages and keys and finally adding some message parts. The coordination could be done within the network itself, web-space is available everywhere today and the adding can be done by each client on its own.

Having said that, in some countries server maintainer must gather information that can trace down communication partners (data retention). Without any server, there is no such information to store. So the protocol proposed in section 1.1 can be modified:

First every participant needs some place to publish any arbitrary information, e. g. a personal blog, a web-forum, a MySpace[7] account or something like that. Each participant publishes his round sums on this account.

Log-in can be done by invitation, where someone already in the network publishes the URL and Diffie–Hellman key of the new participant in the network. There could be websites doing this as substitute, if open membership is desired.

Log-out is done by sending a goodbye message (or automatically by failing to publish a round).

References

[1] Gnu privacy guard. Available from World Wide Web: `http://www.gnupg.org`.

[2] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[3] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976. Available from World Wide Web: `http://citeseer.ist.psu.edu/diffie76new.html`.

[4] Andreas Pfitzmann. Sicherheit in rechnernetzen, 2000. Available from World Wide Web: `http://dud.inf.tu-dresden.de/~pfitza/DSuKrypt.pdf`.

[5] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979. Available from World Wide Web: `http://portal.acm.org/citation.cfm?id=359176`.

[5] Or fortunately – depending on the point of view.

[6] Again, whether this is an advantage or not depends on the point of view.

[7] Provider of free web-space. `http://www.myspace.com`

Hacking
lecture
2007-12-29 11:30
Saal 3
de

Tonnerre Lombard

Grundlagen der sicheren Programmierung

Typische Sicherheitslücken

Dieser Vortrag bietet eine Übersicht über einige Dinge, welche man im Kopf behalten sollte, wenn man Software schreibt - vorausgesetzt, diese soll nachher nur von der Person benutzt werden, die sie auch betreibt. Die theoretischen Aspekte der Sicherheit werden mit Codebeispielen untermalt.

In der Programmierung gilt Sicherheit oft als ein von Schamanen betriebenes und mit Zauberkraft gesichertes Geheimnis. Viele Leute predigen verschiedene Wege, sicheren Code zu schreiben. Die meisten dieser Wege laufen auf die Verwendung bestimmter Programmiersprachen hinaus.Im Laufe des Vortrages wird allerdings gezeigt, dass nur Sachkenntnis über die potentiell auftauchenden Probleme der Schlüssel zu einem sicheren Programm ist. Dabei richtet sich der Vortrag hauptsächlich an Leute, die sich nicht in ihrem alltäglichen Leben mit dem Finden von Sicherheitslücken in Software beschäftigen.

Sicherheitsprobleme in der Programmierung

Tonnerre Lombard

18. Oktober 2007

1 Mythen der Sicheren Programmiersprache

In der Programmierung gilt Sicherheit oft als ein von Schamanen betriebenes und mit Zauberkraft gesichertes Geheimnis. Viele Leute predigen verschiedene Wege, sicheren Code zu schreiben. Die meisten dieser Wege laufen auf die verwendung bestimmter Programmiersprachen hinaus. Im Zweifelsfall laufen die Argumentationen jedoch in's Leere. Einige dieser leeren Versprechungen werden im ersten Teil genauer beleuchtet und im Laufe des Textes widerlegt. Dies umfasst die Verwendung von Skriptsprachen, alternativen Bytecodes sowie Hoch- und Niedersprachen.

2 Arten von möglichen Fehlern

Wie nicht anders zu erwarten, gibt es in der komplexen Welt der Programmierung viele verschiedene Dinge, welche man falsch machen kann.

2.1 Buffer Overflow

Ein Buffer Overflow ist eine sehr grundlegende Art von Fehlern, welche aus der Art und Weise resultiert, wie die Daten ausgeführter Programme im Speicher angeordnet werden. Es gibt dabei praktisch zwei verschiedene Arten von Buffer Overflows: Stack Overflows und Heap Overflows. Beiden ist gemeinsam, dass über den vorgesehenen Speicherbereich hinaus geschrieben werden kann, wodurch zur Programmausführung wichtige Daten manipuliert werden. Auf diese Art kann die Ausführung beliebigen Codes erzwungen werden.

2.2 Synchronisierungsprobleme

Wann immer Code parallel ausgeführt wird, welcher auf dieselben Dinge zugreift, kann es zu Problemen kommen. Dies fängt beim Sperren von geöffneten Dateien an und geht über den parallelen Zugriff auf Daten zwischen Threads

bis hin zur Signalbehandlung. Wann immer der Programmablauf keinen roten Faden darstellt, ist eine Form von Synchronisierung vonnöten.

2.2.1 Fehlende Parallelisierung bei geteilten Zugriffen

Greifen mehrere Prozesse auf dieselbe Ressource zu, können unter Umständen verschiedene sicherheitskritische Situationen entstehen, welche durch Angreifer ausnutzbar sein könnten. Voraussetzung dazu ist lediglich fehlende Synchronisierung der Prozesse.

Ebenfalls in diese Kategorie fallen Angriffe, bei denen ein Prozess Objekte mit den falschen Berechtigungen erstellt und diese nachträglich ändert – und somit einen Zeitraum schafft, während dem sich andere prozesse Rechte an dem Objekt sichern können.

2.2.2 Fehlende Threadsynchronisation

In der Synchronisation zwischen Threads ist das Potential für Probleme noch viel grösser, da sie nicht über getrennte Speicherbereiche verfügen. Die Verwendung reentranter Funktionen spielt hier eine grosse Rolle.

2.2.3 Signalbehandlungsangriffe

Eine weitere, oft unterschätzte Form asynchroner Programmausführung sind Signale, und auch diese können unter Umständen zur Codeausführung verwendet werden.

2.3 Formatstringangriffe

Mit Formatstringangriffen kann in den meisten Fällen erst einmal nur Speicher gelesen werden, aber auch dieser kann bereits interessante Informationen enthalten.

2.4 Injectionangriffe

Wann immer mehrere Sprachen ineinander eingebettet werden, ist es ratsam, dafür zu sorgen, dass Elemente der inneren Sprache nicht mit Elementen der äusseren Sprache gemischt werden. Dieses Problem ergibt sich auch und vor Allem bei benutzerkontrollierten Eingaben in Applikationen, welche in der Ausgabe der Applikation oder in erzeugten Befehlen repräsentiert werden.

2.4.1 Formatinjektion

Formatinjektionen sind die älteste Art von Injection-Sicherheitslücken. Hierbei werden die Begrenzungszeichen eines Formates in einem eingefügten, nicht ge-

prüften Teil verwendet, so dass zusätzliche Daten eingefügt werden. In einem Beispiel wird ein Rootaccount angelegt, wobei der Anlegende lediglich über Benutzerrechte verfügt.

2.4.2 Cross Site Scripting (XSS)

Cross Site Scripting ist ebenfalls das Einbetten von Informationen in eine Sprache in die sie nicht hinein gehören, um JavaScript-Elemente auf Seiten einzublenden, auf die sie nicht gehören, um Kontrolle über die Inhalte zu erlangen.

2.4.3 SQL injection

In diesem Teil wird die Natur der SQL-Injection-Sicherheitslücken erläutert, inklusive Codebeispielen wie eine solche Sicherheitslücke zustande kommt.

2.5 Authentisierungs- und Verifikationsmängel

Eine ganz eigene Klasse von Fehlern liegt in der Logik der Applikation versteckt. Oft werden hier Sicherheitsmerkmale vergessen oder nicht vollständig ausgeführt, oder sie werden aus unsicheren Elementen zusammengesetzt.

2.5.1 Berechtigungsprobleme auf Objekte

Probleme mit den Berechtigungen auf Objekte, welche von mehreren Prozessen gesehen werden können, sind immer wieder eine grosse Fehlerquelle – vor Allem, da zum Beispiel die Benutzerrechte auf Dateien oft nicht nur vom entsprechenden Programm verwaltet werden. Was hierbei zu beachten ist und wie man mit renitenten Benutzern umgeht, wird in diesem Kapitel erläutert.

2.5.2 Unauthentisierte Interfaces

In einigen wenigen Fällen besteht das Sicherheitsproblem darin, dass die Authentisierung oder Autorisierung für ein Interface nicht geprüft wird. Dieser Teil erwähnt den Fall allerdings bloss, da er mehr oder weniger selbsterklärend sein sollte.

2.5.3 Sessiondiebstahl

Eine einfache Möglichkeit, an den Account einer anderen Person zu kommen, sei es um Daten auszuspähen, die Person zu personifizieren, oder um deren Berechtigungen zu missbrauchen, sind oft laufende Sitzungen der Person ein Angriffsziel. Mit Codebeispielen wird darauf eingegangen, auf welchen Wegen man eine Sitzung einer anderen Person übernehmen kann.

Mittels SQL-Injection

Es gibt mehrere Methoden, SQL-Injection auszunutzen, um Zugriff auf fremde Accounts zu erhalten. Ein paar Beispiele werden im Code dargestellt.

Mittels XSS

Hierbei wird darauf eingegangen, wie man mittels Cross Site Scripting das Session-Cookie einer Webseite entwenden kann.

Bei schlechtem Generator

Einige Fälle von Sessiondiebstahl sind auch einfach auf schlecht generierte Cookies zurückzuführen. Es wird beleuchtet, welche Methoden zur Generierung von Session-Cookies als sicher angenommen werden können und welche gar nicht in Frage kommen.

2.5.4 Cross Site Request Forgery (CSRF)

Der modernste unter den modernen Angriffen nennt sich Cross Site Request Forgery. Hierbei wird eine Aktion durch einen bereits angemeldeten Benutzer von einer anderen Seite aus ausgelöst.

3 Spezielle Probleme mit 32-Bit-Code

Dieser letzte Teil des Vortrages behandelt einige Probleme, die nur speziell dann auftreten, wenn Code auf 64-Bit-Prozessoren ausgeführt wird, welcher 32-Bit-Spezifika aufweist.

4 Abschliessende Hinweise

Zuletzt werden noch einige Hinweise zur Architektur sicherer Systeme gegeben. Dies reicht von erneuter Mahnung zum Prüfen gegen Buffer Overflows bis zum Hinweis, wie SSL-Clientzertifikate die nervigen Cookieprobleme ein für alle mal beseitigt werden können.

Society
lecture
2007-12-29 12:45
Saal 1
en

Tomislav Medak
Toni Prug
Marcell Mars

Hacking ideologies, part 2: Open Source, a capitalist movement

Free Software, Free Drugs and an ethics of death

The Open Source initiative re-interpreted Free Software to include it into the neo-liberal ideology and the capitalist economy - whose aims are contrary to the FS starting axioms/freedoms. This platform will focus on ideological and political aspects of this. It will also suggest FS recovery strategies.

Believe. "The World is Yours." (Ian Brown, 2007)
What is Re-interpretation of FS by Open Source ?In The Revenge of the Hackers, Eric Raymond talks about Open Sourcegoals in clear terms: "In conventional marketing terms, our job wasto re-brand the product, and build its reputation into one the corporate world would hasten to buy."
The move of the Open Source initiative to bring Free Softwarecloser to capitalism shows that:
a) there is a gap between the Free Software movement and capitalism;
b) without a significant institutional intervention and re-interpretation that gap can not be overcome;
c) it is the founding documents (practice of Open Source doesn't differ), ethics that Richard Stallman stands by so fiercely, that are the bite that capitalism can not subsume, swallow in its

http://publication.nodel.org/The-Mirrors-Gonna-Steal-Your-Soul The Mirror's Gonna Steal Your Soul
http://rabelais.socialtools.net/FreeSoftware.ToniPrug.Aug2007.pdf Free Software

Hacking ideologies, part 2: Open Source, a capitalist movement

Free Software, Free Drugs and an ethics of death

(by Toni Prug, toni@irational.org)

Written as a 24c3 event proposal. Based on an unpublished dissertation available at http://rabelais.socialtools.net/FreeSoftware.ToniPrug.Aug2007.pdf

---

Believe. "The World is Yours." (Ian Brown, 2007)

The Open Source initiative re-interpreted Free Software to include it into the neo-liberal ideology and the capitalist economy - whose aims are contrary to the FS starting axioms/freedoms. This platform will focus on ideological and political aspects of this. It will also suggest FS recovery strategies.

======

What is Re-interpretation of FS by Open Source ?

In The Revenge of the Hackers, Eric Raymond talks about Open Source goals in clear terms:

"Our success after Netscape would depend on replacing the negative FSF stereotypes with positive stereotypes of our own--pragmatic tales, sweet to managers' and investors' ears, of higher reliability and lower cost and better features. In conventional marketing terms, our job was to re-brand the product, and build its reputation into one the corporate world would hasten to buy."

======

The move of the Open Source initiative to bring Free Software closer to capitalism shows that:

a) **there is a gap between the Free Software movement and capitalism;**

b) without a significant institutional intervention and re-interpretation that gap can not be overcome;

c) it is the founding documents (practice of Open Source doesn't differ), ethics that Richard Stallman stands by so fiercely, that are the bite that capitalism can not subsume, swallow in its original form.

======

O'Reilly role: conferences, books, lobbying.

Here's some neo-liberal text-book propaganda:

"standardization, and thus commodification, are both natural market forces as well as key events in human history".

Ian Murdoch, founder of Debian
in "Open Source and the
Commoditization of Software" (O'Reilly)

======

Multitude misunderstood - time for education

Lack of understanding of the difference between Open Source and Free Software is best seen when in one of the masterpieces of recent social theory (Hardt/Negri's "Multitude") term "open-source" was referenced with the "Free as in Freedom" book on Stallman.

======

Freedom is Politics

Badiou/Zizek: politics is not parliaments, debates compromises, voting. On the contrary, it is subjectively, militantly, unilaterally, deciding what seems impossible at the time of the decision.

Actually existing freedom is to choose outside of the given coordinates in which choice takes place. It is to re-configure the conditions, to change the scope of "possible outcomes".

This is the difference between Lenin's concept of freedom and the liberal, parliamentary, formal freedom, which consists in participating in the what is already given, already structured. (Zizek, "On Belief")

=======

Truth of RMS (Badiou)

Could we not say that this is precisely what Richard Stallman did with his choice of leaving the job he had at the MIT Lab to devote all his time to re-create the world of software, from scratch, with an entirely new set of co-ordinates?

Respect or disrespect the printer licence? Neither! Free Software instead.

Badiou: acting in follow up to an event -- event that prompts our reaction/decision -- and pursuing the truth of it through fidelity to it, through fidelity to the retroactively constructed event that changes us.

======

Paradox/Subject: axioms-openness, a missing link.

One might rightly assume that in the sphere of rational tasks knowledge and science would be the decisive spheres. Yet, some issues can not be settled that way.

No science is pure, cleansed of ideology.
Think maths. Axioms - Not to be questioned.
Does that make maths dogmatic? Of course not.
Sciences rely on axiomatic foundations.

ONE: Free Software relies on the set of principles called freedoms.
These are axioms. Not up for discussion.

TWO: Free Software communities function through open participation.
Its progress is through ongoing collaborative production and critique.

Stallman's truth (printer event + fidelity) stands in the sharp contrast, indeed in total opposition, to the attributes of the movement he founded.

Openness and collaboration flourished from closed starting points, from axioms. This is what i call The Paradox of a becoming Subject.

======

GNU manifesto applied: punishment for capitalists?
food, shelter, health, education, labour

from GNU manifesto:
"Don't programmers deserve a reward for their creativity? If anything deserves a reward, it is social contribution. Creativity can be a social contribution, but only in so far as society is free to use the results. If programmers deserve to be rewarded for creating innovative programs, by the same token they deserve to be punished if they restrict the use of these programs."

Consider applying this to the economy. What would it mean to assert that economic productivity can be a social contribution only if its results can be shared? It is already shared, many would say: one gets a salary for one's work. This would not satisfy FS criteria. If we narrow down the concept of economic productivity to food, shelter, health, education - conclusion could be that capitalists restrict use of the above elements by subverting them into closed, private wealth generation schemes. And hence, deserve punishment.

======

Inbuilt obsolescence - will they expire us one day, on the retirement day?

from GNU manifesto:
"Extracting money from users of a program by restricting their use of it is destructive because the restrictions reduce the amount and the ways that the program can be used. This reduces the amount of wealth that humanity derives from the program. When there is a deliberate choice to restrict, the harmful consequences are deliberate destruction."

Consider inbuilt obsolescence - a capitalist invention whereby products are designed to fail in order for the development of new products to be justified by demand created by the inbuilt timed failure of the old ones.

======

AIDS/malaria today: Parliamentary Capitalist Ethics of Death

First, why parliamentary?
Because, as a supreme body, parliaments have the power to stop this. Yet, they don't. The actively encourage and protect it.

Given today's drugs, AIDS could be contained worldwide in relatively short period of time, but corporations, governments (and the catholic church) stand in the way of dying millions being protected (Alain Badiou, "Century", 2007).

The production of drugs could follow the example of Free Software, be created in a more collaborative way, publishing recipes and allowing it to be freely produced, by anyone, for any purpose.

When ethics and its laws allow death on such scale to occur, although the society has the means to prevent it, we have to ask: what is the difference between tens of millions dead in two world wars and the dead of malaria and AIDS today? The former were killed while later are let to die - by the ethics of death.

=======

Patents, Copyright, Mass Death

Hacking needs access to what it hacks on, and it needs sharing to grow the knowledge of how it operates. Parliamentary Capitalist law aims to limit access. Such law is opposed to the ethics of hacking.

Sharing of drug recipes for the prevention of deadly epidemics is explicitly and deliberately forbidden by the parliamentary capitalist law.

No humans, let alone hackers, should support such law.
Patents and copyright belong to the same ethics.
One that allows millions of AIDS and malaria deaths, annually.

======

Open Source is a neo-liberal, parliamentary capitalist social movement.

Neo-liberalism claims they're ``just doing it'' for the sake of a better economy, without any ideological beliefs. As if any economy, or any act, was possible without decisions determined by a set of ideas and beliefs.

This is why Nike's slogan ``just do it''
is the best summary of the capitalist ideology ever.

And this is why ``*Open source is a development methodology; free software is a social movement*'' (Stallman), misses the crucial point.

Open Source is not just a development methodology, but a social movement too, a social movement of a different kind, with different, parliamentary capitalist, goals.

Another problem lies in the claims that Open Source separates ethics from the technical side of Free Software (Stallman, "Why 'Open Source' misses the point about Free Software"), thus making it acceptable to corporations.

This implies two wrong statements about Open Source:

a) *it has no ethics of its own;*

b) *there are purely technical solutions which can be used without any ethical, political, or ideological commitments.*

The result of these mistakes is widespread comparison of Free Software and Open Source on false, crucially misleading terms:

- one (FS) operating under the weight and demand of its ethics;

- the other (OS) getting away without being examined at all, basking in the purity of its technical attributes and various business-friendly tags

This is how the ethics, the ideology and, indeed, the politics of Open Source slip through unexamined and unchallenged -- like the capitalist ideologies whose key strategy has historically been to accuse any political opponents of ethical commitments, while insisting on their own ``pragmatism'' and on the purely technical aspect of ``just getting things done''.

======

Free Software - Free Drugs?

If we are to agree with Ranciere that democratic process is a process of subjects who ``**reconfigure the distributions of the public and the private''**, who challenge the privatization based on birth, wealth and 'competence', privatization guarded by the police and the State (p.61-2 "Hatred of Democracy", 2006) -- here's how a definition of Free Drugs, another possible process of reconfiguration of the public and the private, could be inherited from Free Software:

- The freedom to use the drug, for any purpose (**freedom 0**).

- The freedom to study how the drug works, and adapt it to your needs (**freedom 1**). Access to the drug recipe (blueprint) and acceptance through regulated clinical trials are preconditions for this.

- The freedom to redistribute copies of the drug and its recipe (blueprint) so you can help your neighbour (**freedom 2**).

- The freedom to improve the drug, and release your improvements to the public, so that the whole community benefits (**freedom 3**). Access to the drug recipe (blueprint) and acceptance through regulated clinical trials are preconditions for this.

======

Finally ... piracy? WHAT PIRACY?

Parliamentary capitalism (in its propaganda materials also known as liberal democracy) is based on the idea that majority should determine, via its representatives, how a state is governed and an economy run. Cisco tells us in their marketing materials that vast majority of the Internet traffic today is p2p traffic. If vast majority of users practice p2p and disregard law, should not then, by the capitalist parliamentary logic of majority rule, law be changed, since that is what majority wants?

Parliaments and representatives act to prevent us from having laws that will act on our behalf, regardless of our position of minority, or majority - the logic by which they operate is different, it is the logic of capital.

The hacks we desperately need are in the realm of thoughts - We need to change our thinking about the law and its relations to politics, while continuing expansion of p2p technologies and techniques.

If more hacking of ideas/thoughts was discussed and encouraged, we would soon render discussions on piracy/laws/illegal/legal obsolete. Instead, we would focus on what political and economic structures do we want, to protect whom, to encourage what? And how do we do it.

=======

Believe. "The World is Yours." (Ian Brown, 2007)

Hacking ideologies, part 2: Open Source, a capitalist movement

(by Toni Prug, toni@irational.org)
based on an essay Free Software

Hacking
lecture
2007-12-28 21:45
Saal 2
en

lucy

Inside the Mac OS X Kernel

Debunking Mac OS Myths

Many buzzwords are associated with Mac OS X: Mach kernel, microkernel, FreeBSD kernel, C++, 64 bit, UNIX... and while all of these apply in some way, "XNU", the Mac OS X kernel is neither Mach, nor FreeBSD-based, it's not a microkernel, it's not written in C++ and it's not 64 bit - but it is UNIX... but just since recently.

This talk intends to clear up the confusion by presenting details of the Mac OS X kernel architecture, its components Mach, BSD and I/O-Kit, what's so different and special about this design, and what the special strengths of it are.
The talk first illustrates the history behind BSD and Mach, how NEXT combined these technologies in the 1980s, and how Apple extended them in the late 1990 after buying NEXT. It then goes through the parts of the kernel: Mach, which does the typical kernel work like memory management, scheduling and interprocess communication, BSD, which provides the POSIX-style syscall interface, file systems and networking to user mode, and I/O-Kit, the driver infrastructure written in C++. In the end, a short overview on how to extend the kernel with so-called KEXT will be given, as well as an introduction on how to hack the (Open Source) kernel code itself.

Lucy <whoislucy(at)gmail.com>

Inside the Mac OS X Kernel

Debunking Mac OS Myths

24th Chaos Communication Congress 24C3, Berlin 2007

Many buzzwords are associated with Mac OS X: Mach kernel, microkernel, FreeBSD kernel, C++, 64 bit, UNIX... and while all of these apply in some way, “XNU”, the Mac OS X kernel is neither Mach, nor FreeBSD-based, it's not a microkernel, it's not written in C++ and it's not 64 bit - but it is Open Source (with reservations) and it's UNIX... but just since recently.

This paper intends to clear up the confusion by presenting details of the Mac OS X kernel architecture, its components Mach, BSD and I/O-Kit, what's so different and special about this design, and what the special strengths of it are.

History

Unlike many other operating systems, the design of Mac OS X has never been strictly planned and implemented from scratch, instead, it is the result of code from very different sources put together over the last decades.

Mac OS

Mac OS started its life in 1984 on the original 128KB Macintosh as a mouse-operated graphical operating system that, due to memory constraints, did not support multitasking. It wasn't until 1988 that Mac OS supported a very simple form of cooperative multitasking (“MultiFinder”). In the mid-90s, Apple ended up having a ten year old code base designed for a single-tasking system on a Motorola 68000 that now ran on PowerPC CPUs. Parts of the kernel code ran in a 68K emulator, and it still did not support memory protection. There was no way to compete even with Windows 95, which is why Apple started the Copland project in 1994 in order to design and implement a new and modern operating system that would have the Mac OS API and user interface - much like Microsoft did with Windows NT. But although Copland had been heavily advertised with developers, programming books had been published and Betas had been given out, the pieces of Copland never fit together, and the unbearably unstable operating system was scrapped in 1996.

Mac OS Successor

As Apple was in bitter need of a successor for Mac OS, they decided to buy an operating system and build Mac OS compatibility into it. Despite negotiations with the company behind BeOS, Apple finally decided to buy NEXT, the company Steve Jobs had founded just after having left Apple in 1985, and to convert NEXTSTEP/OpenStep into the next Mac OS: Mac OS X.

Mach

The NEXTSTEP operating system was heavily based on Mach. Mach was an operating system project at the Carnegie Mellon University that was started in 1985 in response to the ever-increasing complexity of the UNIX and BSD kernels. As one of the first microkernels, it only included code for memory management (address spaces, tasks), scheduling (threads; a concept unknown to UNIX at that time) and inter-process communication (IPC) - all other functionality typically found in an operating system kernel, like filesystems, networking, security and device drivers, had to be implemented in so-called “servers” in user space. This could be a very big plus for reliability, since a crash in a driver didn't necessarily bring the system down, as well as maintainability, since it imposed strict rules on the interface between the core kernel functionality and the userland servers. Unlike in UNIX, operating system components couldn't just call each other arbitrarily (“The big mess” - Tanenbaum). Another advantage of a microkernel like Mach is the possibility to have several personalities, each of which is a set of userspace servers. This way, a Mach-based system could, for example, run UNIX and Windows applications at the same time. Having a minimal piece of code running in privileged mode that abstracts the hardware and allows different operating systems to run on top of it is basically the same approach implemented by virtualization today. But the typical configuration of a Mach operating system was to have a single BSD server in user mode, i.e. the majority of the

BSD kernel with memory management and scheduling stripped out, and process management built on top of Mach tasks.

The problem with the Mach design was that the kernel was slower than a traditional monolithic kernel because of the extra kernel/user context switches when a server communicated with the kernel or servers communicated with each other. On a monolithic kernel, these were just simple function calls. The simplest solution for this problem is "co-location": The personality servers run in kernel mode, and communication is fast again. While it somewhat defeats the original idea of a microkernel, it still has the advantage of well-partitioned kernel components and a more modern core kernel: The Mach memory management code was later integrated into BSD.

NEXTSTEP

NEXTSTEP, which was released in a 1.0 version in 1989, chose to go with this design. NEXT had removed the core kernel parts from the 4.3BSD kernel and layered it on top of Mach, in kernel mode. This way, NEXT was many years ahead of the competition with NEXTSTEP being the first desktop/GUI operating system that supported preemptive multitasking, memory protection and UNIX compatibility. At first NEXTSTEP only ran on their own Motorola 68K-based machines, but was later ported to SPARC, PA-RISC and i386, when NEXT started licensing it under the name "OpenStep" to other hardware manufacturers, so it was highly portable. When Apple acquired NEXT in 1997, they added PowerPC support and removed support for all architectures other than i386; the latter would serve as the fallback solution when Apple switched from PowerPC to i386 in 2005/2006.

Rhapsody and OS X

With Apple's acquisition of OpenStep, many more changes were made to the operating system which now had the interim name "Rhapsody": They replaced the "DriverKit" driver model with the new "I/O-Kit" system, updated Mach 2.5 with the Mach 3.0 codebase, updated the BSD part with 4.4BSD and FreeBSD code and added support for the HFS filesystem and Apple networking protocols to the kernel. In userland, Mac OS X is pretty much NEXTSTEP/OpenStep, with the native "NS" API renamed to Cocoa, the Mac OS 9 API "Toolbox" ported as a compatibility API (now named "Carbon"), "carbonized" versions of the OS 9 Finder and QuickTime technologies, plus a VMware-like Virtual Machine called Blue-Box ("Classic") that runs OS 9 and its applications unmodified.

Architecture

The Mac OS X kernel, named "XNU" ("X is not UNIX") consists of three main components: Mach, BSD and I/O-Kit.

Mach

Being the only operating system that still uses Mach code (not counting GNU/HURD), Mac OS X has evolved from the original code base quite a bit, but the architecture is basically unchanged. Mach ("osfmk" in the kernel source tree, which stands for "OSF microkernel") calls address spaces "tasks", and one task can contain zero or more threads. Being policy-free, there is little information associated with a task, so, for example, there is no UNIX-style current working directory or environment associated with it. While there are few surprises in the memory management code compared to other modern operating systems, the key distinctive feature of Mach is Mach Messaging. A task can have any number of "ports", which are interprocess communication (IPC) endpoints. One task can subsequently send a message from its originating port to its peer port, and Mach will take care of security, enqueueing, dequeueing, network opacity (ports can be on different machines) and, if necessary, byte swapping. For programming convenience, the Mach Interface Generator ("MIG") can generate stub code from interface definitions, so that two processes can talk to each other using simple function calls, but internally, this will be translated into Mach messages.

BSD

The BSD part of the kernel implements UNIX processes on top of Mach tasks, and UNIX signals on top of Mach exceptions and Mach IPC. UNIX filesystem semantics are implemented here just like TCP/IP networking. And while the VFS (virtual filesystem) component allows plugging in BSD-style filesystems, the /dev infrastructure plugs right into I/O-Kit. BSD exports all the semantics that an applica-

tion expects from a UNIX/BSD/POSIX compatible operating system, like "open()" and "fork()", through the syscall interface.

Since there are basically two kernels in XNU - Mach with its message passing API and BSD with the POSIX API - there are two kinds of syscalls. While both use a single int 0x80/sysenter/sc entry point, negative syscall numbers will be routed to Mach, while positive ones go to BSD. Note that, just like on Windows NT, applications may not use int 0x80/sysenter/sc directly, as this is a private interface. Instead, applications must call through libSystem, which is the equivalent of libc on OS X.

I/O-Kit

When NEXTSTEP was ported to different architectures and was renamed to OpenStep, it got a new driver model, called "DriverKit", which was based on the Objective C programming language and therefore was object oriented, and allowed an inheriting hierarchy of device drivers: For example, there could be a generic IDE/ATA device driver that handled reads and writes of blocks on an IDE bus, a hard disk driver and a CD-ROM driver that subclassed the generic IDE driver, and another CD-ROM driver that subclassed the generic CD-ROM driver to work around some quirks for one specific CD-ROM drive model. This architecture helps a lot to combat duplicate code: In contrast to other operating systems like Linux, a new device driver is not written by copying the closest match and modifying it, but by subclassing an existing driver binary and overwriting some methods with new code. "I/O-Kit" is a higher performance reimplementation of DriverKit in a subset of C++ (no exceptions, multiple inheritance, templates, runtime type information). I/O-Kit supports some classes of drivers in user mode.

KEXTs

I/O-Kit drivers are dynamically linked at runtime, as so-called "KEXTs" ("Kernel Extensions"). KEXT can not only link against the I/O-Kit component, but also against other parts of the kernel. This way, filesystem and networking KEXTs (NKEs) are possible. Every KEXT, which typically resides in /System/Library/Extensions, is a bundle, i.e. a subdirectory which contains the actual binary and an XML description of dependencies and the parts of the kernel it links against.

Other interesting details

The following sections describe some other interesting details of or around the Mac OS X kernel.

Booting

While PowerPC-based Macs use OpenFirmware, Intel-based machines use EFI ("Extensible Firmware Interface"). Both kinds of firmware are a lot more powerful than the 16 bit BIOS still shipping on PCs. While EFI can boot off USB and supports GPT partitioning and FAT32 file systems, the rest of the feature sets of OpenFirmware and EFI are pretty similar: Both can boot off FireWire, and both support APM ("Apple Partition Map") partitioning and the HFS file system, as well as firmware-level drivers. BootX is the bootloader for OpenFirmware, and boot.efi the bootloader for EFI. Both can decode HFS and can therefore read the kernel from the root partition. If there is a "KEXT cache", i.e. a file with all prelinked KEXTs suited for this configuration, that is newer than the newest file in /System/Library/Extensions and newer than the running kernel, the boot loader will load this cache; otherwise, it will go through all KEXTs and load the appropriate ones by comparing them to the entries of the "device tree" which has been passed from the firmware to the bootloader. Later, a KEXT cache will be written to disk to speed up the next boot. This is somewhat similar but more flexible than the Linux "initrd" approach.

Mach-O

Mac OS X does not use the ELF file format for binaries (executables, libraries, KEXTs) like practically all other UNIX systems. Instead, it uses Mach-O, which has roughly the same feature set, but one interesting addition: A single, so-called "fat" or "universal" binary can contain code for more than one architecture. So on OS X 10.5 Leopard, for example /usr/lib/libSystem.dylib contains code for PowerPC, PowerPC 64, i386 (32 bit Intel) and x86_64 (64 bit Intel). This way, a single Mac OS X 10.5 Leopard installation DVD can boot on four different architectures, and there is no need for "lib/lib64" (64 bit Linux) or

"SYSTEM/SYSTEM32/SYSTEM64" (64 bit Windows) style duplicate directories for different architecture/bitness versions of the same code. The function grade_binary() in the kernel's Mach-O loader decides which part of the binary to run. If the system is an i386 and the Mach-O file contains only PowerPC code, execution will be handed to Rosetta.

Rosetta

Rosetta is a compatibility solution based on Transitive's QuickTransit technology that allows running (32 bit) PowerPC code on i386 CPUs. This is done by dynamically recompiling the PowerPC code into native i386 code and managing the interfaces between emulated and native code - in practice, this means byte-swapping all data passed between i386 and PPC code, because i386 is Little Endian and PPC is Big Endian. From a performance standpoint, the optimal design would have been to only emulate the application and to use the native versions of all libraries it links against, but this would have been very impractical, since the interface between native and emulated code would have been very broad. A much easier way to achieve high compatibility is to run the complete application including all of its libraries in emulation, and only byte swap when the application makes syscalls to the native kernel. A side effect of this approach is that you potentially need all PPC versions of the system libraries installed on an Intel system, as soon as you only use a single PowerPC application in emulation.

A user can easily make experiments with this amazing technology by invoking /usr/libexec/oah/translate manually to force emulation of PowerPC code, even if an executable is available in native code.

Intel specifics

While i386 support in XNU has existed since the mid-90s, and has been a shipping feature of OpenStep, the i386 part had not been used in Mac OS X until the advent of Intel machines in 2005/2006. And with the introduction of the 64 bit Mac Pro in 2006, x86_64 (AMD64, Intel64, EM64T, x64, ...) support has been added to XNU - but XNU is not a 64 bit kernel, though. XNU supports 64 bit user mode applications, but it is 32 bit itself. Since porting a 32 bit kernel to 64 bit is a big task, it could not be done in just half a year between the introduction of the first Intel machines in January of 2006 (until then, Apple developers had worked on finalizing the 32 bit i386 version) and the introduction of the Mac Pro in August.

There is just a single kernel image for 32 and 64 bit Intel: It is loaded as a 32 bit process in 32 bit protected mode on both kinds of machines, and if 64 bit support is detected, the kernel switches into long mode compatibility mode - a mode that supports running 32 bit code, but also allows easy switching to 64 bit code. So the whole kernel code is still unmodified 32 bit code, but tiny stubs that deal with copying between user address spaces (which can be 64 bit), and the syscall and trap handlers are 64 bit code. Next to being an easy port, this has the extra advantages that the 64 bit capable kernel can still easily support 32 bit KEXTs, and conserves memory by being able to use 32 bit pointers throughout a large part of kernel code. On the flip side, the kernel cannot use the extended x86_64 register set and is restricted to a 32 bit address space.

But while all other common 32 bit operating systems like Linux, Windows and the BSDs split the address space into 2 GB for user and 2 GB for kernel (2/2) or 3 GB for user and 1 GB for kernel (3/1), the i386/x86_64 version of XNU uses a 4/4 split: While the kernel is running, the user's data is not mapped into its address space, and while user code is running, the kernel is not mapped. So user and kernel can each have 4 GB of address space with the disadvantage of being less efficient in copying of data between user and kernel. But this way, kernel mode can map more devices into its address space (like video cards with a lot of memory), and manage more RAM, thus pushing out the limit when a true 64 bit kernel is required.

iPhone

Mac OS X runs on 32 and 64 bit PowerPC and i386/x86_64 ("Intel") Macintosh machines, on the Apple TV set-top-box, which is also i386 based, and on the iPhone and the iPod touch - these devices have ARM CPUs. Specifically for these devices, XNU and parts of the Mac OS X userland have been ported to ARM. The ARM kernel does not support loading arbitrary KEXTs and is digitally signed, but

otherwise mostly equivalent to the PowerPC and i386/x86_64 versions.

What makes XNU great

While XNU might not be as scalable or as tidy as other operating systems (but catching up), it is a very modern UNIX with novel ideas and unique features:

- The kernel extension ABI is stable over several major releases of the OS.
- Fat/universal binaries allow for a single install CD or hard disk installation that runs on different CPU architectures, without the clutter of duplicating files or directories. Furthermore, 3rd party application vendors can ship a single binary that runs on multiple architectures.
- I/O-Kit allows code reuse for drivers without code duplication.
- The KEXT cache is a clean way to speed up boot times.
- The clear separation between Mach, BSD and I/O-Kit helps keeping the cost of code maintenance low.
- The powerful Mach Message API is useful for user mode applications.
- Since Mac OS X 10.5 Leopard, the i386 port of OS X is the only operating system with full POSIX-conformance that doesn't contain AT&T UNIX code.

Open Source & Hacking

With every minor operating system release (i.e. 10.5.0, 10.5.1...), Apple usually releases the whole set of source code for all components of the system that are under an open source license. which is basically everything but the GUI. About half of these packages are patched versions of common open source projects (like “bash” and “perl”), the rest is Apple code, and is released under the “Apple Public Source License” APSL, which is a BSD-style license. This makes it compatible with the standard BSD license, as well as with the OpenSolaris CDDL. But there is no live source code repository for developers visible outside Apple, so there is no real open source community that does any development on the APSL components. But there are other uses for Open Source: It helps KEXT developers debugging, it allows governmental or educational institutions to build their own versions, with added security for example, and it allows commercial companies or universities to add functionality to the kernel, either to sell it, or for research (SEDarwin, L4/Darwin).

But the source code is not necessarily complete. The XNU source code lacks most of the ARM bits, and Apple also states that other parts have been left out because of trade secrets with Intel. But a kernel compiled from the open source can still be used as a drop-in replacement for the shipping binary.

Revisiting the Buzzwords

- The OS X kernel is not Mach. The OS X kernel is called “XNU”, which consists of Mach, BSD and I/O-Kit.
- The OS X kernel is not a microkernel. Although Mach has been used as a microkernel in other projects, XNU is a very traditional monolithic kernel with BSD and (most) drivers in kernel mode.
- The OS X kernel is not based on FreeBSD. The BSD part is based on 4.4BSD with some code from FreeBSD, NetBSD and others. The OS X userland UNIX tools are mostly based on FreeBSD code, though.
- The OS X kernel is not written in C++. The I/O-Kit part is written in a subset of C++, but Mach and BSD are written in C.
- The OS X kernel is not 64 bit. It supports 64 bit user mode applications on a 64 bit PowerPC or Intel CPU, but the kernel itself runs in 32 bit mode and is bound to the 4 GB address space limit.
- The OS X kernel is Open Source, but there is no live source code repository visible outside of Apple, and the released source does not necessarily contain all code, but can be compiled into a working system.
- The OS X kernel is UNIX, but only since OS X 10.5 Leopard, and only for 32 bit i386, since this is the configuration that passed the POSIX conformance test and may therefore use the OpenGroup's “UNIX” trademark.

References

- Singh, Amit: Mac OS X Internals. A Systems Approach; Addison-Wesley, 2006.
- http://kernel.macosforge.org/
- http://www.opensource.apple.com/darwinsource/

Science
lecture
Tag 3 12:45
Saal 3
en

Jens Kaufmann

Introduction in MEMS

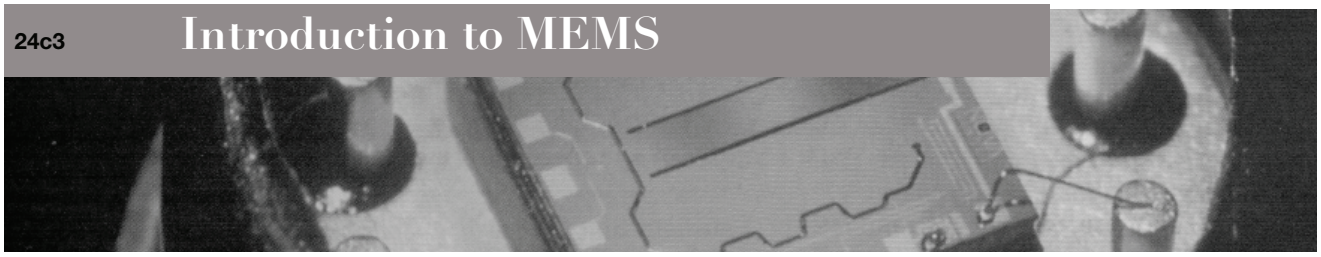
Skills for very small ninjas

MicroElectroMechanical Systems or MEMS are as part of micro system technology, systems with electrical and mechanical subsystems at the micro scale. It is basically an introduction in the technology and in its potential for hardware hacks and potential ways of homebrew devices.

Compared to a micro processor, a small sensor or actuator, which normally consists of just one function a micro system combines the data acquisition, processing, and forwarding in itself. If this micro system now contains mechanical part to interact with its environment it is considered to be a MEMS. With constantly increasing experience in MEMS manufacturing the prices per system dropped and the use of the highly sophisticated devices move from strictly automotive, R&D and military applications into consumer products. The wiimote and the iPhone are just two well known products which improve the user experience by the intelligent use of the smart systems.The delay of invention and market introduction of MEMS is mostly caused by the substantial investments to be done to produce this kind of device. The most technologies commonly used until now are transfered from the microchip manufacturing. The so called silicon

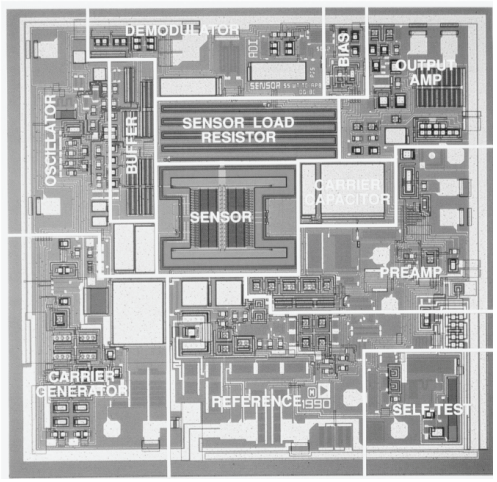
24c3

Introduction to MEMS



What are Mems

MEMS is the acronym for MicroElectroMechanicalSystem and describes a very small device with expanded functionality compared to microelectronics. Mechanical structures are used to interact with the environment to allow sensing or act. The term MEMS is often used in combination with prefixes or alterations to describe the integration of other functionality, like RFMEMS (Radio Frequency), BioMEMS (mostly microfluidics) or MOEMS(optical microsystems).



Monolithic integrated accelerometer form Analog Devices

The first developments that can be considered as Microsystems were made in the 1970s like the compact disc or LC Displays. Also the fundamental processes like anisotropic etching of silicon and the LiGA process were developed at this time. This opened up the path for first the academic successes in the 1980s and than the commercial ones in the 1990. Microsystems can be found today in almost every commercial sector, Information and communication, in entertainment, automotive and avionic, as well as medical and health related applications. But the military is still one of the biggest sectors for potential applications.

MEMS are always systems that consist of different components with three major functions: input, processing and output. This is what differentiates a micro system from a micro structure, and so therewith allowing interactions with the environment. And so this different components can be manufactured separately (modular integration) or all on one substrate (monolithic integration) as shown above. [1]

What kind of MEMS are they

A microsystem can be classified by the functionality of the system, sensor, actor or processing unit. But it is common to classify by the kind of components it consists of.

functionality	components	examples
electronics	*microelectronic components*	logic, memory, mixed signals
	RF microstructures	antennas, transformers, passive components
mechanics	*micro sensor*	pressure, acceleration, momentum, temperature, flux
	micro actuator	micro relays, pumps, valves,
	micro fluidics	reactors, dosing systems, separator
	micro acoustics	transducer, filter, signalling,
optics	*micro optics*	fibre optics, mirror arrays, spectrometer
chemistry/ biology	*micro chemistry/ biology*	Analyse

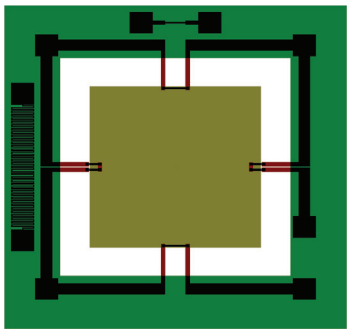
24c3 Introduction to MEMS

How MEMS are made

The typical MEMS are made out of single crystal Silicon discs. These discs are made by pulling a circling start crystal out of a moulded Silicon bath. The rod which was manufactured will than be sliced, lapped and polished. This ensures a bulk material of constant quality.

The typical silicon processing for MEMS is based on the lithography used in micro electronics. A photo mask is necessary for every step in the process that requires selective exposure. The mask can be positive of or negative depending on the chosen resist. The process flow looks always like this:

1. superimpose photoresist
2. expose photoresist
3. develop photoresist
4. etch or modify uncovered material OR growth of a new layer within the resist
5. resist stripping
6. optional: removal of sacrificial layer(s)
7. optional: deposit a layer onto the whole surface
8. go to 1

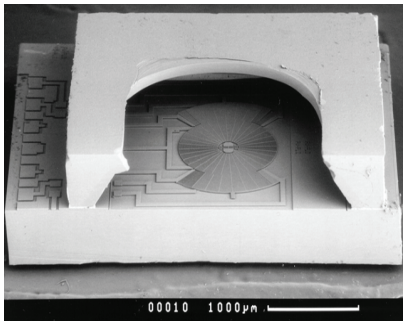


4 layer mask for a bulk micro machined pressure sensor

To achieve a simple system like a pressure sensor it is necessary to repeat this flow 17 times. This pressure sensor is a good example of Silicon Bulk machining. Some structures are formed on the surface of the wafer and than the mechanical structure is formed by modifying the wafer itself - the so called bulk material [2]

The other way to make MEMS from silicon is surface micro machining. In this case the mechanical structure is formed by:

1. depositing and structuring a sacrificial layer,
2. depositing and structuring of a poly silicon layer,
3. removing of the sacrificial layer,



Surface micro machined Gyroscope

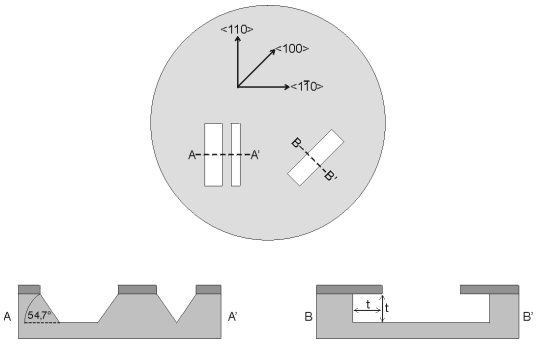
Generally, an accelerometer is often manufactured using this approach. A normal accelerometer is formed by cantilever with a weight at the end.

Another widely used technology is LiGa. LiGa is the German acronym for Lithography, electroplating (Galvanoformen), molding (Abformen). In the beginning it was just possible by utilising high energy x-rays to expose a PMMA resist. This resist was covering a conductive seed layer which made it possible to electroplate in the mould and so electroform large 2.5D metallic structures. The electroplated structure is than removed from the wafer and becomes a mould itself for micro injection moulding. This gives the possibility to make many parts in a relatively cheap way. The biggest disadvantage is the necessity of a synchrotron to generate the x-rays.

Today UV LiGA uses coherent UV light and a negative resist like SU-8; which is commonly used to achieve similar structures ("Poor mans LiGA"). The drawback with this method is the relative low resolution because of the long UV light wavelength.

Why is silicon still used for MEMS

Silicon is still the material of choice against all odds. The main reasons therefore are the very good mechanical properties, the possibility for embedded electronics and the anisotropic atomic crystalline structure. This causes also non uniform etch rates. The rates between the (100) plane and the (111) is from 100:1 up to 400:1, depending on the temperature. That means the (111) plane can be considered as a natural etch stop. The natural etch stops combined with artificial stops make structures possible that cannot be achieved with outer isotropic materials. All this possibilities give the device designer perfect ways to integrate his ideas in one monolithic design. [1]



Standard anisotropic etch geometry

And if he is part of a developer team for a semiconductor manufacturer he will have all the equipment to make the device at his fingertips. That explains why the big players in the MEMS market are mostly semiconductor companies.

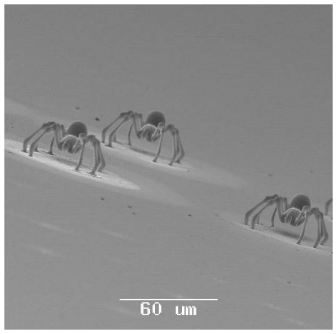
Will we see home grown MEMS in the near future

The manufacturing of MEMS is still a large scale batch process. Even a small cleanroom with the necessary facilities to run one process chain for silicone is between 5 and 10 million €. And such a process has an intrinsic inflexibility to design changes, as they are costly and difficult.

Errors are really costly too, so this which makes it unavoidable to manufacture tremendous quantities to produce just cost-covering.

The industry experiences the same problems at the moment with a drift in the market for tailored solutions. "Responsive manufacturing" is the weapon to face this new development. That means that production capabilities must be build that allow producing a product cost-effectively in a "Batch of one".

In MEMS this is even more difficult than in other industries because everything is based on one material. The academic community is constantly trying to develop new processes with new materials to enable manufacturing by smaller players without heavy financially resources.



STL generated spider models made from Resin at the LTZ Hannover

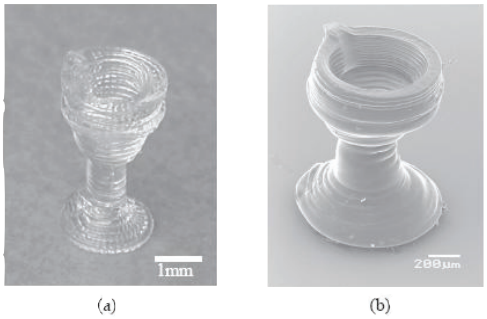
And this is where fabbing takes its place in future home grown MEMS development. A fabber is basically a 3D-Manufactuing device that allows the user to manufacture physical free form objects. The most ideas are based on rapid prototyping/manufacturing of 3D structures. The additive modelling generates 3D structures by successive adding materials at the right place. The most rapid prototyping technologies are working with this approach like stereo lithography and fused deposition modelling. Electro deposition or chemical vapour deposition are also considered as additive modelling. The superiority of this method

compared to subtractive methods is due to the fact that less waste is produced and the design space is not predestined.

Different concepts out of the rapid prototyping have proven themselves as capable of producing microstructures. The stereo lithography (STL) for example uses a liquid epoxy resin with a photo active linker as material. This resin is locally cured by writing with a laser beam onto the liquid level. The cured layer sticks to the vertical moveable stage. This stage then is sunk further into the resin so that liquid resin will cover the object and the next layer can be cured by the Laser. No support structures are necessary. The laser centre in Hannover, Germany has demonstrated they can produce micro parts with this technology. [3]

Based on a similar idea as the STL is the Selective Laser Sintering (SLS). Metal, polymer or ceramic powder are selectively fused together by the laser. The biggest advantage is the different material which can be used. [4]

Fused Deposition Modelling (FDM) uses a standard Cartesian robot to extrude liquefied thermoplastic onto the working stage. The working material can be changed at any time during the process. A support material is needed for overhanging structures. Recent research has shown that this method is also capable of manufacturing micro parts, as well as form part out of LTCC-like materials. [5]



Wineglasses from Nagoya University, (a) is 4mm high, (b) is 1500 µm high

Best technologies for MEMS

The Manufacturing of MEMS needs a high degree of accuracy, which can be only provided by STL, SLS and FDM. The condition for a variety of different materials cannot be satisfied by stereo lithography, which is the most accurate process at the moment (<1µm). The need of the selective laser sintering for a high power laser makes it not commonly affordable. That leaves Fused Deposition Modelling as the method of choice.

Fabbing can also be used by its own or in combination with other techniques. The most processes have been already described before or don't need any explanation. By using FDM and different material a large variety of MEMS can be formed. Further more there are new or hybrid technologies, which needs to explained in more detail.

Plating mould forming (soft lithography)

Electroforming of metallic parts was utilising a patterned photoactive resist onto conductive surface as mould for the electroplating process. This process requires usually a several facilities and steps. Direct deposition of a polymer by FDM or syringe deposition reduces these steps to deposition of the mould, electroplating itself and optional removing the mask and seed layer. [6]

Piezo ceramic FDM process

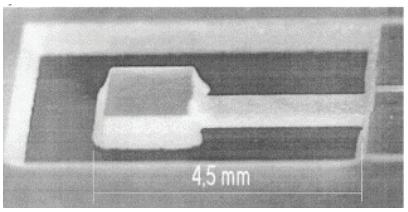
The deposition of ceramic containing polymer can be used to produces 3D-ceramic structures. As proposed by Safari and Danfarth. LTCC (low temperature co-fired ceramic) is a ceramic compound in a polymer matrix. It is then fired at 850 °C. [5]

Local plating nozzle

It was shown that special nozzles can be used to deposit metal in a defined area. They used a double nozzle with inlet and outlet to render a drop of electrolyte between the nozzle and the surface. And so the plating can take place just in the area, which is covered with electrolyte.

Powder blasting

A subtractive method which could allow cheap and fast processing of mesoscale Microfluidic chips is the powder blasting method. Thereby a polymer substrate is covered with a metallic mask. Then the open areas of the substrate are exposed to a stream of a few microns big alumina particles. This particle stream erodes with a different rate, so that it can form 2.5D structures cheap and easily.



Picture of an accelerometer beam realised in two steps by powder blasting from the two substrate sides

References

[1] "Mikrosystemtechnik fur Ingenieure" by W. Menz and P. bley, VCH, ISBN 3-527-29003-6, Weinheim, 1993. (In German)

[2] "Fundalmentals of Microfabrication" by Marc Madou, CRC Press, ISBN 0-8493-9451-1, New York, 1997.

[3] “Metal and polymer microparts generated by laser rapid prototyping “ by Neumeister, A.; Czerner, S.; Ostendorf, A.In: 4th international congress on laser advanced materials processing, 16.-19. Mai 2006, Kyoto. Paper No. 050873

[4] "Selective Laser Micro Sintering with a Novel Process" by Horst Exner, Peter Regenfuss, Lars Hartwig, Sascha Klötzer, Robby Ebert.

[5] "Processing of Piezocomposites by Fused Deposition Technique," A. Bandyopadhyay, R.K. Panda, V.F. Janas, M. Agarwala, S.C. Danforth and A. Safari, J. Am. Cer. Soc., 80, 6, 1366-72, (1997).

[6] “ Fabrication of PLGA scaffolds using soft lithography and microsyringe deposition” by Giovanni Vozzi, Christopher Flaim, Arti Ahluwalia and Sangeeta Bhatia, BiomaterialsVolume 24, Issue 14, , June 2003, Pages 2533-2540.

Hacking
lecture
2007-12-28 17:15
Saal 3
en

Peter Molnar
Roland Lezuo

Just in Time compilers - breaking a VM

Practical VM exploiting based on CACAO

We will present state of the art JIT compiler design based on CACAO, a GPL licensed multiplatform Java VM.
After explaining the basics of code generation, we will focus on "problematic" instructions, and point to
possible ways to exploit stuff.

A short introduction into just-in-time compiler techniques is given: Why JIT, about compiler invocation, runtime code modification using signals, codegeneration. Then theoretical attack vectors are elaborated: language bugs, intermediate representation quirks and assembler instruction inadequacies.With these considerations in mind the results of a CACAO code review are presented. For each vulnerability possible exploits are discussed and two realized exploits are demonstrated.

http://cacaojvm.org/ cacaojvm.org

Just in Time compilers - breaking a VM

Roland Lezuo <roland.lezuo@chello.at>
Peter Molnár <peter.molnar@wm.sk>

November 18, 2007

1 About CACAO

CACAO is a multiplatform Java Virutal Machine featuring a just-in-time compiler. Although CACAO features an interpreter, by default it works in JIT-only mode, so all code gets compiled prior to execution. The CACAO project was started in 1997 as a research project at Vienna University of Technology. Today the project is fully covered by the GPL v2 license.

2 CACAO Codegenerators

CACAO provides code generators for many platforms: currently code generators for ALPHA (FreeBSD, Linux), ARM (Linux) i386 (Cygwin, Darwin, FreeBSD Linux), MIPS (Irix, Linux), POWERPC (Darwin, Linux, NetBSD), SPARC64 (Linux), x86_64 (Linux) and s390 (Linux) are available. A code generator has to implement a defined internal interface consisting of a set of exoported functions and symbols and is linked in statically into the virtual machine.

3 Java bytecode

The Java compiler does not produce machine code which can be executed on the host CPU directly but an intermediate representation called *bytecode* targeting a virtual machine. There are around 200 bytecode instructions defined in the Java Virtual Machine Specification[1] The most notable difference between java byte code and usual machine code is that bytecode instructions

[1] `http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html`

Listing 1: Stack operations

```
iconst_3
iconst_5
iadd
```

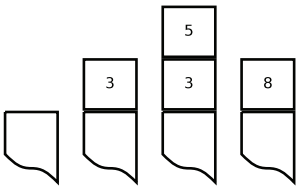


Figure 1: Stack changes

don't use registers as operands, but operate on a *operand stack* instead what leads the notion of a computation model called *stack machine*.

The program in listing 1 manipulates the stack as shown in figure 1: the instruction `iconst_3` pushes the integer 3 on top of the stack, `iconst_5` pushes 5, `iadd` takes the two topmost elements of the stack, adds them and pushes the result back. The stack is growing from the bottom to the top.

The operand stack consists of 32 bit wide *stack slots*. A single stack slot can accomodate a value of the primitive types `boolean, char, byte, short, int` or an object reference. To accomodate a `long` or `double` value, two stack slots are used.

Instructions are variable sized and consist at least of one byte - the opcode optionally followed by several bytes representing operands embedded in the instruction itself. The `getfield` instruction for example is used to retrieve the value of an object's field and contains a two byte field specifying the fields index. The object reference is poped from the stack and the result - the field's value - is pushed on the stack.

Arithmetic instructions are typed and special variands are defined for the various primitive types: (e.g. `iadd` adds two `int` whereas `ladd` adds two `long` values).

4 Register allocation

A naive compiler would generate machine code that would map the java operand stack to a stack located in memory. This is actually the approach used by the Jikes RVM baseline compiler and the approach kaffe's JIT used to use but is suboptimal, because of the property of memory accesses beeing

Listing 2: Codegeneration macros

```
#define M_OP3(opcode,y,oe,rc,d,a,b) \
  do { \
    *((u4 *)cd->mcodeptr) = (((opcode)<<26) | ((d)<<21)\
        | ((a)<<16) | ((b)<<11) | ((oe)<<10) | ((y)<<1)\
        | (rc)); \
    cd->mcodeptr += 4; \
  } while (0)

#define M_IADD(a,b,c) M_LADD(a,b,c)
#define M_LADD(a,b,c) M_OP3(31, 266, 0, 0, c, a, b)
```

expensive. CACAO instead allocates the slots of the java operand stack to CPU registers, for example stack slot 2 to the general purpose register 16. In the case that there are more stack slots needed than registers available, stack slots are mapped to memory locations. On RISC plattforms, they need to be loaded into registers before usage, and stored back afterwards.

5 Code generation macros

The code generator iterates over all instructions of the method to be compiled and depending on the opcode, translates them into native machine code. The generated machine code is written to temporary memory and afterwards copied to an executable memory location. It is generated by macros, so care has to be taken for side effects of arguments which could be evaluated twice. To ease maintenance of the code generators, all platforms try to adhere to naming conventions originally inspired by the alpha architecture. Listing 3 shows the implementation of java's `iadd` operation, and addition of two 32 bit signed values on POWERPC64. First, the operands are loaded, then the macro `M_IADD` is used to emit machine code that adds the values in two registers and stores the result in a desitnation register, `M_EXTSW` is needed for sign extension and is platform specific and finally the result is stored in the destination register. `jd` and `iptr` contain a pointer to the state of the JIT compiler and the currently processed instruction. The implementation of the macro `M_IADD` is shown in listing 2.

The operands of bytecode instructions are allocated to registers or memory. On load-store architectures, memory operands need to be loaded into registers prior to use what is achieved using the function`emit_load_s1`, `emit_load_s2`

Listing 3: Codegeneration for `iadd`

```
case ICMD_IADD:
  s1 = emit_load_s1(jd, iptr, REG_ITMP1);
  s2 = emit_load_s2(jd, iptr, REG_ITMP2);
  d = codegen_reg_of_dst(jd, iptr, REG_ITMP2);
  M_IADD(s1, s2, d);
  M_EXTSW(d,d);
  emit_store_dst(jd, iptr, d);
  break;
```

and `emit_load_s3`. In case the operand was allocated to a register, they simply return the register number, otherwise, code is generated to load the memory operand into a scatch register and the number of the scratch register is returned. The destination register of an operation is retrieved using the function `codegen_reg_of_dst`, which may again return a scratch register for memory destinations and finaly `emit_store` generates code to store the result in case it belongs to memory. See listing 3 for an example showing the implementation of the `iadd` byetcode instruction on POWERPC64.

6 Post compile time code patching

One reason the generated code is written into a buffer is due to unresolved jumps. Imagine a forward jump in a method wheter the target address points into code still not generated and the compiler does not know the exact offset in advance as it depends on the instructions in between. For that reason a post-pass has been added to the compiler which patches the code after generation. During machine code generation a function named `codegen_add_branch_ref` is responsible for collecting positions of branches that could not be resolved and associating them with target basic blocks. The branch instructions are then patched using the machine dependend function `md_codegen_patch_branch` to contain the correct offset after the complete method has been compiled. By using the machine dependent patching function the post compilation phase can be kept platform independent.

7 Data segment

The generated code makes use of constant values: integer constants, address constants (function entry addresses, addresses of static members). Some

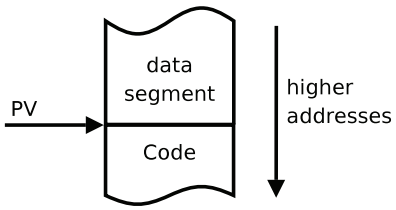


Figure 2: Data segment layout

architectures support immediate values of the native word size, so such values can be embedded in the instruction flow whike other architectures have a fairly limited range of immediate operands, so those values need to be placed into memory. Beacause of this the executable method's code has a block of memory prepended called the *data segment* (see figure 2) holding those constant values. On most architectures, there is one `pv` register reserved to hold the *procedure vector* - the current method's entry point. The values on the data segment can then be loaded relatively to the `pv` register with negative offsets, or relatively to the current program counter with negative offsets.

The data segment of each method always contains a *method header*. This is a data structure containing metadata about the method, like a pointer to a method descriptor, the stack frame size, the exception table, the line number table (see **??** for details).

8 Runtime code patching - Patchers

In java, classes are loaded by the run-time system only if they are needed. If generating code for a method that depends on other classes (uses static fields, calls methods), the runtime system needs information about the referenced class, and therefore it has to be loaded as well. One attempt called *eager loading* consists of loading all those referenced classes at compile time but it showed to be suboptimal, because at run-time, the code using the referenced class may actually never be reached. A better attempt is to deffer expensive class loading to the point, where the code that uses the class is reached. This is called *lazy loading*.

For *lazy loading*, incomplete code that has to be *patched* at run-time with the missing information is generated. The first instruction of the imcomplete code portion is replaced by a trap instruction and a *patcher reference* is created: a datastructure containing data about the missing information associated with the position of the trap instruction.

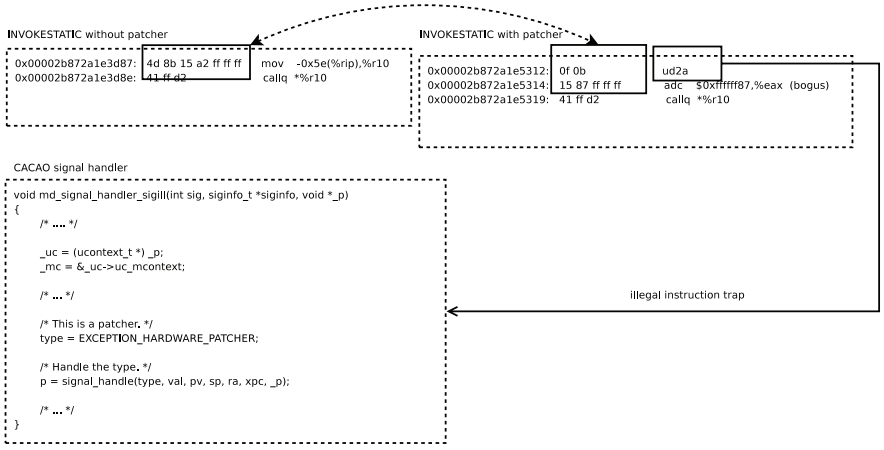


Figure 3: Patcher assembler output (new)

Consider the example of a `getstatic` instruction, which loads a static field of a given class. The class may be unresolved when the bytecode is translated in which case the runtime system has to load and initialize the class, resolve the address of the member prior to execution of the generated code. For this purpose the first instruction of the machine code sequence is replaced by an illegal instruction. Once it is reached, the operating system delivers a signal the the virtual machine and control is passed to the registered signal handler. The signal handler needs to be able to differ patchers from exceptions, so it first examines the failing instruction, whether is really corresponds to a patcher call. The handler then looks up the proper patcher by using the mapping of positions to be patched to *patcher references* and invokes. The code generator needs to provide a function called `emit_trap` capable that generates a trap instrucion.

Figure 3 shows the generated assembler code on the x86_64 architecture: the illegal instruction (`u2da`) is generated where patching is needed and once reached control flows to a signal handler written in C. The disassembler wrongly interpretes the bytes `15 87 ff ff ff` as `adc` instruction. They are part of the offset of the `mov` instruction covered by the `ud2a` instruction.

A race condition exists when patching the trap instruction in case he instruction can not be overwritten atomically on multiprocessor machines. One thread could just patch back the original code, while a different thread executes exactly this code and comes across a half patched instruction. For that reason single word instructions are used for trapping, as they can be written back atomically.

9 Compiler invocation

Beacause just-in-time compilation of methods is expensive and accounts to run-time, CACAO tries to deffer it, simillary as it does for class loading. A method is normally compiled the first time it is called. To achieve this, when a class gets loaded, for each method a so called compiler stub is generated. A compiler stub is a small piece of code, usually a single trap instruction combined with a pointer to the method's descriptor. Pointers to compiler stubs are placed where method entry points would be placed normally: in the class descriptor and in virtual function tables.

If such a compiler stub is invoked, the trap instruction causes control to be passed to a signal handler which extracts the method descriptor from the stub and passes it to the compiler subsystem. The compiler generates machine code for the method and returns the method's entry. Then, the machine code before the call instruction is examined, to determine the *method pointer*: the address where the pointer to the stub's entry was loaded from. This is a virtual function table entry, the data segment, or an immediate operand in executable code. This location is then overwritten with the actual method entry, so that further calls to the method are redirected to the newly generated machine code.

10 Exceptions

Exceptions are an integral part of the Java language used a lot. Nonetheless exceptions are rare events and occur irregularly.

Each method has an exception handler table associated. This table describes the start and end instruction of each exception handler directly corresponding to the Java language `try` clause. When an exception occurs at some point in the program, a lookup is performed in the exception table. The type of the occurring exception is compared to the type of each handler covering the throwing instruction.

If a match can be found the handler is executed, else the exception is propagated outside the method. For the caller this looks like a throwing invoke instruction. As the caller of a method is unknown at compile time, the caller has to be determined at runtime. This is achieved by looking up the return address which is stored on the stack. The offset is known as CACAO knows about the stack usage of each method. Stack space is allocated on method entry and no dynamic allocation is performed.

An operation called "stack unwinding" is performed whenever an exception is propagated to its caller. As control flow continues at the invok-

ing instruction all callee saved registers have to be restored for each stack frame unwound. Callee saved register are stored on the method stack when a method is entered, therefore the restore operation is implemented by loading these registers from known stack locations.

This process either terminates when an appropriate handler has been found or the whole stack is unwound in which case the exception is unhandled and the program will be aborted.

In CACAO no explicit code is generated for calling back the runtime when an exception occurred but an illegal memory operation is performed. POSIX compatible operation systems provide a signal handling mechanism which invokes a function in this case. This signal handler tests if the memory operation was performed intentionally and if so it calls the exception handling code. In case the memory access took place unintentionally an internal exception is thrown and the vm aborts.

When native functions have been called they could have thrown an exception too. Natives can not throw exceptions directly but have to notify the runtime by setting a flag in the environment. When they return the environment is checked for an exception and exception handling code is executed when needed. Exception handling is complex because natives may call back into Java code. The stack layout is only known in JIT code, native code has a different stack layout and stack unwinding would fail when a native frame is found. Therefore a chained data structure called stackframe info is built up when invoking natives. Figure 4 illustrates this chaining. Technically there are no `stackframeinfo` structures for JIT frames, as this stack layout is known and contains all needed information already.

11 Bytecode Verification

Because the java virtual machine was designed to provide a sandbox environment, it can't just start executing untrusted bytecode. It would be easy to construct malicious bytecode that if executed would crash the virtual machine. Therefore all bytecode is subject to verification prior to execution. Bytecode verification includes basic sanity checks of the class file, type checking of bytecode instructions, checks for operand stack underflow and enforcement of access protection as required by the java language.

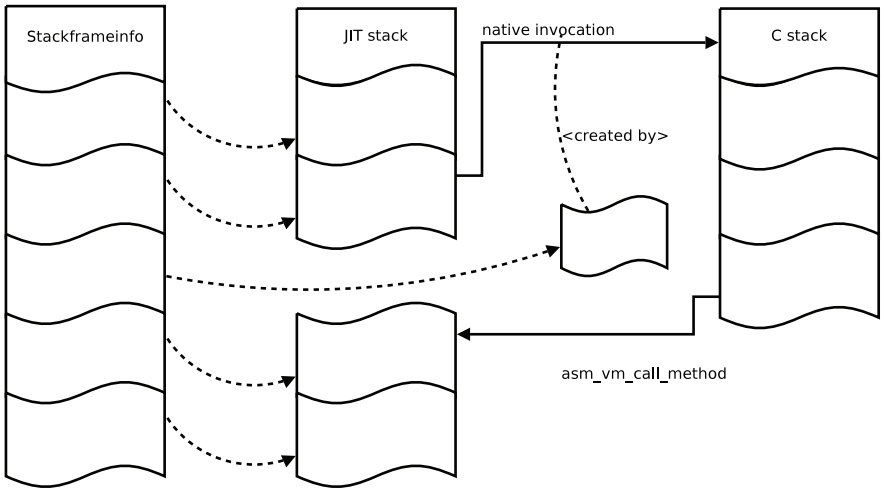


Figure 4: Stackframeinfo chaining with native invocation

12 Problematic byte code instructions

When looking for security problems you should first start by looking at "strange" behaviour defined in the specification. The Java Virtual Machine Specification is available online. Chapter 6 ha a list of all bytecode instructions. A JVM vendor has to implement them acording to their specification. By looking through that list some strange instruction show up.

- TABLESWITCH, LOOKUPSWITCH The tableswitch instruction is used to implement the switch/case statement and is an optimization of the more generic lookuptable instruction. The lookuptable is followed by possible 2^{32} pairs of integer, address pairs. Tableswitch is followed by 2^{32} possible addresses. That is quite a number! Espcially when one also knows that the size of a single method is limited to `0xFFFF` bytes by limitations from the classfile format.

- JSR, RET Another example are the jsr and ret instructions. Their purpose is to implement the try/finally clause of the Java language. The jsr instruction does no invoke any methods (despite its name), it jumps to the finally block and stores the return address on the stack. The ret instruction fetches the return address from a local variable, for an intentional asymetry. The bytecode verifier has to treat return addresses as an additional type to prevent hackers from returning to an integer value they calculated.

This alone are no security problems per se, but they are subtile details which have to be implemented 100% correct to keep the sandbox tight.

13 Problematic assembler instructions

When translating the byte code into machine code appropiate instruction have to be selected. There are different approaches for code generators. Some vendors define a description language and generate the code responsible for instruction selecting, others implement this by hand. Whatever approach is taken, the instructions available are determined by the architectur the code is executed on.

13.1 POWERPC64

The POWERPC64 architecture is an enhancement of the POWERPC architecture and offers 64 bit address space and a 32 bit compatibility mode. All instruction have a fixed 32 bit size. Immediate values are of course even smaller than 32 bits. As a consequence loading a 64 bit address takes more than 1 assembler instruction.

```
lis 4,msg@highest   # load msg bits 48-63 into r4 bits 16-31
ori 4,4,msg@higher  # load msg bits 32-47 into r4 bits  0-15
rldicr  4,4,32,31   # rotate r4's low word into r4's high word
oris    4,4,msg@h   # load msg bits 16-31 into r4 bits 16-31
ori     4,4,msg@l   # load msg bits  0-15 into r4 bits  0-15
```

It takes 5 to be exact. When generating code the size of the generated code is an important factor. Not only for execution speed. And using 5 instruction to load an address (something happening very frequently) can not be afforded. For that reason relative addressing modes are used whenever possible. Assuming that register `r12` contains a valid base address loading an 64 bit value may be implemented as short as the next listing shows.

```
ld      4,0x1234(12)
```

This is just one instruction. In CACAO a datasegment is used to store constant values and a register is reserved to point to the start of the datasegment. So when needing to load an address, a relative addressing load instruction can be used.

The problem here is that the offset is limited to 13 bits, that is 8192 bytes or 8 KiB. The interesting question is what happens for bigger offsets? That depends on the implementation, but it will probably be one of the following 3 cases:

- good: The compiler checks the offset, detects the overflow an emits an instruction sequence capable of correctly handling the case.

- not so good: The offset is trimmed to fit into 13 bit, an integer overflow occures which can lead to an exploit.
- even worse: The offset is not trimmed. As most code generators OR together bitfields it is very likely that the instruction will be changed. This can most likely be exploited.

14 Examples found in CACAO

14.1 PPC64 32 bit interger overflow vulneribility

When loading addresses the offset is truncated to 32 bit (M_LLD macro in codegen.h). This leads to offsets larger than 4 GiB to wrap around and accessing the datasegment at the beginning. The attacker has full control over the contents of the datasegment as the content is determind by the method executed. One way to fill the datasegment is by creating address and interger constans (ICONST and ACONST bytecode instructions). The exploit is of theoretical nature as a 4 GiB sized datasegment implies a 4 GiB sized class file which is not possible.

14.2 PPC64 25 bit integer overflow vulneribility

The POWERPC64 branch instruction takes a 23 bit offset argument, but needs 4 byte aligned target addresses, which effectivley gives a 25 bit branching offset. In CACAO conditional branches are not tested correctly for an overflow and branch addresses are trimmed to fit into 23 bit. An branch offset of `0x3FFFFFF` will be interpreted as `-1` and therfore jump backwards instead of forwards. By jumping backwards the datasegment is targeted which is in control of an attacker. The size of a method must be around 64 MiB for this explot to work. As java methods may only consist of 65535 instructions (classfile limitation) each bytecode instruction would need to use 1024 bytes of instruction code. There is no byte code instruction using 1024 byte of assembler instructions, so no exploit can be developed targeting this weakness.

14.3 x86_64 32 bit integer overflow vulneribility

A similar vulneribility has been found for x86_64. But it can not be exploited by the same argument as above.

14.4 All architecture exception handler exploit

In CACAO there are special conventions for propagating the exception object during *stack unwinding*. A `ATHROW` instruction is implemented as follows: the pointer to the exception object and the faulting program counter are placed into scratch registers `itmp1` and `itmp2` respectively and an assembly language function, `asm_handle_exception` is jumped to that performs *stack unwinding*. The program counter and exception type are then used to find an exception handler block which is jumped to. The handler code expects the register `itmp1` to contain the exception object pointer. This approach makes use of the assumption that the only way to reach an exception handler is via the *stack unwinding* process. This is actually always true for compiler generated bytecode but at bytecode level it is perfectly leagal to directly jump into an exception handler block without an exception thrown. The exception handler code then interprets the contents of the scratch register `itmp1` as exception pointer. Because `itmp1` is used in arithmetic operations as scratch register, it contents can easily be controlled and set to an arbitrary value.

To exploit this vulnerability a virtual method on this arbitrary object pointer is going to be invoked. When calling an object's Nth virtual method, first the pointer to the *virtual function table* is loaded from offset 0 of the object pointer. Then, the method's entry point is loaded from slot N of the virtual function table. Finally, the method's entry point is jumped to.

Using arrays, a fake object and a fake virtual function table with all entries pointing to shell code are constcutred as shown in the source code in figure 5. To set up the pointers in the arrays a method is needed to get the address of the first element of a java array. This can easealy be achieved by abusing of the default `toString()` implementation which outputs a string containing the object's class name and its address in memory. In cacao's implementation, an array starts with a fixed-sized header followed by data elements, so the address of element 0 is calculated by adding a fixed offset to the array pointer. Now if a virtual function on this fake object is called, control is passed to the shell code.

```
int addressOf(Object o) {
        // extract and return address from o.toString()
}

// Architecture dependent size of array header
// First array element is at this offset from array pointer
int arrayHeaderSize = 16;
// Shell code
byte[] code = { /* shell code, null bytes allowed*/ };
// Virtual function table with 100 slots
// Each element (method entry) points to the shell code
int[] vftbl = new int[100];
for (int i = 0; i < vftbl.length; ++i)
        vftbl[i] = addressOf(code) + arrayHeaderSize;
// Object, first words points to virtual function table
int[] obj = new int[1] { addressOf(vftbl) + arrayHeaderSize);
// Object pointer has to point to element 0 of obj
int objPtr = addressOf(obj) + arrayHeaderSize;
```

Figure 5: Constructing a fake java object

14.5 16 and 12 bit invoke virtual integer overflow on PPC32 and S390 exploit

As described in section 14.4, to call a virtual method, two loads are involved: the load of the virtual function table, and then the load of the method entry from a specific slot of the virtual function table. The displacement of a load instruction has a limited range: on i386 and x86_64 it is limited to 32 bits, on ppc to 16 bits, on s390 to 12 bits. If the load of the method entry is implemented as a single load instruction, the maximal load displacement limits the number of virtual methods that can be supported by such a design: $2^{31}/4$ on i386, $2^{31}/8$ on x86_64, 8192 on powerpc and 4096 on s390. The question is, what happens if a class happens to contain more virtual methods? On most achitectures, this case is protected by an assertion. If assertions are turned off, the displacement of the load will just be trimmed to fit into the maximal displacement bitsize. That in turn means that, if we call a virtual method who's entry fails to get loaded because of the displacement limitation, a different method will be called.

To exploit this vulnerability, let's suppose the displacement in the load instruction is unsigned, and that it can be used to load a maximum of MAX methods from the virtual function table. A class with MAX virtual methods is generated, each taking one word sized integer as argument and just returning that argument followed by two methods with the signatures `Object intToObject(int i)` and `int objectToInt(Object o)`. If `objectToInt` is called, its entry should be loaded from slot $MAX + 1$ of the virtual function table but after trimming the offset, the entry will be loaded from slot

1 instead, where a method resides that reinterprets the object reference as integer and just returns it. This way pointers can be converted to integers and vice versa, bypassing the type system.

Once this type unsafe "casting" functions are available a fake object is constructed like in section 14.4 with `objectToInt` used to get the addresses of the arrays and `intToObject` used to "cast" the address of the fake object to an `Object`. If calling some virtual method on this object pointer, controll is passed to the shell code.

Hacking
lecture
2007-12-28 12:45
Saal 3
de

Stefan Strigler
BeF

Konzeptionelle Einführung in Erlang

A jump-start into the world of concurrent programming

Originally developed by Ericson, Erlang was eventually released as open source in 1998. Although Erlang has been around for almost ten years now, it became a rather popular programming environment for communication platforms only recently.The talk will equip the open-minded programmer with concepts of concurrent programming in a functional programming environment supported by real-world examples.Despite the fact that actual code fragments will be in display, there is no need for novices and non-programmers to be scared away.

Konzeptionelle Einführung in Erlang

24C3

Ben Fuhrmannek <ben@fuhrmannek.de>
Stefan Strigler <steve@zeank.in-berlin.de>

Ziel des Vortrags ist es, einen kleinen Einblick in Erlang/OTP zu gewähren, allerdings weniger in der Form "Wie programmiere ich was mit Erlang?" als eher eine Antwort auf Fragen zu liefern wie "Was macht Erlang besonders, was kann es was andere Sprachen nicht oder nicht so gut können?". Es soll mehr um den Einsatz von Erlang in der Praxis gehen, als eine Einführung in das Arbeiten mit Erlang zu geben (sorry, kein 'Hello World' today).

HISTORIE

Erlang was created by the Computer Science Laboratory at Ellemtel (now Ericsson AB) around 1990. It originates from an attempt to find the most suitable programming language for telecom applications. Characteristics for such an application include:

- *Concurrency - Several thousand events, such as phone calls, happening simultaneously.*
- *Robustness - An error in one part of the application must be caught and handled so that it does not interrupt other parts of the applications. Preferably, there should be no errors at all.*
- *Distribution - The system must be distributed over several computers, either due to the inherent nature of the application, or for robustness or efficiency.*

(Quelle: http://www.ericsson.com/technology/opensource/erlang/)

Open Source ist Erlang seit 1998. Die Sprache wurde nach dem dänischen Mathematiker Agner Krarup Erlang benannt, wobei die Doppeldeutigkeit mit Ericson-Language (ErLang) gewollt ist.

PROZESSORIENTIERTE PROGRAMMIERUNG

Joe Armstrong: "The world is parallel."

In Erlang besteht die Welt aus Prozessen, die mit einander Nachrichten austauschen. Dieses Konzept ist für uns sehr leicht zu verstehen, denn wir agieren auf ähnliche Weise: Eine Ampel signalisiert grün, dann fahren wir los. Oder wir fragen die Auskunft nach einer Telefonnummer und sie wird uns genannt. Jede Person und jedes Objekt, das irgendwie interagieren möchte, wird so einfach als Prozess abgebildet. Eine kleine Erweiterung zur Realität stellt die Tatsache dar, dass Prozesse, die sich erwartet oder unerwartet beenden, noch die Ursache preisgeben; z.B. eine Ampel fällt aus, dann sagt sie als Letztes noch 'Glühbirne durchgebrannt'. Falls ein anderer Prozess sich dafür interessiert, dann kann die Ampel passend repariert werden.

In der objektorientierten Entwicklung werden Daten als Objekte und Abläufe als Use-Cases mit Methodenaufrufen von Objekten modelliert. In aktuellen Diskussionen wird das leider allzu oft als Gegensatz aufgegriffen, was wohl daher rührt, dass klassische objekt-orientierte Sprachen Parallelisierung nur mittels Threads unterstützen. Erlang dagegen aber keine Klassen und Objekte kennt. Im Prinzip widersprechen sich die Ansätze aber nicht. So lassen sich Prozesse auch als Objekte begreifen. In Python werden Methodenaufrufe sowieso Nachrichten genannt und sind ohnehin von jeher konzeptionell dasselbe.

Threads teilen Speicher miteinander, dessen Zugriff zum Schutz vor Inkonsistenzen mit Locks abgesichert wird. Sollte während eines bestehenden Locks ein Fehler auftreten, muss explizit sichergestellt werden, dass das Lock wieder freigegeben wird, ansonsten wäre der Programmablauf beim nächsten Zugriff auf das Lock gestoppt.

Erlang dagegen kennt keinen Shared-Memory und keinen globalen Variablen, sondern Prozesse kommunizieren über Nachrichten.

SPRACHLICHE BESONDERHEITEN

- Erlang ist eine sequentiell[1] funktionale[2] Programmiersprache.
- Variablen können nur einmal assoziiert werden, z.B.

```
X = 1.
X = 2 (ERROR)
```

[1] sequentiell: a, b, c

[2] funktional: f(e(d()))

und müssen vorher nicht deklariert werden. Es gibt keine globalen Variablen und keinen von mehreren Prozessen gemeinsam genutzten Speicher.

- Die nahezu platformunabhängige Laufzeitumgebung (footnote: läuft unter Linux, ...) interpretiert Byte-Code.
- Anstatt Threads gibt es Prozesse, die von der Laufzeitumgebung verwaltet werden und daher sehr leichtgewichtig (footnote: sowohl RAM als auch Startdauer) sind.
- Inter-Process-Communication (IPC) ist sehr einfach durch asynchrone Nachrichten abbildbar, z.B.

```
Pid ! nachricht.
```

- Dabei stellt Pid eine Prozess-ID dar, die in einem verteilten System auch auf einen anderen Erlang-Node verweisen kann.
- Erlang unterstützt Hot-Code-Replacement.

ERLANG OTP (OPEN TELECOM PLATFORM)

Äquivalent zu den Standardbibliotheken in anderen Programmiersprachen bietet Erlang die Open Telecom Platform:

- große Bibliotheksklassen für den Programmiereralltag
- integrierte Anwendungen wie Mnesia (Verteiltes Datenbanksystem)
- vordefinierte Archtitekturmuster wie gen_server für Client-Server Architekturen oder gen_fsm für endliche Automaten
- Debugging- und Deployment-Tools

WAS KANN ERLANG FÜR DICH TUN?

Erlang zeigt sein volles Potential, wenn ein oder mehrere der folgenden Kriterien besonders wichtig sind:

Parallelisierung

z.B. typisch für Client-Server-Architektur und um Multi-Core-Systeme auslasten

Es folgt ein vergleichendes Beispiel mit vielen Prozessen/Threads mit Erlang, dann Python:

```
-module(processes).
-export([max/1]).

max(N) ->
     Max = erlang:system_info(process_limit),
     io:format("Max. processes: ~p~n", [Max]),
     statistics(runtime), statistics(wall_clock),
     L = for(1, N, fun() -> spawn(fun wait/0) end),
     {_, Time1} = statistics(runtime),
     {_, Time2} = statistics(wall_clock),
     lists:foreach(fun(Pid) -> Pid ! die end, L),
     U1 = Time1 * 1000 / N,
     U2 = Time2 * 1000 / N,
     io:format("time for ~p processes: ~p/~p (runtime/real)~n", [N,
U1, U2]).

wait() ->
     receive
          die -> void
     end.

for(N, N, F) -> [F()];
for(I, N, F) -> [F()|for(I, N-1, F)].

%% Beispiel aus 'Programming Erlang'
```

output:

```
1> processes:max(32000).
Max. processes: 32768
time for 32000 processes: 1.56250/3.71875 (runtime/real)
```

```
import sys,os
from threading import Thread, Lock

gl = Lock()
class TestThread(Thread):
     def run(self):
          gl.acquire()
          gl.release()

t1 = sum(os.times())

N = int(sys.argv[1])
threads = []
gl.acquire()
for i in range(N):
     t = TestThread()
     t.start()
     threads.append(t)

gl.release()
for t in threads:
     t.join()
t2 = sum(os.times())
print "elapsed cpu time: " + str(t2-t1) + "s"
```

4

Skalierbarkeit durch Verteilheit (Cluster)

Verfügbarkeit durch Fehlertoleranz und Hot-Code-Replacement

99,999% Verfügbarkeit

KILLER-APPLICATIONS

Ejabberd

- High-Performance Jabber/XMPP-Server,
- clusterbar,
- Komponenten für JUD, Groupchat, IRC und PubSub integriert,
- Web-Administration,
- Leicht erweiterbar durch Erlang-Module (ejabberd-modules)
- In-House Benchmarks: Ein Node auf dual Xeon 2.8GHz und 8GB Ram bedient ca. 150.000 c2s Connections.
- MXit Südafrika betreibt Ejabberd-Cluster mit 4.8M registrierten User, 9M logins und 200M pro Tag.

Tsung

- Benchmark-Tool für HTTP und XMPP
- Clusterbar

Yaws

- High-performance Webserver für dynamischen generiertent Content
- embedable

KRITIK

- Useability der Dokumentation nicht auf der Höhe der Zeit - wer mit manpages umgehen kann, kommt aber gut zurecht
- Community noch etwas unorganisiert
- Für Fragen, Hilfe, Support existiert (nur?) eine Mailingliste mit mittlerweile doch sehr hohem Traffic. Dort schreiben aber eben auch Leute aus dem Ericsson Entwicklerteam sowie Joe Armstrong selbst.

GETTING STARTED

- Download und Doku unter [http://www.erlang.org http://www.erlang.org]
- Community-Site: [http://www.trapexit.org Trapexit]

LITERATUR

- Joe Armstrong, Robert Virding, Cleas Wikström, Mike Williams: Concurrent Programming in Erlang, Second Edition, Prentice Hall, 1996
- Joe Armstrong: Programming Erlang - Software for a Concurrent World, The Programatic Programmers, 2007
- http://www.thinkingparallel.com/2007/03/20/ten-questions-with-joe-armstrong-about-parallel-programming-and-erlang/ Ten Questions with Joe Armstrong about Parallel Programming and Erlang
- http://armstrongonsoftware.blogspot.com/2006/08/concurrency-is-easy.html Concurrency is easy
- http://armstrongonsoftware.blogspot.com/2006/09/why-i-dont-like-shared-memory.html Why I don't like shared memory
- http://armstrongonsoftware.blogspot.com/2006/09/pure-and-simple-transaction-memories.html Pure and simple transaction memories
- http://weblogs.mozillazine.org/roadmap/archives/2007/02/threads_suck.html Threads suck
- http://en.wikipedia.org/wiki/Erlang_%28programming_language%29 Wikipedia: Erlang (programming language)
- http://de.wikipedia.org/wiki/Erlang_%28Programmiersprache%29 Wikipedia (de): Erlang (Programmiersprache)
- http://en.wikipedia.org/wiki/Declarative_programming Wikipedia: Declarative programming
- http://en.wikipedia.org/wiki/Functional_programming Wikipedia: Functional programming
- http://lambda-the-ultimate.org/node/2533 Generative Code Specialisation for High-Performance Monte Carlo Simulations

Science
lecture
2007-12-28 16:00
Saal 2
en

Martin 'maha" Haase

Linguistic Hacking

How to know what a text in an unknown language is about?

It is sometimes necessary to know what a text is about, even it is written in a language you don't know. This can be quite problematic, if you do not even know in what language it is written. This talk will show how it is possible to identify the language of a written text and get at least some information about the contents, in order to decide whether a specialist and which specialist is needed to know more.

The talk deals with the following issues:1 How to identify a language* texts in non-latin writing systems and how the writing system can show what language we deal with,* how to identify languages with the help of sample texts (based on a collection of sample texts compiled for this purpose by Soviet linguists will be used),* tricks that help to make at least an intelligent guess.2 How to get an idea about the contents of a text* identifying (important) content words and grammar,* quick and dirty translations,* how to translate a text from a language you hardly know.The talk will introduce a variety of means, ranging from pre-internet (and pre-computational) approaches to contemporary web resources.

Linguistic Hacking
How to know what a text in an unknown language is about?

Martin.Haase@uni-bamberg.de

24th Chaos Communication Congress

It is sometimes necessary to know what a text is about, even it is written in a language you don't know. This can be quite problematic, especially if you do not even know in what language it is written. This talk will show how it is possible to identify the language of a written text and get at least some information about the contents, in order to decide whether a specialist and which specialist is needed to know more.

1 Introduction

In a first and rather brief outline, I will show how to identify the language of a written text in traditional ways and with the help of computer technology. In the second part, I will show how to get at least some information out of an unknown text. This is all about linguistics, but what has it to do with hacking? I will show that some tricks must be used to solve such problems and define hacking in this context according to Eric Raymond's seventh definition as "the intellectual challenge of creatively overcoming or circumventing limitations." [10, 234]

I will confine my analysis to written texts (not necessarily in Roman script), although, based on a multi-language corpus of telephone calls [7], considerable progress has been made in the identification of spoken languages [8]. The main reason for this omission is that with spoken language it is far more difficult (and perhaps even impossible) to get clues about the contents of a conversation without at least some knowledge of the language in question.

2 How to identify a language

2.1 The traditional approach

If the text comes in a non-Roman and non-Cyrillic writing system, it is in most cases quite easy to identify the script and the language, because exotic scripts are often language-

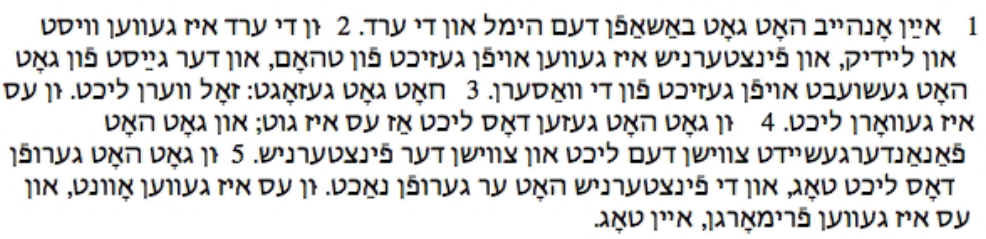

1 איין אָנהייב האָט גאָט באַשאַפֿן דעם הימל און די ערד. 2 ון די ערד איז געווען וויסט
און ליידיק, און פֿינצטערניש איז געווען אויפֿן געזיכט פֿון טהאָם, און דער גייסט פֿון גאָט
האָט געשוועבט אויפֿן געזיכט פֿון די וואַסערן. 3 חאָט גאָט געזאָגט: זאָל ווערן ליכט. ון עס
איז געווארן ליכט. 4 ון גאָט האָט געזען דאָס ליכט אַז עס איז גוט; און גאָט האָט
פֿאַנאַנדערגעשיידט צווישן דעם ליכט און צווישן דער פֿינצטערניש. 5 ון גאָט האָט גערופֿן
דאָס ליכט טאָג, און די פֿינצטערניש האָט ער גערופֿן נאַכט. ון עס איז געווען אָוונט, און
עס איז געווען פֿרימאָרגן, איין טאָג.

Figure 1: Beginning of Genesis in Yiddish

specific. A handbook on writing systems [4] or web resources [1] can easily help to identify a script and thereby the language.

There are some difficult cases of course. One such case is the Hebrew script which is used for:

- Old and Modern Hebrew,
- Ladino (with different varieties),
- Judeo-Arabic,
- Yiddish

Of course, there are some simple tricks to distinguish between Hebrew and the other languages. Normally, Hebrew is written without vowel diacritics (the little dots over and under Hebrew letters). If your text shows no such signs, it is probably Hebrew. If it contains such “vocalization signs”, it may still be Hebrew (a text from the Bible, from a children’s book, or from learning material), but in that case the vocalization can be consistently found throughout the text. If some words show (some) vocalization and others don’t, it is most probably a Yiddish text, where Yiddish words contain a subset of vocalization signs, but loan words from Hebrew are used without vocalization. Ladino doesn’t contain super- or subscript diacritics at all. Moreover, Yiddish and Ladino texts may contain Roman-script arabic numbers and Roman-script punctuation signs, but sometimes even Hebrew texts contain western numbers. Figure 1 shows a Yiddish text (few vocalization, Roman-script arabic numbers, Western punctuation), whereas figure 2 shows the same text from the Hebrew bible (with full vocalization), i. e. the beginning of Genesis, the first book of the Bible (Hebrew numbering, full vocalization, non-Western punctuation).

The problem gets worse when we turn to the Arabic writing systems. Variants are used for about twenty different and partly unrelated languages (and more subvarieties) and Modern Arabic itself has about thirty commonly used varieties. In order to get an idea about the language, it is helpful to work with sample texts [1, 6].

The Cyrillic writing system is even worse, since it is used for more than sixty languages. Cyrillic writing systems for non-slavic languages were conceived mainly in the

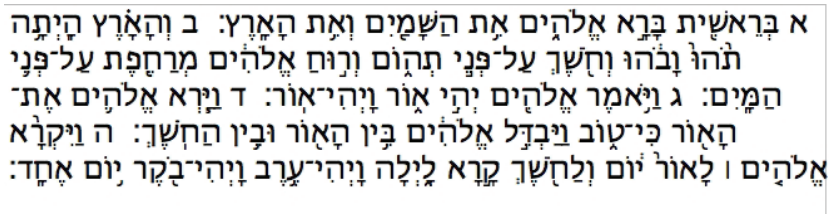


Figure 2: Beginning of Genesis in Biblical Hebrew

middle of the 20th century. When Cyrillic was adapted to different phonological systems, additional letters were introduced that make it easy to identify a language, because every writing system contains different special signs. That is why the identification of Cyrillic languages is mainly done through the identification of character encoding.

2.2 Computer-aided language identification

There are three common techniques [11]:

1. frequencies of unique characters and character strings: this method, known from cryptoanalysis, classifies documents by the frequency of unique characters and the occurrence of typical character strings; a nifty variant of this approach consists in measuring the compression efficiency that a program such as *gzip* achieves when appending an unknown document to various reference documents. [3]

2. common words recognition: this method is based on word frequency lists (generated from sample texts), the unknown text is analyzed word by word and compared to the list of the top 100 words (or so) of the sample texts;

3. n-gram analysis: this method works like common words recognition with the difference that (instead of words) sequences of n characters are used (2-character sequences, 3-character sequences, etc.): if we split the word *text* into 3-grams, this would be the result: `(_TE), (TEX), (EXT), (XT_)`, _ denoting the word boundary.

These approaches all work according to the scheme in Figure 3: a document model is generated from the input text in the unknown language and then this model is compared to the existing models generated from sample texts.

The advantages and shortcomings of this procedure can be critically evaluated [5]: the main drawbacks are that only a closed class of languages can be identified (dialects and varieties of these languages are usually ignored), and normally, multilingual text cannot be processed. If the programs work for non-Roman scripts, they usually reduce the recognition of non-Roman script languages to the detection of the encoding which doesn't work if a writing system is used for several languages and if non-standard or mixed character encodings are used.

Here is a list of free software readily available (and running) on the internet [5, 12, 13]:

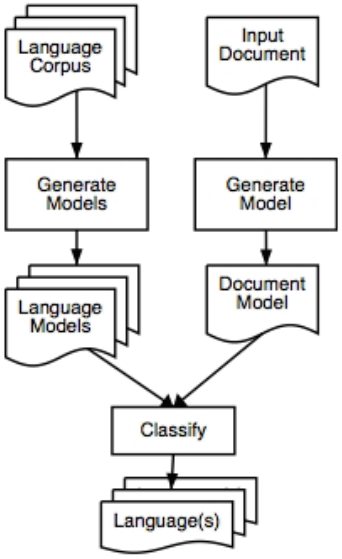


Figure 3: Language Identification Workflow [9]

- TextCat (`http://odur.let.rug.nl/vannoord/TextCat/Demo/`), an n-gram based identification tool for 76 languages, usable as a web application,
- Languid (`http://languid.cantbedone.org/`), a downloadable program, the web application is not running properly,
- Langid (`http://complingone.georgetown.edu/~langid/`), a web-based identification tool for 65 languages, based on n-gram analysis,
- LanguageGuesser (`http://www.xrce.xerox.com/cgi-bin/mltt/LanguageGuesser`) provides for the web-based identification of about 40 languages, based on statistical methods (frequency tests on characters and character sequences) [2],
- Polyglot 3000 (`http://www.polyglot3000.com/`), closed-source Windows freeware, identifying currently 441 languages, corpora and method are unknown.

3 How to get an idea about the contents of a text?

When we have identified the language of the text, it would be helpful to get an idea of its contents before we try and find a specialist who can help us with the translation. Perhaps the text is not interesting at all or has been translated before.

A hacker's approach to this task could be as follows:

- look for things you recognize without any help: numbers, dates, words from another language; a number or a date can be a good hint; if it is a precise number or date, a quick look-up with your preferred search engine might be helpful,
- look for typographic hints to important content: bold or italic print, colored or underlined text chunks, capital letters (they may indicate names that you may recognize or look up in Wikipedia).

Even with these steps you can get important hints about the contents of the text.

Moreover, the principle of least effort or Zipf's law [14] can be very helpful to find out what a text is about: Very frequent words are shorter and contain less lexical information, whereas infrequent words are longer and contain more lexical information; moreover, less lexical information implies more grammatical information and vice versa. For our purpose, we are looking for words with more specific lexical information. So we can ignore all short words, even if they reiterate throughout the text. A longer word that is repeated is therefore more interesting. gagana Here is an example (from Samoan, which is difficult to identify as such, since it is not contained in typical language sample collections):

Ua salalau lenei *gagana* i le lalolagi atoa. 'O lenei fo'i *gagana*, 'ua 'avea ma *gagana* lona lua a le tele o tagata 'o le vasa Pasefika, e pei 'o Samoa. E iai le manatu, 'o le *gagana* **fa'aperetania**, 'ua matuā talitonu i ai le tele o tagata Samoa e fa'apea 'o le *gagana* e maua ai le atamai ma le poto. 'E talitonu fo'i nisi o i latou, 'e lē aoga la latou *gagana*. E lē sa'o lea tāofi, 'auā e 'avatu le *gagana* **fa'aperetania** i Samoa, 'ua leva ona atamamai ma popoto tagata Samoa e fai lo latou soifua ma lo latou lalolagi.

The interesting words in this text are *gagana* and **fa'aperetania**, perhaps latou too, although this is short enough to be a more grammatical item. It is difficult to find a Samoan dictionary, but a quick search reveals that **fa'aperetania** means 'English' (8th Google result) and *gagana* 'language' (11th & 13th Google hit); latou is more difficult to find and less useful, since it is a third person plural pronoun (as the French Wiktionary reveals). So the text is about the English language, probably in Samoa ("*gagana* **fa'aperetania** i Samoa").

The example shows that it is rather simple to get at least minimal information out of a text whose language is unknown to us, even if we don't have direct access to a translator or a dictionary.

References

[1] Omniglot. Writing Systems and Languages of the World. `http://www.omniglot.com/` (2007-11-16).

[2] K.R. Beesley. Language identifier: A computer program for automatic natural-language identification of on-line text. *Language at Crossroads: Proceedings of the 29th Annual Conference of the American Translators Association*, pages 12–16, 1988.

[3] D. Benedetto, E. Caglioti, and V. Loreto. Language Trees and Zipping. *Physical Review Letters*, 88(4):48702, 2002.

[4] P.T. Daniels and W. Bright. *The world's writing systems*. New York etc.: Oxford University Press, 1996.

[5] B. Hughes, T. Baldwin, S. Bird, J. Nicholson, and A. MacKinlay. Reconsidering Language Identification for Written Language Resources. *eprints: http://eprints.infodiv.unimelb.edu.au/archive/00001744 (2007-11-16)*.

[6] N.C. Ingle. *Language Identification Table*. London: Technical Translation International, 1980.

[7] Y.K. Muthusamy, R.A. Cole, and B.T. Oshika. The OGI multi-language telephone speech corpus. *Proceedings of the International Conference on Spoken Language Processing*, pages 895–898, 1992.

[8] Y.K. Muthusamy and A.L. Spitz. Automatic language identification. *Cambridge Studies In Natural Language Processing Series*, pages 273–276, 1997.

[9] A. Poutsma. Applying Monte Carlo Techniques to Language Identification. *Language and Computers*, 45(1):179–189, 2002.

[10] E.S. Raymond. *The New Hacker's Dictionary*. Cambridge, Mass.: MIT Press, 1996.

[11] C. Souter, G. Churcher, J. Hayes, J. Hughes, and S. Johnson. Natural Language Identification Using Corpus-Based Models. *Hermes Journal of Linguistics*, 13(S 183):203, 1994.

[12] G. van Noorden. Language Identification Tools. `http://www.let.rug.nl/~vannoord/TextCat/competitors.html` (2007-11-16).

[13] Wikipedia. Language Identification. `http://en.wikipedia.org/w/index.php?title=Language_identification&oldid=139087517`.

[14] G.K. Zipf. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. New York: Hafner, 1965.

Science
lecture
2007-12-28 18:30
Saal 3
en

Florian

Modelling Infectious Diseases in Virtual Realities

The "corrupted blood" plague of WoW from an epidemiological perspective

World of Warcraft is currently one of the most successful and complex virtual realities. Apart from gaming, it simulates personality types, social structures and a whole range of group dynamics.

In 2005, courtesy of its creators at Blizzard Entertainment, the ancient Blood God "Hakkar the Soulflayer" unleashed a devastating plague, "corrupted blood", upon a totally unprepared population of avatars. Unintentionally, the digital "black death" spread to cities and depopulated whole areas. The epidemic could only be controlled by shutting down and restarting the game world, a measure unfortunately not available in the "real" world. However, other measures such as quarantine or improved treatment are available in the real world and can be simulated by disease modelling. Disease modelling is essentially a virtualisation of reality that tries to gain insights into hitherto unknown inderdependencies and to simulate intervention scenarios.I will give a brief overview of the use of infectious disease modelling in a population and explain the disease dynamics of the "corrupted blood" epidemic in WoW. I will focus on cross references to the "real

http://www.burckhardt.de/24c3_modelling_infdis_in_vr.pdf conference talk

24C3: Modelling Infectious Diseases in Virtual Realities
The „Corrupted Blood" plague of World of Warcraft TM from an epidemiological perspective
by Florian Burckhardt, MSc Epidemiology

I will begin with a brief introduction to modelling diseases, describe how I modelled the „corrupted blood" plague of the online game World of Warcraft and finish with a few ideas on future virtual epidemics.

Epidemiological modelling primer

SIR model

Epidemiology is the study of the pattern of disease in time, place and population. Very often, the goal is to identify the underlying causative factors of disease. One of the early epidemiological successes was the discovery by John Snow of contaminated water pipes as the underlying cause for the great London Cholera epidemic in 1854. Another well known example is the link between smoking and lung cancer.
Infectious diseases as opposed to chronic diseases are somewhat unique in epidemiology because exposure and outcome are the same: an infected person (or animal in case of zoonoses). This leads to non-linear dynamics that make analysis and prediction of infections in a population very challenging.
One approach is to simulate the epidemic in a mathematical model that describes the relationship between sick and healthy people in order to test different interventions.
There are many ways to design a model. Individual or agent based systems allow for single individuals with their distinct characteristics like age, sex, contact pattern, risk taking and healthcare seeking behaviour, etc. These "agents" are then put into a simulation and the spread of disease within the population of agents is observed. Of course, all system parameters have to estimated from real world data, which can be very difficult or in the words of J. Maynard Smith: „Describing complex, poorly-understood reality with a complex, poorly understood model is not progress".
Another modelling paradigm are compartimental models which divide the population into distinct compartments of susceptible to disease (S), infectious (I) and recovered (R), where recovered are considered to have acquired immunity. These SIR models (Kermack-McKendrick 1927) assume homogenous mixing within the compartments, i.e. they imply that all susceptibles have the same probability to meet infectious. This assumption is like most other modelling assumption always wrong, but what matters is the strength of violation. In most cases, the SIR model and its variants are adequate.

The challenge with a SIR model is to estimate the flow between different compartments, most notably between S(usceptibles) and I(nfetious), which will be explained in more detail. For simplicity, birth rate and natural death rate are ignored (closed population).
Assuming homogenous mixing, the overall contact rate is `c`. Since we are only interested in contacting infectious, we multiply with the proportion of infected `I/N` (where N=S+I+R = total population).
However, meeting with an infectious does not always result in an infection event. This only happens with a transmission probability `p`. For tuberculosis for example, one would have to meet approximately 20 infectious people before contracting the disease whereas measles or Ebola have a transmission probability close to one. The term `p*c` is also called „beta" or "force of infection".
So far, we have `p*c*I/N` which corresponds to the rate of transmission from infectious. The total transmission rate in a population is the number of susceptibles `S` multiplied by that rate, finally yielding `p*c*I/N*S`. N, p, c are constants, S and I are state variables and change with time, making the whole system non-linear as mentioned above.
The "flow" from compartment I to R is simply the inverse of the duration of infectiousness (D), usually called delta. For example, if one remains infectious for 10 days (D=10) and time is counted in days, then 1/10 per day (1/D) of I flows to R. However, compartment I also looses individuals due to death at the disease specific death rate sigma. Here, sigma is set to zero.
Summing up, compartment S "looses" individuals at a rate of `p*c*I/N*S`, compartment I gains individuals at that rate but looses individuals at rate delta to compartment R. Compartment R gains individuals at rate delta.
These rates are put into a system of differential equations which are solved numerically by computer programs such as Berkeley Madonna (http://www.berkeleymadonna.com/).
In formula (dS/dt means change of S over time, no birth rate, no natural or disease specific death rate):

```
dS/dt = -p*c*I/N*S
dI/dt = p*c*I/N*S - delta*I
dR/dt = delta*I
```

The SIR model is suited for infections that generate immunity (R compartment). If immunity is lost with time, one would use a SIRS model where the „waning immunity“ rate would determine the „flow“ from compartment R to compartment S back again.
Most sexually transmitted infections such as syphilis, gonorrhoea or chlamydiasis but also the „winter vomiting disease“ caused by Norovirus generate no or only partial immunity. S(usceptible) become I(nfectious) and after curing the infection S(usceptible) again, resulting in a SIS model. Diseases such as Hepatitis C or HIV (!condoms protect!) cannot be cured and leaves people I(nfectious), yielding a SI model.

The basic reproductive number R0

R0 („R naught“, „R zero“) is the average number of secondary infections from one single infected in a totally susceptible population. This is the same as asking: „how many people does one infectious person infect if everybody is susceptible ?“. If R0 is below one, the epidemic dies out.
R0 is the product of mean duration of infectiousness (D), contact rate (c) and transmission probability (p):

```
R0 = D*c*p
```

The concept of R0 allows to assess the impact of different epidemic interventions. Quarantine for example reduces the contact rate whereas treatment would act on duration of disease and/or transmission probability. Tamiflu for influenza e.g. shortens period of infectivity (D) and inhibits viral shedding (p). Wearing face masks would inhibit spread of airborne infections (reduce p) and rigid hand hygiene would greatly reduce any fecal oral transmission (reduce p).
Sometimes, interventions or social customs can also increase R0. If an intervention prolongs duration of disease or increases p, the epidemic gets worse. For example, in the beginning of the SARS epidemic, patients were treated with steam-nebulisers to ease breathing. However, additional aerosolisation of airborne infections is really the last thing you need during an epidemic.

Corrupted Blood

Hakkar the Soulflayer

On September the 13th, Blizzard Entertainment released new gaming content for their acclaimed massively multiplayer online roleplaying game, „World of Warcraft“ (WoW). For the sake of brevity, basic knowledge about WoW is assumed.
A new map region called „Zul Gurub“ with a new challenging end-game opponent „Hakkar the Soulflayer“ were waiting for high level players. During battle, Hakkar cast a spell called „corrupted blood“ (CB) on a random player that hit with severe damage once and additional smaller damage over time (DOT). DOT-spell are not uncommon in Wow, however totally new was the ability of the spell to get „transmitted“ to nearby players and their „pets“ (fighting companions). The spell was infectious. The original intention of the game designers might have been to force players to spread over an area and thus let the infection run out by eliminating contact between players. What happened was that once infected player teleported back to populated cities or hunters (special classes) summoned back their infected pets, CB spread like the famous black death and depopulated whole areas. Worse still, non player characters like in-game shopkeepers or guards got infected as well. The game designers first tried to quarantine the disease but ultimately failed and had to shut down the virtual world and reload it with a non-infectious version of CB. The CB-incident caught a lot of media attention and fuelled discussion on using online games as epidemic simulators.

Modelling CB

First, it has to be said that any epidemiological modeller could have predicted the devastating effects of CB. The basic reproductive rate R0 was so absurdly high, that any natural pathogen would have killed its host population and thereby sealed its own fate: no host, no pathogen.
Model parameters usually have to be estimated from observational data. To the great dismay of the epidemiological community, no observational data on CB incidence is available from Blizzard. However, with a programmed disease like CB, parameters are available directly. Duration of the disease, providing survival, was 10 seconds. Low and mid level players died after two hits by the disease that was 4 seconds. Transmission probability was one, that is everyone in vicinity of an infectious got infected as well. Not even

Ebola is that contagious. Contact rate depended on geographic location. In special WoW meeting places in cities like the auction house, a contact rate of 5 players per second is not uncommon. Outside cities, contact rate was lower.

Low/Mid Level Avatars
Death in WoW is non-permanent: killed players become ghosts on a graveyard and can eventually resurrect later. In terms of modelling this translates into a SIRS model for low-mid level players: S(usceptibles) become I(nfectious) and by „dying" enter the R(ecovered) compartment, only to „resurrect" and become S(usceblible) again (fig. 1).

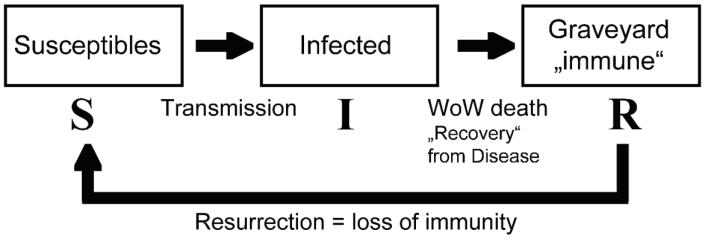


Figure 1: SIRS model

It might seem confusing to think of dead players as recovered, but in terms of disease modelling, they cannot be infected while on the graveyard and are thus, for the sake of CB, recovered.

The graphs in fig. 2 illustrate the course of the epidemic with different contact rates.
A: one infected at start, contact rate 2/s, resulting in 85% of players wasting their subscription fee on the graveyard with a slightly diminished in-game experience.
B: 500 infected at start, contact rate 1/5s, epidemic dies out because of R0= D*c*p=4*1/5*1, which is <1. In words, each infected creates less than one secondary infection.

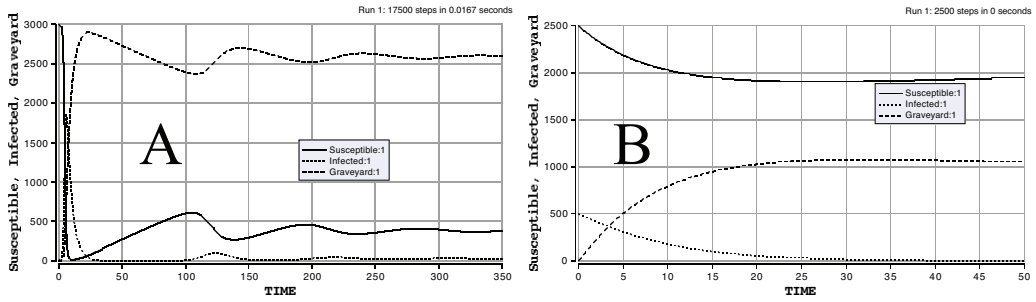


Figure 2: SIRS dynamics depending on contact rate. Susceptible black, infectious thin dotted, recovered thick dotted

High Level Avatars
High level avatars survive CB. They "bounce" back between S(usceptible) and I(nfectious) and are modelled using a SIS-model (fig. 3).

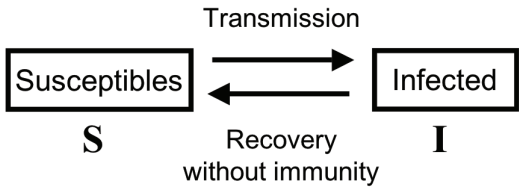


Figure 3: SIS model

The graphs in fig. 4 illustrate the course of the epidemic with different contact rates.
C: one infected at start, contact rate 2/s, resulting in 95% of players staying infectious.
D: 500 infected at start, contact rate 1/20s, epidemic dies out because of R0= D*c*p=10*1/20*1, which is <1 (D is 10 seconds and not 4 as in the SIRS cases A and B, as high level Avatars survive the full duration of the spell).

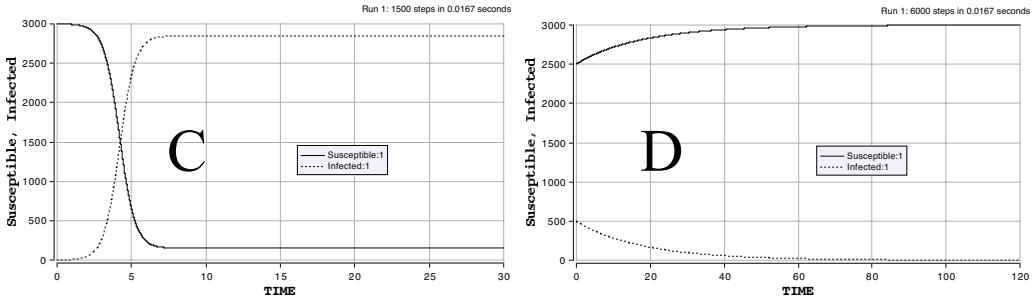


Figure 4: SIS dynamics depending on contact rate; susceptibles black, infectious dotted

Better virtual epidemics

Game designers should take a few cues from nature when introducing infections in virtual worlds. A transmission matrix with different transmission probabilities between races would allow more detailed modelling of interspecies infections (why should an orc-virus infect elves and vice versa?). Transmission could also depend on age and sex. And please note: transmission probability is never one, not even for Ebola or Measles.
Recovery could be made time dependent, i.e. avatars stay infectious for a random length of time.
Introduction of immunity would limit the devastating effects that were seen with CB. Immunity could gradually disappear thus simulating genetic changes in the infectious agent, which is seen with influenza. Immunity would also add the possibility of biological warfare, if eg. immune Alliance players including one infected would raid a susceptible orcish village. That strategy would mirror the distribution of smallpox contaminated blankets to Native American Indians in the 19th century. Immunity would also add vaccination as a service that might be synchronised with real-world flu-jabs.
Addition of an incubation period, where people are infected but not yet infectious, would more closely resemble real diseases.
Transmission routes could vary as well: food-borne, airborne (droplet infection) or injury just to name a few (with all those nasty cuts and flesh wounds in WoW, one wonders why there are not more wound infections...).
Online avatars are probably in no danger of sexually transmitted diseases any time soon.

Links & References

- Short course on epidemiology of infectious diseases, http://www.imperial.ac.uk/cpd/epidemiology/
- The untapped potential of virtual game worlds to shed light on real world epidemics, Lofgren ET, Fefferman NH, Lancet Infect Dis 2007; 7:625-29
- Berkeley Madonna, http://www.berkeleymadonna.com/
- Corrupted Blood, Wikipedia, accessed 16.11.2007, http://en.wikipedia.org/wiki/Corrupted_Blood
- Bapf the „Master Sergeant“
- presentation and paper available at http://www.burckhardt.de/docs.html

Society
lecture
2007-12-30 12:45
Saal 3
en

Olivier Cleynen

Overtaking Proprietary Software Without Writing Code

"a few rough insights on sharpening free software"

Free or "Open-Source" software, and in particular Linux, is doing extremely well technically. However, it fails to secure a significant portion of the protected, lucrative software market, especially for end-users. Can Free Software finally make a full entry into our society? The main obstacles to overcoming the domination of proprietary software, most of them non-technical, require thinking outside of code-writing. "Overtaking Proprietary Software

Pre-requisites are: A good understanding of the notion of Free/"open-source" Software and some of the main themes that surround it, such as DRM. There is no particular technical knowledge required.

Overtaking Proprietary Software Without Writing Code

Proceedings for the 24C3

Overview

This is a brief summary of a 45-min talk aimed at software developers, with the aim of giving rough essential insights on how to overcome proprietary software. The key idea is that it is necessary to *look away* from pure code writing, in order to strengthen free software enough that it overtakes proprietary (non-free) software.

Part I: market overview

A brief reminder that although free software outperforms proprietary products in many respects, it still remains a minor player in the market. We develop the most stable, trustworthy, usable software in the world, and yet we fail to get past the 1% mark almost everywhere.

Perhaps most telling is the success of Microsoft Vista, whose supposedly poor performance we love to describe. In the first month of sales, Microsoft sold 20 million units. That's more Vista sales in one month than there has been GNU/Linux users in ten years.

So it's possible that we lack something to make a difference, and clearly it's not “good software”.

Part II: Obstacles

If we are to make a difference we have to solve or get around four problems.

1. Nobody chooses software

This fact is often forgotten because we typically are people who care so much about software that we build our own. But in our society our consumer lives are getting so impossibly complicated (there is a decision to make for just any purchase, from potatoes to batteries) that by the time they come home in the evening people don't want to worry about software. We have to be already “inside” when Joe buys his computer.

2. We'll never have a killer app

Because of the nature of free software, ideas and code flow quickly and we typically will never have a killer application (they get ported too quickly). We continually forget about this, however, and keep trying to build it anyway (ie. trying to make *the perfect, ultimate unique* application).

3. The legal environment is hostile

This is summed up in one sentence: in most countries you cannot play MP3s and DVDs with free software, legally. The code is here but the patent/DRM laws prevent using it legally. Until this is changed, free software will never make it to the shelves of any large-scale store.

4. The OS is disappearing

Because online services are typically well-designed, practical and sexy, we are losing hold of the “real” operating system. There will always be software needed to run the PC chips, of course, but all of the *interesting* software, with which we exchange ideas, produce work, and build our culture, is all progressively being transferred to private servers. Just ask how many people in a room full of developers regularly use Google apps, and how many use proprietary-software-devices to access some kind of closed network (in their car, pockets, or living room).

Unless we put our focus out of personal-computer-centric software, we are at risk of missing this change in computing trends.

Part III: Fundamentals

Making a real difference in the market means “tackling Joe”, the everyday user who has better things to do than worry about the status of his software's code repository. Two points here:

1. Talk to Joe. The fact is our community is so much focused on software stability and choice, that we shut ourselves on an entirely different planet. Perhaps insisting more on usability, absence of viruses, and simple, easy choices (ie. killing Distrowatch) is the first thing to do.
2. Be relevant. Source code is the least of concerns for 95% of users out there. Speaking of “free software” instead of “open-source” makes much more sense and does make a big difference whenever the Joe has to make a decision.

Getting back to basics, speaking a language that is relevant to Joe, is the sole focus of *GNU/Linux Matters*, a non-profit which aims to explaining Linux and free software to 1 million people in 2008.

Part IV: Breakthrough

The goal of this section is to introduce some “business-thinking” into software development. Because our software is available at no cost, we fail to think in terms of *market*, *customer expectation*, or *segmentation*.

On the proprietary side, knowing exactly what the consumers want and how much they are ready to pay for it is a priority. The products then stem *from* this analysis (for example, the various Vista or Photoshop versions).

In the free software world... we are often simply too busy forking to worry about what the users want. This is because of *The v0.12 Syndrome*, whose symptoms are **1.** A total dedication to quality (“the bug tracker *is* the project”) **2.** An agenda driven by the progression of the software (instead of the opposite, ie, “it's released when it's ready”) **3.** An overwhelming tendency to fork (whenever somebody disagrees on how the code is written). The result: high quality, stable software that's perpetually in a v0.12 state, and ten miles of altitude separating developers from users.

We'll start to break through when we realize that quality never has been a decision factor for the end-user. For example, OpenOffice.org is bloated but seduced 100m users (and is a major player in opening standards) because of good market analysis: being *just like MS Office* was the requirement there. Similarly, the only difference between Firefox and the low-profile Mozilla suite was some wise market analysis – a few cuts and some branding, not better quality, has made all the difference.

Concluding remarks:

Making a lasting dent into the overwhelming domination of proprietary software in the market does not require writing better code. What we lack is better market analysis: a more *tactical* perspective in the development of our projects, and a focus on what the users want. Giving up quality to work on differentiation, and adapting to the online world are two of the biggest requisites for that.

Science
lecture
2007-12-27 12:45
Saal 3
en

Mark Vogelsberger

Simulating the Universe on Supercomputers

The evolution of cosmic structure

The evolution of structure in the Universe is one of the hottest topics in Cosmology and Astrophysics. In the last years the so-called Λ-CDM-model could be established also with great help of very large computer simulations. This model describes a Universe that consists mainly of dark components: 96% are made of dark energy and dark matter.

Ordinary matter made up of baryons give only 4% to the total content of the Universe. The talk will present recent results with the main focus on computational methods and challenges in that field. A state-of-the-art computer code for running these calculations will be presented in detail. The talk will describe recent progress in the field of cosmic structure formation and will mainly focus on computational problems and methods carrying out such large simulations on the fastest Supercomputers available today. At the end of the talk I will also briefly discuss a new method we developed to access the dark matter structure in the Milky way to a scale that was just impossible some month ago with current Supercomputers.To describe the evolution of the Universe from the Big Bang to what we see today is a quite hard task. [...]

http://www.mpa-garching.mpg.de/galform/presse/	Millennium Simulation done by the MPI for Astrophysics
http://www.ucolick.org/diemand/vl/	A recent NASA's Supercomputers Simulation
http://de.wikipedia.org/wiki/Millennium-Simulation	Wikipedia entry for the Millennium Simulation

Simulating the Universe on Supercomputers

Mark Vogelsberger, mark.vogelsbergerATemail.de

The following text is a very brief introduction into the field of cosmological Supercomputer simulations. Those who want to dig deeper into the field should consult the references at the end.

1 The Universe

The goal of cosmological simulations is to model the growth of the structures in the Universe. In other words, these simulations allow us to compress the long times of cosmic evolution into a human lifetime and they can be considered as an experimental tool to verify theories of the origin and the evolution of our Universe.

Today we believe that this evolution started with a Big Bang. Shortly after this event small fluctuations were imprinted into the radiation and matter density field. To understand the Universe, how it looks today, we need to know how these small perturbations to an otherwise homogeneous and isotropic space evolve with time. This calculation is highly complex and can only be done numerically using large computers. Analytic methods can only be used in the linear regime but for the whole evolution of the Universe numerical methods are needed. To run such cosmological simulations one needs two main ingredients: first it is necessary to specify initial conditions, to tell the computer where it should start to calculate. On the other hand one has to tell the computer also how to calculate the evolution of the Universe. The initial conditions for the simulation can be observed. How can we do this? We get the initial conditions from the afterglow of the Big Bang. About 300.000 years after the Big Bang the radiation could decouple. This radiation is still visible today. Due to the expansion of the Universe we can observe it today at an temperature of about 2.7 Kelvin. Modern satellite missions could resolve small fluctuations in this radiation. From these fluctuations it is possible to infer the perturbations in the initial density field of the matter. Thus we know how the initial density field 300.000 years after the Big Bang looked like. This is the input of our simulation. From this initial density field we have to evolve the Universe from the starting point to today, about 13 billion years after the Big Bang.

The leading force for this evolution is gravity in an expanding space. Cosmological codes use particles to trace the density field and evolve them under their mutual gravity. As the simulation samples the smooth density field with such a finite set of particles these computer simulations are called N-body codes. The more particles you have the better the resolution you get. This is why there is a constant competition in getting the highest number of particles and the computational resources you need to run these calculations require the largest computers available today. I will focus here on the simulation of the gravity only. This is by far the most important process and also the easiest thing to simulate. Note that there is also baryonic gas in the Universe - we are for example made out of baryons. Everything you can see like stars, galaxies, planets and so on are made of baryons. Their dynamics is also influenced by hydrodynamics and complicated gas physics. This is a lot more complicated to deal with. Modern simulation codes are also able to treat the baryons and compute a Universe with galaxies.

They allow to form stars and solve the gas physics. The cosmological code Gadget (Springel, 2005) that was developed at our institute is public available and can solve both gravity and hydrodynamics. This is still quite restricted, because there are lots of processes going on that need to be taken into account to get more realistic pictures: black holes, cosmic rays, radiative transfer, magnetic fields and so. The current internal production version of the Gadget code has more than 200 options corresponding to physical processes you can turn on or off. But the main evolution of cosmic structure does not need gas physics. It can purely be calculated using the gravitational force in an expanding Universe.

The fact that we can ignore the baryons for structure formation is because they only make up four percent of the total energy content in the Universe. The largest mass component comes from what is known as Dark Matter. It is called dark, because it does not shine like stars or gas. It is invisible and therefore called dark. Today we know that about 23 percent of the Universe are made up of this Dark Matter. Dark Matter only interacts by gravitation. This is why we can indirectly observe it by its gravitational interaction on visible objects like galaxies and gas. For example, Dark Matter can act as a gravitational lens and can deflect light from visible galaxies. Besides baryons and Dark Matter the largest component of the Universe consists of Dark Energy. In Einstein's equations of general relativity this corresponds to the so called cosmological constant. Due to the small fraction of baryons in the Universe most simulations of structure formation only take into account the dark components, so Dark Matter and Dark Energy. Based on physical models and assumptions galaxies, stars and gas can be added in a post processing by so called semi-analytic codes. These codes take the output of the N-body simulations and use physical laws to infer the baryonic physics. At the moment simulations start also to explore more and more the gas physics because the relevant codes are good enough and available machines are fast enough to simulate both gas and Dark Matter within one simulation.

Although we are very sure that there is Dark Energy and Dark Matter, we actually do not know what these main components of the Universe are made of. Dark Energy is very mysterious and for Dark Matter we have some particle candidates that are well motivated from particle physics. These are particles that are beyond the Standard Model of particle physics, like supersymmetric particles.

The fact that lots of structure formation simulations only take into account the dark components means, that the simulation particles represent the Dark Matter density field. Dark Matter behaves as a collisionless fluid and one needs to take some care to model this correctly. Therefore every particle in the simulation is not treated like a point source of a gravitational potential. The force is softened to avoid what is called two-body relaxation. This is needed to preserve the collisionless character of the Dark Matter fluid. One has to take into account one very important fact when representing the Dark Matter density distribution by a discrete set of particles. These particles are not real Dark Matter particles. Typical masses for some proposed Dark Matter particles are in the range of 100 GeV. The mass of the particles in the simulation are in the range of thousands of solar masses. It is totally impossible to simulate each Dark Matter particle on its own. So to speak the particle distribution of the Dark Matter fluid is only a Monte-Carlo representation.

After running the simulation its output can be statistically compared to observations. The important point is that both statistics show very good agreement. An agreement of those statistics then proves that our model of structure formation that we have put into the computer simulation is correct.

2 Some details

Gravity is the dominant force at large scales. At the beginning of the Universe there were small density perturbations. These were magnified by gravity during the evolution of the Universe. The main gravitational effect comes from Dark Matter, only at smaller galaxy like scales baryonic physics has to be taken into account. To simulate the Dark Matter one has to solve the equations for gravity in an expanding Universe. Normally the expansion is taken into account by a tricky time integration scheme and the coordinates in the simulation are so called comoving coordinates. These are the physical coordinates rescaled by the current size of the Universe. The main challenge for the force calculation lies in the long range $1/r^2$ character of the gravitational force. The long range character implies that every particle in the simulation feels every other particle. This results in N^2 force interactions. Typical particle numbers for cosmological simulations that are required, are too high to solve this N^2 problem. Without clever techniques to reduce the N^2 for these so called Particle-Particle methods (PP) it is therefore impossible to run such a simulation. The PP method only works for quite low number of particles. With special hardware it can also be used for higher number of particles. So called GRAPE chips are specially designed to calculate the gravitational force with an extreme speed. Using special hardware like this it is possible to use PP methods also with higher number of particles. But this is still by far not enough for cosmological structure formation applications.

A very common method to solve this problem is the Tree method. The idea is that the force of a distant group of particles can be approximated by the force of the center of mass force of that group. This approximation reduces the scaling of the number of calculations from N^2 to a lot better $N \log(N)$. The question is how to arrange the particles in an efficient way. A good way is the so called Tree method. For that the simulation volume is divided into smaller cubes with $1/8$ the volume each at every stage till the smallest cells have only one particle in them. The question for the force calculation is then whether to open a cell, or whether it is fine to take a whole group for the force calculation. Cells that are far away from the point of force evaluation do not have to be opened. Nearby groups need to be opened. To decide on whether to open or not is given by a so called acceptance criterion. This criterion in the end determines the force accuracy you get.

Another very popular method to calculate the gravitational forces are so called Particle-Mesh (PM) methods. In fact they were the first methods used to run larger cosmological simulations. These methods use the fact that the Poisson equation relevant for the gravitational forces is a simple algebraic equation in Fourier space. With a Fast Fourier Transformation (FFT) the forces can be calculated very fast. The FFT requires sampling functions at uniformly spaced points. A grid/mesh is used for this. In the simulation particles are used for representing the density and velocity field. This means that the density field at the mesh points has to be interpolated. The fact that both particles and meshes are used in the simulation gives this technique its name. The Fourier method has some advantages: it automatically implies periodic boundary conditions, softens the forces at small scales because of the mesh resolution and the FFT can easily be parallelised. These points are very important for cosmological simulations. But PM methods have also very critical disadvantages: the softening on mesh scales is very fine because softening is needed to simulate the collisionless Dark Matter fluid, but this also means the the PM code cannot resolve scales below the mesh scale. This is a very serious limitation of the dynamical range of PM simulations. An extension of classical PM methods are so called Adaptive Mesh Refinement (AMR) codes. In these methods the grid is refined in higher density regions. This way the resolution is increased where it is needed.

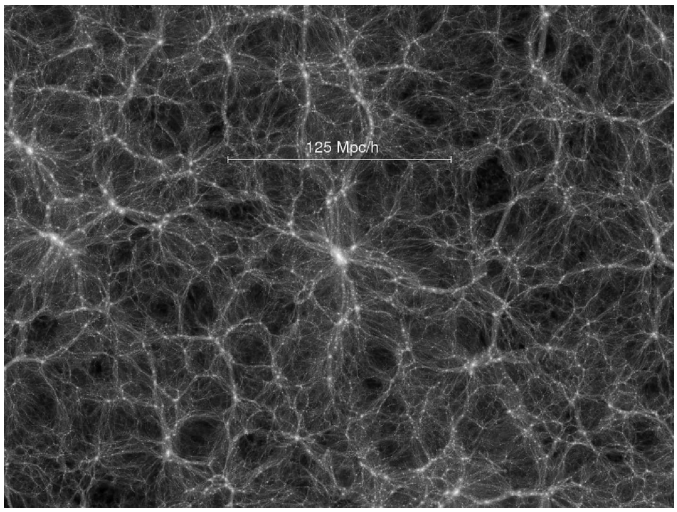


Figure 1: Dark Matter density field. This is a slice through the Millennium Simulation (see references). One can clearly see that the Dark Matter shows a filament like structure. There are also very dense and under dense regions. These under dense regions correspond to very large voids in the Universe.

Another possibility to get rid of the low resolution on mesh scales is to combine the mesh method with a particle based method. This means that the "bad" forces of the mesh on small scales are corrected by a summation of the direct particle forces for close neighbors. These methods are called $\mathrm{PP} + \mathrm{PM} = \mathrm{P^3M}$ methods (Particle-Particle plus Particle-Mesh). The direct summation of the PP part can also be replaced by a Tree based method. These codes are then called hybrid codes. A very efficient hybrid method is the TreePM method. It uses a force splitting between short and long range force. The short range force is calculated with a Tree whereas the long range part uses the PM method to calculate the forces.

The algorithm for the force calculation is only one problem in simulations. Another important issue is the so called domain decomposition strategy to divide the work between lots processors. Cosmological simulations are often run with a number of processors of the order of 1000. The goal is to reach optimal load and memory balance. There are different schemes around. The cosmological code Gadget uses a fractal space-filling Peano-Hilbert curve as decomposition scheme.

Once all the forces are calculated the simulation can be advanced one time step. The time integration algorithm that is mostly used is a quasi-symplectic leapfrog.

Cosmological simulations have to face lots of other technical issues like for example I/O issues, because the data needs to be stored in parallel, because the typical snapshot size is extremely large.

3 The Millennium Simulation

The Millennium Simulation is a project of the VIRGO consortium, a group of scientists from Germany, UK, Canada, Japan and the USA. The focus of this international team is to run large cosmological simulation and answer important questions by analyzing the output of these runs. The Millennium Simulation was running for about a month

on a 512 CPU cluster. After finishing the simulation lots of scientists started to analyze it and they still do until today. The amount of data is very large and the simulation gives us a perfect tool to test our models and see whether they are correct or not. The simulation was done with the Gadget code. Fig. 1 shows one output of the simulation. It is the Dark Matter density field of a slice through the simulation box.

4 Further reading

1. **How to simulate the Universe in a Computer** (Alexander Knebe)
   `http://arxiv.org/abs/astro-ph/0412565`
2. **Cosmological N-Body Simulations** (J.S. Bagla, T. Padmanabhan)
   `http://arxiv.org/abs/astro-ph/0411730`
3. **Cosmological N-Body simulation: Techniques, Scope and Status** (J.S. Bagla)
   `http://arxiv.org/abs/astro-ph/0411043`
4. **Millennium Simulation** (Springel et al)
   `http://www.mpa-garching.mpg.de/galform/press/`

Hacking
lecture
Tag 2 17:15
Saal 2
en

Jens Kubieziel

To be or I2P

An introduction into anonymous communication with I2P

I2P is a message-based anonymizing network. It builds a virtual network between the communcation endpoints. This talk will introduce the technical details of I2P and show some exemplary applications.

I2P has a different approach than most other known anonymous applications. Maybe you know about the anonymisation networt Tor. Here you have central directory servers, onion routers (relaying traffic), onion proxies (send and receive data from the user) and other software roles within the network. I2P calls every software a router and it can send and receive data for the user as well as relay traffic for other users. Furthermore I2P uses no central server for distributing information about routers. You'll get the information from I2P's network database. This is a pair of algorithms which share the network metadata. The routers participate in the Kademlia algorithm. It is derived from distributed hash table.My talk will tell you in detail how I2P work, what roles routers, gateways, netDb etc. plays. Furthermore I'll show differences and similarities to other anonymizing networks e. g. Tor and introduce some exemplary applications.

http://www.i2p.net/ I2P website

An Introduction to Anonymous Communication with I2P

To be or I2P

Jens Kubieziel <jens@kubieziel.de>

2007-12-27

Abstract Many of you may know about Tor or JonDo. These are widely deployed anonymising systems. Another promising approach is I2P. This paper will show the basic concepts of this network and introduce some applications.

1 Introduction

Anonymous communications are getting more important nowadays. On the one hand are companies which try to invade your privacy by using several well-known techniques (i. e. Cookies, JavaScript). These are used to build individual profiles of your behaviour and to send you better crafted spam. ☺ The government, on the other hand, creates laws (e. g. the data rentention law) designed to help improve law enforcement. But they can easily be abused to spy on you. And several "interested third parties" have declared a strong interest in the data gathered in this way. Therefore users see an increased need for protection against traffic analysis.

At past Chaos Communication Congresses, several solutions have been presented. There were remailers like Mixmaster[1] or Mixminion[2] as well as the anonymous network Tor[3] introduced. One interesting approach has however not yet been mentioned. The I2P[4] anonymous network tries to build VPN-like connections between its participants using a P2P-approach. The following document will give you a short overview of I2P. If you want a more detailed view of I2P's working principles have a look at the documents at the above mentioned website.

2 Nomenclature

I2P uses a special nomenclature for some parts of their protocol. To better understand the following it is important to know about it.

router Software which participates in the network.

tunnel A path through several routers which is used to transport encrypted packets.

inbound and outbound tunnel Every tunnel in I2P is unidirectional. The tunnel

[1] http://mixmaster.sourceforge.net/
[2] http://mixminion.net/
[3] https://www.torproject.org/
[4] http://www.i2p.net/

for incoming connections is called the inbound tunnel and the one for outgoing connections is called the outbound tunnel. A router usually has several inbound and outbound tunnels.

tunnel gateway This collects messages, does some preprocessing, encrypts the data and sends it to the next router. A gateway of an outbound tunnel is the creator of that tunnel. The gateway of an inbound tunnel receives messages from any peer and forwards them until they reach the creator.

endpoint The endpoint of a tunnel is either the creator (inbound) or the last hop of that tunnel (outbound). In the case of an outbound tunnel the endpoint is not necessarily the desired location. In fact, the endpoint looks for another tunnel gateway to send the packets along.

netDb is the short name for network database. It is a pair of algorithms which are used to share the network metadata. It gives your router all necessary data to contact other routers.

As you can see there is no client, server or exit nodes—in I2P every router can be client and server. It forwards packets from your computer as well as for other computers. Furthermore *all* communication stays within the I2P-network[5] and is end-to-end encrypted. A router doesn't know about its role and as the message is encrypted it has no possibility of learning about its contents.

[5]There are proxies for non-I2P communication.

3 Anonymous communication with I2P

What happens exactly if Alice wants to send a message to Bob? First, Alice's router must know how to reach Bob's. She asks the netDb for Bob's `leaseSet`. This is special metadata and gives Alice's router the gateways of Bob's inbound tunnels plus other information. Now Alice picks one of her outbound tunnels and sends it. The message has instructions for Alice' endpoint on how to forward the message to Bob's inbound gateways. The endpoint forwards the message as requested and Bob's gateway forwards it to Bob's router. If a reply from Bob to Alice's message is desired, Alice's destination is also sent in her message, so saving Bob from performing a netDb lookup.

This is the basic working principle of I2P. The following sections will show you details of I2P's components.

3.1 netDb

The network database, called netDb, shares network metadata consisting of a pair of algorithms. First there is a small set of routers called "floodfill peers". The rest of the routers participate in a special algorithm, Kademlia.

3.1.1 Network metadata

There are two types of network metadata: `routerInfo` and `leaseSet`.

The `routerInfo` structure supplies routers with the data necessary for contacting a particular router. It contains their public keys (2048 bit ElGamal, 1024 bit DSA plus a certificate), the transport address (IP address and port) and some arbitrary uninterpreted text options. All of this

information is signed with the included DSA key.

The other structure `leaseSet` is similar in some ways. It also contains the public keys (ElGamal, DSA and certificate) and includes a list of leases and a pair of public keys for encrypting messages to the destination. The leases specify one of the destination inbound tunnel gateways. This is achieved by including the SHA-256 hash of the gateway's identity, a 4 byte tunnel id and the expiration time of that tunnel.

3.1.2 Bootstrapping

How is the netDb initially built? A router needs at least one `routerInfo` of a reachable peer. It then queries that peer for references for other routers and uses the Kademlia healing algorithm. Each `routerInfo` reference is stored in an individual file in the router's netDb subdirectory. This allows these references to be easily shared, so bootstrapping new users.

3.2 Tunnels

As described above tunnels are unidirectional and consist of an inbound and an outbound tunnel. Both work along similar principles. They have a gateway, an endpoint and (probably) some routers inbetween. The gateway collects messages and performs some preprocessing. After these initial steps it encrypts the data and sends it to the first router in the tunnel. All subsequent routers check the integrity of the message and add a layer of encryption. At some point the message arrives at the endpoint, where it is forwarded as requested.

4 Applications

As you have seen I2P is an anonymous IP layer. What applications could you use with I2P? The developers have implemented several commonly-used programs. At the moment, programs for mail, websites, chat, filesharing and more exist. For most of these tasks, special programs are needed as commonly available software has no support for I2P.

4.1 Websites

Websites in I2P are called *eepsites* and have the top level domain `.i2p`. To visit an eepsite, point your browser's proxy to port 4444. Your local I2P client handles the request. Unlike Tor's hidden services, all eepsites use readable names. You can reach the eepsite of I2P via `http://www.i2p/` and I2P's forum at `http://forum.i2p/`.

If you want to provide information at your own eepsite, you must follow several steps:

1. pick a lowercase name for your eepsite
2. start the eepsite at your I2P configuration window and configure it
3. add content to `i2p/eepsite/docroot`
4. add your site to an I2P address book (`http://orion.i2p/` or `http://trevorreznik.i2p/`)
5. wait for your first visitor ☺, additionally you can make your site public by posting to the forum, to the wiki or telling others about it in IRC

Additionally you can browse to websites outside of I2P. Just set your local

HTTP proxy to localhost with port 4444 and enter "normal" domain names.

4.2 Email

For email there is a web interface or you can also use your mail client. An email address in i2p has the form `username@mail.i2p`. The username can be freely chosen. Just go to the Postman HQ[6] and create a new mailbox. This site also has instructions on how to setup your mail client. Once you are ready, you can send emails. Another way to send your emails is to use the web interface called *Susimail*. Just log on with your username and password.

You can also use I2P to communicate with the outside world. I2P mail can connect to an internet mail server[7] where it rewrites your email address with `username@i2pmail.org`. The receiver can answer it. The mail server will restore the domain name to `mail.i2p` and forward it to your mailbox.

4.3 Blogging

Syndie is a censor resistant, anonymous blogging tool. You can write postings which are then published on your local pc and on distributed archives. The software is not part of the I2P distribution. It can be downloaded from `http://syndie.i2p/` and, like I2P, is written in Java. After installation is finished, the software has to be configured. If you only want to read other postings, you can subscribe to the forum. In case you also want to publish blog postings, more work must be done. First choose a nickname, then choose how Syndie connects to archive servers and in the end add any desired forums. Syndie contains a button labelled `Post`. Click on it and write your postings.

4.4 Chat

The main chat protocoll is IRC. Point your chat client to localhost with port 6668 and choose a channel.

4.5 File sharing

There are several clients for several networks. I2PSnark is bundled with I2P and offers you access to Bittorrent. Furthermore the developers of Azureus have written azneti2p, which is also a Bittorrent client. I2Phex is a port of the Phex Gnutella client and, lastly, IMule allows access to eMule.

[6] `http://hq.postman.i2p/`
[7] `mx.i2pmail.org`

Culture
lecture
Tag 1 23:00
Saal 3
en

SkyOut

VX

The Virus Underground

The listeners will be introduced in the world of virus coding. They will understand how this can be seen as a way of expressing yourself and why it is a way of hacking. Furthermore they will get to know, which important groups, authors and viruses have been there in the last years and which are still active nowadays. Important technical terms will be explained as well as trends of the last years and the future.

The aim of the lecture shall be to introduce to the world of the virus underground. They shall understand how this little community of about fifty people think and act and why they code viruses. The audience may understand coding of viruses as a type of hacking and a way of expressing it as art. Furthermore it is the aim to make them familiar with different words, that are typically used by Virus Coders (VX), for example Appender, Prepender and Overwriter Virus. Even more different aspects of multiplatform malware and payloads shall be explained. Then the audience shall be introduced to different authors and groups of the scene, that are somehow the idols of many VXers, groups like EOF, DoomRiderz and more. People like Roy G Biv, Virusbuster and Benny and more. Going on, the lecture will describe the relationship between VXers and the AntiVirus companies, even it does not seem so, there is a connection between both groups. [...]

http://vx.netlux.org/
http://www.smash-the-stack.net
http://www.eof-project.net/
http://vx.netlux.org/
http://vxchaos.official.ws/
http://www.freewebs.com/purgatory-vx/
http://vx.eof-project.net/
http://www.29a.net/
http://www.rrlf.de.vu/
http://www.doomriderz.co.nr/
http://vxchaos.official.ws/

Für Einsteiger

VX – The Virus Underground

Marcell Dietl

Schwierigkeitsgrad

Immer wieder liest man in den Medien von neuen Viren und Würmern, welche in Umlauf kommen und großen Schaden anrichten. Doch kaum einer weiß, dass es eine kleine Gruppe von Leuten gibt, welche so genannte Malware programmiert, da sie es als eine Kunst des Hackings ansieht. Dieser Gruppe von Menschen und deren Ideologie wollen wir uns im Folgenden widmen.

Wenn die Schlagzeilen in Zeitschriften und Fernsehen wieder von einem neuen Virus oder Wurm geprägt sind, läuft es vielen kalt über den Rücken, sie befürchten eine neue Infektionswelle und entwickeln einen Zorn auf die Autoren solcher Programme, welche wir im Weiteren als Malware klassifizieren werden. Kaum einer bedenkt dabei jedoch, dass solche Programme auch durchaus aus positiven Beweggründen entstehen können. Leider wird das oft nicht erwähnt, da nur solche Malware die Schlagzeilen füllt, welche meist aus Profitgier entsteht. Die kleine Szene von vielleicht fünfzig Leuten, welche sich zum Ziel gesetzt hat, immer neue und kreative Programme zu erstellen, leidet unter dieser negativen Verallgemeinerung und wird in vielen Ländern mittlerweile genauso bekämpft wie die Kriminellen selbst. Im nachfolgenden Artikel wollen wir uns jedoch genau dieser Gruppe von Menschen widmen und lernen ihre Denkweise zu verstehen und erkennen, dass Viren durchaus auch eine Kunst des Hacking sein können, wenn auch eine komplett andere, als viele denken. Lassen Sie uns also einen Blick auf diese kleine Gruppe, ihre Idole, ihre Ideologie und Szenestruktur werfen und vielleicht werden Sie Ihre Meinung auch ein wenig ändern, wenn Sie das nächste Mal von einem neuen Virus oder Wurm lesen, welcher wieder einmal nur aus Geldgier entstanden ist.

Wichtige Gruppen der Szene

Später werden wir noch genauer die Probleme der VX Szene betrachten, doch hier sei schon

In diesem Artikel erfahren Sie...

- Wie die Virenszene aufgebaut ist;
- Welche Fachbegriffe es bei Virenschreibern gibt;
- Welche wichtigen Personen, Gruppen etc. es in der Szene gibt;
- Welche Verbindung zwischen Virenschreibern und Anti-Viren Firmen besteht;
- und vieles mehr...

Was Sie vorher wissen/können sollten...

- Außer einem Interesse an der Materie ist kein Vorwissen nötig.

einmal erwähnt, dass eines der Probleme eine ständige Veränderung der Gruppen- und Szenekonstellation ist. So gibt es ständig neue Virusautoren und Virusgruppen, doch nur wenige bleiben auf längere Sicht gesehen aktiv, was es durchaus erschwert genau festzulegen, welche Gruppen und Autoren es derzeit gibt und inwieweit diese noch aktiv sind. Nichtsdestotrotz möchte ich Sie hier mit einigen Namen von Gruppen vertraut machen, die zumindest nach jetzigem Stand der Dinge als aktiv gelten können. Ob sie es in Zukunft weiterhin bleiben ist fraglich. Lassen Sie uns mit den zwei wichtigsten Gruppen anfangen, die eine trägt den Namen Ready Rangers Liberation Front (rRlf) und ist eine deutsche Gruppe, welche es seit nunmehr sieben Jahren gibt. Sie haben viele Veröffentlichungen in den letzten Jahren hervogebracht und es ist unwahrscheinlich, dass sie in nächster Zeit inaktiv werden. Die andere und wohl bekannteste Gruppe der ganzen Szene ist 29A, was den Hexcode von 666 darstellt. Es ist eine internationale Gruppe, welche ebenso vor vielen Jahren zu Beginn des VX (Virus Coding) gegründet wurde. Sie brachte viele Veröffentlichungen heraus und ist auch heute noch aktiv, jedoch hat sich die Struktur leicht verändert. Um Ihnen eine gute Basis zu geben, seien hier noch die Gruppen DoomRiderz aus Amerika (wo es mittlerweile als Verbrechen gilt Viren zu schreiben), Purgatory aus dem Iran, F-13 Labs und das EOF-Project(.net), welches ich im Jahre 2006 gründete und das seit 2007 unter neuer Leitung steht, genannt. Natürlich kann ich Ihnen hier nur einen groben Überblick geben, da es ständig zu Veränderungen der Szenestruktur kommt, aber zumindest sind Ihnen jetzt schon einige Begriffe geläufig. Alle Gruppen bestehen meist aus Mitgliedern verschiedener Staaten, so ist etwa der Leader (=Anführer/Organisator) des amerikanischen Teams selbst nicht aus Amerika, jedoch wurde diese Gruppe ursprünglich in Amerika gegründet, hat sich jedoch mittlerweile stark gewandelt. Auch das EOF-Project besteht aus Mitgliedern der verschiedensten Staaten dieser Welt, eine typische Eigenart der Szene, da es meist nicht länderspezifisch organisiert ist. Ich hoffe, dass Ihnen dieser Abschnitt schon einmal einen kleinen Einblick in die wichtigsten Gruppen der Szene gegeben hat.

Die Ideologie der Autoren

Im letzten Absatz haben Sie eine kleine Einführung in einige wichtige Gruppen der Szene erhalten, nun wollen wir uns der Ideologie der Autoren widmen, denn nicht alle programmieren Viren aus den gleichen Grundsätzen heraus und es gibt enorme Unterschiede. Natürlich könnte man hier als erstes solche nennen, welche Malware nur erstellen, um damit Geld zu verdienen. Sei es, weil sie etwas VCKs (Virus Creation Kits) weiterverkaufen oder weil sie Rechner infizieren, um sensible Daten auszuspähen. Doch denen wollen wir uns gar nicht weiter widmen, denn mit diesen Kriminellen hat ein VXer nichts zu tun. Ansonsten könnte man die Virenschreiber in zwei große Gruppen einteilen, die Hobbyisten und die Ideologen. Gängiger Definition nach versteht man unter der Hobbyisten solche Programmierer, welche gelegentlich einen Virus schreiben oder auch nur ein einziges bis einige wenige Male, um das als kreative Herausforderung zu sehen. Daher kommt auch die ständige Umstrukturierung der Szene, da es ständig neue Autoren gibt, welche jedoch nach wenigen Viren meist wieder verschwinden. Hobbyisten sehen das Programmieren eines Virus als interessante Herausforderung, aber widmen sich schnell wieder anderen Themengebieten. Dann gibt es noch die Ideologen. Sie programmieren Viren und sonstige Malware, da es für sie eine künstlerische Herausforderung darstellt. Sie wollen ständig neue Techniken erlernen und umsetzen und suchen neue Möglichkeiten, etwa neue Verbreitungswege. Man kann schwer sagen, wer in welche Gruppe einzuordnen ist, meist ist die Grenze recht fließend, doch lässt es sich ganz gut zeitlich festlegen. Wer lange in der Virenszene war und immer aktiv blieb, gehört eher zu den Ideologen als den Hobbyisten. Als ein Beispiel sei hier der Autor Roy G Biv von 29A genannt, welcher heutzutage als einer der längsten und aktivsten in der Szene tätig ist. Er gehört eindeutig zu den Ideologen. Er ist zwar nicht der einzige, doch ein sehr gutes Beispiel dafür. Einen entscheidenden Unterschied gibt es schließlich noch, nämlich VXer, welche ihre Viren in die *Wildbahn* aussetzen und solche, welche Viren zwar veröffentlichen, jedoch nicht, um damit destruktiv Schaden anzurichten. Würde man es mit Hackern vergleichen, könnte man bei den meisten VXern von Whitehats sprechen, da sie alles im gute Sinne programmieren und bei solchen, welche Schaden anrichten wollen von Blackhats. Doch dies nur als kleine Parallele zu der Welt des Hacking (was ja VX auch bedingt ist).

Abbildung 1. *Das Logo der Gruppe Ready Rangers Liberation Front*

Vergangenheit – Gegenwart – Zukunft

Nachdem wir uns zuerst mit einigen Namen und der Ideologie der Szene vertraut gemacht haben, wollen wir jetzt einen Blick auf die Entwicklung der Virenszene vom Beginn bis heute und auch zukünftig werfen. Den Anfang von Malware, damals vor allem simple Viren, kann man etwa auf den Beginn der achtziger Jahre datieren. Erste Viren tauchten zum Beispiel 1982 für den Apple II Computer auf und waren mehr scherzhaft als wirklich destruktiv. Im Laufe der achtziger Jahre tauchten immer neue Viren auf, später vor allem für Windows, das Betriebssystem von Microsoft, welches sich immer weiter verbreitete. Ziel der damaligen Autoren war es vor allem möglichst interessante neue Techniken aufzuzeigen, teilweise indem die Viren immer destruktiver wurden oder einfach in ihrer Art immer einfallsreicher. Seien es scherzhafte Fehlermeldungen oder das Formatieren der kompletten Festplatte, viele Virenschreiber wollten Aufmerksamkeit und Anerkennung. Nachdem die ersten Viren noch über Disketten verbreitet wurden, kam zu Beginn der neunziger Jahre beziehungsweise Ende der achtziger Jahre das Internet immer mehr auf. Viel mehr Computer wurden internetfähig und boten somit auch ein immer interessanteres Ziel für VXer, vor allem da es mittlerweile zum Standard gehörte Windows zu nutzen. So kam es schließlich zu den ersten größeren Wurminfektionen, angefangen bei Würmern, wie dem Morris-Wurm bis hin zu dem I LOVE YOU-Wurm. Dieser Trend der Würmer erreichte seinen Höhepunkt schließlich zu Beginn des 21. Jahrhunderts mit Würmern wie Sasser, MyDoom, Netsky und weiteren, welche noch heute in vielen Varianten aktiv sind (Anmerkung: Netsky führt auch heute noch immer die Wurmstatistiken an, basierend auf Analysen verschiedener Antivirenfirmen über den prozentualen Anteil an verseuchten Emails. Außerdem sei hier noch der StormWorm genannt, welcher mit bis zu 12 Millionen Infektionen einen der größten, wenn nicht den größten, Wurm darstellt. Als Botnet eingerichtet stellt er den leistungsstärksten Rechner der Welt dar). Blickt man zurück könnte man sagen, dass es zuerst die Viren waren, teilweise sogar als Studentenarbeiten geschrieben, welche Aufsehen erregten, später waren es die Würmer.

Wie sieht es heute aus? Betrachtet man die aktuelle Lage der Virenszene geht der Trend eindeutig zu Trojanern, also solche Programme, welche sich unter einem falschen Vorwand auf dem System einnisten und dem Autor eine Backdoor (=Hintertür) öffnen, durch die er immer wieder Zugriff auf den Computer erhalten kann. Im gleichen Zuge werden Botnets immer populärer und haben ihren Höhepunkt erreicht oder streben ihn zumindest immer weiter an. Bots sind solche Programme, welche trojanerähnlich aufgebaut sind, sich jedoch zum Beispiel via IRC zu einem kompletten Netzwerk zusammenschließen, welches dann von einer kleinen Gruppe von Leuten administriert wird. Das wohl derzeit bekannteste Beispiel ist der StormWorm, welcher Millionen von Rechnern infiziert hat und welcher dazu dient das wohl größte Botnet aller Zeiten aufzubauen. Im Vordergrund steht heute also immer häufiger Geld und Diebstahl sensitiver Daten. Im Kontrast zu früher, als es eher eine kreative Herausforderung darstellte Viren und Würmer zu programmieren. Doch vergessen Sie eines hierbei nie: Auch wenn immer mehr Malware heutzutage zu Geldzwecken geschrieben wird, gibt es eine kleine Gruppe von Leuten, die diesem Trend strikt entgegenwirkt und weiterhin Viren nur als kreative Herausforderung schreibt, denen widmen wir uns schließlich hier.

Betrachten wir als letztes noch, welchen Trend es wohl in der Zukunft geben wird. Zwar ist das reine Spekulation, doch scheint es derzeit so, dass Handyviren immer weiter an Popularität gewinnen werden. Schon heute gibt es verschiedene Proof-of-Concept Viren, welche beweisen, dass es möglich ist ein Handy zu infizieren und wie es früher mit einigen wenigen Viren begann, welche einfach nur beweisen sollten, dass es geht, wird es sich vielleicht auch mit den Handyviren verhalten. Dies war natürlich nur ein grober Überblick und er kann gewiss nicht alle Faktoren abdecken, bedenken Sie das bitte immer.

Plattformübergreifende Malware

Ein Trend, der eben noch nicht angesprochen wurde, der jedoch derzeit in der VX Szene auch immer populärer wird ist das Schreiben von Malware, welche sich plattformübergreifend verbreiten kann. Lassen Sie uns also einen Blick auf verschiedene Möglichkeiten werfen, wie dies derzeit realisiert wird und das Ganze auch beispielhaft betrachten. Es gibt natürlich die verschiedensten Möglichkeiten plattformübergreifende Programme, in unserem Fall Viren, Würmer und Trojaner (zusammenfassend Malware), zu schreiben und es wäre schwierig jede Möglichkeit abzudecken, doch lassen Sie uns hier zumindest die typischsten Varianten betrachten. Ein Trend, welcher erst vor kurzem für großen Wirbel gesorgt hat, ist die Möglichkeit Viren zu schreiben, welche als Makroprogramme in Drittprogrammen ausgeführt werden. Klingt erst einmal sehr komplex, ist jedoch recht

Abbildung 2. *Das Logo von 29A*

simpel. Jedes größere Office Paket enthält heutzutage die Möglichkeit bestimmte Programmroutinen zu automatisieren, es enthält also eine eigene kleine Programmiersprache, mit der man Zusatzkomponenten erstellen und diese in sein Dokument auch einbinden kann. Diese Programmteile, welche man in ein Dokument einbinden kann, sind Makros, kleine Programmroutinen also. Der Vorteil eines Makrovirus ist es, dass die großen Office Pakete meist auf verschiedenen Betriebssystemen lauffähig sind. So ist es also möglich einen Makrovirus zu schreiben, in ein Dokument einzubinden und schließlich zu versenden. Für den Makrovirus ist es egal auf welchem Betriebssystem er ausgeführt wird, da die Interpretierung seiner Routinen dem jeweilig eingesetzten Office Paket überlassen wird.

Lassen Sie uns das ganze an einem populären Beispiel betrachten, um Ihnen das Ganze besser verständlich zu machen. Zusammen mit dem Team DoomRiderz, welches ich Ihnen vorhin kurz vorgestellt habe, begann ich vor etwa einem Monat einen großen Proof-Of-Concept Wurm zu programmieren, der sich über OpenOffice installierte und etwa über IRC verbreitete. Der Wurm wurde als Makrovirus gestaltet und in der für OpenOffice üblichen Programmiersprache StarBasic programmiert. Diesen Teil übernahm Necronomikon von DoomRiderz. Nun war es an der Reihe dieses Konstrukt mit den jeweils für die Betriebssysteme Windows, Linux und Mac OS X gestalteten Droppern zu versehen. Ein Dropper stellt dabei ein kleines Programm dar, welches für das jeweilige Betriebssystem geschrieben ist und auf diesem als Datei abgelegt wird, um dann zur Ausführung zu kommen. Meine Aufgabe bestand nun darin, einen Dropper für Mac OS X zu programmieren, welchen ich in Ruby verfasste. Weitere Dropper wurden schließlich noch für Windows und Linux geschrieben. Alles zusammen entstand schließlich ein StarBasic Wurm, der erkannte auf welchem Betriebssystem er lief und dementsprechend eine Datei ablegte, welche ausgeführt wurde. Genau das ist das Prinzip eines Makrovirus. Ich hoffe, dass Ihnen dieses kleine Beispiel ein besseres Verständnis der Materie vermitteln konnte. Genannt haben wir den Wurm BadBunny, passend zu seinem Payload (Begriff wird später noch genauer betrachtet). Sie können unter Google genügend Material dazu finden. Eine weitere Möglichkeit, einen plattformübergreifenden Virus zu erstellen, stellt das Benutzen von zur Verfügung gestellten Frameworks dar. Hierbei wäre das wohl typischste Beispiel das .NET Framework von Microsoft, welches es mittlerweile dank der Arbeit vieler Freiwilliger auch auf anderen Betriebssystemen gibt. Programmiert man einen Virus etwa in der für dieses Framework typischen Programmiersprache C#, so muss der Code nicht aufwendig angepasst werden, um die gleiche Wirkung auf anderen Betriebssystemen zu haben, allein die Pfadangaben und andere Kleinigkeiten können natürlich variieren. Einen sehr guten Artikel hat Paul Sebastian Ziegler zu diesem Thema in Hakin9 4/2007 verfasst, auch mit gutem Beispielcode. Ähnlich dieser Technik funktioniert die nächste Möglichkeit Viren für verschiedene Systeme zu erstellen: Das Schrei-

Abbildung 3. *Das Logo der DoomRiderz Gruppe*

ben von Programmen in Skriptsprachen. Bekannte Beispiele wären etwa PHP, Perl, Python oder Ruby. Codes, welche in solchen Sprachen erstellt werden, sind wieder insoweit plattformunabhängig, dass sie von einem Programm, dem Interpreter, auf dem jeweiligen Betriebssystem ausgeführt werden. Ein Programm, welches zum Beispiel die Funktion enthält den aktuellen Ordner zu löschen, wird dabei auf jedem System laufen, welches den passenden Interpreter installiert hat. Wenn Sie Interesse an so etwas haben schauen Sie sich zum Beispiel den sehr einfachen Beispielvirus Cyanotic an, welchen ich auf meiner Webseite (*www.smash-the-stack.net*) zur Verfügung stelle. Er ist in Java geschrieben und macht nichts anderes als die Dateien des aktuellen Ordners, in dem er sich befindet mit einem String (ein aus mehreren Zeichen bestehender Satz) zu überschreiben. Jedes System, was über Java verfügt kann diesen Virus auch zur Ausführung bringen, es bedarf keinerlei Anpassungen. Eine letzte, jedoch eher seltener genutzte Möglichkeit ist es einen Virus zum Beispiel in einer LowLevel Programmiersprache zu erstellen, etwa Assembler und den Code so zu gestalten, dass er sich dem jeweiligen Betriebssystem entsprechend verhält. Der Aufwand ist demnach enorm, sollten Sie es dennoch sehr interessant finden und weitere Informationen wünschen, die diesen Artikel und Abschnitt sprengen würden, so empfehle ich eine Suche nach dem wohl bekanntesten Beispielcode, mit dem Namen Winux, eine Anspielung auf die Kombination der Wörter Windows und Linux (Anmerkung: Der Virus ist unter verschiedenen Bezeichnungen zu finden: Linux.PEElf.2132, W32.Winux, Linux.Winux, W32/Lindose).

Abbildung 4. *Das Logo des iranischen Purgatory VX Teams*

Genutzte Verbreitungswege

Die letzten Abschnitte ermöglichten Ihnen nun schon einen kleinen Einblick in die VX Szene und ihre Entwicklung über die letzten Jahre hinweg. Mit was wir uns bisher nicht beschäftigt haben, ist das Thema, wie typische Viren aufgebaut sind und welche Fachbegriffe oder Techniken häufig eingesetzt werden. Dem wollen wir uns jetzt widmen. Lassen Sie uns einen Blick darauf werfen, welche typischen Verbreitungswege VXer bei ihren Viren einsetzen. Natürlich gibt es mehr Möglichkeiten, als die, welche ich Ihnen hier vorstellen werde, aber es ist zumindest ein kleiner Einblick in typische Wurmtechniken. Wie wir vorhin schon erfahren haben, wurden die ersten Viren noch als kleine Projekte von vorwiegend Studenten geschrieben, verbreitet wurden sie zumeist über Freunde und Disketten. Man reichte die Datei untereinander weiter und so entstanden die ersten *Würmer*, obwohl sie es im eigentlich Sinne noch nicht waren. Heute greift man auf andere Techniken zurück. Eine Möglichkeit, die häufig genutzt wird, stellt es dabei dar, von der Autostart Funktion einer CD oder DVD ROM Gebrauch zu machen. Nur wenige Nutzer von Windows haben diese deaktiviert und so ist es ein Leichtes einen Virus zu schreiben, welcher sich auf ein Medium kopiert, um von dort andere Rechner zu infizieren. Eindeutiger Nachteil dieser Methode ist erneut, dass es eines Anfangs bedarf, also etwa einem Nutzer, welcher absichtlich eine CD mit dem Virus beschreibt und weiterreicht. In neuerer Zeit werden USB Sticks immer mehr genutzt und erste USB Sticks enthalten eine Autostart Funktion von Programmen. Auch diese Möglichkeit eignet sich gut, um den eigenen Virus schnell an andere zu verbreiten und dort eine automatische Ausführung zu provozieren. Doch auch dies Bedarf der Tatsache, dass ein USB Stick eingesteckt ist und weitergereicht wird. Welche Möglichkeiten gibt es noch und welche sind vor allem hiflreicher einen Wurm schnell zu verbreiten? Diese Frage lässt sich nicht eindeutig beantworten, jedoch zeichnen sich einige Trends ab, die man hier benennen kann. In den letzten Jahren wurden so genannte Peer-To-Peer- oder kurz P2P-Netzwerke immer beliebter. Sie erlauben es beliebige Dateien und Ordner des eigenen Rechners anderen zur Verfügung zu stellen, damit diese darauf zugreifen können. Viele solcher Tauschprogramme enthalten Standardordner, in denen diese Freigaben gespeichert werden und so ist es unter VXern sehr beliebt ihren Virus mit interessanten Dateinamen zu versehen – etwa ein Crack für Windows – und in diesem Ordner abzuspeichern. Diese Methode funktioniert ähnlich bei Tauschseiten, so genannten Sharehostern, wie etwa Rapidshare oder ähnlichen. Man stellt seinen Virus möglichst anonym online, gibt ihm einen interessanten Inhalt und wartet, bis er sich verbreitet. Man könnte sich nun noch vorstellen, dass man automatisiert in ungeschützten Foren oder Blogs Werbung für diese Datei macht und hofft, dass so mehr Leute das Programm herunterla-

den und ausführen werden. Diese Methode funktioniert erwiesenermaßen ziemlich gut und wird zum Beispiel auch von dem aktuellen StormWorm eingesetzt, welcher sich unter anderem über Blogs verbreitet. Doch nichtsdestotrotz bleibt die beliebteste Verbreitungsvariante weiterhin die Email. Viele Nutzer benutzen noch immer Outlook zum Versenden von Emails und es gibt genügend Beispielcodes, wie man es schafft seinen eigenen Virus beziehungsweise Wurm an alle Emails weiterzuversenden, welche im Adressbuch eingetragen sind. Auch der eben schon genannte StormWorm nutzt Emails als Hauptverbreitungsweg und verweist darin auf infizierte Webseiten, welche den Virus enthalten. Wie man sieht: Es funktioniert. Ich hoffe, dass Ihnen dieser Abschnitt einen kurzen Überblick über beliebte Verbreitungswege der Würmer gegeben hat, von früher bis zur Gegenwart. Betrachtet man die Zukunft wird wohl eine Verbreitung über MMS oder SMS Würmer oder Bluetooth Malware immer häufiger anzutreffen sein, letztendlich bleibt es jedoch vorerst reine Spekulation. (Anmerkung: Erste Konzepte liegen bei Bluetooth Malware schon vor, wie der Artikel von Marko Rogge in Ausgabe 9/2007 zeigt)

Payloads

Allein schon die Überschrift dürfte bei manchen Fragen aufgeworfen haben. Was ist ein Payload überhaupt? Genau dieser Frage wollen wir uns nun widmen und auch der Fragestellung, welche unterschiedlichen Typen von Payloads es gibt. Der Payload ist wohl der wichtigste Teil, den ein VXer bei seiner Arbeit an einem Virus bedenken muss; er beschreibt sozusagen den Kern des Virus. Allgemein könnte man sagen, dass der Payload der Teil des Codes ist, welcher zur Ausführung kommt, wenn der Virus aktiviert wird (etwa das Löschen eines Ordners etc.). VXer ordnen dabei die Art eines Virus entsprechend ihres Payloads in mehrere Kategorien ein. Die Oberkategorien wären Overwriter, Appender und Prepender. Wenn Ihnen diese Begriffe nichts sagen, so seien Sie beruhigt, wir werden sie nun genauer betrachten. Will man diese Kategorien verstehen, hilft es schon, wenn man die Namen aus dem Englischen übersetzt. Ein Overwriter stellt hierbei die simpelste Form eines Virus dar. Der Virus überschreibt einfach nur spezifische Dateien, entweder mit einem bestimmten Inhalt oder aber in vielen Fällen mit seinem eigenen Code. Ein Overwriter könnte also etwa so aussehen, dass ein Perl Virus im aktuellen Ordner nach Perl Dateien sucht und deren Inhalt so manipuliert, dass er deren Inhalt mit dem seinigen ersetzt. Führt der Anwender nun seine Perl Skripte aus, wird er in Wahrheit den Virus zur Ausführung bringen. So entsteht ein simpler Overwriter Virus. Die Begriffe Appender und Prepender sind von ihrer Struktur her recht ähnlich, nur der Code ist leicht angepasst. Im Prinzip bezeichnen sie das Voranstellen (Prepend) oder Hintenanstellen (Append) des Viruscodes an eine andere Datei. Klingt erst einmal sehr komplex, ist jedoch sehr leicht zu verstehen. Stellen sie sich wieder eine Perl Datei vor, welche eine Umrechnung von Grad in Fahrenheit vornimmt und einen Virus, welcher alle Dateien im aktuellen Verzeichnis infiziert, etwa einen Appender. Der Virus wird den Code des Perl Skriptes insofern nicht löschen, dass er nicht, wie bei einem Overwriter den kompletten Inhalt löscht. Vielmehr wird er sich hinten an den Code anhängen. Was dann passiert ist folgendes: Das Programm ist in seiner eigentlichen Funktion als Umrechnungsinstrument weiter nutzbar, führt jedoch am Anfang einen Jump (=Sprung) zum Viruscode aus, um anschließend wieder den Originalcode auszuführen. Ein Prepender tut in etwa das Gleiche, führt jedoch erst den Viruscode aus, um mit einer anschließenden Verzögerung ein Backup des Programmes zur Ausführung zu bringen. Wie sie sich denken könnten, waren dies nur wieder einige wenige Möglichkeiten eines Payloades, die typischsten sozusagen. Natürlich gibt es auch weitere, der Fantasie ist dabei keine Grenze gesetzt. Ein Payload wäre es auch, wenn etwa der aktuelle Hintergrund des Systems mit einer Signatur des Autors versehen wird oder aber wenn bestimmte AntiViren Seiten gesperrt werden. Zu unterscheiden gilt es hierbei noch einmal zwischen zwei Gruppen von Payloads: Den auffälligen und unauffälligen Payloads. Auffällige Payloads sind solche, die den Anwender eindeutig spüren lassen, dass er mit einem Virus infiziert wurde, etwa durch ein *merkwürdiges* Verhalten des Computers. Unauffällige Payloads sind solche, welche etwa bei einem Trojaner zum Einsatz kommen. Der Anwender soll von der Infektion keine Kenntnis nehmen und möglichst keinen Verdacht schöpfen. Ein gutes Beispiel sind hierbei die Codes von Joanna Rutkowska, welche gezeigt hat, wie es mit Virtualisierungstechniken möglich ist, das komplette System in eine Virtuelle Maschine zu versetzen, sodass der Anwender nicht merkt, dass er infiziert wurde. Auch aktuelle Anti-Viren Programme sind dann meist machtlos den Übeltäter überhaupt erst aufzuspüren. Bei Würmern stellen die Payloads meist zustäzlich die eingesetzte Verbreitungsroutine dar, etwa das Versenden via Email oder P2P-Netzwerk. Auch wäre es möglich Würmer über Netzwerkfreigaben zu verbreiten oder ähnliches. Bei Trojanern widerum stellt der Payload die Routinen dar, welche zur Verschleierung des Trojaners gedacht sind, etwa das Aktivieren des Trojaners als Linux Kernel Modul.

Abbildung 5. *Das Logo der Gruppe Electrical Ordered Freedom (EOF)*

Kommunikation unter VXern

Mittlerweile sollten Sie schon einen großen Einblick in die Welt der Virusprogrammierung gewonnen haben, doch ein paar wenige Aspekte stehen noch aus. Denen wollen wir uns nun abschließend noch widmen. Da wäre zum einen die Frage, wie VXer miteinander kommunizieren und ihre Informationen untereinander austauschen. Wie die meisten Szenen im Internet betreiben fast alle Virenschreiber ihre eigenen Webseite, auf denen Sie ihre Kunst, ihre Viren, zur Schau stellen und anderen Einblicke in den Quellcode ermöglichen. In den seltensten Fällen liegen den Quelldateien die kompilierten Versionen bei, das liegt an der Ideologie, die sie nun schon kennengelernt haben. Es geht darum Wissen zu verbreiten, nicht aber Schaden anzurichten. Ein erster Anlaufpunkt für VXer stellt dabei die Seite vx.netlux.org (mittlerweile verlinkt diese auf vx.org.ua) dar, welche ein Riesenarchiv an Texten und Quellen enthält, über die man sich grundlegendes bis sehr spezifisches Wissen aneignen kann. Viele VXer Seiten sind direkt von dort auch verlinkt. Weiterhin lässt sich sagen, dass die meisten Seiten in Englisch gehalten sind. Wie auch im Berufsleben stellt Englisch die wichtigste Sprache zum Informationsaustausch dar, auch wenn manche Gruppen länderspezifisch organisiert sind. Nächstes Mittel der Kommunikation sind Emails, welche vorwiegend dann genutzt werden, wenn Informationen gezielt untereinander ausgetauscht werden, etwa über ein neues Projekt oder eine Technologie, welche bisher noch geheim gehalten werden soll. Ähnlich verhält es sich hier mit diversen Instant Messengern wie ICQ, MSN und Yahoo. Eines der wohl wichtigsten Kommunikationskanäle für VXer stellt das Undernet dar. Ein auf IRC basierendes Chatnetzwerk, in dem die wichtigsten VX Channels anzutreffen sind und über die die meisten Leute Einstieg in die Szene gewinnen und Kontakte knüpfen. IRC ist DAS Chatmedium unter VXern, wenngleich es nicht zu den sichersten gehört, ist es doch das beliebteste. Eine Hierarchie gibt es dabei kaum, jeder Fremde wird meist freundlich empfangen und nach seinen Interessen gefragt. Hilfe wird meist schnell und konkret geboten. Das Undernet stellt eine perfekte Anlaufstelle für jeden Interessierten dar. Von anderen Chattechnologien wie SILC halten sich Vxer bislang fern, auch wenn diese sicherheitstechnisch betrachtet sinnvoller erscheinen könnten. Doch all diese Mittel erscheinen erst einmal trivial, es bleibt jedoch noch eine letzte und wohl die wichtigste Kommunikationsart offen, die E-Zines. Der Begriff bedeutet nichts anderes als elektronisches Magazin. Im Normalfall ist damit eine auf HTML basierende Dateiensammlung gemeint, welche Quellcodes und Texte enthält und über den aktuellen Stand einer Gruppe oder über die Szene allgemein informiert. Diese E-Zines werden von verschiedenen Gruppen in regelmäßigen Abständen herausgegeben und werden dabei immer von der ganzen Szene mit Spannung erwartet, da im Vorfeld meist nicht bekannt ist, welche neuen Viren und Techniken veröffentlicht werden. Immer häufiger wird anstatt auf E-Zines auch auf Foren zurückgegriffen, doch hat sich diese Technik noch nicht genügend bewährt und wird eher weniger genutzt. Wie man sieht, nutzen auch VXer ganz normale Kommunikationswege, wie jeder andere auch, greifen jedoch auch auf szenetypische Eigenarten zurück.

Zusammenarbeit zwischen VXern und AntiViren Firmen

Auf den ersten Blick mag diese Überschrift sehr merkwürdig erscheinen. Welchen Grund sollte auch nur eine der beiden Parteien haben mit der anderen in irgendeiner Weise zusammenzuarbeiten, sind sie doch das komplette Gegenteil und kämpfen immer darum, dass sie die Oberhand gewinnen? Doch betrachtet man das ganze intensiver, wird deutlich, dass es eine Verbindung zwischen beiden gibt. Sie ist bei Weitem nicht mit der unter VXern zueinander zu vergleichen, aber es gibt sie. Eine Eigenart von VXern ist es zum Beispiel ihre Viren in ausführbarer Form (also etwa als *.exe Datei) an AntiViren Firmen zu schicken, damit diese die Viren untersuchen und eine Diagnose auf-

Über den Autor

Unter seinem Nicknamen SkyOut arbeitet Marcell Dietl seit 2007 an seiner Projektseite *www.smash-the-stack.net*, auf der er Programme und Texte rund um IT Security (VX eingeschlossen) veröffentlicht. Ebenso führt er dort seinen Blog und ist unter *skyout@smash-the-stack.net* zu erreichen. Derzeit bildet er sich persönlich im Bereich IT Security immer weiter und strebt nächstes Jahr eine Lehre oder ein Studium zum Informatiker an.

Im Internet

- *http://www.29a.net* – Die Seite der berühmten Gruppe 29A;
- *http://www.rrlf.de.vu* – Die Seite der deutschen Ready Rangers Liberation Front;
- *http://www.doomriderz.co.nr* – Die Seite des amerikanischen DoomRiderz Teams;
- *http://www.freewebs.com/purgatory-vx/* – Die VX Seite des iranischen Purgatory Teams;
- *http://www.eof-project.net* – Das EOF-Project;
- *http://vx.eof-project.net* – Das VX Forum des EOF-Projects;
- *http://vx.netlux.org* – Zentrale Anlaufstelle rund um Viren – mit Texten, Quellcodes und Links.
- *http://vxchaos.official.ws* – Datensammlung rund um VX und Security

stellen. Für beide Seiten bietet diese Verhaltensweise Vorteile. Die Firmen werden über Viren informiert, bevor diese in Umlauf kommen und können ihre Kunden besser sichern, die VXer entwickeln mit jedem mal ein wenig mehr Stolz. Unter VXern ist eine Virus Beschreibung seitens einer AntiViren Firma wie Sophos oder F-Secure, wie eine Trophäe, mit der man anderen imponieren kann und sein eigenes Selbstwertgefühl steigern kann. Eine weitere Verbindung, die zwischen VXern und Mitarbeitern bei Firmen, wie auch Symantec, besteht, ist die Tatsache, dass sich eben diese Mitarbeiter auch immer mit der Virusszene auseinandersetzen müssen. So hat mittlerweile fast jede große Firma einen eigenen Blog, in dem sie Neuigkeiten über die Szene niederschreibt. Es besteht also ein ständiges Interesse seitens der Firmen über die Szene auf dem Laufenden zu bleiben. Das schafft in gewisser Weise eine ironische Verbindung, trotz der gegensätzlichen Positionen, haben sie doch einiges gemein. Sie arbeiten zwar auf zwei verschiedenen Seiten, doch sind sie auch voneinander abhängig oder wie sollte eine Firma weiter bestehen, wenn es nichts mehr zu schützen gäbe, weil keine Gefahr mehr bestünde? Interessanterweise lässt sich hier noch anmerken, dass es zwei Arten von Firmenmitarbeitern gibt. Solche, die eher friedlich gegenüber der Szene eingestellt sind und solche, welche die Szene mit allen Mitteln zu bekämpfen versuchen, wenngleich dies komplett kontraproduktiv erscheint. Namen sollen hier nicht fallen, doch sind unter den VXern einige Analysten sehr in Verruf geraten, da sie sich zum Beispiel versuchten in die Szene einzuschleußen, um sie dann im Kern zu zerschlagen. Zum Glück ist ihnen das nie gelungen.

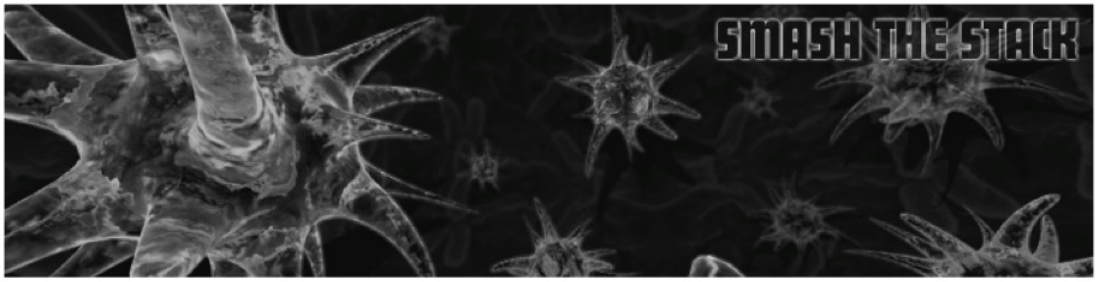
Abbildung 6. *Das Logo eines der größten Fileserver rund um VX*

Programmiersprachen der Szene

Abschließend wollen wir noch einen Blick auf typische Programmiersprachen der VX Szene werfen. Es gibt gewisse Sprachen, welche mehr vertreten sind und gewisse Trends, die sich in den letzten Jahren abgezeichnet haben. So lässt sich zum Beispiel sagen, dass Sprachen, welche plattformübergreifende Malware ermöglichen in letzter Zeit an Bedeutung gewonnen haben. So wird immer häufiger auf C# und das .NET Framework zurückgegriffen. Doch auch andere Sprachen dieses Frameworks werden immer wichtiger, etwa VB .NET. Weiterhin lässt sich sagen, dass vor allem Sprachen sehr beliebt sind, welche unter Windows funktionieren, da Windows noch immer die auch bei VXern meistgenutzte Plattform darstellt. Viele VXer programmieren zum Beispiel in Visual Basic, eine typische Programmiersprache für das Windows Betriebssystem. Als Trend zeigt sich hier vor allem eines: Skriptsprachen werden immer beliebter, hingegen sind LowLevel Sprachen immer weniger vertreten. So gibt es immer weniger Viren in Sprachen wie C, jedoch immer häufiger welche in Perl oder ähnlichen Skriptsprachen. Als Königsdisziplin gilt unter VXern jedoch das Programmierern in einer bestimmten Sprache: Der Assemblersprache. Welche Assemblervariante, sei es AT&T oder Intel Syntax, ist dabei erstmal zweitrangig. Die wichtigsten Viren der Szene wurden meist in ASM geschrieben und etwa die berühmte Gruppe 29A programmierte beeindruckende Viren in ASM, teils mit Quellcodes von bis zu 10 000 Zeilen. Auch sehr beliebt sind Makrosprachen, etwa für das produzieren von Viren für Word oder, wie wir gesehen haben, auch OpenOffice. Was lässt sich hier also zusammenfassend sagen: LowLevel Sprachen gelten weiterhin als Königsdisziplin eines jeden VXers, wenngleich Skriptsprachen immer beliebter werden, ebenso wie Sprachen für das .NET Framework.

Probleme der VX Szene

Als letzten Aspekt der Szene wollen wir noch einmal auf die Probleme dieser eingehen. Hier lassen sich vor allem zwei Dinge anmerken: Die Größe und die Dezentralisierung. Was stellt man sich aber darunter nun vor? Größe soll hier einfach nur verdeutlichen, dass die Szene in den letzten Jahren immer kleiner geworden ist und es teilweise an vielversprechendem Nachwuchs fehlt. Immer mehr VXer betreiben das Coden von Viren nur als ein nebensächliches Hobby und bleiben auch nur kurz in der Szene, das schafft einen ständigen Wandel der Konstellation von Gruppen und Autoren. Ein enormes Problem mittlerweile, da es kaum Gruppen beziehungsweise Autoren gibt, welche über viele Jahre in der Szene bleiben. Ständig kommen Neue hinzu und Alte verschwinden. Schätzungsweise gibt es derzeit etwa fünfzig VXer weltweit, wobei davon nicht alle gleichermaßen aktiv sind. Das zweite Problem, welches gewissermaßen aus dem ersten resultiert ist die Dezentralisierung der Szene. Es ist ganz einfach zu erklären: Jede neue Gruppe versucht ihr eigenes Projekt zu gründen und ruft etwa ein Forum ins Leben, viele dieser Foren sterben schnell wieder aufgrund der Inaktivität der Benutzer. Anstatt also alle an einem Strang zu ziehen, versucht jeder seinen Weg alleine zu gehen. Zwar gibt es zentrale Anlaufstellen, wie etwa vx.netlux.org, doch sind diese eher die Ausnahme. So gibt es derzeit etwa eine handvoll Foren in der Szene, das wohl aktivste ist das von EOF-Project, welches unter vx.eof-project.net zu erreichen ist. Meiner Meinung nach sollten VXer mehr zusammenarbeiten, anstatt *gegeneinander.* Gerade in einer solch kleinen Szene würde dies eine gewisse Stabilität schaffen. Wie die Entwicklung weitergehen wird, wird sich in den nächsten Monaten und Jahren zeigen. ●



VX – The Virus Underground

(27th Dec. 2007 at the 24c3 in Berlin/Germany)

About me

Name: Marcell Dietl (SkyOut)
Website: www.smash-the-stack.net
Email: skyout@smash-the-stack.net

Age: 18 years
City: Wiesbaden/Germany
Company: MK Mediaconcept (www.mymk.de)
Characteristics: Gothic, Social Engineer, Autodidact
Addicted to: Cecurity, Cigarettes and Coffee (CCC)

The term VX

- Also stands for a gas with the chemical structure $C_{11}H_{26}NO_2PS$
- In our case it means Virus Coding
- The term has been made in the early days of the scene
- VXers are the Virus Coders and groups are named VX Groups

Groups of the scene

- 29A (www.29a.net)
- rRlf (www.rrlf.de.vu)
- DoomRiderz (www.doomriderz.co.nr)
- Purgatory (www.purgatory.net.tf)
- F-13 Labs (www.f13-labs.net)
- EOF-Project (www.eof-project.net)
- NE365 (wwww.vxer.cn)

The ideology behind it

- People can be divided into several categories
- Criminals making viruses to earn money
- Hobbyists, that only code some viruses for fun and leave again
- Ideologists, that stay in the scene for years
- WhiteHats, who code without the intention to harm anyone
- BlackHats coding viruses and spreading them

History, Present and Future

- History: Viruses were made to get attention, mostly spread by floppy disks, the first worms fort he Windows OS came out
- Present: Many worms are coded to build up botnets and earn money, cross-platform malware as a new trend, more malware for *nix based operating systems, much Spyware and Adware
- Future: Much more mobile device malware

Cross-Platform malware

- Several techniques to build a virus, that works cross-platform
- Macroviruses for Office Suites like MS Office or OpenOffice
- .NET (Mono) viruses
- Scripting languages, that normaly have an interpreter for different platforms
- Most difficult: Coding a virus in a LowLevel language, that changes ist behavior at start, example: Winux

Typical spreading techniques of modern viruses

- Floppy disks (not used anymore)
- Autostart function in CDs or DVDs
- USB flash drives with autostart functionality
- P2P Networks
- Sharehosters like Rapidshare (In forums and blogs the virus author links to the file and makes the users interested in downloading and executing it)
- Email

- Bluetooth
- IRC (Over the DCC function)
- Instant messenger like ICQ, MSN etc.
- Network shares, that are writable by others in the network
- Warez infected with malware
- Exploits automatically attacking a common weakness

Types of reproduction

- Appender = The virus puts its code at the end of the original file and does a jump to this part at the start
- Prepender = The virus puts its code at the beginning of the original file and starts this code and after a little delay the original code gets executed
- Overwriter = The virus overwrites the whole file with its own code

Types of payloads

- Payload = The routine the virus starts apart from reproduction
- Conspicuous payload = The user shall realize, that he got infected with a virus, for example with a message box
- Inconscpicuous payload = The user shall NOT realize, that he got infected, for example by putting the whole OS into a VM
- Polymorphism/Metamorphism = The virus code is not static and changes during reproduction
- Anti-Debugging features

Types of malware

- Virus = Self-reproducing code starting with a host file
- Worm = Malware with the ability to automatically spread
- Trojan Horse = A program, that acts as a normal application and starts the evil code silently
- Hoax = Malware, that does not harm any system as it is only a joke

Ways of communication between VXers

- Central file servers to store source code and sometimes binary viruses, for example vx.netlux.org as a central website for viruses
- Websites (see the section about the groups of the scene)
- Emails and instant messenger used for personal information exchange
- IRC channels used for public information exchange (#eof-project, #virus, #vir, #vxers, #vx-lab @undernet.org)
- Electronical magazines (E-Zines) coming out once a year by a group

Connection between the VX scene and AV (Anti Virus) companies

- It is a continuous fight
- Both groups are observing the other side, for example AV companies blogging about the VX scene
- Some AVers even tried to get in touch with the scene to get internal information and bring some VXers into jail
- In best case the connection between both groups works as follows: VXers coding a virus and sending it to Avers -> AV companies are doing an anylsis and can protect their customers -> VXers have some like a trophy

Used programming languages

- .NET languages are getting more and more popular
- Programming languages for Windows are still the most used ones, for example Visual Basic
- Many new viruses are coded in scripting languages like PHP
- Assembler is still the language, that VXers respect most

Problems of the scene

- Very small scene, only about 50 more or less active people
- Many virus coders only code some viruses and leave the scene again (see above -> Hobbyists), that leads to no continuity

- Decentralization, which means, that every groups tries to do their own thing instead of working together
- The whole community is based on a few websites and a few big VX hosters

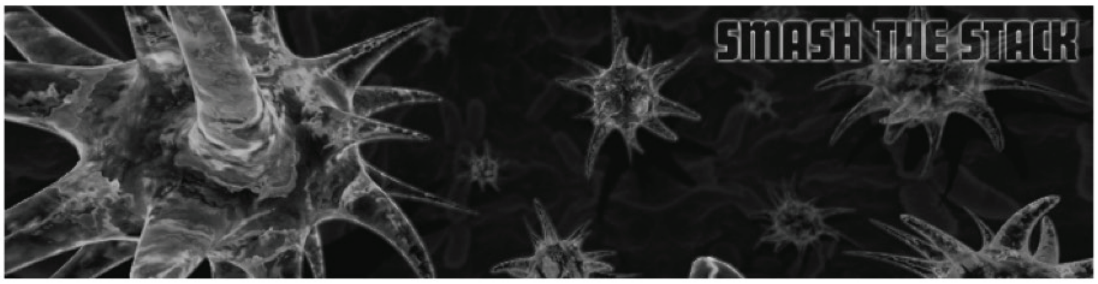
Social Engineering and VX

- Social Engineering is needed for Worms to spread
- Examples are texts in Emails to make the user click on the attachement or good messages while spreading over an instant messenger

CONCLUSION

Most VXers are not coding viruses to harm anything or anybody. For them it is a way to express their feelings and ideas in a technological and often misunderstood way. That is why most VXers do not make their viruses available in binary format and only upload the source code to show it to others. Doing so they want to conduct a knowledge exchange between each other without destroying something. So in relation to Hacking: VX can be considered playing with technology and finding out new ways of coding, that have not been there before, therefore it is hacking and most VXers can be considered WhiteHats as a conclusion of this paper!

Thanks for your attention – SkyOut



Society
lecture
2007-12-29 14:00
Saal 2
de

Markus Schneider

Wahlchaos

Paradoxien des deutschen Wahlsystems

Wahlchaos beschäftigt sich mit Wahlverfahren aus mathematischer und politischer Sicht. So wurden die Wahlen von 1998, 2002 und 2005 betrachtet und a-postpriori manipuliert und ihre Auswirkungen diskutiert.

Wir haben mit "Stimmstörungstheorie der Bundestagswahl" verschiedene Szenarien betrachtet und einige Paradoxien unter die Lupe genommen. Genauer werden Themen wie Zuteilungsverfahren, Überhangmandate, Erst- und Zweitstimmen, Wahlkreisreorganisation betrachtet.Außerdem wird die Frage analysiert, wo und wie viele Stimmen man ändern muss, um einen Patt bei der Regierungsbildung zu erreichen.

http://univis.uni-magdeburg.de/form?__s=2&dsc=anew/lecture_view&lvs=fgse/ipw/zentr/psy_0&anonymous=1&founds=fgse/ipw/zentr/psy_0,fma/iag/zentr/comput,/linear,/mab,/oberse&nosearch=1&ref=main&sem=2006s&__e=

Seite des Seminars aus dem Universitätsinformationssystem

Wahlchaos – Paradoxien des deutschen Wahlsystems

Thomas Rehn Markus Schneider

24C3: 27.-30. Dezember 2007

Das komplexe deutsche Bundestagswahlsystem weist einige Besonderheiten auf. In dieser Arbeit wird der Einfluss von Wählerstimmen auf Überhangmandaten untersucht. Zudem wird auf Wahlparadoxien und alternative Zuteilungsverfahren eingegangen.

1 Einführung und Motivation

Die Wahl zum 16. Deutschen Bundestag im Jahr 2005 führte einer größeren Öffentlichkeit vor Augen, welche ungeahnte Komplexität im deutschen Wahlverfahren zum Bundestag steckt, die mitunter zu paradoxen Situationen führt. Zur Erinnerung: im Wahlkreis 160 (Dresden I) fand am 2. Oktober 2005 zwei Wochen nach der Bundestagswahl im übrigen Bundesgebiet unter besonderer Medienbeobachtung eine Nachwahl statt. Haette bei dieser die CDU *mehr* als 41226 Zweitstimmen bekommen, hätte sie einen Sitz im Bundestag verloren. Eine tatsächliche Konsequenz des Urnengangs in Sachsens war unter anderem eine Verschiebung eines Sitzes der CDU in Nordrhein-Westfalen zur CDU im Saarland. Diese beiden Phänomene sind zwei Eigenschaften des deutschen Wahlsystems geschuldet: Überhangmandate und das Hare-Niemeyer-Sitzzuteilungsverfahren.

1.1 Juristische Grundlagen

Die Grundmerkmale der Wahl zum deutschen Bundestag – Allgemeinheit, Unmittelbarkeit, Freiheit, Gleichheit und Geheimheit – sind in den Artikeln 38ff des Grundgesetzes verankert. Alle weiteren Details zur Wahl sind im Bundeswahlgesetz (BWahlG oder BWG) geregelt. Dort ist auch die Art des Sitzzuteilungsverfahrens bestimmt, seit 1987 Hare-Niemeyer.

1.2 Überhangmandat

Von den im Bundestag zu vergebenden Sitzen werden eine Hälfte über Direktmandate und die andere Hälfte über Landeslisten der Parteien vergeben. Grundsätzlich hat ein Wähler eine Erststimme und eine Zweitstimme zur Verfügung. Mit der Erststimme wählt er das Direktmandat seines Wahlkreises (relative Mehrheit). Die Zweitstimme wählt eine Partei für die Mandatsverteilung.

Unter Berücksichtigung der Sperr-(5-Prozent)- und der Grundmandatsklausel werden zunächst die insgesamt zu vergebenden (zur Zeit 598) Mandate per Verteilungsverfahren an Hand des Zweitstimmenergebnisses auf die Parteien umgelegt (Oberverteilung). Innerhalb jeder Partei findet dann eine Verteilung der Mandate aus der Oberverteilung an Hand des Zweitstimmenergebnisses in den Bundesländern auf die jeweiligen Bundesländer statt (Unterverteilung).

Wenn hier in einem Bundesland eine Partei weniger Mandate zugeteilt bekommt als ihr durch die direkt gewählten Direktmandate zustehen, wird diese Differenz als Überhangmandat vergeben.

1.3 Zuteilungsverfahren

1.3.1 Grundlagen

Um bei einer Verhältniswahl Wählerstimmen in Mandate übersetzen zu können, kommt ein Zuteilungsverfahren zum Einsatz, das die in der Regel nicht geradzahligen Anteile auf die zu vergebenden Sitze verteilt. Man unterscheidet zwischen Quotenverfahren wie dem Hare-Niemeyer-Verfahren, das bei der Bundestagswahl zum Einsatz kommt, und Divisorverfahren wie Sainte-Laguë- oder D'Hondt-Verfahren (angewandt bei Bundestagswahlen bis einschließlich 1983).

Sei $Q := \frac{Parteistimmenzahl}{Gesamtstimmenzahl} \cdot Gesamtsitzzahl$ die sogenannte Quote einer Partei. Dann werden beim Hare-Niemeyer-Verfahren jeder Partei sicher $\lfloor Q \rfloor$ Sitze zugeteilt, die verbleibenden werden nach absteigenden Resten $Q - \lfloor Q \rfloor$ an die Parteien vergeben.

Bei Divisorverfahren werden die Anzahlen der auf eine Partei entfallenen Stimmen S_P sukzessive durch Elemente einer Zahlenfolge $N_1, N_2, N_3, \ldots$ geteilt. Die sich ergebenden Höchstzahlen genannten Quotienten $\frac{S_P}{N_i}$ werden absteigend der Größe nach sortiert; in Reihenfolge dieser Zahlen werden die zu vergebenden Mandate auf die Parteien aufgeteilt. Divisorverfahren unterscheiden sich in der Wahl der Werte N_i. Für D'Hondt ist $N_i = i$, für Sainte-Laguë $N_i = 2i - 1$.

1.3.2 Gütekriterien

Für die Bewertung von Zuteilungsverfahren gibt es drei Kriterien:

- *Quotenbedingung* Die tatsächliche Mandatszahl M einer Partei weicht nie mehr um 1 von ihrem Idealanspruch Q (s.o.) ab: $|Q - M| \leq 1$
- *Hausmonotonie* Eine Vergrößerung der Gesamtzahl an zu vergebenden Sitzen führt nie zur Verringerung der Sitzzahl einer Partei.

2

- *Stimmenmonotonie* Ein Stimmenzuwachs einer Partei führt nie zur Mandatsverschiebung zwischen anderen Parteien.

Gemäß dem Unmöglichkeitssatz von Balinski und Young [1] kann es kein Zuteilungsverfahren geben, dass alle drei Bedingungen gleichzeitig erfüllt. Quotenverfahren erfüllen nur die Quotenbedingung, Divisorverfahren genügen nur der Monotonie.

2 Methoden

Um Rechnungen und Simulationen von Bundestagswahlen durchführen zu können wurde ein modulares Java-Programm implementiert, das drei verschiedene Zuteilungsverfahren implementiert (D'Hondt, Hare-Niemeyer, Sainte-Laguë). Wahlergebnisse können aus CSV-Dateien importiert werden, deren Format sich an den Veröffentlichungen des Bundeswahlleiters orientiert. Bei der Berechnung der Sitzverteilung werden auch die Sperr- und die Grundmandatsklausel berücksichtigt. Zur Erforschung der Robustheit des deutschen Bundestagswahlsystems wurden verschiedene Störungsklassen implementiert, die auf Wahlkreisebene die Erst- und Zweitstimmenergebnisse der Parteien verändern. Störungen, die im Code realisiert wurden, sind:

- singuläre Störungen auf Wahlkreisebene zur Untersuchung von Paradoxien
- Störungen auf Länderebene, bspw. zur Untersuchung des Einflusses von Wahlbeteiligung

3 Ergebnisse

3.1 Hare-Niemeyer-Verfahren und Überhangmandate

Wie schon im einführenden Beispiel zum Dresdner Wahlkreis 2005 gezeigt, ist das seit 1987 bei Bundestagswahlen angewandte Hare-Niemeyer-Verfahren nicht konsistent. Das Auftreten einer solchen Paradoxie war kein Einzelfall. Ein ähnliches Szenario gab es bei der Wahl 2002 im Vogtlandkreis.

- Es gewinnt die CDU einen Sitz, wenn sie selbst 3750 Zweitstimmen verliert.
- Gewinnt die SPD 1500 Erststimmen, so verliert die CDU einen Sitz.

Diese Sitzverschiebung rührt von einem Überhangmandat in Sachsen her, das gewonnen oder verloren wird.

Legt man das Sainte-Laguë-Zuteilungsverfahren zu Grunde, blieibt im ersten Fall die Sitzverteilung unveraendert, auch unter anderen größeren negativen Veränderungen der Zweitstimmenzahl. Der zweite Fall ist alleine eine Sache von Überhangmandaten und tritt auch bei anderen Zuteilungsverfahren auf.

Dass die Wahlbeteiligung sich auf die Überhangsmandate auswirkt kann man deutlich sehen, wenn man sie drastisch verändert. Halbiert man im bevölkerungsreichsten

Land Nordrhein-Westfalen die Wahlbeteiligung, ergeben sich deutliche Verschiebungen bei der Sitzverteilung und den Überhangmandaten. Das konkrete Ergebnis ist in Tabelle 1 zu sehen, die SPD erhält alleine in NRW 10 Überhangmandate zusätzlich.

	Gesamt	SPD	CDU	CSU	B90/G	FDP	Linke
Mandate original	614	222	180	46	51	61	54
Mandate gestört	617	226	170	52	51	61	57
Überhang original	16	9	7	–	–	–	–
Überhang gestört	19	17	2	–	–	–	–

Tabelle 1: Bundestagswahl 2005: Wahlbeteiligung und Überhangmandate

3.2 Wahlkampfanalyse: kleine Parteien

Die kleine Parteien malen sich meistens keine großen Chancen aus, viele Wahlkreise über die Erststimme zu gewinnen. Deshalb machen sie gezielt Zweitstimme-Werbung. Es stellt sich die Frage, ob dies auch Nutzen bringt und ob das Zweistimmensystem überhaupt einen Unterschied macht. Hierfür haben wir die gleiche Stimmverteilung der Erststimme auch auf die Zweitstimme sowie die Umkehrung für die Wahlen 1998 und 2005 angewendet.

	Gesamt	SPD	CDU	CSU	B90/G	FDP	Linke
Mandate original	614	222	180	46	51	61	54
Mandate gestört	604	262	219	55	36	32	–
Überhang original	16	9	7	–	–	–	–
Überhang gestört	6	5	1	–	–	–	–

Tabelle 2: Bundestagswahl 2005: Zweitstimmen wie Erststimmen

Wie man in Tabelle 2 deutlich sieht, schrumpfen die Fraktionen von Grünen und FDP zusammen und die Linkspartei zieht gar nicht in den Bundestag ein.

	Gesamt	SPD	CDU	CSU	B90/G	FDP	Linke
Mandate original	614	222	180	46	51	61	54
Mandate gestört	616	225	179	46	51	61	54
Überhang original	16	9	7	–	–	–	–
Überhang gestört	18	12	6	–	–	–	–

Tabelle 3: Bundestagswahl 2005: Erststimmen wie Zweitstimmen

Umgekehrt (Tabelle 3) verschieben sich nur ein paar Überhangmandate bei SPD und CDU, die anderen Parteien sind nicht betroffen.

Ein ähnliches Bild ergibt sich in Tabelle 4 für das Jahr 1998: FDP und PDS zögen gar nicht in den Bundestag ein, die SPD erhielte die absolute Mehrheit an Sitzen.

4

	Gesamt	SPD	CDU	CSU	B90/G	FDP	Linke
Mandate original	669	298	198	47	47	43	36
Mandate gestört	663	332	240	54	37	–	–
Überhang original	13	13	–	–	–	–	–
Überhang gestört	7	7	–	–	–	–	–

Tabelle 4: Bundestagswahl 1998: Zweitstimmen wie Erststimmen

	Gesamt	SPD	CDU	CSU	B90/G	FDP	Linke
Mandate original	614	222	180	46	51	61	54
Mandate gestört	613	224	178	46	50	61	54
Überhang original	16	9	7	–	–	–	–
Überhang gestört	15	10	5	–	–	–	–

Tabelle 5: Bundestagswahl 2005: Patt mit Hare-Niemeyer

Es lässt sich deutlich erkennen die Wähler bei Erststimmen ein anderes Verhalten als bei Zweitstimmen an den Tag legen, das zweigeteilte Wahlsystem zeigt Wirkung.

3.3 Manipulation zum Patt

Die Wahl 2005 ist sehr knapp ausgefallen. Es gab nur einen Unterschied von vier Mandaten zwischen den beiden großen Fraktionen SPD und CDU/CSU, umgerechnet $\frac{4}{614} \cdot 47.194.062 \approx 310.000$ Stimmen. Es stellt sich die Frage, ob man durch gezielte Manipulation eventuell weniger Stimmen benötigt um diese vier Sitze zu verschieben. Wir suchten also nach Wahlkreisen mit knappem Direktmandate. Außerdem untersuchten wir, welche Überhangsmandate leicht zu kippen sind. Hierzu verwendeten wir kaskadierte probabilistische Stimmenstörungen in Wahlkreisen. Am Ende genügte die folgende a-posteriori Störung mit 70.500 Stimmen:

- Chemnitz: +3500 CDU, +3500 SPD Zweitstimmen
- Altenburger Land: +2500 SPD Erststimmen
- Heidelberg: +1000 SPD Erststimmen
- z.B. Saarland: +30.000 SPD, -30.000 CDU Zweitstimmen

Das Ergebnis ist in Tabelle 5 zu sehen.

3.4 Wahlkreise und Manipulationsfähigkeit

Nach den vorherigen Schlaglichtern auf bestimmte Phänomene wollen wir an dieser Stelle die Robustheit des Wahlsystems gegenüber kleineren Änderungen der Wählerstimmen untersuchen. Dabei störten wir einzeln sukzessive die Erst- und Zweitstimmenergebnisse der Wahlkreise mit einer gewissen Fluktuation an Wählerstimmen. Beginnend bei sehr kleinen Störungen, die nur 0,01% der Stimmen verändern steigerten

wir das Ausmaß der Fluktuation bis sich an der Sitzverteilung im Bundestag etwas verändert.

Die Ergebnisse der für die Bundestagswahlen 2002 und 2005 durchgeführten Analyse verhalten sich indifferent. Es gibt drei typische Fälle:

- Ein knappes Direktmandat in einem Wahlkreis kippt und zieht ein Überhangmandat nach sich.
- Die Sitzzuteilung einer Landesliste ist knapp, so dass Änderungen am Zweitstimmenanteil im ganzen Bundesland eine Veränderung eines Überhangmandates nach sich ziehen.
- Wahlkreis und Bundesland sind stabil gegenüber massiven zufälligen Fluktuationen, es gibt keine Änderung an der Sitzverteilung.

Neben dieser Einteilung in drei erkennbaren Klassen konnten keine auffälligigen Tendenzen eines Bundeslandes oder eines Wahlkreises störanfällig zu sein, gefunden werden.

4 Zusammenfassung und Ausblick

Wir haben eine Reihe von Besonderheiten des deutschen Wahlsystems gesehen. So kann es passieren das eine vergleichsweise kleine Störung eine Sitzveränderung hervorruft. Die Eigenschaften des Hare-Niemeyer-Verfahren können zu paradoxen Effekten führen. Da unsere Analyse nur a-posteriori ist, kann man kaum konkrete Rückschlüsse für die nächste Wahl ziehen, aber dennoch wird es, spätestens 2009, interessant sein, die Wahl mit den gleichen Methoden zu untersuchen.

Da nicht alle drei Kriterien von einem Zuteilungsverfahren erfüllt werden können, muss es aus politscher Sicht eine Entscheidung geben, welches als das wichtigere angesehen wird und dann erst kann die Mathematik über das am besten geeignete Verfahren entscheiden.

Bezüglich der Untersuchungen zur Störanfälligkeit wäre es interessant ausgiebiger mit statistischen Mitteln zu untersuchen, ob es eine langfristig Tendenz von Wahlkreisen oder Bundesländern gibt, bei kleinen Stimmfluktuationen eine Mandatsveränderung hervorzurufen. Da die Wahlkreise jedoch einer ständigen Restrukturierung (zuletzt 2002) unterliegen, sind diesem Ansatz auch Grenzen gesetzt.

Eins ist aber sehr deutlich geworden, jede Stimme zählt. Auch Nichtwähler haben einen Einfluss, den man nicht unterschätzen sollte, keiner kann sich seiner politischen Verantwortung entziehen.

Literatur

[1] Balinski und Young: Fair Representation: Meeting the Ideal of One Man, One Vote. Yale University Press, New Haven, London, 1982

Volldampf vorraus!
24. Chaos Communication Congress

Veranstaltungen

Tag 1 - Saal 1

Tim Pritlove — 2007-12-27 10:30 — Saal 1 — en — lecture — Community

Opening Event

Welcome Keynote

Welcome to the Congress!

SkyTee, Jens Ohlig, Ingo Schwitters, Sebastian Velke — 2007-12-27 11:30 — Saal 1 — en — lecture — Making

Steam-Powered Telegraphy

Wherein a League of Telextraordinary Gentlemen present the marvel of Telex on the global Internet -- driven by a steam engine

We have built and modified a steam-powered Telex machine and connected it to the new-fangled invention for modern telegraphy known as "the Internet". We will present this steampunkish invention in form of a lecture, thus hoping to enlighten interested ladies and gentlemen on the principles of steam engine physics, 5-bit Baudot encoding, and historic telegraphy in general.

Constanze Kurz, Andreas Bogk — 2007-12-27 12:45 — Saal 1 — de — lecture — Society

Der Bundestrojaner

Die Wahrheit haben wir auch nicht, aber gute Mythen

Der Bundestrojaner wird von der politischer, juristischer und technischer Seite beleuchtet.

Julius Mittenzwei, Erdgeist — 2007-12-27 14:00 — Saal 1 — de — lecture — Society

TOR

Rop Gonggrijp — 2007-12-27 16:00 — Saal 1 — en — lecture — Society

It was a bad idea anyway...

The demise of electronic voting in The Netherlands

2007 has been yet another a turbulent year in The Netherlands with regard to electronic voting. If you remember the presentation at 23c3, 2006 saw the emergence of a campaign against the use of non-auditable voting systems.

Frank Rieger, Constanze Kurz — 2007-12-27 17:15 — Saal 1 — de — lecture — Society

NEDAP-Wahlcomputer in Deutschland

Wir bringen Euch auf den neuesten Stand,
was den Einsatz der NEDAP-Wahlcomputer in Deutschland betrifft.

Anna H. — 2007-12-27 18:30 — Saal 1 — de — lecture — Society

Was ist eigentlich Terrorismus?

Und wer terrorisiert hier eigentlich wen?

ladyada — 2007-12-27 20:30 — Saal 1 — en — Culture

Design Noir

The seedy underbelly of electronic engineering

In contemporary Western society, electronic devices are becoming so prevalent that many people find themselves surrounded by technologies they find frustrating or annoying. The electronics industry has little incentive to address this complaint; I designed two counter-technologies to help people defend their personal space from unwanted electronic intrusion. Both devices were designed and prototyped with reference to the culture-jamming "Design Noir"; philosophy. The first is a pair of glasses that darken whenever a television is in view. The second is low-power RF jammer capable of preventing cell phones or similarly intrusive wireless devices from operating within a user's personal space. By building functional prototypes that reflect equal consideration of technical and social issues, I identify three attributes of Noir products: Personal empowerment, participation in a critical discourse, and subversion.
http://www.ladyada.net/make/wavebubble/
http://www.ladyada.net/make/tvbgone/
http://www.ladyada.net/pub/research.html

Ilja 2007-12-27 23:00 Saal 1 en lecture Hacking

A collection of random things

look what I found under the carpet

random things I'll cover - using oob data to bypass ids - /dev/[k]mem race conditions in suids- tcp fuzzer that goes beyond the 3-way handshake- ...

Johannes Grenzfurthner 2007-12-27 00:30 Saal 1 en lecture Culture

"I can count every star in the heavens above

Computers as a thankful subject in pop music

A talk (with examples) by monochrom, presented by Johannes Grenzfurthner

Tag 1 - Saal 2

Rose White 2007-12-27 11:30 Saal 2 en lecture Community

The Role of Brilliant Deviants in the Liberalization of Society

How People Like Us Make People Like Them Accept Us

I'm planning to look at how hackers and other "folks like us" get the "real world" to let us be crazy deviants, and continue to pay us anyway. Clearly not everyone is able to do this -- hence the sort of person who says, "I'd love to [go to Burning Man] [blow things up] [dress eccentrically]" but never does any of it. But some of us *are* able to get the world to play along, and I am looking at that from a sociological point of view.

Antoine Drouin, martinmm 2007-12-27 12:45 Saal 2 en lecture Making

Paparazzi - The Free Autopilot

Build your own UAV

Autonomous unmanned aerial vehicles are becoming more and more popular as suitable electronics and sensors are available and affordable. This talk will describe Paparazzi, a complete system enabling you to build and control your own UAV.

http://paparazzi.nongnu.org/ Paparazzi Project Page

Leon Hempel 2007-12-27 16:00 Saal 2 de lecture Science

Verteilte Sicherheit

Zur Ordnung der Überwachung

Die Integration visueller Überwachungssysteme sowie die Verknüpfung militärischer und nicht-militärischer Verwendungen der Technologien verläuft schleichend, aber stetig.

Victor Muñoz 2007-12-27 17:15 Saal 2 en lecture Hacking

AES: side-channel attacks for the masses

AES (Rijndael) has been proven very secure and resistant to cryptanalysis, there are not known weakness on AES yet. But there are practical ways to break weak security systems that rely on AES.

http://www.ingenieria-inversa.cl/AES02.pdf AES: side-channel attacks for the masses

Cristian Yxen, Erdgeist, Denis Ahrens 2007-12-27 18:30 Saal 2 de lecture Hacking

Trecker fahrn

Vom Gefühl, einen offenen Bittorrent Tracker zu fahren

Bittorrent aus der Sicht derer, die die Infrastruktur machen und natürlich auch selber nutzen.

http://opentracker.blogs.h3q.com/ Das opentracker Blog
http://erdgeist.org/arts/software/opentracker Opentracker Projektseite

Maarten Van Horenbeeck 2007-12-27 20:30 Saal 2 en lecture Hacking

Crouching Powerpoint, Hidden Trojan

An analysis of targeted attacks from 2005 to 2007

Targeted trojan attacks first attracted attention in early 2005, when the UK NISCC warned of their wide spread use in attacks on UK national infrastructure. Incidents such as "Titan Rain" and the compromise of US Department of State computer systems have increased their profile in the last two years. This presentation will consist of hard, technical information on attacks in the form of a case study of an actual attack ongoing since 2005. It covers exploitation techniques, draws general conclusions on attack methodologies and focuses on how to defend against the dark arts.

http://www.daemon.be/maarten/targetedattacks.html A brief introduction to targeted attacks

Daniel Otte, Sören Heisrath 2007-12-27 21:45 Saal 2 de lecture Hacking

AnonAccess

Ein anonymes Zugangskontrollsystem

AnonAccess ist ein elektronisches System, welches anonymen Zugang nicht nur zu Hackerspaces ermöglicht.

http://www.das-labor.org/wiki/AnonAccess AnonAccess im Labor wiki

Jeroen Massar 2007-12-27 23:00 Saal 2 en lecture Hacking

IPv6: Everywhere they don't want it

Global connectivity even in the places that you are not supposed to have it

This talk will discuss a new feature in AICCU which allows one to have IPv6 virtually everywhere, including most places where a lot of network operators will not want to have it.

http://www.sixxs.net/tools/aiccu/ AICCU - Automatic IPv6 Connectivity Client Utility
http://www.sixxs.net/tools/ayiya/ AYIYA - Anything In Anything
http://www.sixxs.net/ SixXS - IPv6 Tunnel Broker and IPv6 Deployment
http://unfix.org/jeroen/ Jeroen Massar's homepage

Tag 1- Saal 3

Gregers Petersen 2007-12-27 11:30 Saal 3 en lecture Society

Freifunkerei

And a Do-It-Yourself society against the state.

The term Freifunk Firmware has found a place on the shelf's in the life of numerous people. It has become an immense knot of activities, not just sitting silently like a dusty heirloom. "Freifunkerei"; has become an example of how DIY-cultures can act and re-create alternatives in a world which seems both confronted and abandoned by the state.

Mark Vogelsberger 2007-12-27 12:45 Saal 3 en lecture Science

Simulating the Universe on Supercomputers

The evolution of cosmic structure

The evolution of structure in the Universe is one of the hottest topics in Cosmology and Astrophysics. In the last years the so-called Λ-CDM-model could be established also with great help of very large computer simulations. This model describes a Universe that consists mainly of dark components: 96% are made of dark energy and dark matter.

http://www.mpa-garching.mpg.de/galform/presse/ Millennium Simulation done by the MPI for Astrophysics
http://www.ucolick.org/diemand/vl/ A recent NASA's Supercomputers Simulation
http://de.wikipedia.org/wiki/Millennium-Simulation Wikipedia entry for the Millennium Simulation

Lars Weiler, Jens Ohlig 2007-12-27 14:00 Saal 3 en lecture Community

Building a Hacker Space

A Hacker Space Design Pattern Catalogue

With the help of Design Patterns we will show you how to set up your own Hacker Space. The Design Patterns are based on more than 10 years of experience with setting up and running a Hacker Space.

Arien Vijn 2007-12-27 16:00 Saal 3 en Hacking

10GE monitoring live!

How to find that special one out of millions

There are many open source tools available to do packet capturing and analysis. Virtually all networkers use these tools. However millions of packets per seconds are just too much for general-purpose hardware. This is a problem as 10 Gigabit networks allow for millions of packets per second. The obvious solution for that issue is to lower the data rates by filtering out 'uninteresting' data out before it gets processed by the general purpose computer hardware.

Nils Magnus 2007-12-27 17:15 Saal 3 de lecture Hacking

Desperate House-Hackers

How to Hack the Pfandsystem

Wie funktionieren eigentlich diese Pfandflaschenrücknahmeautomaten? Wir finden es heraus.

Mitch 2007-12-27 18:30 Saal 3 en workshop Making

Make Cool Things with Microcontrollers

Hacking with Microcontrollers

Learn how to make cool things with microcontrollers by actually making fun projects at the Congress -- blink lights, hack your brain, move objects, turn off TVs in public places -- microcontrollers can do it all. Ongoing workshops each day of the Congress.

http://www.tvbgone.com/cfe_mfaire.php Documentation for Projects
http://makezine.com/10/brainwave/ Brainwave Machine in MAKE

Thorsten Holz 2007-12-27 20:30 Saal 3 en lecture Hacking

Cybercrime 2.0

Storm Worm

Not only the Web has reached level 2.0, also attacks against computer systems have advanced in the last few months: Storm Worm, a peer-to-peer based botnet, is presumably one of the best examples of this progress. Instead of a central command & control infrastructure, Storm uses a distributed communication channel based on Kademlia / Overnet. Furthermore, the botherders use fast-flux service networks (FFSNs) to host some of the content. FFSNs use fast-changing DNS entries to build a reliable hosting infrastructure on top of compromised machines. Besides using the botnet for DDoS attacks, the attackers also send lots of spam - most often stock spam, i.e., spam messages that advertize stocks. This talk presents more information about Storm Worm and the other aspects of modern cybercrime.

http://honeynet.org/papers/ff/ Fast-Flux Service Networks
http://honeyblog.org my blog

Meike Richter 2007-12-27 21:45 Saal 3 en lecture Society

How to Reach Digital Sustainability

The Impact of Intellectual Property Rights

Happy digital world: Everything is information, and it grows by sharing. Scarcity seems to be a problem of the "meatspace". On the internet, there is space for everybody, for every activity and for every opinion. Really? This lectures explores the power of intellectual property rights and their impact on everyday (digital) life. The net as we know it is in danger. What is needed to make it stay a resource which is valuable, open and free for everybody? How could a concept of digital sustainability look like?

http://www.commonspage.net/ Blog of Meike Richter

SkyOut 2007-12-27 23:00 Saal 3 en lecture Culture

VX

The Virus Underground

The listeners will be introduced in the world of virus coding. They will understand how this can be seen as a way of expressing yourself and why it is a way of hacking. Furthermore they will get to know, which important groups, authors and viruses have been there in the last years and which are still active nowadays. Important technical terms will be explained as well as trends of the last years and the future. And more.

http://vx.netlux.org/	Virus database	http://vxchaos.official.ws/	VX File Server
http://www.smash-the-stack.net	Smash-The-Stack	http://www.freewebs.com/purgatory-vx/	Purgatory Virus Team
http://www.eof-project.net/	EOF-Project	http://vx.eof-project.net/	
http://vx.netlux.org/	VX	http://www.29a.net/	29A Labs
http://www.rrlf.de.vu/	Ready Rangers Liberation Front	http://vxchaos.official.ws/	VX CHAOS File Server
http://www.doomriderz.co.nr/	Doomriderz VX Team		

Tag 2 - Saal 1

Erik Josefsson 2007-12-28 12:45 Saal 1 en lecture Society

Data Retention and EURODAC

The Brussels Workshop

New EU legislation emphasises and in some cases creates new crimes of consumer infringement of intellectual property laws. Consumer Warnings about consumers' requirements to respect copyright could become mandatory; worse, such infringement cases could move from civil cases to criminal ones across the EU. But nowhere is there legislation either clarifying or defending consumers' rights under IP law, in our changing digital environment.

Christian Kurtsiefer, Ilja Gerhardt, Antia Lamas 2007-12-28 14:00 Saal 1 en lecture Science

Quantum Cryptography and Possible Attacks

Quantum cryptography is the oldest and best developed application of the field of quantum information science. Although it is frequently perceived as an encryption method, it is really a scheme to securely distribute correlated random numbers between the communicating parties and thus better described as quantum key distribution (QKD). Any attempt at eavesdropping from a third party is guarantied to be detected by the laws of physics (quantum mechanics) and shows up as an increased error rate in the transmission (the QBER).

http://arXiv.org/abs/0702152 A. Acin, N. Brunner, N. Gisin, S. Massar, S. Pironio, and V. Scarani, Physical Review Letters 98, 230501 (2007)
http://arxiv.org/abs/quant-ph/0606072 I. Marcikic, A. Lamas-Linares, and C. Kurtsiefer, Applied Physics Letters 89, 101122 (pages 3) (2006)
http://arxiv.org/abs/0704.3297 A. Lamas-Linares and C. Kurtsiefer, Optics Express 15, 9388 (2007)
http://quantumlah.org/ Center for Quantum Technologies, National University of Singapore

Michael Steil 2007-12-28 16:00 Saal 1 en lecture Hacking

Why Silicon-Based Security is still that hard: Deconstructing Xbox 360 Security

Console Hacking 2007

The Xbox 360 probably is the video game console with the most sophisticated security system to date. Nevertheless, is has been hacked, and now Linux can be run on it. This presentation consists of two parts.

http://www.free60.org/ Free60 Project

Constanze Kurz, Frank Rosengart, Andreas Lehner 2007-12-28 17:15 Saal 1 de lecture Community

Chaos Jahresrückblick

Ein Überblick über die Aktivitäten des Clubs 2007

Wir stellen die Aktivitäten des und Geschehnisse im Chaos Computer Club im abgelaufenen Jahr vor. Hierunter fallen sowohl die Kampagnen des CCC, die Lobbyarbeit sowie Berichte und Anekdoten von Veranstaltungen innerhalb des CCC als auch Vorträge und Konferenzen, an denen CCC-Vertreter teilgenommen haben.

FX of Phenoelit, fabs 2007-12-28 21:45 Saal 1 en lecture Hacking

Port Scanning improved

New ideas for old practices

Port-Scanning large networks can take ages. Asking yourself how muchof this time is really necessary and how much you can blame on theport-scanner, you may find yourself integrating your own scanner intothe linux-kernel. Or at least we did.

http://www.recurity-labs.com Who we are

Bre 2007-12-28 23:00 Saal 1 en lecture Making

DIY Survival

How to survive the apocalypse or a robot uprising

The apocalypse could happen any day. You're going to need things to survive and your going have to make them yourself.

Andreas Bogk, tina, Erdgeist, nibbler 2007-12-28 00:00 Saal 1 en contest Culture

Rule 34 Contest

There is porn of it.

Rule 34 says: there is porn of it. This contest will challenge the best and brightest to prove the rule under adverse circumstances in a race against the clock.

Tag 1 - Saal 2

Anoushirvan Dehghani 2007-12-28 12:45 Saal 2 de lecture Science

Absurde Mathematik

Paradoxa wider die mathematische Intuition

Ein kleiner Streifzug durch die Abgründe der Mathematik. Eigentlich ist der Mensch mit einer recht gut funktionierenden Intuition ausgerüstet. Dennoch gibt es Paradoxa, welche mathematisch vollkommen korrekt und beweisbar sind, jedoch unserer Intuition widersprechen. Der Vortrag bietet einen Streifzug durch einige dieser Paradoxa, die kurz und anschaulich erklärt werden.

Vladsharp 2007-12-28 14:00 Saal 2 en lecture Community

After C: D, libd and the Slate project

A clean slate for operating systems

We present libd, a high-level runtime for the D programming language and the Slate project, an attempt at a high-level OS and environment built upon libd, as the next major step in improving the state of programming environments and operating systems. With high-level abstractions, and sensible design, the state of implementation of open-source OSes can improve. We leverage existing kernels when implementing Slate, and put an extensive (abstraction-oriented) architecture above the kernel to present the user (or programmer) with a system they can use by having to do less to perform a specific function. Our virtual machine approach also allows for security verification on a level not seen in *nix OSes before.

http://www.slate-project.org/res/os_2_0_talk.pdf Slides

Martin 'maha" Haase 2007-12-28 16:00 Saal 2 en lecture Science

Linguistic Hacking

How to know what a text in an unknown language is about?

It is sometimes necessary to know what a text is about, even it is written in a language you don't know. This can be quite problematic, if you do not even know in what language it is written. This talk will show how it is possible to identify the language of a written text and get at least some information about the contents, in order to decide whether a specialist and which specialist is needed to know more.

Jens Kubieziel 2007-12-28 17:15 Saal 2 en lecture Hacking

To be or I2P

An introduction into anonymous communication with I2P

I2P is a message-based anonymizing network. It builds a virtual network between the communcation endpoints. This talk will introduce the technical details of I2P and show some exemplary applications.

http://www.i2p.net/ I2P website

Hannes 2007-12-28 18:30 Saal 2 en lecture Science

Automatic memory management

Why should I care about something that a computer could handle better, anyway?

Since Java is widespread, automatic memory management is a commonly used technology. There are several approaches to memory management, realtime, parallel, probabilistic algorithms. The lecture will give an overview of different algorithms and current research topics.

http://www.cs.kent.ac.uk/people/staff/rej/gc.html Richard jones GC page
http://www.ravenbrook.com/project/mps/ Memory Pool System
http://www.hpl.hp.com/personal/Hans_Boehm/gc/ Boehm GC
http://www.research.ibm.com/people/d/dfb/papers/Vechev05Derivation.pdf Derivation and Evaluation of Concurrent Collectors
http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=454 Realtime Garbage Collection
http://www.memorymanagement.org/ The Memory Management Reference

Rainer Fromm, Frank Rosengart 2007-12-28 20:30 Saal 2 de podium Society

Spiel, Freude, Eierkuchen?

Die Gamerszene und ihre Reaktion auf kritische Berichterstattung

Der Journalist Rainer Fromm berichtet über seine Erfahrungen mit der Gamerszene, mit Filmbeispielen und anschließender Diskussion.

http://www.zdf.de/ZDFde/inhalt/26/0,1872,2285338,00.html ZDF Frontal21: Gewalt ohne Grenzen

lucy 2007-12-28 21:45 Saal 2 en lecture Hacking

Inside the Mac OS X Kernel

Debunking Mac OS Myths

Many buzzwords are associated with Mac OS X: Mach kernel, microkernel, FreeBSD kernel, C++, 64 bit, UNIX... and while all of these apply in some way, "XNU", the Mac OS X kernel is neither Mach, nor FreeBSD-based, it's not a microkernel, it's not written in C++ and it's not 64 bit - but it is UNIX... but just since recently.

Ralph Kusserow, Christine Ketzer, Yvette Krause 2007-12-28 23:00 Saal 2 de movie Society

Das Panoptische Prinzip - Filme über die Zeit nach der Privatsphäre

Ergebnisse des Minutenfilmwettbewerbs des C4 und des Kölner Filmhauses

In den letzten Jahrennicht zuletzt seit dem 11. Septemberist es zu einem Abbau von Bürgerrechten und einer immer umfassender werdenden Überwachung seitens des Staates, aber auch der Wirtschaft gekommen. Erkennungsdienstliche Verfahren wie z. B. die Abnahme von Fingerabdrücken oder andere biometrische Verfahren, treffen zunehmend auch Normalbürger. Das rechtsstaatlich garantierte Paradigma der Unschuldsvermutung wird demontiert: Jeder ist potenziell verdächtig.

http://www.panoptisches-prinzip.de/ Das panoptische Prinzip

Tag 2 - Saal 3

Bianca Drefahl 2007-12-28 11:30 Saal 3 de lecture Science

Computersimulationen als Prognose- und Planungsinstrumente

Grenzen und Möglichkeiten kalkulierbarer Zukünfte und dynamischer Planspiele

Mit den computertechnologischen Entwicklungen seit Mitte des 20. Jahrhunderts rückte ein alter Traum der Menschheit in greifbare Reichweite: kalkulierbare Zukünfte. Die stetige Steigerung an Rechnergeschwindigkeit, Speicherplatz und Verarbeitungspotential erlaubt es, am Computer Experimente virtuell mit quasi-empirischen Charakter ablaufen zu lassen und visuell eindrucksvoll zu inszenieren.

Stefan Strigler, BeF 2007-12-28 12:45 Saal 3 de lecture Hacking

Konzeptionelle Einführung in Erlang

A jump-start into the world of concurrent programming

Simon Wunderlich, Marek 2007-12-28 14:00 Saal 3 en lecture Hacking

Wireless Kernel Tweaking

or how B.A.T.M.A.N. learned to fly

Kernel hacking definitely is the queen of coding but in order to bring mesh routing that one vital step further we had to conquer this, for us, unchartered territory. Working in the kernel itself is a tough and difficult task to manage, but the results and effectivity to be gained justify the long and hard road to success. We took on the mission to go down that road and the result is B.A.T.M.A.N. advanced which is a kernel land implementation of the B.A.T.M.A.N. mesh routing protocol specifically designed to manage Wireless MANs.

http://www.open-mesh.net www.open-mesh.net

Markus Beckedahl 2007-12-28 16:00 Saal 3 de lecture Society

23 ways to fight for your rights

Wie man sich selbst mit den eigenen Stärken für unsere Bürgerrechte einsetzen kann

Bürgerrechtsabbau steht auf der Tagesordnung. Bei der Vielzahl an Vorhaben und Gesetzesinitiativen haben viele mittlerweile das Gefühl, dass sich politisches Engagieren nicht mehr lohnt.

http://www.netzpolitik.org netzpolitik.org

Peter Molnar, Roland Lezuo 2007-12-28 17:15 Saal 3 en lecture Hacking

Just in Time compilers - breaking a VM

Practical VM exploiting based on CACAO

We will present state of the art JIT compiler design based on CACAO, a GPL licensed multiplatform Java VM. After explaining the basics of code generation, we will focus on "problematic" instructions, and point to possible ways to exploit stuff.

http://cacaojvm.org/ cacaojvm.org

Florian 2007-12-28 18:30 Saal 3 en lecture Science

Modelling Infectious Diseases in Virtual Realities

The "corrupted blood" plague of WoW from an epidemiological perspective

World of Warcraft is currently one of the most successful and complex virtual realities. Apart from gaming, it simulates personality types, social structures and a whole range of group dynamics.

http://www.burckhardt.de/24c3_modelling_infdis_in_vr.pdf conference talk

Raoul "Nobody" Chiesa, mayhem 2007-12-28 20:30 Saal 3 en lecture Hacking

Hacking SCADA

how to own critical infrastructures

SCADA acronym stand for "Supervisory Control And Data Acquisition";, and it's related to industrial automation inside critical infrastructures. This talk will introduce the audience to SCADA environments and its totally different security approaches, outlining the main key differences with typical IT Security best practices. We will analyze a real world case study related to Industry. We will describe the most common security mistakes and some of the direct consequences of such mistakes to a production environment. In addition, attendees will be shown a video of real SCADA machines reacting to these attacks in the most "interesting"; of ways! :)

http://conference.hitb.org/hitbsecconf2007kl/materials/D1T2%20-%20Raoul%20Chiesa%20and%20Mayhem%20-%20Hacking%20SCADA%20-%20How%20to%20own%20Critical%20National%20Infrastructure.pdf Our slides @hitb07

Peter Fuhrmann 2007-12-28 21:45 Saal 3 en lecture Hacking

C64-DTV Hacking

Revisiting the legendary computer in a joystick

The C64-DTV is a remake of the classic homecomputer sold as a joystick-contained videogame. The talk gives an overview about the structure of the dtv, and showes different hardware and software modifications that can be done.

2) Food and Coins Available On Landing. 2007-12-28 23:00 Saal 3 en Society

Vending Machine for Crows

Saving the World, or Manufacturing Minions?

As humanity spreads its population across the globe and in ever-increasing densities we are forcing darwinian selection on all species, selecting for those which can best adapt to us. Crows are one such example of a synanthropic (human-adapted) species which has been selectively breeding for intelligence, tool use, and flexible, logical thought. This experiment attempt to autonomously train crows to pick up lost change and deposit it into a machine in exchange for peanuts.

Aside from the monetary potential ($216million USD/year in the US), this effort highlights the otherwise unexamined relationship between humanity and the species we impact. Are we simply the propegators of attempted genocide against "pest" species, or are we willing to engage synanthropic species in mutually beneficial relationships? If we can autonomously train crows to engage in tasks for us (and there is every indication we can - see www.wireless.is/crows), what will it mean for our ethical responsibilities as stewards of the planet we are busily destroying and the species who are adapting to us?

Tag 3 - Saal 1

2007-12-29 11:30 Saal 1 en lecture Society

What can we do to counter the spies?

What it was like to be recruited and work for MI5.

A presentation about the role of intelligence agencies in the current era of the unending "war on terror";, how they monitor us, the implications for our democracies, and what we can do to fight back.

Tomislav Medak, Toni Prug, Marcell Mars 2007-12-29 12:45 Saal 1 en lecture Society

Hacking ideologies, part 2:

Free Software, Free Drugs and an ethics of death

The Open Source initiative re-interpreted Free Software to include it into the neo-liberal ideology and the capitalist economy - whose aims are contrary to the FS starting axioms/freedoms. This platform will focus on ideological and political aspects of this. It will also suggest FS recovery strategies.

http://publication.nodel.org/The-Mirrors-Gonna-Steal-Your-Soul The Mirror's Gonna Steal Your Soul

http://rabelais.socialtools.net/FreeSoftware.ToniPrug.Aug2007.pdf Free Software

Rose White 2007-12-29 14:00 Saal 1 en lecture Making

The history of guerilla knitting

"Guerrilla knitting" has a couple of meanings in the knitting community - to some, it merely means knitting in public, while to others, it means creating public art by knitted means.

Frank Rieger, Ron 2007-12-29 16:00 Saal 1 de lecture Society

Die Wahrheit und was wirklich passierte

Jede Geschichte hat vier Seiten.

Jede Geschichte hat vier Seiten. Deine Seite, Ihre Seite, die Wahrheit und das, was wirklich passiert ist.

Wolfgang Wippermann 2007-12-29 17:15 Saal 1 de lecture Science

Agenten des Bösen

Verschwörungstheorien

Wolfgang Wippermann hat 2007 unter dem Titel "Agenten des Bösen" ein Buch über "Verschwörungstheorien von Luther bis heute" veröffentlicht. Darin geht es unter anderem auch um Verschwörungstheorie, die in Hackerkreisen auf Interesse stoßen (Illuminanten, 9/11...). Interessant ist seine Einordnung solcher Verschwörungstheorien in größere Zusammenhänge.

http://www.dradio.de/dkultur/sendungen/kritik/645433/ Buchkritik Agenten des Bösen (dradio)
http://www.media-mania.de/index.php?PHPSESSID=cd7e73d2ef22df76bdded374d65350ca&action=rezi&p=2&id=5770 Buchkritik Agenten des Bösen

Steven J, Murdoch 2007-12-29 18:30 Saal 1 en lecture Hacking

Relay attacks on card payment:

Keeping your enemies close

Relay attacks allow criminals to use credit or debit cards for fraudulent transactions, completely bypassing protections in today's electronic payment systems. This talk will show how using easily available electronics, it is possible to carry out such attacks. Also, we will describe techniques for improving payment systems, developed by Saar Drimer and me, in order to close this vulnerability.

http://www.cl.cam.ac.uk/sjm217/papers/usenix07bounding.pdf Academic paper
http://www.cl.cam.ac.uk/research/security/projects/banking/relay/ Summary website

FX of Phenoelit 2007-12-29 20:30 Saal 1 en lecture Community

Toying with barcodes

The line of least resistance

The talk focuses on 1D and 2D barcode applications with interference possibilities for the ordinary citizen. Ever wondered what is in these blocks of squares on postal packages, letters and tickets? Playing with them might have interesting effects, reaching from good old fun to theft and severe impact.

Florian Bischof 2007-12-29 21:45 Saal 1 de Society

Sex 2.0

Hacking Heteronormativity

Der lange Schwanz der Dating-Communities sowie die De- und Rekonstruktion von Geschlecht und sexueller Orientierung haben ungeahnte Auswirkungen auf unser Sexualleben. Ein Überblick darüber, was Sex ist, wie Dating-Communities funktionieren und wie man zu einem erfüllten Sexualleben kommen kann.

http://www2.gender.hu-berlin.de/gendermediawiki/index.php/Hauptseite Gender@Wiki

Ray 2007-12-29 23:00 Saal 1 de contest Community

Hacker Jeopardy

Die ultimative Hacker-Quizshow

Das bekannte Quizformat - aber natürlich mit Themen, die man im Fernsehen nie zu sehen bekäme.

Tag 3 - Saal 2

Jens Muecke, Sven Übelacker 2007-12-29 11:30 Saal 2 de lecture Hacking

Hamburger Wahlstift

Am 24. Februar wollte Hamburg als Pilotprojekt mit dem Digitalen Wahlstift wählen.

http://www.24-februar.de/ Werbeseite zur Wahl

jz 2007-12-29 12:45 Saal 2 en lecture Society

Distributed campaigns for promoting and defending freedom in digital societies

Sharing experience about campaigning on the political field in France

A presentation of a few successful campaigns in France lead by libre software activists for defending freedom in a digital world: bringing awareness of the politicians about the dangers of the EUCD transposition and DRM, and their economical, social and political impact and influencing the candidates at a presidential election to talk about Libre Software, software patents, DRM, etc. How did we do that? What have we learned? Maybe for political action _too_, sharing is a way of just doing it better.

http://www.april.org/ APRIL, french non-profit organization for promoting and defending libre software
http://www.eucd.info/ Campaign for raising awareness about DRM, the criminalization of their circumvention, and their effects on economics, law, innovation
http://www.candidats.fr/ Campaigns to make the candidates to elections work on freedom in the digital world
http://www.stopDRM.info/ campaigns to educate consumers about music and video locked-down with DRM

Markus Schneider 2007-12-29 14:00 Saal 2 de lecture Society

Wahlchaos

Paradoxien des deutschen Wahlsystems

Wahlchaos beschäftigt sich mit Wahlverfahren aus mathematischer und politischer Sicht. So wurden die Wahlen von 1998, 2002 und 2005 betrachtet und a-postpriori manipuliert und ihre Auswirkungen diskutiert.

http://univis.uni-magdeburg.de/form?__s=2&dsc=anew/lecture_view&lvs=fgse/ipw/zentr/psy_0&anonymous=1&founds=fgse/ipw/zentr/psy_0,fma/iag/zentr/comput,/linear,/mab,/oberse&nosearch=1&ref=main&sem=2006s&__e= Seite des Seminars aus dem Universitätsinformationssystem

Tomasz Rybak 2007-12-29 16:00 Saal 2 en lecture Science

Analysis of Sputnik Data from 23C3

Attempts to regenerate lost sequences

In December 2006, in BCC 1000 atendees were wearing Sputnik Tags. Data was stored, and then made available for analysis. Unfortunately all IDs of tags were lost. This lecture presents what was stored, what happened to it, and attempts of reconstructing IDs and sequences of movements.

http://www.openbeacon.org/ Main page of Sputnik Project
http://www.bogomips.w.tkb.pl/sputnik.html My page with some analysis
http://pmeerw.net/23C3_ Page with analysis made by Peter Meerwald
http://wiki.openbeacon.org/wiki/Datamining Open Beacon Wiki about analysing data

Roger Dingledine 2007-12-29 17:15 Saal 2 en lecture Hacking

Current events in Tor development

Come talk with Roger Dingledine, Tor project leader, about some of the challenges in the anonymity world.

https://tor.eff.org/ Tor

Emerson 2007-12-29 18:30 Saal 2 en lecture Society

Hacking in the age of declining everything

What can we do when everything we thought turns out to be wrong

It is thought by many that the world may be facing Peaks in fossil fuel production and catastrophic climate change. These huge problems put into question the Industrial Civilisation and call for, at the very least, massive changes to society if humanity is to survive. Do hackers have a role to play in a post transition society? What sort of things should hackers know and prepare for in such a future?

starbug, Constanze Kurz 2007-12-29 20:30 Saal 2 de lecture Society

Meine Finger gehören mir

Die nächste Stufe der biometrischen Vollerfassung

Zum 1. November 2007 ging der biometrische Reisepass in die nächste Ausbaustufe. Seitdem müssen reisewillige Bürger neben dem frontalen Gesichtsbild auch noch ihre Fingerabdrücke abgeben.

Johannes Grenzfurthner 2007-12-29 21:45 Saal 2 en Culture

All Tomorrow's Condensation

A puppet extravaganza by monochrom and friends

A long time ago in a post-apocalyptic region far, far away. Sympathetic outlaws battle against hyper-villains. Some people die, some people get famous. Societal business as usual. But wait! Something is _happening_!monochrom (featuring Bre Pettis, Sean Bonner and others) try to reinterpret the steampunk genre in form of a steamy puppet extravaganza. A journey into the backwaters of imagination!

Oona Leganovic, Daniel Kulla 2007-12-29 23:00 Saal 2 en other Culture

Space Communism

Communism or Space first?

Following "Chaos und Kritische Theorie" from 23C3, another verbal battle: Oona Leganovic (aka Ijon Tichy) will promote the idea to sublate the capital relation and bring about communism first and only then to go to Space, because otherwise the earthly problems will be spread everywhere. Daniel Kulla (impersonating Captain Kathryn Janeway) will, on the other hand, defend the exploration humanism that once already ended the middle ages and of which can be expected to do the same to the crusted planetary commodity circus.

http://events.ccc.de/camp/2007/Fahrplan/events/1856.en.html "Weltraumkommunismus" auf dem Camp '07
http://dewy.fem.tu-ilmenau.de/CCC/CCCamp07/video/m4v/cccamp07-de-1856-Weltraumkommunismus.m4v Videomitschnitt vom Camp (m4v, 144 MB)

Tag 3 - Saal 3

Tonnerre Lombard 2007-12-29 11:30 Saal 3 de lecture Hacking

Grundlagen der sicheren Programmierung

Typische Sicherheitslücken

Dieser Vortrag bietet eine Übersicht über einige Dinge, welche man im Kopf behalten sollte, wenn man Software schreibt - vorausgesetzt, diese soll nachher nur von der Person benutzt werden, die sie auch betreibt. Die theoretischen Aspekte der Sicherheit werden mit Codebeispielen untermalt.

Jens Kaufmann 2007-12-29 12:45 Saal 3 en lecture Science

Introduction in MEMS

Skills for very small ninjas

MicroElectroMechanical Systems or MEMS are as part of micro system technology, systems with electrical and mechanical subsystems at the micro scale. It is basically an introduction in the technology and in its potential for hardware hacks and potential ways of homebrew devices.

Henning Westerholt 2007-12-29 14:00 Saal 3 de lecture Hacking

OpenSER SIP Server

VoIP-Systeme mit OpenSER

Der Vortrag stellt OpenSER und das Open Source Projekt dahinter vor. OpenSER ist ein flexiber und leistungsfähiger SIP Server, mit dem alle Arten von Voice over IP Infrastrukturen realisiert werden können. Er ist sowohl im DSL Router als Telefonanlage für die Wohngemeinschaft als auch von Carriern mit mehreren Millionen Kunden einsetzbar. Anhand dieser Beispiele werden einige gebräuchliche Einsatzszenarien aufgezeigt. Dafür ist es notwendig, kurz auf die Konfiguration, die Anbindung an Datenbanken und die wichtigsten Module einzugehen. Abschließend wird anhand des aktuellen Release 1.3 und der Roadmap die weitere Entwicklung des Projektes vorgestellt.

http://openser.org/dokuwiki/ OpenSER Dokumentation

Stephan Schmieder 2007-12-29 16:00 Saal 3 de lecture Culture

Getting Things Done

Der Antiverpeil-Talk

Eine Einführung ins Antiverpeilen mit Tools und Techniken rund um David Allens "Getting Things Done"-Methodik.

http://unixgu.ru/papers/gtd.html Keylearnings mindmap
http://www.amazon.de/dp/0142000280 The Manual bei Amazon
http://unixgu.ru/lib/exe/fetch.php?id=papers&cache=cache&media=gtd-mrmcd-slides.pdf Slides from the same talk at mrmcd110b
http://freemind.sf.net/ http://www.lifehack.org/
http://www.zenhabits.net/ http://www.lifeoptimizer.org/ http://www.thinkingrock.com.au/

twiz, sgrakkyu 2007-12-29 17:15 Saal 3 en lecture Hacking

From Ring Zero to UID Zero

A couple of stories about kernel exploiting

The process of exploiting kernel based vulnerabilities is one of the topic which have received more attention (and kindled more interest) among security researchers, coders and addicted.

http://www.phrack.org/issues.html?issue=64&id=6#article Phrack #64: Attacking the Core : Kernel Exploiting Notes

Nicolas Cannasse 2007-12-29 18:30 Saal 3 en lecture Hacking

haXe

hacking a programming language

haXe is a programming language for developing both server AND client side of a website. haXe can do Javascript/AJAX, Database access and even Flash and video streaming. All with one single programming language.

http://haxe.org	haXe website
http://nekovm.org	neko website
http://haxe.org/hxasm	hxASM website
http://haxevideo.org	haxeVideo website

dash 2007-12-29 20:30 Saal 3 en lecture Hacking

Reverse Engineering of Embedded Devices

The event aims on reverse engineering small boxes you can buy at your local Saturn or Media Market like SOHO Routers.

Frederik Ramm 2007-12-29 21:45 Saal 3 en lecture Making

OpenStreetMap, the free Wiki world map

3 years done - 10 to go?

The OpenStreetMap project has achieved remarkable successes in creating a free world map, and is growing fast. This talk gives an overview of what we do, why we do it, and what our data can be used for.

Tag 4 - Saal 1

Peter Eckersley 2007-12-30 11:30 Saal 1 en lecture Hacking

A Spotter's Guide to AACS Keys

AACS is the DRM system used on HD-DVD and Blu-Ray discs. It is one of the most sophisticated DRM deployments to date. It includes around twelve different kinds of keys (in fact, even counting the different kinds of keys is non-trivial), three optional watermarking schemes, and four revocation mechanisms (for keys, hardware, players, and certain disc images).

2007-12-30 14:00 Saal 1 en lecture Society

Wearables of the electronic and digital ages and the female cyborg

Historians of technology usually argue that in the mediation of technology, female icons served two purposes: firstly, attracting the male buyer as erotic signals; secondly, representing the simplicity of a technology`s handling. This scheme is obviously too simple and in itself stereotyped. It neglects the nuances of how women are envisioned in relation to what technologies and what this means for both the semiotics of a technology and the identities of women. For the case of the portable electronics, I will demonstrate such nuances. E.g. the radio was connected to female users as long as it served leisurable entertainment in public spaces.

However, when marketed as an information tool back home or on business tours, it was put in male hands. Furthermore, the popular ascriptions which condensed in the visions of media, advertising and manuals, also materialized in the artifacts themselves. Thus, radios or cell phones which were targeted explicitly at women had feminized designs, colours and features which should relate to their life experiences. In my talk, I will also include this dimension of the artifacts, analyzing them as frozen envisions of social and cultural values.

Luke Jennings 2007-12-30 16:00 Saal 1 en Hacking

One Token to Rule Them All

Post-Exploitation Fun in Windows Environments

The defense techniques employed by large software manufacturers are getting better. This is particularly true of Microsoft who have improved the security of the software they make tremendously since their Trustworthy Computing initiative. Gone are the days of being able to penetrate any Microsoft system by firing off the RPC-DCOM exploit. The consequence of this is that post-exploitation has become increasingly important in order to "squeeze all the juice" out of every compromised system.Windows access tokens are integral to Microsoft's concept of single sign-on in an active directory environment. Compromising a system that has privileged tokens can allow for both local and domain privilege escalation.

TyRaNiD 2007-12-30 17:15 Saal 1 en lecture Hacking

Playstation Portable Cracking

How In The End We Got It All!

The Sony PSP is over 3 years old yet barely a day has gone by without some part of it getting attacked. This lecture will go through how hacker ingenuity and systematic failures in Sony's hardware, software and business practices ended up completely destroying the hand held's security including some previously unreleased information about how it was achieved.

Alexander Kornbrust 2007-12-30 18:30 Saal 1 en lecture Hacking

Latest trends in Oracle Security

Oracle databases are the leading databases in companies and organizations. In the last 3 years Oracle invested a lot of time and engery to make the databases more secure, adding new features ... but even 2007 most databases are easy to hack.

http://www.red-database-security.com/ Homepage Red-Database-Security GmbH

Ron, Frank Rieger 2007-12-30 20:30 Saal 1 de lecture Hacking

Security Nightmares 2008

Oder: worüber wir nächstes Jahr lachen werden

Security Nightmares - der jährliche Rückblick auf die IT-Sicherheit und der Security-Glaskugelblick für's nächste Jahr.

Tim Pritlove 2007-12-30 21:45 Saal 1 en lecture Community

Closing Event

Tag 4 - Saal 2

Peter Voigt 2007-12-30 11:30 Saal 2 de lecture Society

GPLv3 - Praktische Auswirkungen

Was der Umstieg auf die GPLv3 an Neuerungen mit sich bringt, welche Fehler beim Wechsel vermieden werden können und an welchen Stellen rechtliche Fragestellungen lauern, für deren Klärung technische Überlegungen nicht ausreichen, schildert dieser Vortrag.

Marc-Andr Beck, Bernd R. Fix 2007-12-30 12:45 Saal 2 en lecture Hacking

Smartcard protocol sniffing

This talk will introduce you to the theoretical and practical issues involved in cloning/simulating existing smartcards. It is based on the lessons learned from cloning the Postcard (swiss debit card) issued by PostFinance.

http://postcard-sicherheit.ch/ postcard-sicherheit.ch

Jonathan Weiss 2007-12-30 14:00 Saal 2 en lecture Hacking

Ruby on Rails Security

This talk will focus on the security of the Ruby on Rails Web Framework. Some dos and don'ts will be presented along with security Best Practices for common attacks like session fixation, XSS, SQL injection, and deployment weaknesses.

Machtelt 2007-12-30 16:00 Saal 2 en lecture Society

Lobbying for Open Source

From one angry mail to writing national policy on Open Source

This talk is about our experiences with talking to the government. The focus is on how to get the job done, talking politics to people who are clueless about the need for free and open software.

kuza55 2007-12-30 17:15 Saal 2 en lecture Hacking

Unusual Web Bugs

A Web Hacker's Bag O' Tricks

While many issues in web apps have been documented, and are fairly well known, I would like to shine some light on mostly unknown issues, and present some new techniques for exploiting previously unexploitable bugs.

2007-12-30 18:30 Saal 2 en lecture Hacking

I know who you clicked last summer

A swiss army knife for automatic social investigation

One-mode and two-mode networks: This talk introduces some techniques of social network analysis and graph theory. It aims at using simple approaches for getting interesting facts about networks. I will use the data of a popular community to demonstrate some of the techniques.
* modelling possibilities* basic measures of networks and some algorithms of network and graph theory

Felix von Leitner 2007-12-30 20:30 Saal 2 de lecture Making

Abschlussbericht FeM-Streaming und Encoding

Das Streaming-Team der FeM e.V. möchte zum Abschluss des 24C3 einen Überblick über die Streaming-Aktivitäten geben, ein paar Statistiken jonglieren und sonstige (Un-)Auffälligkeiten und Stories berichten.

Tag 4 - Saal 3

Benjamin Henrion 2007-12-30 11:30 Saal 3 en lecture Society

OOXML

A twelve euros campaign against Microsoft's Office broken standard

Microsoft is currently trying to buy an ISO stamp for their flawed Office OpenXML (OOXML) specification.

http://www.noooxml.org/ Say NO to Microsoft Office broken standard

Olivier Cleynen 2007-12-30 12:45 Saal 3 en lecture Society

Overtaking Proprietary Software Without Writing Code

"A few rough insights on sharpening free software"

Free or "Open-Source" software, and in particular Linux, is doing extremely well technically. However, it fails to secure a significant portion of the protected, lucrative software market, especially for end-users. Can Free Software finally make a full entry into our society? The main obstacles to overcoming the domination of proprietary software, most of them non-technical, require thinking outside of code-writing. "Overtaking Proprietary Software Without Writing Code" will relate experience gained from the activities of the GNU/Linux Matters non-profit, and provide some hands-on advice for community members, taking a handful of relevant examples.

Immanuel Scholz 2007-12-30 14:00 Saal 3 en lecture Science

Dining Cryptographers, The Protocol

Even slower than Tor and JAP together!

Imi gives an introduction into the idea behind DC networks, how and why they work. With demonstration!

http://www.eigenheimstrasse.de/imi/dc DC Network Client (Java WebStart)
http://www.eigenheimstrasse.de/svn/dc/ Source Code to the DC Network Client
http://www.eigenheimstrasse.de/svn/dc/doc/dcnetwork.pdf Slides

Cyworg 2007-12-30 16:00 Saal 3 de lecture Society

Lieber Cyborg als Göttin

Politischer Hacktivismus und Cyborgfeminismus

Das Cyborgmanifest verbindet die Analyse der heutigen Gesellschaft als "Informatik der Herrschaft" mit dem Aufruf von politischem, kreativem Umgang mit Technik, der Möglichkeit des Angreifens von Machtstrukturen und mit der Überwindung der starren Grenzen zwischen den Geschlechtern.

24. Chaos Communication Congress
27. - 30. Dezember 2007, Berlin

Tagungsband

ISBN 978-3-934-63606-4

books-on-demand.de