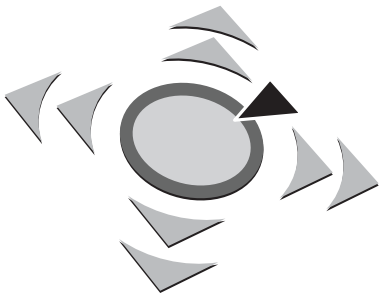
9. GI FG SIDAR Graduierten-Workshop über Reaktive Sicherheit

SPRING

Matthias Meyer (Hrsg.)

31.07.-01.08. 2014, Bochum



SIDAR-Report SR-2014-01
ISSN 2190-846X

Vorwort

SPRING ist eine wissenschaftliche Veranstaltung im Bereich der Reaktiven Sicherheit, die Nachwuchswissenschaftlern die Möglichkeit bietet, Ergebnisse eigener Arbeiten zu präsentieren und dabei Kontakte über die eigene Universität hinaus zu knüpfen. SPRING ist eine zentrale Aktivität der GI-Fachgruppe SIDAR, die von der organisatorischen Fachgruppenarbeit getrennt stattfindet. Die Veranstaltung dauert inklusive An- und Abreise zwei Tage und es werden keine Gebühren für die Teilnahme erhoben. SPRING findet ein- bis zweimal jährlich statt. Die Einladungen werden über die Mailingliste der Fachgruppe bekanntgegeben. Interessierte werden gebeten, sich dort einzutragen (`http://www.gi-fg-sidar.de/list.html`). Für Belange der Veranstaltung SPRING ist Ulrich Flegel Ansprechpartner innerhalb der Fachgruppe SIDAR. Inzwischen blickt die SPRING auf neun erfolgreiche Veranstaltungen zurück; angefangen mit der Premiere in Berlin über Veranstaltungen in Dortmund, Mannheim, Stuttgart, Bonn, München und Bochum.

2014 wurde sie erstmalig von einem Industrieunternehmen, der G DATA Software AG, ausgerichtet, mit dem Ziel, den Transfer zwischen Forschung und Industrie weiter zu vertiefen.

Die diesjährigen Vorträge decken ein breites Spektrum ab. Die Teilnehmer konnten sich in angeregten Diskussionen zu Problemen der grundlegenden Netzwerkkomponenten, Angriffen auf Internet-Technologien und Einblicken in Botnetze sowie zu Erkennung und Analyse von Schadsoftware und der Prüfung von Sicherheitssoftware vernetzen und austauschen.

Die zugehörigen Abstracts sind in diesem technischen Bericht zusammengefasst. Der Bericht ist über das Internet-Portal der Fachgruppe SIDAR (`http://www.gi-fg-sidar.de/`) und die Homepage der Veranstaltung (`https://spring2014.gdata.de/`) verfügbar.

Ich möchte mich bei allen Beteiligten, die zum Erfolg der Veranstaltung beigetragen haben, bedanken. Ein besonderer Dank geht an die G DATA Software AG, die mit ihrem Sponsoring einen freundlichen Rahmen für angeregte Diskussionen geschaffen hat.

Bochum, August 2014 Matthias Meyer

Contents

Diesen Bericht zitieren als:
Matthias Meyer, editor. Proceedings of the Ninth GI SIG SIDAR Graduate Workshop on Reactive Security (SPRING). Technical Report SR-2014-01, GI FG SIDAR, Bochum, August 2014,

Beiträge zitieren als:
Autor. Titel. In Matthias Meyer, editor, Proceedings of the Ninth GI SIG SIDAR Graduate Workshop on Reactive Security (SPRING). Technical Report SR-2014-01, page xx. GI FG SIDAR, Bochum, August 2014.

Malicious Code and Access Control in Software-Defined Networks

Christian Röpke

Ruhr-University Bochum
D-44801 Bochum, Germany
christian.roepke{at}rub.de

Software-Defined Networking (SDN) is a new paradigm for building networks and it may change the network market drastically. The main idea is to decouple network control features from packet forwarding hardware. This promises many advantages like more reliable and easier to manage networks or better cost-effectiveness compared to traditional networks. Because of that, the last few years people from both academia and industry has taken a great interest in this concept, and also big companies like Google benefit from SDN by increasing backbone performance and fault tolerance while reducing operation costs [Ho12]. In practice, so called *SDN controllers* manage programmable switches via open protocols like *OpenFlow*. Network control features are implemented as so called *SDN applications* which tell SDN controllers how to program the switches.

Beside its advantages, with SDN we also have to face new threats and attack vectors. One threat may arise through *malicious* SDN applications but little is known about it up to now [Ca13, Kr13]. On the one hand, such programs could harm on network-level, *e.g.*, by re-programming switches in order to deny certain network connections or reroute traffic to an adversary. On the other hand, they could attack SDN controllers on system-level, *e.g.*, by executing arbitrary code or shutting down the SDN controller program.

We focus on system-level attacks and analyzed popular and state-of-the-art SDN controllers, *i.e.*, *Floodlight*, *Beacon*, *OpenDaylight*, and *HP VAN*. Our results reveal a significant lack in confinement mechanisms of so called SDN *module* applications. We implemented several malicious SDN module applications with which we were easily able to shut down all examined SDN controllers, download and execute malware on such systems, and remotely control the SDN controllers. To tackle this problem, we implemented a security tool that allows to limit access of SDN module applications to sensitive operations which can harm SDN controllers on system-level. To fulfill that task, our tool needs to know which SDN module application depends on what sensitive operations in order to run correctly. Therefore, our current work is about designing and implementing an access control framework for SDN module applications. In our framework, we first want to provide a mechanism which allows developers to specify access requests to sensitive operations. Next, we plan to enable network operators to review and modify access requests in an easy and automatic way while they install a SDN module application. Our goal is then to use our security tool to install according security rules which come into effect before the installed SDN module application is started the first time. Finally, we aim to resolve situations in which no information about the sensitive operations used by a SDN application is known.

References

[Ca13] M. Canini *et al.*: *A NICE Way to Test OpenFlow Applications*, in *USENIX Symposium on Networked Systems Design and Implementation (NSDI'13)*, San Jose, USA, 2013.

[Ho12] Urs Hölzle: *OpenFlow @ Google*, at *Open Networking Summit (ONS)*, Santa Clara CA, USA, 2012.

[Kr13] D. Kreutz *et al.*: *Towards Secure and Dependable Software-Defined Networks*, in *ACM SIGCOMM workshop on Hot Topics in Software Defined Networking (HotSDN'13)*, Hong Kong, China, 2013.

Klassifizierung von Schadsoftware anhand von simuliertem Netzwerkverkehr

Florian Hockmann
Ruhr-Universität Bochum
D-44801 Bochum
florian.hockmann{at}rub.de

Hersteller von Antiviren-Produkten stoßen täglich auf hunderttausende Samples von Schadsoftware, die allerdings größtenteils aus Varianten bereits bekannter Schadsoftware bestehen. In diesem Vortrag wird eine Technik zur automatisierten Klassifizierung vorgestellt, die das Kommunikationsverhalten der Schadsoftware als Grundlage der Analyse verwendet. Bisherige Arbeiten auf diesem Gebiet haben hauptsächlich versucht eine möglichst weitreichende Kommunikation der Schadsoftware zu ermöglichen, indem ihr ein nur teilweise beschränkter Zugang zum Internet geboten wird [BC09][PL10]. Dies bringt allerdings das Problem mit sich, dass das Kommunikationsverhalten der Schadsoftware abhängig vom Verhalten und der Erreichbarkeit der erwarteten Kommunikationspartner (wie z.B. Command-and-Control-Server oder Peer-to-Peer-Netzwerke) ist. Daher wird in diesem Beitrag Inetsim zur Simulation von weit verbreiteten Internet-Diensten verwendet, wodurch eine Kommunikation zwischen der Schadsoftware und simulierten Kommunikationspartnern mithilfe von Standardantworten ermöglicht wird, ohne dass Pakete an die eigentlichen Kommunikationspartner weitergeleitet werden. Dadurch wird ein deterministisches Verhalten der Schadsoftware erreicht. Für die Analyse werden die Samples in einer virtuellen Maschine ausgeführt, wobei sämtliche Netzwerkpakete mit Tcpdump mitgeschnitten und in PCAP-Dateien abgespeichert werden.

Zur Klassifizierung wird der dichtebasierte Clustering-Algorithmus DBSCAN [EK96] verwendet, der mit Clustern beliebiger Form, Größe und Anzahl umgehen kann. Dabei wird eine eigene Distanzfunktion verwendet, um die aus den PCAP-Dateien extrahierten Eigenschaften für jedes verwendete Netzwerkprotokoll miteinander zu vergleichen. Eine implizite Erkennung von verschlüsselter Kommunikation und von Domain-Generations-Algorithmen wird unter anderem durch die Verwendung von Entropie-basierten Eigenschaften ermöglicht.

Mit diesem Verfahren wird ein korrekte Klassifizierung von bis zu über 90% der Samples erreicht. Dies zeigt, dass eine gute Klassifizierung von Schadsoftware ausschließlich anhand des Netzwerkverkehrs möglich ist ohne Pakete über das Internet weiterzuleiten.

Literatur

[BC09] Ulrich Bayer and Paolo Milani Comparetti and Clemens Hlauschek and Christopher Kruegel and Engin Kirda *Scalable, Behavior-Based Malware Clustering* 2009

[PL10] Roberto Perdisci and Wenke Lee and Nick Feamster *Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces* 2010

[EK96] Martin Ester and Hans-Peter Kriegel and Jörg Sander and Xiaowei Xu *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise* 1996

Improved Routing Anomaly Detection to Protect End Users

Matthias Wübbeling

Fraunhofer FKIE & University of Bonn
D-53113 Bonn
matthias.wuebbeling{at}fkie.fraunhofer.de

The Internet is the most important communication platform nowadays. Hence, it is considered critical infrastructure by at least all european governments. Although Internet security presently is a hot topic in the research community as well as in current affairs, many aspects are still covered from the general perception. Security discussions in the post Snowden and Heartbleed era are mainly focussed on the application layer and how to secure bi- or multilateral communication. Besides the public focus threats located on other layers of the Internet are well known in the research community. The routing system located in the network layer is still unstable and error prone. Due to implicit trust between Autonomous Systems (ASs) there are no viable opportunities to prevent routing anomalies and no solutions to mitigate the effects. Several proposed countermeasures share common shortcomings as network routers located in the Internet backbone are not capable of additional cryptography.

We are still working on solutions to protect end users from the consequences of Internet routing anomalies [1]. Common solutions depend on two large routing archives, RIPE RIS [2] and the RouteViews [3] project. With this archived information, prefix hijacking is reliably detectable while the detection of path spoofing still suffers from incomplete databases. Thus, to improve the detection and classification of routing anomalies it is necessary to query additional sources besides the information found within the routing system itself. We select Looking Glass servers and websites of Internet Exchange Points (IXPs) to gain independent information on actual peering relationships between Autonomous Systems (ASs). The approach presented in [4] improved the general database used for anomaly detection with the likelihood of peering relationships based on IXP participation shown on member lists and the evidence of peering collected from Looking Glass servers.

Utilizing the MonIKA framework for cooperative distributed monitoring [5], it is possible to further extend the detection of routing anomalies to end user systems. This framework allows the seamless integration in general network usage scenarios and the capability to automatically block network connections to those communication partners affected by routing anomalies.

References

[1] M. Wübbeling: *Visibility of Routing Anomalies for End Users* in *Proceedings of the Eight GI SIGSIDAR Graduate Workshop on Reactive Security (SPRING)*. Munich. February 2013.

[2] http://www.ris.ripe.net/

[3] http://www.routeviews.org/

[4] M. Wübbeling, T. Elsner, M. Meier: *Inter-AS Routing Anomalies: Improved Detection and Classification* in *Proceedings of 6th International Conference on Cyber Conflict (CyCon)*. Tallin. June 2014.

[5] M. Wübbeling, M. Meier: *Utilization of Traceroutes to Improve Cooperative Detection of Internet Routing Anomalies* in *tba.*. Berlin. September 2014

Patchwork: Stitching against malware families with IDA Pro

Daniel Plohmann
Fraunhofer FKIE
D-53113 Bonn, Germany
daniel.plohmann{at}fkie.fraunhofer.de

When performing in-depth analysis of malware families using Hex-Rays' IDA Pro, an analyst often faces situations where the malware sample should be code-transformed or automatically annotated as a preparation allowing for easier and therefore faster analysis. Typical cases include resolving dynamic imports (depending on the technique used by the sample, e.g. through kernel32!GetProcAddress or custom import hashing), run-time string or function decryption, or overall deobfuscation of larger code fragments.

In this talk, we present "Patchwork", an IDAPython plugin designed to perform such tasks in a structured way. Patchwork serves as a platform for managing a collection of "stitches". Stitches are single or combined transformations that are applicable to certain code patterns found within strains of or even across multiple malware families. The transformation methodology for each stitch contains the following phases:

- Localization of transformation candidates and a suitability check for the stitch
- (Optional) code emulation to allow for implementation agnostic transformations using PyEmu
- Application of the transformation templates (using emulated results) and reporting

The code of the plugin will be published [DP14] with the talk, including an initial collection of stitches that is planned to be extended over time by the author or through community submissions.

References

[DP14] Daniel Plohmann: *IDA Patchwork*, `https://bitbucket.org/daniel_plohmann/idapatchwork`

Detecting Host-Based Code Injections Attacks

Thomas Barabosch

Fraunhofer FKIE,
Friedrich-Ebert-Allee 144, 53113 Bonn, Germany
thomas.barabosch{at}fkie.fraunhofer.de

Nowadays security researchers are facing a steadily increasing flood of malware samples. Furthermore, malware authors focus more and more on other operating systems (OS) such as Mac OS X and Android. But not only the amount of malware samples and the number of their targeted operating systems increase, but so does also their level of sophistication ([CA14], [UR14]). However, fundamental behaviours of malware do not change over time and are reused on many operating systems. Such behaviours include network communication, autostart capability and file infection. While many of these behaviours are only shown by a small fraction of all malware families, some behaviours are inherent to a vast majority of all malware families. Therefore, these behaviours are suitable for reliable malware detection.

The host-based code injection attack (HBCIA) is such a behaviour: a local attacker copies code into a foreign process space residing on the same system and triggers its execution within the targeted foreign process space. This code can be either executed in parallel to the original code of the victim process (concurrent execution) or it can replace the original code of the victim process and execute exclusively (process hollowing). The main benefits of using HBCIAs are covert operation on the target system, interception of critical information and bypassing of personal firewalls. Recent investigations suggest that HBCIAs can be seen as an inherent behaviour of a malware family. It is unlikely to change between versions and variants of a malware family [BM14]. Furthermore, HBCIAs are not only used by mass malware like Zbot and its many derivatives but also by advanced persistent threats such as Careto [CA14] and Uroburos [UR14].

Bee Master is a novel approach for dynamically detecting HBCIAs [BM14]. It applies the honeypot paradigm to OS processes. Its basic idea is to expose regular OS processes as a decoy to malware. *Bee Master* focuses on concepts – such as threads or memory pages – present in every modern operating system. Therefore, it does not suffer from drawbacks of low-level OS-based approaches such as API hooking. Furthermore, it can be quickly ported to other platforms and thus allows OS independent detection of HBCIAs.

We implemented *Bee Master* on Microsoft Windows as well as Linux and evaluated it thoroughly in qualitative and quantitative evaluations. The results show that it reaches reliable and robust detection for various current malware families.

References

[CA14] Kaspersky Lab: Unveiling Careto - The Masked APT, 2014

[UR14] G Data SecurityLabs: Uroburos - Highly complex espionage software with Russian roots, 2014

[BM14] Thomas Barabosch, Sebastian Eschweiler and Elmar Gehards-Padilla: Bee Master: Detecting Host-Based Code Injection Attacks, *Detection of Intrusions and Malware, and Vulnerability Assessment: 11th International Conference, DIMVA 2014 Egham, England, July 2014 Proceedings*, Springer.

Amplification DDoS Attacks

Marc Kührer*, Thomas Hupperich*, Christian Rossow†, Thorsten Holz*

* Horst Görtz Institute for IT-Security
D-44780 Bochum, Germany
firstname.lastname{at}rub.de

†Saarland University
D-66123 Saarbrücken, Germany
christian.rossow{at}gmail.com

Vulnerabilities in various network protocols have been abused to launch large-scale amplification DDoS attacks. In such an attack, the attacker forwards relatively small requests with spoofed source address to so-called *amplifiers*—often open DNS resolvers or public NTP servers—that reflect significantly larger responses to the attack victim, resulting in a large traffic volume that exhausts the available bandwidth of the victim. Typically, these attacks rely on UDP-based network protocols that lack handshake mechanisms to verify the sender identity (i.e., the source IP address).

In recent work [1, 2], we tried to shed light onto the landscape of hosts that can be abused for such amplification attacks. In a first step, we performed Internet-wide scans for five vulnerable UDP-based protocols: DNS, NetBIOS, NTP, SNMP, and SSDP. For none of the protocols we find less than 2.5 million amplifiers. More specifically, we identified in-between 5 to 10 million vulnerable systems for SNMP, SSDP, and NTP, while the by far highest number was observed for DNS, fluctuating between 23 and 25.5 million amplifiers. Quite interestingly, we find some systems to be vulnerable to amplification for multiple UDP-based protocols. The largest overlap was observed between DNS and SNMP: a third of the SNMP hosts were also vulnerable to amplification via DNS. Yet, for most of the protocols the intersection is negligible, thus the number of amplifiers basically sums up. In total, we measured almost 46 million amplifiers for all five UDP-based protocols.

TCP-based protocols employ a three-way-handshake in which the two communication partners are verified by exchanging sequential and equally-sized packets before the (larger) payload data is transmitted, thus even when a spoofed `SYN` packet is sent to an end host, only a single `SYN/ACK` or `RST` packet should be reflected to the potential victim. In principle, the TCP protocol thus does not allow for amplification attacks using IP spoofed traffic. Unfortunately, the TCP handshake itself often yields amplification. We observed hosts that respond with an excessive number of `SYN/ACK` or `RST` packets upon a single `SYN` packet, while other systems transmitted payload data via `PSH` packets without a completed handshake. To determine the number of hosts that are vulnerable to TCP-based amplification attacks, we performed Internet-wide scans for common TCP protocols. In total, we identified more than 4.8 million devices (for the protocols FTP, HTTP, NetBIOS, SIP, SSH, and Telnet) that amplified a single `SYN` packet by a factor of > 20, resulting in an average amplification factor of 112x. For FTP and Telnet, we observed almost 2% of the responsive hosts (i.e., 2,913,353 hosts for FTP and 2,120,175 hosts for Telnet) to be vulnerable for amplification higher than 20x. For NetBIOS, we identified 3,087 amplifiers that transmitted 5,291 `RST` segments on average within 60 seconds and allowed an amplification up to a factor of 79,625x.

References

[1] Kührer, M., Hupperich, T., Rossow, C., and Holz, T. Exit from Hell? Reducing the Impact of Amplification DDoS Attacks. In *USENIX Security Symposium* (2014).

[2] Kührer, M., Hupperich, T., Rossow, C., and Holz, T. Hell of a Handshake: Abusing TCP for Reflective Amplification DDoS Attacks. In *USENIX Workshop on Offensive Technologies (WOOT)* (2014).

Mobile Device Fingerprinting

Thomas Hupperich

Ruhr-Universität Bochum
D-44801 Bochum
thomas.hupperich@rub.de

Online shop operators make increasing profits but also face more and more attempts to defraud [RR14]. Fraudsters usually provide fake invoice addresses and choose delivery addresses where they are able to intercept the shipment easily, e. g., public buildings. The debt is never paid as the invoice is sent to the fake address (which may actually exist) while the shipment is received by the fraudster. Typically, fraudulent orders are placed repeatedly with slight differences in address or name but assumably from the same internet client device. To tackle the problem of fraudulent online orders we aim to detect malicious multi-orders and recidivists by fingerprinting their devices.

Mobile browsing is becoming more and more popular [CV14] and this trend also attracts cybercriminals to use mobile devices for their deceptive business practice [EM13]. This yields a new challenge: While a variety of fingerprinting methods yield good results for high customizable computer systems like desktop PCs or notebooks these methods are hardly applicable for mobile devices [NA13]. These smartphones, tablets, etc. are less customizable and therefore less distinguishable.

We aim to develop new methods to fingerprint mobile devices including features from browser, system and hardware. As mobile browsers are customized very rarely we plan to gather system specific data like timezone settings, display data and installed applications. Also hardware features like sensor data should be considered as accelerometers already have been proven to be distinguishable [DR14]. We intend to determine the diversity and significance of these features to assign weights. Based on the resulting weighted feature set we compile a fingerprint which needs to be as unique as possible. In practice, the feature data is gathered while a customer shops online and when an order is placed a device fingerprint is compiled and stored.

This fingerprint enables online shop operators to identify and distinguish customer's devices. If two orders with different and suspect data are made from the same device it may be an attempt to defraud. As a remedy, online shop operators could offer only payment in advance to customers using devices which were involved in attempts to defraud.

References

[RR14] Center for Retail Research. Online Retailing: Britain, Europe and the US 2014. *http://www.retailresearch.org/onlineretailing.php*

[CV14] Cisco Systems, Inc. Global Mobile Data Traffic Forecast Update 2013 – 2018. In *Cisco Visual Networking Index*, February 5, 2014.

[EM13] EMC Corporation. White Paper: The Current State of Cybercrime 2013. *http://www.emc.com/collateral/fraud-report/current-state-cybercrime-2013.pdf*

[NA13] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13.

[DR14] Sanorita Dey, Nirupam Roy, Wenyuan Xu, Romit Roy Choudhury, and Srihari Nelakuditi. AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium*, NDSS 2014.

Virtual Machine-based Fingerprinting Schemes

Moritz Contag

Ruhr University Bochum
D-44801 Bochum, Germany
moritz.contag{at}rub.de

Fingerprinting describes the process of embedding an unique identifier into an object, enabling the fingerprinting party to track the object by extracting the fingerprint mark later [CTT07]. Shipping fingerprinted software offers a wide range of possibilities. A practical example is the tracking of stolen software, e. g., as exercised by Hex-Rays. However most proposed fingerprinting schemes lack *resilience* and the embedded marks are easily distorted or completely removed. For example, the first known scheme embeds fingerprint marks in the order of basic blocks of a function. This approach has been patented by Microsoft in 1994 [DM96]. Unfortunately, said scheme is prone to subsequent application which destroys the embedded mark [CTT07].

In order to counteract these drawbacks, we suggest to combine known fingerprinting schemes with *Virtual Machines* (VM for short). In the last few years these structure gained increasing popularity and are nowadays applied in countless software protection solutions in order to obfuscate the underlying code. Generally speaking, the basic idea is the translation of code in a known architecture to a custom one. This method has been proven to be non-trivial to circumvent in a general manner [Rol09].

In our work we propose a design for a Virtual Machine protection system that supports different fingerprinting schemes. These specifically exploit intrinsic properties of the Virtual Machine instance itself and are inspired by already existing approaches. This includes schemes proposed by Davidson and Myhrvold [DM96], Monden et al. [MIM00] as well as Linn et al. [LDK03].

Our reference implementation is able to both embed and extract said fingerprint marks reliably. However, future work includes the research and development of more fingerprinting schemes specific to Virtual Machines as well as improving the resilience of the VM instances, e. g., by employing common obfuscation and tamper-proofing techniques.

References

[CTT07] Christian S Collberg, Clark Thomborson, and Gregg M Townsend. Dynamic Graph-based Software Fingerprinting. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(6):35, 2007.

[DM96] Robert I Davidson and Nathan Myhrvold. Method and System for Generating and Auditing a Signature for a Computer Program, September 24 1996. US Patent 5,559,884.

[Rol09] Rolf Rolles. Unpacking Virtualization Obfuscators. In *3rd USENIX Workshop on Offensive Technologies.(WOOT)*, 2009.

[MIM00] Akito Monden, Hajimu Iida, K-i Matsumoto, Katsuro Inoue, and Koji Torii. A Practical Method for Watermarking Java Programs. In *Computer Software and Applications Conference, 2000. COMPSAC 2000. The 24th Annual International*, pages 191–197. IEEE, 2000.

[LDK03] Cullen Linn, Saumya Debray, and John Kececioglu. Enhancing Software Tamper-Resistance via Stealthy Address Computations. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003)*. Citeseer, 2003.

Measuring Software Security by Known Vulnerabilities

Arnold Sykosch

University of Bonn
D-53113 Bonn, Germany
sykosch{at}cs.uni-bonn.de

What makes secure software? Where should I put my money / efforts in? Is Firefox really more secure than the Internet Explorer? These questions are hard to answer. Most of the time, *secure* means the absence of a known *vulnerability*. There are a lot of things to consider when trying to measure the security of a given software product by its known vulnerabilities. Studies that try to analyze trends in software security [1] or even try to compare two products, seem to be biased in multiple ways [2]. Model building in this setting is no trivial task [3].

Sophisticated models need to take into account what makes people publish vulnerability reports and what makes these people look for vulnerabilities in the first place. A model like this needs to handle bias by exploiting auxiliary data like project meta data (e.g. team size, open source / closed source, lines of code, etc.). It should also include momentum effects, that are created when a vulnerability is found [4].

The first step towards building such models would be to explain motivation for a security researcher to look for a vulnerability. It is safe to assume that a researcher will start investigating, where he assumes a possible finding most beneficial. This might be in rather popular software, software that communicates over the network or even software that operates in a strict security related context (e.g. SSL implementations, virus scanners or firewalls). The motivation in case of a zero-day-exploit is clear. The researcher tries to rework the vulnerability to a given exploit. Reaction time on a zero-day-exploit might also be an indicator on the researcher's interest in a particular type of software. These hypotheses need to substantiated by looking for correlating data with a hight frequency of CVE reports. Next steps include building hypotheses on the reason, that makes people publish CVE reports.

A complete model will explain which expressiveness lies in CVEs regarding security of software and hopefully provides more insight on how secure software is made.

References

[1] Stefan Frei, et al.: *Large-scale vulnerability analysis.* Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense. ACM, 2006.

[2] Steve Christey and Brian Martin: *Buying Into the Bias: Why Vulnerability Statistics Suck.* BlackHat 2013.

[3] Dan Geer and Michael Roytman: *Measuring vs. Modeling.* ;login:, Vol 38(6), December 2013.

[4] Jim Finkle: *New bugs found in software that caused 'Heartbleed' cyber threat.* Reuters, Online: `http://www.reuters.com/article/2014/06/05/us-cybersecurity-internet-bug-idUSKBN0EG2MV20140605`, accessed: 16 June 2014;

Ein Verfahren zum Prüfen der Funktion von Anti-Malware-Software

Tonke Hanebuth

eicar{at}hanebuth.de

Der Autor stellt ein Forschungsvorhaben vor, das sich mit der klassischen herstellerübergreifenden Funktionsprüfung von Anti-Malware-Software durch den Benutzer befasst.

Die Landschaft der zu schützenden Betriebssysteme hat sich in den letzten Dekaden verändert. Die Systeme sind mittlerweise konsequent vernetzt und werden entsprechend anders genutzt. Als Reaktion ist die Komplexität von Malware-Schutz-Systemen gewachsen. Nicht mehr nur das Dateisystem, sondern auch Netzwerkanschlüsse werden überwacht. Zusätzlich zum signaturbasierten Anti-Virus sind von den unterschiedlichen Herstellern diverse Heuristiken, Cloud-Erkennung, URL-Filter, Spam-Schutz und weitere Komponenten entwickelt worden.

Es wird untersucht, welche Verfahren zur Funktionsprüfung von Anti-Malware-Software dem Benutzer zur Verfügung standen und stehen und welche Grenzen in modernen Umgebungen erfahren werden. Der Bedarf an veränderten oder weiteren Verfahren wird erfasst.

Zusätzlich sollen alternative und ergänzende bestehende Verfahren berücksichtigt werden.

Eine Vision soll entwickelt werden und als Basis für konkrete Implementationen von ergänzenden oder neuen Verfahren dienen.

Der Prozess wird ergebnisoffen durchgeführt und soll durch Öffentlichkeit und Transparenz mit Hilfe einer unabhängigen Arbeitsgruppe legitimiert werden.

Auf der 23rd Annual EICAR Conference 2014 in Frankfurt wird der Prozess in einem Workshop fortgeführt, weitere Erfahrungen zusammengebracht und soweit vorhanden Lösungsmöglichkeiten diskutiert.

Literatur

[1] www.eicar.org

[2] www.caro.org

[3] www.amtso.org

[4] spamassassin.apache.org/gtube