Looking Back on Three Years of Flash-based Malware

Christian Wressnegger and Konrad Rieck
Institute of System Security, TU Braunschweig

Abstract

Adobe Flash is about to be replaced by alternative technologies, yet Flash-based malware appears to be more common then ever. In this paper we inspect the properties and temporal distribution of this class of malware over a period of three consecutive years and 2.3 million unique Flash animations. In particular, we focus on *initially undetected* malware and thus look at a subset for which traditional methods have failed to provide timely detection. We analyze the prevalence of these samples and characterize their nature.

1. INTRODUCTION

For years dynamic and multimedia content was equatable to the use of the Adobe Flash platform. This started to slowly change with the upswing and advancement of JavaScript in modern web browsers, and accelerates with the gradual implementation of the HTML 5 standard [10]. However, according to Adobe the Flash Player is still used on a billion desktop computers with different browsers and operating systems [2]. The prevalence of the platform is further emphasized by the fact that in 2016 about every *fourth* webpage among the Top 1000 was still using Adobe Flash for video streaming, gaming, or advertisements [6].

This unprecedented pervasiveness has naturally attracted adversaries that take advantage of security issues to attack end-users. Unfortunately, the Adobe Flash platform has a long-standing history of severe security vulnerabilities. In the last two years, the Flash Player has been assigned 595 CVEs, marking an all-time high and a tremendous increase when compared to previous years [17]. Several of these vulnerabilities have been found to be used for targeted attacks [24] and also exploit kits are known to use recently published vulnerabilities as well as zero-day exploits for specially tailored Flash-based malware [3, 20].

Now that the end of an era presumably is within sight [1, 10] we take a look back on the past years of the Adobe Flash platform and, in particular, the malware which targets it. To this end, we first study different attack vectors of Flash-based malware and highlight popular strategies to hide malicious content, such as staged execution, source-code obfuscation, and most prominently probing of the execution environment.

Building upon this knowledge we conduct an extensive empirical study, where we consider a total of 2.3 million unique Flash animations that have been collected over a period of more than three consecutive years. For each of these samples, we have acquired multiple scan results of 66 different virus scanners from the VirusTotal service, which allows us to effectively track detections over time and thus study malware samples that have initially been missed by the AV industry. This is particularly interesting for a number of reasons: First, to understand the nature and evolution of malware and, second, to evaluate the performance of detection methods in cases where traditional methods have failed to trigger alarms.

For our analyses we define different subsets of our data that reflect this issue with varying degree. In particular, these subsets are defined using different thresholds for the minimal number of detections at the first and last scan of a sample. In our strictest setting, a sample must not be detected by a single virus scanner at first, but later on by at least 10. We then focus our evaluation on the malware samples in these subsets and characterize them with respect to basic properties, such as the use of environment fingerprinting, the dependency on Flash versions, and whether the malware uses dynamically loaded code or even relies on commercial runtime packers.

In summary, we make the following contributions:

- **Large-scale analysis.** For our study we consider a dataset of 2.3 million unique Flash animations comprising benign and malicious samples from three years.
- **Identification of initially undetected malware.** On the basis of multiple scans at varying points in time we identify Flash-based malware that went unnoticed when seen in the wild for the first time.
- **Characterization.** We examine the basic properties and features of these at-the-time undetected malware samples to better understand the nature of novel and difficult to detect malware.

The rest of the paper is structured as follows: Section 2 provides an overview of Flash-based malware and its attack vectors. The composition of the used dataset is described in Section 3. Section 4 narrows the overall data down to novel malware instances and characterizes these before Section 5 discusses related work. Section 6 concludes the paper.

EuroSec'17, April 23 2017, Belgrade, Serbia

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4935-2/17/04...5.00

DOI: http://dx.doi.org/10.1145/3065913.3065921

2. FLASH-BASED MALWARE

For 20 years now, the Adobe Flash platform has been used to provide dynamic and multimedia content on webpages, bringing forth a multitude of different releases. Unfortunately, serious security flaws accompany this evolution, which have steadily supported the proliferation of web-based malware. In comparison to the 5-year average from 2010 to 2014 the number of vulnerabilities that have been assigned a CVE number has more than quadrupled in both subsequent years 2015 and 2016 (Figure 1).

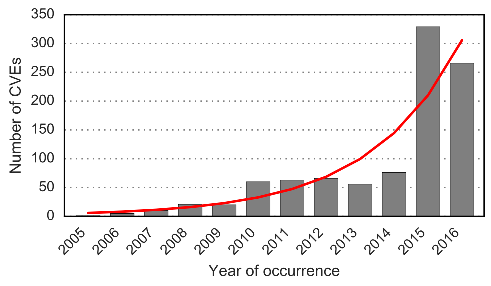

Figure 1: Number of vulnerabilties discovered in the Adobe Flash Player over the past 10 years.

In the following, we provide a brief overview of the versions and components of Flash that are relevant today, and highlight different attack scenarios and common obfuscation techniques employed by Flash-based malware.

2.1 Adobe Flash

Most Flash files contain ActionScript (AS) code to handle user input or drive animations. Today, two versions of the language are in frequent use, ActionScript 2 and 3, where the latter is a complete redesign based on the ECMAScript standard (ECMA-262) to support object-oriented programming. Along with the new language, ActionScript 3 also introduces a new virtual machine known as AVM2, whereas ActionScript version 1 and 2 are executed on the initial version of the ActionScript VM (AVM1).

The SWF file format is common to all versions of ActionScript, although it has been occasionally extended when new features emerged. Internally, Flash animations are composed of so-called *tags*, containers that are used to store ActionScript code as well as data of various kinds, including audio, video, image, and font data. Current versions of Flash support several dozen different types of tags, some of which occur nested, offering a huge attack surface for memory corruption exploits.

2.2 Attack Vectors and Scenarios

The complexity of the SWF file format, a powerful scripting language and the fact that both have evolved over years make Adobe Flash a perfect target for adversaries. We can roughly differentiate between three general categories of attacks, that may overlap in practice.

First, a piece of malware may be specially crafted to *exploit the Flash Player* during normal processing of its input, which does not necessarily involve the execution of ActionScript. An early example for such a vulnerability is `CVE-2007-0071`, that can be exploited to execute arbitrary code by leveraging an integer overflow caused by a negative Scene Count value [17]. Second, by utilizing the rich capabilities of ActionScript, an adversary can further advance the *launching or preparation of exploits* against either the Flash Player or different parts of the browser. `CVE-2015-3113`, for instance, allows to trigger a bug in the Microsoft Internet Explorer using the external interface of ActionScript 3. This can then be used to corrupt the length of a Flash `Vector` object and write outside its initially allocated memory [3, 17]. ActionScript can also be used to perform heap spraying or to obfuscate the presence of an attempt to launch an exploit (Section 2.3). Third, malware may use ActionScript to fingerprint the execution environment in a first stage of an attack and then redirect the user to an instance of an exploit kit, serving the actual malware. Adversaries have frequently used this scenario to reach a vast number of victims by distributing malicious advertisements over ad networks [see 4, 11].

2.3 Obfuscation

Malware authors employ different types of obfuscation to hinder analysis by automated systems and human experts [e.g., 13]. For malware based on Adobe Flash, three techniques are particularly prominent: First, so-called *staged execution* can be performed. At the first stage the malware only contains functionality to load further code, which in the second stage triggers the payload—a procedure that may be stretched over multiple rounds, potentially using encryption. Second, *source-code obfuscation* by inserting junk code, redirecting the control flow, or the use of rare or invalid instructions is frequently used in practice. Third, malware may *fingerprint the environment* to select a particular exploit for the current version of the platform or withhold its malicious intend if under analysis. In the following, we describe each of these techniques in more detail and provide examples of malware that employs them.

Staged execution. Both, ActionScript version 2 and 3 allow to dynamically load code in the form of Flash animations using the `loadMovie` function and the `Loader` class, respectively. This opens the door for encrypted payloads, polymorphism and runtime-packers such as secureSWF [9], SWFLock [19], or DoSWF [23]. The latter for instance is used by a sample[1] extracted from the Fiesta exploit kit, but also custom tailored solutions are equally widespread[2]. Due to the fact that recent versions of the Adobe Flash Player maintain backwards compatibility to the AVM1, for a long time it had been popular to load exploits for that particular platform, such as `CVE-2007-0071`, from within the AVM2. In light of the vast amount of newly discovered exploits for recent versions of the Adobe Flash Player this attack vector has become less important recently.

Source-code obfuscation. This obfuscation technique is geared towards thwarting systems trying to decompile the AVM bytecode, which can be used for the analysis by human experts and detectors based on static analysis. The landscape of Flash-based malware features numerous variants of this technique, that are also known from other platforms, such as variable substitution, string assembly, dead code insertion or changing the control flow. One example of such techniques is a malware exploiting `CVE-2015-5122` that has been found in the popular Angler exploit kit[3].

[1]md5: `5bf447627975b9ac6d0c68aa7f0b7d9a`
[2]md5: `7a322e01234ae1261428efe384956a26`
[3]md5: `7a322e01234ae1261428efe384956a26`

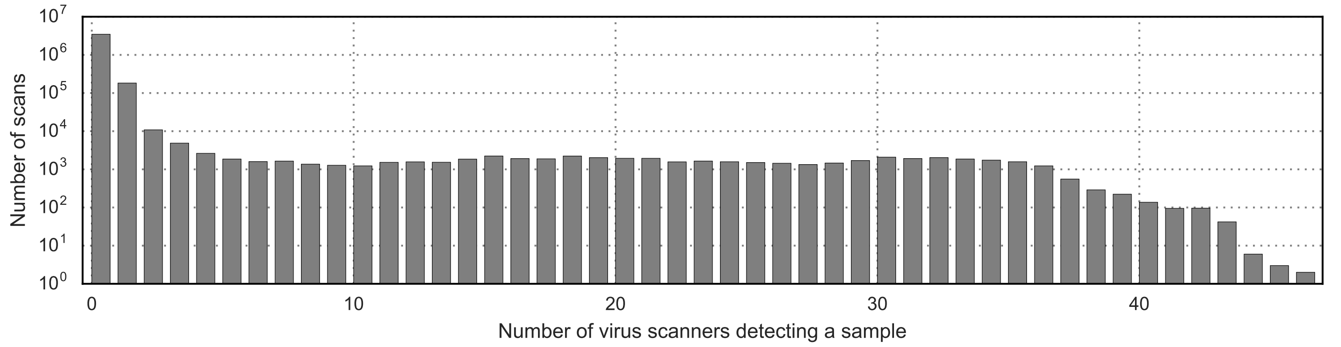

Figure 2: The number of scans in our dataset that contain detections from a certain number of virus scanners.

Probing the environment. As with other types of web-based malware, environment checks are heavily used in Flash-based attacks to (a) check for analysis environments and (b) fit the attack strategy to the version of the Flash platform and the browser [e.g., 4, 21]. With the `System.capabilities` (AS-2) and `flash.system.Capabilities` (AS-3) structures Flash provides various information on the execution environment.

As an example of such probing, a recent malware sample[4] exploiting the vulnerability `CVE-2015-5119` uses the variable `flash.system.Capabilities.isDebugger` for detecting potential debugging of its code. Similarly, other samples[5] make use of sanity checks `(version == 0)` or range-based version checks `(version >= 150000189 && version <= 150000239)` for exploring the environment.

Information about the environment may also be forwarded to the Flash animation using *FlashVars*—key/value pairs encoded in the `object` or `embed` HTML tags, or passed over as part of the URL. These variables can than be accessed using the `flash.display.LoaderInfo` object and might be used to trigger suitable exploits as frequently seen in exploit kits[6].

3. DATA COMPOSITION

The dataset for our analysis has been collected from December 2013 to January 2017 with the aid of the VirusTotal service and contains 2.3 million unique Flash animations in total. We have hence obtained malicious as well a benign samples for 38 months or roughly three years.

For each malware sample, we have additionally received the scan reports of all virus scanners run by the service using their—at the time—most recent signature databases. As a consequence, we have gathered an extensive portrayal of the detection performance of individual scanners for Flash-based malware over several years. As we have observed the data stream completely passively, that is, we have received the malware samples as they were submitted to VirusTotal, the number of reports per sample and their intervals vary considerably. Figure 3 shows the number of scans per sample in bins of 10. Note the logarithmic scale on the y-axis.

In the scope of this work we are primarily interested in inspecting how well Flash-based malware has been detected over the years and whether Flash-based malware still is an issue nowadays. As Adobe Flash has been declared dead for years now one might expect that also the number of malware samples has decreased with the downturn in usage numbers. The fact that the number of discovered vulnerabilities has reached an all-time high in the past two years (cf. Figure 1) proofs a certain interest in the Adobe Flash platform with respect to security. This, however, does not necessarily tell anything about the propagation of malware or whether this existing problem has been successfully contained. To shed light on the latter, we first have a look at the results of the virus scanners for each detection report we have collected. Figure 2 visualizes the results. We observe that the vast amount of scans find completely benign files without a single detection of any virus scanner hosted at VirusTotal. We credit this to the natural imbalance of benign to malicious samples submitted to such services.

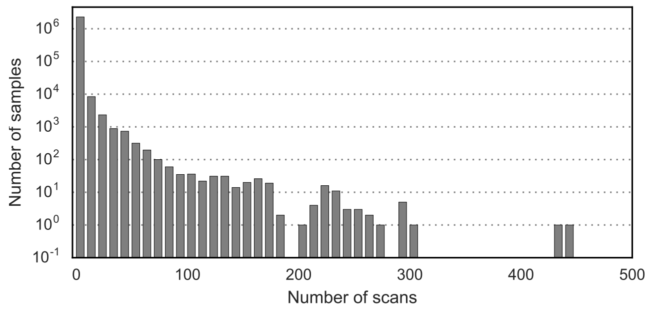

Figure 3: The number of malware samples associated with a certain amount of scans in bins of 10.

Second, we further look into the detections per sample and investigate how confident the collective decision towards a classification is in terms of the number of virus scanners agreeing on a sample being malware. Figure 4 shows the distribution of the *temporal change* of a sample's detection rate for the respective number of virus scanners detecting a sample as malicious (whiskers are limited to 1.5×IQR). The temporal change is measured as the difference between the current and very last scan we have observe in our dataset. For example, a sample detected by 4 and later 41 scanners is visible at the very top of the distribution on index 4 of the x-axis.

Similarly to Figure 2, the scans producing no detections are dominated by truly benign samples that are classified as such. The same holds true for scans where only one or two virus scanners raise an alarm. However, starting with 3 detections the variability and distance to the final number of detections increases drastically and only starts to flatten out towards higher values, meaning that the overall detection

[4]md5: `708e22f5a806804293d3c2b90e7d62ba`
[5]md5: `eeb243bb918464dedc29a6a36a25a638`
[6]md5: `439fea2f3f9b7ef0a0fdc01fb97b99a9`

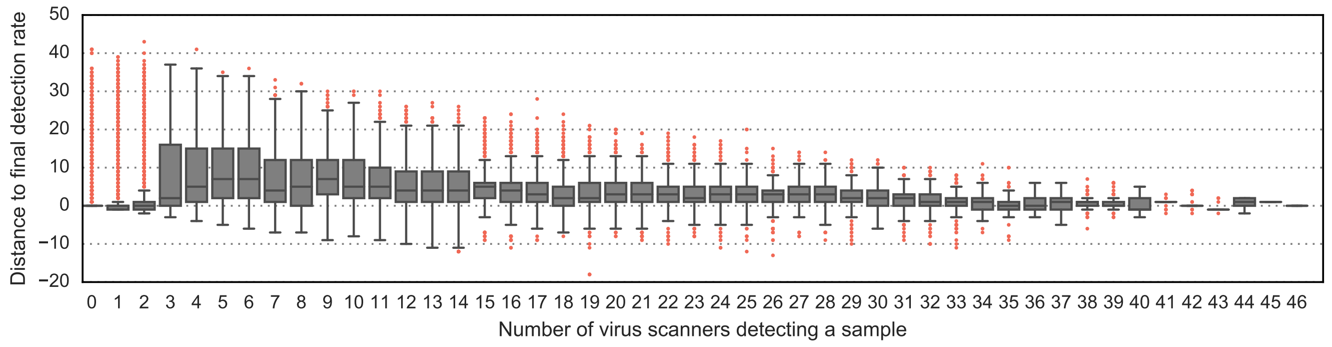

Figure 4: The detection rate's temporal change for different numbers of virus scanners detecting a sample.

rate becomes more stable with larger agreement. This bluntly reveals the reactive nature of current security solutions that allow to detect known malware but do not extrapolate very well to new and previously unknown instances.

From a malware detection point of view, the most interesting samples are those that heavily change in the number of detections, as these indicate cases where the virus scanners at first lack sufficient detection patterns but have adjusted over time. In the most extreme case, we are dealing with samples that have initially not been detected by any scanner but later on by a significantly large fraction. In Figure 4 these outliers are indicated as red dots.

4. SLIPPED THROUGH THE NET

For a more detailed analysis of Flash-based malware, we inspect those samples that drastically change in the number of detections over time. In particular, we consider three different subsets of varying difficulty for being detected which we refer to as Zarkov sets. The subsets are constructed using two thresholds $T = (t_1, t_2)$ such that the number of detections for a sample has to first be lower or equal to t_1 and subsequently greater or equal to t_2:

Z-1. For this subset we draw a sharp line at five detections. A sample is included whenever it receives five or more detections but did not in a previous scan: $T = (4, 5)$.

Z-2. Here we ignore scans with 5–9 detections and only consider samples that change from less than five to ten or more detections: $T = (4, 10)$.

Z-3. Finally, we make use of a rather strict specification that merely includes samples that start off with no detection at all and eventually are detected by ten or more virus scanners: $T = (0, 10)$.

Applying these thresholds leads to a total number of 3,321 unique Flash-based malware samples in *Z-1*, 2,904 in *Z-2*, and 814 instances in *Z-3*. Note that by construction, each of these sets is a strict subset of the former.

To convey a better feeling for how frequent these *initially undetected* instances occur, we inspect the points in time when we have first received them. Figure 5 breaks down the first occurrences of samples in *Z-1* by month revealing sporadic highs through out the years. In June/July 2014, for instance, `CVE-2014-0515` has been heavily used in various malware samples—a vulnerability that was found in the wild in mid-April 2014 in the context of a number of targeted attacks [24].

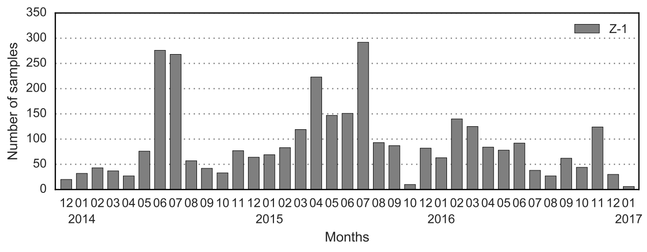

Figure 5: Initially undetected samples per month.

Subsequently we further inspect these sets with respect to basic properties and features to better understand the nature of at the time undetected Flash-based malware.

Adobe Flash version numbers. First, we look at the version of the Adobe Flash Player that is targeted by the malware. Figure 6 shows the relative frequency of the used versions for the samples in datasets *Z-1* to *Z-3*. Almost all platforms that have been released in the last 20 years are targeted by the samples we have identified. This is particular noteworthy as we are dealing with Flash-based malware that (a) has been collected in the last three years and (b) has initially not been detected by a single virus scanner: *Z-3*.

The heavy use of the exploit `CVE-2014-0515` can be linked to this figure as well. While in principle versions up to 13 are affected, this only applies to Windows and Mac OS releases. For Linux versions of the Flash Player, for which the development of the platform stagnates, the exploit is limited to versions up to 11.2. The same holds true for `CVE-2015-0311` which likewise concerns the Adobe Flash Player up to version 11.2 across all major platforms, explaining the burst for adjoining versions. For version 17 `CVE-2015-0359` and `CVE-2015-5119`, for instance, are very common.

Included ActionScript. Next, we evaluate whether the samples from the sets *Z-1* to *Z-3* contain ActionScript code and if so, which version it is of and which version of the ActionScript VM is targeted. Table 1 lists the exact numbers.

Dataset	AS-1	AS-2	AS-3	Total
Z-1	0.8%	1.4%	97.3%	99.5%
Z-2	0.4%	0.4%	99.0%	99.8%
Z-3	0.0%	0.4%	99.4%	99.8%

Table 1: Malware samples by the used ActionScript dialect for different Zarkov sets *Z-1* to *Z-3*.

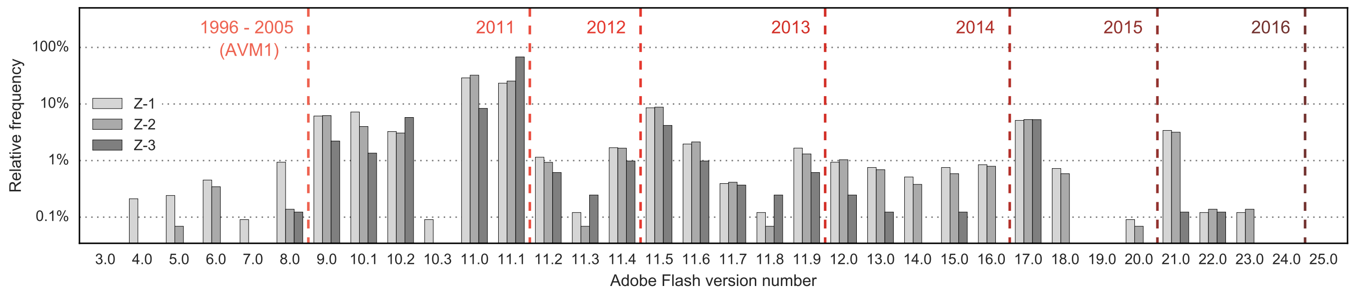

Figure 6: Flash version numbers in our datasets and the years the corresponding platform has been released.

The great majority of the Flash-based malware we consider makes use of ActionScript 3 to fingerprint the environment, obfuscate the actual payload, or facilitate the actual exploit. Only 0.2–0.5% of the samples do not contain any ActionScript code, which suggests that these few instances solely make use of structural exploits to implement the attack.

A few Flash-based malware samples even use ActionScript 1 or 2 for their attacks and consequently rely on the AVM-1 which is deprecated for more than 10 years and kept in recent versions of the Flash Player to provide backwards compatibility.

Runtime packers. ActionScript code is often used to obfuscate the malicious intention of the malware, as for instance, via generic runtime packers. The functionality of these is similar to their counterparts on other platforms: A minimal unpacking routine extracts the packed and/or encrypted program code that subsequently gets executed. Additionally, function and variable names often are aggressively mangled, or dead code is inserted to further complicate analysis.

For this experiment we have looked for indicators of three popular runtime packers: *DoSWF* [23], *secureSWF* [9], and *SWFLock* [19]. Table 2 summarizes the results and shows that by merely looking for these few commercial products we can already identify 12% of the Flash-based malware as being packed/protected. The importance of such runtime packers for day-to-day analysis of Flash-based malware is further emphasized by the fact that tools from the AV industry incorporate functionality to specially handle such packers [5].

Dataset	DoSWF	secureSWF	SWFLock	Total
Z-1	8.40%	2.62%	1.23%	12.26%
Z-2	7.64%	2.24%	1.24%	11.12%
Z-3	4.42%	0.49%	0.12%	5.04%

Table 2: Number of Flash-based malware protected with one of three popular commercial packers.

Malware characteristics. Finally, we derive more specific characteristics of Flash-based malware with the aid of VirusTotal and visualize these for the *Z-1* dataset in Figure 7. For instance, roughly 40% of the samples make use of dynamically loaded code. While the previous section already provides evidence for the importance of obfuscation for this kind of malware, the difference in the use of dynamic code and identified packers suggests that significantly more instances use similar products or custom solutions. This however must not be confused with the compression natively used by the SWF format which is the case for 92% of the samples. Furthermore, about 20% of the samples use the `capabilities` struct to perform environment fingerprinting. For 60% we are able to detect specific exploits, 10% use the external interface to communicate with the webpage the sample is embedded in, 5% contain very long ASCII hex-strings, and very few samples even contain raw executable files.

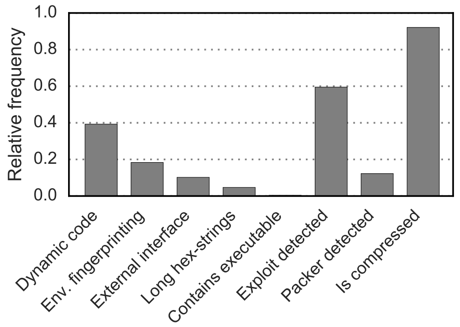

Figure 7: Security relevant properties of initially undetected Flash-based malware samples in *Z-1*.

5. RELATED WORK

Our analysis touches two disjunct fields of security research: (a) The detection performance of virus scanners and (b) the analysis of Flash-based malware. Subsequently, we briefly discuss recent work of both in relation to our study.

Evaluating virus scanners. The research community has repeatedly looked at the detection performance of virus scanners [e.g., 16], the quality of their textual labels [e.g., 14, 18], and how these can be used to build better ground-truth for evaluation [e.g., 7, 8]. The latter approaches share similarities with our work as they likewise analyze detections of virus scanners and go even further in that regard. Kantchelian et al. [8] also discuss the evolution of detections and observe that labels stabilize over time. Hurier et al. [7] briefly note that it is not sufficient to merely use thresholds on individual scans to build ground-truth, as detection results may change over time. Miller et al. [15], on the other hand, point out the importance of strict temporal separation of labels for training and testing malware detectors.

However, none of this work filters and processes malware based on the temporal evolution of detections as used for the composition of the ZARKOV sets and our subsequent analysis. While our approach is not intended for deriving ground-truth for a complete dataset, it enables us to determine initially missed malware instances, which are crucial test cases for the evaluation of detection approaches in general.

Flash-based malware. Surprisingly, the analysis of malware targeting the Adobe Flash platform has received relatively little attention over the years. Ford et al. [4] presents OdoSwiff a system designed to detect malicious advertisements based on static and dynamic analysis, whereby the latter lays special focus on ActionScript 2. Its successor FlashDetect [21] by contrast acts as a general purpose detector for Flash-based malware using ActionScript 3, and thus, fills a gap OdoSwiff left behind. Gordon [22] improves the detection performance for Flash-based malware over FlashDetect by employing a pragmatic approach to multi-path exploration for ActionScript 2 and 3 code in combination with structural features of the file format. Lindner [12] proposes an alternative to the mere analysis and detection: Blitzableiter prevents malicious Flash files from executing *as is* and normalizes the file instead, thereby eliminating potentially malicious parts. Each of these systems would greatly benefit from demonstrating their effectivity on the three Zarkov datasets.

6. CONCLUSIONS

Adobe Flash repeatedly suffers from vulnerabilities and malware targeting the platform. In this paper, we look at Flash-based malware from three consecutive years. By exploiting the *temporal change* of detection rates we are able to specially focus our study on malware that has initially been missed by traditional methods due to their reactive nature. We show that such instances are constantly observed throughout the years with peaks that relate to popular attack campaigns. Our analysis enables carving out interesting test cases from large malware collections and thus is a perfect tool for evaluating detection methods in non-ideal scenarios.

Acknowledgments

The authors gratefully acknowledge funding from the German Federal Ministry of Education and Research (BMBF) under the projects APT-Sweeper (FKZ 16KIS0307) and VAMOS (FKZ 16KIS0534).

References

[1] Adobe Systems. Flash, HTML5 and open web standards. https://blogs.adobe.com/conversations/2015/11/flash-html5-and-open-web-standards.html, visited March 2017.

[2] Adobe Systems. Adobe Flash runtimes: Statistics. http://www.adobe.com/products/flashruntimes/statistics.html, visited March 2017.

[3] D. Caselden, C. Souffrant, and G. Jiang. Flash in 2015. https://www.fireeye.com/blog/threat-research/2015/03/flash_in_2015.html, visited March 2017.

[4] S. Ford, M. Cova, C. Kruegel, and G. Vigna. Analyzing and detecting malicious flash advertisements. In *Proc. of Annual Computer Security Applications Conference (ACSAC)*, 2009.

[5] T. Hirvonen. Dynamic instrumentation tool for adobe flash player built on intel pin. https://github.com/F-Secure/Sulo, visited March 2017.

[6] HTTP Archive. http://www.httparchive.org.

[7] M. Hurier, K. Allix, T. F. Bissyandé, J. Klein, and Y. L. Traon. On the lack of consensus in anti-virus decisions: Metrics and insights on building ground truths of android malware. In *Proc. of Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2016.

[8] A. Kantchelian, M. C. Tschantz, S. Afroz, B. Miller, V. Shankar, R. Bachwani, A. D. Joseph, and J. D. Tygar. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *Proc. of ACM Workshop on Artificial Intelligence and Security (AISEC)*, 2015.

[9] KINDI Software. secureSWF: Protect, encrypt, and optimize swf flash. http://www.kindi.com, visited March 2017.

[10] A. LaForge. Flash and chrome. https://blog.google/products/chrome/flash-and-chrome, visited March 2017.

[11] Z. Li, K. Zhang, Y. Xie, F. You, and X. Wang. Knowing your enemy: Understanding and detecting malicious web advertising. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, 2012.

[12] F. Lindner. Preventing Adobe Flash exploitation - Blitzableiter - a signature-less protection tool. In *Proc. of Black Hat USA*, 2010.

[13] C. Linn and S. Debray. Obfuscation of executable code to improve resistance to static disassembly. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, 2003.

[14] F. Maggi, A. Bellini, G. Salvaneschi, and S. Zanero. Finding non-trivial malware naming inconsistencies. In *Proc. of International Conference on Information Systems Security (ICISS)*, 2011.

[15] B. Miller, A. Kantchelian, M. C. Tschantz, S. Afroz, R. Bachwani, R. Faizullabhoy, L. Huang, V. Shankar, T. Wu, G. Yiu, A. D. Joseph, and J. D. Tygar. Reviewer integration and performance measurement for malware detection. In *Proc. of Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2016.

[16] A. Mohaisen and O. Alrawi. AV-Meter: an evaluation of antivirus scans and labels. In *Proc. of Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2014.

[17] S. Özkan. CVE Details. http://www.cvedetails.com, visited March 2017.

[18] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero. AVclass: A tool for massive malware labeling. In *Proc. of International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2016.

[19] SWFLock.com. SWFLock: Online encryption software for flash. http://www.swflock.com, visited March 2017.

[20] Trustwave Holdings, Inc. Trustwave global security report. Technical report, Trustwave Holdings, Inc., 2016.

[21] T. van Overveldt, C. Kruegel, and G. Vigna. FlashDetect: ActionScript 3 malware detection. In *Proc. of International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2012.

[22] C. Wressnegger, F. Yamaguchi, D. Arp, and K. Rieck. Comprehensive analysis and detection of flash-based malware. In *Proc. of Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2016.

[23] Yushi High Technology Ltd. DoSWF – professional flash swf encryptor. http://doswf.org, visited March 2017.

[24] V. Zakorzhevsky. New Flash Player 0-day (CVE-2014-0515) Used in Watering-hole Attacks. https://securelist.com/blog/incidents/59399/new-flash-player-0-day-cve-2014-0515-used-in-watering-hole-attacks/, visited March 2017.