Scargos: Towards Automatic Vulnerability Distribution

Florian Rhinow[1,2] and Michael Clear[1]
[1]*School of Computer Science and Statistics, Trinity College Dublin, Dublin 2, Ireland*
[2]*SAP Business Intelligence R&D, 1012-1014 Kingswood Ave., Citywest Business Campus, Dublin 24, Ireland*

Keywords: Dynamic Taint Analysis, Self-certifying Alerts, Vulnerability Distribution, Zero Day Attacks.

Abstract: Recent work has suggested automated approaches to vulnerability distribution, but their usage has been limited to local networks and memory corruption detection techniques and has precluded custom vulnerability response processes. We present Scargos, a novel approach to automate the distribution and verification of vulnerabilities across the internet, while allowing for automatic, custom countermeasures without the need to trust a central authority. By leveraging collaborative detection, vulnerability reports can be contributed by anybody and are announced to an open network by using packet-based self-certifying alerts (SCA), which are a proof of the existence of a vulnerability by capturing the original, unmodified attack. We show that our approach allows for detection of previously unknown attacks, while an entire life cycle including distribution and verification is achieved on average in under 2 seconds.

1 INTRODUCTION

The security of companies and organisations relies heavily on their security departments. Often specialised incident response teams are employed to surveil the state of the network. These teams react to intruders or analyse past attacks. However they also take action if a new vulnerability is announced for one of their used applications. New vulnerabilities are conventionally first announced by the vendor of an application and are later redistributed by secondary sources such as a CVE. The time from a vulnerability announcement until it is known by an incident response team is critical, yet it often relies on manual checking of websites or newsletters, or is heavily reliant on secondary sources. The situation becomes worse when considering full-disclosure announcements, which can be made by virtually everybody. As a consequence, the number of primary sources seems indefinite. Automated software exists, but likewise, relies on the publication by a primary source. Furthermore, vendors often know about a vulnerability much earlier, but delay its announcement in order to write an appropriate patch in time – time in which applications are vulnerable in companies and organisations, without their knowledge. Informed attackers can intrude and steal valuable assets, while there is no effective way to ensure protection.

However, there are vulnerabilities which remain unknown for an even longer period: zero-day vulnerabilities. These vulnerabilities are not known to the public, but only to attackers. A recent study by Symantec (Bilge and Dumitras, 2012) involved the development of the worldwide intelligence network environment (WINE) to collect data from over 11 million hosts around the world. The data was analysed in retrospect from 2008 until 2011 to determine whether vulnerabilities were previously used by attackers prior to their official discovery. In total, 18 zero-day vulnerabilities were found and the data suggests that the attacks remained undetected for between 19 days to 30 months; on average 312 days (Bilge and Dumitras, 2012).

The motivations for engineering a solution are clear:

- Detection of zero-day attacks or other attacks that are unknown to a group of entities
- Interested entities are notified automatically and fast (within seconds) about such an attack and can verify its existence
- Verified attacks can be automatically processed by the entities to e.g. set up countermeasures

In this work, we present Scargos, a framework for automatic vulnerability distribution. Scargos can detect zero-day attacks by using dynamic taint analysis in a honeypot. The information about the vulnerability is then decoded as a packet-based self-certifying alert and verified by all interested parties. As we will

Rhinow F. and Clear M..
Scargos: Towards Automatic Vulnerability Distribution.
DOI: 10.5220/0005566203690376
In *Proceedings of the 12th International Conference on Security and Cryptography* (SECRYPT-2015), pages 369-376
ISBN: 978-989-758-117-5
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

show in this work, the entire process can be accomplished in on average less than 2 seconds, while not requiring that the participants trust each other.

Self-certifying alerts (SCAs) have been proposed previously by (Costa et al., 2005) as a way to distribute vulnerabilities in local networks without the need of a trusted central authority. The solution in (Costa et al., 2005), known as Vigilante, uses a form of SCAs that involve modifying the content of the original attack and then sending it as a broadcast message. It can only be used in conjunction with memory-violation detection engines. In contrast, Scargos employs packet-based SCAs, which preserve an attack in its original form. Additionally, Scargos allows for distribution of vulnerabilities across the internet, enables a custom vulnerability response process by end users, and can be used with any attack detection engine.

2 RELATED WORK

Dynamic taint analysis (DTA) also known as *dynamic dataflow analysis* or *dynamic information flow tracking* has been proposed independently by multiple groups (Newsome and Song, 2005; Costa et al., 2004; Crandall and Chong, 2004; Suh et al., 2004). DTA taints all external data in-memory and recognizes if the data is used to change the execution flow to detect attacks. Experiments that have been conducted with DTA suggest that its false-positive rate is extremely low. Newsome et al. (Newsome and Song, 2005) and Clause et al. (Clause et al., 2007) independently conducted accuracy experiments, which both resulted in no false positives. However, slowdown depends on the application and implementation that is being used ranging from 300-100.000% – *TaintCheck*(Newsome and Song, 2005), 4.000% – *Dytan* (Clause et al., 2007), 1.500%-3.000% – *Minemu* (Bosman et al., 2011), 1%-9% – *FlexiTaint (Hardware)* (Venkataramani et al., 2008).

Argos (Portokalidis et al., 2006) is a honeypot that attempts to mitigate many of the drawbacks that accompany conventional high-interaction honeypots. It decreases the risk and maintenance of honeypots by using DTA to accurately detect attacks. Therefore, Argos can stop intrusions before a compromise of the system occurs.

CWSandbox (Willems et al., 2007) is a tool that is developed for the Win32 platform to automatically generate malware analysis reports (Willems et al., 2007; Provos and Holz, 2009).

Honeycomb is a plug-in for the low-interaction honeypot *Honeyd* (Provos, 2003). Using Honeyd, Honeycomb collects malicious traffic to automatically create IDS signatures of the received data (Kreibich and Crowcroft, 2004).

Roleplayer (Cui et al., 2006) is a tool which observes and mimics interactions with network services. It operates protocol independently and heuristically adjusts network addresses, ports, cookies.

The *network of affined honeypots* (NoAH) is a network of honeypots which occupies unused IP-addresses on the internet. Automated attacks which scan portions of the Internet's IP-addresses are detected by using Argos as a DTA honeypot. The results from Argos are then converted into IDS signatures and are intended to be used by third parties (Kontaxis et al., 2010; Kohlrausch, 2009). NoAH allows for detection of zero-day attacks by using DTA honeypots. However, the system has a significant drawback: IDS signatures cannot be verified and discard the original content of the attack. This disallows any alternative vulnerability response process. Furthermore, malicious IDS signatures could be injected into the system by compromising Argos, which can have serious ramifications. The possibility of such an attack is fair, taking into account (1) that we have found real-world attacks that have remained undetected by Argos, as shown in Section 6.1, and (2) that there have been vulnerabilities that have allowed a VM escape from QEMU, the VM used by Argos.

Vigilante (Costa et al., 2005) is a proposed system to stop worms and viruses from spreading in a local network. Vigilante uses so called *self-certifying alerts* (SCAs) as a means to transfer information about a software vulnerability. SCAs are unique because they contain some parts of the exploit code itself. SCAs are generated when a host detects an attack by using DTA or *non-executable pages*. Other hosts then verify SCAs by executing modified parts of the attack code in a virtual machine. If the exploit succeeds, the host sets a custom filter onto its network interfaces to protect against the worm (Costa et al., 2005).

In contrast to Scargos, Vigilante has fundamental disadvantages for sharing vulnerability information globally:

A) Engineered for local networks – Vigilante was engineered for usage in local networks/production networks. The system uses flooding to propagate vulnerability alerts across networks, which is unsuitable for distribution across the internet. Additionally, Vigilante modifies the original attack before encoding it as a SCA and verifies it with further modifications. Therefore, they cannot be processed by other services because information about the original attack might be missing.

B) Binary Instrumentation – Vigilante relies on binary instrumentation of its network facing services for the usage of DTA. The drawback of this is that calls to other programs or kernel vulnerabilities cannot be detected because only the binaries of the network-facing service are surveilled. In contrast, using honeypots with in-built attack detection engines such as Argos allows for full-system protection.

C) Limited to Memory-Based Detection Techniques: Any detection mechanisms that Vigilante uses must be translated prior to use into binary instrumentation algorithms. This essentially limits the choices for Vigilante's detection engines to memory violation-detection techniques. In contrast, Scargos' usage of packet-based SCAs allow for virtually any detection engine.

3 ARCHITECTURE

The Scargos architecture is an approach to detect, distribute and verify the existence of previously unknown network-based attacks, conventionally called *zero-day attacks*. By combining very accurate detection engines such as dynamic taint analysis (DTA) and honeypots with traffic replaying, zero-day attacks can be distributed and verified much faster than non-automatic approaches. Figure 1 shows the relationship between all of Scargos' components. A prerequisite for Scargos' process is the detection of an attack; the process continues with SCA publishing, distribution and verification. Vulnerability response is optional and depends on the need of the end user.

By using highly accurate detection engines such as DTA, the detection component detects previously unknown network-based attacks. The goal of the detection component is not necessarily to gather as many threats as possible but rather to gather the most recently developed attacks. After a threat has been detected, all attack information is decoded into a custom format: *packet-based self-certifying alerts*.

New SCAs are submitted and aggregated in SCA repositories. The SCA repository stores self-certifying alerts and distributes them. User groups which are interested in receiving the newest attacks can subscribe to SCA repositories. For every unique payload that the repository receives, subscribers receive a notification.

Subscribers will usually verify a SCA before it is further processed to initiate an individual vulnerability response process, given the SCA is valid. SCA verification is done by leveraging the same methods that were used for SCA generation, by using highly accurate attack detection systems such as DTA. What happens after a SCA has been verified depends on the needs of the user group. There are different user groups which may benefit from using Scargos:

- *Researchers:* can study new threats to complement their vulnerability research for academia, security vendors, ISPs, institutes or corporations.
- *Incident Response Teams:* need to be aware of the newest attacks in order to take appropriate action.
- *IT-security Processes:* Scargos can be operated in a fully-automated fashion. As such, Scargos can be part of a larger IT process to automate handling of new vulnerabilities in a predefined manner.

Figure 1: Scargos' life cycle. Detection is usually handled by a DTA honeypot and handling of vulnerability response depends on the needs of the end user.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

4 PACKET-BASED SELF-CERTIFYING ALERTS

Packet-based SCAs store the transport-protocol payload of all packets of an attack in binary format. Each transport-protocol payload that has been made during the conversation which led to the compromise is separated as an element in a list. As such, packet-based SCAs lead to many simplifications when compared to conventional SCAs.

In contrast to Vigilante, Scargos uses packet-based SCAs, which preserve the original attack. This requires us to use new approaches to verify an SCA. We propose that verification of an attack is done in the same manner as that in which the original attack compromised the system. Using this approach we allow for a custom vulnerability response process by end users and any attack detection engine can be used with Scargos.

SCAs always refer to a vulnerability of a specific network-facing service. By using unique application IDs, all mentioned components can address a specific application. The IDs are assigned by each repository individually and SCA publishers and subscribers need to be aware of existing IDs for their surveilled application or create a new entry, if one of the used applications has not been registered yet. Figure 2 shows an SCA for *WFTPD Server*, which has been given the application ID of 1 in this example.

Attacks on network-facing services are usually conducted by sending a series of malicious packets.

Application ID:	1
Version:	3.23
Port:	21
Transport:	TCP

Layer-5 PDUs:	`VVNFUiBhbm9ueW1vdXMNCg==\n`
	`UEFTUyBtb3ppbGxhQGV4YW1wbGUuY29tDQo=\n`
	`U0laRSAvHUc0JUIcs6kURvm4Q0sEtJE1t`
	…

Figure 2: An example SCA of the application WFTPD Server. The attack consists of three Layer-5 PDUs, which are encoded using Base64. The last PDU has been truncated and is 727-Bytes long.

Weidong et. al (Cui et al., 2006) have demonstrated that simple replaying of captured malicious traffic is sufficient to trigger a vulnerability using *Roleplayer Replay*. Given multiple conversations of a protocol, dynamic fields of the target conversation are identified and changed accordingly. The conversation is replayed based upon a ”script” of a dialogue between the two communicating systems (Cui et al., 2006).

In contrast to Roleplayer we developed *Exact Stream Replay* as part of our contribution: Given only the layer-5 PDU conversation, which led to the compromise of one host, a second host that has the same vulnerability can be compromised by replaying the entire conversation as it was previously recorded. It requires no previous training, leads to smaller SCAs and is for all of our tested attacks sufficient to trigger the vulnerability. While exact stream replay requires dynamic replacement of fields such as URL, Host name and IP source/destination, it seems to be better suited for detection of zero-day attacks, because roleplayer might fail to categorise conversations with scrambled attack code. Also training might not be available and can be manipulated.

5 GENERATING PACKET-BASED SCAS

With the implementation of server architecture and replay mechanisms being trivial, we now turn our attention to the process of generating packet-based self-certifying alerts with a suitable detection engine. As discussed earlier, any type of highly accurate detection engine can be used to verify and generate non-packet based SCAs. The process is divided into four steps:

1. The detection engine alarms us of an intrusion;
2. We retrieve the packet which led to the intrusion;
3. Using this packet we reconstruct the entire attack;
4. Given the attack stream we marshal the data into a SCA.

Argos provides an in-built functionality to find the malicious packet which was used to taint the memory. As this packet is in fact on the second layer of the OSI stack we refer to it as an *Exploit Layer-2 Frame*. Although this information is helpful to us in some cases, we will show in our evaluation that Argos is often unable to trace back to the malicious packet. Consequently, we developed a combination of algorithms to significantly increase the probability of finding the correct packet stream of an attack by analysing two additional memory footprints of Argos: *Compromised Memory Block* and *Jump Target Memory Block*.

6 EXPERIMENTAL RESULTS

This section evaluates Scargos by discussing a variety of experiments. We implemented Scargos as a prototype on a x86 machine running *GNU/Linux 3.5.0* and using *Argos 0.5.0* as the DTA honeypot. Experiments were carried out on a machine with 4GB of RAM and an Intel(R) Core(TM) i3 CPU M 370, which is a 2.40GHz dual-core processor.

For our evaluation we chose to concentrate on one of the most severe attacks: *Remote Command Execution* vulnerabilities on server applications. Scargos is evaluated with 24 real attacks, which we describe in more detail in the full version. We have included high profile attacks against Windows including vulnerabilities *MS03-026 (RPC)* used in the *Blaster Worm* (Bailey et al., 2005), *MS04-011 (LSASS)* used in the *Sasser* (Sullivan, 2004) worm, and *MS08-067 (NetAPI)* used more recently by *Conficker* (Faulhaber et al., 2011). We expanded our selection of attacks by including attacks on services such as Windows SMB, HTTP and FTP.

6.1 Accuracy

A high accuracy rate is vital to ensure Scargos' widespread use. The accuracy of Scargos is defined as its success rate in *both* correctly generating and verifying SCAs. When evaluating accuracy it is important to look at all parts which can affect the result. In Scargos, accuracy depends on three components:

- *Argos*: We use Argos as our DTA honeypot and rely on the correct detection of attacks. An SCA cannot be created, if attacks remain undetected.
- *Packet Extraction*: The SCA publisher relies on a robust algorithm to extract the needed packets. If

packet extraction fails, no SCA can be published.

- *Packet Replay*: The SCA verifier needs a replay algorithm which can successfully replay SCAs such that honeypots become compromised. If packets cannot be replayed or are replayed in a manner such that the attack fails, false negatives occur because correct SCAs are not verified.

In our accuracy experiment, we tested all three components and subcomponents by conducting 10 runs for each of our selected applications. The results are shown in Table 1 and we analyze them as follows.

6.1.1 SCA Generation

Argos uses DTA which is known to have a very low false negative rate. However, the results from Table 1 show that out of our 24 applications an attack against *Mercury Mail* and *Mdeamon Pro* was not detected by Argos and successfully circumvented its DTA. We can not say with certainty why Argos did not detect the attack, but can only suspect that it is either due to under-tainting; a bug in the implementation or Argos was trusting input which should not be trusted. Due to the detection failure, we cannot create a SCA and the subsequent steps fail accordingly.

In conclusion, we can say that the best approach to finding a matching packet of the attacker's conversation is by combining all three methods in the following order: compromised memory block search, jump target memory block search and exploit layer-2 frame search. This way we only use the more risky exploit layer-2 frame search when the other methods have failed.

6.1.2 SCA Verification

SCA verification can only be successful if there exists a SCA for the attack. In Table 1 we show that we are always successful in triggering the vulnerability when an SCA was available, which indicates that our replay method works accurately.

6.2 Performance

In the following, we want to measure the performance of verifying and generating packet-based SCAs as well as the overall performance of Scargos. We then compare our results with the results of Vigilante (Costa et al., 2005). We define performance as the speed in which a process finishes successfully.

We compiled the results of evaluating SCA generation and verification in Table 2. When evaluating exact stream replay, the results seem at first glance quite mixed and a general pattern is not easily found.

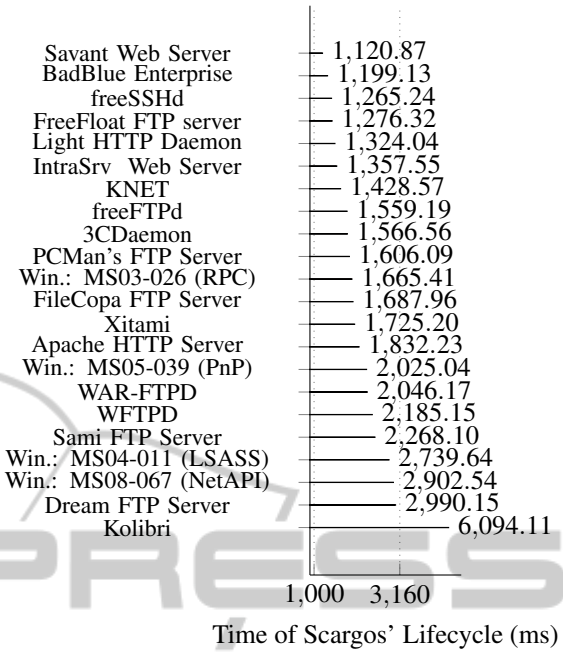

Figure 3: Total time from SCA generation to successful verification including distribution. The results are the average of 10 runs.

The application vulnerabilities that can be verified the fastest targets *FreeFloat FTP server*, while the slowest is targeting *Kolibri*. At first glance, there is not much difference between the two SCAs, as both need only one request to compromise the target. While *Kolibri's* SCA does carry twice as many bytes as *FreeFloat FTP server*, other attacks such as the attack against *freeFTPd* carry 20x more bytes and are still much faster than *Kolibri*. This implies that there is another factor that affects the performance of exact stream replay, which is the performance of the attacked applications and its time to process the attack.

Applications that should have a high performance and behave quite uniformly are system services offered by Windows itself. Yet, if we look at the performance of these applications, we see quite a fluctuation. The reason for this lies in our implementation. Exact stream replay waits 100*ms* after each request was sent for a reply from the server. This enables exact stream replay to be very accurate and protocol-independent, since the attacked application transitions into its next input state automatically. However, the downside of a timeout is poorer performance.

Figure 3 show the results of our experiments for Scargos' full life cycle. It shows that Scargos operates fast. The majority of vulnerabilities of our applications could be verified and thus detected in under 2 seconds. On average, it takes 1993.88*ms* from the first detection of an attack by a honeypot until all in-

Table 1: Success or Failure of each component/subcomponent to process its previous input correctly. The symbol ✹ signifies success, — signifies failure. Failure in packet extraction is either due to the fact that no logs were generated successfully or that no matching packet could be found in the session packet capture.

		Packet Extraction				Packet Replay
Application Name	Argos DTA	Jump Target Memory Block	Compro-mised Memory Block	Exploit Layer-2 Frame	Combination	Exact Stream Replay
Mercury/32 Mail	—	—	—	—	—	—
Mdeamon PRO	—	—	—	—	—	—
3CDaemon	✹	✹	—	—	✹	✹
Dream FTP Server	✹	✹	—	—	✹	✹
FileCopa FTP Server	✹	—	✹	✹	✹	✹
FreeFloat FTP server	✹	—	✹	✹	✹	✹
Sami FTP Server	✹	—	✹	✹	✹	✹
WAR-FTPD	✹	—	✹	✹	✹	✹
WFTPD	✹	—	✹	✹	✹	✹
Savant Web Server	✹	—	✹	✹	✹	✹
MS05-039 (PnP)	✹	✹	—	—	✹	✹
PCMan's FTP Server	✹	—	✹	✹	✹	✹
freeSSHd	✹	—	✹	✹	✹	✹
freeFTPd	✹	—	✹	✹	✹	✹
Apache HTTP Server	✹	✹	✹	—	✹	✹
BadBlue Enterprise	✹	✹	—	—	✹	✹
Light HTTP Daemon	✹	—	✹	✹	✹	✹
KNET	✹	—	✹	✹	✹	✹
Kolibri	✹	—	✹	✹	✹	✹
Xitami	✹	—	✹	✹	✹	✹
MS03-026 (RPC)	✹	—	✹	✹	✹	✹
MS04-011 (LSASS)	✹	—	—	✹	✹	✹
IntraSrv Web Server	✹	✹	✹	—	✹	✹
MS08-067 (NetAPI)	✹	—	✹	✹	✹	✹

terested SCA verifiers have verified the vulnerability on their systems. In the worst case, the process takes 6094.11*ms* (*Kolibri*), which should still give enough leeway to safely initiate a vulnerability response process.

6.2.1 Comparison to Vigilante

In this section we want to compare our solution to Vigilante in terms of performance. Scargos and Vigilante are not easily comparable because Vigilante's performance is worse the bigger the SCA file size, while Scargos's performance is mainly influenced by the number of layer-5 PDUs. The performance of the attacked application does also play a role for the performance of both frameworks. Additionally, comparison is hampered because Vigilante is not open-source or otherwise available.

In the original paper by Costa et. al (Costa et al., 2005), only three applications were evaluated. All three were either system services or a well-known windows application: *Microsoft SQL Server*, *Microsoft IIS Server* and *Microsoft Windows (MS03-026 (RPC))*. To enable a fair comparison we selected equally well-known applications and service vulnerabilities: *Apache HTTP Server*, *Microsoft Windows MS03-026 (RPC)*, *Microsoft Windows MS04-011 (LSASS)*, *Microsoft Windows MS05-039 (PnP)* and *Microsoft Windows MS08-067 (NetAPI)*. We compared the total time needed for SCA generation and verification in relation to SCA file size. The results are depicted in Figure 4.

We can see that Vigilante performs better for smaller SCA file sizes while Scargos performs better for bigger file sizes. Even taking into account that the SCA file size of Vigilante is truncated and in fact bigger than in the graph shown (reduction of about 20%), Scargos outperforms Vigilante. Our results from Table 2 and the data given by Costa et. al (Costa et al., 2005) indicate that Vigilante performs consistently better than Scargos in SCA verification, while SCA generation takes much more time in Vigi-

Table 2: Performance of stream extraction and exact stream replay in relation to number and size of layer-5 PDUs (L5-PDUs).

Application Name	Number of L5-PDUs	Length of all L5-PDUs (bytes)	SCA Generation (ms)	SCA Verification (ms)
3CDaemon	1	2055	927.12	171.08
Dream FTP Server	1	1049	488.5	1937.52
FileCopa FTP Server	3	815	475.34	238.68
FreeFloat FTP server	1	571	684.73	126.47
Sami FTP Server	1	567	696.28	822.31
WAR-FTPD	1	1031	697.91	890.91
WFTPD	3	579	471.96	727.71
Savant Web Server	1	401	487.75	164.26
Win.: MS05-039 (PnP)	13	3266	487.42	442.1
PCMan's FTP Server	1	5007	839.07	297.76
freeSSHd	1	20272	504.1	241.8
freeFTPd	1	20609	500.86	229.17
Apache HTTP Server	1	7145	851.3	498.38
BadBlue Enterprise	1	4285	537.43	185.54
Light HTTP Daemon	1	782	702.87	128.81
KNET	1	1039	716.1	242.09
Kolibri	1	1013	485.1	5103.76
Xitami	1	857	854.32	401.56
Win.: MS03-026 (RPC)	2	2272	488.12	207.06
Win.: MS04-011 (LSASS)	27	11287	980.18	494.18
IntraSrv Web Server	1	4792	505.89	180.28
Win.: MS08-067 (NetAPI)	36	7711	760.74	754.17

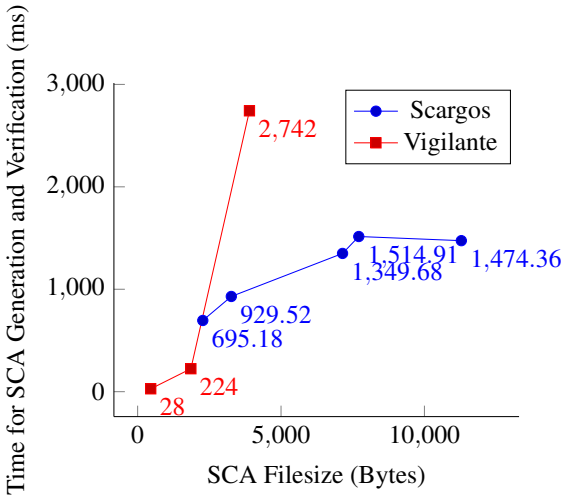

Figure 4: Comparing Vigilante's and Scargos' performance for the combined times of SCA generation and verification excluding distribution.

lante than in Scargos.

Although both data sets are hard to compare, an observation can be made that in our data set, the worst-case performance in Scargos is about 45% better than that of Vigilante, while Vigilante has a better best-case performance. Due to space limitations, additional results are presented in the full version.

7 CONCLUSION

In this work, we presented Scargos, a framework to distribute vulnerabilities in seconds without needing a central authority. We proposed packet-based SCAs, which are engineered to be distributed in the internet and differ from previously presented SCAs due to the fact that an attack is in its original form, in packets; furthermore, they are independent from the detection engine used and allow for a custom vulnerability response process. Packet-based SCAs re-

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

quire different verification methods compared to previously known approaches. We presented the replay mechanism "exact stream replay" as a way to verify SCAs. Our proof-of-cocept implementation used the DTA honeypot Argos and it was evaluated with 24 real-world attacks using exact stream replay. SCAs were successfully generated and verified for 22 of these attacks (the remaining 2 could not be detected by Argos), and the overall life cycle from detection to verification was shown to be on average less than 2 seconds. Finally, we showed that packet-based SCAs perform better for bigger SCA file sizes and seem to have a much better worst-case performance than conventional SCAs. For future work we plan to investigate accuracy and performance of client-side attacks and attacks on newer operating systems and mobile devices.

REFERENCES

Bailey, M., Cooke, E., Jahanian, F., Watson, D., and Nazario, J. (2005). The blaster worm: Then and now. *Security & Privacy, IEEE*, 3(4):26–31.

Bilge, L. and Dumitras, T. (2012). Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 833–844. ACM.

Bosman, E., Slowinska, A., and Bos, H. (2011). Minemu: The worlds fastest taint tracker. In *Recent Advances in Intrusion Detection*, pages 1–20. Springer.

Clause, J., Li, W., and Orso, A. (2007). Dytan: a generic dynamic taint analysis framework. In *Proceedings of the 2007 international symposium on Software testing and analysis*, pages 196–206. ACM.

Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Shannon, C., and Brown, J. (2004). Can we contain internet worms. In *Proceedings of the 3rd Workshop on Hot Topics in Networks (HotNets-III)*. Citeseer.

Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou, L., Zhang, L., and Barham, P. (2005). Vigilante: End-to-end containment of internet worms. In *ACM SIGOPS Operating Systems Review*, pages 133–147. ACM.

Crandall, J. R. and Chong, F. T. (2004). Minos: Control data attack prevention orthogonal to memory model. In *Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium on*, pages 221–232. IEEE.

Cui, W., Paxson, V., Weaver, N., and Katz, R. H. (2006). Protocol-independent adaptive replay of application dialog. In *NDSS*.

Faulhaber, J., Lambert, J., Probert, D., Srinivasan, H., Felstead, D., Lauricella, M., Rains, T., and Stewart, H. (2011). Microsoft security intelligence report. Technical Report 11, Microsoft Corporation, Redmond, WA 98052-6399.

Kohlrausch, J. (2009). Experiences with the noah honeynet testbed to detect new internet worms. In *IT Security Incident Management and IT Forensics, 2009. IMF'09. Fifth International Conference on*, pages 13–26. IEEE.

Kontaxis, G., Polakis, I., Antonatos, S., and Markatos, E. P. (2010). Experiences and observations from the noah infrastructure. In *Computer Network Defense (EC2ND), 2010 European Conference on*, pages 11–18. IEEE.

Kreibich, C. and Crowcroft, J. (2004). Honeycomb: creating intrusion detection signatures using honeypots. *ACM SIGCOMM Computer Communication Review*, 34(1):51–56.

Newsome, J. and Song, D. (2005). Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Network and Distributed System Security Symposium (NDSS 2005)*.

Portokalidis, G., Slowinska, A., and Bos, H. (2006). Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. In *ACM SIGOPS Operating Systems Review*, pages 15–27. ACM.

Provos, N. (2003). Honeyd-a virtual honeypot daemon. In *10th DFN-CERT Workshop, Hamburg, Germany*, volume 2.

Provos, N. and Holz, T. (2009). *Virtual honeypots: from botnet tracking to intrusion detection*. Addison-Wesley Professional, third edition.

Suh, G. E., Lee, J. W., Zhang, D., and Devadas, S. (2004). Secure program execution via dynamic information flow tracking. In *ACM SIGPLAN Notices*, pages 85–96. ACM.

Sullivan, B. (2004). Sasser infections begin to subside. *NBC News*. http://www.nbcnews.com/id/4890780/ns/technology_and_science-security/t/sasser-infections-begin-subside/#.UhANu3byrUI.

Venkataramani, G., Doudalis, I., Solihin, Y., and Prvulovic, M. (2008). Flexitaint: A programmable accelerator for dynamic taint propagation. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 173–184. IEEE.

Willems, C., Holz, T., and Freiling, F. (2007). Toward automated dynamic malware analysis using cwsandbox. *Security & Privacy, IEEE*, 5(2):32–39.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS