Signature Based Intrusion Detection for Zero-Day Attacks: (Not) A Closed Chapter?

Hannes Holm
Royal Institute of Technology (KTH), Sweden
hannesh@ics.kth.se

Abstract

A frequent claim that has not been validated is that signature based network intrusion detection systems (SNIDS) cannot detect zero-day attacks. This paper studies this property by testing 356 severe attacks on the SNIDS Snort, configured with an old official rule set. Of these attacks, 183 attacks are zero-days' to the rule set and 173 attacks are theoretically known to it. The results from the study show that Snort clearly is able to detect zero-days' (a mean of 17% detection). The detection rate is however on overall greater for theoretically known attacks (a mean of 54% detection). The paper then investigates how the zero-days' are detected, how prone the corresponding signatures are to false alarms, and how easily they can be evaded. Analyses of these aspects suggest that a conservative estimate on zero-day detection by Snort is 8.2%.

1. Introduction

Cyber security is a critical topic for both researchers and practitioners as successful cyberattacks can result in severe costs due to losses of confidentiality, integrity or availability. Various security mechanisms have been suggested for detecting cyberattacks; one of the more popular being intrusion detection systems (IDS) in general and network based intrusion detection systems (NIDS) in particular [1].

The plethora of NIDS detection methods and techniques that have been introduced are commonly categorized as either anomaly based (ANIDS) or signature (a.k.a. misuse) based (SNIDS) [2], [3]. Anomaly based schemes estimates the normal behavior of a system and generates alarms when the deviation from the normal exceeds some threshold [2]. Signature based schemes look for patterns (signatures) in the analyzed data and raise alarms if the patterns match known attacks [2].

Most NIDS that are commercially available and used in practice are signature based [4]. However, the large majority of research conducted on the topic in the past years concerns ANIDS [2], [5].

To support the large amount of research conducted on ANIDS it is often (e.g., in [2], [6]) stated that SNIDS cannot detect *zero-day attacks,* i.e., attacks (a.k.a. exploits) that utilize vulnerabilities that are unknown to the public community [7]. Zero-day vulnerabilities are especially attractive to attackers as their exploitation cannot be prevented by applying software security updates, and as a consequence the price of such a vulnerability can reach up to $250,000 [8].

For instance, in [2] it is stated that "*Signature-based schemes provide very good detection results for specified, well-known attacks. However, they are not capable of detecting new, unfamiliar intrusions, even if they are built as minimum variants of already known attacks"*; [6] states that "[...] *anomaly-based NIDS have one great advantage over signature-based ones: they can detect threats for which there exists no signature yet, including zero-day and targeted attacks*". Strangely enough, no empirical support for such claims can be found in literature.

So why does it need to be tested? After all, it might seem obvious that zero-day attacks cannot be properly detected by SNIDS as alarms only can be provided for attacks that match known signatures. The rationale is that new attacks sometimes have characteristics that already are covered in the SNIDS rule set. If a zero-day attack shares a trait with a publicly known previous attack, even a SNIDS would have a possibility to detect it. The question is how often this is the case in practice.

This paper studies the portion of zero-day attacks that the industry standard SNIDS Snort [9] is able to detect. The Metasploit Framework is utilized as a source for attacks and zero-day detection rate is measured by utilizing a Snort rule set older than the vulnerabilities corresponding to the tested attacks. To estimate the relative significance of the results, the outcome is compared to detection rate for attacks corresponding to vulnerabilities disclosed before the release of the utilized Snort rule set (hereafter referred to as *known attacks*). In total, 183 zero-day attacks and 173 known attacks were tested.

978-1-4799-2504-9/14 $31.00 © 2014 IEEE
DOI 10.1109/HICSS.2014.600

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 31,2024 at 05:03:31 UTC from IEEE Xplore. Restrictions apply.

The rest of the paper unfolds as follows: Section 2 describes related work. Section 3 describes the methodology utilized during the study. Section 4 presents and analyses the results. Finally, Section 5 concludes the paper.

2. Related Work

There are to the author's knowledge no valid or reliable studies regarding detection rate of zero-days' for SNIDS. However, there have been several empirical studies on other properties involving NIDS accuracy.

An extensive comparative study of 18 different IDSs was made during 1998 and 1999 by the Lincoln Laboratory at MIT [10], [11]. The authors found that the best IDSs detected between 63% to 93% of the tested known attacks and approximately 50% of the tested zero-days'. However, significant shortcomings of this study have been identified by [12] and [13] – shortcomings that delimits the usefulness of the results to a real-world scenario. For example, regarding the chosen attacks.

Hadžiosmanović et al. [6] studied the effectiveness of four significant ANIDS mechanisms at detecting various attacks (e.g., the dataset provided by [10], [11]). The authors found that the mechanisms could not provide both high detection and low false positive rates in presence of data with high variability.

Ktata et al. [14] studied the detection rate of Nmap probes for four different NIDS (Snort, Prelude, Tamandua, and Firestorm). The authors found that Snort, Tamandua, and Firestorm detected about 70% of the tested Nmap probes, while Prelude detected a bit less than 60% of them.

The effectiveness of five IDSs at detecting attacks against three different web applications was studied by Elia et al. [15]. This study included Apache Scalp, Anomalous Character Distribution (ACD) monitor, GreenSQL, Snort, and DB IDS. The authors found that Apache Scalp detected 18%, ACD monitor 59%, Green SQL 24%, Snort 56%, and DB IDS (with optimized settings chosen) 74% of the tested SQL injections.

To sum up, there are various empirical tests of different NIDS properties. However, there has been no reliable and valid test of SNIDS zero-day detection capabilities. Furthermore, a constant challenge for IDS tests is in regard to the attempted attacks – to gain useful results it is important that the tested attacks are representative for their purpose [16], [17]. This is a property that is difficult to achieve given typical project time and resource constraints.

This study analyses Snort detection rates for both zero-days' and known attacks using exploits in the Metasploit Framework. Furthermore, it utilizes a method for reliable analysis of a large set of the exploits in Metasploit, even though few vulnerable configurations are present in the experimental architecture.

3. Methodology

This chapter describes the experiment architecture, the attacks that were employed and how zero-days' were measured.

3.1. Overview of Experiment Architecture

Two physical hosts and a total of seven virtual hosts were employed in the experiment architecture. One physical host contained the five attacked virtual hosts (Windows 2000, Windows XP, Windows 2003, Windows Vista and Ubuntu 10.04) and the NIDS (a Windows XP host). The second physical host contained the virtual attacker host, a Backtrack 5 Linux system.

Each of the attacked hosts were configured with large amounts of different services running, for example, HTTP, SMB/Samba, SMTP, POP3, IMAP, WINS, FTP/TFTP, Telnet, MySQL, MSSQL and SSH. Furthermore, several different software providing the same (or similar) types of services were tested; for example, the HTTP servers Apache Tomcat, IIS HTTP and Apache HTTP were all part of the overall architecture. The NIDS was configured with both Wireshark and Snort 2.6. Pilot tests were conducted with both custom and existing Snort rules to ensure that everything was set up properly. Furthermore, the virtual machines were restored to previously saved states if required (e.g., when an attack had an impact on their functionality).

3.2. Used Exploits

In order to test the effectiveness of a NIDS, there is a need to have a useful set of attacks [17]. There are various sources that can be used to extract attacks, for example, Exploit-DB, Packet Storm, Canvas and the Metasploit Framework. This study uses Metasploit, due to its wide usage in practice (and thus arguably its representativeness of the interests in the security domain), and that the source codes of all exploits are available and standardized (and possible to easily revise).

The exploitation process of server software using Metasploit involves three properties that can be detected by SNIDS: (i) the attack code (in the case of a typical buffer overflow, an overly-long character

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 31,2024 at 05:03:31 UTC from IEEE Xplore. Restrictions apply.

sequence and a return address), (ii) the command payload (e.g., a reverse shell), and (iii) traffic generated by a successfully executed payload (e.g., activity of a dropper).

Executions of payloads and their activity are however typically rather uninteresting to study as the choice of payload generally is limited only to size (and given common dropper techniques such as the Metasploit stager, even size is rather irrelevant). Payloads can also be obfuscated in various ways, making them very difficult to detect for anomaly- and signature-based detection mechanisms alike [6], [18]. Furthermore, actions by successful payloads are often carried out through encrypted streams and thus better detected by agent-based IDS or black/white-lists of IPs (both being out of scope for this study). For these reasons, the present study only analyzes detection of attack code and negates detection of payloads and their activity. In practice, this was managed by revising exploit source code to make sure payloads were not sent. For instance, the exploit `dreamftp_format` appends the payload at the end of its injected sequence of characters (cf. Code 1). If ‘payload.encoded’ is removed, the exploit will still function as designed (redirect the control flow of the application to arbitrary code) – even though there is no code to execute.

Notably, for some exploits the payload is necessary to enable its execution. For instance, a buffer overflow exploit that has little space available for a payload might put it in the overflow itself. This is however not a problem for the study as the tested exploits calculate offsets and buffers depending on the size of the payload (in the context of this study: zero).

Testing whether an attack in Metasploit is detected by Snort is in theory a very simple task. That is, there is only a need to install vulnerable software in an appropriate environment and with an appropriate configuration, set Snort to monitor the network activity between the attacker host and the targeted host, then run the attack and observe whether there is an appropriate alarm produced by Snort. There is however a significant problem with this approach. In practice, testing all Metasploit attacks would mean installing hundreds of different software, in different environments and with various configurations. Clearly, resource constraints of most projects infer that only a small set of attacks can be attempted through such an approach, something that could end up biasing results. This project utilized an alternate approach that allowed reliable testing of Snort’s detection rate for most of the server attacks in Metasploit, even though vulnerable (or sometimes even correct) software were not always in place. This methodology is described in the remainder of this section.

In the employed version of Metasploit, there were 716 exploits. Of these 716 exploits, 423 concerned server side attacks and 293 client side attacks. This study is limited to the 423 server side exploits. Server side attacks in Metasploit can roughly be categorized in three categories, according to their ease of testing:

- Exploits that proceed as long as a session can be achieved.
- Exploits that proceed only when vulnerable software is in place.
- Exploits that proceed only when vulnerable software is in place and application-specific actions are successfully performed.

The first category is very simple to test out – this type of attack will trigger even though there is no vulnerable software at its destination. For example, an Apache HTTP exploit of this type would send all malicious content to an IIS HTTP server (giving the IDS a chance to produce alarms).

```
def check
    connect
    banner = sock.get(-1,3)
    disconnect
    if (banner =~ /Dream FTP Server/)
        return Exploit::CheckCode::Appears
    end
    #always return vulnerable state...
    #return Exploit::CheckCode::Safe
    return Exploit::CheckCode::Appears
end

def exploit
    connect
    select(nil,nil,nil,0.25)
    sploit = "\xeb\x29"
    sploit << "%8x%8x%8x%8x%8x%8x%8x%8x%"
    + target['Offset'].to_s + "d%n%n"
    #do not append any payload...
    sploit << "@@@@@@@@" #+ payload.encoded
    sock.put(sploit + "\r\n")
    select(nil,nil,nil,0.25)
    handler
    disconnect
end
```

Code 1. An excerpt of the revised version of the exploit `dreamftp_format`

The second category is a bit more difficult to test as there are constraints in the exploit prohibiting it to properly transmit all of its content if the correct product and version (and sometimes also configuration) is not in place. Some such exploits could be attempted without issues as the correct software were in the experiment architecture. However, others required revision to enable reliable testing. In other words, the source codes of exploits in this category were rewritten and software validation checks were removed. Revision of exploits during this

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 31,2024 at 05:03:31 UTC from IEEE Xplore. Restrictions apply.

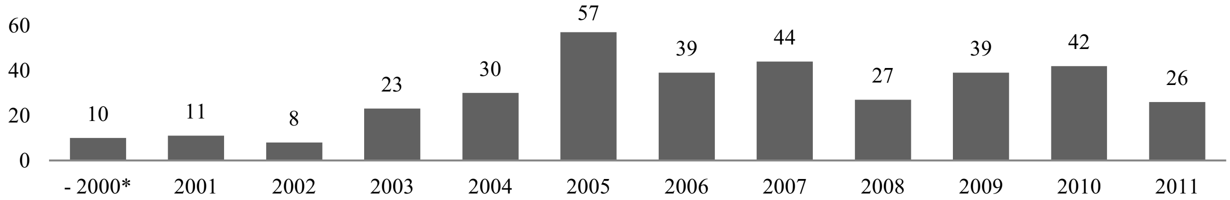

Figure 1. An overview of the tested attacks categorized according to the disclosure of their vulnerabilities. *An aggregation of data for 1990 (1 sample), 1994 (1), 1998 (2), 1999 (2), 2000 (4)

study can be categorized according to three scenarios.

In the first scenario, there is a software check, but only one type of exploit string is sent. This scenario was managed by simply removing the software check. Code 1 illustrates `dreamftp_format`, the simplest type of exploit revised according to this methodology. `dreamftp_format` conducts a banner grab, and if the banner matches Dream FTP Server the exploit proceeds; if not, the exploit is cancelled. The exploit itself starts a session with the targeted service, passes the malicious string *sploit* to the open socket and calls the Metasploit *multihandler* (that sets up a listener for the payload). The second type of scenario concerns when different exploit strings are sent depending on the result of a software version check (e.g., `hp_nnm_toolbar_02`). Given this scenario, the exploit string corresponding to the most recent software version was chosen. The third scenario concerns when different exploit strings are sent depending on the configuration of the targeted software. Given this scenario, the used exploit string was arbitrarily chosen from the available ones.

In total, 56 exploits had minor revisions to remove software validation checks (based on the three scenarios described above). Of these, 21 were zero-days' and 35 known (i.e., their vulnerabilities were disclosed prior to the release of the utilized Snort rule set - cf. Section 3.4).

The third category of attacks is the most difficult to test as they not only require the correct software and version, but also application-specific actions. For example, a web application exploit involving injecting malicious data through a cookie that is generated when querying the application (e.g., `manageengine_apps_mngr`) or exploits that involve application-specific authentication (e.g., `novell_netmail_auth`). Some of these attacks could be attempted without issues as the correct software was in the experiment architecture; however, others could not. While it is possible to conduct major changes to the source code of such attacks to enable testing, this was not done during the present study as there is risk of manual error and would require significant effort.

Of the 423 possible server attacks in the overall set, 67 were excluded due to this factor – resulting in a sample of 356 tested attacks. An overview of these 356 attacks according to the disclosure dates of their corresponding vulnerabilities can be seen in Figure 1. For instance, 57 of the used attacks correspond to vulnerabilities disclosed during 2005.

Some exploits in Metasploit are targeted against services that operate on ports not covered by the experiment architecture. For example, the exploit `realwin_scpc_initialize` targets a service operating on port 912. This exploit, and others like it, were instead targeted against appropriate services available in the architecture. This is however an issue for Snort as many alarms only signal for traffic against specific ports. To manage this problem, Snort rules for attacks against ports not covered by the exercise architecture were revised to instead alert for attacks against the employed ports. In summary, significant manual effort was spent to enforce the reliability of results.

3.3. Monitoring Attacks

The NIDS was configured with a combination of Snort 2.6 and Wireshark. Wireshark was used to confirm that the attacks had been attempted as designed (verified through comparisons of network data and exploit source code). An important aspect when measuring the effectiveness of Snort is in regard to the employed rule set. That is, there are multiple rule sets that can be employed, many that are developed by the Snort community. However, it is not known how commonly used such customized rule sets are in practice compared to the official rule set provided by the Sourcefire team [9]. Consequently, Snort was configured with an official rule set provided by the Sourcefire team. Another important factor is the type of Snort alarms that are studied. The default rule set of Snort grades alarms according to three levels of priority [19].

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 31,2024 at 05:03:31 UTC from IEEE Xplore. Restrictions apply.

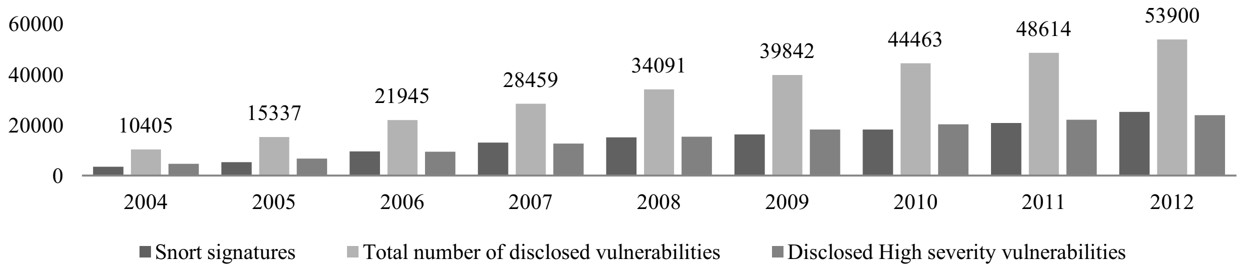

Figure 2. An overview of the evolution of Snort's official rule set along the number of disclosed vulnerabilities in the NVD (at the end of each year)

A priority of 1 signifies "*dangerous and harmful attacks*", for example, signature 2103 (a buffer overflow of Samba). A priority 2 alarm signifies "*suspicious signatures potentially preparing attacks*", for example, signature 3677 (denial of service attack against Ethereal). A priority 3 alarm signifies "*unusual traffic not identified as dangerous*", for example, signature 2465 (access to resources using Samba). Naturally, a less severe alarm is likely to provide more false alarms, and require greater rule customization to be useful. To make the results of the experiments useful to a real-world scenario, only alarms of priority 1 or 2 were seen as proper identifications of carried out exploits.

3.4. Studied Zero-Day Attacks

A zero-day attack is an attack for a vulnerability that is unknown to the public community [7]. An attack for a vulnerability discovered post the release of a SNIDS rule set is thus per definition a zero-day to the SNIDS as it is not publicly known at the release of the rule set. For instance, CVE-2007-0882 was disclosed the 12th of February 2007. Consequently, a default SNIDS rule set published prior to this date cannot possibly contain a custom signature for this particular vulnerability – unless the SNIDS rule vendor itself found it (and did not disclose it) – which is unlikely.

This study employed a Snort rule set released the 14th of November 2006[1]; any vulnerabilities disclosed post this date are thus per definition zero-days' to the rule set. This rule set was chosen as it would enable a satisfactory amount of zero-day samples in Metasploit (183 out of the 356 attacks), while at the same time allow a benchmark test with attacks corresponding to known vulnerabilities (173 out of the 356 attacks). That is, in order to properly analyze the effectiveness of a zero-day detection rate there is a need to compare it to the detection rate of known attacks.

A theoretical estimate of Snort's potency at detecting attacks can be gained by observing the vulnerability coverage of its rule set. There were 9128 signatures in the tested Snort rule set. At the time of the release of this rule set, there were 21166 vulnerabilities disclosed in the US National Vulnerability Database (NVD). Of these vulnerabilities, 9104 were of high severity as defined by the Common Vulnerability Scoring System [20]. A high severity vulnerability can loosely be seen as a vulnerability that can be remotely exploited to gain privileges of a host (e.g., user or admin). Metasploit exploits typically cohere to such vulnerabilities. These are also a focus area of the Snort rule set due to their severity. The ratio between Snort signatures, disclosed vulnerabilities and disclosed vulnerabilities of high severity seems to be rather consistent over time: the number of Snort alarms in the official rule set is about as large as the number of disclosed high vulnerabilities, and a bit less than half of the total number of disclosed vulnerabilities (cf. Figure 2). No formal conclusions regarding vulnerability coverage (and thus detection rate) should however be made from this dataset as multiple Snort rules can address the same vulnerability. Similarly, a single Snort rule sometimes covers multiple vulnerabilities. However, it serves to illustrate that the coverage of the chosen rule set is neither larger nor smaller than a typical Snort Sourcefire rule set.

4. Detection Rate of Zero-Day Attacks

This chapter address whether SNIDS can detect zero-days' from five different viewpoints: 1) overall zero-day detection rate, 2) comparison to detection of known attacks, 3) reasons behind detection, 4) possibility of false alarms and 5) possibility of an attacker evading the triggered signatures. The complete dataset is available for download[2].

[1] http://www.snort.org/vrt/docs/ruleset_changelogs/changes-2006-11-14.html

[2] www.ics.kth.se/snortdetection/snort_detection.xls

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 31,2024 at 05:03:31 UTC from IEEE Xplore. Restrictions apply.

4.1. Overall Results

A total of 31 zero-day attacks were detected by Snort (out of 183). There were a total of 39 *unique alarms* for these attacks, giving a mean of 1.29 unique alarms for each detected attack. In this paper, a unique alarm, or alert, means that the signature corresponding to this alarm only is counted once for each attack (there are sometimes multiple alerts triggered for the same signature during a single attack).

The detection rates for the zero-day attacks given different operating system environments and software services (as categorized by Metasploit) can be seen in Table 1. The different operating system environments denote the available payloads corresponding to each exploit. For example, 135 exploits had payloads for Windows operating systems (i.e., attacks that could be used to compromise Windows hosts). The environment *Multi* denotes exploits that have payloads for multiple operating systems. Some combinations did not have any tested exploits. These are denoted as '-'. The service *Other* corresponds to a set of 23 different services that each had very limited amount of exploit samples.

On average, one sixth of the zero-day exploits are detected by Snort. A t-test shows that this statistic is significantly larger than zero ($p = 2.83 \cdot 10^{-9}$). Thus, a signature based NIDS *can* detect zero-day exploits.

Table 1. Zero-day detection rate given different operating system environments and software services (sample sizes are given within brackets)

	Detection rate			
Service	Total	Windows	Unix	Multi
Total	17% (183)*	17% (135)	20% (40)	0% (8)
FTP/TFTP	85% (13)	90% (10)	67% (3)	-
HTTP	10% (49)	12% (42)	0% (4)	0% (3)
Web applications	25% (16)	-	25% (16)	-
SMB/Samba	75% (4)	75% (4)	-	-
SMTP/POP3/IMAP	75% (4)	100% (3)	0% (1)	-
Other [a]	5% (97)	4% (76)	13% (16)	0% (5)

* $p = 2.83 \cdot 10^{-9}$

[a] 23 different services

4.2. How does Zero-Day Detection Compare to Detection of Known Attacks?

There were 93 known attacks that were detected (out of 173), with a mean of 1.65 unique alarms for each detected attack. These statistics are slightly more favorable for actual prevention of attacks than the zero-day statistics: more unique alarms means that the individual monitoring the SNIDS will have a greater opportunity to spot the attack.

Detection rates for the known attacks given different operating system environments and services can be seen in Table 2. All detection rates except for SMTP/POP3/IMAP are higher than the corresponding zero-day detection rates. The low sample size in this category (10 for known attacks and 4 for zero-days') could be the reason behind this curious result. The overall detection rate is approximately three times higher than for the zero-days'.

Table 2. Detection rate of known attacks given different operating system environments and software services (sample sizes are given within brackets)

	Detection rate			
Service	Total	Windows	Unix	Multi
Total	54% (173)	55% (125)	50% (42)	50% (6)
FTP/TFTP	92% (26)	95% (22)	67% (3)	100% (1)
HTTP	24% (38)	26% (35)	0% (3)	-
Web applications	60% (15)	-	60% (15)	-
SMB/Samba	82% (11)	83% (6)	80% (5)	-
SMTP/POP3/IMAP	60% (10)	67% (9)	0% (1)	-
Other[a]	49% (73)	53% (53)	40% (15)	40% (5)

[a] 32 different services

4.3. How Are Zero-Day Attacks Detected?

This section analyses the properties of the alarms provided by Snort for the detected zero-day attacks in order to investigate the reasons behind discovery. As a basis for analysis, the detected attacks are categorized according to the vulnerabilities they exploit. The industry standard vulnerability classification system Common Weakness Enumeration (CWE) [21] is employed for this purpose. Detailed descriptions of all CWE's discussed in this chapter can be found in [21].

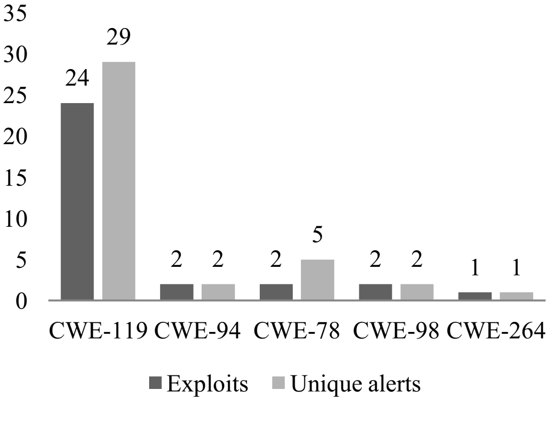

Figure 3. An overview of exploited zero-day vulnerability types and number of unique alerts tripped for these exploits

The 31 detected zero-day attacks correspond to six different CWE categories: *Buffer Error* (*CWE-119*),

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 31,2024 at 05:03:31 UTC from IEEE Xplore. Restrictions apply.

Command Injection (*CWE-78*), *Code Injection* (*CWE-94*), *PHP File Inclusion* (CWE-98), and *Permissions, Privileges, and Access Control* (*CWE-264*). An overview of the detected exploits along these categories, and the number of unique Snort signatures tripped for these, can be seen in Figure 3. Buffer error, a.k.a. buffer overflow, with a total of 24 exploits, is the most commonly detected type of exploited vulnerability. This does however not suggest that detection of buffer overflow is more efficient than detection of other exploit types as buffer overflow is dominant in the overall sample of used exploits as well.

Buffer Error (*CWE-119*), concerns when software performs operations on a memory buffer and can read from - or write to - a memory location that is outside of the intended boundary of this buffer. A total of 29 unique alerts were tripped for the 24 exploits corresponding to this vulnerability type. These alerts can be classified into five different categories:

The most common type of unique alert for buffer overflow (15/29, or 51.7%) checks a specified byte-position or range following a call to a certain protocol function (that is found anywhere in a packet payload). If there is data at the specified position or outside of the range, then an alert is triggered. In other words, valid arguments of such a function are not expected to exceed a certain size. For instance, signature 1529 alerts if there still is data 100 bytes after the FTP command 'SITE'.

The second most common type of unique alert (8/29, or 27.6%) can roughly be classified as *inquiry of vulnerable resource*, which includes use of sensitive commands (e.g., the TFTP 'PUT' command from an external address) and inquiries of certain files (e.g., '/webadmin.dll' on HTTP).

Three of the 29 unique alerts (10.3%) were for NOP's (No OPeration instructions) – each for a long set of consecutive '90's (an x86 assembly language 1-byte instruction that does not affect the program state), 'A's (`inc ecx`) and 'C's (`inc ebx`). 'A' act as a NOP if the register `ecx` is not used by the exploit; 'C' if `ebx` is not used. NOP's are typically used to 'slide' the program counter to the payload, or as for the detected exploits: as junk data to overflow a buffer.

Two unique alerts (6.9%) concern authentication bypass attempts; these are triggered as two exploits interact with FTP's authentication mechanism.

Finally, one unique alert (3.5%) not only checks whether there is data at a specified byte-position following a specific command (TFTP 'PUT'), but also whether a string terminator ('00') is present.

Code Injection (CWE-94) concerns when software allows a user's input to contain code syntax that can allow an attacker to alter the intended control flow of the software. Of the two alarms corresponding to this vulnerability type, one concerns inquiry of a vulnerable resource ('/Setup.php'), and one concerns a long set of consecutive NOP's ('90's). Interestingly, the latter alert was given for the exploit `ms10_061_spoolss`, one of the zero-days' that Stuxnet utilized. In other words, Snort's default rule set had detection possibility of Stuxnet long before the malware was discovered in the wild.

Command Injection (CWE-78) concerns when operating system commands can be invoked using externally-influenced input to the software. The two exploits concerning this category received a total of five unique alerts, all corresponding to inquiries of vulnerable resources (e.g., '/calendar.php' or '/admin.php').

PHP File Inclusion (CWE-98) concerns when a PHP application receives input from an upstream component, but incorrectly restricts the input before its usage in "require," "include," or similar functions. The two alerts given for the two exploits of this type were both for inquiries of vulnerable resources (weblogic/tomcat '.jsp' and '/upload.php').

Permissions, Privileges, and Access Control (CWE-264) are related to the management of permissions, privileges, and other security features that are used to perform access control. The only used exploit corresponding to this type of vulnerability is `vsftpd_234_backdoor`; this exploit concerns a malicious backdoor that was introduced to the vsftpd-2.3.4.tar.gz archive. An alert concerning authentication bypass was triggered as the exploit sends a malformed 'USER' argument (':)').

4.4. What About False Alarms?

A critical property of NIDS detection concerns false alarms: A signature prone to alert for legitimate traffic is difficult for an operator to trust.

Each Snort signature has a documented qualitative evaluation of its overall rate of false positives. Thus, this study analyzes these documents in order to gain overall estimates for rate of false positives. An overview of these results can be seen in Figure 4.

The majority (69%) of signatures given for the zero-day exploits are denoted to have no known false positives. Alerted signatures with denoted false positives typically correspond to *inquiry of vulnerable resource*, which was discussed in the previous section. For instance, usage of the TFTP 'PUT' command from an external address can certainly be non-malicious in some scenarios. NOP-type signatures for '90' are also considered to have possible false positives due to frequent existence of such NOP's in transfers of binary data.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 31,2024 at 05:03:31 UTC from IEEE Xplore. Restrictions apply.

It is however important to keep in mind that the actual false positive rate greatly depends on the employed architecture. Given a scenario where the used application protocols rarely employ NOPs, this type of signature could be considered trustworthy.

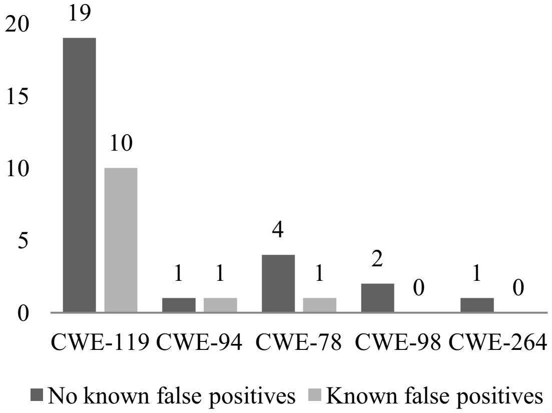

Figure 4. Documented false positive rates for the alerts triggered by zero-days'

Another aspect that need be considered is the type of software that the signatures correspond to - a common method for reducing false alarms in practice is to disable signatures corresponding to software that are not employed in the monitored architecture.

A total of 47% of all triggered alerts for zero-days' correspond to generic software (e.g., any x86 software or a generic FTP server). These are likely (but not necessarily) enabled in real-world scenarios. The remainder of alerts (in total 53% of all alarms) corresponds to specific software such as WU-FTPD. Not even a single one of these alerts correspond to the actually exploited software, suggesting that they might be disabled in real-world scenarios.

It should however be mentioned that all alarms (including the generic) were for the correct types of software products. For example, `phpmyadmin_config` is an exploit for the software phpMyAdmin. The alarm for this exploit denoted an attack against the software MediaWiki. While these are different software, their application type (i.e., PHP) is the same.

4.5. Signature Evasion Techniques

One significant question is how difficult it is for an attacker to circumvent the signatures triggered for the zero-days'. If they are easily evaded by minor changes to the exploits then their usefulness is limited: any attacker capable of zero-day attacks is likely also capable of various intrusion detection evasion techniques.

To the author's knowledge, there is no useful taxonomy of the effort required to bypass different SNIDS signatures. Thus, for this purpose this research classifies the triggered signatures in two categories: signatures that are *simple to evade*, and signatures that are *difficult to evade*.

An example of a simple to evade rule is signature 1390, which triggers when an overly-long set of 'C's is spotted. If the content of the overly-long sequence used to produce the buffer overflow is not executed, the 'C's could be replaced by arbitrary data. If it is executed, it could be replaced with other (combinations of) NOP's such as 'FN' (`inc,dec esi`), 'AI' (`inc,dec ecx`), or return-oriented programming (ROP) instructions that serve the same purpose.

An example of a difficult to evade rule is signature 1972, which triggers when there is data 100 bytes after the FTP command 'PASS' (i.e., a password should not exceed 100 characters). This signature is very difficult to circumvent for a buffer overflow exploit of 'PASS' as FTP does not support encoding of variables (such as 'PASS') and the vulnerable buffer itself most likely requires (at least) 100 characters to overflow.

An overview of these results can be seen in Figure 5. Most of the signatures would be difficult to evade for an attacker, especially given buffer overflow attacks. The main exception is signatures that trigger based on queries for certain resources on web applications. Most of these are easily circumvented as HTTP requests supports URL encoding. For instance, '/' can be URL encoded as '%2F'.

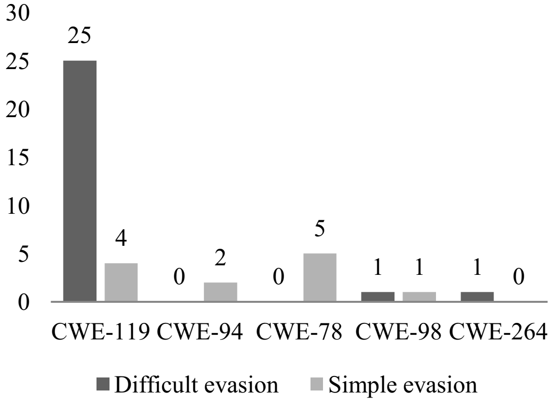

Figure 5. Possibility of evading zero-day signatures

5. Conclusions and Future Work

This chapter first presents a conservative estimate on zero-day detection rate. It then discusses the results in from three different viewpoints, namely: impact for researchers, impact for practitioners and limitations involved in the study.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 31,2024 at 05:03:31 UTC from IEEE Xplore. Restrictions apply.

5.1. A Conservative Estimate on the Effectiveness of Zero-Day Detection

While the actual detection rate of the tested zero-days' was 17%, this number does not consider the possibility of false alarms or signature evasion techniques. A more conservative estimate of detection rate can be gained by only considering attacks detected by signatures that have no known false positives and are difficult to evade (as presented in Section 4.4 – 4.5); these are presented as *effective signatures* in Figure 6. As can be seen, an overall of 48.8% of all alerts can be considered effective. Thus, a conservative estimate on the overall detection rate by Snort for zero-day attacks is 8.2%.

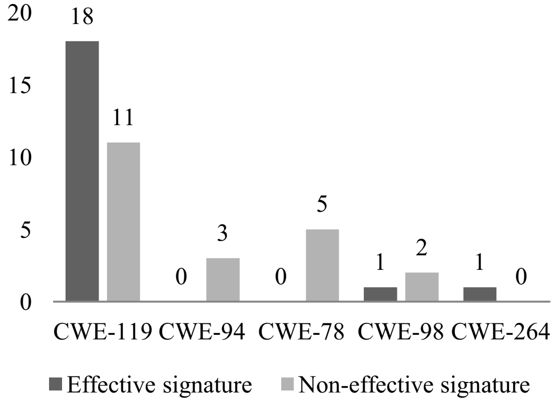

Figure 6. Signatures that have low false alarm rates and are difficult to circumvent

5.2. Impact for Researchers

Research on intrusion detection typically concerns ANIDS [4], much due to the frequent claim that SNIDS cannot detect zero-days' [2]. This study clearly highlights the error with this claim; the observed detection rate for zero-days' is significantly larger than zero.

However, while detection is evident, SNIDS is not able to provide complete detection of either known attacks or zero-days'. Future research on intrusion detection should reflect upon these observations; the results can be seen as a base-line for what any proposed zero-day detection mechanism must detect to be of any use (as SNIDS such as Snort typically are employed in practice [4]).

This paper also provides a method for reliable generation of attacks, something that there is no practiced standard for [17]. The foundation for this approach lies with the industry standard exploitation framework Metasploit as it enables a useful interface for low-effort testing of attacks representative to the interests of the community. However, in order to utilize Metasploit with little effort many exploits need to be rewritten (as they by default do not properly execute without vulnerable software and configurations present). Fortunately, this process does not require significant effort – each exploit revised during the present study required roughly between 5 and 15 minutes to rewrite. A continually updated set of valid and easy-to-use exploits would significantly benefit both researchers and practitioners as it would enable valid and reliable low-effort testing of IDS detection rates. To complement such a set of exploits, tests could also be automated to some degree. To implement such an approach is left to future work.

A means to improve the Snort rule set in regard to inquiries of vulnerable resources could be to add different encoding options for rules. If a certain encoding option, such as URL encoding, is set for a rule, then any content matching specified in the rule is conducted for all possible encoding/non-encoding combinations of that content. The performance impact of a valid rule of this type could however certainly be questionable, especially if several encodings (such as both hex and URL) are necessary to test.

Buffer overflow signatures that trigger based on starting commands and consecutive byte-lengths seem rather robust. Future research might thus benefit from focusing on this type of signature.

5.3. Impact for Practitioners

The overall detection rate for known attacks (54%) might seem alarmingly low; especially as most of the tested exploits could lead to compromised systems with admin/root privileges. This figure is however biased from the poor detection of attacks against (generally) uncommon software located on non-standard ports. For example, the exploit `message_engine_heap`, a buffer overflow for CA BrightStor ARCserve (operating on port 6071), was missed Snort. A scenario where no such software is present would yield significantly higher detection rates. Similarly, this study did not include detection of payloads and their activity. Including these would likely yield higher estimates on detection rates (assuming that a host-based detection mechanism is present).

This study also shows that Snort is capable of detecting zero-days'. It is however questionable if an overall detection of one sixth of all attacks, or 1/12 given the conservative estimate, is sufficient for a real-world scenario - other mechanisms should be implemented to complement SNIDS.

Finally, the observed zero-day detection rates could also be seen as a baseline for how well SNIDS that

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 31,2024 at 05:03:31 UTC from IEEE Xplore. Restrictions apply.

rarely are updated perform; a practice that unfortunately seems common in practice.

5.4. Limitations

There are more than 20,000 high severity vulnerabilities available today, but only exploits corresponding to 356 such flaws were tested by this study. Thus, the sample size might be too small to allow completely valid results. It is nonetheless important to recognize this is significantly larger than what has previously been used in evaluations of SNIDS effectiveness (for instance, [10] only tested 58 different attack types – many that were not of high severity).

Another potential bias is that the chosen Snort rule set is significantly more or less potent than the average Snort rule set. This does however not seem likely, especially as the rule set seems to improve at a rather predictable rate (cf. Figure 2). Nevertheless, the observed detection rates for known attacks should be interpreted with care as they are unlikely to fully reflect the current standard of the default Snort rule set.

6. References

[1] M. Sumner, "Information security threats: a comparative analysis of impact, probability, and preparedness," *Information Systems Management*, vol. 26, no. 1, pp. 2–12, 2009.

[2] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers and Security*, vol. 28, no. 1–2, pp. 18–28, 2009.

[3] E. Biermann, "A comparison of Intrusion Detection systems," *Computers & Security*, vol. 20, no. 8, pp. 676–683, Dec. 2001.

[4] M. A. Faysel and S. S. Haque, "Towards Cyber Defense : Research in Intrusion Detection and Intrusion Prevention Systems," *Journal of Computer Science*, vol. 10, no. 7, pp. 316–325, 2010.

[5] P. Kabiri and A. A. Ghorbani, "Research on Intrusion Detection and Response: A Survey," *International Journal of Network Security*, vol. 1, no. 2, pp. 84–102, 2005.

[6] D. Hadžiosmanović, L. Simionato, D. Bolzoni, E. Zambon, and S. Etalle, "N-Gram against the machine: on the feasibility of the n-gram network analysis for binary protocols," in *Research in Attacks, Intrusions, and Defenses*, 2012, pp. 354–373.

[7] L. Bilge and T. Dumitras, "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*, 2012, pp. 833–844.

[8] C. Miller, "The legitimate vulnerability market: Inside the secretive world of 0-day exploit sales," in *Workshop on the Economics of Information Security*, 2007.

[9] Sourcefire, "Snort," 2013. [Online]. Available: An intrusion-detection model. [Accessed: 12-Apr-2013].

[10] R. Lippmann et al., "Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation," *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, pp. 12–26, 1998.

[11] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 DARPA on-line intrusion detection evaluation," *Computer Networks*, vol. 34, 2000.

[12] J. McHugh, "Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory," *ACM Transactions on Information and System Security*, vol. 3, no. 4, pp. 262–294, Nov. 2000.

[13] M. Mahoney and P. Chan, "An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection," in *Recent Advances in Intrusion Detection*, 2003, pp. 220–237.

[14] F. B. Ktata, N. El Kadhi, and K. Ghédira, "Agent IDS based on Misuse Approach," *Journal of Software*, vol. 4, no. 6, pp. 495–507, Aug. 2009.

[15] I. A. Elia, J. Fonseca, and M. Vieira, "Comparing SQL Injection Detection Tools Using Attack Injection: An Experimental Study," in *21st International Symposium onSoftware Reliability Engineering (ISSRE)*, 2010, pp. 289–298.

[16] B. I. A. Barry and H. A. Chan, "Intrusion detection systems," in *Handbook of Information and Communication Security*, vol. 2001, no. 6, P. Stavroulakis and M. Stamp, Eds. Springer, 2010, pp. 193–205.

[17] P. Mell, V. Hu, R. Lippmann, J. Haines, and M. Zissman, "An overview of issues in testing intrusion detection systems," *Citeseer*. National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, 2003.

[18] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo, "On the infeasibility of modeling polymorphic shellcode," *Machine learning*, vol. 81, no. 2, pp. 179–205, 2010.

[19] S. Riebach, E. Rathgeb, and B. Toedtmann, "Efficient deployment of honeynets for statistical and forensic analysis of attacks from the internet," in *NETWORKING*, 2005, pp. 507–517.

[20] P. Mell, K. Scarfone, and S. Romanosky, "A Complete Guide to the Common Vulnerability Scoring System (CVSS), Version 2.0, Forum of Incident Response and Security Teams." 2007.

[21] Mitre, "Common Weakness Enumeration," 2012. [Online]. Available: http://cwe.mitre.org/. [Accessed: 12-Apr-2013].

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 31,2024 at 05:03:31 UTC from IEEE Xplore. Restrictions apply.