So Long, and Thanks for Only Using Readily Available Scripts

1. Introduction

There is a general belief that it has become easier to conduct cyber attacks and that little skill or knowledge is required to compromise systems. In particular, it is often argued that the technical expertise required to mount successful cyber attacks is declining as a consequence of new offensive resources, such as attack scripts and user-friendly attack execution tools, becoming available in the public domain. These resources include, for example, exploit databases such as Packet Storm and Exploit Database, and attack execution frameworks such as Metasploit, Armitage and BurpSuite.

This theory is in referenced by various works in the cyber security domain. For example, Liu and Chen (Liu & Cheng 2009) state that "*attack tools and techniques have become increasingly automated and sophisticated over the past decade, whereas the technical knowledge and skills required to use these tools has decreased dramatically*". The same trend is described by the NATO (Robinson 2016), among others. Liu and Chen (Liu & Cheng 2009) even suggests that "*all it takes is the basic ability to follow instructions and push buttons*". An attacker with this low level of skill, knowledge or ambition is often referred to as a "*script kiddie*". Simmonds et al. (Simmonds et al. 2004) refer to a script kiddie as "*someone who uses already established and part automated techniques in attacking a system*"; Hald and Pedersen (Hald & Pedersen 2012) refer to the script kiddie as "*one who relies on premade exploit programs and files ("scripts") to conduct his hacking, and refuses to bother to learn how they work*". Most other taxonomies and ontologies that characterize hackers describe script kiddies similarly (Rogers 2006).

The idea that script kiddies and other novice cyber criminals represent an increasing threat due to the increased sophistication and automation is widespread and influence decision making on many different levels. For example, ENISA (Lévy-Bencheton et al. 2015) considers the script kiddie as threat agent that can realize some 30 different cyber threats and are, with the exception of physical attacks, relevant to the same type of threats as nation states. Others' have suggested that anybody could use exploit kits to steal online banking credentials (Choo 2011).

The present paper aims to analyze whether unsophisticated cyber attacks that a typical script kiddie is able to perform actually pose a threat to systems today. This is summarized by the research question (RQ) stated below:

RQ: *How difficult is it for a script kiddie to succeed with cyber attacks*?

More specifically, the paper empirically examines whether the current standard of tools indeed do provide novices, or script kiddies, with the means to mount successful cyber attacks against presumably vulnerable network services. A total of 1223 server-side attacks involving 45 different exploit modules are executed against presumed vulnerabilities identified by network vulnerability scanners. The test allows estimation of the probability that a script kiddie can compromise a machine given that a vulnerability is found in it and that an exploit is readily available.

The rest of the paper is formulated as follows. Section 2 presents related work. Section 3 describes the methodology of the study. Section 4 presents the results. Section 5 discusses these results, and section 6 concludes the paper.

2. Related work

In general, a cyber attack involves two activities: (i) the identification of a vulnerability, and (ii) the exploitation of this vulnerability, e.g., to gain remote control of the target system (Holm et al. 2015; McQueen et al. 2006).

As described by (Rogers 2006; Simmonds et al. 2004; Liu & Cheng 2009; Hald & Pedersen 2012), a script kiddie is limited to utilizing existing tools for these activities. On overall, the usability of such tools is arguably the same as for any other software. ISO 9241 defines usability as "*the effectiveness, efficiency and satisfaction with which specified users achieve specified goals in particular environments*".

To the authors' knowledge, no tests have been conducted regarding the effectiveness, efficiency or satisfaction of offensive cyber tools for attackers in general, or for script kiddies in particular. There are however empirical studies of somewhat related variables. The remainder of this section describes such works.

Regarding vulnerability discovery, Holm et al. (Holm et al. 2011; Holm 2012) studied the accuracy of automated network vulnerability scanners. Their results indicated that vulnerability scanners on overall have rather few false positives but many false negatives, and that their accuracy significantly improves if they are allowed to log in to the targeted system with administrator/root privileges.

As previously stated, availability of complete exploit code is imperative to attacks performed by script kiddies. Some works analyze cyber attacks in the wild given the presence or absence of public exploit code, e.g., due to different vulnerability disclosure strategies.

Allodi et al. (Allodi et al. 2013; Allodi & Massacci 2012) studied the occurrence of exploits in the wild through five datasets: (i) the overall number of public vulnerabilities of different severity (using the US National Vulnerability Database, NVD[1]), (ii) vulnerabilities that have public exploits (using Exploit Database, EDB[2]), (iii) Symantec's threat databases of vulnerabilities that are exploited in the wild[3], (iv) observations of actual data gathered in the wild by Symantec (Worldwide Intelligence Network Environment, WINE (Dumitras & Shou 2011)), and (v) vulnerabilities actively exploited by exploit kits gathered by the authors' (EKITS (Allodi & Massacci 2012)). Among other things, the authors' observed that presence of an exploit in EKITS was correlated with the volume of attacks observed in WINE, and that the availability of exploit code in EDB was a poor indicator on whether the corresponding vulnerability had an entry in Symantec's threat database.

Ransbotham et al. (Ransbotham et al. 2012) examined whether vulnerability disclosure through a market mechanism (Zero Day Initiative or iDefense) or open mechanism (e.g., Secunia or Security Focus) influence exploitation by using intrusion detection alerts as a proxy for attacks. They found that market-mechanisms did not reduce the likelihood that a vulnerability would be exploited. They also found that markets increased the time to first exploit as well as decreased the overall volume of alerts.

In a later similar study, Mitra and Ransbotham (Mitra & Ransbotham 2012) examined the diffusion of cyber attacks (i.e., how widespread they are) against systems when the exploited vulnerabilities have been disclosed through full disclosure or limited disclosure. The authors' found that full disclosure reduced the delay in the attack diffusion process as well as increased the risk of first attack on any specific day after the vulnerability is reported. While the culprits of the studied attacks were unknown, the study does suggest that increased exploit availability reduces the time until intrusions are attempted.

While exploit availability is necessary for script kiddies, it is also imperative that they understand how to configure an exploit to maximize its likelihood of success. An attack tool that has a user-friendly interface which is able to present detailed technical information in a means that is understandable would also make it easier to understand how to appropriately configure exploits. We are unaware of works that examine offensive cyber security tools in this regard. However, there are works that study the ease of use of defensive cyber security tools, in particular regarding visualization (Goodall 2009). These tests indicate that proper visualization increases the usability of cyber security tools. Thus, cyber attack tools with graphical user interfaces, such as Armitage, could increase the effectiveness of script kiddies at compromising systems.

Another aspect of relevance for the success rate of a script kiddie attack is the reliability of the exploit. In other words, if the exploit works without manually tuning it for the victim's particular configuration. This is important as small changes of the target environment often require adjustments of the exploit. As a script kiddie is presumed to be unable to manually overcome such hurdles, he/she requires exploits that are robust against this variation. For instance, the return address (EIP) used by a buffer overrun exploit in a kernel service of a Windows 7 machine might be located at different offsets for different language packs. If this is the case, a well-designed exploit should include a run-time check for language pack and pick a configuration that is relevant to the result of the check.

The research presented in this paper address this third aspect and investigates the reliability of exploits against machines with different configurations. Only one previous empirical study on this topic was found. In that study, Dondo et al. (Dondo et al. 2015) tested exploit reliability by repeatedly executing twelve exploits against eight Windows machines of different type. They found that exploit reliability was highly dependent on whether the machine was rebooted between exploits, but that the payload delivery mechanism (Reverse TCP or Bind TCP) was unimportant for the reliability. With reboots, eight of twelve exploits worked every time, while four did not work at all. The reasons for why four exploits failed every time was not identified. The present study continues along this path, but involves a significantly larger sample of target machines and exploits, as well as greater automation and control.

[1] nvd.nist.gov/

[2] www.exploit-db.com/

[3] www.symantec.com/security_response/landing/azlisting.jsp

3. Method and materials

One major reason behind the lack of empirical analyses of cyber attacks is the difficulty for researchers to obtain observational data from live networks and systems. Such data are typically deemed too sensitive for public research. Furthermore, because it is next to impossible to completely distinguish malicious activity from benign activity in an operational network, such data do not offer a ground truth to run security tests against.

These issues have been acknowledged by the scientific community. For example, a roadmap for experimental cybersecurity research is presented in (Balenson et al. 2015). This roadmap stress, among other things, that the community needs shared, validated models and tools that help researchers to rapidly design meaningful experiments and test environments.

Fortunately, recent advances in virtualization technology and the advent of large-scale laboratory computer environments, often called cyber ranges, offer alternative means of obtaining empirical data and conducting experiments. In cyber ranges, systems and software can be configured similarly to the real world, and attacks can be carried out in a controlled manner without influencing any operational business. Cyber ranges can therefore be used to generate data suitable for testing both methods for vulnerability assessment and methods for detecting malicious behavior. This is demonstrated by a number of published papers that use data generated in cyber ranges, for instance (Mirkovic et al. 2008; Holm et al. 2012; Sommestad & Sandström 2015).

This paper used the cyber range CRATE to host the experiment. The tool AutoVAS was used to identify vulnerabilities in machines and the tool SVED (Holm & Sommestad 2016) to run exploits against these machines. An overview of the design and execution of the experiment is given in Figure 1. Step 1 in Figure 1 concerns the instrumentation of target machines using the management tool CRATEweb (cf. section 3.1) and the identification of vulnerabilities within these machines using AutoVAS (cf. section 3.2). Step 2 concerns choosing exploits that reference the vulnerabilities which were identified during step 1 (cf. section 3.3). Finally, step 3 concerns a process for reliably executing all discovered exploits using SVED (cf. section 3.3).

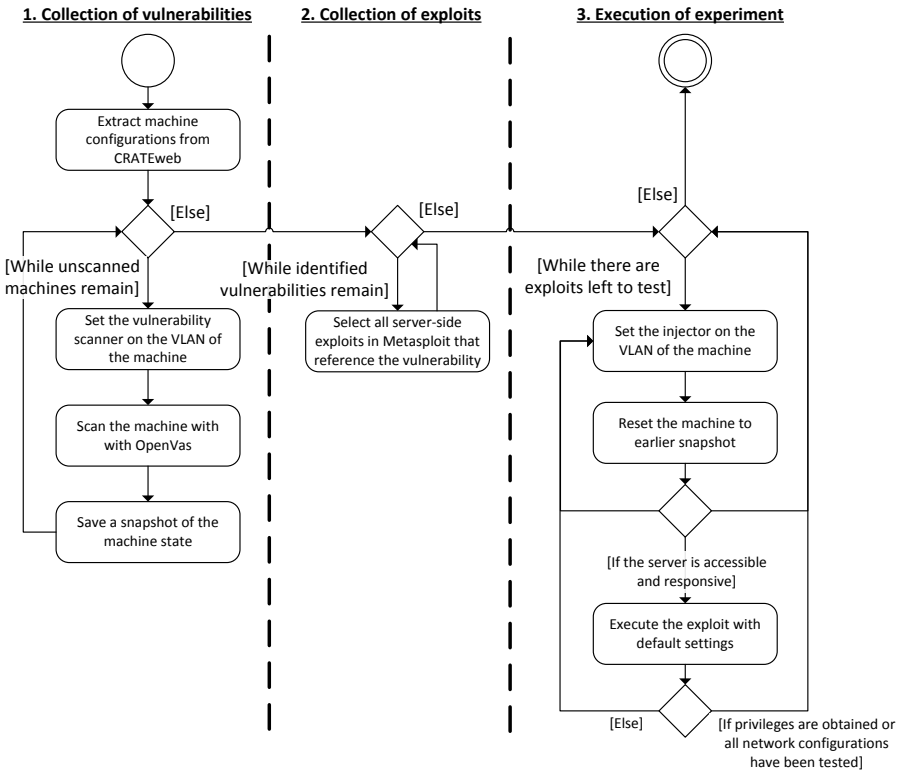

Figure 1. Experiment design and execution

3.1. Target machines

The Cyber Range And Training Environment (CRATE) is a cyber range built and maintained by the Swedish Defence Research Agency (FOI). It has been used for a number of research studies related to vulnerability assessments and situational awareness (Sommestad 2015). Physically, CRATE consist of approximately 750 servers, a number of switches and various auxiliary equipment (e.g., for remote access). During an experiment these servers are instrumented with between one and twenty VirtualBox machines of various types. Instrumentation is straightforward for any type of operating system and application software that VirtualBox can handle. In addition to the typical enterprise software, simplified versions of social network services, industrial control systems and search engines are available in the cyber range.

Instrumentation is either scripted or managed using a web-based graphical user interface called CRATEweb. This web interface allows the researcher to configure operating systems, application software, firewall configurations, network topologies, users, windows domains, and more. A set of scripts then deploy virtual machines that match the design to the 750 physical servers.

CRATE was used as the base experimental platform for conducting tests. Approximately 350 virtual machines (VM) were instrumented for the test. None of these 350 systems were created for the experiment presented in this paper. They were created by a plethora of persons for various purposes during the years the cyber range has been operational. Most of them was created for use in cyber defense exercises. A large number of different operating systems are included among these 350 machines, such as Windows-based (e.g., XP, 7, Server 2003 and Server 2012) and Linux-based (e.g., Ubuntu, Kubuntu, Gentoo, Debian and CentOS). As several of the machines were created several years before the test as potential targets in exercises, many contain a large number of software vulnerabilities, making them suitable for the test at hand. Out of the 350 machines, 329 were vulnerable and 204 contained vulnerabilities with matching exploits in Metasploit. Vulnerability identification is described in section 3.2; choice of exploit modules is described in section 3.3. An overview of these machines is described in Table 1.

Table 1. Overview of machines used during the experiment

Operating System	VMs	Exploits	Exploit modules
Debian	7	46	19
Fedora	1	4	4
Gentoo	37	497	31
Red Hat	3	24	17
Ubuntu	99	858	19
Windows Server 2003	3	12	6
Windows Server 2008	2	2	1
Windows 7	11	22	2
Windows XP	41	98	11
Total	**204**	**1563**	**110**

3.2. Vulnerability identification

As previously described, script kiddies rely heavily on publicly available tools when they search for vulnerabilities. OpenVAS is an open source vulnerability scanner which, arguably, is the most readily available vulnerability scanner today. This experiment used the tool AutoVAS, which has been developed to supervise and control vulnerability scans in CRATE using OpenVAS.

AutoVAS consists of one manager and several agents that perform the actual vulnerability scans. The manager is responsible for keeping track of agents as well as administrate what scans that different agents should carry out. The latter is accomplished through a load-balancing function that distributes the desired scans evenly between all agents. Scans are generated based on machine inclusion/exclusion lists that are defined within a web-based graphical user interface. Basic machine information (such as machine names and network interface details) is automatically gathered by queries to CRATEWeb.

Each AutoVAS agent conducts authenticated scans of their provided machines and stores the results in a database. Authenticated scans are not realistic from a script kiddie perspective, as they rely on administrator access to the scanned systems. This type of scan profile is used in the present study as we do not want to focus on the performance of vulnerability scanners (this has been done previously by Holm et al. (Holm et al. 2011; Holm 2012)), but on the performance of the exploitation process.

Before each AutoVAS scan is carried out the agent ensures that the targeted machine is in a valid state. For example, it ensures that the machine is powered on, that the administrator/root credentials are correct, that existing active directory service connections are not broken, and that the remote registry services of Windows systems are running (given that the machine is running Windows). The agent attempts to correct any issue that it identifies. If the system already is in a valid state, or can be put into a valid state, the scan is carried out. When a scan report has been successfully parsed, the machine state is saved in a VirtualBox snapshot. This enables relating a machine to a known vulnerable state even if it is changed in the future.

Out of the 350 machines instrumented in CRATE for the present study, 329 were vulnerable. They contained a total of 108 373 vulnerabilities, i.e., on average 329 vulnerabilities per machine. The number of vulnerabilities in a vulnerable machine varied greatly, from a few to several thousand.

3.3. Exploit execution

This test used the software tool SVED (Holm & Sommestad 2016) (Scanning, Vulnerabilities, Exploits, and Detection) to execute exploits and measure their effect on machines in CRATE. SVED uses the configuration tool CRATEweb (cf. section 3.1) to obtain information on machines in the cyber range and AutoVAS (cf. section 3.2) to obtain information regarding their vulnerabilities. SVED also has interfaces to other services not employed for the present study, e.g., a threat intelligence database called Centaur.

Within SVED, the machines in CRATE will appear as either attack systems that can be used to carry out actions, as detector systems that can be used to detect malicious activity (e.g., Snort), or as potential victims of attacks. SVED offers several features that supports for cyber security experimentation.

First, SVED offers a standardized interface for actions that can be carried out during an experiment. Four types of actions are available in SVED: reconnaissance (e.g., port scans), execution of exploits (e.g., the exploit modules of the Metasploit Framework), interacting with compromised machines (e.g., running shell commands), and auxiliary actions (e.g., relocating an attack machine to another VLAN, interfacing with the VirtualBox API, emulating a USB drive or directly starting an application within a machine).

Second, SVED allows users to generate attack graphs, graph-structures of actions connected by AND/OR gates, in the tool. This can be done either manually in a graphical user interface, automatically using predefined scripts, or by interacting with a REST API.

Third, SVED ensures that an experiment is carried out in an efficient and controlled manner. Before an experiment is started, SVED validates that each participating attack system has the necessary functionality to participate in the experiment. For example, that it contains the desired set of actions, and that all required services are running. If an experiment configuration is deemed valid, it continues by loading the complete experiment design, including all actions and their relations. During an experiment, SVED distributes commands to the participating attack systems. Each attack system carries out its given instructions and periodically report the status of the actions that they currently are responsible for back to SVED. This enables multiple actions to be executed in parallel, effectively reducing the duration of experiments linearly with the number of attack systems that are utilized.

The remainder of this section describes the experiment design of the present research. More specifically, it describes how offensive actions were chosen and instrumented, how attack graphs were constructed, and how the experiment was controlled.

In terms of offensive actions, the experiment focused on server-side exploits available in Metasploit that provided administrative privileges. The default settings were used for all exploits.

The use of Metasploit is motivated by its popularity and accessibility. In addition, it is integrated into user-friendly tools such as Armitage, which script kiddies are likely to use.

The focus on server-side attacks is motivated by their ease of use. Server-side attacks do not require any user interaction and are therefore attractive to script kiddies. In addition, it is more straightforward to test sever-side attacks than client-side attacks, which also require interaction on the client-side. Automating this interaction for a large number of applications is non-trivial. For example, the API command to have Firefox access a web server is different than that of Internet Explorer, and starting a video is done differently on VLC media player than on Windows Media Player.

The use of default settings is motivated by the limited technical knowledge of script kiddies. The number of settings that can be configured for exploits are often large, and to customize a configuration requires knowledge. For instance, the exploit module *exploit/solaris/samba/lsa_transnames_heap* has 52 settings that can be altered. To specify most settings, such as which SMB pipe name to use, deep technical understanding of the exploit as well as the target application is required. We assume that script kiddies do not possess such knowledge, and only alter what is required to run an exploit. Typically, there is only a need to specify the address of the target for Metasploit exploit modules as all other settings have technically valid default parameters. One important setting that should be mentioned is Metasploit's concept of "Target", i.e., a configuration within an exploit that correspond to a certain application version or environment. For instance, *exploit/solaris/samba/lsa_transnames_heap* allows the user to choose between two targets (x86 and SPARC). This study used Metasploit's default target for all exploits, which typically meant "Automatic targeting" (i.e., a run-time check for which configuration to choose).

Execution of exploits was the basis for the attack graphs created for the experiment. However, a number of other actions were also included in the attack graphs. More specifically, execution of an exploit was preceded by three actions which ensured that the victim was in a valid state before exploitation was attempted:

1. The victim was first restored to its corresponding AutoVAS snapshot for which vulnerability information is known.
2. When necessary, the VLAN of the attack system was changed to the same VLAN as the victim machine.
3. A ping-check was made to ensure that the system was alive and reachable.

The third action could fail because the machine has multiple network interfaces, or because the vulnerable service only listened on specific network interfaces. For this reason, the process was repeated for other network interfaces if necessary. When an exploit succeeded, or there were no more network interfaces available, the execution flow was directed to the next exploit for the victim. When there were no more exploits available for a victim machine, the execution flow was directed to the next victim.

The exploitation process is described by step 3 in Figure 1 and exemplified for a partial attack graph for a machine in Figure 2. In short, the exploitation process ensured that all possible exploits were tested under valid conditions.

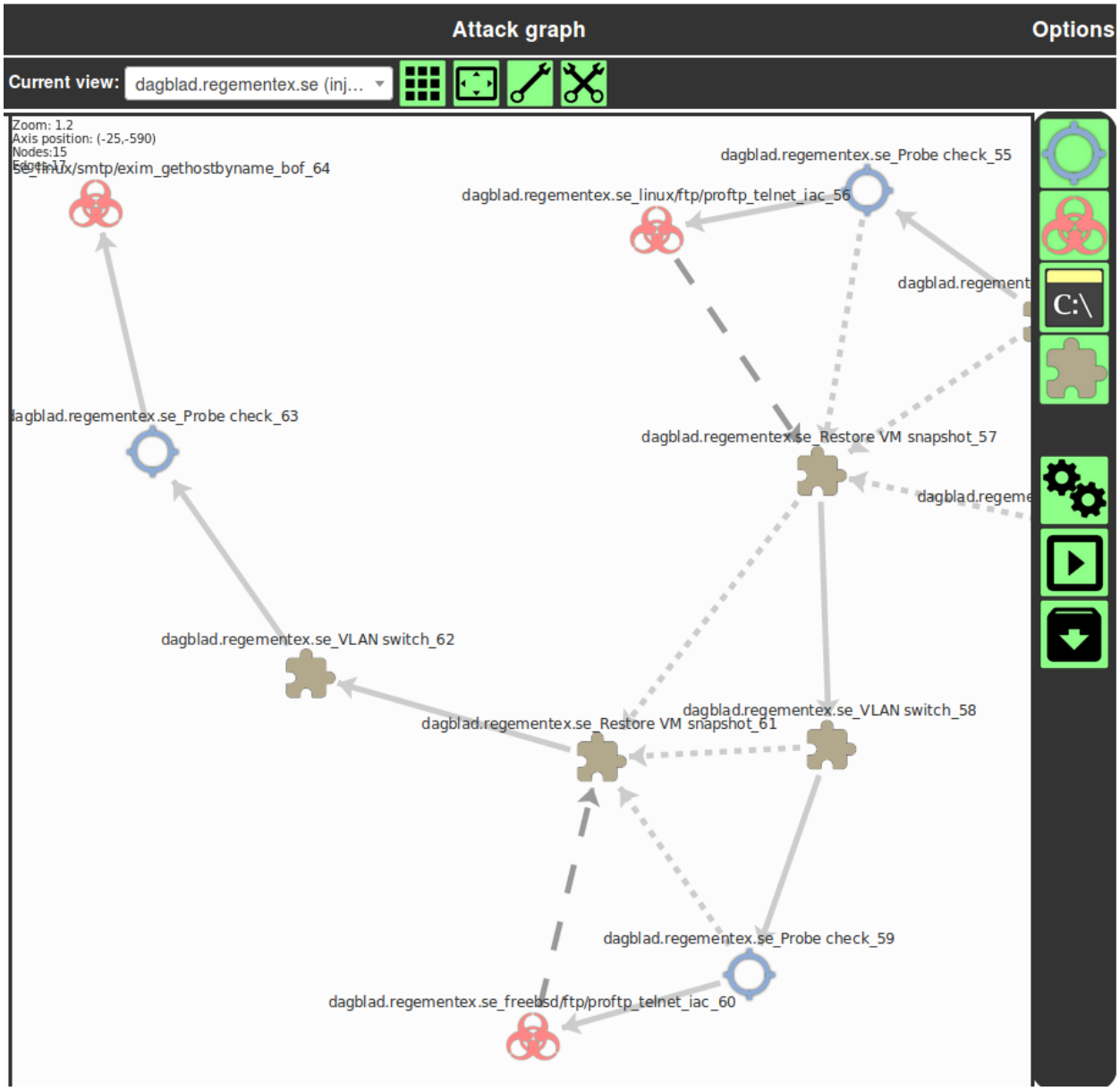

Figure 2. Partial attack graph for a target machine

As described in Table 1, 204 of the 329 vulnerable machines had server-side vulnerabilities with available exploit modules in Metasploit. A total of 1563 exploits were part of the experiment, corresponding to a 45 different exploit modules. Some of these exploit modules were valid for multiple different operating systems, amounting to an overall of 110 different operating system/exploit module pairs. The complete attack graph contained 5514 actions, including actions that enforced experiment reliability (see **Fel! Hittar inte referenskälla.**).

4. Results

The experiment was initialized the 20th of May 2016 and ended approximately 30 hours later during the 21st of May. SVED ensured that the targets of the attack graphs were distributed to the attack systems and executed in a

controlled manner. A total of 26331 log entries were recorded by SVED during the experiment. The entire dataset as well as the attack graph configuration are available for download[4].

The remainder of this section is formulated as follows. Section 4.1 provides an overview of the results and answers the RQ "*How difficult is it for a script kiddie to succeed with cyber attacks?*". Section 4.2 describes a high-level analysis of the data to identify general trends that can help explain the result. Section 4.3 describes a low-level manual technical analysis of a subset of all conducted exploits to understand why these exploits in particular succeeded/failed.

4.1. Overall results

Out of 1223 attempted exploits, a mere eight succeeded. These eight successful exploits all corresponded to the exploit *ms08_067_netapi* (CVE-2008-4250). *ms08_067_netapi* was successful every time it was attempted, while none of the 44 other exploit types that were tested were successful at all. This result clearly suggests that script kiddies in general should have a hard time succeeding with server-side attacks.

4.2. High level analysis

There are several possible reasons for why so few of the exploits succeeded. One reason could be that the employed vulnerability scanner (OpenVAS) is more prone to false positives then a general vulnerability scanner. In other words, that many of the reported vulnerabilities did not exist in the target systems. However, this seems a bit farfetched given the fact that it builds on similar architecture as its competitors, that have been shown to have rather few false positives, especially when used in authenticated mode (Holm et al. 2011; Holm 2012). This is however further explored in section 4.3.

Another reason could be related to Metasploit's exploit reliability ranking system: each exploit module in the Metasploit framework has a reliability rank on a scale from one to seven based on its potential impact to the target system (Beardsley 2013). An exploit with rank 7 (Excellent) "*will never crash the service*", whereas an exploit with rank 1 (Manual) "*is unstable or difficult to exploit and is basically a DoS*". It can be expected that exploits with a higher rank more often are successful. An overview of the reliability ratings for the 45 tested exploits is shown in Figure 3. *ms08_067_netapi*, the only exploit that worked, has a reliability rank of *Great*. As can be seen, the majority of tested exploits have a rank of Great or Excellent. Thus, the Metsploit exploit reliability ranking system offer little help explaining why *ms08_067_netapi* worked, while all others' failed.

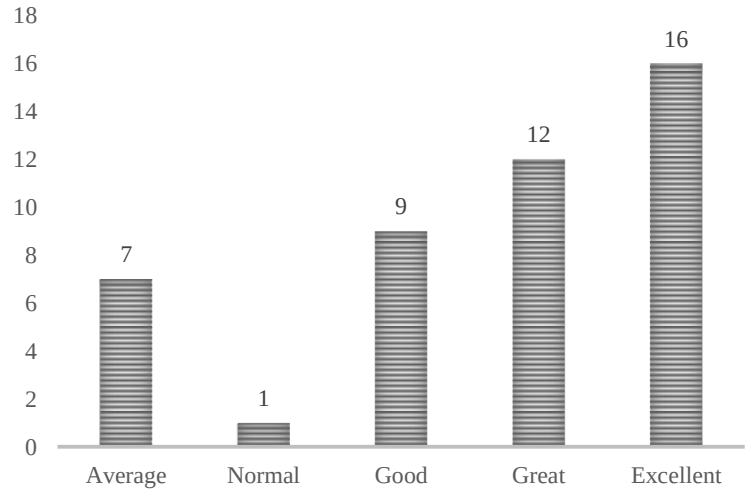

Figure 3. Metasploit exploit reliability ranks for the tested exploits

Naturally, there are many important aspects that currently are not covered by Metasploit's reliability ranking system. One such variable could be related to an exploits robustness with respect to small configuration changes of

[4] ftp://download.iwlab.foi.se/sved/

the target. As described in section 3.3, the parameter "Target" describe a specific exploit configuration. For example, a buffer overflow exploit might encapsulate buffer offsets and return addresses for different target environments (e.g., FreeBSD 8.0 and FreeBSD 8.1) into different target configurations. While there is a one-to-many relation between an exploit target and a victim configuration (a single exploit target can cover multiple victim configurations), it does provide an indication of how well an exploit is able to handle different victim configurations.

This robustness may be the primary statistical explanation for the success of *ms08_067_netapi* and the failure of the other exploits. *ms08_067_netapi* has 72 targets, while the 44 other tested exploits have 1-9 targets, with a mean number of two targets. Thus, *ms08_067_netapi* can be expected to have received significantly higher development effort than the other exploits. This could be a consequence of its large attack surface - vulnerable software include Microsoft Windows 2000, XP, Server 2003, Vista, Server 2008, and 7 (Pre-Beta). It could also be due to its wide prevalence in the wild. *ms08_067_netapi* has been exploited by prolific malware such as Stuxnet and Conficker. In fact, it was the only non-zero-day attack used by Stuxnet. On the other hand, its effectiveness could also be the reason for why it has been so widely exploited.

4.3. Low level analysis

This section is divided in two parts. First, it examines the circumstances where cyber attacks succeeded. Then, it examines circumstances where attacks failed.

An overview of the eight machines that were compromised by *ms08_067_netapi* is given in Table 2. All compromised machines were running either Windows Server 2003 or Windows XP (which employ essentially the same kernel) with network-shared folders enabled. The exploit was executed in two configurations (out of the 72 available) - one target for the six Windows XP SP1 machines, and another for the two Windows Server 2003 machines. Each vulnerable machine was thus appropriately configured to enable successful exploitation.

Table 2. Compromised machines

Machine compromise	Operating System	Service Pack
2016-05-20 17:54:13	Windows XP	SP1
2016-05-20 18:57:03	Windows XP	SP1
2016-05-20 22:04:13	Windows Server 2003	SP2
2016-05-21 05:03:15	Windows Server 2003	SP2
2016-05-21 06:14:36	Windows XP	SP1
2016-05-21 08:27:23	Windows XP	SP1
2016-05-21 11:49:41	Windows XP	SP1
2016-05-21 14:50:46	Windows XP	SP1

The remainder of this section analyses the exploits that failed. In order to obtain valid technical explanations, there is a need to manually study machine configurations, exploit source codes, vulnerability information, network captures, as well as OpenVAS scan reports. As analyzing all 1215 failed exploits through such means would require too much manual effort, this research utilizes a randomized sample of 20 exploits as a base for this analysis. While this sample is too small for any general conclusions, it should provide some insight into why the exploits failed. An overview of the results from this analysis is given in Table 3.

Table 3. Reasons for failure of exploits

Reason for failure	Count
Incorrect translation between CVE and exploit module	14
False positive by scanner	4
Missing exploit target	1
Error with victim	1

The most common cause (70% of all cases) for failure was a poor matching between the vulnerabilities (CVEs) reported by scanners and the vulnerabilities referenced by exploits in Metasploit. In other words, that a vulnerability was correctly identified by a scanner and then incorrectly translated to an exploit module invalid for the victim configuration.

For example, CVE-2014-6271, often known as "*Shellshock*" as it concerns remote code execution in the UNIX Bash shell, was correctly identified by OpenVAS for many Linux-based machines. Metasploit has several exploits that concern this vulnerability, such as *linux/http/advantech_switch_bash_env_exec* and *multi/http/apache_mod_cgi_bash_env_exec*. As all of these exploits concern CVE-2014-6271, a match is made regardless on whether the software server that expose the Shellshock vulnerability is present. Similarly, there are several exploit modules in Metasploit that concern CVE-2012-6329, a remote code execution vulnerability in Perl. These exploits were invalid as the web applications that exposed this vulnerability (Foswiki and Twiki) did not exist on the victims.

The second most common cause (20% of all cases) concerned false positives by the vulnerability scanner OpenVAS. All four false positives were very simple errors that should be easily mitigated by developers of OpenVAS. For example, a Gentoo kernel security patch was correctly identified as missing by OpenVAS. However, CVE-2007-4370 was referenced as one of the vulnerabilities that are mitigated by this patch. As CVE-2007-4370 concerns a vulnerability in a racing game for Windows, it has little to do with the Gentoo kernel. A false positive rate of 20% is slightly higher than what has been observed for competing commercial vulnerability scanners (Holm et al. 2011; Holm 2012).

One exploit, *windows/ftp/warftpd_165_pass*, did not succeed even though the victim was in a vulnerable configuration (e.g., version 1.65 and with anonymous logon allowed). A confirmatory manual attempt to exploit this vulnerability was made, also without success. The reason behind this was shown to be a lack of exploit target for the victim configuration. This exploit has a Metasploit reliability rating of Average, i.e., the lowest rating possible.

Finally, one exploit failed due to the target server being in a partially broken state. This server was only responsive to some basic commands.

5. Discussion

This section is divided in two parts. Section 5.1 discusses the results of the study regarding the stated RQ, namely, "*How difficult is it for a script kiddie to succeed with server-side cyber attacks?*". Section 5.2 discusses the reliability and validity of the results.

5.1. How difficult is it for a script kiddie to succeed with cyber attacks?

On overall, the results from this study indicate that script kiddies given the current status of publicly available tools rarely would be able to succeed with server-side cyber attacks. On the other hand, the results also indicate that if a vulnerability scanner identifies a system as being vulnerable to CVE-2008-4250, attack success is next to assured.

The statistical analysis of exploit success rate could not offer any sound explanations to why most exploits failed, and *ms08_067_netapi* always succeeded. The manually analyzed sample of successful and failed exploits however provided some preliminary indications that could help explain the result. The reasons behind exploit failures were primarily related to vulnerabilities that were correctly identified by scanners, but then incorrectly translated to inapplicable exploit modules. It can be expected that it is a non-trivial matter for a script kiddie to understand this connection. This lack of understanding could also lead to a script kiddie executing multiple unsuccessful attacks against non-vulnerable servers. Defenders could use this knowledge to pinpoint the competence of an adversary. In particular, a defender could create specific intrusion detection rules that correspond to attacks on kernel vulnerabilities that have multiple public exploits available, where each exploit concern a certain server that expose this vulnerability (and none of the vulnerable servers exist). If such a rule is triggered, it could indicate a script kiddie attempting to compromise a machine.

Some commercial vulnerability scanners such as Nexpose automatically couple identified vulnerabilities to exploit modules in Metasploit. It would be valuable to reexamine the findings from the present study using such a tool.

The sample suggest that exploits rarely fail based on errors or lack of features in the exploit source code. However, this could be a different story if the translation from CVE to exploit module was more reliable.

5.2. Validity and reliability

There are several issues regarding validity and reliability that need be addressed.

Authenticated vulnerability scans were used. In practice, a script kiddie would have to rely on unauthenticated scans, which are more prone to false negatives (but have a similar rate of false positives). Authenticated scans were chosen as the focus of this research was not to identify the accuracy of vulnerability scanners, but the reliability of exploits. It is however important to view the results in this light: in practice, it is likely that even fewer attacks would have been successful than what was observed.

The experiment covered a small number of machines and configurations. A mere 204 machines and 45 unique exploits were part of the experiment design, with a total of 1223 carried out exploitation attempts. While these numbers are larger, and the configurations more diverse, than any previous controlled experiment on the topic, they are limited. There is thus a need to conduct further controlled experiments to reexamine the findings of the present study.

Virtualization can impact the success of an exploit. Some software modules can be mapped to slightly different memory offsets when run in a virtualized environment such as VirtualBox. This can in theory cause exploits that depend on the existence of specific memory addresses, such as buffer overrun exploits, to fail. That said, as this issue rarely is reported by practitioners or researchers it is likely a minor problem.

Some tools couple vulnerabilities to exploit modules in Metasploit. The present study coupled CVEs identified by OpenVAS to CVEs corresponding to exploits in Metasploit. This translation process was the primary cause for exploit failure. There are commercial vulnerability scanners such as Nexpose that provide this mapping. It is likely that there would be fewer errors in the translation process from vulnerability scanner output to exploits if such a tool is used. However, it should also be noted that these tools are not as readily available to novices as OpenVAS. The validity of such a test with respect to a script kiddie profile could thus be questioned.

The manual technical analysis only covered a small subset of all carried out exploits. While the technical analysis (cf. section 4.3) provide some reasonable explanations for the result, the sample is very limited. A larger

sample size would not only provide more credible conclusions, but also enable a better understanding of why exploits that are carried out given valid circumstances fail. It is nevertheless important to recognize that such manual analysis is expensive; large scale analyses of this kind require automation to be feasible.

Server-side attacks only constitute part of the script kiddie arsenal. The present research only tested server-side attacks, while client-side attacks such as drive-by-downloads and e-mails with appended malicious files also are easy to employ for novices. One reason for focusing on server-side attacks was the effort required to reliably test client-side attacks. However, it might be possible to test a substantial portion of the client-side attacks in Metasploit by focusing on a few well-chosen software that have similar interaction mechanisms, such as Adobe Flash (29 exploits in Metasploit) and Internet Explorer (59 exploits). We aim to reexamine the findings from the current study using client-side exploits in future work.

An exploit can disrupt a machine, making consecutive attacks on this machine invalid. Previous researchers (Dondo et al. 2015) have attempted to tackle this issue by rebooting machines between attacks. This is an imperfect method as an exploit could permanently alter a software configuration stored on disk. The present study uses a more reliable method - to restore each machine to a known vulnerable snapshot before an exploitation attempt is made.

6. Conclusions and future work

This study examined how difficult it is for a script kiddie to succeed with server-side cyber attacks by running 1223 exploits corresponding to 45 different exploit modules against 204 machines in a cyber range.

The results are optimistic from a defenders perspective: a mere eight attacks were successful, and it is likely that even fewer would be successful if the employed vulnerability scanners would not be allowed to authenticate to victims during scans. Exploits failed primarily due to poor relations between vulnerabilities (CVEs) and exploit modules, and rarely due to reliability issues with the exploits themselves. This result could however be different if a commercial vulnerability scanner that has built-in support for this translation was used. We aim to reexamine the findings of the present study using such a tool in future work. This might also enable a greater understanding of the reliability of the exploits themselves, rather than the choice of correct exploit.

Finally, an important topic not examined by this work is client-side attacks, which certainly also could be employed by script kiddies. We aim to study such attacks in future work.

7. References

Allodi, L. & Massacci, F., 2012. A preliminary analysis of vulnerability scores for attacks in wild: the ekits and sym datasets. In *Proceedings of the 2012 ACM Workshop on Building analysis datasets and gathering experience returns for security*. pp. 17–24.

Allodi, L., Shim, W. & Massacci, F., 2013. Quantitative assessment of risk reduction with cybercrime black market monitoring. In *Security and Privacy Workshops (SPW), 2013 IEEE*. pp. 165–172.

Balenson, D., Tinnel, L. & Benzel, T., 2015. *Cybersecurity Experimentation of the Future (CEF): Catalyzing a New Generation of Experimental Cybersecurity Research*,

Beardsley, T., 2013. Metasploit reliability ranking system. *Github*. Available at: https://github.com/rapid7/metasploit-framework/wiki/Exploit-Ranking [Accessed June 9, 2016].

Choo, K.-K.R., 2011. The cyber threat landscape: Challenges and future research directions. *Computers & Security*, 30(8), pp.719–731.

Dondo, M., Risto, J. & Sawilla, R., 2015. Reliability of exploits and consequences for decision support.

Dumitras, T. & Shou, D., 2011. Toward a standard benchmark for computer security research: The Worldwide Intelligence Network Environment (WINE). In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. pp. 89–96.

Goodall, J.R., 2009. Visualization is better! a comparative evaluation. In *Visualization for Cyber Security, 2009. VizSec 2009. 6th International Workshop on*. pp. 57–68.

Hald, S.L.N. & Pedersen, J.M., 2012. An updated taxonomy for characterizing hackers according to their threat properties. In *Advanced Communication Technology (ICACT), 2012 14th International Conference on*. pp. 81–86.

Holm, H. et al., 2011. A quantitative evaluation of vulnerability scanning. *Information Management & Computer Security*, 19(4).

Holm, H. et al., 2015. P CySeMoL: Predictive, Probabilistic Cyber Security Modeling Language. *Dependable and Secure Computing, IEEE Transactions on*, 12(6), pp.626–639.

Holm, H., 2012. Performance of automated network vulnerability scanning at remediating security issues. *Computers & Security*. Available at: http://www.sciencedirect.com/science/article/pii/S0167404811001696 [Accessed November 21, 2012].

Holm, H., Ekstedt, M. & Andersson, D., 2012. Empirical analysis of system-level vulnerability metrics through actual attacks. *IEEE Transactions on Dependable and Secure Computing*, 9(6).

Holm, H. & Sommestad, T., 2016. SVED: Scanning, Vulnerabilities, Exploits and Detection. In *MILCOM*.

Lévy-Bencheton, C. et al., 2015. *Threat landscape and good practice guide for internet infrastructure*, ENISA.

Liu, S. & Cheng, B., 2009. Cyberattacks: Why, what, who, and how. *IT Professional Magazine*, 11(3), p.14.

McQueen, M. et al., 2006. Time-to-compromise model for cyber risk reduction estimation. *Quality of Protection*. Available at: http://www.springerlink.com/index/JP46737M7466870N.pdf [Accessed June 23, 2010].

Mirkovic, J. et al., 2008. Testing a Collaborative DDoS Defense In a Red Team/Blue Team Exercise. *IEEE Transactions on Computers*, 57(8), pp.1098–1112.

Mitra, S. & Ransbotham, S., 2012. The effects of vulnerability disclosure policy on the diffusion of security attacks. In *Thirty Third International Conference on Information Systems, Orlando 2012*. pp. 1–17.

Ransbotham, S., Mitra, S. & Ramsey, J., 2012. Are Markets for Vulnerabilities Effective? *MIS Quarterly*, 36(1), pp.43–64.

Robinson, N., 2016. NATO: changing gear on cyber defence. *NATO*. Available at: http://www.nato.int/docu/review/2016/Also-in-2016/cyber-defense-nato-security-role/EN/ [Accessed June 15, 2016].

Rogers, M.K., 2006. A two-dimensional circumplex approach to the development of a hacker taxonomy. *Digital investigation*, 3(2), pp.97–102.

Simmonds, A., Sandilands, P. & van Ekert, L., 2004. An ontology for network security attacks. *Lecture notes in computer science*, pp.317–323. Available at: http://www.springerlink.com/index/1LQ24LD9UDHX9G8Q.pdf.

Sommestad, T., 2015. Experimentation on operational cyber security in CRATE. In *NATO STO-MP-IST-133 Specialist Meeting*. Copenhagen, Denmark, pp. 7.1–7.12.

Sommestad, T. & Sandström, F., 2015. An empirical test of the accuracy of an attack graph analysis tool. *Information and Computer Security*, 23(5), pp.516–531.