2014 IEEE 15th International Symposium on High-Assurance Systems Engineering

Using Attack Surface Entry Points and Reachability Analysis to Assess the Risk of Software Vulnerability Exploitability

[1]Awad A. Younis, [1]Yashwant K. Malaiya, and [1]Indrajit Ray
[1]Computer Science Department, Colorado State University, Fort Collins, CO 80523, USA
{younis,malaiya,indrajit}@cs.colostate.edu

***Abstract*— An unpatched vulnerability can lead to security breaches. When a new vulnerability is discovered, it needs to be assessed so that it can be prioritized. A major challenge in software security is the assessment of the potential risk due to vulnerability exploitability. CVSS metrics have become a de facto standard that is commonly used to assess the severity of a vulnerability. The CVSS Base Score measures severity based on exploitability and impact measures. CVSS exploitability is measured based on three metrics: Access Vector, Authentication, and Access Complexity. However, CVSS exploitability measures assign subjective numbers based on the views of experts. Two of its factors, Access Vector and Authentication, are the same for almost all vulnerabilities. CVSS does not specify how the third factor, Access Complexity, is measured, and hence we do not know if it considers software properties as a factor. In this paper, we propose an approach that assesses the risk of vulnerability exploitability based on two software properties – attack surface entry points and reachability analysis. A vulnerability is reachable if it is located in one of the entry points or is located in a function that is called either directly or indirectly by the entry points. The likelihood of an entry point being used in an attack can be assessed by using damage potential-effort ratio in the attack surface metric and the presence of system calls deemed dangerous. To illustrate the proposed method, five reported vulnerabilities of Apache HTTP server 1.3.0 have been examined at the source code level. The results show that the proposed approach, which uses more detailed information, can yield a risk assessment that can be different from the CVSS Base Score.**

Keywords—Risk assessment; Measurement, Software Vulnerability; Software Security Metrics; Attack Surface; CVSS Metrics.

1. INTRODUCTION

The security of computer systems and networks depends on the security of the software running on them. Vulnerabilities are security related defects that might be exploited by a malicious user causing loss or harm. In spite of recent advances in vulnerability avoidance (e.g., formal and informal design methods, software development process control), vulnerability identification and removal (e.g., testing, model checking), and intrusion prevention (e.g., firewall, anti-virus software), it is unlikely that completely vulnerability free systems will become possible anytime soon [1]. Therefore, evaluating the risk associated with software vulnerabilities is needed to assess and allocate the resources needed to address them.

A security metric is a quantifiable measurement that indicates the level of security for an attribute of the system [2]. Security metrics give a way to prioritize threats and vulnerabilities by considering the risks they pose to information assets based on quantitative or qualitative measures. The metrics proposed include: vulnerability density, attack surface, flaw severity and severity-to-complexity, security scoring vector for web applications, the Common Vulnerability Scoring System (CVSS) metrics etc. [3]. Each of them is based on specific perspective and assumptions and measures different attributes of software security. They are intended to objectively help decision makers in resource allocation, program planning, risk assessment, and product and service selection.

Problem Description. Assessing the risk associated with software vulnerabilities is accomplished by assessing their severity. CVSS metrics are the de facto standard that is currently used to measure the severity of vulnerabilities. CVSS Base Score measures severity based on exploitability (the ease of exploiting vulnerability) and impact (the effect of exploitation). Exploitability is assessed based on three metrics: Access Vector, Authentication, and Access Complexity. However, CVSS exploitability measures have come under some criticism. First, they assign static subjective numbers to the metrics based on expert knowledge regardless of the type of vulnerability, and they do not correlate with the existence of known exploit [4]. Second, two of its factors (Access Vector and Authentication) have the same value for almost all vulnerabilities [5]. Third, there is no formal procedure for evaluating the third factor (Access Complexity) [6]. Consequently, it is unclear if CVSS considers the software structure and properties as a factor. Thus, there is a need for an approach that can take into account detailed information about the vulnerabilities for a less subjective risk measure.

Contribution. The objective of this research is to propose an approach that can help in assessing the severity of a vulnerability by considering the detailed software structure. In this paper, the concept of *structural severity* is introduced. A vulnerability has to be reachable in order to be exploitable. Our approach evaluates vulnerability exploitability based on software properties. The evaluation is based on the presence of a function call connecting attack surface entry points to the vulnerability location within the software under consideration. If such a call exists, we estimate how likely the entry point is going to be used in an attack based on damage potential-effort ratio [7] in the attack surface metric and dangerous system calls [8]. The damage potential-effort ratio assesses how an attacker might choose an entry point based on benefit (privilege) and cost (effort) that are needed to invoke the targeted method. The dangerous system calls paradigm has been considered as these system calls allow attackers to escalate a method privilege and hence cause more damage. To determine the effectiveness of the proposed approach, the Apache HTTP server was selected as a case study. Apache has been chosen in particular because web servers form a major component of the Internet and Apache has the highest market share among the HTTP servers [9]. Besides, its source code availability allows evaluation of its attack surface and its richness of known vulnerability dataset allows investigation of CVSS exploitability sub-scores.

The paper is organized as follows. Section 2 presents the related work. In Section 3, the background of the attack surface metric, CVVS metrics, the Apache HTTP server, and Exploit database are discussed. In the following section, the key steps of our approach are introduced. In sections 5, the Apache's exploitability measures are examined. In section 6, the exploitability of five vulnerabilities is assessed using the

978-1-4799-3466-9/14 $31.00 © 2014 IEEE
DOI 10.1109/HASE.2014.10

proposed approach. Section 7 presents the observations and results. Finally, concluding comments are given along with the issues that need further research.

2. Related Work

The Attack surface metric has been proposed to quantify the opportunity that an attacker has to compromise the security of a software system. The attack surface notion was first introduced by Howard in his Relative Attack Surface Quotient metric [10]. It was later formally defined by Manadhdata and Wing in [7]. They proposed a framework that included the notion of Entry and Exit Points and the associated damage potential-effort ratio. They have applied their formally defined metric to many systems and the results show the applicability of the notion of attack surface. Their new metric has been adapted by a few major software companies, such as Microsoft, Hewlett-Packard, and SAP. Manadhdata et al in [11] relate the number of reported vulnerabilities for two FTP daemons with the attack surface metric along the method dimension. Younis and Malaiya [12] have compared vulnerability density of two versions of Apache HTTP server with the attack surface metric along the method dimension. Neither [10] nor [11], however, related entry points with the location of the vulnerability to measure its exploitability.

Brenneman [13] has introduced the idea of linking the attack surface entry point to the attack target to prioritize the effort and resource required for software security analysis. Their approach is based on path-based analysis, which can be utilized to generate an attack map. This helps visualizing the attack surfaces, attack target, and functions that link them. This is believed to make significant improvement to software security analysis. In contrast to their work, we not only utilize the idea of linking attack surface entry point with the reported vulnerability location to estimate vulnerability exploitability, but also apply the damage potential-effort ratio in the attack surface metric and checked for the dangerous system calls inside every related entry point to estimate how likely the entry point is going to be used in an attack. This is helpful for inferring attacker's motive in invoking the entry point method.

Bozorgi et al. [4] aimed at measuring vulnerabilities severity based on likelihood of exploitability. They argued that the exploitability measures in CVSS Base Score metric cannot tell much about the vulnerability severity. They attributed that to the fact that CVSS metrics rely on expert knowledge and static formula. To that end, the authors proposed a Machine Learning and Data mining technique that can predict the possibility of vulnerability exploitability. They observed that many vulnerabilities have been found to have high severity score using CVSS exploitability metric although there were no known exploits existing for them. This indicates that CVSS score does not differentiate between exploited and non-exploited vulnerabilities. This result has also been confirmed by Allodi et al. [5], [14], [15]. However, unlike their work, ours relies on attack surface metric, source code analysis, and the reported vulnerabilities location to estimate vulnerability exploitability.

Joh and Malaiya in [16] formally defined a risk measure as a likelihood of adverse event and the impact of this event. In one hand, they utilized the vulnerability lifecycle and applied Markov stochastic model to measure the likelihood of vulnerability exploitability. On the other hand, they used the impact related metrics from CVSS to estimate the exploitability impact. They applied their metric to assess the risk of two systems that had known unpatched vulnerabilities using actual data. In contrast, we assess vulnerability exploitability based on vulnerability reachability regardless of the availability or unavailability of a patch.

Sparks et al in [17] extended the black box fuzzing using a genetic algorithm that use the past branch profiling information to direct the input generation in order to cover specified program regions or points in the control flow graph. The control flow is modeled as Markov process and fitness function is defined over Markov probabilities which are associated with state transition on control flow graph. They generated inputs using grammatical evolution. These inputs are capable of reaching deeply vulnerable code which is hidden in a hard to reach locations. In contrast to their work, ours relies on source code analysis, a link between vulnerability location and attack surface entry points, and dangerous system call analysis that were specifically intended for measuring vulnerability exploitability.

E.Gabrielli and L.Mancini in [8] have presented a detailed analysis of the UNIX system calls and classify them according to their level of threat with respect to system penetration. To control these system calls invocation, they proposed Reference Monitor for UNIX System (REMUS) mechanism to detect intrusion that may use these system calls which could subvert the execution of privileged applications. Nevertheless, our work applies their idea to deduce the motive of an attacker in using an entry point, as attackers usually looks to cause more damage to targeted systems. Thus, our work is not about intrusion detection but rather measuring the exploitability of a known vulnerability.

3. Background

3.1 Attack Surface Metric

A system's attack surface is the subset of the system's resources that are used by an attacker to attack the system [7]. The resources are referred to as methods (e.g., API), channels (e.g., sockets), and data items (e.g., input strings). This means that more number of available resources indicate larger attack surface and hence the system is less secure. Notably, only some of these resources are considered as part of the attack surface. To the relevant resources to be identified, the entry point and exit point framework is used. Besides, the resource contribution is estimated using damage potential-effort ratio. In this paper, the entry point along the method dimension has been chosen. This is due to the fact that most software vulnerabilities exist in a method(s). Besides, in order to exploit a vulnerability in a method an attacker needs to invoke that method either directly or indirectly.

3.2 Software Vulnerability & CVSS Metrics

Software vulnerability is defined as a defect in software systems that presents considerable security risk [18]. A subset of the security related defects, vulnerabilities, are to be discovered and become known eventually [18]. The finders of the vulnerabilities disclose them to the public using some of the common reporting mechanisms available in the field. The databases for the vulnerabilities are maintained by several organizations such as National Vulnerability Database (NVD), Open Source Vulnerability Database (OSVDB), BugTraq, CVE database, etc., as well as the vendors of the software. Vulnerabilities are assigned a unique identifier using MITRE Common Vulnerability and Exposure (CVE) service.

Common Vulnerability Scoring System (CVSS) is the standard measure for vulnerability risk. The CVSS score system provides vendor independent framework for communicating the characteristics and impacts of known vulnerabilities [6]. It is used to evaluate the degree of risks posed by vulnerabilities so mitigation efforts can be prioritized. CVSS defines three metric groups that can be used to characterize vulnerabilities: Base Score, Temporal and Environmental. The Base Score metrics represent the intrinsic characteristics of vulnerability, and are the only mandatory metrics. The optional environmental and temporal metrics are used to augment the Base Score metrics and depend on the target system and changing circumstances. CVSS score from 0.0 to 3.9 corresponds to low severity, 4.0 to 6.9 to medium severity and 7.0 to 10.0 to high severity.

The Base Score metrics include two sub-groups, exploitability and impact metrics. Exploitability observed as a metric for describing the ease of exploiting vulnerability. It is measured based on three factors: Access Vector (AV), Authentication, (AU), and Access (attack) Complexity (AC) [6] :

$$\textbf{Exploitability} = 20 * AV * AC * AU$$

Access complexity sub-score is assigned as low, medium, and high. Low complexity means one that involves no specialized conditions such as default configuration or the attack can be implemented with not much skills. Medium complexity means that access conditions are somewhat specialized such as involving no default configuration or require specific system knowledge. High complexity requires specialized access conditions such as elevated privileges.

3.3 Apache HTTP Server

Apache HTTP server is a Web server that is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation. Apache HTTP server is simply a piece of software that responds to requests for information sent by web browsers [19]. It has gone through a number of improvements after its initial launch, which led to the release of several versions: 1.3.x, 2.0.x, 2.2.x, 2.3.x, and 2.4.x. According to [9], Apache web server has over 64% market share of the top web servers on the Internet.

3.4 The Exploit Database (EDB)

EDB is an essential collection of exploits and vulnerable software [20]. It is used by penetration testers, vulnerability researchers, and security fanatics. It reports vulnerability for which a there is a proof-of-concept exploit. EDB is considered as the white market for exploits. EDB contains around 24075 exploits as the time of writing this paper. Most of its data are derived from Metasploit Framework, a tool for creating and executing exploit code against a distant target machine. It provides a search using vulnerability CVE number for variety of vulnerabilities types and software.

4. APPROACH

Our approach in assessing software vulnerability exploitability is based on the following steps:

- Define the entry points, methods that contained a call to a function in *input* functions, of the chosen system.
- Use function call analysis to find a connection (path) between the entry point and the vulnerability location.
- Estimate the likelihood of an entry point being used in an attack using:
  - Damage potential-effort ratio along the method dimension in the attack surface metric as given in (1).
  - Dangerous system calls in the entry points and their dangerous level as in (2).
- Assess an individual vulnerability exploitability based on I, II, and III and then assign its structural severity value.

4.1 Entry Point and Exit Point Framework

The entry point and exit point framework is a formal framework that defines the set of entry points and exit points (methods), the set of channels, and the set of untrusted data items from the source code of a system [7]. Entry and Exit points are the methods that an attacker uses to either send or receive data from the system. In this paper, we will use only the entry points as they are the main target by malicious attacks. A method can be either a direct or an indirect entry point [7]. In one hand, a method is a direct entry point if it receives data directly from the environment; read method defined in unistd.h in C library is an example [11]. Besides, a method is an indirect entry point if it receives data from direct entry point.

4.2 Damage Potential-Effort Ratio

Damage potential and access effort ratio is an informal means that are used to estimate damage potential-effort in terms of resources attributes [7]. The damage depends on the method's privilege, the channel's type, and the data item's type, whereas, the effort depends on the rights of the resource that the attacker needs to acquire to use a resource in an attack. The likelihood of a method being used in an attack is given in (1) [18]:

$$\textbf{ac(method)= privilege/access right} \qquad (1)$$

where ac() is the attackability. The user of this metric is responsible for assigning a numeric values for privilege levels, types of the channel, and types of data items [9]. However, the following should be taken in considerations: the higher the privilege, the higher the damage, whereas the higher the access right the higher the effort [9].

As it can be seen in Table 1, a value to each privilege level and access right level is assigned based on our knowledge of Apache HTTP server and Linux (Ubuntu). The privilege in (1) is used as an indicator of the exploitability impact (damage), while the access right is an indicator of exploitability difficulty (attacker effort).

Table 1: Numeric values of Privilege & Access Rights

Method Privilege	Value	Access Rights	Value
root	5	root	5
apache or (www-data, manager.sys, or nobody)	3	apache or (www-data, manager.sys, or nobody)	3
authenticated	3	authenticated	3
unauthenticated	1	unauthenticated	1
		Anonymous	1

4.3 Dangerous System Calls

System calls are the entry points to privileged kernel operations [21]. They are the essential interface between an application program and the operating system kernel. Operating systems contain groups of calls for performing various low-level operations. Hence, if we want to execute an operating system call from a program, we need to make a system call. Calling a system call from a method (a user function not library function) in program can violets the least privilege principle. This helps an attacker to directly invoke a system call inside a vulnerable method or indirectly invoking a system call within the scope of that vulnerable function. Thus, this will further help the attackers escalate their privilege and hence causing more damage to the compromised system.

E.Gabrielli and L.Mancini in [8] define dangerous system calls as specific system calls that can be used to take complete control of the system, cause a denial of service, or other malicious acts. These system calls (UNIX calls) have been identified and classified into four levels of threat. Level one allows full control of the system while level two used for denial of service attack. On the other hand, level three used for disrupting the invoking process and level four is considered harmless. In this paper, the focus will be mainly in threat level one and two. Choosing system calls of threat level one is due to the fact that taking full control of the system can cause more harm to the system. While particularly selecting level two is because that our case study is Apache where denial of service vulnerabilities represents 30% of total number of vulnerabilities across all versions and releases. Table 2 shows the dangerous system calls of level threat one and two as classified by [8]. There are 22 system calls of threat level one and 32 of threat level two.

Table 2: Dangerous System Calls

Threat Level	Dangerous System Calls
1. Full control of the system	chmod, fchmod, chown, fchown, lchown, execve, mount, rename, open, link, symlink, unlink, setgroups, setgid, setfsgid, setresgid, setregid, creat_module, setresuid, setreuid, setuid, setfsuid
2. Denial of service	umount, mkdir, rmdir, umount2, ioctl, nfsservctl, truncate, ftruncate, quotactl, dup, dup2, flock, fork, kill, iopl, reboot, ioperm, clone, modify_ldt, adjtimex, vhangup, vm86, delete_module, stime, settimeoday, socketcall, sethostname, syslog, setdomainname,_sysctl,exit, ptrace

However, as dangerous system calls have different threat level, every level has been assigned a weight as shown in Table 3. As a method might include one or more dangerous system calls with different levels, the following equation has been devised to estimate the dangerous level of a system call:

$$\text{Dangerous Level} = \sum_{i=1}^{n} \boldsymbol{DSCi} \times \boldsymbol{TLi} \qquad (2)$$

where n is the number of system calls in a given method, DSC is dangerous system calls, and TL is the threat level weight.

Table 3: Threat Level Weights

Threat Level	Weight
1	1
2	0.6
3	0.3
4	0

4.4 Structural Severity

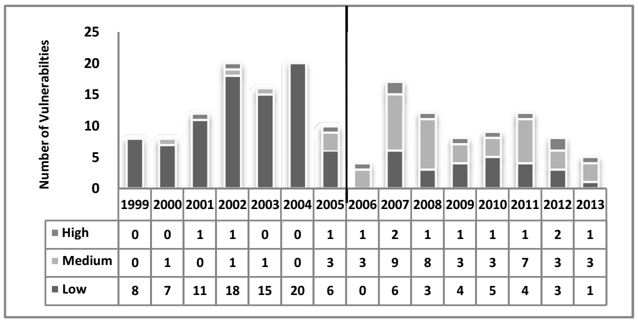
Structural severity is introduced as a measure that uses software attributes to evaluate the risk of an attacker reaching a vulnerability location from attack surface entry points. It is measured based on three values: high, medium, and low. It is high if a vulnerability is reachable from an entry point with dangerous system calls. It is medium if it is reachable from an entry point with no dangerous system calls. It is low if it is not reachable from any entry points.

5. APACHE'S EXPLOITABILITY MEASURES

The vulnerability datasets of Apache HTTP server releases have been obtained from NVD [22] which is maintained by National Institute of Standers and Technology and sponsored by the department of Home Land Security. The collected discovered vulnerabilities were of a period from 1999 to 2013 and they are 169 vulnerabilities. In the following subsections we will investigate the access complexity sub-score trends. Then, we will look at the distribution of the main Apache's vulnerability types and the access complexity sub-scores. Next, we will check the authentication and the access vector.

5.1 Access Complexity

To understand the ease and difficulty of Apache's vulnerabilities exploitability, we have collected data of the Apache's access complexity CVSS sub-score of the period 1999-2013 for all reported vulnerabilities. As it can be seen from Fig. 1, most of the discovered vulnerabilities during the period 1999 to 2005 had a low access complexity (from 90.4% low to 6.4% medium). However, starting from the period 2006 to 2013 the trends have dramatically changed from majority low to half medium access complexity (from 34.6% low to 52% medium). This could be attributed to security improvement that Apache has made to its product code and elimination of its old vulnerabilities. However, it has also been noticed that high access complexity vulnerabilities have increased from 3.2% in the period 1999-2005 to 13.3% in period 2006-2013. This could be attributed to the recent rise in the market share price of high access complexity vulnerabilities and the sophistication of the vulnerability discovers scanners and techniques.

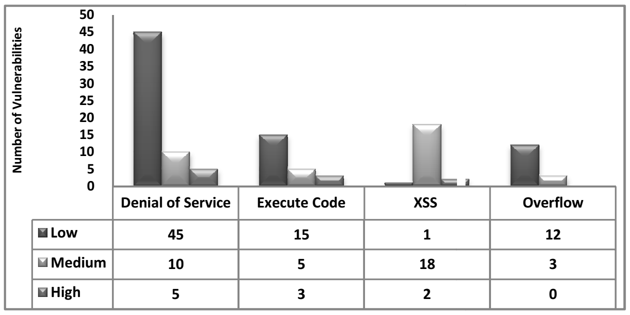


	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
■ High	0	0	1	1	0	0	1	1	2	1	1	1	1	2	1
■ Medium	0	1	0	1	1	0	3	3	9	8	3	3	7	3	3
■ Low	8	7	11	18	15	20	6	0	6	3	4	5	4	3	1

Figure 1: Vulnerabilities by Access Complexity

5.2 Access Complexity across Vulnerability Types

Looking at the type of apache's vulnerabilities, we have noticed that the following vulnerabilities represent around 70% of the overall reported vulnerabilities: denial of service (DoS) 33.5%, execute code 13.6%, cross site scripting, (XSS) 12.4%, overflow 8.8%, and others around 30%. Fig. 2 presents the distribution of access complexity sub-score with respect to the selected vulnerabilities types for the overall reported vulnerabilities. By looking at the vulnerabilities distribution during the two periods 1999-2005 and 2006-2013, we have observed a dramatic decline in the low access complexity sub-score among the four types as shown in Table 4. On the other hand, we have also witnessed an increase in the medium access complexity from the period 1999-2005 to 2006- 2013. However, unlike the other three types of the vulnerabilities, majority of XSS vulnerabilities had a medium access complexity while the overflow vulnerabilities did not have any vulnerability of high access complexity.



	Denial of Service	Execute Code	XSS	Overflow
▨ Low	45	15	1	12
▨ Medium	10	5	18	3
▨ High	5	3	2	0

Figure 2: Access Complexity across Vulnerability Types

5.3 *Authentication and Access Vector*

We have checked the authentication and the access vector of Apache HTTP server from 1999-2013. Our studies show that all vulnerabilities are accessed remotely and only three vulnerabilities, out of the 168, required authentications (1.7%). The three vulnerabilities required a single system authentication.

Table 4: % of Access Complexity across Vulnerability Types

Vulnerability Type	Period	Low %	Medium %	High %
1. Denial of service	1999-2005	96.7	0	3.22
	2006-2013	51.7	34.5	13.8
2. Execute Code	1999-2005	86.7	13.3	0
	2006-2013	25	37.5	37.5
3. XSS	1999-2005	16.7	83.3	0
	2006-2013	86.7	86.7	13.3
4. Overflow	1999-2005	25	8.3	0
	2006-2013	33.3	66.7	0

6. VULNERABILITY EXPLOITABILITY ASSESSMENT

6.1 *System Vulnerabilities*

In [12], Younis and Malaiya recognized that Apache HTTP server has new and inherited vulnerabilities for every version. The former represents the vulnerabilities that have been introduced in a specific version, while the latter represents the vulnerabilities that have been introduced from another versions and/or releases due to concept of reuse. However, Apache 1.3.0 version has *two* new vulnerabilities and *eleven* inherited vulnerabilities. To apply our method, we have chosen the following known vulnerabilities based on the availability of information about their locations and their types: CVE-2011-0419 (DoS), CVE-2012-0031 (DoS), CVE-2010-0010 (Execute code), CVE-2004-0940 (XSS), and CVE-2004-0488 (Overflow).

6.2 *System Attack Surface Entry Points*

In this section, we will identify the attack surface entry points along the method dimension for Apache HTTP Server 1.3.0. However, identifying the attack surface entry points requires looking at the code base and finding all entry points which could be part of the attack surface. By finding such points, what is needed next is classifying each one of them into an attack class. The code bases of the chosen version was obtained from [23].

The entry points along the method dimension have been defined using cflow tool. The tool analyzes a code base written in C programming language and produces a graph charting dependencies between various functions [24]. From the call graph, the methods that contained a call to a function in *input* functions are identified. However, assessing the privilege level and access right level of the entry points along the method dimension is required. Determine each method privilege and access right can be determined by looking at: Privilege: setUID() calls and Access Right: location where the authentication is performed.

We have identified the entry points for the whole system and selected the ones that are related to the chosen vulnerabilities location. We realized that the related entry point methods had an apache privilege and access right and as a result we did not include them in Table 5. Besides, we have checked whether an entry point has a dangerous system call of threat level one and two. These entry points and the dangerous system call are shown in Table 5.

Table 5: Apache 1.3.0 Entry Points and Dangerous System Calls

File	Input Method	Entry Point	Dangerous System Calls
1. http_core.c	gethostbyaddress	ap_get_remote_host()	
	gethostbyname	ap_inline()	
2. http_main.c	getopt	realmain(),	exit
		mastermain(),	
		main()	
	signal	siglist_init(),	
		set_signals(),	
		child_main(),	setgid and setuid dup, exit, and flock
		make_child(),	fork
		standalone_main(),	setuid, open, fork, kill, exit, and unlink
		child_sub_main()	dup
3.http_protocol.c	fread	ap_send_fd_length()	
	getline	read_request_line(),	
		getmime_headers(),	
		ap_get_client_block()	
4.proxy_util.c	gethostbyname, gethostbyaddress	ap_proxy_host2addr()	
	scanf	ap_proxy_hex2c(), ap_proxy_date_canon(), proxy_match_ipaddress()	
5.mod_include.c	getc	GET_CHAR()	
6.ssl_util.c	-	-	

6.3 *Mapping the Entry Points to the Vulnerabilities*

Once the vulnerabilities have been identified and the entry points of the system have been determined, mapping each vulnerability to an entry point can be achieved by first finding the vulnerability location in the source code. Finding a vulnerability location can be determined from vulnerability report or by using static code analyzers when the report does not finalize such information. The static code analyzers are tools that are used to

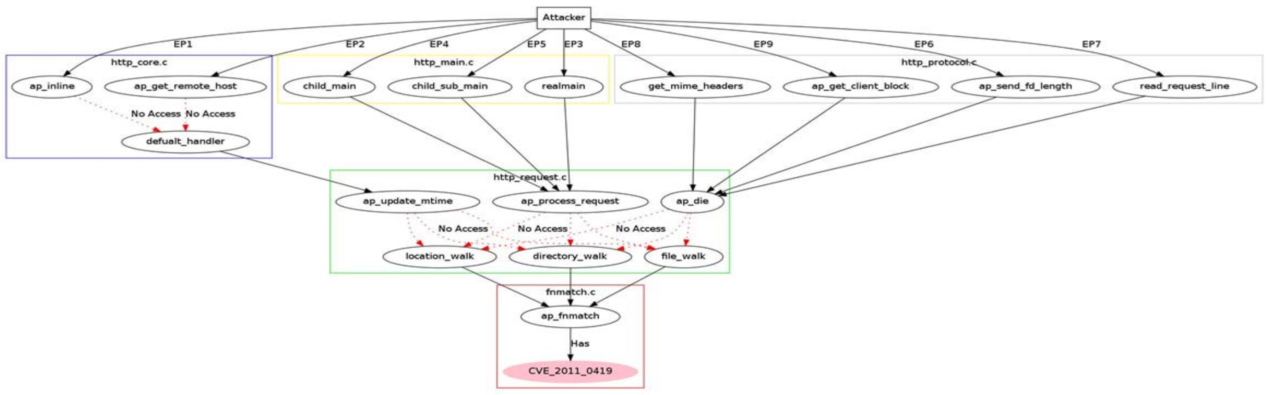


Figure 4: Indirectly Mapping the Entry Points in http_core, http_main, and http_prorotocol to the vulnerable method

find common bugs or vulnerabilities in the code base without the need to execute the code. Splint (Secure Programming Lint) is an example. It is a tool that uses static analysis to detect vulnerabilities in programs [25]. However, in this paper we will use the vulnerability report to find the location of the vulnerability and leave using the static tools as a future work. Then, we use cflow to find whether the vulnerable method is called by the entry point(s) or not. Due to the pages limits, only the analysis of the first and the second vulnerabilities will be presented in the following subsections.

6.3.1 CVE-2011-0419:

Vulnerabilities are either located in one of the entry points or are located in a function that is called by the entry points directly or indirectly. From the following report description, the location of the vulnerabilities CVE-2011-0419 has been determined:

"Stack consumption vulnerability in the *fnmatch* implementation in *ap_fnmatch.c* in the Apache Portable Runtime (APR) library before 1.4.3 and the Apache HTTP Server before 2.2.18, and in fnmatch.c in libc in NetBSD 5.1, OpenBSD 4.8, FreeBSD, Apple Mac OS X 10.6, Oracle Solaris 10, and Android, allows context-dependent attackers to cause a denial of service (CPU and memory consumption) via *? sequences in the first argument, as demonstrated by attacks against mod_autoindex in httpd [26]".

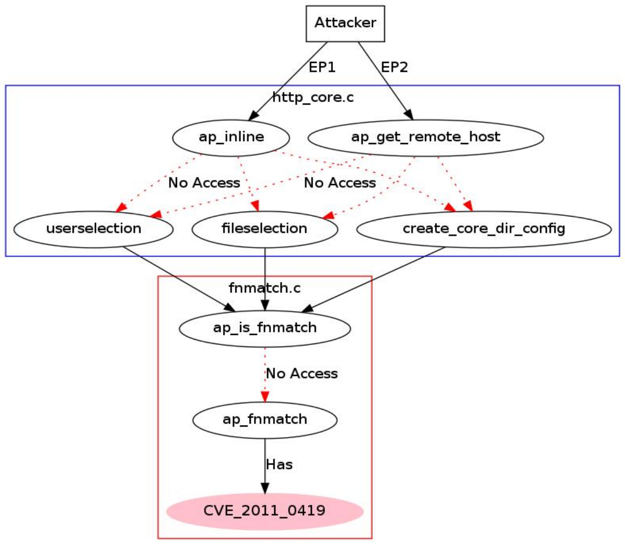


Figure 3: Directly Mapping the EP in http_core.c to the vulnerable method.

As it can be seen, the vulnerability located in *fnmatch.c*. The *fnamtch* has two methods namely *ap_fnmatch()* and *ap_is_fnmatch()* that can be invoked by outsider methods. By analyzing the source code we have found out that the vulnerability located in the *ap_fnmatch()* method. Using the entry points, the attacker can have an access to the vulnerability by two ways:

- **Directly**: the *http_core.c* component has to entry points and is able to call the vulnerable component *fnmatch.c* by using any of its three methods: *userselection(), fileselection(),* and *create_core_dir_config() which in turn call the ap_is_fnmatch()* method in the *fnmatch.c*. As it can be seen from Fig.3, the two entry pints had no access (*no path*) to any of the *http_core* three methods. Besides, *ap_is_fnmatch*() method has no access to the *ap_fnamtch()* method which makes it even harder to the attacker to invoke the *ap_fnamtch()* method using the entry points in *http_core* component. As a result, it could be concluded that there is no call relationship between the *http_core.c* entry point and the vulnerable method.
- **Indirectly**: *http_request.c* does not have any entry points but can be accessed by one of the three components namely: *http_core.c, http_main.c, and http_protocol.*c which have an entry points. *http_request* has three methods: *directory_wallk(),location_walk,* and *file_walk()* which can invoke the vulnerable method ap_fnmatch(). From Fig.4, the following have been observed:
  - ***http_core*** uses its method *default_handelr()* to invoke the *ap_update_mtime()* method in the http_request. We have found out that the two entry points in the *http_core* cannot invoke the method *default_handelr()*. Besides, the a*p_update_mtime()* cannot invoke any of the three *http_request* methods which call the vulnerable method.
  - ***http_main*** has three entry points: child_main(), child_sub_main(), and realmain() that can invoke the *ap_process_request()* method in the *http_request*. However, *ap_process_request()* had no access to any of the three *http_request* methods which in turn call the vulnerable method.
  - ***http_protocol*** has four entry points: *get_mime_headers(), ap_get_client_block(), ap_send_fd_length(),* and *read_request_line()* that are able to invoke the *ap_die()* method in the *http_request*. In spite of this, the *ap_die()* possessed no access to the vulnerable methods.

Based on the indirect access using the entry points of the system to the vulnerable method, it can be concluded that there is no indirect call relationship.

Table 6: Vulnerabilities Exploitability Assessment

Application	Vulnerability	Vulnerable Method	Path from Entry Points	Entry Point Privilege	Entry Point Access Rights	Entry Point's Dangerous System Calls	Reachability
Apache 1.3.0	CVE-2011-0419	ap_fnmatch()	No Path	-	-	-	Not reachable
	CVE-2012-0031	ap_cleanup_scoreboard(), ap_creatscoreboard()	Path	apache	apache	setuid, open, fork, kill, exit, unlink, setgid, dup, and flock	Reachable
	CVE-2010-0010	ap_proxy_send_fb()	Path	-	-	-	Reachable
	CVE-2004-0488	ssl_util_uuencode_binary()	No path	-	-	-	Not reachable
	CVE-2004-0940	get_tag()	Path	apache	apache	-	Reachable

6.3.2 CVE-2012-0031:

"scoreboard.c in the Apache HTTP Server 2.2.21 and earlier might allow local users to cause a denial of service (daemon crash during shutdown) or possibly have unspecified other impact by modifying a certain type field within a scoreboard shared memory segment, leading to an invalid call to the free function. Scoreboard issue could allow an unprivileged child process to cause the parent to crash at shutdown rather than terminate cleanly".

To determine the location of the vulnerability in the *scoreboard.c*, we looked at the patch report and we have found out that the vulnerable code is in the methods *ap_cleanup_scoreboard* and *ap_creat_scoreboard.* As it can be seen in Fig.5, in one hand the entry points in the *http_main.c* have a direct access to the vulnerable methods by either passing the parameter *ap_scoreboard_immage* or *ap_scoreboard_fname*. On the other hand, the entry points in the *http_core.c* have no access to the vulnerable methods. As a result, it could be concluded that there is a call relationship between *http_main.c* entry points and the vulnerable code.

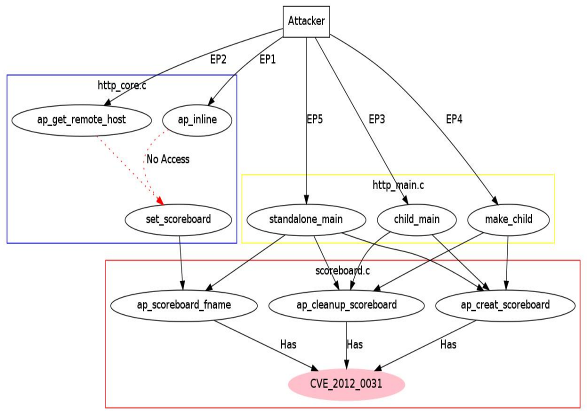


Figure 5: Directly Mapping the Entry Points in http_core and http_main to the vulnerable method

6.4 Vulnerability Exploitability Estimation

After mapping the entry points to vulnerability location, determining the privilege and the access right to the entry points, and identifying the dangerous system calls, estimating the individual vulnerability structural severity can be achieved based on our results in Table 6. Looking at Table 6, a vulnerability is either:

1. Reachable with Dangerous System Calls,
2. Reachable with No Dangerous System Calls,
3. Not reachable.

In our case study, the privilege and the access right of the methods were all apache. Hence the ratio in (1) was one for all methods. In the following subsections, the chosen five vulnerabilities will be assessed as follows.

6.4.1 CVE-2011-0419:

According to the CVSS metric, this vulnerability had a medium severity, medium access complexity sub score, and is of a type DoS. Based on software structure analysis, we did not find any call relationship between the vulnerable function and the entry point functions. It should be also noted that the values of the entry point privilege, access right, and dangerous system calls have been left blank as a result of having no call relationship between the entry points and the venerable function. Additionally, no exploit has been found for this vulnerability in [20], which makes the CVSS score a suspect. Thus, based on reachability, it can be concluded that this vulnerability is not reachable.

6.4.2 CVE-2012-0031:

Based on the CVSS metric scores, this vulnerability had medium severity, low access complexity, and is of type DoS. The analysis show that multiple call relationships with the vulnerable method and the entry point functions existed. However, although the three functions have an apache privilege and access right, our analysis has shown that two of the three entry points (child_main() and standalone_main()) contain some dangerous system calls. The vulnerable method had nine system dangerous calls. Four are of threat level 1 and five are of threat level 2. Using (2) and the weights values in table 6, this vulnerability has been found to be reachable with dangerous level 7.

6.4.3 CVE-2010-0010:

This vulnerability had medium severity, medium access complexity, and it is of the type executing code. Based on our analysis, we have found that it had indirect function calls form the entry points. Besides, no system calls had been found. We also looked for an exploit for this vulnerability in Exploit database and we did not find one. However, in [27] a proof of concept of an existence of an exploit has been provided. Therefore, it can be concluded that this vulnerability is reachable.

6.4.4 CVE-2004-0488:

This vulnerability had high severity, low access complexity, and it is of the type overflow. Based on our analysis, we have found that it had no direct or indirect entry points and no system dangerous calls. We have also looked at the Exploit database and no exploit was found. Thus, this vulnerability is not reachable.

6.4.5 CVE-2004-0940:

Based on the CVSS metric scores, this vulnerability had medium severity, a low access complexity, and it is of a type XSS. We have found out that a call relationship with the vulnerable method and the entry point functions exist. However, no dangerous system calls have been found. As a result, this vulnerability is reachable with no dangerous system calls.

7. OBSERVATIONS AND RESULTS

Entry points and reachability analysis are good indicators of vulnerability exploitability. Structural severity is measured using three values: High, medium, and low as described in section 3.6. Based on these three values, the chosen vulnerabilities have been assessed and compared to CVSS Access Complexity as shown in Table7.

Table 7: The Obtained Measures Compared to CVSS Access Complexity Metric

Vulnerability	Reachability	Structural Severity	CVSS (AC)
1. CVE-2011-0419	Not reachable	Low	Medium
2. CVE-2012-0031	Reachable with Dangerous System Calls	High	Low
3. CVE-2010-0010	Reachable with No Dangerous System Calls	Medium	Medium
4. CVE-2004-0488	Not reachable	Low	Low
5. CVE-2004-0940	Reachable with No Dangerous System Calls	Medium	Low

For instance, the vulnerable method (CVE-2012-0031: ap_cleanup_scoreboard()) is reachable by a method that has dangerous system calls. Thus, the structural severity of this vulnerability has been considered High, whereas (CVE-2004-0940: get_tag()) which is reachable without dangerous system calls has structural severity as Medium. On the other hand, (CVE-2011-0419: ap_fnmatch()) is not reachable and hence it has structural severity Low. However, in the case when two vulnerabilities are both reachable with dangerous system calls, the dangerous level value or the damage potential-effort ratio can break the tie.

In Table 7, vulnerabilities one and three have been assigned medium access complexity sub-score using CVSS metric whereas in fact they are unreachable based on software structure analysis. Considering network accessibility factor is useful but not sufficient. Hence, software accessibility attributes should be also taking into consideration when evaluating vulnerability exploitability. Assessing vulnerabilities exploitability based on source code analysis provide valuable information. Beside measuring vulnerability exploitability, it also help us better knowing our software and make it more secure by securing paths that are likely to be used by attackers.

8. CONCLUSION AND FUTURE WORK

Assessing the severity of a vulnerability requires evaluating the potential risk. Existing measures rely on subjective judgment. In this paper, we have proposed an approach that uses system related attributes such as attack surface entry points, vulnerability location, call function analysis, and the existence of dangerous system calls. This approach requires us to explore some of the major software security issues such as the paths to the vulnerable code starting from the entry points. We have demonstrated our approach and have compared resulting measures with CVSS access complexity metrics. Our preliminary results showed that this approach is encouraging because it allows assessment of the system security based on systematic evaluation and not subjective judgment.

While five vulnerabilities were considered as examples in this study, future studies for more vulnerability with variety of types and for different software systems should be performed to establish applicability of the proposed approach. We have noticed that the location of most of the vulnerabilities has not been given even when their severity is high. Hence, coming up with a technique for determining the vulnerability location is essential. It will be useful if the location identification can be supported by a tool. While the main parts of analysis have been automated, providing a framework that can automate the entire analysis will be helpful in reducing the analysis overall effort. Even though measuring the possibility of reaching a vulnerability is important, quantifying the degree of difficulty of reaching a vulnerability is also valuable for comparing the severity among similar vulnerabilities, and thus needs to be examined. Finally, devising a way of estimating the impact of the reachable vulnerabilities will be valuable for estimating the overall risk of individual vulnerabilities and the whole system, in addition to what CVSS metrics currently offer.

REFERENCES

[1] S. Farrell, "Why didn't we spot that? [Practical Security]," *Internet Computing, IEEE*, vol. 14, no. 1, pp. 84 –87, Feb. 2010.

[2] W. Jansen, *Directions in security metrics research*. NIST, NISTIR 7564, p. 21, 2009.

[3] K. M. Goertzel, T. Winograd, H. L. McKinley, L. J. Oh, M. Colon, T. McGibbon, E. Fedchak, and R. Vienneau, "Software Security Assurance: A State-of-Art Report (SAR)," DTIC Document, 2007.

[4] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2010, pp. 105–114.

[5] L. Allodi and F. Massacci, "A preliminary analysis of vulnerability scores for attacks in wild," *ACM Proc. of CCS BADGERS*, vol. 12, 2012.

[6] P. Mell, K. Scarfone, and S. Romanosky, "A complete guide to the common vulnerability scoring system version 2.0," in *Published by FIRST-Forum of Incident Response and Security Teams*, 2007, pp. 1–23.

[7] P. K. Manadhata and J. M. Wing, "An Attack Surface Metric," *Software Engineering, IEEE Transactions on*, vol. 37, no. 3, pp. 371 –386, Jun. 2011.

[8] M. Bernaschi, E. Gabrielli, and L. V. Mancini, "REMUS: A security-enhanced operating system," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 1, pp. 36–61, 2002.

[9] "Usage Statistics and Market Share of Web Servers for Websites, August 2013." [Online]. Available: http://w3techs.com/technologies/overview/web_server/all. [Accessed: 02-Aug-2013].

[10] M. Howard, J. Pincus, and J. Wing, "Measuring Relative Attack Surfaces," in *Computer Security in the 21st Century*, D. T. Lee, S. P. Shieh, and J. D. Tygar, Eds. Springer US, 2005, pp. 109–137.

[11] P. Manadhata, J. Wing, M. Flynn, and M. McQueen, "Measuring the attack surfaces of two FTP daemons," in *In Proceedings of the 2nd ACM workshop on Quality of protection, 2006.*

[12] A. A. Younis and Y. K. Malaiya, "Relationship between Attack Surface and Vulnerability Density: A Case Study on Apache HTTP Server," in *in ICOMP, The 2012 International Conference on Internet Computing, 2012.*

[13] D. Brenneman, "Improving Software Security by Identifying and Securing Paths Linking Attack Surface to Attack Target," McCabe Software Inc., White Paper, Apr. 2012.

[14] L. Allodi and F. Massacci, "My Software has a Vulnerability, should I worry?," *arXiv preprint arXiv:1301.1275*, 2013.

[15] L. Allodi, W. Shim, and F. Massacci, "Quantitative assessment of risk reduction with cybercrime black market monitoring.," *2013 IEEE Security and Privacy Workshops*, 2013.

[16] H. Joh and Y. K. Malaiya, "Defining and assessing quantitative security risk measures using vulnerability lifecycle and cvss metrics," in *The 2011 International Conference on Security and Management (sam)*, 2011, pp. 10–16.

[17] S. Sparks, S. Embleton, R. Cunningham, and C. Zou, "Automated vulnerability analysis: Leveraging control flow for evolutionary input crafting," in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, 2007, pp. 477–486.

[18] C. P. Pfleeger and S. L. Pfleeger, *Security in computing*. Prentice Hall PTR, 2006.

[19] R. B. Bloom and B. Foreword By-Behlendorf, *Apache Server 2.0: The Complete Reference*. Osborne/McGraw-Hill, 2002.

[20] "Exploits Database by Offensive Security." [Online]. Available: http://www.exploit-db.com/. [Accessed: 07-Aug-2013].

[21] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating system concepts*. J. Wiley & Sons, 2009.

[22] "National Vulnerability Database Home." [Online]. Available: http://nvd.nist.gov/. [Accessed: 09-Aug-2013].

[23] "archive.apache.org." [Online]. Available: http://archive.apache.org/dist/httpd/. [Accessed: 02-Aug-2013].

[24] "GNU cflow." [Online]. Available: http://www.gnu.org/software/cflow/manual/cflow.html. [Accessed: 02-Aug-2013].

[25] D. Evans and D. Larochelle, "Improving security using extensible lightweight static analysis," *software, IEEE*, vol. 19, no. 1, pp. 42–51, 2002.

[26] "CVE security vulnerability database. Security vulnerabilities, exploits, references and more." [Online]. Available: http://www.cvedetails.com/. [Accessed: 02-Aug-2013].

[27] "Files ≈ Packet Storm." [Online]. Available: http://packetstormsecurity.com/1001-exploits/modproxy-overflow.txt. [Accessed: 14-Jul-2013].