Relationship between Attack Surface and Vulnerability Density: A Case Study on Apache HTTP Server

[1]Awad A. Younis and [1]Yashwant K. Malaiya
[1]Computer Science Department, Colorado State University, Fort Collins, CO 80523, USA

Abstract – Software Security metrics are quantitative measures related to a software system's level of trustworthiness. They can be used to aid in resource allocation, program planning, risk assessment, and product and service selection. Recently researchers have proposed several software security metrics. Among these are attack surface and vulnerability density. The attack surface measure has been used by a few major software companies, such as Microsoft, Hewlett-Packard, and SAP. The vulnerability density measure has been applied by some researchers to Windows and Linux family of operating systems, in addition to some web servers and browsers. Despite their promise, establishing the validity of software security metrics remains a key challenge. A single security metric may be unable to measure all aspects of security and hence the use of multiple metrics may be needed in some situations. To assess the applicability of the metrics quantifying individual as well as multiple aspects of security, we explore the relationship between the attack surface and vulnerability density metrics. For this examination, the source code and vulnerabilities data of two releases of Apache HTTP Server have been examined. While the results show that the attack surface and vulnerability density are related, further investigations are needed to develop methods that combine them.

Keywords – Software security metrics; Attack surface; Software Vulnerabilities; Quantitative security; Web servers;

1 Introduction

Software has become a critical part of practically every part of the human life, from personal use, private and public organizations, to industry. It is also well known that all major software systems have defects. Some of these defects are security related, and are termed vulnerabilities. If they are exploited by malicious users; they can lead to considerable damage. The Code Red worm [5], a computer worm that exploited security related defects in Microsoft's HTTP Server IIS (Internet Information Services) in 2001, illustrates the kind damage that can occur due to presence of vulnerabilities. Security of software has become a main concern for both vendors and users, and thus there is a need to develop methods that can lead to more secure software.

The measurement to a specific security aspect of a system is represented by a metric. A security metric is a quantifiable measurement that indicates the level of security for an attribute of the system [6]. Recently several software security metrics have been proposed and explored by researchers in academia, industry, and government. These include: relative vulnerability, static analysis tool effectiveness, attack surface, vulnerability density, flaw severity and severity-to-complexity, security scoring vector for web applications, CVSS scores etc. [7]. Each of them based on its specific perspective, target, and assumptions and measures different attributes of software security. They are intended to help decision makers in resource allocation, program planning, risk assessment, and product and service selection. They attempt to ensure that decisions are not only based on subjective perceptions.

While quantifiable security metrics, as in the case of physical sciences, are supposed to provide precise knowledge to decision-makers, software does not obey laws as exactly as in physics and thus this field has its own challenges [5]. As security is a software attribute, its measurement is not a trivial task [2]. According to Verendel [8], the lack of validation and comparisons among such metrics is a main challenge, which in turn makes their usability risky. Moreover, there is no single security metric yet that can incorporate all aspects of security, and hence having a framework that combines different metrics is an important step toward measuring the system security [8].

The main objective of this research is to understand the nature of those challenges by studying the relationship between two major security metrics namely: attack surface and vulnerability density. Our choice of these two metrics is based on the following factors. First, the attack surface metric is already in use by Microsoft and has been applied by a few major software companies, such as Hewlett-Packard (HP) and SAP. The latter is selected due to the fact that vulnerability density, as a normalized metric, allows subsequent releases of varying size to be compared [4]. Thus, this metric satisfies our objective of comparing the number of vulnerabilities of two different releases of a specific system, Apache HTTP server. Apache has been chosen in particular because the web servers form a major component of the internet, and Apache has the highest market share among the HTTP servers [5]. Its source code availability allows evaluation of its attack surface.

P. Manadhata et al. in [3] have investigated the validity of one of the components of the attack surface metric by relating the number of vulnerability reports with the attack surface for two FTP Daemons, ProfFTPD and Wu-FTPD, by examining the methods involved. They have found that the attack surface of Wu-FTPD and ProfFTPD is related to their reported number of vulnerabilities. In their investigation, they applied the attack surface metric to compare the security of two different systems with the same functionality and then related it to the reported number of vulnerabilities. In this study we apply the attack surface metric to a much larger software. Also in our study we compare the security of two different versions of the same system and then relate it to their vulnerability density values.

Vulnerability density metric was initially introduced and applied to major operating systems, namely Microsoft

Windows and Red Hat Linux [4]. S. Woo et al in [5] have compared the vulnerability density of two HTTP web servers: Apache and IIS. In both studies the vulnerability density metric has been found meaningful. However, vulnerability density metric has been applied to initial versions of each software, and not to separate releases of the same system. Moreover, to the limit of our knowledge it has not been compared with another security metric. In this study, we compare vulnerability density metric with the attack surface metric along the method dimension. In addition, we discuss how these two metrics can complement each other.

The rest of the paper is organized as follows. Section 2 presents a brief overview of the attack surface metric. In Section 3, the software vulnerabilities are discussed and the vulnerability density metric is introduced. In the next section, the key aspects of Apache HTTP server are introduced. In sections 5 and 6, the vulnerability density and the attack surface measurements of the two Apache releases are examined. Section 7 presents the results of the comparison and the observations on the two metrics. Finally, the concluding comments are given along with the issues that need further research.

2 Attack Surface Metric

A system's attack surface depends on the subset of the system's resources that can be used by an adversary to attack the system [2]. The resources are referred to as methods (e.g., API), channels (e.g., sockets), and data items (e.g., input strings). A system that has more of such resources has a larger attack surface and hence the system is less secure [1]. Notably, only some of these resources are considered as part of the attack surface. Their contribution to the attack surface measurement is weighted appropriately. The main resources are the entry points and exit points [2] and the relative contribution is estimated using damage potential and effort ratio.

2.1 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 [2]. Entry and Exit points are the methods that an attacker uses to either send or receive data from the system [2]. Channels are the means that used by an attacker to connect to the system [2]. Untrusted data items are the data that the attacker can either send or receive from the system [2].

A method can be either a direct or indirect entry point and/or a direct or indirect exit point [2]. 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 [3]. Besides, a method is indirect entry point if it receives data from direct entry point [2]. On the other hand, a method is a direct exit point if it sends data directly to the environment's system [2]; fprintf method defined in C library is an example [3]. Moreover, a method is indirect exit point if it sends data to direct exit point [2].

2.2 Damage Potential and Effort Ratio

Damage potential and access effort ratio is an informal means that are used to estimate *damage potential* and *effort* in terms of resources attributes [2]. The damage potential depends on the method's privilege, the channel's type, and the data item's type [9], whereas, the effort depends on the rights of the resource that the attacker needs to acquire to use a resource in an attack [9]. The *attackability* ac() of a method, a channel, and a data item is given as follows:

$$ac(method) = \frac{privilage}{access\ right} \quad (1)$$

$$ac(channel) = \frac{type}{access\ right} \quad (2)$$

$$ac(data\ item) = \frac{type}{access\ right} \quad (3)$$

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, channel type, or data item, the higher the damage potential, whereas the higher the access right the higher the effort [9].

2.3 Attack Surface Measurement Method

A system attack surface, for a given system, is measured along three dimensions: method, channel, and data items as follows:

- The entry and exit points' framework is used to identify the set of methods (M), channels (C), and untrusted data items (I).
- Damage potential and effort ratio means, der(), is used to estimate the total contribution of the method: $der_m(m)$, channel: $der_c(c)$, and data items: $der_d(d)$. Where m ∈ M, c ∈ C, and d ∈ I.
- The system's attack surface is the triple

$$< \sum_{m \in M} \mathrm{derm(m)}, \sum_{c \in C} \mathrm{derc(c)}, \sum_{d \in I} \mathrm{derd(d)} >$$

3 Vulnerability Density Metric

3.1 Software Vulnerabilities

Software vulnerability is defined as a defect in software system that presents a security risk [4]. The subset of the security related defects, termed vulnerabilities, are to be discovered and are announced eventually [4]. 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 organizations such as National Vulnerability Database (NVD) [10], Open Source Vulnerability Database (OSVDB) [11] and BugTraq [12], as well as the vendors of the software. Vulnerabilities are assigned a unique identifier (local to individual information providers and across multiple vulnerability databases) using MITRE Common Vulnerability and Exposure (CVE) service [13].

3.2 Classification of Vulnerabilities

Distinction among vulnerabilities can be significant, especially when examining the nature and extent of the problem is required. It also helps in determining the most effective kinds of protective actions. Several classification schemes have been proposed [5]. A vulnerability

classification should be considered ideal if it possesses desirable properties such as mutual exclusiveness, clear and unique definition, and coverage of all software vulnerabilities [5]. Vulnerabilities can be classified mainly based on how they arise and their relative impact [5].

Woo et al. [5] observe that the high severity vulnerabilities tend to be discovered and patched earlier than the medium and low vulnerabilities. Mandhata and Wong observe that patching two types of vulnerabilities associated with Authentication Issues and Permission, Privilege and Access control reduce the attack surface measurement [2]. However, it should be expected that not only the type but also the severity of vulnerabilities can have an effect on the attack surface measurement.

3.2.1 Vulnerability Severity

The severity level of vulnerability indicates how serious the impact of exploitation can be. Three severity levels are often defined; high, medium and low. However, some other organizations use up to five levels and use their own definition for severity [5]. The NVD of the National Institute of Standards and Technology has used the Common Vulnerability Scoring System (CVSS) metric that assesses the vulnerability severity with range from 0.0 to 10.0 [10]. CVSS uses several factors to determine the severity; the range from 0.0 to 3.9 is taken to be low severity, 4.0 to 6.9 to medium severity and 7.0 to 10.0 to high severity. The NVD (2010) describes three severity levels as follows:

- High Severity: vulnerabilities make it possible for a remote attacker to violate the security protection, or permit a local attack that gains complete control, or are otherwise important enough to have an associated CERT/CC advisory or US-CERT alert.
- Medium Severity: vulnerabilities are those not meeting the definition of either 'high' or 'low' severity.
- Low Severity: vulnerabilities typically do not yield valuable information or control over a system but may provide the attacker with information that may help him find and exploit other vulnerabilities or may be inconsequential for most organizations.

3.2.2 Vulnerability Type Classification

Vulnerability type in National Vulnerability Database (NVD) is assigned using the Common Weakness Enumeration (CWE) maintained by the MITRE Corporation [13]. CWE classifies CVEs by the type of vulnerability they represent [10]. As in the [10], the type of vulnerabilities that are related to attack surface can be classified into:

- Authentication Issues: failure to properly authenticate users.
- Permissions, Privileges, and Access Control: failure to enforce permissions or other access restrictions for resources, or a privilege management problem.
- Cross-Site Scripting (XSS): failure of a site to validate, filter, or encode user input before returning it to another user's web client.

Table 1- Market share of the top web servers on the Internet [15]

Product	Vendor	Web Sites Hosted	Percent
Apache	Apache	378,267,399	**64.91%**
IIS	Microsoft	84,288,985	14.46%
nginx	Igor Sysoev	56,087,776	9.63%
GWS	Google	18,936,381	3.25%

- Format String Vulnerability: the use of attacker-controlled input as the format string parameter in certain functions.
- SQL Injection: when user input can be embedded into SQL statements without proper filtering or quoting, leading to modification of query logic or execution of SQL commands.
- OS Command Injections: allowing user-controlled input to be injected into command lines that are created to invoke other programs, using system () or similar functions.
- Information Leak / Disclosure: exposure of system information, sensitive or private information, fingerprinting, etc.

Other CWE types which might not relate to the attack surface are: Credentials Management, Buffer Errors, Cross-Site Request Forgery (CSRF), Cryptographic Issues, Path Traversal, Code Injection, etc. [10].

3.3 Vulnerability Density Metric

Vulnerability density is a measure of the total number of known vulnerabilities, divided by the size (per thousand lines of code) of the software entity being measured, V_{KD}, [5]:

$$V_{KD} = \frac{\text{Known Vulnerabilities}}{\text{Size}} \qquad (4)$$

Vulnerability density is a normalized measure, given by the number of vulnerabilities per unit of code size. The size is typically measured either in Lines of Code or the installed system in bytes. The former has been chosen due to its simplicity and its correspondence to defect density metric in the software engineering domain. Vulnerability density is used to compare software systems that come under the same category. It can also be used to estimate the number of residual vulnerabilities. Assessing the risk can also be achieved by using vulnerability density. Besides, deciding when to stop testing as well as maintenance planning can be realized using vulnerability density metric [5].

4 Apache HTTP Server

Apache HTTP server has been the most popular Web server. It is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation. Apache HTTP server is the software that responds to requests for information sent by web browsers [14]. It provides information using static HTML pages as well as dynamic and interactive services to the clients using database queries, executable script, etc. It also

supports functions such as serving streaming media, email, etc. In the presence of the cloud computing systems, it can also support virtual implementations of applications and operating systems. Apache can run on a variety of operating systems such as UNIX, Windows, Linux, Solaris, Novell NetWare, FreeBSD, and Mac OS X.

Apache HTTP server 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. This diversity enables us to observe the improvement of the security among the subsequent releases. Besides, the availability of it source code and wide usage makes it a good candidate for a study such as ours. According to [15], Apache web server is used by over 64% (Table1). As Apache HTTP server releases 2.2.x takes more than 88% of the usage share among Apache releases [16], and as there is a significant time gap between the releases 2.2.x and 1.3.x, which might help in observing security improvement, the two releases have been chosen for this study.

The two main components of Apache web server are *Apache Core* and *Apache Modules* [14]. The Apache Core provides the main functionality of Apache HTTP server such as allocating requests and maintaining and pooling all the connections. The Apache Modules handle the other types of processing the server has to provide; performing user Authentication.

Apache Core consists of several small components that manage the essential implementation of Apache web server main functionality. Figure1 depicts the main elements of the Apache Core and how they interact with each other to provide the required functionalities. These components are described below.

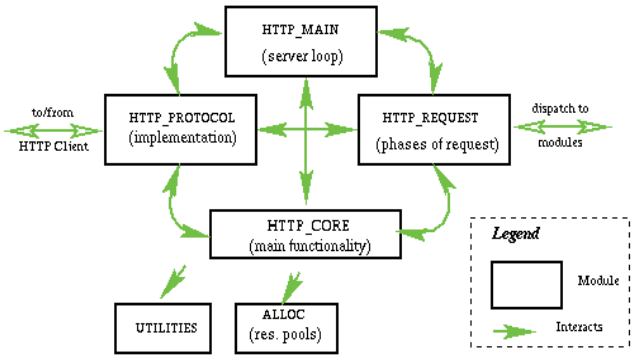

Figure 1: Apache HTTP Server Core Components [14]

- **http_protocol.c** handles the routines that communicate directly with the client by using the HTTP protocol. It handles the socket connections that the client uses to connect to the server and it manages data transfer.
- **http_main.c** is in charge for the startup of the server, contains the main server loop that waits for and accepts connections, and manages timeouts.
- **http_request.c** deals with the flow of request processing, passing control to the modules as needed in the right order, and error handling.
- **http_core.c** implements serving of documents.
- **alloc.c** manages allocation of the resource pool and keeps track of them.
- **http_config.c** provides tasks for other utilities such as reading configuration files, managing the information collected from those files, and aids for virtual hosts.

On the other hand, Apache Modules provide additional functionality that extend and implement the functionality of the Apache HTTP server. Each module is connected to the Apache core, HTTP_REQUEST component, and provides separated functionality. Thus, no module can progress without sending the information to the HTTP_REQUEST component which in turn checks and handles errors.

5 Measuring System Vulnerabilities

The vulnerability datasets for Apache HTTP server 1.3.x and 2.2.x releases have been obtained from NVD which is maintained by National Institute of Standards and Technology and is sponsored by the department of Home Land Security. Thus the datasets are likely to be reliable. The vulnerabilities examined were discovered during the period 1999 to 2012. Most of those vulnerabilities were found to be applicable to both major platforms MS-Windows and Linux. Only few were found to be specific to Mac OS.

5.1 Number of Known Reported Vulnerabilities

The known reported vulnerabilities of Apache web server 1.3 and 2.2 versions are presented in figure 2. In version 1.3, the numbers of the new vulnerabilities, which have been introduced in this version, are 18 whereas the number of the inherited vulnerabilities, which have been inherited from another releases and versions, are 11. However, the total numbers of the new and inherited vulnerabilities in version 2.2 are 11 for the two of them. As it has been observed in figure 2, the numbers of the new vulnerabilities of version 1.3 are larger than its counterpart, version 2.2. However, the inherited numbers of vulnerabilities are the same.

Table 2: Apache Releases Vulnerabilities

Release	Number of Versions	New Vulnerabilities	Inherited Vulnerabilities	Total
1.3	47	51	932	983
2.2	23	23	415	438

Table 2 presents values of number of versions, new and inherited vulnerabilities, and over all vulnerabilities for two Apache web server releases 1.3.x and 2.2.x. The table shows that the numbers of the new and inherited vulnerabilities of Apache 1.3.x release are more than half of 2.2.x release. Moreover, the numbers of the inherited vulnerabilities of 1.3.x release and the 2.2.x release are way larger than the numbers of new vulnerabilities. This could be attributed to the share of code between successive versions and releases as it has been stated by Kim et al [17]. The number of the new and inherited vulnerabilities for all versions of Apache

release 1.3.x and 2.2.x are presented in Figures 3 and 4 respectively.

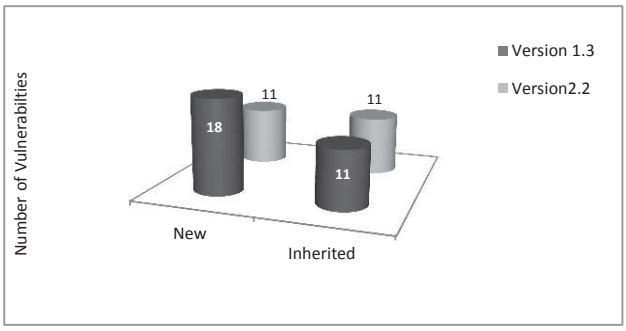

Figure2: Apache HTTP Server Version 1.3 & 2.2 Vulnerabilities

There are forty seven versions for Apache 1.3.x release while twenty three for Apache 2.2.x release. It has been noted that the first two versions for both releases have more new vulnerabilities than the other versions. This is due to the fact that the data available are for the beta versions. The successive versions had significantly less new vulnerability. Half of the versions in both releases did not have any new vulnerability, possibly because their market shares were not as high. Woo et al in [5], have pointed out that higher market share is one of the most important factors impacting the effort spent in exploring and exploiting potential vulnerabilities. Surprisingly, it has been observed that some of the inherited vulnerabilities affect only a specific version(s) in one release but not the others.

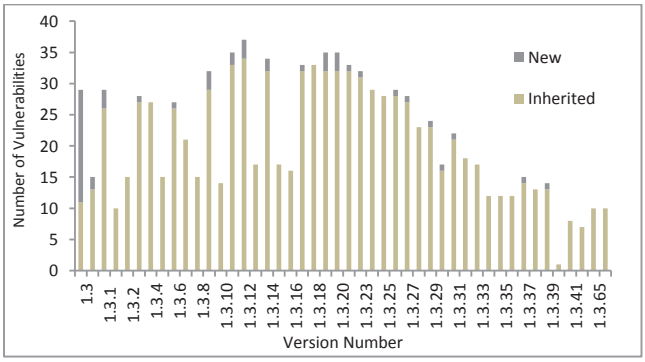

Figure 3: Apache HTTP Server Release 1.3.x Vulnerabilities

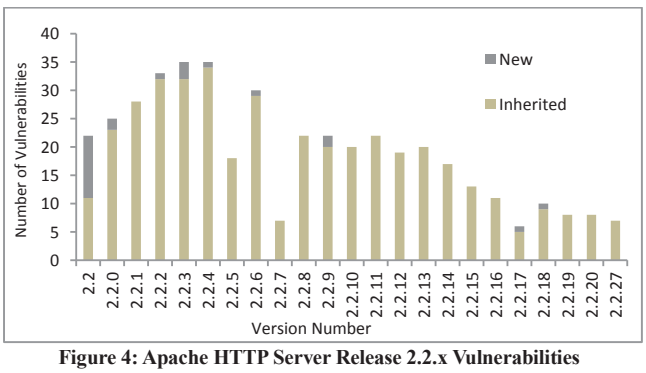

Figure 4: Apache HTTP Server Release 2.2.x Vulnerabilities

Table 3 presents known vulnerability density V_{KD} for two versions 1.3 and 2.2 of Apache web server from two different releases, 1.3.x and 2.2.x. Since the Apache web server is an open-source project, the source code for the two chosen versions was downloaded from Apache HTTP server archive download site [18]. The code size for the two selected versions was determined using SLOCCount tool, a software metrics tool for counting physical source lines of code [19].

The major fractions of the source code of the two versions are C based and was counted using the SLOCCount tool. The available tools that can be used to determine the attack surface can use C as well as Java code. The known vulnerabilities column gives a recent count of the vulnerabilities found since the release date. Despite having smaller size than version 2.2, version 1.3 had more known vulnerabilities and thus higher known vulnerabilities density.

Table 3- New Known Vulnerabilities Density

Application	Ksloc	Known vulnerabilities	VKD
Apache 1.3	50,712	18	3.549E-4
Apache 2.2	222,029	11	4.95E-05

5.2 Vulnerabilities classified by Severity and Type

Figure 5 shows the number of vulnerabilities classified by severity for Apache 1.3 and 2.2 versions. While the former had eight vulnerabilities of high severity, the latter did not have any. Both versions had the same number of vulnerabilities of medium severity. No vulnerabilities of low severity were found in version 1.3 whereas one was found in version 2.2. Overall, version 1.3 had a higher fraction of high and medium severity vulnerabilities. This presents a significant risk and could lead to higher rates of occurrences of denial of service (DoS) attacks, exposure of sensitive information, etc.

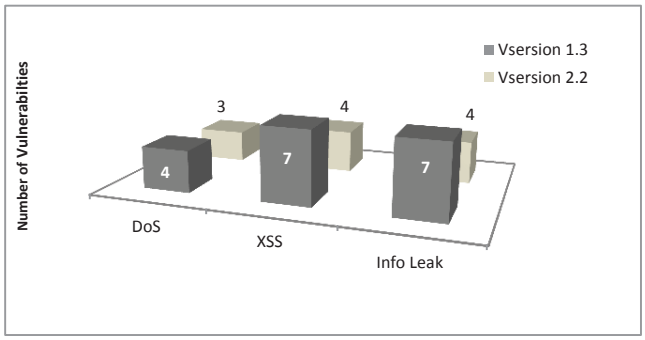

Figure 5: Apache HTTP Server version 2.2 Vulnerability Severity

6 Measuring System' Attack Surface

In this section, we will measure the attack surface along the method dimension for Apache HTP Server 1.3 and 2.2 versions. Choosing the method dimension in particular is due

to the fact that attackers can only exploit vulnerability in a method if they can invoke that method either directly or indirectly. However, measuring the attack surface requires looking at the source code and finding all places which could be part of the attack surface. By finding such places, what needed next is classifying each one of them into an attack class. The source code of the two chosen versions was obtained from [18].

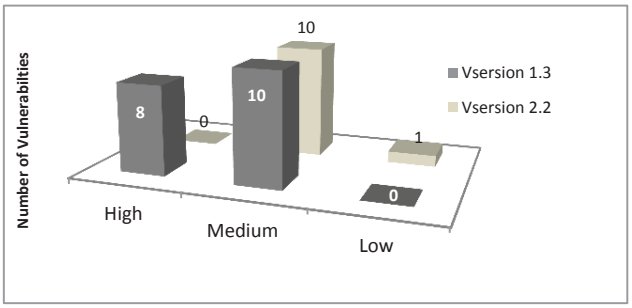

Figure 6: Apache HTTP Server version 2.2 Vulnerability Type

The entry and exit points along the method dimension have been defined using *cflow* tool. It analyzes a source code written in C programming language and produces a graph charting dependencies between various functions [20]. Figure 7 shows some sample cflow output from Apache HTTP Server version 1.3. Before counting each method, functions reachable from the main(), in the attack surface, the privilege level and access right level of the entry and exit points are needed to be determined and marked.

```
1 API_EXPORT() < at http_protocol.c:1927>:
2    ap_table_get()
3    strncasecmp()
4    strcasecmp()
5    strchr()
6    parse_byterange() <*support code ... */int parse_byterange
(char *range,long clength,long *start,long *end) at
http_protocol.c:85>
7    ap_pstrdup()
```

Figure 7: Fractional cflow output from Apache Web Server 1.3

First, the privilege of a process runs on a UNIX system can change from one level to the other using one of uid-setting system call such as *setuid*. If a process drops it privilege, then every method that are invoked before the drop of the privilege will have that level of privilege whereas every method invoked after the drop of the privilege will have the new privilege. Thus, a method might be counted multiple times, once per each privilege level. The methods in the Apache HTTP Server version 1.3 and 2.2 run with a root and nobody, www-data, manager.sys, or apache privilege.

Second, in order to the access right of a method to be determined, we need to define the source code where the authentication is presented. Based on that, every method invoked before user authentication performed has unauthenticated access right while those methods invoked after user authentication take place has authenticated access right.

As it can be seen in table 4, a value to each privilege level and access right level is assigned based on our knowledge of Apache HTTP server and Linux (Ubuntu). The number of the direct entry point and the direct exit point are shown in table 5. Nonetheless, using table 4 and 5, the attackability along the method direction considering the effect of the resource's Damage Potential-Effort Ratio was calculated as shown in table 6. The measure of Apache 1.3 attack surface along the method dimension is **<235.3>** whereas the measure of Apache 2.3 along the same dimension is **<207.3>.**

Table 4: Numeric values of Privilege & Access Rights

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

Table 5: Direct Entry and Exit Points

Apache 1.3			
Privilege	**Access Right**	**Direct Entry Point**	**Direct Exit Point**
root	root	4	7
root	authenticated	11	6
apache	authenticated	10	18
apache	unauthenticated	9	15
apache	anonymous	2	17
manager.sys	authenticated	1	12
Apache 2.2			
Privilege	**Access Right**	**Direct Entry Point**	**Direct Exit Point**
root	root	1	5
root	authenticated	9	14
apache	authenticated	7	16
apache	unauthenticated	5	23
apache	anonymous	3	14

Table 6: Attackability Measurements

Code Base	Total Contribution of the Methods	Total
Apache 1.3	11 (5/5) + 17 (5/3) + 28 (3/3) + 24 (3/1) + 19 (3/1) + 13 (3/1)	**235.3**
Apache 2.2	6 (5/5) + 23 (5/3) + 23 (3/3) + 28 (3/1) + 7 (3/1)	**207.3**

7 Observations

The total numbers of vulnerabilities as well as vulnerability density for Apache HTTP Server version 1.3 and its attack surface (Entry/Exit points along the method dimension) have been found to be more than those in Apache HTTP Server version 2.2. This result suggests that the vulnerability density seems to have a non-linear relationship with the attack surface. Information about the type and severity of vulnerabilities could simplify the process of the attack surface measurement. Knowing the attack points of a system could help developers to identify code with a higher likelihood of exploitable vulnerabilities. This result also suggests that a possible framework that combines the two metrics should be further investigated. Further experiments with different software systems need to be conducted to confirm and extend the result of this study.

8 Conclusion & Future work

This paper examines the relationship between the attack surface and vulnerability density metrics. It also investigates the visibility of how the two metrics can be used to guide each other.

Quantitative techniques are needed by the developers and users for optimizing resource allocation, program planning, risk assessment and product and service selection. Software Security metrics are supposed to fulfill this need. However validation and comparison of the metrics are needed.

Attack surface measures system attackability while vulnerability density measures the density of the known vulnerability of a system. While both metrics have been applied to some of the major software systems they have not been compared.

In this study, we have measured the attackability and the vulnerability density of Apache HTTP server 1.3 and 2.2 using the two metrics. Results suggest that the two metrics are related but not linearly. The attackability in terms of the methods and the vulnerability density measures of the Apache HTTP Server version 1.3 are both greater than the version 2.2. Further studies need to be undertaken to model the specific relationship between the two. It has been also observed that not only the number of vulnerability or vulnerability density that is related to attack surface but also the type and the severity of vulnerabilities are of degree of importance.

While only new vulnerabilities were considered in this study, examining the inherited vulnerabilities may lead to further insight. Moreover, identifying the vulnerable component of a software system by applying a compound metric that can define the most vulnerable modules can be very useful. Such a metric may involve using both vulnerability density as well as attack surface.

9 References

[1] M. Howard, J. Pincus, and J.M. Wing “Measuring Relative Attack Surfaces ,” Chapter 8, in Computer Security in the 21st Century , D.T. Lee, S.P. Shieh, and J.D. Tygar, editors, Springer, March 2005, pp. 109-137.

[2] P.K Manadhata and J.M Wing. An Attack Surface Metric. IEEE Transactions on Software Engineering, vol. 37, No. 3, May, 2011.

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

[4] O. H. Alhazmi, Y. K. Malaiya, and I. Ray, “Measuring, analyzing and predicting security vulnerabilities in software systems,”Computers and Security Journal, vol. 26, no. 3, pp. 219–228, May 2007.

[5] S.-W. Woo, H. Joh, O. H. Alhazmi and Y. K. Malaiya, "Modeling Vulnerability Discovery Process in Apache and IIS HTTP Servers", Computers & Security, January 2011, pp. 50-62.

[6] W. Jansen, “Directions in Security Metrics Research,” NIST, NISTIR 7564, p. 21, Apr., 2009.

[7] K. Goertzel, T. Winograd, H. McKinley, L. Oh, M. Colon, T. McGibbon, E. Fedchak, R. Vienneau, Software security assurance: A state-of-art report (soar), Tech. rep., Information Assurance Technology Analysis Center (IATAC), 2007.

[8] V. Verendel, Quantified security is a weak hypothesis: a critical survey of results and assumptions. In: NSPW ’09: Proceedings of the 2009 workshop on new security paradigms workshop, pp 37–50. ACM, New York, NY, USA.

[9] P. Manadhata and J. M. Wing. An attack surface metric. In Technical Report CMU-CS-05-155, 2005.

[10] National Vulnerability Database (NVD), http://nvd.nist.gov/; April 2012.

[11] Open Source Vulnerability Database (OSVDB), http://osvdb.org; April 2012.

[12] Securityfocus, http://www.securityfocus.com/; April 2012.

[13] http://cve.mitre.org/; February 2012.

[14] The Complete Reference, Apache Server 2.0. Ryan B. Bloom. McGraw-Hill/Osborne Media; 1st edition (June 26, 2002).

[15] http://news.netcraft.com/archives/2012/01/03/january-2012-web-server-survey.html; March 2012.

[16] http://w3techs.com/technologies/overview/web_server/all; March 2012.

[17] Kim J, Malaiya YK, Ray I., Vulnerability discovery in multi-version software systems. In: Proc. 10th IEEE Int. symp. On high assurance system engineering (HASE), Dallas; Nov. 2007. p. 141-148.

[18] http://archive.apache.org/dist/httpd/; March 2012.

[19] SLOCCount, http://dwheeler.com/sloccount; March 2012.

[20] http://www.gnu.org/software/cflow/manual/cflow.html