Measuring the Attack Surfaces of Two FTP Daemons

Pratyusa Manadhata, Jeannette Wing
Carnegie Mellon University
{pratyus,wing}@cs.cmu.edu

Mark Flynn, Miles McQueen
Idaho National Laboratory
{Mark.Flynn, Miles.McQueen}@inl.gov

ABSTRACT

Software consumers often need to choose between different software that provide the same functionality. Today, security is a quality that many consumers, especially system administrators, care about and will use in choosing one software system over another. An *attack surface metric* is a security metric for comparing the relative security of similar software systems [7]. The measure of a system's *attack surface* is an indicator of the system's security: given two systems, we compare their attack surface measurements to decide whether one is more secure than another along each of the following three dimensions: methods, channels, and data. In this paper, we use the attack surface metric to measure the attack surfaces of two open source FTP daemons: ProFTPD 1.2.10 and Wu-FTPD 2.6.2. Our measurements show that ProFTPD is more secure along the method dimension, ProFTPD is as secure as Wu-FTPD along the channel dimension, and Wu-FTPD is more secure along the data dimension. We also demonstrate how software consumers can use the attack surface metric in making a choice between the two FTP daemons.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics; D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Security

Keywords

Attack Surface, Attack Surface Metric, Security Metric

1. INTRODUCTION

Measurement of a software system's security, both qualitatively and quantitatively, has been a long standing challenge to the research community, and is of practical import today. Software consumers often face the task of choosing one software product from a set of competing and alternative products that provide similar functionality. For example, system administrators often make a choice between different available web servers, IMAP servers, and FTP servers for their organization. Several factors such as ease of installation, maintenance, and use, and interoperability with existing enterprise software are relevant to software selection; however, to consider the security of a system when choosing between alternative software systems is of heightened interest to consumers today.

Manadhata and Wing have proposed an *attack surface metric* to compare the security of similar software systems, i.e., different versions of the same system or different systems with similar functionality [7]. Intuitively, a system's attack surface is the set of ways in which an adversary can attack the system. Hence the larger the attack surface, the more insecure the system. Given two systems, Manadhata and Wing measure their *attack surfaces*, and compare their attack surface measurements along three dimensions to indicate whether one system is more secure than another along each dimension. While it is very difficult to devise metrics that definitively measure the security of software, prior work has shown that a system's attack surface measurement is a good indicator of the system's security ([5, 6]).

In this paper, we use the attack surface metric to measure the attack surfaces of two open source FTP daemons: ProFTPD 1.2.10 and Wu-FTPD 2.6.2. Our choice of the FTP daemons is guided by two factors: popularity and availability of source code. We compare the attack surface measurements of the two FTP daemons along three dimensions: methods, channels, and data. Our results show that the measure of ProFTPD's attack surface is smaller than Wu-FTPD along the method dimension, the same as Wu-FTPD along the channel dimension, and larger than Wu-FTPD along the data dimension. We partially validate the attack surface measurements by correlating the attack surface measurements of both daemons with the number of vulnerability reports found for the daemons.

The rest of the paper is organized as follows. In Section 2, we give a brief overview of the attack surface metric. We measure the attack surfaces of the two daemons and compare the measurements in Section 3. We validate the measurements in Section 4. We discuss related work in Section 5. Finally, we conclude with a discussion of future work in Section 6.

2. ATTACK SURFACE METRIC

Intuitively, a system's attack surface is the set of ways in which an adversary can enter the system and potentially cause damage. We know from the past that many attacks,

QoP'06, October 30, 2006, Alexandria, Virginia, USA.

e.g., exploiting a buffer overflow, on a system take place by sending data from the system's operating environment into the system. Similarly, many other attacks, e.g., symlink attacks, on a system take place because the system sends data into its environment. In both these types of attacks, an attacker connects to a system using the system's *channels* (e.g., sockets), invokes the system's *methods* (e.g., API), and sends *data items* (e.g., input strings) into the system or receives data items from the system. An attacker can also send data indirectly into a system by using data items that are persistent. Specific examples of persistent data items are files, cookies, registry entries, and database records. An attacker can send data into a system by writing to a file that the system later reads. Similarly, an attacker can receive data indirectly from the system by using shared persistent data items. Hence an attacker uses a system's methods, channels, and data items present in the system's environment to attack the system. We collectively refer to a system's methods, channels, and data items as the system's *resources*.

Given the above observation, a system's attack surface is defined in terms of the system's resources. Not all resources, however, are part of the attack surface, and not all resources contribute equally to the measure of a system's attack surface. In order to measure a system's attack surface, we need to identify the relevant resources that are part of the system's attack surface, and determine the contribution of each such resource to the system's attack surface. Manadhata and Wing have introduced a formal *entry point and exit point framework* to identify the relevant resources that contribute to a system's attack surface; they also have introduced the informal notions of *damage potential* and *effort* to estimate a resource's contribution to a system's attack surface [7].

2.1 Attack Surface Definition

A system's attack surface is the subset of resources that an attacker can use to attack the system. An attacker attacks a system either by sending data into the system, or by receiving data from the system; hence any resource that the attacker can use either to send data into the system or to receive data from the system is part of the system's attack surface. Intuitively, the more resources available to an attacker, the more ways a system can be attacked, hence the more insecure it is. We use the entry point and exit point framework to identify the resources that are part of a system's attack surface.

2.1.1 Entry Points

The methods of a system that receive data items from the system's environment are the system's entry points. For example, a method that receives input from a user or a method that reads a configuration file is an entry point. A method m of a system s receives data items *directly* if either (a) a user or a system in s's environment invokes m and passes data items as input to m, or (b) m reads from a persistent data item, or (c) m invokes the API of a system in s's environment and receives data items as the result returned. A method m receives data items *indirectly* if either (a) a method m_1 of s receives a data item d directly, and either m_1 passes d as input to m or m receives d as result returned from m_1, or (b) a method m_2 of s receives a data item d indirectly, and either m_2 passes d as input to m or m receives d as result returned from m_2. A method m of s is a *direct entry point* if m receives data items directly, and an *indirect entry point* if m receives data items indirectly.

2.1.2 Exit Points

The methods of a system that send data items to the system's environment are the system's exit points. For example, a method that writes into a log file is an exit point. A method m of a system s sends data items *directly* if either (a) a user or a system in s's environment invokes m and receives data items as results returns from m, or (b) m writes to a persistent data item, or (c) m invokes the API of a system in s's environment and passes data items as input to the API. A method m sends data items *indirectly* if either (a) m passes a data item d as input to a method m_1 and m_1 passes d either directly or indirectly to s's environment, or (b) a method m_2 receives a data item d as result returned from m and m_2 passes d either directly or indirectly to s's environment. A method m of s is a *direct exit point* if m sends data items directly, and an *indirect exit point* if m sends data items indirectly.

2.1.3 Channels

An attacker uses a system's channels to connect to the system and attack the system. Hence a system's channels act as another basis for attacks. Specific examples of channels are TCP/UDP sockets and pipes.

2.1.4 Untrusted Data

An attacker uses persistent data items either to send data indirectly into the system or receive data indirectly from the system. A system might read from a file after an attacker writes into the file. Similarly, the attacker might read from a file after the system writes into the file. Hence the persistent data items act as another basis for attacks on a system. An *untrusted data item* of a system s is a persistent data item d such that a direct entry point of s reads from d or a direct exit point of s writes into d.

2.1.5 Attack Surface

By definition, the relevant resources that contribute to the attack surface are the set of entry points and exit points, the set of channels, and the set of untrusted data items. Hence a system's *attack surface* is the triple, $\langle M, C, I \rangle$, where M is the set of entry points and exit points, C is the set of channels, and I is the set of untrusted data items of the system.

2.2 Attack Surface Measurement

A naive way of measuring a system's attack surface is to count the number of resources that contribute to the attack surface. This naive method that gives equal weight to all resources is misleading since all resources are not equally likely to be used by an attacker. For example, a method, m_1, running as `root` is more likely to be used in an attack than a method, m_2, running as `non-root`; hence m_1's contribution to the attack surface is larger than m_2.

We estimate a resource's contribution to a system's attack surface as a *damage potential-effort ratio* where *damage potential* is the level of damage the attacker can cause to the system in using the resource in an attack, and *effort* is the effort the attacker spends to acquire the necessary access rights in order to be able to use the resource in an attack.

The higher the damage potential, the higher the contribution; the higher the effort, the lower the contribution.

2.2.1 Damage Potential-Effort Ratio

We use informal means to estimate damage potential and effort in terms of the attributes of a resource. The estimates depend on the kind of the resource, i.e., method, channel, or data item.

An attacker gains the same privilege as a method by using a method in an attack. For example, the attacker gains `root` privilege by exploiting a buffer overflow in a method running as `root`. Hence we estimate a method's damage potential in terms of the method's *privilege*.

The attacker uses a system's channels to connect to a system, and send (receive) data to (from) a system. A channel's *protocol* imposes restrictions on the data exchange allowed using the channel, e.g., a `TCP socket` allows raw bytes to be exchanged whereas a `RPC endpoint` does not. Hence we estimate a channel's damage potential in terms of the channel's protocol.

The attacker uses persistent data items to send (receive) data indirectly into (from) a system. A persistent data item's *type* imposes restrictions on the data exchange, e.g., a file can contain executable code whereas a `registry entry` can not. Hence we estimate a data item's damage potential in terms of the data item's type.

The attacker can use a resource in an attack if the attacker has the required *access rights*. The attacker spends effort to acquire these access rights. Hence for the three kinds of resources, i.e., method, channel, and data, we estimate the effort the attacker needs to spend to use a resource in an attack in terms of the resource's access rights.

We assign numbers to the values of the attributes to compute a numeric damage potential-effort ratio. We describe a specific method of assigning numbers in Section 3.2

2.2.2 Attack Surface Measurement Method

We measure a system's attack surface along three dimensions by estimating the total contribution of the methods, the total contribution of the channels, and the total contribution of the data items to the system's attack surface.

1. Given a system s and its environment, we identify a set, M, of entry points and exit points, a set, C, of channels, and a set, I, of untrusted data items of s.

2. We estimate the damage potential-effort ratio, $der_m(m)$, of each method $m \in M$, the damage potential-effort ratio, $der_c(c)$, of each channel $c \in C$, and the damage potential-effort ratio, $der_d(d)$, of each data item $d \in I$.

3. The measure of s's attack surface is the triple $\langle \sum_{m \in M} der_m(m), \sum_{c \in C} der_c(c), \sum_{d \in I} der_d(d) \rangle$.

3. ATTACK SURFACE MEASUREMENT OF FTP DAEMONS

In this section, we describe the process of measuring the attack surfaces of the two FTP daemons. ProFTPD was implemented and is maintained by the ProFTPD project group [9]. Wu-FTPD was implemented and is maintained at Washington University [**?**]. The ProFTP codebase contains 28K lines of C code and the Wu-FTP codebase contains 26K lines of C code; we only considered code specific to the FTP daemon.

Figure 1 shows the steps followed in measuring the attack surfaces of the FTP daemons. The dotted boxes show the steps done manually, and the solid boxes show the steps done programmatically. The dotted lines represent manual inputs required for measuring the attack surfaces of the FTP daemons.

3.1 Identification of Relevant Resources

In Step 1 of the attack surface measurement method, we identified the set of entry points and exit points, the set of channels, and the set of untrusted data items for both code bases. We also determined the privilege levels of the set of entry points and exit points, the protocols of the set of channels, the types of the set of untrusted data items, and the access rights levels of all the resources. Most of the substeps in this step are automatic and we used off-the-shelf tools for the automated steps.

3.1.1 Entry Points and Exit Points

Recall that a direct entry point of a system is a method that receives data items directly from the system's environment. As proposed by DaCosta et al. [3], we assume that a method of a system can receive data items from the system's environment by invoking specific `C` library methods. Hence a method is a direct entry point if the method contains a call to one of the specific `C` library methods. For example, a method is a direct entry point if it contains a call to the `read` method defined in `unistd.h`. The only exception is the `main` method. The `main` method is a direct entry point as the attacker can invoke `main` and pass inputs to `main`. We identified a set, $Input$, of `C` library methods that a method must invoke to receive data items from the environment. We identified the methods of the ProFTP and the Wu-FTP codebases that contained a call to a method in $Input$ as the direct entry points.

A direct exit point of a system is a method that sends data items directly to the system's environment. We assume that a method can send data items to the system's environment by invoking specific `C` library methods. We identified a set, $Output$, of `C` library methods that a method must invoke to send data items to the environment. We identified the methods of the ProFTP and the Wu-FTP codebases that contained a call to a method in $Output$ as the direct exit points. Please see the appendix for the $Input$ and $Output$ sets of methods.

An indirect entry point of a system is a method that receives data items from a direct entry point. An indirect exit point of a system is a method that sends data items to a direct exit point. We could not find a source code analysis tool that enables us to determine whether a direct entry point m_1 receives a data item d from the environment and a method m receives the data item d from m_1, or whether a method m passes a data item d to a direct exit point m_2 and m_2 sends the data item d to the environment; hence we could not identify the indirect entry points or the indirect exit points in an automated manner. Hence our measurements are under-approximations of the measure of the attack surfaces.

We also identified the privilege level and access rights level of the entry points and the exit points. On a UNIX system, a process changes its privilege through a set of *uid-setting*

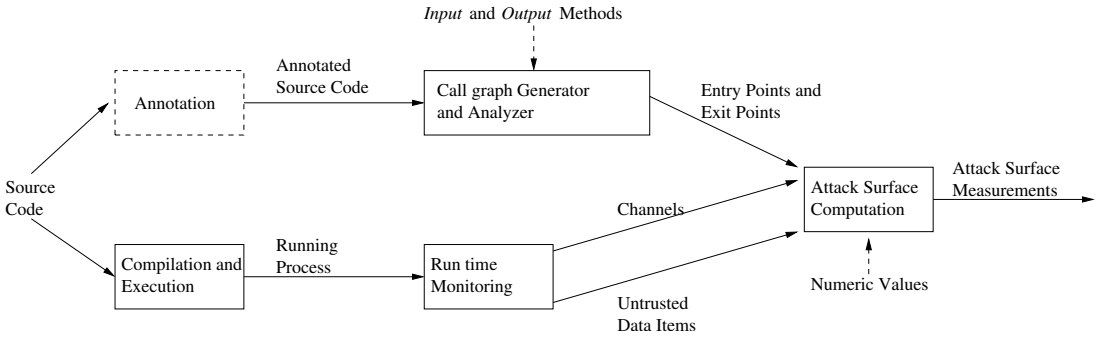

Figure 1: Attack surface measurement steps.

system calls such as `setuid`. If a process changes its privilege level from p_1 to p_2 by invoking a uid-setting system call, then we assume that all methods invoked before the uid-setting call run with privilege p_1 and all methods invoked after the uid-setting system call run with privilege p_2. For example, if a process starts with `root` privilege, and then drops privilege by calling `setuid`, then all methods that are invoked before `setuid` have `root` privilege, and all methods that are invoked after `setuid` have `non-root` privilege.

In order to determine the access rights levels, we identified the code locations where authentication is performed in both codebases. We assumed that any method that is invoked before user authentication takes place has unauthenticated access rights, and any method that is invoked after successful authentication has authenticated access rights.

We annotated each codebase to indicate the code location where privilege levels and access rights levels change. We generated the call graph of the annotated code using `cflow` [2]. From the call graph, we identified the methods that contained a call to a method in *Input* or a method in *Output*, and the privilege and access rights of each such method. These identified methods are the direct entry points and direct exit points respectively. Notice that a method may run with different privilege levels during different executions of the method. Similarly, a method may be accessible with multiple access rights levels. We identified such a method as a direct entry point (direct exit point) multiple times, once per each pair of privilege level and access rights level.

The methods in the ProFTP codebase run with `root` and a special UNIX user, `proftpd`, privilege. The methods are accessible with `root`, `authenticated` user, `unauthenticated` user, and `anonymous` user access rights. The methods in the Wu-FTP codebase run with `root` and `authenticated` user privilege. The methods are accessible with `authenticated` user, `anonymous` user, and `guest` user access rights. We show the number of direct entry points (DEP) and direct exit points (DExP) for each privilege level and access rights level pair in Table 1.

3.1.2 Channels

We monitored the run time behavior of the default installations of both FTP daemons to identify the channels opened by both FTP daemons, and to determine the protocol and access rights level of each such channel. Both daemons open a `TCP` channel so that FTP clients can communicate with the daemons. These channels are accessible with `remote unauthenticated (RU)` user access rights. We show the number of channels for each protocol and access rights (AR) pair in Table 2.

ProFTPD			
Privilege	Access Rights	DEP	DExP
`root`	`root`	8	8
`root`	`authenticated`	12	13
`root`	`unauthenticated`	13	14
`proftpd`	`authenticated`	6	4
`proftpd`	`unauthenticated`	13	6
`proftpd`	`anonymous`	6	4
Wu-FTPD			
Privilege	Access Rights	DEP	DExP
`root`	`authenticated`	9	2
`root`	`unauthenticated`	30	9
`authenticated`	`authenticated`	11	3
`authenticated`	`anonymous`	11	3
`authenticated`	`guest`	27	14

Table 1: The number of direct entry points and direct exit points in both codebases.

ProFTPD			Wu-FTPD		
Protocol	AR	Count	Protocol	AR	Count
`TCP`	`RU`	1	`TCP`	`RU`	1

Table 2: The number of channels opened by both daemons.

3.1.3 Untrusted Data Items

We monitored the run time behavior of the default installations of both FTP daemons to identify the untrusted data items of both FTP daemons, and the type and access rights level of each untrusted data item. Both daemons read or wrote persistent data items of `file` type; both daemons used configuration files, authentication files, executable files, libraries, and log files. The files of ProFTPD can be accessed with `root`, `proftpd` user, and `world` access rights. The files of Wu-FTPD can be accessed with `root`, `authenticated` user, and `world` access rights. Recall that an attacker can use an untrusted data item in an attack by reading or writing the data item. Hence we identified the read and the write access rights levels of a file separately; we counted each file twice, once for the read access rights level, and once for the write access rights level. We show the number of untrusted

data items for each data item type and access rights (AR) pair in Table 3.

ProFTPD			Wu-FTPD		
Type	AR	Count	Type	AR	Count
`file`	`root`	12	`file`	`root`	23
`file`	`proftpd`	18	`file`	`auth`	12
`file`	`world`	12	`file`	`world`	9

Table 3: The number of untrusted data items used by both daemons.

3.2 Estimation of a Resource's Damage Potential-Effort Ratio

In Step 2 of the attack surface measurement method, we quantified the damage potential-effort ratios of the resources of both FTP daemons.

We imposed a total ordering among the privilege levels such that a method running with a higher privilege level in the total ordering has a higher damage potential. We assigned numeric values to the privilege levels in accordance to the total ordering, i.e., if a privilege level, p_1, is greater than a privilege level, p_2, in the total ordering, then we assign a higher number to p_1 compared to p_2. For example, we assumed a method running as `root` has a higher damage potential than a method running as `authenticated` user; hence `root` > `authenticated user` in the total ordering, and we assigned a higher number to `root` than `authenticated user`. The exact choice of the numeric values is subjective and depends on a system and its environment. We assigned the numeric values based on our knowledge of UNIX security and the FTP daemons. Similarly, we assigned numeric values to channel protocols, data item types, and access rights levels. We estimated a resource's damage potential-effort ratio from the numeric values assigned to the resource's damage potential and effort. For example, we estimated the damage potential-effort ratio of a method from the numeric values assigned to the method's privilege and access rights level.

We assigned the following total ordering among the set of privilege levels: `root` > `proftpd` > `authenticated`. A method running with `proftpd` privilege in ProFTPD has access to all the files on the FTP server, hence we assumed a method running as `proftpd` user has higher damage potential than a method running as `authenticated` user. We assigned the following total ordering among the set of access rights levels of the methods: `root` > `authenticated` > `anonymous` = `unauthenticated` = `guest`. Both FTP daemons have channels with only `TCP` as the protocol and `remote unauthenticated` access rights; hence assigning a total ordering was trivial. Also, both FTP daemons have untrusted data items of `file` type only; hence assigning a total ordering was trivial. We assigned the following total ordering among the access rights levels of the data items: `root` > `proftpd` > `authenticated` > `world`. The `proftpd` user is a special user, hence we assumed the attacker spends more effort to acquire `proftpd` access rights compared to `authenticated` access rights. We show the numeric values in Table 4.

3.3 Attack Surface Measurements

In Step 3 of the attack surface measurement method, we estimated the total contribution of the methods, the total contribution of the channels, and the total contribution of the data items to the attack surfaces of both FTP daemons using the number of relevant resources identified in Step 1 and the numeric values assigned in Step 2. From Table 1 and Table 4, the total contribution of the methods of ProFTPD is $(16 \times (\frac{5}{5}) + 25 \times (\frac{5}{3}) + 10 \times (\frac{4}{3}) + 19 \times (\frac{4}{1}) + 10 \times (\frac{4}{1}))$ = 321.9. From Table 2 and Table 4, the total contribution of the channels of ProFTPD is $1\times\ (\frac{1}{1}) = 1$. From Table 3 and Table 4, the total contribution of the data items of ProFTPD is $(12 \times (\frac{1}{5}) + 18 \times (\frac{1}{4}) + 12 \times (\frac{1}{1})) = 18.9$. Hence the measure of ProFTPD's attack surface is the triple $\langle 312.99, 1.00, 18.90 \rangle$. Similarly, the measure of Wu-FTPD's attack surface is the triple $\langle 392.33, 1.00, 17.60 \rangle$. We show the attack surface measurements in Figure 2.

Method Privilege	Value	Access Rights	Value
`root`	5	`root`	5
`proftpd`	4	`authenticated`	3
`authenticated`	3	`anonymous`	1
		`unauthenticated`	1
		`guest`	1
Channel Protocol	Value	Access Rights	Value
`TCP`	1	`remote unauth`	1
Data Item Type	Value	Access Rights	Value
`file`	1	`root`	5
		`proftpd`	4
		`authenticated`	3
		`world`	1

Table 4: Numeric values assigned to the values of the attributes.

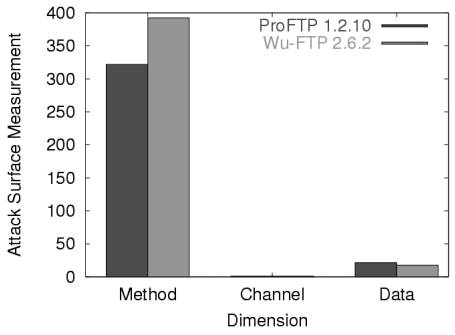

Figure 2: Attack surface measurement results.

We carried out a parameter sensitivity analysis for the numeric values shown in Table 4. We doubled all the values once, doubled the largest values, and halved the smallest values; the ordering of the attack surface measurements of both daemons along the three dimensions did not change in any of the cases.

Wu-FTPD has a higher measure along the method dimension as it has a larger number of methods running with `root` privilege and accessible with `unauthenticated` user access rights. Similarly, ProFTPD has a higher measure along the data dimension as it has a larger number of files accessible with `world` access rights. Also notice that the damage potential-effort ratio does not make a difference along the method dimension, but it does along the data dimension. Wu-FTPD has a larger number of entry points and exit

points and a higher measure of the attack surface along the method dimension; hence the naive measure of the attack surface as the total number of entry points and exit points (damage potential-effort ratio = 1 for all entry points and exit points) will result in the same ordering of the daemons along the method dimension. On the other hand, Wu-FTPD has a larger number of data items, but a lower measure of the attack surface along the data dimension; hence the naive measure will result in a misleading ordering of the daemons along the data dimension. Hence the estimation of the damage potential-effort ratio is a required step in the attack surface measurement process.

The attack surface metric tells us that ProFTPD is more secure along the method dimension, ProFTPD is as secure as Wu-FTPD along the channel dimension, and Wu-FTPD is more secure along the data dimension. In order to choose one FTP daemon over another, we use our knowledge of the FTP daemons and the operating environment to decide which dimension of the attack surface presents more risk, and choose the FTP daemon that is more secure along that dimension. For example, if we are concerned about privilege elevation on the host running the FTP daemon, then the method dimension presents more risk, and the attack surface metric suggests that we choose ProFTPD over Wu-FTPD. If we are concerned about the number of open channels on the host running the FTP daemon, then the channel dimension presents more risk, and we may choose either of the daemons. If we are concerned about the safety of files stored on the FTP server, then the data dimension presents more risk, and we choose Wu-FTPD.

The measurement of the attack surface along three dimensions offers a design choice to software consumers. For example, keeping the attack surface measurement separated along three different dimensions, rather than coalescing the numbers into one, lets system administrators choose a dimension appropriate for their need.

4. VALIDATION

In this section, we describe our process of partially validating the attack surface measurements of both FTP daemons. We validate the attack surface measurements along the method dimension by correlating the measurements with the number of vulnerability reports found for each FTP daemon. A vulnerability report for a software system describes an *exploitable* vulnerability present in the system. An attacker can exploit a vulnerability if the attacker can directly or indirectly invoke the method that contains the vulnerability. In the entry point and exit point framework, an attacker can invoke a method of a system directly or indirectly only by invoking an entry point or exit point of the system. Hence there is a relationship between a vulnerability report for a system and an entry point or an exit point of the system. A system's attack surface measurement along the method dimension depends on the entry points and exit points of the system; hence we expect to see a correlation between the attack surface measurement along the method dimension and the number of vulnerability reports for the system. Given two systems, A and B, if A has a larger attack surface compared to B along the method dimension, then we expect to see a larger number of vulnerability reports for A compared to B.

We counted the number of times ProFTPD 1.2.10 and Wu-FTPD 2.6.2 appear in MITRE CVEs [8], CERT advisories [1], and the SecurityFocus vulnerabilities database [11]. We show the results in Table 5. Wu-FTPD has a larger attack surface compared to ProFTPD along the method dimension; as expected, the number of vulnerability reports for Wu-FTPD is larger than ProFTPD.

Database	ProFTPD	Wu-FTPD
CERT	0	1
MITRE	2	4
SecurityFocus	3	7

Table 5: There are more vulnerability reports for Wu-FTPD 2.6.2 than for ProFTPD 1.2.10.

The project goals mentioned on the ProFTPD website also validate our measurements [10]. Many developers of ProFTPD had spent considerable amount of time fixing bugs and adding new features to Wu-FTPD; they realized that a redesign was necessary to add security, configurability, and new features. Hence ProFTPD was designed and implemented from the ground up to be a secure and configurable FTP server.

5. RELATED WORK

Michael Howard informally introduced the notion of attack surface for the Windows operating system [4]. Pincus and Wing [5] further elaborated, and Manadhata and Wing [6] generalized Howard's Relative Attack Surface Quotient (RASQ) measurements. Howard, Pincus, and Wing identified twenty attack vectors for the Microsoft Windows operating system and compared the attack surface measurements of seven versions of Windows [5]. An attack vector or an attack class of a system is a feature of the system that is often used in attacks on the system. For example, Windows is frequently attacked through the services running in the system. Howard, Pincus, and Wing used the history of attacks on Windows to identify the twenty attack vectors. Similarly, Manadhata and Wing identified fourteen attack classes for Linux using the history of attacks on Linux, and compared the attack surface measurements of four different versions of Linux [6].

The Windows and Linux measurements confirm perceived beliefs about the relative security of the different versions. The measurement methods, however, are based on intuition and are hard to replicate. Hence Manadhata and Wing introduced the attack surface metric to measure a system's attack surface in a systematic manner [7]. Their attack surface measurement method does not rely on the history of attacks on a system; it measures a system's attack surface in terms of the system's inherent attributes and does not require the identification of attack vectors.

6. SUMMARY AND FUTURE WORK

In this paper, we have used the attack surface metric to compare the security of two open source FTP daemons. Our results indicate that software consumers can use the attack surface metric to compare the security of similar systems. In practice, it is the process of measuring the attack surface that is as or more telling than the actual measurements; the process reveals how certain design decisions impact a system's security. In the future, we plan to measure the attack surfaces of two popular open source database servers.

7. ACKNOWLEDGMENTS

We thank Roy Maxion and the anonymous reviewers for their valuable comments and suggestions.

8. REFERENCES

[1] CERT. Cert advisories. http://www.cert.org/.
[2] GNU cflow. http://www.gnu.org/software/cflow.
[3] D. DaCosta, C. Dahn, S. Mancoridis, and V. Prevelakis. Characterizing the security vulnerability likelihood of software functions. In *Proc. of International Conference on Software Maintenance*, 2003.
[4] M. Howard. Fending off future attacks by reducing attack surface. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncode%/html/secure02132003.asp, 2003.
[5] M. Howard, J. Pincus, and J.M. Wing. Measuring relative attack surfaces,. In *Proc. of Workshop on Advanced Developments in Software and Systems Security*, 2003.
[6] P. Manadhata and J. M. Wing. Measuring a system's attack surface. In *Technical Report CMU-CS-04-102*, 2004.
[7] P. Manadhata and J. M. Wing. An attack surface metric. In *Technical Report CMU-CS-05-155*, 2005.
[8] MITRE. Common vulnerabilities and exposures. http://cve.mitre.org/.
[9] The ProFTPD Project. http://www.proftpd.org/.
[10] The ProFTPD Project. Project goals. http://www.proftpd.org/goals.html.
[11] SecurityFocus. Securityfocus vulnerabilities. http://www.securityfocus.com/vulnerabilities.

APPENDIX

A. *INPUT* AND *OUTPUT* METHODS

Input = {canonicalize_file_name, catgets, confstr, ctermid, ctermid, cuserid, dgettext, dngettext, fgetc, fgetc_unlocked, fgets, fgets_unlocked, fpathconf, fread, fread_unlocked, fscanf, getc, getchar, getchar_unlocked, getc_unlocked, get_current_dir_name, getcwd, getdelim, getdelim, __getdelim, getdents, getenv, gethostbyaddr, gethostbyname, gethostbyname2, gethostent, gethostid, getline, getline, getlogin, getlogin_r, getmsg, getopt, _getopt_internal, getopt_long, getopt_long_only, getpass, getpmsg, gets, gettext, getw, getwd, ngettext, pathconf, pread, pread64, ptsname, ptsname_r, read, readdir, readlink, readv, realpath, recv, recv_from, recvmesg, scanf, __secure_getenv, signal, sysconf, ttyname, ttyname_r, vfscanf, vscanf}

Output = {dprintf, fprintf, fputc, fputchar_unlocked, fputc_unlocked, fputs, fputs_unlocked, fwrite, fwrite_unlocked, perror, printf, psignal, putc, putchar, putc_unlocked, putenv, putmsg, putpmsg, puts, putw, pwrite, pwrite64, send, sendmsg, sendto, setenv, sethostid, setlogin, ungetc, vdprintf, vfprintf, vsyslog, write, writev}