An Approach to Measuring A System's Attack Surface

Pratyusa K. Manadhata Kymie M. C. Tan
Roy A. Maxion Jeannette M. Wing

August 2007
CMU-CS-07-146

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This TR supersedes CMU-CS-05-155, "An Attack Surface Metric".

This research was sponsored by the US Army Research Office (ARO) under contract no. DAAD190210389, SAP Labs, LLC under award no. 1010751, and the Software Engineering Institute. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Attack Surface, Attack Surface Metric, Parameter Sensitivity Analysis, User Survey, Security Bulletins

Abstract

Practical software security measurements and metrics are critical to the improvement of software security. We propose a metric to determine whether one software system is more secure than another similar system with respect to their *attack surface*. We use a system's attack surface measurement as an indicator of the system's security; the larger the attack surface, the more insecure the system. We measure a system's attack surface in terms of three kinds of *resources* used in attacks on the system: methods, channels, and data. We demonstrate the use of our *attack surface metric* by measuring the attack surfaces of two open source IMAP servers and two FTP daemons. We validated the attack surface metric by conducting an expert user survey and by performing statistical analysis of Microsoft Security Bulletins. Our metric can be used as a tool by software developers in the software development process and by software consumers in their decision making process.

1 Introduction

Measurement of security, both qualitatively and quantitatively, has been a long standing challenge to the research community and is of practical import to software industry today [6, 23, 34]. Software industry has responded to demands for improvement in software security by increasing effort for creating "more secure" products and services. How can industry determine whether this effort is paying off and how can consumers determine whether industry's effort has made a difference? Our work looks at an important question faced by both industry and consumers today: How can we quantify a software system's security?

In this paper, we propose to use the measure of a software system's attack surface as an indicator of the system's security. Intuitively, a system's attack surface is the set of ways in which an adversary can enter the system and potentially cause damage. Hence the larger the attack surface, the more insecure the system. We intend our attack surface metric to be used in a *relative* manner to compare the security of similar systems, i.e., different versions of the same system or different systems with similar functionality (e.g., different IMAP servers).

1.1 Background And Motivation

Our work on attack surface measurement is inspired by Michael Howard's Relative Attack Surface Quotient (RASQ) measurements [13]. Michael Howard informally introduced the notion of attack surface for the Windows operating system and Howard, Pincus, and Wing measured the attack surfaces of seven versions of Windows [16]. We later measured the attack surfaces of four different versions of Linux [20]. The results of both the Linux and Windows measurements confirm perceived beliefs about the relative security of the different versions. While it is very difficult to devise security metrics that definitively measure the security of software [2, 24], the Windows and Linux measurement results show that the attack surface measurement method holds promise. The Windows and Linux measurement methods, however, were ad-hoc in nature. In this paper, we formalize the notion of a system's attack surface and propose a method to measure a system's attack surface systematically .

We envision our attack surface metric to be useful to both industry and consumers. Software designers and developers can use our attack surface metric as a tool in the software development process; they can measure their system's attack surface periodically during the software development phase, and compare the results with previous measurements. They should strive toward reducing their system's attack surface from one version to another to mitigate the security risk of their system. Software consumers can also use our metric in their decision making process to compare and differentiate between alternative and competing software systems. For example, system administrators often make a choice between different available web servers, IMAP servers, or FTP servers for their organization. Though several factors such as ease of installation and use are relevant to the selection, security is a quality that is of heightened interest to system administrators today. Hence they can compare the attack surface measurements of alternative software in choosing one for their organization.

1.2 Attack Surface Metric

We know from the past that many attacks, 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 (e.g., files). 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* and thus define a system's attack surface in terms of the system's resources.

Not all resources, however, are part of the attack surface. A system's attack surface is the subset of the system's resources that an attacker can use to cause damage to the system. We introduce an *entry point and exit point framework* to identify these relevant resources. Moreover, not all resources contribute equally to the measure of a system's attack surface. A resource's contribution to the attack surface reflects the likelihood of the resource being used in attacks. For example, a method running with `root` privilege is more likely to be used in attacks than a method running with `non-root` privilege. We introduce the notion of a *damage potential-effort ratio* to estimate a resource's contribution to the attack surface. A system's attack surface measurement is the total contribution of the resources along the methods, channels, and data dimensions; the measurement indicates the level of damage an attacker can potentially cause to the system and the effort required for the attacker to cause such damage. Given two systems, we compare their attack surface measurements to indicate, along each of the three dimensions, whether one is more secure than the other.

A system's attack surface measurement does not represent code quality; hence a large attack surface measurement does not imply that a system has many vulnerabilities and few vulnerabilities in a system does not imply a small measurement. Instead, a larger attack surface measurement indicates that an attacker is likely to exploit the vulnerabilities present in the system with less effort and cause more damage to the system. Since a system's code is likely to contain vulnerabilities, it is prudent to choose a system with a smaller attack surface measurement in order to mitigate security risk. Also notice that our measurements are with respect to only attacks that require the attacker to either send (receive) data into (from) the system. For example, our attack surface metric does not cover side channel attacks. The 2001 Workshop on Information-Security-System Rating and Ranking observed that there will be no successful single security metric that can be used to quantify the security of a system and multiple metrics will most certainly be used [34]. The attack surface metric can be used as one of such multiple metrics.

1.3 Contribution and Roadmap

We make the following key contributions in this paper.

1. We outline a method to measure a system's attack surface systematically and demonstrate the use of our method by measuring the attack surfaces of two popular IMAP servers and two FTP daemons.
2. We perform a careful study of the impact of the parameter values on our method and provide guidelines to the users for choosing appropriate parameter values.

3. We validate the steps in our method by conducting a survey of twenty expert system administrators and by performing statistical analysis of Microsoft Security Bulletins published over a period of two years.

The rest of the paper is organized as follows. We present the definition of a system's attack surface in Section 2 and present our measurement method in Section 3. We apply our method to two IMAP servers and two FTP daemons in Section 4. We perform parameter sensitivity analysis of our method in Section 5 and discuss our validation approach in Section 6. We compare our work with previous and related work in Section 7. We conclude with a discussion of future work in Section 8.

2 Attack Surface Definition

We use the entry point and exit point framework to identify the resources that are part of a system's attack surface. Informally, *entry points* of a system are the ways through which data "enters" into the system from its environment, and *exit points* are the ways through which data "exits" from the system to its environment. The entry points and exit points of a system act as the basis for attacks on the system. Our technical report contains a formal description of the entry point and exit point framework [21].

Consider a set, S, of systems, a user, U, and a data store, D. For a given system, $s \in S$, we define its environment, $E_s = \langle U, D, T \rangle$, to be a three-tuple where U is the user, D is the data store, and $T = S \setminus \{s\}$, is the set of systems excluding s. Every system in S has a set of methods. A method of a system receives arguments as input and returns results as output. Examples of methods are the API of a system. Every system also has a set of communication channels. The channels of a system s are the means by which the user U or any other system in the environment communicates with s. Examples of channels are TCP/UDP sockets, RPC end points, and named pipes. The user U and the data store D are global with respect to the systems in S. The data store is a collection of *data items*. Examples of data items are strings, URLs, files, and cookies. We model the data store D as a separate entity to allow sharing of data among all the systems in S. For simplicity, we assume only one user U is present in the environment. U represents the adversary who attacks the systems in S.

2.1 Entry Points

The methods of a system that receive data items from the system's environment are the system's entry points. A method of a system can receive data directly or indirectly from the environment. A method, m, of a system, s, receives a data items *directly* if either (a) the user U or a system, s_1, in the environment invoke m and passes data items as input to m, or (b) m reads data items from the data store, or (c) m invokes the API of a system s_1 in the environment and receives data items as results returned. A method is a *direct entry point* if it receives data items directly from the environment. Few examples of the direct entry points of a web server are the methods in the API of the web server, the methods of the web server that read configuration files from the file system, and the methods of the web server that invoke the API of an application server.

A method, m, of s 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 is a *indirect entry point* if it receives data items indirectly from the environment. For example, a method in the API of the web server that receives login information from a user might pass the information to another method in the authentication module; the method in the authentication module is an indirect entry point. The set of entry points of a system is the union of the set of direct entry points and the set of indirect entry points.

2.2 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 of a system can send data directly or indirectly into the environment. A method, m, of a system, s, sends a data items *directly* if either (a) the user U or a system, s_1, in the environment invoke m and receive data items as results returned from m, or (b) m writes data items to the data store, or (c) m invokes the API of a system s_1 in the environment and passes data items as input to s_1's API. A method m of s is a *direct exit point* if m sends data items directly to the environment. A method, m, of s sends data items *indirectly* if either (a) m passes a data item, d, as input to a method, m_1, of s or m_1 receives d as result returned from m , and m_1 passes d directly to s's environment, or (b) m passes a data item, d, as input to a method, m_2, of s or m_2 receives d as result returned from m, and m_2 passes d indirectly to s's environment. A method m of s is a *indirect exit point* if m sends data items indirectly to the environment. The set of exit points of a system is the union of the set of direct exit points and the set of indirect exit points.

2.3 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. An example of a channel of an IMAP server is the TCP socket opened by the IMAP server.

2.4 Untrusted Data Items

The data store D is a collection of persistent and transient data items. The data items that are visible to both a system s and the user U across different executions of s are the persistent data items. Specific examples of persistent data items are files, cookies, database records, and registry entries. The persistent data items are shared between s and U, hence U can use the persistent data items to send (receive) data indirectly into (from) s. Hence the persistent data items act as another basis for attacks on s. An *untrusted data item* of a system s is a persistent data item d such that a direct entry point of s reads d from the data store or a direct exit point of s writes d to the data store. The configuration files of an IMAP server are examples of the IMAP server's untrusted data items.

Notice that the attacker sends (receives) the transient data items directly into (from) s by invoking s's direct entry points (direct exit points). Since the direct entry points (direct exit points) of s act as a basis for attacks on s, we do not consider the transient data items as a basis for attacks on s.

2.5 Attack Surface

The set of entry points and exit points, the set of channels, and the set of untrusted data items are the resources that the attacker can use to either send data into the system or receive data from the system and hence attack the system. Hence given a system, s, and its environment, E_s, s'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 s.

3 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 is misleading as it assumes that all resources contribute equally to the attack surface. In real systems, not all resources contribute equally to the attack surface. 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 contributes higher to the attack surface than m_2.

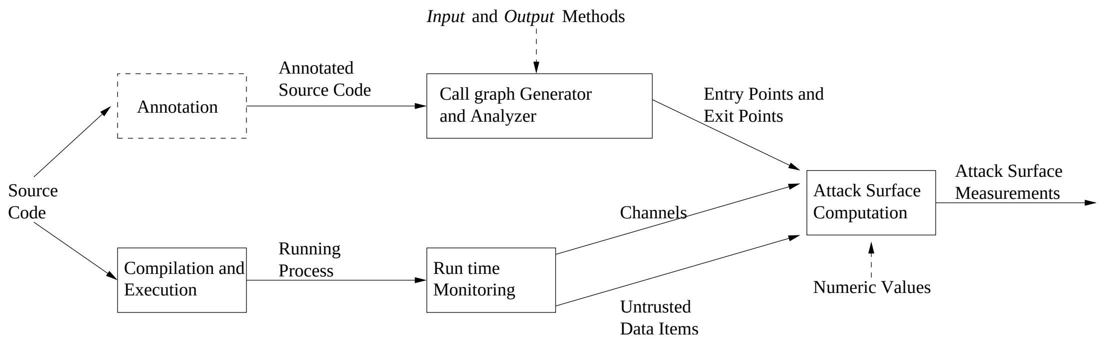
We estimate a resource's contribution to a system's attack surface as a *damage potential-effort ratio* where *damage potential* is the level of harm the attacker can cause to the system in using the resource in an attack and *effort* is the amount of work done by the attacker to acquire the necessary 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.

3.1 Damage Potential-Effort Ratio

In this section, we describe a method of estimating a resource's damage potential and effort in terms of the attributes of the resource. Our estimation method is a specific instantiation of our general measurement framework. Our estimation of damage potential includes only technical impact (e.g., privilege elevation) and not business impact (e.g., monetary loss) though our framework does not preclude this generality. We do not make any assumptions about the attacker's capabilities or resources in estimating damage potential or effort.

Our estimates of damage potential and effort depend on the kind of the resource, i.e., method, channel, or data item. We estimate a method's damage potential in terms of the method's *privilege.* 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`. The attacker can cause damage to the system after gaining `root` 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. The attacker can send executable code into the system by using a `file` in an attack, but the attacker can not do the same using a `registry entry`. 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.,



Input and *Output* Methods
Annotation
Annotated Source Code
Call graph Generator and Analyzer
Entry Points and Exit Points
Source Code
Attack Surface Computation
Attack Surface Measurements
Channels
Compilation and Execution
Running Process
Run time Monitoring
Untrusted Data Items
Numeric Values

Figure 1: Attack surface measurement steps.

methods, channels, 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 4.2.

3.2 Attack Surface Measurement Method

Our attack surface measurement method consists of the following three steps.

1. Given a system, s, and its environment, E_s, 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 $\langle \sum_{m \in M} der_m(m), \sum_{c \in C} der_c(c), \sum_{d \in I} der_d(d) \rangle$.

4 Case Studies

In this section, we describe the process of measuring the attack surfaces of two Internet Message Access Protocol (IMAP) servers and two File Transfer Protocol (FTP) daemons. Figure 1 shows the steps followed in our attack surface measurement method. 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 surface.

Two keys steps in our attack surface measurement method are the identification of relevant resources that are part of the attack surface and the estimation of the damage potential-effort ratio of each such resource. We describe the steps in Section 4.1 and Section 4.2 respectively. We report the IMAP measurement results in Section 4.3 and the FTP measurement results in Section 4.4.

4.1 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.

4.1.1 Entry Points and Exit Points

As proposed by DaCosta et al. [9], 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`. 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 a system that contained a call to a method in *Input* as the direct entry points of the system.

We also 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 a system that contained a call to a method in *Output* as the direct exit points. Please see Appendix A for the *Input* and *Output* sets of methods.

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. We only identified the indirect entry points reachable from the `main` method in both IMAP codebases; we did not identify any indirect entry points of the FTP daemons. 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* 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` [27]. 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.

4.1.2 Channels and Untrusted Data Items

It is difficult to statically determine the channels opened by a system and the data items accessed by the system. Hence we monitored the run time behavior of the default installations of the FTP daemons and the IMAP servers to identify the channels opened by the systems and to determine the protocol and access rights level of each such channel. We similarly used run time monitoring to identify the untrusted data items and to determine the type and access rights level of each untrusted data item. Run time monitoring, however, does not guarantee completeness; we may not be able to identify all possible open channels and untrusted data items. Our approach may produce an under-approximation of a system's attack surface measurement. In case of the IMAP servers and the FTP daemons, the systems opened all their channels as soon as they started running; hence we could identify all open channels. This, however, may not be true for complex software systems.

4.2 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. In order to quantify a resource's damage potential-effort ratio, we assigned numeric values to the resource's attributes. We imposed a total ordering among the values of an attribute and assigned numeric values in accordance to the total ordering. For example, 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. 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. The numeric values also reflect the relative damage an attacker can cause to a system with different privilege levels. For example, given a system and its environment, if we believe that an attacker can cause thrice as much damage in privilege level p_1 compared to p_2, then the numeric value assigned to p_1 is thrice the numeric value assigned to p_2.

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 the FTP daemons, the IMAP servers, and UNIX security. We, however, performed a sensitivity analysis of the effects of the range of the values on our method; we describe the analysis in details in Section 5. 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.

4.3 IMAP Measurement Results

We measured the attack surfaces of two open source IMAP servers: Courier-IMAP 4.0.1 and Cyrus 2.2.10. Our choice of the IMAP servers was guided by two factors: popularity and the availability of source code. Courier-IMAP server is the IMAP server included in the Courier mail server [17]. The Cyrus IMAP server was implemented and is maintained by Project Cyrus [7]. We considered only the code specific to the IMAP daemon in our attack surface measurements. The Courier code base contains nearly 33K lines of C code specific to the IMAP daemon, and the Cyrus code base contains nearly 34K lines of C code specific to the IMAP daemon.

Courier				
Privilege	Access Rights	DEP	DExP	IEP
`root`	`unauthenticated`	28	17	11
`root`	`authenticated`	21	10	0
`auth`	`authenticated`	113	28	1
Cyrus				
Privilege	Access Rights	DEP	DExP	IEP
`cyrus`	`unauthenticated`	16	17	7
`cyrus`	`authenticated`	12	21	2
`cyrus`	`admin`	13	22	2
`cyrus`	`anonymous`	12	21	2

Table 1: The number of entry points and exit points of the IMAP servers.

Courier		
Type	Access Rights	Count
`TCP`	`remote unauth`	1
`SSL`	`remote unauth`	1
`UNIX socket`	`local auth`	1
Cyrus		
Type	Access Rights	Count
`TCP`	`remote unauth`	2
`SSL`	`remote unauth`	1
`UNIX socket`	`local auth`	1

Table 2: The number of channels opened by the IMAP servers.

4.3.1 Entry Points and Exit Points

All the methods in the Cyrus codebase run with a special UNIX user, `cyrus`, privilege. The methods are accessible with `admin`, `authenticated` user, `unauthenticated` user, and `anonymous` user access rights. The methods in the Courier codebase run with `root` and `authenticated` user privileges. The methods are accessible with `authenticated` user and `unauthenticated` user access rights. The Courier codebase does not support `admin` user and `anonymous` user. We show the number of direct entry points (DEP), direct exit points (DExP), and indirect entry points (IEP) for each privilege level and access rights pair in Table 1.

4.3.2 Channels

Both IMAP daemons open a `TCP` channel on port 143 and a `SSL` channel on port 993 to listen to user requests. In addition, the Cyrus daemon opens a `TCP` channel on port 2000 for users to edit their `sieve` filters. These channels are accessible with `remote unauthenticated` user access rights. The Courier IMAP daemon opens a local `UNIX socket` channel to communicate with the authentication daemon. The Cyrus IMAP daemon opens a local `UNIX socket` channel to communicate with the Local Mail Transfer Protocol (LMTP) daemon. These channels are accessible with `local authenticated` user access rights. We show the number of channels for each channel type and access rights pair in Table 2.

4.3.3 Untrusted Data Items

Both IMAP daemons read or wrote persistent data items of `file` type; both daemons used configuration files, authentication files, executable files, libraries, lock files, user mail files, and mail metadata files. The files of the Courier IMAP daemon can be accessed with `root`, `authenticated` user, and `world` access rights. The files of the Cyrus IMAP daemon can be accessed with `root`, `cyrus`, 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 pair in Table 3.

Courier		
Type	Access Rights	Count
`file`	`root`	74
`file`	`authenticated`	13
`file`	`world`	53

Cyrus		
Type	Access Rights	Count
`file`	`root`	50
`file`	`cyrus`	26
`file`	`world`	50

Table 3: The number of untrusted data items accessed by the IMAP servers.

Method Privilege	Value	Access Rights	Value
`root`	5	`admin`	4
`cyrus`	4	`auth`	3
`authenticated`	3	`anonymous`	1
		`unauthenticated`	1
Channel Type	Value	Access Rights	Value
`TCP`	1	`local auth`	4
`SSL`	1	`remote unauth`	1
`UNIX socket`	1		
Data Item Type	Value	Access Rights	Value
`file`	1	`root`	5
		`cyrus`	4
		`authenticated`	3
		`world`	1

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

4.3.4 Estimation of the Damage Potential-Effort Ratio

We assigned the following total ordering among the set of privilege levels: `root` > `cyrus` > `authenticated`. A method running with `cyrus` privilege in the Cyrus IMAP daemon has access to every user's email files, hence we assumed a method running as `cyrus` 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: `admin` > `authenticated` > `anonymous` = `unauthenticated`. `admin` users are special users in Cyrus, hence we assumed the attacker spends higher effort to acquire `admin` access rights compared to `authenticated` access rights. We could not assign a total ordering among the protocols of the channels, hence we assumed that each channel has the same damage potential. We assigned the following total ordering among the access rights levels of the channels: `local authenticated` > `remote unauthenticated`. Both IMAP 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` > `cyrus` > `authenticated` > `world`. The `cyrus` user is a special user, hence we assumed the attacker spends more effort to acquire `cyrus` access rights compared to `authenticated` access rights. We show the numeric values in Table 4.

4.3.5 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 IMAP daemons. From Table 1 and Table 4, the total contribution of the methods of Courier is $(56 \times (\frac{5}{1}) + 31 \times (\frac{5}{3}) + 142 \times (\frac{3}{3})) = 522.00$. From Table 2 and Table 4, the total contribution of the channels of Courier is $(1 \times (\frac{1}{1}) + 1 \times (\frac{1}{1}) + 1 \times (\frac{1}{4})) = 2.25$. From Table 3 and Table 4, the total contribution of the data items of Courier is $(74 \times (\frac{1}{5}) + 13 \times (\frac{1}{3}) + 53 \times (\frac{1}{1})) = 72.13$. Hence the measure of the Courier IMAP daemon's attack surface is the triple $\langle 522.00, 2.25, 72.13\rangle$. Similarly, the measure of the Cyrus IMAP daemon's attack surface is the triple $\langle 383.60, 3.25, 66.50\rangle$. We show the measurement results in Figure 2.

The attack surface metric tells us that the Cyrus IMAP daemon is more secure along the method and data dimension whereas the Courier IMAP daemon is more secure along the channel

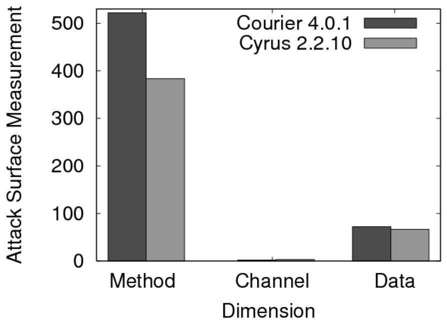


Figure 2: Attack surface measurements of the IMAP servers.

dimension. In order to choose one IMAP daemon over another, we use our knowledge of the IMAP daemons and the operating environment to decide which dimension of the attack surface presents more risk and choose the IMAP daemon that is more secure along that dimension. For example, if we are concerned about privilege elevation on the host running the IMAP daemon, then the method dimension presents more risk, and the attack surface metric suggests that we choose the Cyrus daemon over the Courier daemon. Similarly, if we are concerned about the number of open channels on the host running the IMAP daemon, then the channel dimension presents more risk, and we choose the Courier daemon. If we are concerned about the safety of email files, then the data dimension presents more risk, and we choose the Cyrus daemon.

4.4 FTP Measurement Results

We also measured the attack surfaces of two open source FTP daemons: ProFTPD 1.2.10 and Wu-FTPD 2.6.2 [22]. ProFTPD was implemented and is maintained by the ProFTPD project group [28]. Wu-FTPD was implemented and is maintained at the University of Washington [12]. 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. We briefly describe the measurement results in the following paragraphs.

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 5. Notice that a subset of the methods in the ProFTPD codebase run with a special UNIX user, `proftpd`, privilege.

Both FTP daemons open a `TCP` channel so that FTP clients can communicate with the daemons. These channels are accessible with `remote unauthenticated` user access rights.

Both daemons read or wrote persistent data items of `file` type; both daemons used configuration files, authentication files, executable files, libraries, and log files. We show the number of untrusted data items for each data item type and access rights pair in Table 7.

We show the numeric values assigned to the attributes of the resources in Table 6. Notice that 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 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. The measure of ProFTPD's attack surface is the triple $\langle 312.99, 1.00, 18.90 \rangle$ and the measure of Wu-FTPD's

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 5: The number of entry points and exit points of the FTP daemons.

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

Table 6: Numeric values assigned to the values of the attributes.

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

Table 7: The number of untrusted data items accessed by the FTP daemons.

attack surface is the triple ⟨392.33, 1.00, 17.60⟩. We show the measurement results in Figure 3. 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. Similar to the IMAP daemons, in order to choose one FTP daemon over another, we identify the dimension that presents more risk and choose the daemon that is more secure along that dimension.

5 Parameter Sensitivity Analysis

In our attack surface measurement method, we rely on the domain knowledge of the users of our metric to estimate a resource's damage potential-effort ratio. For example, users of our metric use this knowledge to impose a total ordering among the values of an attribute and then to assign numeric values according to the total ordering. To provide guidelines to the users for choosing appropriate numeric values, we perform parameter sensitivity analysis. In this analysis, we assume that the users have already imposed total orderings among the values of the attributes.

5.1 Method

The attack surface measurement along the method dimension depends on the following three parameters: the number of entry points and exit points, the numeric values assigned to the privilege levels, and the numeric values assigned to the access rights levels. Given two systems, either both systems have comparable numbers of entry points and exit points (e.g., ProFTPD = 107 and Wu-FTPD = 109) or the number of entry points and exit points of one system differs significantly from

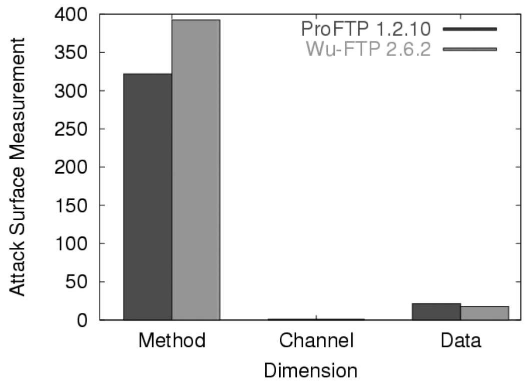


Figure 3: Attack surface measurements of the FTP daemons.

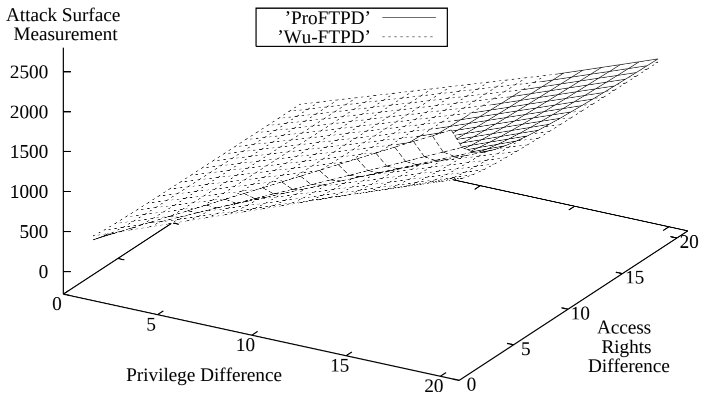


Figure 4: Attack surface measurements of the FTP daemons along the method dimension.

the other system (e.g., Cyrus = 147 and Courier = 239). For both these cases, we analyze the effects of the privilege numeric values and the access rights numeric values on the attack surface measurement.

We study the effects on the measurement as we increase the difference in the numeric values assigned to the attributes. To keep our analysis simple, we assume that the difference, `diff`, in the numeric values assigned to successive privilege and access rights levels is uniform. For example, the difference in the numeric values assigned to `authenticated` and `proftpd` is the same as the difference in the numeric values assigned to `proftpd` and `root`. We assign a fixed numeric value, 3, to the lowest privilege level `authenticated`. We then assign the numeric value (3 + `diff`) to `proftpd` and the numeric value (3 + 2 * `diff`) to `root`. We similarly assign a fixed numeric value, 1, to the lowest access rights level `unauthenticated` and then assign numeric values to the rest of the access rights levels. We observe the effects of changing the value of `diff` from a low value of 1 to a high value of 20.

5.1.1 FTP Measurements Analysis

We show the effects of changing the value of `diff` on the attack surface measurements of the FTP daemons in Figure 4. For example, when the privilege difference is 2 and the access rights difference is 17, ProFTPD's attack surface measurement is 349.7 and Wu-FTPD's attack surface measurement is 444.6. Hence Wu-FTPD has a larger attack surface measurement than ProFTPD. Similarly, when the privilege difference is 17 and the access rights difference is 6, ProFTPD's attack surface measurement is 1785.3 and Wu-FTPD's attack surface measurement is 1672.1. Hence Wu-FTPD has a smaller attack surface measurement than ProFTPD.

We show the attack surface measurements as a projection into a two dimensional plane in Figure 5. In the projection, we only see the FTP daemon that has a larger attack surface measurement. For example, when the privilege difference is 2 and the access rights difference is 17, Wu-FTPD has a larger attack surface than ProFTPD and hence we see Wu-FTPD in the projection. Similarly, when the privilege difference is 17 and the access rights difference is 6, we see ProFTPD in the projection.

From Figure 5, when the privilege difference is low (1-2), Wu-FTPD has a larger attack surface measurement for all possible values of the access rights difference. When the privilege difference

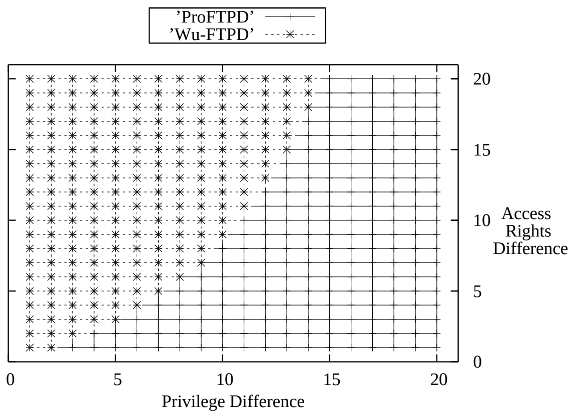


Figure 5: Projection of the measurements of the FTP daemons.

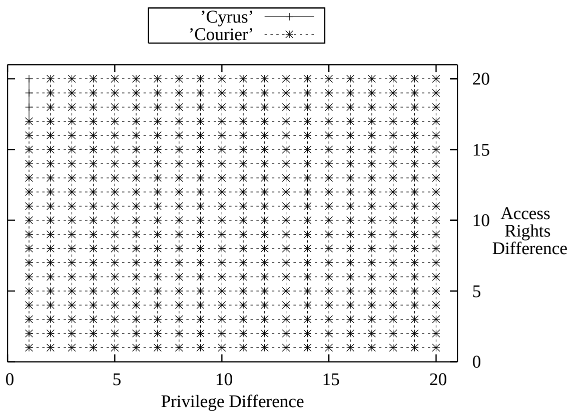


Figure 6: Projection of the measurements of the IMAP servers.

is low, the access rights values do not matter; the number of entry points and exit points is the dominating parameter. Since Wu-FTPD has a larger number of entry points and exit points, it has a larger attack surface measurement.

When the privilege difference is high (15-20), ProFTPD has a larger attack surface measurement for all possible values of the access rights difference. The `proftpd` privilege level contributes more in case of ProFTPD compared to the `authenticated` privilege level of Wu-FTPD. The access rights values do not matter; the privilege values are the dominating parameter.

When the privilege difference is medium (3-14), the access rights values do matter. ProFTPD has a larger number of methods accessible with `authenticated` access rights than the number of methods accessible with `unauthenticated` access rights. Wu-FTPD has a smaller number of methods accessible with `authenticated` access rights than the number of methods accessible with `unauthenticated` access rights. Hence with increasing access rights difference, the methods of ProFTPD make smaller contribution to the attack surface compared to the methods of Wu-FTPD. Hence ProFTPD has a larger measurement for lower access rights difference and Wu-FTPD has a larger measurement for higher access rights difference.

5.1.2 IMAP Measurements Analysis

We show the attack surface measurements of the IMAP servers as a projection into a two dimensional plane in Figure 6. In the projection, we only see the IMAP server that has a higher attack surface measurement. From Figure 6, Courier has a larger attack surface measurement for almost all possible privilege difference and access rights difference. The privilege values and access rights values do not matter; the number of entry points and exit points is the dominating parameter. Courier has a significantly larger number of entry points and exit points than Cyrus; hence Courier has a larger attack surface measurement.

5.1.3 Observations

Our choice of the numeric values should be such that both the privilege values and the access rights values affect the outcome of the attack surface measurements comparison. The FTP measurements analysis shows that if both systems have comparable numbers of entry points and exit points, then the access rights values do not affect the measurements if the privilege difference is low or high. Hence we should choose a medium difference for the privilege values. The IMAP measurement analysis shows that if one system has a significantly larger number of entry points and exit points than the other, then no choice of the privilege difference or the access rights difference affects the measurement. Also, if we choose a medium or a high difference for the privilege values, then we should not choose a low difference for the access rights values; otherwise the privilege values will dominate the access rights values in the damage potential-effort ratio. Hence combining the observations, we suggest that users of our metric choose a medium difference for the privilege values and either a medium or a high difference for access rights values.

5.2 Channel

We performed a similar analysis for the measurements along the channel dimension by changing the difference in the numeric values assigned to the protocols and the access rights levels. ProFTPD and Wu-FTPD open the same set of channels; hence ProFTPD and Wu-FTPD have the same measurements for all possible differences in the protocol values and the access rights values. The set of channels opened by Courier is a subset of the channels opened by Cyrus; hence Cyrus has a larger attack measurement than Courier for all possible differences in the protocol values and the access rights values.

Both the FTP measurements and the IMAP measurements show that similar systems open comparable sets of channels, i.e., either they open the same set of channels or the set of channels opened by one system is a subset of the other. Then we do not need to impose a total ordering and assign numeric values to the attributes; we can determine whether one system has a larger attack surface along the method dimension from the number of channels opened by the systems. If the channels opened by the system, however, are not comparable, then we should follow the recommendations for the method dimension (discussed in Section 5.1.3) to assign numeric values to the protocols and the access rights levels.

5.3 Data

We performed a similar analysis for the measurements along the data dimension by changing the difference in the numeric values assigned to the data types and the access rights levels. ProFTPD has a larger attack surface measurement than Wu-FTPD for all possible differences in the numeric values assigned to the data types and the access rights levels. The number of files accessed by ProFTPD (42) is comparable to the number of files accessed by Wu-FTPD (44). Wu-FTPD has a smaller attack surface measurement because Wu-FTPD has a large number of files accessible with the `root` access rights. These files make the least contribution to the attack surface measurement as we assign the highest numeric value to the `root` access rights level.

Courier has a larger attack surface measurement than Cyrus for all possible differences in the numeric values assigned to the data types and the access rights levels. Cyrus and Courier access comparable numbers of files. The larger attack surface measurement of Courier is due to a larger number of files accessible with the `unauthenticated` access rights. These files make the greatest contribution to the attack surface measurement as we assign the lowest numeric value to the `unauthenticated` access rights level.

Both the FTP and the IMAP measurements show that the systems access data items of only `file` type. Hence assigning numeric values to the data types is trivial. If a system, however, accesses data items of other type, then we should follow the recommendations for the method dimension (discussed in Section 5.1.3) to assign numeric values to the data types. We should also follow the same recommendations for assigning numeric values to the access rights levels.

5.4 Discussion

Recall that our analysis assumed the existence of user-imposed total orderings among the values of the six attributes. Natural total orderings exist among the privilege levels of the methods, the access rights levels of the methods, the access rights levels of the channels, and the access rights levels of the data items. For example, `root` has a higher damage potential than `authenticated` user in UNIX. No such natural orderings, however, exist among the protocols of the channels and the types of the data items. Our analysis of Section 5.2 and 5.3 show that there might not be a need for imposing total orderings among channel protocols and data item types in real systems. We, however, plan to provide guidelines to users for imposing total orderings among the values of these two attributes as part of future work.

6 Validation

A key challenge in security metric research is the validation of the metric. We follow two different approaches to validate the assumptions made in our attack surface measurement method. There are three key hypotheses in our attack surface measurement method.

1. Methods, channels, and data are the right dimensions of a system's attack surface.
2. The damage potential-effort ratio is a good indicator of how likely a resource is going to be used in attacks.
3. The six attributes (the privilege and the access rights of a method, the protocol and the access rights of a channel, and the type and the access rights of a data item) of the resources are good indicators of damage potential and effort.

We validate hypotheses 1, 2, and 3 using an expert user survey discussed in Section 6.1 and hypotheses 1 and 3 using statistical analysis of data collected from the Microsoft Security Bulletins discussed in Section 6.2.

6.1 Expert Survey

We conducted an email survey of experts to validate the steps in our attack surface measurement method [11]. System administrators are potential users of our method; hence we chose experienced Linux system administrators as the participants of our survey.

6.1.1 Subjects

We identified twenty system administrators as the subjects of our survey. We chose the subjects from diverse backgrounds to avoid any bias: fifteen of them work in ten universities, four of them work in four corporations, and one works in a government agency. Nineteen of the subjects are geographically distributed over the US and one is based in Europe. We also chose experienced system administrators who were knowledgeable about software security in order to obtain reliable responses. Six of the subjects have 2-5 years of full time experience of managing Linux systems;

eleven, 5-10 years of experience, and the remaining three, more than 10 years of experience. The subjects have installed and maintained software such as web servers, IMAP servers, and database servers on Linux. The subjects either have implemented or posses the knowledge to implement software attacks such as buffer overflow exploitation, format string exploitation, and cross-site scripting attack.

6.1.2 Questionnaire

The survey questionnaire consists of six explanatory questions. The questions were designed to measure the attitude of the subjects about the steps in our attack surface measurement method. The questions asked the subjects to indicate their degree of agreement or disagreement with the assumptions made in our measurement method on a five point Likert scale [18]. We also asked the subjects to explain the reasons behind their choices and to suggest alternative ways to carry out the steps in our measurement method.

We conducted six rounds of *pretesting* of the questionnaire to identify and remove leading questions, ambiguous terms, and overall confusing questions from the questionnaire [33]. After each round of pretesting, we interviewed the participant and refined our questions using the feedback from the participant.

6.1.3 Results and Discussion

We analyzed the Likert scale responses using descriptive techniques [30]. We combined the strongly agree and the agree responses and the strongly disagree and the disagree responses, and computed the proportion of the subjects who agree (strongly or otherwise), disagree (strongly or otherwise), and are neutral with the steps in our measurement method. We performed one sample t-tests to determine the statistical significance of our findings. We chose a p-value of 0.05 as the threshold; findings with p-values less than 0.05 are statistically significant. We below summarize the findings of the survey.

1. Methods, channels, and data are the right dimensions of the attack surface (Table 8).
2. A resource's damage potential-effort ratio is an indicator of the likelihood of the resource being used in attacks (Table 9).
3. A method's privilege is an indicator of damage potential (Table 10, row 1) and a method's access rights (Table 10, row 2), a channel's access rights (Table 10, row 4) , and a data item's access rights (Table 10, row 6) are indicators of attacker effort.
4. The findings are not conclusive with respect to channel protocol (Table 10, row 3) and data item type (Table 10, row 5).

Our first set of questions probed the subjects about their perception of our choice of methods, channels, and data as the right dimensions of the attack surface. We show the percentage of the subjects who agree, disagree, and neither agree or disagree with our choice in Table 8. The findings show that a majority of the subjects agree with our choice of the dimensions; the p-values show that the findings are statistically significant. We conclude that methods, channels, and data are the right dimensions of the attack surface.

Our second set of questions asked subjects whether the damage potential-effort ratio of a resource is an indicator of the likelihood of the resource being used in attacks. We show the responses

Dimension	A	D	N	p-value
Methods	95%	0%	5%	$p < 0.0001$
Channels	95%	0%	5%	$p < 0.0001$
Data	85%	0%	15%	$p < 0.0001$

Table 8: A majority of the subjects agree with our choice of the dimensions. (A = Agree, D = Disagree, N = Neutral)

	A	D	N	p-value
der	70%	20%	10%	$p = 0.0141$

Table 9: Perception of the subjects about the damage potential-effort ratio (der). (A = Agree, D = Disagree, N = Neutral)

Attribute	A	D	N	p-value
Method Privilege	90%	0%	10%	$p < 0.0001$
Access rights	70%	5%	25%	$p = 0.0001$
Channel Protocol	45%	25%	30%	$p = 0.2967$
Access Rights	75%	5%	20%	$p < 0.0001$
Data Item Type	45%	5%	50%	$p = 0.8252$
Access Rights	85%	10%	5%	$p < 0.0001$

Table 10: Perception of the subjects about the choice of our attributes. (A = Agree, D = Disagree, N = Neutral)

in Table 9. A majority of the subjects agree with our choice of the damage potential-effort ratio and the finding is statistically significant. We conclude that the damage potential-effort ratio of a resource is an indicator of the likelihood of the resource being used in attacks.

Our third set of questions probed the subjects about their perception of our choice of the six attributes of the resources as indicators of the resources' damage potential and effort. We show the percentage of the subjects who agree, disagree, and neither agree or disagree with our choice of the attributes in Table 10. A majority of the subjects agree that a method's privilege is an indicator of the method's damage potential and a method's access rights, a channel's access rights, and a data item's access rights are indicators of the attacker effort. The p-values show that the findings are statistically significant. We conclude that these four attributes are indicators of damage potential and effort.

Though a majority of the subjects agree that channel's protocol is an indicator of the channel's damage potential, the p-value shows that the result is not statically significant. Similarly, though a majority of the subjects disagree that a data item's type is an indicator of damage potential, the finding is not statically significant. Hence the findings of the survey are not conclusive with respect to our choice of a channel's protocol and a data item's type as indicators of damage potential. The subjects who disagreed with our choice were of the opinion that the damage potential of a channel or a data item is dependent on the methods of a system that process the data received from the channel or the data item. A `TCP socket` and an `RPC end point` are equally attractive to an attacker. Similarly, a `file` and a `cookie` are equally attractive an attacker. The findings suggest that perhaps we should assign the same damage potential, i.e., 1, to all channels and data items. In that case, we do not have to perform the difficult step of assigning total orderings among the protocols of the channels and the types of the data items.

6.2 Statistical Analysis of Microsoft Security Bulletins

A Microsoft Security Bulletin describes a vulnerability present in a Microsoft software product [3]. The bulletin includes information on how an attacker may be able to exploit the vulnerability. It mentions the code (method in our framework) of the software the contains the vulnerability and the resources (communication channels, data items, or a combination of both) that the attacker

Resource	Count
Methods	202
Channels	170
Data	108

Table 11: Number of observations that mention methods, channels, and data.

Attribute	Coefficient	Standard Error	p-value
Privilege	0.948	0.236	$p < 0.001$
Access Rights	-0.584	0.110	$p < 0.001$

Table 12: Significance of the privilege and the access rights of the methods.

has to use to exploit the vulnerability. Each bulletin is also assigned a severity rating that reflects Microsoft's assessment of the impact of the exploitation of the vulnerability on the users of the software. There are four levels of severity ratings: Critical, Important, Moderate, and Low.

We collected data from 110 bulletins published over a period of two years from January 2004 to February 2006. From the description contained in each bulletin, we identified the resources (method, channel, and data) that the attacker has to use to exploit the vulnerability. For each such resource, we also identified the values of the attributes of the resource that indicate the resource's damage potential and effort. For each bulletin, we identified the privilege and the access rights of the method, the protocol and the access rights of the channel, the type and the access rights of the data item, and the criticality rating. Many bulletins contained multiple vulnerabilities and many vulnerabilities were assigned different ratings for different versions of Windows. Hence the 110 bulletins resulted in 202 observations.

6.2.1 Validation of Hypothesis 1

Out of the 202 observations, 202 observations mention methods, 170 observations mention channels, and 108 observations mention data items as the resources used in exploiting the vulnerabilities reported in Microsoft security bulletins (Table 11). These findings suggest that methods, channels, and data are the resources used in attacks on software systems. Hence we conclude that methods, channels, and data are the right dimensions of the attack surface.

6.2.2 Validation of Hypothesis 3

Microsoft Security Bulletins are assigned severity ratings based on the impact of the exploitation on the users of the software and the difficulty of exploitation [4]. In our attack surface measurement method, impact on the user is directly proportional to damage potential and difficulty of exploitation is directly proportional to effort. Hence we expect an indicator of damage potential to be an indicator of the severity rating and to be positively correlated with the severity rating. Similarly, we expect an indicator of effort to be an indicator of the severity rating and to be negatively correlated with the severity rating.

For each of the 202 observations, we assigned numeric values to the attributes following the method described in Section 4.2. We used Ordered Logistic Regression analysis to test for the significance of the attributes in explaining the severity rating [36]. We used the sign of the coefficient of an attribute and the p-value of a two sided z-test to determine if the attribute is an indicator of the severity rating. A positive coefficient indicates a positive correlation between the attribute and the severity rating, and a negative coefficient indicates a negative correlation. A p-value of less than 0.05 indicates the attribute is an indicator of the severity rating. We below summarize our findings that suggest that the six attributes are indicators of damage potential and effort.

Attribute	Coefficient	Standard Error	p-value
SMTP	2.535	0.504	$p < 0.001$
TCP	0.957	0.466	$p = 0.040$
Pipe	0.948	0.574	$p = 0.099$
Access Rights	-0.312	0.109	$p = 0.004$

Table 13: Significance of the protocol and the access rights of the channels.

Attribute	Coefficient	Standard Error	p-value
HTML	-0.651	0.263	$p = 0.013$
DHTML	-0.589	0.437	$p = 0.177$
ActiveX	1.522	0.480	$p = 0.002$
WMF	46.314	2.58e+09	$p = 1.000$
Doc	-1.123	0.462	$p = 0.015$
Access Rights	-0.310	0.078	$p < 0.001$

Table 14: Significance of the type and the access rights of the data items.

1. A method's privilege is an indicator of the severity rating and is positively correlated (Table 12).
2. A channel's protocol (Table 13) and a data item's type (Table 14) are indicators of the severity rating. We could not determine the correlation between these two attributes and the severity rating.
3. A method's access rights (Table 12), a channel's access rights (Table 13), and a data item's access rights (Table 14) are indicators of the severity rating and are negatively correlated.

From Table 12, the positive coefficient of the privilege attribute shows that privilege is positively correlated with the severity rating. The p-value shows that privilege is an indicator of the severity rating. Similarly, access rights is negatively correlated with the severity rating and is an indicator of the severity rating.

In case of channels, we could not impose a total ordering among the protocols. Hence we assigned values to the protocols on a nominal scale [10]. We identified the top three frequently mentioned protocols in the observations and tested for the significance of these three protocols with respect to the other protocols in explaining the severity rating. Since we assigned values on a nominal scale, we could not use the signs of the coefficients to make any conclusions about the correlation of the protocols with the severity rating. We show the results in Table 13. The p-values show that both `SMTP` and `TCP` are significant and `pipe` is not significant in explaining the severity rating. Since two of the three protocols are significant, the finding suggests that protocol is an indicator of the severity rating. Similarly, the negative coefficient of access rights and the p-value show that access rights is negatively correlated with the severity rating and is an indicator of the severity rating.

In case of data items, 108 observations mentioned data items of only the `file` type. Since all these observations have the same numeric value for the data type attribute, our initial logistic regression analysis did not include the data type attribute in the analysis. Hence we identified the file format (e.g., doc and html) of each data item mentioned in the bulletins and assigned numeric values to the file formats on a nominal scale. We identified the top five frequently mentioned file formats in the observations and tested for the the significance of these five formats with respect to the other formats in explaining the severity rating. We show the results in Table 14. The p-values show that `HTML`, `ActiveX`, and `Doc` are significant and `DHTML` and `WMF` are insignificant in explaining the severity rating. We could not use the sign of the coefficients to make any conclusions about the correlation of the data items with the severity rating. The findings suggest that the file format is an indicator of the severity rating and hence data item type is an indicator of the severity rating. Similarly, the negative coefficient of access rights and the p-value show that access rights is negatively correlated with the severity rating and is an indicator of the severity rating.

6.3 Validation of Measurement Results

While we validated the steps in our measurement method, we did not validate specific measurement results (e.g., the FTP measurement results). As part of future work, we plan to validate a system's attack surface measurement by correlating the measurement with real attacks on the system. There is, however, anecdotal evidence suggesting the effectiveness of our metric in assessing relative security of software. Our attack surface measurements show that ProFTPD is more secure than Wu-FTPD along the method dimension. The project goals mentioned on the ProFTPD website validate our measurements [29]. 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.

There is also anecdotal evidence illustrating the effectiveness of attack surface reduction in mitigating security risk [14]. The Sasser worm exploited a buffer overflow vulnerability present in an RPC interface of Windows. The interface was accessible by every one in Windows 2000 and Windows XP. The interface, however, was made to be accessible only by local administrators in Windows Server 2003 as part of the attack surface reduction process. Hence the Sasser worm did not affect Window Server 2003. Similarly, the Zotob worm did not affect Windows XP and Windows Server 2003 because of the raising of the access rights level of an RPC interface as part of the attack surface reduction process.

7 Related Work

We compare our work with prior work on attack surface measurement in Section 7.1 and with previous work on quantitative assessment of security in Section 7.2.

7.1 Attack Surface Measurement

Howard, Pincus, and Wing have measured the attack surfaces of seven versions of Windows [16] and we have measured the attack surfaces of four versions of Linux [20]. A key step in both the measurement method was the identification of *attack vectors (classes)*, i.e., the features of a system often used in attacks on the system (e.g., services running as SYSTEM in Windows) and the assignment of weights to these attack vectors (classes). Howard et al. and we used the history of attacks on a system to identify the attack vectors (classes) and assign weights to them. Both the Windows and Linux measurement method were based on intuition. In our current work, we use the entry point and exit point framework to identify the relevant resources that contribute to a system's attack surface and we use the notion of the damage potential-effort ratio to estimate the weights of each such resource. Hence our attack surface measurement method entirely avoids the need to identify a system's attack vectors (classes).

7.2 Other Security Metrics

Our attack surface metric differs from prior work in two key aspects. First, our attack surface measurement is based on a system's inherent properties and is independent of any vulnerabilities present in the system. Previous work assumes the knowledge of the past and current vulnerabilities present in the system [1, 35, 26, 31]. In contrast, our identification of all entry points and exit points encompasses all past and current vulnerabilities as well as future vulnerabilities not yet discovered or exploited.

Second, prior research on measurement of security has taken an *attacker-centric approach* [26, 31]. In contrast, we take a *system-centric approach.* The attacker-centric approach makes assumption about attacker capabilities and resources whereas the system-centric approach assesses a system's security without reference to or assumptions about attacker capabilities [25]. Our attack surface measurement is based on a system's design and is independent of the attacker's capabilities and behavior; hence our metric can be used as a tool in the software design and development process.

Alves-Foss et al. use the System Vulnerability Index (SVI)—obtained by evaluating factors such as system characteristics, potentially neglectful acts, and potentially malevolent acts—as a measure of a system's vulnerability [1]. Alves-Foss et al., however, identify only the relevant factors of operating systems; their focus is on operating systems and not individual software applications such as IMAP servers. Moreover, we may not always be able to quantify the factors that determine a system's SVI.

Littlewood et al. explore the use of probabilistic methods used in traditional reliability analysis in assessing the operational security of a system [19]. In their conceptual framework, they propose to use the effort made by an attacker to breach a system as an appropriate measure of the system's security. They, however, do not propose a concrete method to estimate the attacker effort.

Voas et al. propose a relative security metric based on the fault injection technique [35]. They propose a Minimum-Time-To-Intrusion (MTTI) metric based on the predicted period of time before any simulated intrusion can take place. The MTTI value, however, depends on the threat classes simulated and the intrusion classes observed. Moreover, the MTTI computation requires the knowledge of system vulnerabilities.

Ortalo et al. model a system's known vulnerabilities as a privilege graph [8] and combine assumptions about the attacker's behavior with the privilege graphs to obtain attack state graphs [26]. They analyze the attack state graphs using Markov techniques to estimate the effort an attacker might spend to exploit the vulnerabilities; the estimated effort is a measure of the system's security. Their technique, however, requires the knowledge of the vulnerabilities present in the system and the attacker's behavior. Moreover, their approach focuses on assessing the operational security of operating systems and not individual software applications.

Schneier uses attack trees to model the different ways in which a system can be attacked [31]. Given an attacker goal, Schneier constructs an attack tree to identify the different ways in which the goal can be satisfied and to determine the cost to the attacker in satisfying the goal. The estimated cost is a measure of the system's security. Construction of an attack tree, however, requires the knowledge of the following three factors: system vulnerabilities, possible attacker goals, and the attacker behavior.

8 Summary and Future Work

In summary, we propose an attack surface metric that can be used by system designers to mitigate the security risk of their systems and by software consumers to compare alternative software systems. The results of our expert survey show that a majority of the subjects agree with the steps in our measurement method. The results of the statistical analysis of Microsoft Security Bulletins confirm our choice of the dimensions of the attack surface and the choice of the six attributes as indicators of damage potential and effort. Our parameter sensitivity analysis provides a set of guidelines to users of our metric for assigning appropriate numeric values to the six attributes.

Attack surface measurement is already used in a regular basis as part of Microsoft's Security Development Lifecycle [15]. Mu Security's Mu-4000 Security Analyzer also uses the attack surface framework for security analysis [32]. In the future, we plan to extend our work in three directions. First, we are collaborating with SAP to apply our method to an industrial-sized software system [5]. Second, we plan to develop suitable techniques for validating specific measurement results. Third, we plan to extend our method so that we can approximate a system's attack surface measurement in the absence of source code.

We view our work as a first step towards a meaningful and practical security metric. There is a pressing need for practical security metrics today; we hope that our work will rekindle interest in security metric research. We believe that our understanding over time would lead us to more meaningful and useful quantitative security metrics.

References

[1] J. Alves-Foss and S. Barbosa. Assessing computer security vulnerability. *ACM SIGOPS Operating Systems Review*, 29(3):3–13, 1995.

[2] S. M. Bellovin. On the brittleness of software and the infeasibility of security metrics. *IEEE Security and Privacy*, 04(4):96, 2006.

[3] Microsoft Corporation. Microsoft security bulletin search. `http://www.microsoft.com/technet/security/current.aspx`.

[4] Microsoft Corporation. Microsoft security response center security bulletin severity rating system. `http://www.microsoft.com/technet/security/bulletin/rating.mspx`.

[5] SAP Corporation. SAP - business software solutions applications and services. `http://www.sap.com/`.

[6] Computing Research Association (CRA). Four grand challenges in trustworthy computing. `http://www.cra.org/reports/trustworthy.computing.pdf`, November 2003.

[7] Project Cyrus. Cyrus IMAP server. `http://asg.web.cmu.edu/cyrus/imapd/`.

[8] M. Dacier and Y. Deswarte. Privilege graph: An extension to the typed access matrix model. In *Proc. of European Symposium on Research in Computer Security*, 1994.

[9] 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.

[10] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., Boston, MA, USA, 1998.

[11] A. Fink and J. Kosecoff. *How to Conduct Surveys: A Step-By-Step Guide*. Sage Publications, Beverly Hills, CA, USA, 1985.

[12] The WU-FTPD Development Group. Wu-ftpd. `http://www.wu-ftpd.org/`.

[13] 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.

[14] M. Howard. Personal communication, 2005.

[15] M. Howard and S. Lipner. *The Security Development Lifecycle*. Microsoft Press, 2006.

[16] 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.

[17] Double Precision Inc. Courier-IMAP sever. `http://www.courier-mta.org/imap/`.

[18] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):5–55, June 1932.

[19] B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, S. Page, D. Wright, J. Dobson J. McDermid, and D. Gollman. Towards operational measures of computer security. *Journal of Computer Security*, 2(2/3):211–230, 1993.

[20] P. Manadhata and J. M. Wing. Measuring a system's attack surface. In *Technical Report CMU-CS-04-102*, 2004.

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

[22] P. K. Manadhata, J. M. Wing, M. A. Flynn, and M. A. McQueen. Measuring the attack surfaces of two FTP daemons. In *ACM CCS Workshop on Quality of Protection*, October 2006.

[23] G. McGraw. From the ground up: The DIMACS software security workshop. *IEEE Security and Privacy*, 1(2):59–66, 2003.

[24] J. McHugh. Quality of protection: Measuring the unmeasurable? Invited Talk at the ACM CCS Workshop on Quality of Protection, October 2006.

[25] D. M. Nicol. Modeling and simulation in security evaluation. *IEEE Security and Privacy*, 3(5):71–74, 2005.

[26] R. Ortalo, Y. Deswarte, and M. Kaâniche. Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Transactions on Software Engineering*, 25(5):633–650, 1999.

[27] S. Poznyakoff. GNU cflow. `http://www.gnu.org/software/cflow`.

[28] The ProFTPD Project. The ProFTPD project home. `http://www.proftpd.org/`.

[29] The ProFTPD Project. Project goals. `http://www.proftpd.org/goals.html`.

[30] P. H. Rossi, J. D. Wright, and A. B. Anderson, editors. *Handbook of Survey Research.* The Academic Press, New York, NY, USA, 1983.

[31] B. Schneier. Attack trees: Modeling security threats. *Dr. Dobb's Journal*, 1999.

[32] Mu Security. What is a security analyzer. `http://www.musecurity.com/solutions/overview/security.html`.

[33] W. R. Shadish, T. D. Cook, and D. T. Campbell. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference.* Houghton Mifflin Company, Boston, MA, 2001.

[34] R. B. Vaughn, R. R. Henning, and A. Siraj. Information assurance measures and metrics - state of practice and proposed taxonomy. In *Proc. of Hawaii International Conference on System Sciences*, 2003.

[35] J. Voas, A. Ghosh, G. McGraw, F. Charron, and K. Miller. Defining an adaptive software security metric from a dynamic software failure tolerance measure. In *Proc. of Annual Conference on Computer Assurance*, 1996.

[36] J. M. Wooldridge. *Econometric Analysis of Cross Section and Panel Data.* The MIT Press, Cambridge, MA, USA, 2002.

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}