M.I.D.A.S.

METRICS IDENTIFICATION OF ATTACK SURFACES.

A THESIS

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

MASTER OF SCIENCE IN COMPUTER SCIENCE

BY

JOSHUA A. MEEK

WAYNE ZAGE

BALL STATE UNIVERSITY

MUNCIE, INDIANA

MAY 2012

Table of Contents

INTRODUCTION

Over the last several decades, computing has grown from a small, expensive hobby to an integral part of everyday life. Nearly every aspect of our daily lives is governed, regulated, or made easier through the use of software. While this has proven to generally be a great benefit, it is not without dangers. Relying heavily on software means that in many cases it has become the weak point in the chain, the easiest thing to attack to gain access to private data and information. Today, keeping our software secure from attackers has become a growing concern due to this situation.

Security, however, is very difficult to achieve, especially now that much of the software in use daily is distributed solely over the Internet and is rapidly updated. This means that most testing time is spent looking for software defects, and oftentimes security testing is neglected or not completed at all. Security testing can be more difficult and time consuming than functional testing, and for the average end consumer, the relative secureness of a piece of software is a difficult concept to grasp, while defects in software are easier to observe.

One method of conceptualizing the vulnerability of an application is through its attack surface. A system's attack surface is "the set of ways in which an adversary can enter the system and potentially cause damage" [12]. So, the smaller an attack surface,

the safer from harm and more secure the system. The channels that facilitate an attack include system entry and exit points (method calls), input strings, and data items (files). These channels are all considered attack resources that affect the size of an attack surface. However, not every resource is a part of the attack surface, only those that can be used in an attack are included.

Attack surface measurements are helpful in defining the security of a piece of software, but the attack surface is normally something that can't be truly measured until an application is completed and running. At that point, actually addressing the issues found during the attack surface measuring is very challenging. It often involves redoing large portions of code, and would also require a patch to be pushed out to software users, many of whom would never install it anyway. Due to the difficulty of addressing the attack surface of a program after the program has been written, it would be very beneficial to have a method to gauge the probable value of a program's attack surface while still in development, and even during the design phase, before any code has actually been written.

Design Metrics is a growing field of study in which a software design constructed as a UML model is analyzed and measured to gain insight into some property of the software. The number of different design metrics is large, but all are measurements of data such as information flow, size, complexity, cohesion, or coupling. Since design metrics give us meaningful information about software before any code has been written, it would be beneficial for a particular design metric to be found closely linked with attack surface measurements. Instead of waiting until a program is completely written, a developer could estimate his program's future attack surface and make changes in the

design phase to improve it. Upon first release, software could be much more secure than previously possible.

Thus, in this thesis a strong correlation is sought between several different design metrics and the attack surface for several open source java applications to determine which is the best suitable as an early indicator of a program's attack surface.

LITERATURE REVIEW

Quantifying an attack surface has gone through many iterations. One of the earlier methods used attack vectors [6]. An attack vector is a feature in a system that is often used in attacks on that system. Attack vectors can be many different things, such as a file with weak permissions, an unpatched loophole in software execution, a possibility of buffer overflow, etc. The particular set of attack vectors is different for each system analyzed.

In this early attack vector focused work, the choice of attack vectors was not systematic, and was completely up to the discretion of the individual who was analyzing the system. Typically, a security expert chose appropriate attack vectors to identify all possible attack points and ranked them in order of damage potential. Often not only would a security expert be required, but also an expert on the particular system being analyzed in order to identify it's known security flaws and issues. While this method proved effective for measuring the attack surface of a program, it was clearly not practical, as it could not be completed by an educated security novice, or completed systematically, such as through an analysis program. It required expert knowledge of both security and the program itself.

The next method for measuring an attack surface adopted a qualitative approach. An input/output automaton was developed for a system, which allowed the easy

identification of channels of attack and attackable resources [13]. This method allowed systems to be compared to each other, but was ordinal and not truly quantitative, so it was impossible to determine how much larger one attack surface was than another. For example, you could say that program A had a larger attack surface than program B, thus it was less secure, but you couldn't say that it was twice as large and half as secure. This method was helpful, but lacked the ability for such comparisons.

Finally, a third method built upon the previously mentioned qualitative approach to develop a quantitative measurement of attack surface size. This method still used automata, but added the concept of damage potential to effort ratios for each of the resources in the attack surface [12]. These were arrived at by determining how much damage a breach of a particular resource could cause, compared to how easy it was to compromise that resource.

After an appropriate ratio was calculated for each resource, all the ratios were summed together for each type of resource. This gave us an interval scale measurement, and resulted in an actual number that represented a system's attack surface, with larger numbers indicating a system that was less secure. This made the attack surface measurement a true metric that was easily compared and correlated with other values.

Applying the above technology, Microsoft developed and released their attack surface analyzer tool early last year, which scanned a system and listed the threats to a program's attack surface [1]. The tool first scanned the system to achieve a baseline. Next, the user installed and executed the program being analyzed. The analyzer tool scanned the system again. The difference between the second scan and the baseline scan was the attack surface of the installed program.

RESEARCH OBJECTIVES

The objective of this research is to identify a design metric that best correlates to attack surface size. Finding a design metric that strongly correlates will allow developers to predict their software's future attack surface size during the design process, before any code is actually written.

Possessing a metric to predict attack surface size would benefit developers immensely. Rather than waiting until the first release of their software to measure attack surface, they can have an early predictor during design. Predicting the attack surface earlier would allow developers to be more proactive in identifying security issues affecting the attack surface. Changes during design are much simpler than trying to patch already completed software.

In the process of identifying a design metric that is an appropriate predictor of attack surface size, this research explores the appropriateness of each design metric for the task. Several metrics are compared and measured to determine their uniqueness, and what constructs of a particular system affect attack surface size.

MATERIALS AND METHODS

This study focuses solely on Java-based applications. Analyzing the same development language limits external influences affecting attack surface size. For example, this eliminates the possibility that one programming language tends to produce more secure code, or that the development practices of a particular programming language could inflate the source code automatically.

Java Applications

Several open source java applications are selected to be the case studies for this research. These applications are chosen to span a wide range of software applications, such as text editing, playing media, source code analysis, and email. The applications are as follows:

- aTunes version 2.1.0 – A full featured audio player and organizer, very similar to iTunes or Windows Media Player. http://www.atunes.org/
- DavMail version 3.9.8 – A gateway that allows users to use any mail client with a Microsoft Exchange mail server. http://davmail.sourceforge.net/
- jEdit version 4.5.1 – A text editor designed for use by programmers. It features plugin support, many customizable features, and syntax highlighting for over 200 languages. http://www.jedit.org/

- Freemind version 0.9.0 – A mind-mapping tool that is used to track your tasks and to-do lists. http://freemind.sourceforge.net/wiki/index.php/Main_Page
- Sonar version 2.14 – A platform to manage code quality, which checks for violations of coding rules and gives feedback on many aspects of a project. http://www.sonarsource.org/
- Sweet Home 3D version 3.4 – An interior design application that helps you place your furniture in a house with a 2D floor plan and a 3D preview. http://www.sweethome3d.com/index.jsp

First, the source code for each of these applications is downloaded. The source is the basis for the collection of design metrics. Typically, design metrics are collected during software design, and before any actual source code has been written. However, for these applications it is necessary to extract the underlying design by analyzing the source code itself, since design artifacts are not available. Then, from that extracted design, design metrics are collected. The advantage of this approach is that we are certain that the reverse-engineered design is a true representation of the current software.

Extracting Design

In order to extract the design from the source code, it is necessary to reverse engineer a UML diagram for each of the applications from the initial source code. Reverse engineering source code is a process in which a program scans through the source java files, and based upon the interactions and connections contained, a UML model is created. This model represents the design for each particular version of each application, had the designs been followed precisely.

Possessing both a UML model and the finished, compiled application of each project implies that design metrics can be collected from the UML model, while also analyzing the attack surface of the installed program. These two sources derive the basis for our comparisons. In order to create the UML models, MagicDraw 17.0.1 is used to reverse engineer the source code. Once each application is reverse engineered into a UML model, each is exported to an XMI file.

XMI stands for XML metadata interchange, and is a format that is typically used to trade UML models between different modeling tools. XMI is XML based. Opening an XMI file in a text editor presents the XML tags of the model. These XMI files are the input for the SDMetrics design metrics tool. The design metrics tool analyzes and collects the metric values from the XMI files.

Collecting Metrics

After the XMI files are created, metrics are collected using the tool SDMetrics (http://www.sdmetrics.com/). SDMetrics is java based, and reads in a source XMI file, then reports back many different metrics.

Five metrics are chosen based on the belief that what they measure is likely to have a significant impact on an attack surface measurement. The first metric analyzed is NumCls, which simply counts the number of classes in a package. Totaling up all the packages' NumCls values yields the total number of classes in the project as a whole. This metric gives some insight into the size of the application as a whole, though it does not take into account the size of each individual class or the number of operations each class performs.

In order to consider the operations, the metric NumOpsCls is included for consideration. It totals the number of operations in all classes. This metric is a slightly more accurate size representation. The NumOpsCls indicates how many operations a class can perform.

The next metric that examined is R, which gives the number of relationships between all the classes and interfaces in an application. R essentially measures the complexity of the application, and is included to determine if a complex application with many links between classes leads to a larger attack surface.

Finally, two coupling metrics are considered, Ca and Ce. Coupling is the degree to which elements in a design are connected. Ca tracks afferent coupling, or the number of elements outside the given package that depend on the internal classes within the given package. Similary, Ce is efferent coupling, or the number of elements outside the given package that classes inside the package depend on. Both coupling metrics are included to determine if many dependencies lead to a system that is less secure, and thus possesses a larger attack surface.

For each java application's extracted design, the metrics NumCls, NumOpsCls, R, Ca, and Ce are collected from the XMI representation.

Measuring Attack Surface

The Microsoft Attack Surface Analyzer beta version is used to measure the attack surface of the java applications. To calculate the attack surface with the analyzer, a baseline scan of the system is completed. The application to be analyzed is installed and executed. Next, another scan is completed. Finally, the analyzer completes a final report on the attack surface by comparing the differences in the two scans. Any changes to the

system are a direct result of the application installed, thus any changes in the attack surface of the system as a whole are from the installed application. The final report displays the possible security issues the installed application introduces into the system.

The analyzer scans for items such as running processes, open ports, new firewall rules, directories with weak permissions, and other unsecure events or states. The analyzer assigns a severity ranking to the security issues found, which represents the ratio of damage potential to effort value mentioned in the literature review. Security experts could assign different values to better represent the true damage potential for each individual application, but in order to keep each application equal for analysis, the default values are used. After each application is scanned and a report is generated, the issues are totaled to get a measurement of the attack surface for each java application.

The attack analyzer output for Davmail can be seen below, and the analyzer output for the remaining java applications can be found in appendix A.

Microsoft

Attack Surface Report

Report Summary | Security Issues | Attack Surface

Security Issues: Table of Contents

Directories With Weak ACLs

Explain...

Severity: 1

Weak ACL on C:\Program Files\DavMail allows tampering by NT SERVICE\TrustedInstaller.

Description:

The ACL on the directory C:\Program Files\DavMail allows tampering by NT SERVICE\TrustedInstaller.

Details:

Path: C:\Program Files\DavMail
Weak ACLs:

Account	Rights
NT SERVICE\TrustedInstaller (S-1-5-80-956008885-3418522649-1831038044-1853292631-2271478464)	WRITE_OWNER WRITE_DAC FILE_ADD_FILE FILE_ADD_SUBDIRECTORY FILE_DELETE_CHILD FILE_WRITE_ATTRIBUTES FILE_WRITE_EA GENERIC_ALL

Action:

Weak ACL on C:\Program Files\DavMail\lib allows tampering by NT SERVICE\TrustedInstaller.

Description:

The ACL on the directory C:\Program Files\DavMail\lib allows tampering by NT SERVICE\TrustedInstaller.

Details:

Path: C:\Program Files\DavMail\lib
Weak ACLs:

Account	Rights
NT SERVICE\TrustedInstaller (S-1-5-80-956008885-3418522649-1831038044-1853292631-2271478464)	WRITE_OWNER WRITE_DAC FILE_ADD_FILE FILE_ADD_SUBDIRECTORY FILE_DELETE_CHILD FILE_WRITE_ATTRIBUTES FILE_WRITE_EA GENERIC_ALL

Action:

Registry Keys With Weak ACLs

Explain...

Severity: 1

Weak ACL on registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\DavMail allows tampering by BUILTIN\Users.

Description:

The ACL on the registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\DavMail allows tampering by BUILTIN\Users.

Details:

Registry Key: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\DavMail
Sensitive Values: UninstallString
Weak ACLs:

Account	Rights
BUILTIN\Users (S-1-5-32-545)	WRITE_OWNER WRITE_DAC KEY_SET_VALUE

Action:

Weak ACL on registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\davmail.exe allows tampering by BUILTIN\Users.

Description:

The ACL on the registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\davmail.exe allows tampering by BUILTIN\Users.

Details:

Registry Key: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\davmail.exe
Sensitive Values:
Weak ACLs:

Account	Rights
BUILTIN\Users (S-1-5-32-545)	WRITE_OWNER WRITE_DAC KEY_SET_VALUE

Action:

Processes With NX Disabled

Explain...

Severity: 1

davmail.exe (Pid 10156) has the NX setting disabled.

Description:

The process davmail.exe (Pid 10156) was detected with the NX setting disabled.

Action:

javaw.exe (Pid 10180) has the NX setting disabled.

Description:

The process javaw.exe (Pid 10180) was detected with the NX setting disabled.

Action:

Microsoft

Attack Surface Report

Report Summary | Security Issues | Attack Surface

Attack Surface Report: Table of Contents

Service Information

Running Processes

Explain...

New	Total
6	65

Image Name (PID)	Command Line	Account	Process Flags
svchost.exe (1316)	C:\Windows\system32\svchost.exe -k LocalService		NX: Enabled (Linker Version: 9.0.-1) (ASLR)(Uses SafeSEH)(Uses /GS)
wermgr.exe (3200)	"C:\Windows\system32\wermgr.exe" "-queuereporting_s_user" "C:\Users\Josh\AppData\Local\Microsoft\Windows\WER\ReportQueue\NonCritical_Microsoft Window_bd5996727e9ea1acda90841fa2c99a88df4fb9d6_cab_20a4973f"	Josh-BootCamp\Josh	NX: Enabled (Linker Version: 9.0.-1) (ASLR)(Uses SafeSEH)(Uses /GS)
wuauclt.exe (9160)	"C:\Windows\system32\wuauclt.exe"	Josh-BootCamp\Josh	NX: Enabled (Linker Version: 9.0.-1) (ASLR)(Uses SafeSEH)(Uses /GS)
wmpnetwk.exe (9492)	"C:\Program Files\Windows Media Player\wmpnetwk.exe"		NX: Enabled (Linker Version: 9.0.-1) (ASLR)(Uses SafeSEH)(Uses /GS)
davmail.exe (10156)	"C:\Program Files\DavMail\davmail.exe"	Josh-BootCamp\Josh	NX: Disabled (Linker Version: 2.56.-1)
javaw.exe (10180)	"C:\Program Files\Java\jre1.6.0_06\bin\javaw.exe" -Xmx268435456 "-Dsun.net.inetaddr.ttl"="60" -classpath "C:\Users\Josh\AppData\Local\Temp\temp1.jar;C:\Program Files\DavMail\davmail.jar;C:\Program Files\DavMail\lib\activation-1.1.1.jar;C:\Program Files\DavMail\lib\commons-codec-1.3.jar;C:\Program Files\DavMail\lib\commons-collections-3.1.jar;C:\Program Files\DavMail\lib\commons-httpclient-3.1.jar;C:\Program Files\DavMail\lib\commons-logging-1.0.4.jar;C:\Program Files\DavMail\lib\htmlcleaner-2.1.jar;C:\Program Files\DavMail\lib\jackrabbit-webdav-1.4.jar;C:\Program Files\DavMail\lib\jcharset-1.3.jar;C:\Program Files\DavMail\lib\jcifs-1.3.14.jar;C:\Program Files\DavMail\lib\jdom-1.0.jar;C:\Program Files\DavMail\lib\log4j-1.2.16.jar;C:\Program Files\DavMail\lib\mail-1.4.3.jar;C:\Program Files\DavMail\lib\slf4j-api-1.3.1.jar;C:\Program Files\DavMail\lib\slf4j-log4j12-1.3.1.jar;C:\Program Files\DavMail\lib\stax-api-1.0.1.jar;C:\Program Files\DavMail\lib\stax2-api-3.1.1.jar;C:\Program Files\DavMail\lib\swt-3.7-win32-x86.jar;C:\Program Files\DavMail\lib\woodstox-core-asl-4.1.2.jar;C:\Program Files\DavMail\lib\xercesImpl-2.8.1.jar;" davmail.DavGateway	Josh-BootCamp\Josh	NX: Disabled (Linker Version: 7.10.-1) (Uses SafeSEH)(Uses /GS)

Network Information

Ports

Explain...

Type	TCP	UDP
All New Ports (62 total)	18	4
Running as System	0	0
Running as Local Service	0	0
Running as Network Service	0	0
Running as Other	18	4

Port Name	State	Process	Account
5004/UDP -- Unknown Protocol	Unknown	wmpnetwk.exe (PID 9492)	
5005/UDP -- Unknown Protocol	Unknown	wmpnetwk.exe (PID 9492)	
5004/UDP -- Unknown Protocol	Unknown	wmpnetwk.exe (PID 9492)	
5005/UDP -- Unknown Protocol	Unknown	wmpnetwk.exe (PID 9492)	
554/TCP -- Unknown Protocol	Listen	wmpnetwk.exe (PID 9492)	
1025/TCP -- Unknown Protocol	Listen	javaw.exe (PID 10180)	Josh-BootCamp\Josh
1080/TCP -- Unknown Protocol	Listen	javaw.exe (PID 10180)	Josh-BootCamp\Josh
1110/TCP -- Unknown Protocol	Listen	javaw.exe (PID 10180)	Josh-BootCamp\Josh
1143/TCP -- Unknown Protocol	Listen	javaw.exe (PID 10180)	Josh-BootCamp\Josh
1389/TCP -- Unknown Protocol	Listen	javaw.exe (PID 10180)	Josh-BootCamp\Josh
49177/TCP -- Unknown Protocol	Established	svchost.exe (PID 1316)	
49179/TCP -- Unknown Protocol	TimeWait	(PID)	
2869/TCP -- Unknown Protocol	Listen	System (PID 4)	
10243/TCP -- Unknown Protocol	Listen	System (PID 4)	
554/TCP -- Unknown Protocol	Listen	wmpnetwk.exe (PID 9492)	
1025/TCP -- Unknown Protocol	Listen	javaw.exe (PID 10180)	Josh-BootCamp\Josh
1080/TCP -- Unknown Protocol	Listen	javaw.exe (PID 10180)	Josh-BootCamp\Josh
1110/TCP -- Unknown Protocol	Listen	javaw.exe (PID 10180)	Josh-BootCamp\Josh
1143/TCP -- Unknown Protocol	Listen	javaw.exe (PID 10180)	Josh-BootCamp\Josh
1389/TCP -- Unknown Protocol	Listen	javaw.exe (PID 10180)	Josh-BootCamp\Josh
2869/TCP -- Unknown Protocol	Listen	System (PID 4)	
10243/TCP -- Unknown Protocol	Listen	System (PID 4)	

Firewall

Firewall Rules

Explain...

New	Total
2	636

Name	Direction	Protocol	Local Endpoint	Remote Endpoint	Enabled
Java(TM) Platform SE binary	In	UDP	*:*	*:*	true
Java(TM) Platform SE binary	In	TCP	*:*	*:*	true

Analyzing the Data

After collecting the design metrics and the attack surface measurements for each java application, the correlation coefficients are be calculated between each of the design metrics and the attack surface measurement. The correlation measures the relationship between the attack surface measurement and each metric value obtained. These values determine if it is feasible to predict the attack surface measurement from the design and if further research is warranted to analyze other systems.

Correlation coefficients range from -1 to 1. For correlations between two sets of data, a value of 0 means that no correlation whatsoever exists. If you have a value from one set, with a correlation of 0 you can make no assumptions at all about the corresponding value from the other set. If the correlation coefficient is positive, however, it means that as the values of one data set get larger so do the values in the other set. Similarly, if the correlation is negative, as values of one data set get larger, the values in the other data set will get smaller. A correlation coefficient of 1 or -1 means there is a perfect correlation between the two data sets; if the sets were graphed as a scatter plot, connecting the dots would yield a perfectly straight line.

Typically, a correlation coefficient from .85 to 1 or from -.85 to -1 is considered a strong correlation. In this context, it would mean that there is a strong correlation between a given metric and the attack surface size for this data set.

RESULTS

The collected metric values and attack surface measurements for each of the java applications can be seen in Table 1.

Table 1: Design Metrics and Attack Surface for 6 Java Open-Source Programs

	NumCls	NumOpsCls	R	Ca	Ce	Ca + Ce	Attack Surface
Davmail 3.9.8	340	1494	261	491	373	864	33
aTunes 2.1.0	1034	5627	1169	2921	1809	4730	33
Jedit 4.5.1	807	5702	1695	2233	1023	3256	19
Sonar 2.1.4	2464	13780	3581	5123	2737	7860	54
SweetHome 3.4	475	3110	827	1130	593	1723	12
Freemind 0.9.0	641	4371	829	2430	1412	3842	26

The correlation coefficients are calculated for each metric and the attack surface as two data sets, yielding five different coefficients, which can be seen in Table 2.

Table 2: Correlation of the Design Metrics and Attack Surfaces

	Correlation Coefficient
NumCls	.8192
NumOpsCls	.7505
R	.6777
Ca	.7334
Ce	.7718
Ca + Ce	.7491

The first observation is that none of the correlation coefficients are .85 or greater, indicating that the relationships between each metric and attack surface are not strong correlations, and none of the tested metrics are a strong predictor of attack surface size based on the previously stated baseline. Despite not achieving the .85 benchmark, each metric except R is .73 or greater. Therefore, one can conclude that at least there is a positive correlation between the collected design metrics and the attack surface.

Surprisingly, the metric with the highest correlation coefficient and best suited according to correlation value as a predictor of attack surface in these java applications is NumCls, or a simple count of the number of classes in an application. More sophisticated metrics, such as the relationship metric R or the two coupling metrics, actually have less positive relationships. This result seems contrary to initial speculation, as a program's complexity might lead to more areas to attack and vulnerabilities.

T-Test Analysis

A t-test is conducted to determine if the means of two different populations, each of the calculated design metrics with the attack surface measure, are statistically different from each other. The T-tests answer the question whether the statistical relationship

between the design metrics and the attack surface measure is real or just the result of chance. All of the t-tests are significant at the 0.05 level, and Ce and Ca are significant at the 0.01 level. The p-values for each of the t tests are listed in Table 3.

Table 3: T-Test P-Values Pairing Each Collected Metric with the Attack Surface Measurements.

	T-Test P-Values
NumCls	.0154
NumOpsCls	.0114
R	.0173
Ca	.0077
Ce	.0070

Eliminating Outliers

In an effort to obtain a stronger correlation, Ca and Ce are added together for each application, and are treated as a single metric, which can be seen in Tables 1 and 2 above. This is a total of all coupling in each given program. However, combining the two metrics does not yield significant improvement over either metric alone.

To further investigate the findings, the scatter plots of each metric vs. attack surface are examined in Figures 1 through 5.

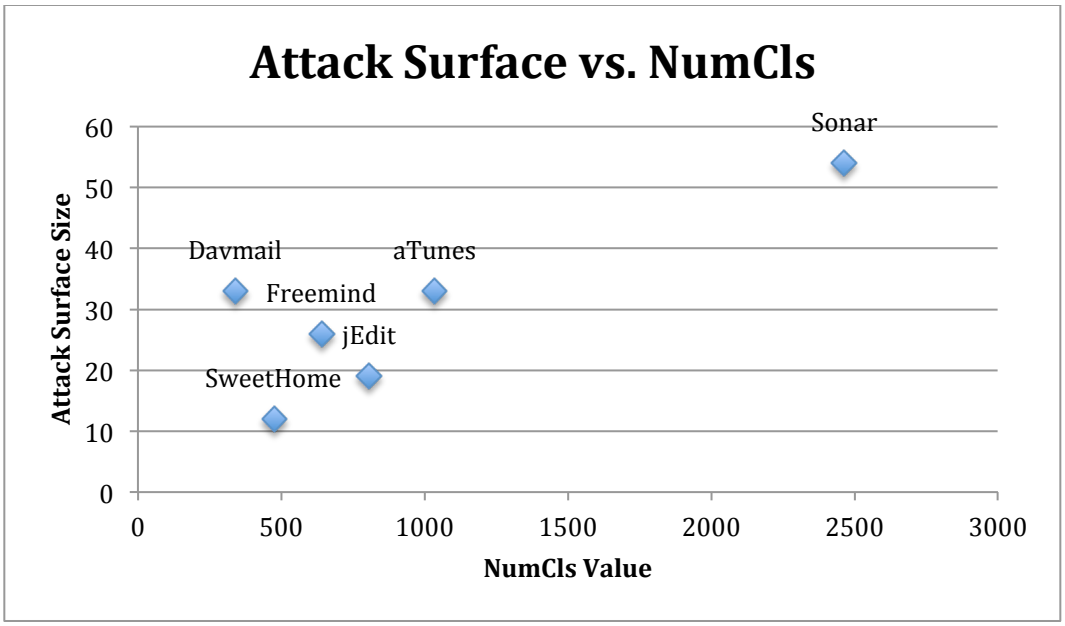

Figure 1: Attack Surface Size versus the Design Metric NumCls Value.

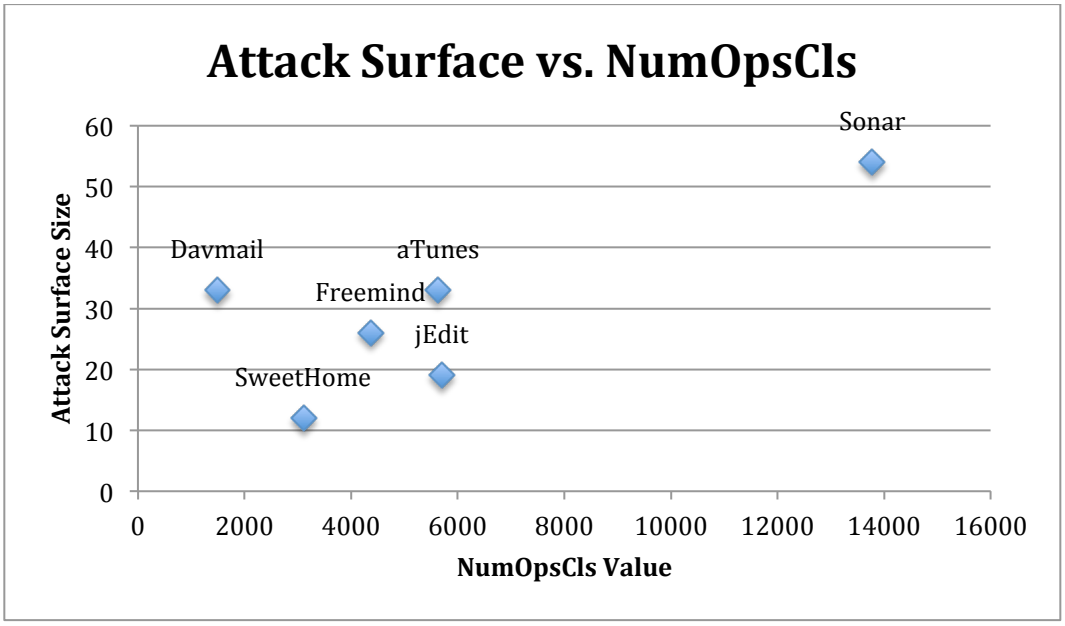

Figure 2: Attack Surface Size versus the Design Metric NumOpsCls Value.

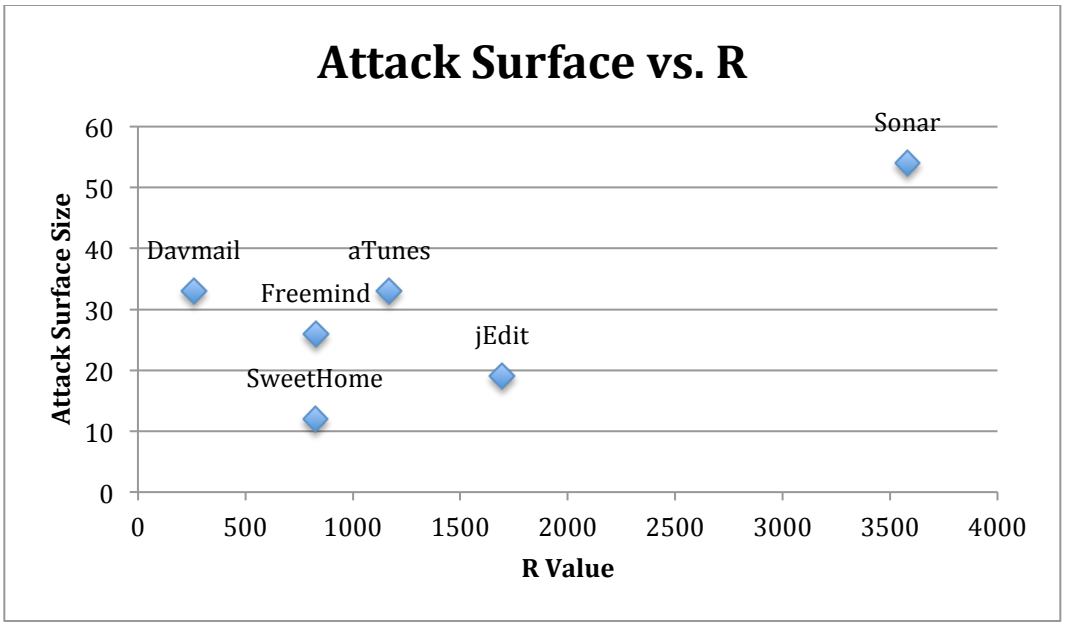

Figure 3: Attack Surface Size versus the Design Metric R Value.

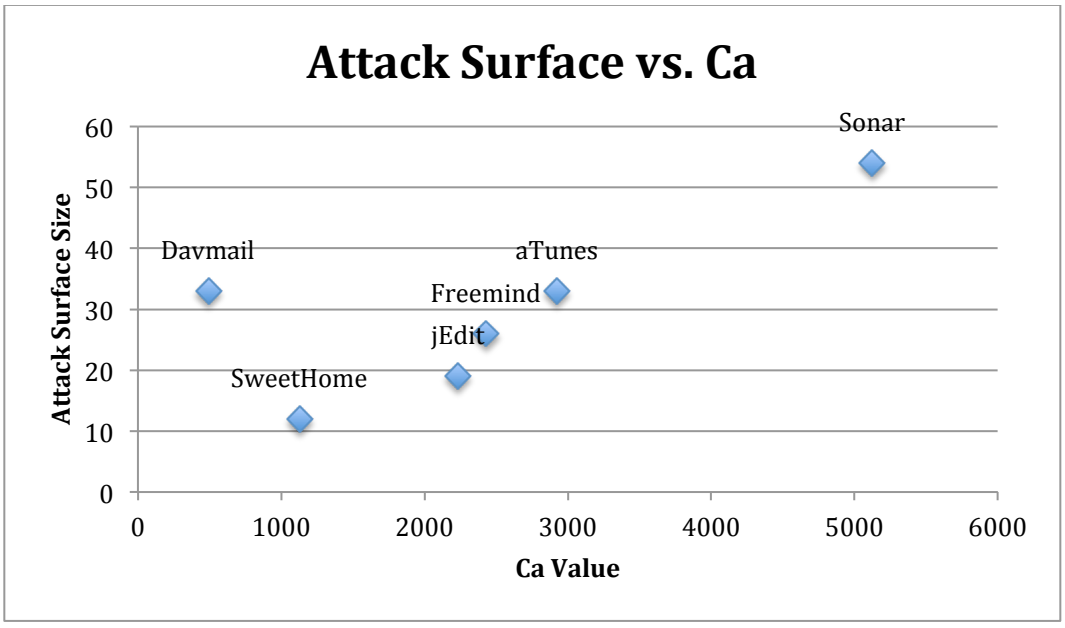

Figure 4: Attack Surface Size versus the Design Metric Ca Value.

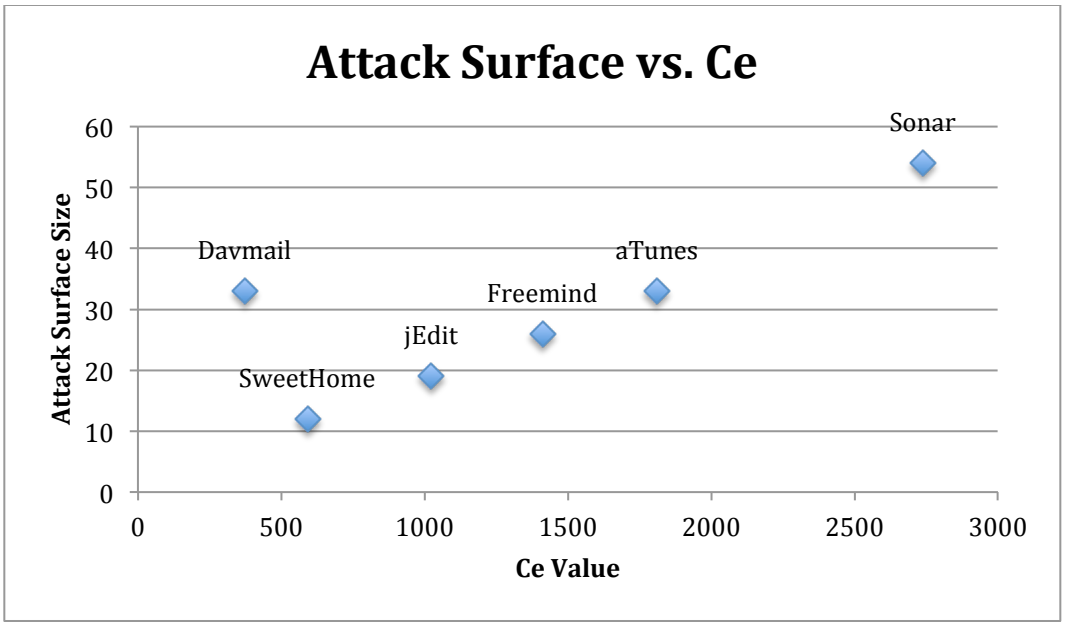

Figure 5: Attack Surface Size versus the Design Metric Ce Value.

Analyzing the scatter charts, specifically those of the coupling metrics Ca and Ce, it becomes very clear that the results obtained from Davmail were outliers. For every analyzed metric, Davmail possessed the lowest metric value, while having a relatively large attack surface of 33. Davmail is examined further to determine the cause of its larger attack surface and perhaps identify additional constructs that can be exposed during design.

Davmail is unlike the other java programs analyzed. Davmail is designed as an interface between an existing Exchange mail server and any mail client. It works as a translator, swapping proprietary Exchange commands for standard compliant protocols. It is a singularly-focused tool that deals solely in network features and requires open ports, special firewall rules and extra running processes, all of which increase the attack surface measurement. At the same time, since it doesn't include a mail client, or a robust user interface, the actual code is relatively small and compact.

Due to its size and efficiency, the analyzed metric values are the lowest of all the programs analyzed. The code does not possess much coupling or many tight, complicated relationships, nor is Davmail composed of an extremely large number of classes. However, due to the nature of it's function, dealing with Internet email protocols and acting as an always running server process, Davmail ranks second highest in attack surface size, compared to the other five programs analyzed.

Unlike the other java applications analyzed, Davmail is a special case. Removing the values obtained from Davmail and reevaluating the correlation coefficients yield much stronger results, as seen in Table 4 below.

Table 4: Correlation of the Design Metrics and Attack Surfaces Excluding Davmail

	Correlation Coefficient Excluding Davmail
NumCls	.9471
NumOpsCls	.9263
R	.8329
Ca	.9883
Ce	.9970
Ca + Ce	.9955

Every metric except for R has a correlation with attack surface that is greater than .85, indicating that the number of classes, the number of operations in classes, afferent coupling, and efferent coupling are all strong predictors of attack surface size.

The strongest correlation is Ce, efferent coupling. It has a .9970 correlation, which is nearly perfect. Recall that efferent coupling is the total number of elements outside a given package that classes inside the package rely on. One explanation as to why efferent coupling ties so closely to attack surface size is that many of the flows that a package relies on come from the external sources such as files, ports, and entities that are

evaluated in attack surface measurements. These packages that rely heavily on outside components operate in a large context, and a developer must understand a wide range of outside services, and how to properly use them. Using one incorrectly, or not properly understanding the consequences of calling it, could easily make the application's attack surface larger.

Another reason that efferent coupling is linked so closely to attack surface is that every time a package calls an external service, it essentially opens an extra avenue of attack into the package itself. While not every time an outside service is relied upon will make the application easier to attack, many will, and the developer is then basing the security of his package on the security of the outside service that he is relying on. Essentially, it only takes one weak link to break a chain, and the higher the efferent coupling, the more chance of adding weak links to the security chain of a program.

One effect of Ce and attack surface being closely linked is that a high Ce value means that a program is much more difficult to maintain. A change in one package might force changes in many others, because of the dependencies inherent in high coupling. This makes a program more difficult to patch. However, with a higher attack surface, the necessity of patches is much more likely in order to improve the software's security.

Thus, having a strong predictor of attack surface, like Ce, is extremely helpful. Rather than waiting to develop and deliver a challenging software patch, during the design phase developers should strive to lower their Ce metric values in order to reduce their future attack surface. This approach would likely lead to a more secure computing world.

FUTURE RESEARCH

This empirical research shows that design metrics are a feasible predictor of attack surface size. In the future, research in this area should be taken in several different directions. First, other languages should be analyzed to determine if these java specific findings still hold true.

Another avenue of research would be to pinpoint offending packages or portions of code. Currently, the selected design metrics are collected for the application as a whole. While these design metrics may be able to predict attack surface measurements, they are not able to specify the area of design that most contributes to attack surface. Having that knowledge would allow developers to more easily adjust their design in order to improve their attack surface measurement.

CONCLUSION

In summary, design metrics appear to be feasible predictors of attack surface size. Considering attack surface and program security during the design process could potentially lead to a smoother development process while yielding more secure applications.

REFERENCES

[1] "Attack Surface Analyzer - Beta." Microsoft. Web. 18 Jan. 2011. <http://www.microsoft.com/download/en/details.aspx?id=19537>.

[2] DaCosta, Dan, Christopher Dahn, Spiros Mancoridis, and Vassilis Prevelakis. "Characterizing the 'Security Vulnerability Likelihood' of Software Functions." Drexel University. Web. 11 Oct. 2011.

[3] Goodman, Seymour E., and Herbert S. Lin, eds. "Toward a Safer and More Secure Cyberspace." Committee on Improving Cybersecurity Research in the United States. National Academic Press. 2007.

[4] Howard, Michael. "Fending Off Future Attacks by Reducing Attack Surface." MSDN. Web. 11 Oct. 2011. <http://msdn.microsoft.com/en-us/library/ms972812>.

[5] Howard, Michael. "Mitigate Security Risks by Minimizing the Code You Expose to Untrusted Users." MSDN Magazine 1 November 2004: 7.

[6] Howard, Michael, Jon Pincus and Jeannette M. Wing. "Measuring Relative Attack Surfaces." Proceedings of Workshop on Advanced Developments in Software and Systems Security (2003).

[7] Kitchenham, Barbara, Pfleeger, Shari Lawrence, & Fenton, Norman. 1995. Towards a Framework for Software Measurement Validation. IEEE Trans. Softw. Eng. 21, 12 (December 1995), 929-944. DOI=10.1109/32.489070 <http://dx.doi.org/10.1109/32.489070>

[8] Leversage, David John, & Byres, Eric James. 2008. Estimating a System's Mean Time-to-Compromise. IEEE Security and Privacy 6, 1 (January 2008), 52-60. DOI=10.1109/MSP.2008.9 <http://dx.doi.org/10.1109/MSP.2008.9>

[9] Lynch, N., M. Tuttle. An introduction to input/output automata. CWI-Quarterly, 2(3):219–246, September 1989.

[10] Madan ,Bharat B., Goseva-Popstojanova, Katerina, Vaidyanathan, Kalyanaraman, and Trivedi, Kishor S. Modeling and quantification of security attributes of soft-ware systems. In DSN, pages 505–514, 2002.

[11] MagicDraw 17.01. *Best UML Tool | Best modeling tool | BPMN 2.0 | DoDAF*. Web. 6 Feb. 2012. https://www.magicdraw.com/

[12] Manadhata, Pratyusa K. An Attack Surface Metric. Pittsburgh: Carnegie Mellon University, 2008.

[13] Manadhata, Pratyusa K., & Wing, Jeannette M. "An Attack Surface Metric." Transactions on Software Engineering (2010): 17.

[14] Manadhata, Pratyusa K., et al. "Measuring the Attack Surfaces of Two FTP Daemons." Computer and Communications Security Workshop on Quality of Protection (2006): 7.

[15] Narang, Mukta, & Mehrotra, Monica. "Security Issue – A Metrics Perspective." International Journal of Information Technology and Knowledge Management 2.2 (2010): 567-571.

[16] Ozment, Andy. "Improving Vulnerability Discovery Models - Problems with Definitions and Assumptions." (2007). MIT Lincoln Laboratory & University of Cambridge.

[17] "Pearson Product-moment Correlation Coefficient." *Wikipedia, the Free Encyclopedia*. Web. 11 Dec. 2011. <http://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient>.

[18] SDMetrics. *SDMetrics - the design quality metrics tool for UML models*. Web. 19 March 2012. http://www.sdmetrics.com/.

[19] Spafford, Eugene H., & DeMillo, Richard A. "Four Grand Challenges in Trustworthy Computing." (2003). Computer Research Association. Web.

[20] "Spearman's Rank Correlation Coefficient." *Wikipedia, the Free Encyclopedia*. Web. 11 Oct. 2011. <http://en.wikipedia.org/wiki/Spearman's_rank_correlation_coefficient>.

[21] Stineburg, J., Zage, W., & Zage, D. “Measuring the Effect of Decisions on Software Reliability”, International Society of Software Reliability Engineers (ISSRE) 2005 Conference, Chicago, November 2005.

[22] Yanguo Liu, Michael. "Quantitative Security Analysis for Service-Oriented Software Architectures." Thesis. University of Victoria, 2003.

[23] Zage, W.M., & Zage, D.M., “Evaluating Design Metrics on Large-ScaleSoftware”, *IEEE Software*, Vol. 10, No. 4, July 1993.

[24] Zage, W.M., & Zage, D.M. “Metrics Directed Verification of UML Designs”, SERC Technical Report 281, January 2006.