uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Brad Metz

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Third Party Security Evaluations of Web-based Systems

TITRE DE LA THÈSE / TITLE OF THESIS

C. Adams

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

G-V. Jourdan

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

A. Matrawy

A. Miri

P. Van Oorschot

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Third Party Security Evaluations of Web-Based Systems

Brad Metz

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc. degree in Electrical Engineering

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-34089-9
Our file Notre référence
ISBN: 978-0-494-34089-9

Abstract

With the increased adoption of web-based technologies, it is important to understand the associated risks with using these systems. In an effort to make web-based systems more secure, the organizations managing those applications are increasingly relying on the expertise of external testing teams to evaluate their systems. In this thesis, we propose a third-party security evaluation methodology for testing web-based systems which we were able to utilize in an actual security evaluation. This methodology was used to perform a third-party evaluation of a production web-based system. The results of the evaluation, problems encountered, and possible mitigation solutions for the system evaluated are also discussed.

Acknowledgments

I would like to thank my thesis supervisors, Prof. Carlisle Adams and Prof. Guy Jourdan, for their help in finding such an enjoyable yet challenging thesis project. I would also like to thank the managers and administrators who allowed me to evaluate their systems as part of my thesis project as well as for their financial support. I hope that this experience was as beneficial to you as it was to me. I would also like to thanks Prof. Matrawy, Prof. Miri, and Prof. Van Oorschot for their valuable feedback and advice.

I would also like to thank all of my friends both at the University of Ottawa and at Carleton University as well as all my friends from back home in Thunder Bay for their support. Special thanks go out to my wonderful girlfriend Robyn who managed to stand by me and support me through all of my highs and lows over that past two years. Also, I would like to thank my parents, Gary and Debra, my brother, Devin, and Nanny (Doreen) for their love and ongoing support. Even though it may not have seemed like it at the time, their interruptions were greatly appreciated as they allowed me to take my mind off of my work when I needed it most. I would also like to thank my uncle Laurie who peaked my interest in information technology and engineering.

I want to dedicate this thesis to my niece, Makayla. You always manage to cheer me up no matter how bad a day I may be having. Remember that your uncle Brad is always there for you. I love all of you guys and thanks.

– Brad Metz

Table of Contents

List of Figures

List of Tables

List of Terms

CERT/CC – Computer Emergency Response Team Coordination Center

CMU – Carnegie Mellon Univeristy

CRUD – Create, Read, Update, Delete

DBMS – Database Management System

DFD – Data Flow Diagram

DoS – Denial of Service

DREAD – An acronym used in the Microsoft threat modeling methodology to measure the risk of a particular threat. It stands for **D**amage potential, **R**eproducibility, **E**xploitability, **A**ffected users, **D**iscoverability.

IDS – Intrusion Detection System

LDAP – Lightweight Directory Access Protocol

MITM – Man-in-the-middle

NASL – Nessus Attack Scripting Language

OCTAVE – Operationally Critical Threat, Asset, and Vulnerability Evaluation

OSJS – Online Student Job Seach

OSSTMM – Open-Source Security Testing Methodology Manual

OWASP – Open Web Application Security Project

Pen-test – Short for penetration testing.

RMI – Remote Method Invocation

RTA – Remote Teaching Assistant

AAP – Academic Assistant Program

SAT – Security Auditing Tool

SDLC – Software Development Life Cycle

SEI – Software Engineering Institute

SOAP – Simple Object Access Protocol

SQL – Structured Query Language

SQL Injection - An attack technique where an attacker attempts to execute unauthorized SQL

commands through some front-end application.

STRIDE – An acronym used in the Microsoft threat modeling methodology to describe threat categories. It stands for **S**poofing identity, **T**ampering with data, **R**epudiation, **I**nformation disclosure, **D**enial of service, **E**levation of privilege.

TOCTTOU – Time-of-check-to-time-of-use

Trike – A threat modeling framework proposed by Paul Saitta *et al.*

U of M – University of Makayla: an alias used to refer to the university whose system we evaluated

WAVES – Web Application Vulnerability and Error Scanner

WWW – World Wide Web

XSS – Cross-Site Scripting

1. Introduction

1.1. Problem Definition

With the increased popularity of web-based systems, more institutions are rushing to get online systems implemented and deployed. This is becoming easier than in the past due to a number of web application development frameworks such as the Microsoft ASP.NET technology suite and Sun Microsystems Java J2EE framework. These technologies have helped to make the web a powerful and dynamic tool for managing one's day-to-day activities. Internet users are now able to perform banking transactions, shop, and publish personal profiles about themselves. However, because many of these systems have been developed over a number of years, sometimes decades, there exists a large amount of legacy code and applications which have been patched into the system which may not have been developed with security in mind. In many cases, web-based systems designed using newer technologies do not take security requirements into consideration either which introduces new vulnerabilities. When systems coded using legacy technologies are combined with systems developed with more modern technologies that often overlook the security requirements of deploying a modern web-based system, the number of possible attack areas within the system increases. In a university setting, a number of services may also be offered by different organizational groups which could introduce new attack vectors since not all the organizations involved will have the same policies in place with respect to protecting information resources. Also, when providing a single sign-on style interface which allows the users to access any of the applications offered on any one of a number of hosts, the attack surface is expanded even further.

Due to the fact that a large number of web-based systems that are used within organizations such as universities exhibit the qualities described above, it is essential that these systems be evaluated in some way in order to uncover as many potential vulnerabilities within the system as possible and to determine a reasonable mitigation for each of those problems. In this work, I have examined an actual system deployed in a university setting in

order to determine possible vulnerabilities within the system which may violate the intended security policy of the university. The identifying information from the actual evaluation has been changed to protect the privacy of the organization but the methodology used, generalized descriptions of the vulnerabilities found, and the mitigation techniques proposed for those vulnerabilities will be discussed. In the following sections, I will give an overview of the system examined and what my goals were in conducting this research.

1.2. System Overview

The system examined throughout the duration of this study is a system run by a university that is used to provide access to a number of resources pertaining to the academic studies offered. The services are written in a variety of programming languages and make use of a number of application frameworks. Because of the variation in application frameworks that are being leveraged as well as the large amount of legacy applications being employed, the services offered by the university are hosted on multiple servers. In this case, all of the applications written in Java are hosted on one server, all of the applications written using legacy technologies are hosted on a second server, and so on. The system, which will be referred to as the Academic Assistant Program or AAP from this point forward, has a number of different user classes which have access to specific resources within the system. The main focus of my evaluation was on the "student" user group which means that I evaluated the system based on what a student user was able to do legitimately and, more importantly, what a student user was able to do illegitimately. A student user in the AAP system represents an actual student enrolled at the university. The focus of the evaluation was on student users because that user group will most likely be the least trusted. In order to maintain the privacy of the actual university involved, we will refer to the university which owns and operates the AAP system as the University of Makayla or U of M for short. That said, students at the U of M can make use of the AAP to check their academic standing in form of their own grades as well as class averages, their financial standing with the university, as well as update information such as their contact information. Students are also able to register for courses as well as drop courses that they are currently enrolled in. This implies that there is a considerable amount of sensitive information stored in the AAP. Not

only does the system store information regarding academic activities but there is also other personal information such as social insurance numbers (SIN) which, if compromised, could lead to identity theft attacks that are well outside the scope of the AAP system but are arguably more dangerous.

Since the various services offered by the AAP are hosted on a number of different servers, the designers of the system found it necessary to implement a single sign-on solution. This would allow the AAP users to login to the system at a single point and have seamless access to all the services that they should have access to. The single sign-on mechanism in this case will allow users to cross inter-organizational trust boundaries as well. This is due to the fact that not all of the services offered within the AAP are managed by the same inter-university organization. Some of the applications, such as the Remote Teaching Assistant (RTA) application and the Online Student Job Search (OSJS) application are run by a different department than the AAP itself. Since the AAP provides access to these outside applications, it stands to reason that information which may be used to access the AAP may be stored on hosts outside the domain managed by the AAP team. Also, when considering the security of the individual hosts in the U of M network, it is not necessary during this evaluation to perform a host configuration analysis on every host in the system. This is due in large part to the fact that the adversary considered, in this case student users of the system, already have access to the internal network and therefore do not need to compromise a host to gain access.

For the purposes of the security evaluation, some resources were provided by the U of M which were used to help make the evaluation of the AAP run more efficiently. Since the AAP has evolved over upwards of 20 years, design documentation is scarce. In most cases, the only artifact that was provided was the source code. No network maps or any other high-level design documents were provided.

1.3. Contributions of this Thesis

In performing the evaluation, I not only provide valuable information regarding perceived vulnerabilities in the AAP system but also solutions to help mitigate the risk associated with those vulnerabilities. Specifically, the contributions that I have made in

writing this thesis are the following:

1) Combine ideas from previous work in the field of penetration testing and threat modeling in order to create a security testing methodology that will allow future security evaluators to efficiently test systems.

2) Provide a detailed description and analysis of the evaluation of a production, web-based system.

3) Examine the vulnerabilities found in the production AAP system from an architectural point of view and provide mitigation solutions for the problems found. This includes a solution for dealing with potentially untrusted hosts that are located within the trusted portion of the system.

4) Provide an overview of previous work done in in the area of session management and provide a list of best practices for session management based on previous work.

The remainder of this thesis will be organized in the following way. Chapter 2 will summarize the necessary background information that the remainder of this work is based on. Chapter 3 will provide a detailed description of the methodology used to analyze the AAP system. In chapter 4, I provide a summary of attacks which target the session management systems of web-based applications and provide a list of best practices based on previous work. Chapter 5 provides a description of the AAP system tested using the methodology proposed in chapter 3 as well as an overview of the flaws found in the system. In chapter 6, mitigation strategies are proposed to help mitigate the problems found during the evaluation. Finally, chapter 7 will conclude the thesis and present considerations for future work in the field of penetration testing and secure web design.

2. Summary of Previous Work

2.1. Current Penetration Testing Methodologies

Penetration testing, simply put, is the process of testing the security of a system in its production environment by simulating the actions of a malicious attacker to try and gain unauthorized access to protected assets. The goal of a penetration test is to try and uncover as many potentially exploitable vulnerabilities in a specific system as possible as well as determine the proper techniques to help mitigate those threats in the most cost effective manner possible. There are a number of different schools of thought on how a penetration test should be carried out, but the majority of these methodologies are based on the original work by Linde which introduces the Flaw Hypothesis Methodology for performing penetration testing on operating systems [57]. Although this approach was originally proposed for performing penetration tests (pen-tests) on non-networked operating systems, this work is arguably the foundation of modern pen-testing methodologies as well as the motivation behind modern pen-testing tools and vulnerability databases such as that found on the security focus website [87]. The flaw hypothesis methodology can be broken down into four stages:

1) Obtain detailed knowledge of the system
2) Generate a list of suspected flaws (known as the flaw hypotheses)
3) Confirm the hypotheses to be either true or false
4) Generalize the flaws found

This methodology is currently used to some extent in modern pen-tests and sparked a new branch of security research with the suggestion that many parts of this process lend themselves to automation and that the IT community at large would benefit from publicly available vulnerability databases.

There are two more recent security testing methodologies which have developed in both government and the open-source community. The first is the "Open-Source Security Testing Methodology Manual (OSSTMM)" released by the Institute for Security and Open Source Methodologies (ISECOM) [34] and the second is the "Guidelines on Network

Security Testing" released by the National Institute of Standards and Technology in the United States [101]. The OSSTMM is an excellent reference for performing a third-party security test on an organization from the perspective of an outsider. The methodology includes a list of tests that should be performed on various IT resources such as firewall, wireless access points, dial-up accounts as well as other aspects of the organization's security such as physical security. An extensive list of tests which can be performed on each component in the organization is also provided; however, details on how to perform the actual tests are not described in detail. This is expected as the methodology is designed to be used by experienced security testers. The OSSTMM provides an excellent starting point for a security evaluation; however, the methodology focuses on external attackers with no prior knowledge of access to the system. In performing this work, the attacker being considered is one who has access to various resources within the network and therefore, a methodology which takes the ability and initial access of the attacker into consideration. Also, in the OSSTMM, there is very little emphasis put on recommendations of mitigation solutions to the vulnerabilities found in the system. The guidelines compiled by NIST are similar to the OSSTMM in that they also focus on what an attacker is able to accomplish without any prior knowledge of the system. The main goal of the NIST paper is to provide system administrators with guidelines for performing security testing during the life cycle of a particular system. This guide is an excellent starting point as it provides an extensive list of tools which can be used to perform various testing tasks as well as detailed instructions on how to use a number of those tools. Since the NIST approach is designed to be used by administrators who may not have extensive IT security knowledge, it only covers the use of automated tools such as port scanners and vulnerability scanners in order to gain a better understanding of the security posture of the system. It is not specifically designed for third party security evaluations. Additionally, much like the OSSTMM, there is very little emphasis on possible mitigation solutions.

2.2. The Various Levels of Penetration Testing

As more research is being done in the penetration testing field and as networked applications become more complex, a number of specializations within the field have begun

to form. In his paper entitled "Penetration Testing: A Duet" [30], Dan Geer states that as a field matures, specializations will start to form. In the pen-testing field those specialized fields are network penetration testing, host penetration testing, and the previously mentioned application penetration testing.

Network penetration testing can be described as the act of testing an organization's network perimeter to see if it is possible to access hosts and services which should be unreachable. This could include testing such things as firewall configurations and dial-up connection settings which could grant an outside user access to the internal network resources. This type of testing can be done in large part with security tools. For a detailed case study which focuses on network penetration testing, see the case study by Baharin *et al.* titled "Third Party Security Audit Procedure for Network Environment" [7].

The next classification, host penetration testing, focuses on finding vulnerabilities in the services hosted by a particular machine which may be exploited to gain unauthorized access. Network pen-testing activities are generally performed before host pen-testing in order to determine which ports are open and potentially vulnerable. Because of the large number of hosts on the Internet and the relatively small number of possible software applications which may run on a single host, discovering a vulnerability in a widely deployed service may give a malicious user the opportunity to compromise a large number of hosts. One example of a vulnerability that has been around for a long that still appears in modern applications is the buffer overflow. Even though the dangers of buffer overflow vulnerabilities are very well documented, they still make up a very large part of all reported software vulnerabilities. According to the National Vulnerability Database managed by the National Institute of Standards and Technology in the United States, 291 remotely exploitable buffer overflow vulnerabilities have been reported so far in 2006 [69]. Since it can be quite tedious to cycle through all published vulnerabilities for a particular set of services running on a target host, a number of vulnerability scanners, both open-source and commercial, have been created to help automate the task of enumerating all of these potential vulnerabilities. Although host pen-testing can be automated to a large extent, a host pen-testing professional is still required to decipher the results of the vulnerability scans and ensure that there are no false positives. More details on the work done in the vulnerability

detection field is presented in section 2.8. For a detailed case study which focuses on host level penetration testing, see the paper written by Lo and Marchand entitled "Security Audit: A case study" [58].

The third classification of penetration testing is application penetration testing [30]. Application pen-testing is the newest of the three classifications and is quickly becoming one of the more important specializations due to the increasing popularity of web-based systems. These systems may have evolved over the years and may include large amounts of legacy code which were not written with security in mind. This, combined with the inherent complexity of many web-based application frameworks as well as new forms of attacks which specifically target web-based applications, make it even harder to protect valuable assets. Since application pen-testing is a relatively new discipline and deployed web-based systems can vary greatly from company to company, this specialization cannot be completely automated at the time of this writing. Although a number of tools exist which attempt to detect certain classes of attacks within web applications, these tools still require a fair amount of user interaction in order to work effectively. Because there is still a significant reliance on human resources in performing application pen-testing, as well as the fact that web-based systems can incorporate a potentially large number of smaller systems, it is crucial that a threat model defining the scope of the pen-testing activities be performed before testing commences. Threat modeling will be covered in more detail in the following sections. Also, since application pen-testing will make up a large part of the system evaluation performed in this work, specific application level threats with respect to web-based applications will also be addressed in the following sections.

2.3. Penetration Testing Methodologies Based on Modeling and Simulation

In this section we describe a number of methodologies which use modeling and simulation to help formalize the pen-testing process. The first work done in this field was method proposed by Phillips and Swiler based on modeling and simulation [75]. They make the argument that the pen-testing methodologies presented above lack formality and that

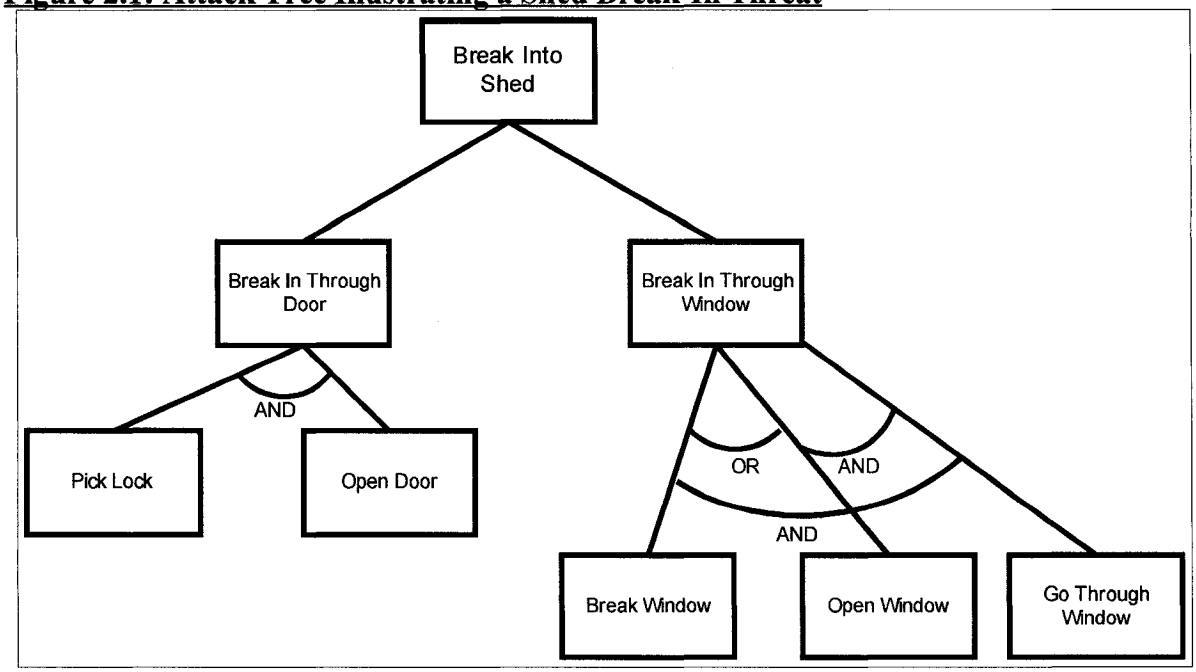
these approaches only provide a select subset of all the attack scenarios possible. By modeling the computer network in a more formal way that includes host configurations, connectivity, trust relationships between hosts, an intruder model, and a set of actions the intruder can take, the authors claim that it is possible to determine the total number of attack paths in the system. First, a graph called an attack template is created for each attack in a predefined attack database. The attack template is a generic description of an attack which includes the initial state of the attacker, necessary conditions required for a successful attack, the actions performed by the attacker, and the final state of the system after the attack. Next, an graph called an attack graph is generated by specifying the starting state of the host or network and the desired end state of the attacker and then adding edges and intermediate nodes based on the attack templates generated in the previous step. Edges are added if a relevant attack template that would allow the attacker to change from their current state to some other state in the target system is found. An attack path is any path in the attack graph that allows the attacker to go from an initial state to the desired, final state. Once all of the attack paths are identified, they may be analyzed and mitigations may then be considered. The goal of this approach is to not only determine system vulnerabilities based on known vulnerability databases but to also model what an attacker could do given a set of exploitable vulnerabilities within a system. Sheyner *et al.* extend this work further by proposing a method by which the attack graphs used to set up the model simulations may be generated automatically based on network and vulnerability information [90, 91]. Attack graphs generated using Sheyner's method will be made up of many intermediate states each representing an atomic exploit. A successful attacker will make use of a number of exploitable flaws to achieve his/her goals. This is represented in the attack graph as a root-to-leaf path. Because these models can become very large very quickly, the simulations which need to be performed become quite complex. Ammann *et al.*, propose a simplified model which, instead of modeling a number of aspects in the network as states, models them as attributes [4]. This approach is similar to the generation of attack templates which was explained above. In this approach, the authors model an exploit as a transformation which takes a number of preconditions and produces a number of post conditions. This has the effect of condensing each of the root-to-leaf paths into a single node with a multiple

preconditions and postconditions. Further, they eliminate the need for an attacker to backtrack when performing attacks. This means that once an attacker uses a particular exploit, the attacker maintains access to the resources exposed by that successful exploitation. The attacker does not have to reexploit previously exploited vulnerabilities in order to launch a more complicated attack. Another solution that has been proposed by Singh, Lyons, and Nicol to deal with the scalability problem present in Phillips' work is to use estimation techniques based on rare event simulation and importance sampling in order to estimate the number of paths through the attack graph [92]. This is a statistical approach used to analyze large attack graphs that would be infeasible to analyze using the afore mentioned methods. This approach will yield information such as the number of attack paths through the attack graph as well as statistics regarding attack path lengths.

2.4. Attack Tree Based Methodologies

An attack tree is a way describing how potential threats to a system may be realized by breaking each threat down into steps used to realize an attack. They were popularized by Bruce Schneier and have since been incorporated into most threat modeling and IT risk management methodologies [82,83]. The reason they have been incorporated into most threat modeling methodologies is that they present low level and potentially complicated details about potential threats in a way that technical teams as well as managers can understand. An attack tree is made up of a root node which describes the threat that is realized and child nodes which describe in increasing detail how the threat may be realized. Each non-root node describes an attack and the child nodes provide the details of how to realize the attack. Branches coming off a particular node may be connected using logical connectors, meaning all of the child branches may need to be true for the attack to be successful or the attack may succeed if a single branch is realized. Figure 2.1 shows a sample attack tree for breaking into a shed which should help illustrate how attack trees are used to illustrate potential threats and attacks.

Figure 2.1: Attack Tree Illustrating a Shed Break-In Threat



Attack trees were later formalized in a work by Mauw and Oostdijk [60]. Dalton *et al.*, have also proposed using Petri Nets as a way to represent attack trees as they offer a number of potential advantages such as being able to perform simulation as well as the ability to represent concurrency [17]. This idea of creating attack nets based on Petri Nets is extended into the field of penetration testing by McDermott where he demonstrates that attack nets may be used as the basis of a new overall penetration testing methodology [61]. Xu and Nygard make the claim that in addition to create threat models that Petri Nets may be used to verify the absence of threats in a system as well [112]. Since attack tree generation and attack modeling in general is largely reliant on expert knowledge in the security field, a collaborative model is proposed by Stefan and Schumacher which combines attack trees with a web interface which can be updated by the security community at large [96]. The goal is to model attack patterns in the form of attack trees in a public database similar to vulnerability databases like the SecurityFocus database [87] and CERT [14] and allow users to collaborate by updating the database using a Wiki type interface much like sites such as Wikipedia.org [111]. Similarly, Tidwell *et al.*, propose a formal syntax for attack trees which also includes any preconditions needed for an attack to be successful, such as a certain version of a certain OS deployed, as well as relevant postconditions of a particular attack [98]. This method also

allows for the modeling of more complex, multi-stage attacks since the focus of the work is on modeling Internet based attacks which focus on n-tier web applications.

2.5. Threat Modeling

In order to design a secure system, the stakeholders must understand the risks associated with the application as well as who is using the system and what assets they are trying to restrict access to. Therefore, we provide an overview of the work that has been done in threat modeling and make use of these ideas as a way to provide scope for third-party security evaluations.

2.5.1. Trike [79]

The Trike threat modeling framework is a methodology proposed by Saitta, Larcom, and Eddington and the focus of this methodology is to provide a complete description of the system to be tested before performing any actual testing [44]. The authors claim that by gaining a complete understanding of the system, the testing team will be able to find more vulnerabilities than testing with little or no prior knowledge. The Trike methodology is made up of four distinct models: the requirements model, the implementation model, the threat model, and the risk model.

The goal of the requirements model is to describe the actors, the assets, the intended actions, and the rules governing those actions. An actor is essentially a role in the system. The case of a university system, one group of actors may be students while another may be professors. An asset is a discrete data entity that is inherently meaningful in the problem domain of the system. For example, in this model, a student's grades in an online student services system are considered an asset. An intended action is something that an actor performs on an asset. The list of intended actions should only include actions which are meant to occur during normal system use. Rules describe the context in which an intended action may be executed.

After the system requirements have been defined, the details of the implementation are then considered. The actions allowed by the system are described using data-flow diagrams [113, 29]. Once the initial data-flow diagram is completed, more specific system

information should be added such as host OS, platforms used, application servers, protocols used in data flows, and any other relevant information including version information. It is also important to include information about trust boundaries as well as specifications of any interfaces on endpoints which cross a trust boundary.

The final step in creating the implementation model is to combine the data-flow diagrams from the previous step and the actions, both intended and supporting, into what Saitta *et al.* refer to as use flow diagrams. Use flow diagrams are used to describe the valid use cases of the system. Use flow diagrams help in discovering complex attacks on the system [79].

The third step in the Trike methodology is to create the threat model. According to the authors of Trike, a threat is any action within the program domain which should not occur. The set of threats to a system are generated from the set of actions described in the requirements model and the implementation model. Every action that should not occur is a threat and the denial of every action that should occur is a threat. Attacks which realize these threats are described using attack trees (described above in section 2.4.).

The final step of the Trike methodology is to create a risk model. The goal of the risk model is to assign risk values to each of the threats to the system. This allows the stakeholders to prioritize the threats in order of severity. The more severe threats will be mitigated sooner. The risk values are assigned based on the estimated value of the affected asset, the trust level of the actor performing the action on the asset, the reproducibility of the attack realizing the threat, and the ease of exploitability of the vulnerability in the system.

2.5.2. STRIDE/DREAD [35]

The STRIDE/DREAD methodology is a threat modeling and risk ranking framework created by Microsoft. This methodology is designed to be used during the software development life cycle in order to help identify potential risks to the application and come up with some way to mitigate those threats. The STRIDE methodology is broken down into seven steps: (1) form the threat modeling team, (2) decompose the application, (3) determine the threats to the system, (4) rank the threats in decreasing order, (5) figure out how to respond to the threats, (6) choose a technique to mitigate the threat, and (7) find an

appropriate technology to implement the mitigation technique. This framework is similar in a lot of ways to the Trike methodology. When decomposing the application, a number of data-flow diagrams are produced which describe the system in much the same way as the implementation model of the Trike framework. One noticeable difference at this point in the process is the absence of some equivalent to the requirements model. There is no formal stage in the STRIDE methodology where the actual assets, actors, and actions are described in detail. This information may be inferred from the data-flow diagrams.

In the third step of the methodology, each component in the system (that is, each entity in the data-flow diagram) is analyzed according to STRIDE in order to find all of the potential threats to that component. STRIDE is an acronym for the threat classification system used in the methodology and it stands for Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, and Elevation of privilege. There may be some overlap when trying to categorize certain threats. Once all of the threats have been identified, a threat tree should be created which should illustrate all the ways that a threat could occur in the system. These are similar to the attack trees created in the Trike methodology.

When performing step four of this methodology, risk is calculated according to another acronym, DREAD. DREAD stands for Damage potential, Reproducibility, Exploitability, Affected users, and Discoverability. A value from one to ten is assigned to each of these qualities for each threat and then an average score is taken which then becomes the risk ranking of the threat. This allows threats to be ranked and prioritized. Again, there are some similarities between this model and the Trike model. The only difference is that the DREAD model makes use of some additional qualities, such as discoverability (which the authors state is almost always 10) and affected users (which realistically can be coupled with the Damage potential quality). When all of the threats have been identified, the next step is to choose how to respond to each threat. In this model there are four options: (1) do nothing, (2) warn the user, (3) remove the problem, (4) fix the problem. Although option (4) appears to be the best solution, it can also be the most costly in terms of resources. Therefore these options are presented in order from least ideal and least costly to most ideal and most costly. Once the appropriate option is selected, the final two steps may be performed. Those are to

select the mitigation technique and to choose the technology to implement that technique.

The STRIDE/DREAD threat modeling methodology seems to be the most widely used framework when design teams are modeling a new project. It allows the teams to incorporate security considerations from the beginning of the software development life cycle and it is recommended by the Open Web Application Security Project when developing new applications [1]. A series of papers were written and presented at the Conference on Communications and Multimedia Security which provide examples of threat models for a number of applications and frameworks such as ASP.NET [31], SQL Server [9], Active Directory [15], web services based web applications [20], and smart card based security tokens for web applications [18].

2.5.3. OCTAVE

As previously mentioned, there are number of threat modeling methodologies in use. In this section we present a brief overview of those methodologies not covered above. The OCTAVE information security risk evaluation framework is a process which is heavy on documentation and whose goal is to introduce and maintain a security aware company culture [3]. The processes described are to be used by a team within an organization to ensure the company assets are sufficiently protected. The goal of the OCTAVE approach is to introduce a corporate security culture from a management point of view. Therefore the specific technical details regarding evaluating the security posture of an application with a web-based interface are not covered to the extent necessary for this type of evaluation.

2.6. Adversary Modeling Methodologies

An interesting point that needs to be considered when performing a third-party security evaluation is who the organization is trying to protect its assets from. In order to further define the scope of a third party security evaluation, an attacker profile should be considered. This idea is introduced by Salter *et al.* [80]. They claim that in order to protect a system, the organization must know who they are trying to protect themselves against. By adding an adversary profile based on attributes such as risk tolerance and objectives of the specific group of attackers, the organization will have a better idea of who they are up

against. All of this information falls under what the authors call the adversary model. The adversary model will help to limit the scope to relevant attackers and provide data that is more useful to the stakeholders once the evaluation has been completed. Attacker capabilities are considered in the threat modeling process proposed by Hasan *et al.*, when creating a threat model for storage systems [32] and again later by Myagmar *et al.* [67]. When performing a third party security evaluation of a system, attacker capabilities need to be specified. Qualities such as technical skill level, budget, tools available, specific system knowledge, and any other attributes that are deemed to be relevant, should be specified when creating the adversary model.

2.7. Web Application Attacks

When describing the wide range of web application specific attacks that have emerged over the years, we must consider the vast amount of information made available by the private sector. This is due to the fact that the majority of new attacks are being proposed and tested within the private sector. Therefore, this section will contain a significant amount of information that has been drawn from the private sector rather that the academic research literature.

The Open Web Application Security Project (OWASP) is an online community that offers free, open-source security tools and information to help raise awareness of security in the web development community. As part of this initiative, OWASP periodically publishes a "top ten" list of the most common and dangerous flaws found in web applications [97]. The OWASP top ten is an excellent starting point for developers who wish to have a better understanding of how web applications may be exploited. This is not a standard way of classifying vulnerabilities, it is more of a guide on what to look for when performing a code review. A number of researchers and organizations have attempted to define a vulnerability classification scheme [74, 88]. The one common trend among the majority of web application vulnerabilities is that they all exploit the same root cause, which is also the number one flaw on the top ten, and that is the use of unvalidated input. It can be said that almost all attacks on web-applications make use of the fact that there are a large number of

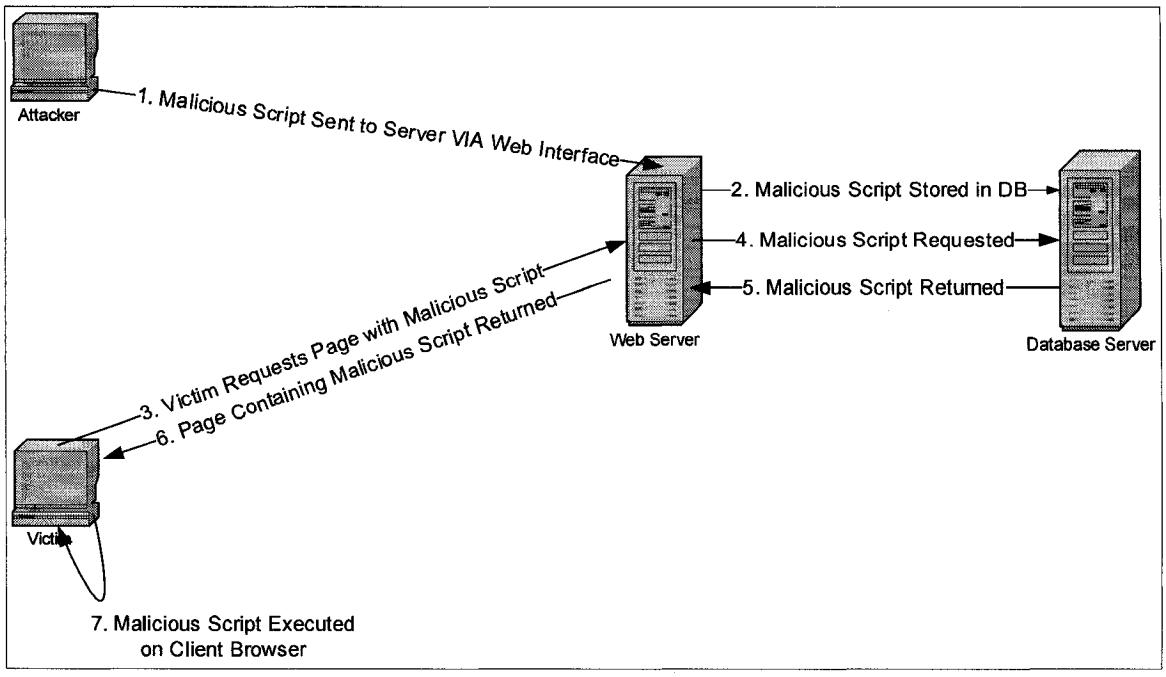
input points within a web application, the data used by web applications is usually weakly typed, and the input data is not properly validated. In the remainder of this section, I will provide an extensive list of widely known vulnerability types which can be used to attack web applications. This may not be an exhaustive list as new classes of attacks continue to be discovered.

2.7.1. Cross-Site Scripting (XSS)

As was mentioned in the previous section, a large number of web application vulnerabilities are caused by web-applications using client supplied input without performing any validation. It is not uncommon for a web application to take user input and incorporate it into dynamically generated pages. This type of application behaviour combined with the various powerful client-side scripting technologies can allow an attacker to deface websites and steal session tokens used for authentication. A cross-site scripting vulnerability, simply put, is a vulnerability in a web-based application that allows an attacker to enter client side code that will be run by another user under the pretense that it came from a trusted site. This may seem like a fairly benign attack; however, XSS attacks can be used as a step in a more complex attack such as a session hijacking attack. Session hijacking and session management are covered in detail in chapter 4.

There are two main types of XSS attacks: stored and reflected. A stored XSS attack, also known as a passive XSS attack [114], is where the attack code is stored in persistent storage, such as in a database, on the server where it is later retrieved and executed by an unsuspecting user. This type of XSS vulnerability is common in applications such as guest books or mailing lists which allow users to add records that will be retrieved by other users at a later time. Stored XSS attacks are particularly dangerous because they can be executed without any interaction from the victim and they can be programmed in such a way that the malicious actions go completely unnoticed. Figure 2.1 below illustrates how a stored XSS is attack is executed.

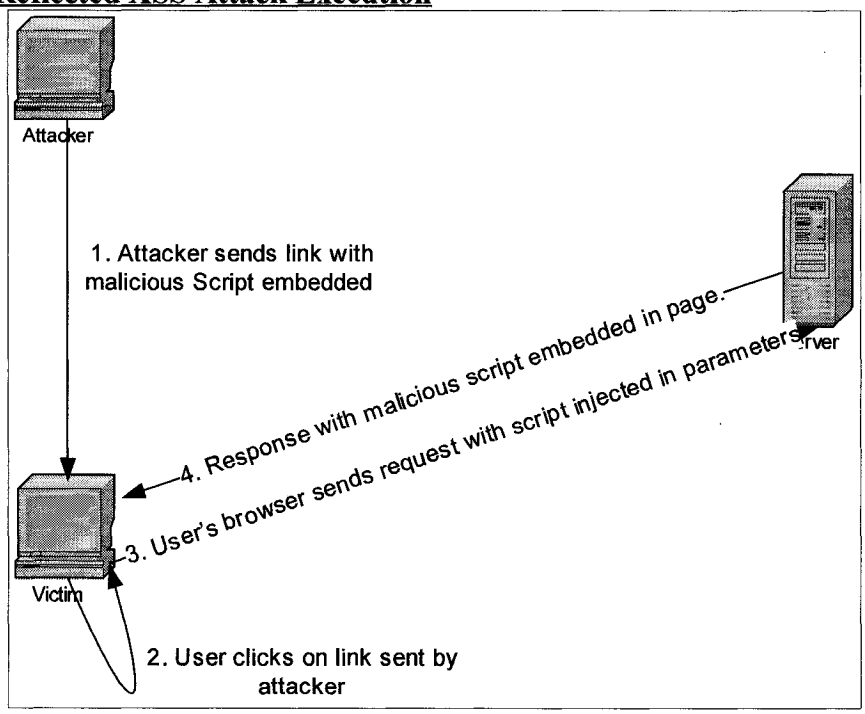
Figure 2.2: Stored XSS Attack Execution



A reflected XSS attack, as its name implies, reflects the attack code off of the server to the victim. As described on the OWASP website, this is done by sending the the victim a link via a third source such as an e-mail or another website. If the vulnerable site encodes parameters used by server-side applications in the URL, a malicious user can inject script into the URL using those parameters. When the user clicks on the malicious link, the attacker's code is submitted to the server via the URL parameters and then reflected back to the victim. The malicious code then appears to have come from the trusted site and is executed on the victim's browser [97]. A reflected XSS attack, which is also known as an active XSS attack [114], can be used to steal user credentials, but it will most likely be used to temporarily deface a legitimate website. This attack is particularly serious when executed against reputable information sites such as major news websites. An attacker could provide the victim with a link to the vulnerable site that has a fake article encoded in the URL parameters so that when the victim clicks on the link, it appears that the fake story is legitimate since it is coming from a reputable site. Figure 2.2 below illustrates how the reflected XSS attack is performed. This type of vulnerability can also be used to launch a phishing attack to obtain valid user credentials from valid clients. Phishing is an attack

where the malicious user attempts to obtain sensitive information by pretending to be a representative of a legitimate business and tricking the victim into providing information such as user credentials and credit card numbers. Again, this can be done by contacting the victim via e-mail and providing a link that, when clicked, will display a form which requests the victim's information. An XSS vulnerability can be used to make this type of attack more believable since the attacker will be able to reflect the malicious form off of the vulnerable host thus giving the victim the impression that the page is legitimate.

Figure 2.3: Reflected XSS Attack Execution



2.7.2. Back-end Injection

Back-end script injection attacks are similar to XSS attacks in that if an application is vulnerable, it allows a malicious user to inject script into the application which will then be executed in another part of the system. Since the majority of web-based applications make use of some form of SQL database to manage their information, SQL injection has become quite popular. This is because it can allow an attacker to completely bypass the authentication mechanisms in place or allow him/her to harvest potentially sensitive information within the vulnerable application. This is done by using the web application as a

portal to the back end SQL database. Data can be passed via URL parameters or any other input available in the application which, when received by the database server, will change the intended actions of the original query. These vulnerabilities exist because data that is submitted by the browser is trusted on the server and is used to form queries without being checked and sanitized. The following example is used to illustrate this point further. The code snippet below is an example written in PHP (with some pseudo code for DBMS independence) of a server side script which accesses a database to determine whether a particular user is authorized based on a login.

```
<?php
        ...
        $username = $_POST['username'];
        $password = $_POST['password'];
        $md5pass = md5($password);
        $queryStr = "Select  * from users where username=\'".$username."\' AND
        password=\'".$md5pass."\'";
        [execute query]
        if([recordset is not empty])
                $redirectURL = "sensitive.php";
        else
                $redirectURL = "notauthorized.php";
        http_redirect($redirectURL);
?>
```

In a normal use case, the user would submit a form containing his/her user name and password which would be passed to the database. If the query returns an empty record set then no record matching the information submitted exists and the user is not authorized. To illustrate this, assume that the user submits the user name "gary_metz" and the password "let me in". The query constructed and processed by the database server will be the following:

```
SELECT * FROM USERS
        WHERE username='gary_metz'
        AND password='efa88170397c87c264cb471f3cf86e6d'
```

If a record exists with this user name/password combination then the user will be granted access to the sensitive.php page. However, because the inputs submitted by the user are not sanitized, a malicious user can submit a form which changes the syntax of the query. Table 2.1 below shows some possible inputs which would alter the syntax of the query and an explanation of the results.

Table 2.1: Possible SQL Injection Attack Inputs

Input	*Resulting Query*	*Side Effects*
username = gary_metz'-- password = pass	SELECT * FROM USERS WHERE username='gary_metz'--' *AND password='1a1dc91c907325c69271ddf0c944bc72'*	By adding the single quote at the end of the username input, this has the effect of ending the string literal which then allows an attacker to insert arbitrary SQL code. In this case, the double dash is used to denote a comment which means that everything after that point is not executed. This allows the attacker to login to the system using any valid user name without knowing the password.
username = ' OR "="-- password = pass	SELECT * FROM USERS WHERE username=' ' OR "="--' *AND password='1a1dc91c907325c69271ddf0c944bc72'*	Again, the single quote is used to escape from the string literal and allow the attacker to insert code. In this case, "=" is always true so this query will return the entire users table in the result set.
username = brad';DELETE * FROM USERS;-- password = pass	SELECT * FROM USERS WHERE username='brad';DELETE * FROM USERS;--' *AND password='1a1dc91c907325c69271ddf0c944bc72'*	In this case, the semi-colon is used to create a compound statement which allows the attacker to delete the entire users table. Although this may not result in accessing the restricted part of the system, it may cause some problems for the site owners as all of the user information will be deleted.

Table 2.1 only gives a small number of the possible attacks that can be launched on a system designed in this way. Also, unless the web server is completely compromised, the attacker will generally not have access to server-side script code. Instead, attackers will rely on information leaked by the application in the form of error messages. Error messages thrown by web application frameworks as well as the DBMS used can provide important information such as valid field names, DBMS type and version, and the query structure of the target application. This information is valuable to the attacker because without it, it becomes much more difficult to form a valid SQL query and the DBMS will only execute valid queries. An excellent resource on SQL injection can be found in [94].

Although SQL injection vulnerabilities tend to be the most talked about, they are not the only command injection vulnerabilities. Some web applications make use of built-in operating system functions using parameters supplied by the user's browser. This type of system has the potential to provide an attack vector that could allow a malicious user to execute operating system commands on the host through the web application. The same is true for applications which allow users to upload and access files on the server. Anytime the web application makes a call to an external resource (such as a database server, command shell, LDAP server, etc.) using user supplied data, the threat of a command injection attack is there. One more script injection type attack which is particularly relevant to web-based applications as well as web-services is the XPATH injection attack [50]. XPATH is essentially a querying language used by application programmers for extracting data from an XML document. This means that if an application makes use of unsanitized user input to form XPATH queries, it may be possible for an attacker to extract any information from the target XML document by altering the query string. This attack can be performed in a similar fashion to the SQL injection attack described above.

2.7.3. Other Attacks on Web Based Applications

One of the common flaws mentioned in the OWASP top ten which may not get as much attention as command injection attacks or XSS attacks is the danger of weak authentication and session management practices. Access control and authentication schemes are notoriously hard to design and implement properly. However, despite the

difficulty inherent in designing these systems, many web application developers create their own solutions which, in a lot of cases, are based on flawed assumptions. Session management, because it is such a large part of all web-based applications, will be covered in great detail in chapter 4. It will also include descriptions of a number of recent attacks whose goal is to obtain authentication or session management information from unsuspecting users. Another set of vulnerabilities that can be exploited in a more indirect way are host configuration errors in widely used software packages. It is extremely uncommon for an organization to program a web-server application which can be used to host said company's web-applications. With this in mind, when a flaw is discovered in a web application server, widespread automated attacks are inevitable. The same is true for generic web applications such as phpBB [76] which is used to set up online bulletin boards or SquirrelMail [95] which is used to set up a web-based e-mail service. As applications like these are more widely used, it becomes more tempting to try and find flaws which may be used to exploit the systems running them. Also, once a vulnerability goes public, it is not hard to download a vulnerability scanner and start scanning hosts at random in the hopes of finding a host running the vulnerable software. As was previously mentioned in section 2.1, penetration testing lends itself to the development and use of automated testing tools. Since that original work was published in 1975, there have been dozens of automated tools created for performing many different types of scanning activities. Many of these tools are available publicly. In the next section, I will cover the work done in the field of vulnerability scanning and detection.

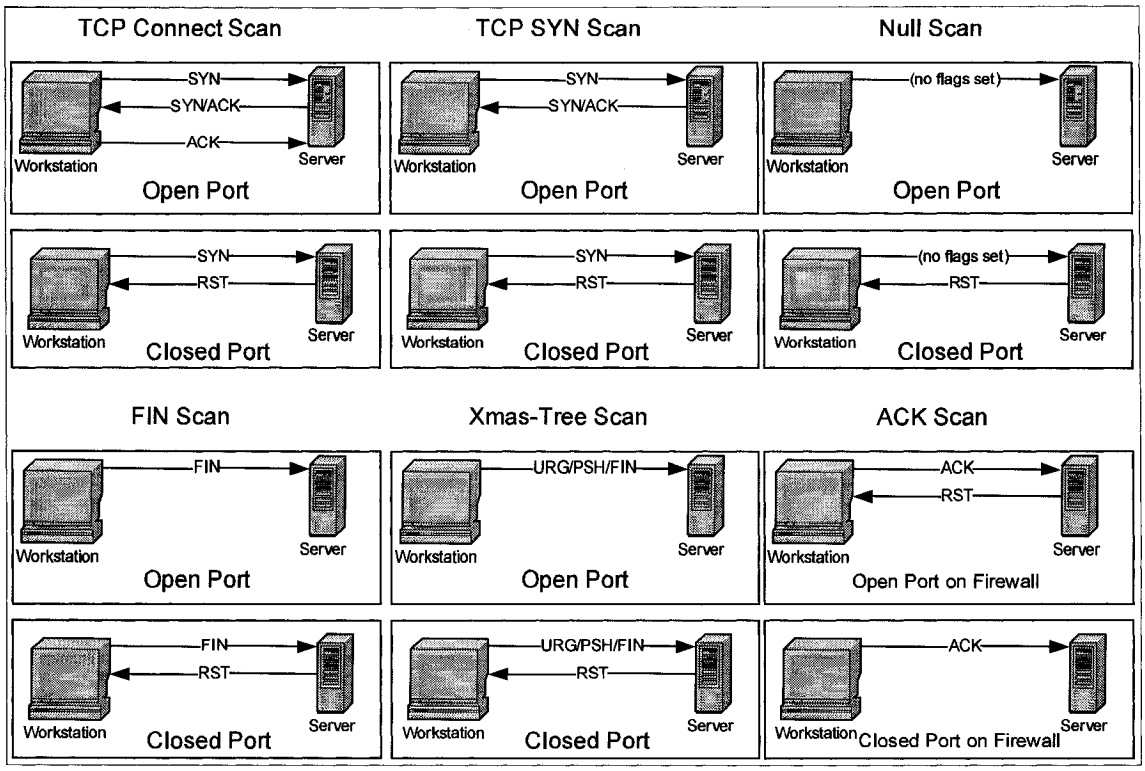
2.8. Automated Scanning Techniques and Tools

2.8.1. Port Scanning

When describing the different scanning techniques that have emerged over the years, it makes sense to categorize these techniques by which pen-testing specialization they are used in. The specializations, as described above, are network pen-testing, host pen-testing, and application pen-testing. As was previously mentioned, network level pen-testing aims to determine which ports are open to the attacker as well as to map out firewall rule sets. Typically this is done using a technique called port scanning. There are a number of

different port scanning techniques which may determine whether a host is listening on a specific port but the basic is idea is to send a specially crafted TCP packet to a particular port on the target host and observe the response. Based on the response sent by the host, it is possible to determine whether the port is listening or not. A number of different scans may be performed by setting different combinations of the TCP flags. This idea was first published in Phrack Magazine in 1997 by Fydor [28]. This is also the publication where the NMAP tool was first introduced. NMAP is now one of the most widely used port scanning applications in the field. There are a number of different ways to perform a port scan. These techniques are illustrated in below in Figure 2.3 which is a modified version of the figures provided in [110]. An excellent overview of these techniques can be found in [19] which also explains a number of techniques which allow the tester to perform a scan without being detected by intrusion detection systems (IDS). In short, there are two types of scanning techniques, regular and inverse. A regular scan, such as the TCP connect scan and the SYN scan, determine that a port is open based on the successful receipt of an ACK message. Conversely, inverse scans, such as the NULL scan, FIN scan, and Xmas-Tree scan, determine that a port is open if there is no response to the packet sent. The ACK scan is a special type of regular scan which can be used to map out firewall rules. The ACK scan cannot be used to map whether a port on a host is open or closed but it can be used to determine whether a firewall stateful or stateless [2]. A stateless firewall is a firewall that blocks all SYN packets to ports that are closed. If a stateless firewall is in place, it will allow the ACK packet to pass and the service responding to the probe will respond with a RST packet. In this case the port is said to be unfiltered. Conversely, a stateful firewall will not only block incoming SYN packets but it will also block any ACK packets that are not linked to an open TCP session. Therefore, when a the random ACK packet is sent, it will be blocked by the stateful firewall and no response will be sent back. In this case the port is said to be filtered.

Figure 2.4: Port Scanning Techniques



Once the available ports are known, the tester can move on to host penetration testing. Network pen-testing and host pen-testing are almost always used together since it is important to determine the exposed ports on the target host before finding a vulnerability to exploit. Once the open ports are known, there are a number of techniques which can be used to determine the operating system running on the host, which services are running on the host, and if any of the services running are vulnerable to attack. In Fydor's work entitled "Remote OS Detection via TCP/IP Stack FingerPrinting" [27] the author explains how by observing the subtle differences in the TCP stack, an educated guess can be made as to which OS is running on the target host. This is due to variations in the implementation of the TCP protocol across different operating systems. This functionality is offered in a number of scanners including the aforementioned NMAP tool. Other techniques such as connecting to the target host using services such as Telnet will often display a banner which will identify the OS and version information as well.

2.8.2. Service Scanning

Service identification is generally done by cross referencing a list of standard ports. For example, if a host is found to have ports 80, 23, and 443 open, it would be a good guess that an HTTP server, a Telnet service, and an HTTPS server are running on those respective ports. This may not always be the case as there is nothing stopping a system administrator from running their Telnet service on port 80; however, the tester can start by assuming that services are running on standard ports and prove that assumption to be true or false as the test continues. The NMAP tool offers a service identification function which sends a number of specially crafted requests and observes the response. This response is then mapped to a typical response in the service database provided with the tool. Again, based on the response, NMAP can make an educated guess as to which service is running on the port. A similar tool proposed by Skaggs *et al.*, combines all of the above techniques to create what they call a machine fingerprint [93]. The machine fingerprint proposed by Skaggs *et al.* consists of OS type and version information, open ports, and services running on those ports.

2.8.3. Host Vulnerability Scanning

Once a machine fingerprint has been developed for each target host, all of the services running are checked against a vulnerability database to find potential vulnerabilities and attempt to exploit them. Again, the idea of keeping an updated public record of known vulnerabilities in widely used software systems as well as the idea of creating automated tools to scan hosts for these vulnerabilities were proposed in 1975 by Linde in his seminol paper on penetration testing [57]. Since then, a number of databases have been created which can be used to find published vulnerabilities in hundreds of widely used software systems [16,13,12]. Also, a large number of security scanners which make use of these databases to perform scans on systems and identify potential security flaws have also been produced. A comparative study of a number of vulnerability scanners was performed by Lerida *et al.* in their study entitled "Detecting Security Vulnerabilities in Remote TCP/IP Networks: An Approach Using Security Scanners" [55] but some of the scanners should be mentioned in more detail. One of the original security scanners used to scan UNIX hosts for potential vulnerabilities is the COPS scanner [23]. Although this scanner is used to

specifically identify vulnerabilities within a host running UNIX locally, it is one of the earlier works on the topic. A more recent scanner based on COPS improves on the original work by creating a centralized application used to run custom plug-ins [89]. The scanner, called Ferret, also improves on the reporting functionality to make scan results easier to read or output results in customized formats such as XML for further processing. Although both COPS and Ferret can be used to identify vulnerabilities on a host, they were designed to be used locally. This means that the tester would need to have access to the machine before running these tools. This illustrates the need for a vulnerability scanner which can be used to detect potential security holes in a remote host. One of the first remote host vulnerability scanners developed was a scanner called SATAN. Originally described in a Usenet posting, Farmer and Venema describe a method of testing host security where the host is viewed from the point of view of the attacker. They also describe their security scanner as an extensible application which is designed in such a way that as new exploits are discovered, new modules written in one of a number of programming languages can be added and used during testing [24]. Although these early examples of security scanners were somewhat crude (for the most part they were just big collections of shell scripts and applications which had to be updated manually), they provided many of the design objectives used in more modern scanners. One of those scanners that is widely used today is Nessus. The Nessus security scanner has a modular construction which allows it to be updated by the security community through a plug-in interface. Nessus plug-ins are coded using the Nessus Attack Scripting Language (NASL). It also scans non-standard ports for potentially vulnerable services [70]. Similar scanners have also been proposed such as the Security Auditing Tool (SAT) which also incorporates an inference engine which has the ability to create new probes based on the results of previously executed probes during the scan [64].

2.8.4. Web-Application Vulnerability Scanning

Although the tools mentioned above are quite useful for finding vulnerabilities in a particular host, they do rely on large repositories of known bugs. When performing application level pen-testing, these types of tools become much less useful because web-based web-based systems tend to be custom made and are therefore not widely deployed.

Individual systems have unique structures, attributes, combinations of technologies used, and functional requirements. This means that any tools developed to assist in application pen-testing must test based on a more reactive strategy which can account for the differences in the systems. The tools proposed for this type of testing perform their analysis using either a black-box or white-box testing approach.

A black-box testing approach is used when there is no access to the source code or any other formal system documentation about the system being tested is available. Black-box testing is performed by supplying inputs to the system (or some component within the system) and observing the output [56]. This method of testing is generally preferred in the security community as it models the abilities of an actual attacker more closely. Conversely, a white-box (or glass-box) approach is used when source code and/or other design documents for the system are available [56]. If the source code is available then it can be checked for potentially dangerous code. Although the white-box approach may not test the system in the same way that an attacker might, having the source code available for analysis helps increase the efficiency of a security audit as a number of flaws in the system may be identified more quickly and with less interaction with the deployed system.

A number of tools have been developed in both testing domains. One of of the most widely cited works in black-box security testing of web-based applications is a solution proposed by Huang *et al.* They propose an application scanner based on fault injection and behaviour monitoring [41]. In their original work, Huang *et al.*, propose a scanner which attempts to mimic the actions of an attacker in order to uncover potential security vulnerabilities within a web-application. Their scanner, which they call the Web Application Vulnerability and Error Scanner (WAVES), starts by crawling the web application and identifying all the potential input points. The WAVES scanner employs what is known as "deep crawling" [8] when performing the crawling stage of the application scan. The idea behind deep crawling is that a large part of the content available on the world wide web is hidden behind some type of query interface (e.g. some type of login interface). Although the claim made by the designers of WAVES is that they locate all input points, it seems that only HTML form inputs are actually considered when performing their tests. The original WAVES scanner also did not directly search for XSS vulnerabilities, it only searched for

SQL injection and OS command injection vulnerabilities. However, in a follow-up paper, Huang added functionality to the WAVES scanner to allow for XSS detection as well as the introduction of a "safe-mode". The goal of the safe-mode was to offer a scanning method which does not make any permanent changes to the system while performing the evaluation. The original WAVES scanner had the potential to make permanent changes to the back-end database while checking for SQL injection attacks. The safe-mode introduced by Huang *et al.* aims to give the tester the option of preventing this potential side-effect [42]. A more recent work by Kals *et al.*, proposes another black-box based application scanner called SecuBat [48]. The SecuBat scanner aims to be a more extensible scanner than WAVES in that new exploits may be added to the test bed in the form of plug-ins. The modular design also allows the crawling and attacking actions to be done separately at different times. Like the WAVES scanner, SecuBat only searches for HTML form inputs as attack points. Other inputs such as URL parameters and cookies are not considered when performing tests. Also, the SecuBat scanner does not make use of any deep scanning techniques which means some potential weaknesses may be missed.

White-box testing is equally as important as black-box testing when performing a security analysis. The down-side of white-box testing is that it can be a daunting task and once the testers have been analyzing the code for some time, they may start to miss certain flaws due to fatigue. A number of tools have been proposed to help perform white-box testing from a security point of view. One of the earlier works in this area was written by Huang *et al.*, and it proposes a security scanner called WebSSARI which performs static analysis of PHP application code and inserts runtime guards into potentially unsafe code [43]. Although WebSSARI only works with PHP code, it is designed in such a way that new languages may be supported by adding new code walker modules to the system. A code walker is a module that creates an abstract syntax tree which the scanner uses to perform analysis. Another scanner called Pixy was proposed which specifically addresses issues present in the PHP scripting language [47,46]. Because the Pixy scanner is specific to PHP, it seems much less likely that it will be useful in analyzing any other technologies. A similar approach is also proposed by Martin *et al.* except that they offer a solution where the testers or developers can create their own attack signatures using a language called the Program

Query Language (PQL) [59]. PQL attack descriptions are passed to the analysis tool which can act as both a static analyzer and a runtime error checker.

Even though a significant amount of work has been done in the field of vulnerability scanning, simply running scanner tools in order to test a system does not provide adequate assurance that a system is safe. Vulnerability assessment tools do suffer from certain downfalls [99]. One is that the results produced tend to have a large amount of false positives. This is due to the fact that most scanners simply look for certain information which may indicate that a system has a particular flaw. The scanner will not always perform the actual exploit of the vulnerability in order to prove the system is vulnerable. Therefore, when performing a security test, it is important that a tester go through all the records produced by the scanner to check that there is actually a problem. Scanners can also provide inconsistent results. If a number of scans are performed on a particular system, the results may differ. This may be due to a number of reasons but inconsistent results can most likely be tied to a software bug in the scanner itself. After all, a vulnerability scanner is still a piece of software. Finally, vulnerability scanners are able to look for widely known configuration errors or certain, relatively simple attack patterns. In other words, they help identify any "low hanging fruit". It is much less likely that a security scanner is going to locate a complex attack which leads to an authentication bypass or session hijacking. Also, it is unlikely that a vulnerability scanner will be able to find flaws in security policies or even implementations of security policies. This does not mean that security scanners are completely useless. They simply need to be part of a larger testing process and should not be relied on solely when performing a third-party security evaluation.

2.9. Summary

In this chapter, we presented the necessary background information needed to better understand the common vulnerabilities found in web-based application as well as some of the techniques used to uncover these vulnerabilities. Also, because we believe that threat modeling is an important part of the third-party security evaluation process, a summary of threat modeling techniques proposed by both the private sector as well as the academic sector is given. Since this thesis is based on third-party security evaluations as well as

penetration testing, previous work in the these fields was also covered. In the following chapter, we present our proposed third-party testing methodology. This includes the steps taken when performing the evaluation on the AAP system as well as a description of how to create a threat model that is better suited for defining the scope of a third-party evaluation.

3. Proposed Third Party Testing Methodology

In this chapter I will explain a general methodology which can be used to perform a third party security evaluation. The methodology described here draws on information from previous testing methodologies presented in section 2.1 as well as incorporates aspects of threat modeling (see section 2.5) and adversary modeling (see section 2.6). The methodology proposed here is broken down into four phases which will be discussed in greater detail in the following sections: the initial scoping phase, the information gathering phase, the vulnerability testing and analysis phase, and the post-audit phase.

3.1. The Initial Scoping Phase

Before the security evaluation is started, the scope of the evaluation must be determined and agreed upon by all parties involved. The scope should include all available information regarding which systems are to be included in the evaluation, which hosts are to be included in the evaluation, specific actions which the testers should avoid, and an initial model of the type of adversary being considered. In the following sections, details about the information that should be gathered is presented.

3.1.1. System Information Gathering

Any information regarding the system being tested will be helpful in obtaining an accurate view of the system's functionality as well as in finding potential vulnerabilities. Many organizations will have a significant amount of documentation about their systems. A large amount of relevant information may be obtained in the form of the following types of documents:

- network architecture documents and diagrams
- design specifications for custom software systems
- source code for custom software systems
- documented security policies for the organization
- previous security evaluation reports or data
- any information on previous security breaches

If the information has not been formally written up, it may be obtained through general meetings with representatives of the organization. In the case where information is gathered from organization representatives, it is important to document all of the information gathered verbally. This way the information gathered can be submitted to those representatives for verification to ensure that the testers have an accurate view of the system in question. A summary of all the information gathered directly from the organization is created so all members of the testing team are aware of what information they have access to.

3.1.2. Initial Threat and Adversarial Model

The initial threat model is created to outline the scope of the evaluation before it is performed. This threat model will contain the initial descriptions of all the potential classes of users that are relevant to the evaluation. For example, in the case of the AAP system described in chapter 1, there are three relevant user classes, namely students, professors, and administrative personnel. Each of these classes of users will have a different set of actions which they can perform in the system. Since this is a pre-audit phase, the testing team may not have a complete description of what each class of user should and should not be able to do. The goal during the pre-audit phase is to simply determine all of the classes of users in the system and, if there are any classes of users which do not need to be considered while performing the evaluation, identify which classes of users are to be considered during testing. Although it may seem counterintuitive to eliminate user classes to consider during the evaluation, allowing the testing team to focus on a particular subset of all the users in the system may be necessary due to budget or time constraints. The following is a sample user profile produced for the AAP system described in section 5.1.

> *There are a number of different types of users using the AAP system. Among them are students, professors, and administrative personnel. End users are presented with a number of different options based their role in the system. Students can make changes to their own registration information (drop or add courses) as well as change their contact information. Students are also able to view their grades and financial standing with the university; however, they cannot make changes to this information. Also, one student*

may not view the information of another student in the system. Administrative users have access to the AAP through the web-based interfaces but also have access via a proprietary command line interface.

Although it is a possibility that administrative staff or professors will attempt to alter information that they do not have access to, the scope of this project is to focus on what a student is able to do with their given access. Therefore, the primary user group that we will focus on will be the student user group.

The organization may also wish to only test a particular part of a larger system. Again, this could be for a number of reasons. Software systems have the potential to evolve over many years and the organization may wish to only test newer additions to their overall system. They may also wish to omit certain parts of the system from the evaluation since they may no longer be in use or may be phased out in the near future. In the information gathering phase, more resources will be devoted to determining the exact system landscape that will be subject to more extensive testing. Ultimately, the goal of a malicious user is to compromise some asset within the system. Therefore, it is also important at this time to outline what the assets in the system are as well as the rules which should enforce how those assets are accessed by the relevant classes of users. A sample asset description is given for the AAP application below.

The main asset we are trying to protect is the student information stored in the AAP. Again, users should only be able to view information they are allowed to view and alter information they are allowed to alter. This information includes both academic and financial information. For example, a student is allowed to view but not alter their grades and they are allowed to view and alter their contact information. A student should not be able to view or alter the information of another student and a student should not be able to alter certain information about themselves (such as grades). A student's financial information in this case refers to a students account with the university which is used to bill a student for such things as tuition, residence fees, and library fees. A student's account can also be credited when the student makes a payment, is awarded a scholarship, or pays a library fine. Students should not be able to change financial information about themselves or others in the system. Some of these systems may have their own temporary storage databases which are used to update information in the AAP. These are another set of targets if a malicious user is able to compromise those systems.

Once the initial user profiles and assets are specified, the final step in preparing the

initial threat and adversarial model is to prepare the adversarial model. The goal of the adversarial model is to outline an accurate description of the potential attacker being considered in terms of resources, risk tolerance, and system knowledge. The adversarial model is used to further define the scope in terms of attacker profile. This is a necessary step since considering all possible attackers may not be possible and will not likely be useful to the stakeholders. For example, the organization managing the AAP system will likely be interested in what a student with a valid account can do with the system. Conversely, they may not be as interested to know that their system is vulnerable to an air strike from a foreign government. When creating the adversarial model, the following attributes must be described in detail:

- *Time:* Any time limits imposed on the attacker should be stated. This includes any absolute time limit that the attacker may have to observe system functionality as well as any relative time outs imposed by the system. For example, a web-based system will generally allow a user to use the system as long as they wish (absolute time limit) but once he/she has logged in to the system, the session associated with that user is only valid for 60 minutes (relative time limit).

- *Equipment:* Computing resources need to be considered when describing the attacker. Descriptions of computing resources do not necessarily need to include actual computer specs. Rather, any specialized equipment should be mentioned such as super computers or wireless scanners and detectors.

- *Software Tools:* Software tools, much like equipment, will correspond largely with financial resources. There are many software tools available both commercially and free of charge. An individual may generally use free tools, commercial enterprises may pay for commercial tools, and governments may hire development teams to create their own custom tools.

- *Money:* Financial resources are considered because they may be used to increase other attributes in the adversarial model. If the attacker considered is well

funded, he/she will be able to pay for knowledge, equipment and tools.

- *System Knowledge and Access:* If the attacker is assumed to have any knowledge of the specific system being evaluated, that knowledge should be outlined. This will necessarily be the case in systems where the attacker is assumed to have a valid account or where the attacker is assumed to have access to a network. This attribute is especially important as it will make it easier for the testers to focus on more relevant areas of the system. For example, in the AAP system, it can be assumed that the student user is able to access the local network of the university without compromising a host from outside the network. Conversely, the stakeholders may wish to know what an outside adversary can accomplish without any valid credentials.

- *General Knowledge:* General knowledge includes assumed knowledge of the attacker in any relevant discipline. This can include possible education in or knowledge of information security and any other relevant experience.

- *Risk Tolerance:* The risk tolerance attribute is used to describe the amount of risk the attacker is willing to undertake while attacking the system. An attacker outside of the jurisdiction of the host that he/she is attacking will have a higher risk tolerance than a student attacking his/her own university since the student has the potential of being reprimanded if caught.

- *Objectives:* Not all attackers will attempt to compromise a system for the same reasons. Some will try to access assets they should not be able to access while others will be content with rendering the service unusable. A general description of the objectives of the potential attacker should be included in the adversarial model as well.

3.1.3. Known Limitations

If there are any limitations on any aspect of the security evaluation, they should be

explained in detail. There may be particular hosts that the stakeholders don't want tested. All of the hosts which are subject to testing should be outlined for the testing team to ensure only authorized tests are performed. Another area that may need to be limited is the types of attacks and tools that may be used. The rules of engagement regarding the use of social engineering, DoS attacks, use of back-door or trojan software, and experimental or potentially unsafe tests performed by vulnerability scanners need to be outlined in detail. When performing host vulnerability scanning, the testers may discover that a particular service running on one of the hosts has a buffer overflow flaw which could allow an attacker to take control of the host. If the testing team were to perform this attack, it is likely that the affected process will be brought down causing an inadvertent DoS attack. In many cases, the organization will not want the testing process to interfere with daily operations. This means that those aforementioned attacks which have the potential to interfere with the normal operations of the organization need to be avoided. Attacks that aim to cause a DoS as well as social engineering certainly have the potential to interfere with daily operations and need to be addressed explicitly. However, if social engineering and DoS attacks are not being tested for explicitly, they should still be considered where they are relevant. For example, in the case where the testers are attempting to exploit a potential script injection vulnerability and this test results in the affected process crashing, this should be noted and brought to the attention of the stakeholders.

Before moving on to the next phase of the testing process, all of the information regarding the threat model, information provided by the organization, and the adversarial model should be presented to and approved by the stakeholders to ensure that both parties (the stakeholders and the testing team) are on the same page. Once the threat model has been agreed upon, the testers may move on to the next phase.

3.2. Information Gathering Phase

Although the testing team may have a significant amount of information from the previous phase, it is still necessary to perform further reconnaissance. During the information gathering phase, the goal is to obtain as much information about the target as possible. There are two types of reconnaissance activities which may be used to obtain

information: active reconnaissance and passive reconnaissance. In this report, active reconnaissance is used to describe any activity that probes the hosts on the target network directly while passive reconnaissance describes activities which attempt to gain information about the target from publicly accessible sources not directly linked to the target network or host.

3.2.1. Passive Reconnaissance

Most works on penetration testing only consider active reconnaissance when performing information gathering. Although active reconnaissance will provide the most updated view of the target network in terms of each host machine's fingerprint (IP, open ports, services running, OS), there are many online sources which may also provide valuable information about the target as well. One tool in particular which allows users to query a number of these sources from a single application is the Sam Spade tool [81]. Spade offers both a Windows GUI interface as well as an online interface for non-windows users. The tool allows users to query databases such as the ARIN Whois database [6] which will allow the testers to determine the IP block of the organization as well as contact information for administrators and DNS servers. Spade also allows testers to PING hosts, determine if the host has been blacklisted due to spamming activity, perform traceroute queries, and find any aliases which may also be used by a particular host. If any hosts are discovered in the target network that were not mentioned during the initial scoping phase, this information should be brought to the attention of the stakeholders and approval should be given if they want those hosts to be part of the testing procedure. Another interesting service offered over the Internet is the web-server search provided by Netcraft [71]. This tool allows testers to enter the URL or the IP address of the target host and the tool will provide a summary which includes the organization which owns the IP block the host is located in as well as a complete history of the OS that the host is running, and the web server software that the host is running. Another feature of the Netcraft database is that a list of public facing web servers found in the same IP block can be obtained as well.

In addition to querying public databases to obtain network information, other sources such as employment web sites and developer forums and mailing lists may also be used to

obtain valuable information. Many organizations advertise job openings on large job sites such as Monster.ca or Workopolis.ca. By searching for job openings posted by the target organization the testers may be able to get a better idea of what technologies the target applications may be developed with. The testing team may also be able to obtain job posting information from the organization's own website as well. Many businesses will keep an online database of their current openings which can reveal some potentially useful information about network infrastructure and software technologies used. In addition to this information, searching for posts made by developers from the target organization to mailing lists or newsgroups may also leak information about potential weaknesses in the system. Sometimes developers will post code snippets that they are having problems with or just detailed descriptions of particular features which are posing problems as well.

3.2.2. Active Reconnaissance

Although passive reconnaissance can yield useful information about underlying technologies used and information about the target network, it must be combined with more direct (active) techniques to get an accurate description of the target hosts and applications. The goal of performing active host reconnaissance is to create a machine fingerprint for each relevant host on the target network. Machine fingerprints are described in detail in chapter 2. We extend this idea a little further in this section so that more useful information is provided to the testing team which will allow them to test each host more efficiently. The machine fingerprint will allow the testers to narrow down the scope of attack for each host by only attempting to exploit the relevant services. The machine fingerprint defined here should include the following information for each host:

- Publicly accessible host IP address
- Host name and aliases
- Operating System
- Open Ports
    - Most probable service running on the port
    - Other possible services running on the port
    - Software package used to implement the service
    - Software version information and patch level
- List of all custom applications running on host

Some of this information may be available from the passive reconnaissance activities but that information should be verified by a more active technique. The target host IP and host names can be obtained using Sam Spade as described above. Once the IP addresses of all the target hosts are known, the open ports that are accessible on each host can be determined by performing a port scan such as NMAP. Details about NMAP and port scanning techniques in general are provided in chapter 2. A number of different types of scans should be performed in order to determine if there is a firewall in place and what types of traffic are allowed through. A simple connect scan or SYN scan can be performed initially to determine which ports are accessible through the firewall without any additional manipulation. The list of open ports should be documented for future use. Some additional tests may be performed to determine if it is possible to bypass the firewall and gain access to services that should only be available on the local network. For example, some firewalls will allow traffic through if the source port is set to TCP port 20 or UDP port 53 [77]. If performing these types of attack yields access to services which should not be accessible then this behaviour should be noted.

Once a list of accessible ports is compiled, the testing team should determine the most likely service running on each port as well as any alternatives in the case where it is not absolutely certain which service is running. Security scanners such as Nessus offer functionality that can help determine which services are running on which ports. Many services will present the user with a banner upon connecting which can be used to identify the service running as well as the software used to implement that service, the version, and possibly the underlying OS. Another way to detect the service running on a particular port is to connect to the open port using a Telnet client. Many services will respond to the successful connection attempt by sending a banner indicating the service name, version, and sometimes the OS type and version. Although banners may not always display valid information, they can be used in combination with other techniques to obtain an accurate identification of the running service. Other tools such as web proxies can be used to obtain more detailed information about web-based applications. One such tool is WebScarab [107]. WebScarab can act as a man-in-the-middle (MITM) proxy which allows the tester to observe

all communications between the browser and the server as well as intercept and change requests and responses. This tool is especially useful when performing application pen-testing as it provides the tester with a scripting interface to automate certain tasks and the necessary functionality to create and send manual HTTP requests to web applications as well as a number of other functions. In terms of reconnaissance, WebScarab allows the tester to view responses from the server which, in a lot of cases, will contain information about application server software being used as well as the host OS. In the case where the service cannot be precisely identified, the most likely service running on the port should be noted as well as possible alternatives. In the case where a large number of unknown services are running, it may indicate that the host is infected with some sort of malware or that the host is not protected by a firewall. This should be illustrated in the overall system map produced at the end of this phase.

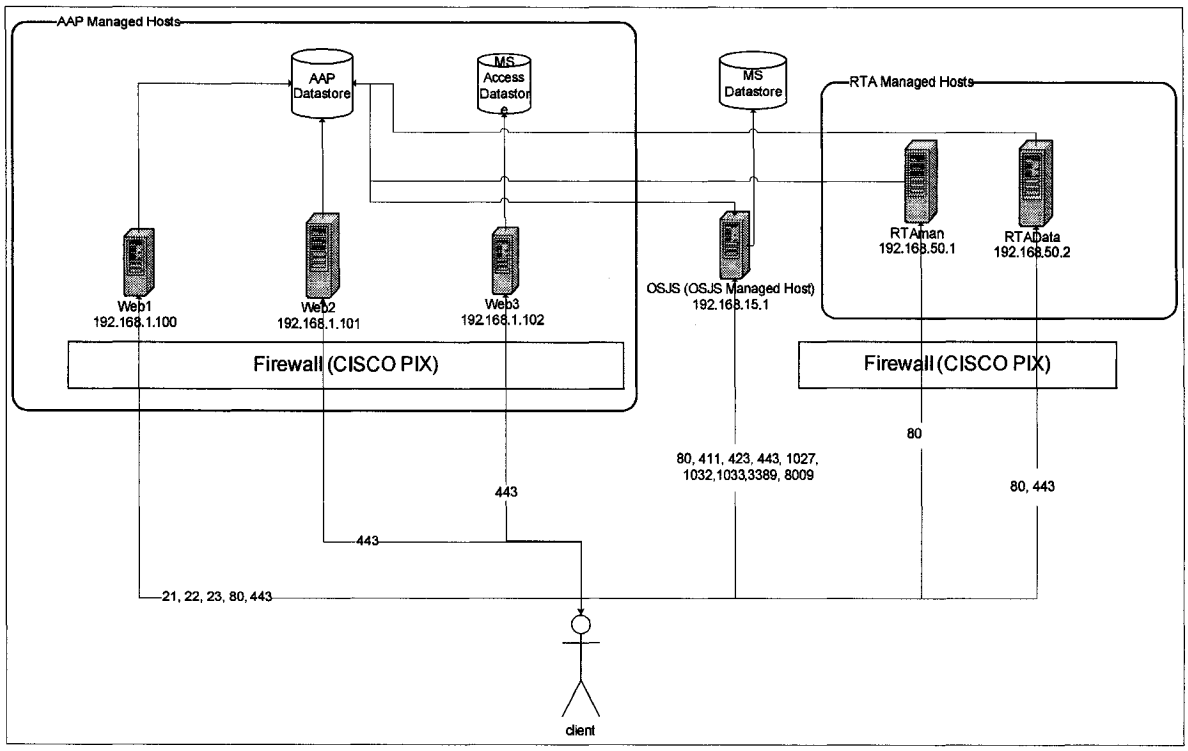
The final step in creating the machine fingerprint is to determine which custom applications are running on each host. Custom applications include web-applications deployed in whole or in part on the host as well as any web service components which are accessible to the Internet. In order to determine which web-applications are running on which hosts, a web crawler application can be used to download a local copy of the web site while making note of which host each page was obtained from. Additionally, if test accounts have been provided, those accounts should be verified by using them to access the system and ensuring that all allowable actions may in fact be performed. This may be done using some sort of deep crawling tool but it may be beneficial to perform this step manually. This will allow the testers to ensure that they have access to all the resources they should have access to as well as allow them to become familiar with the general intended use of the system. As the testers navigate the applications, they should make note of any functions which are not available and, if the use of any part of the system is unclear, its use should be cleared up as well. It is important that the testers know how the system is supposed to work before attempting to exploit it. Once all of the relevant custom applications have been mapped to a host in the target network, all of the information for each host can be presented in the form of a machine fingerprint as follows:

Table 3.1: Sample Machine Fingerprint

Host Names: web1.sample.ca		**Host IP: 192.168.1.100**	**OS: Red Hat Linux 6.1**
Open Port	**Service Likely Running**	**Other Possibilities**	**Software Package and Version**
21	File Transfer Protocol (FTP)	None	Not Available
22	Secure Shell (SSH)	None	OpenSSH 2.9.9/Protocol v. 1.99
23	Telnet	None	N/A, OS Info Provided in banner
80	HTTP Server	None	Apache 2.0.54/mod_perl/2.0
443	HTTPS (SSL) Server	None	Apache 2.0.54/mod_ssl/2.8.28
AAP Applications Hosted			
- initial login - change password - check grades - check registration			

After all of the machine fingerprints have been compiled, a high level system diagram should be created. This diagram should include all hosts accessible to the Internet including host name, IP, and open ports, as well as any information about firewalls in place and information about the DBMSs being used in the back end. In the case where some hosts are managed by different sub-groups of a particular organization, the subgroups should also be noted. Since not all groups within an organization will necessarily abide by the same information security policies, the boundaries between these groups should be noted as they represent trust boundaries in the system. This information is valuable because it may allow testers to find weaknesses in a related system which may help in compromising a more secure target system. Figure 3.1 shows a sample high level network diagram that would be created based on a set of machine fingerprints in a target system. In describing the components we make use of the sample AAP system previously described.

Figure 3.1: Sample High Level Diagram for AAP System



3.2.3. Perceived Policy Generation

Once all of the information about the system is gathered and all the valid use cases have been verified, an overview of the security policy which governs access to the assets on the target applications should be described. The policy description should be based on the observed behaviour of the system when executing all of the valid use cases. For each class of user described in the initial threat model, a detailed description of all possible actions those users can perform should be provided as well as a description of the access control policy. The access control policy will include details about how the user authenticates him/her self to the system, what systems the user is authorized to use once authenticated, and any time limits or other constraints placed on the user. The details of the access control policy should be described in an implementation independent way. The mechanisms used to enforce and implement the access control policy should be described in detail following the description of the high level policy in place. A sample policy description as well as policy

implementation details are provided in Appendix A. Although this information is covered to some extent in the scoping phase, the testers will be able to produce a more accurate description of what a valid user can and cannot do with the system. The goal of the access control policy is simply to outline what users in the system are supposed to be able to do and how these privileges are realized or restricted.

3.3. The Vulnerability Testing and Analysis Phase

After all possible information about the target system has been obtained and all valid use cases verified to be executable, the vulnerability testing and analysis phase may be started. The testing and analysis phase can be broken down into two subsections: configuration analysis and application level analysis. These two sections correspond directly to activities associated with host penetration testing and application penetration testing respectively that were presented in chapter 2. The activities associated with these two phases are described in further detail in the following sections.

3.3.1. Configuration Analysis

Given the information from the previous phase, the testing team will search for potential vulnerabilities in the deployed services on each relevant host. Determining whether a host is relevant in the context of the security evaluation is largely dictated by the scope of the evaluation agreed upon in the first phase. Because flaws discovered during the configuration analysis will generally lead to an attack which will allow the host to be compromised and consequently give an external attacker access to local network resources, an assumption that the attacker already has access to the network may be relevant. If the attacker already has access to the target local network through legitimate means, it may be unlikely that this attacker will start by attacking hosts from outside. There may also be stipulations put in place by the target organization which would state that certain risks are acceptable and that certain hosts are not to be scanned. These types of limitation must be taken into consideration as well. In the case of the AAP test system, it is assumed that the adversary considered already has access to the local network. This means that scanning hosts which are not directly related to the operations of the web-application tier may be

omitted in this case. If the adversary is not assumed to have any knowledge or legitimate access to the local network, it stands to reason that all hosts that are accessible from the Internet become relevant. Regardless of whether the proposed adversary is assumed to have access to local resources, a configuration analysis should be performed on the hosts which are running any components of the web tier. This is due to the fact that if these hosts are compromised then the resources associated the web tier will also be compromised including any configuration information and any source code stored on the server.

Although it is possible for the testing team to manually search for potential host level vulnerabilities in a deployed service and then test the service manually, testing teams will usually make use of some automated vulnerability scanning tools to perform an initial scan. Vulnerability scanners such as Nessus [70] allow testers to check for a large number of known vulnerabilities in deployed services. The output from the scanner will provide the testers with details about potential vulnerabilities but the testers will need to verify the results manually. Since a large part of the known vulnerabilities in commonly deployed software are buffer overflow vulnerabilities, some care should be taken not to crash the affected processes. Many organizations may not want the testing team to interfere with daily operations so if actively exploiting potential vulnerabilities will interfere with the running process, the testing team may only be able to tell the organization that the vulnerability may exist based on software version information and publicly available vulnerability information. Another approach which would help to produce more accurate results would be to take the results from the automated tools and set up a test host using the same software deployed on the target. This would allow the test team to try to actively exploit the potential vulnerabilities that were found. Setting up a test host with the same configuration does require more resources such as spare machines and it may not always be easy to find the exact versions of the software deployed on the target host. For example, the Apache Software Foundation [109] only makes the most recent major versions of their software available. If a client is running an older version of the Apache web server, it may not be available to install on a test host.

In addition to searching for potentially flawed deployed software, vulnerability scanners can also be used to help locate vulnerabilities in the web tier which stem from

common web server configuration errors. Some web servers, if not configured correctly, will allow users to list the contents of the root web directory. This could allow attackers to obtain server side source code and any another data stored in the root web directory. In addition to configuration errors, automated scanning tools will also help to uncover potentially "hidden" administrative applications. It is not uncommon for website or system administrators to implement some kind of hidden interface which they use to manage their sites. The flawed assumption here is that if there is no public link to the administrative site that no one will access the application. This is a classic example of "security through obscurity". There are a number of scanners available, such as Nikto [73], which will perform a scan for known vulnerable server-side scripts as well as any hidden directories. The hidden directory search is similar to a brute force attack on a login screen. Given a base URL, the scanner will append directory and file names from a dictionary list, send a request for the generated URL, and wait for a response. If the scanner successfully locates any hidden directories or applications, they will be displayed and the testers can document them and attempt to exploit them. Once the configuration analysis is complete, the findings should be documented for each service running on each host. The findings for each service should include all relevant information regarding any vulnerabilities found, notes on the potential risks associated with having the service running, and possible mitigations for any weaknesses found.

3.3.2. Application Level Analysis

Application level testing is performed once the configuration analysis phase is complete or near completion. Unlike the configuration analysis, the application level analysis is used to identify application level vulnerabilities in the web-based applications deployed. There are two main types of testing that can be performed to try and uncover security bugs in web-based applications: black box testing techniques and white box testing techniques. Both of these approaches are adopted from the more general quality assurance/software testing field.

In a black box approach, each application is treated as a black box that takes some form of input and produces some form of output in response. In a traditional testing

environment these tests can be performed using automated testing suites. Although some security tests can likely be done in a similar fashion, at times, more detailed analysis of application responses is needed so more human interaction is needed in performing these tests. This allows the tests to be modified accordingly in order to find application errors which could lead to security vulnerabilities. In performing black box security testing on a web-application there are a number of input locations that need to be considered. These include the following:

- URL encoded parameters
- HTTP cookies
- HTML form inputs
- hidden form fields
- HTTP header information

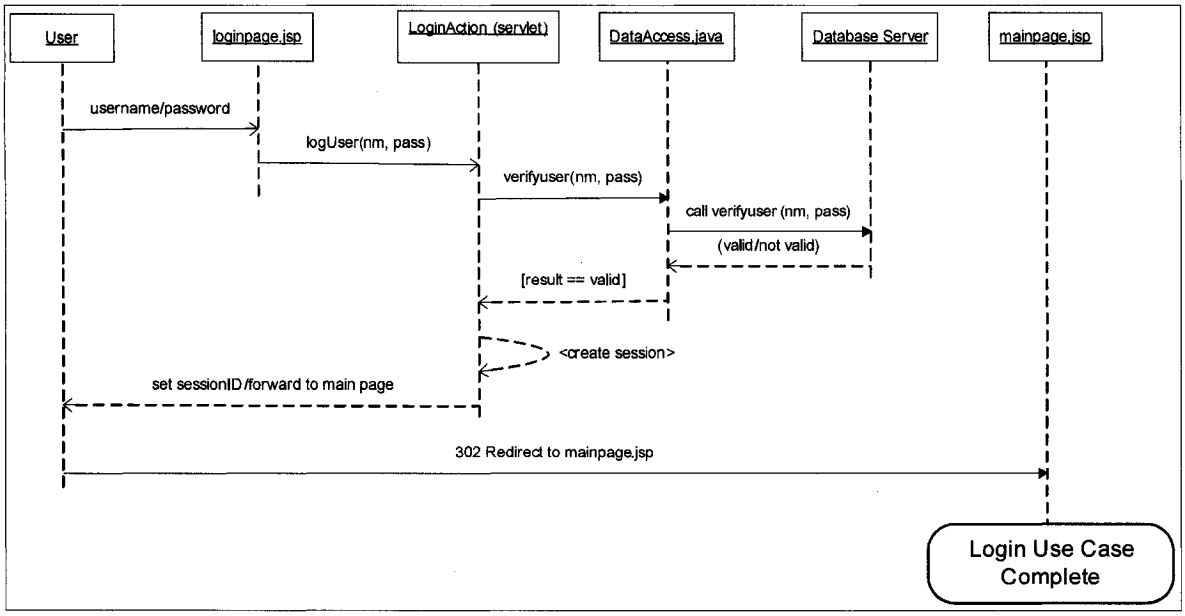
These can be used to inject malicious data into the application which can then be used to extract sensitive information or cause the system to perform in some unintended way. Malicious data can be entered into HTML forms through a regular web browser; however, these forms may be subject to client side data validation routines. It is also inconvenient to try and manipulate data in the other attack vectors mentioned above using the browser only. It is also possible to inject malicious data into any of the above data areas by creating all requests to the server manually using a program like a Telnet client. This method requires a detailed knowledge of the HTTP protocol as well as implementation details of the web-based system being tested. This technique can prove to be error prone as all requests need to be typed in manually and HTTP requests can vary from application to application. A third solution, and arguably the most flexible and easy to use, is to make use of a MITM web-proxy application like WebScarab [107]. Web proxies will allow the testers to create manual requests from previously observed requests, replay previously seen requests, and manipulate HTTP requests on the fly. This provides an easy way to change any of the data areas mentioned above. The testers should try to execute the web application attacks described in chapter 2 as well as the attacks related to session hijacking to be discussed in the following chapter.

The black-box approach is generally the more accepted approach when it comes to

testing web-applications but that does not mean that white-box techniques should be completely ignored. If the source code for a system is available, it should be analyzed. Since most web-application security vulnerabilities can be linked to implementation flaws, it is important that the code be analyzed for possible exploitable flaws as well as insecure coding practices which could lead to a breach if discovered. Source code analysis is a relatively quick way to determine how (or if) input data is being sanitized as well as determine if there are any hard coded backdoors which may be exploited. Sometimes developers will insert checks which will allow them to bypass certain parts of the system that should always be executed in a production environment. This could be done for any number of reasons but should definitely not be left in the production version of the system. Therefore any hard-coded backdoors should be noted. After an initial scan of the code is made, each of the use cases performed in the information gathering phase should be mapped to a UML sequence diagram based on the available code to help analyze data flow in the system. Other diagramming techniques such as data flow diagrams can be used instead of UML diagrams. Sequence diagrams were chosen in this case because it is easier to represent the individual use cases in terms of implementation details. When creating the sequence diagrams, separate time lines should be added for each accessible page and servlet involved, any RPC calls made on other hosts, any database queries, and any other object calls which seem relevant. These diagrams will allow the testers to better understand the data flow and inter-host communications involved in each use case. Other attributes which should be noted in the diagrams are any locations where the user is authenticated (either by means of login or session verification) and any location where sensitive information is accessed. This type of analysis is quite beneficial as it allows the testing team to uncover vulnerabilities that would possibly go unnoticed using black-box testing alone. More specifically, the sequence diagrams allow the testers to find potential flaws in how authentication is performed or if it is performed at all before executing a restricted action. This technique is also useful in finding time-of-check-to-time-of-use (TOCTTOU) type vulnerabilities. By analyzing the data flow of the application, it is possible to determine if there are flaws in the system that, for example, set session variable values before authentication is performed or that rely on a particular access sequence of the pages involved. Web-application developers should never

assume that the pages in a particular application will be accessed in the expected order. A sample sequence diagram illustrating the login use case of the AAP system is shown in Figure 3.2.

Figure 3.2: Sample Sequence Diagram Illustrating Login Use Case



As vulnerabilities are found, a bug report should be written up detailing the flaw discovered and the potential risk associated with that vulnerability. This should be done for each vulnerability found in the system. By creating individual, self-contained bug reports, the task of implementing mitigation strategies can be divided more easily amongst the development team of the organization at a later time. Since numerous layouts for bug reports have already been established by organizations such as CERT [13], we made use of this layout when creating bug reports during the security evaluation. The bug report should include information detailing the type of bug it is, a bug ID, a descriptive title, the severity, the root cause, the potential impact if not fixed, possible mitigation strategies, and the date the bug was found. Additionally, other information to justify the report can be provided such as proof-of-concept code and/or screen shots illustrating the undesired behaviour. A sample bug report is provided in Appendix B. One point in the bug report that should be addressed is the issue of severity ratings. Many rating systems that have been used in the past vary a

great deal and the majority revolve around describing host level vulnerabilities. When describing application level vulnerabilities, it would be beneficial to explain what the severity ratings given to a particular bug actually mean. Unfortunately, there is currently no standard severity classification system in place so in the following section, we describe the severity ranking system used during the evaluation.

3.3.3. Application Level Vulnerability Classifications

In presenting the results of the application analysis, the bugs found are classified according to how much information an attacker could obtain compared to how much information about a victim and how much interaction with the victim is require to obtain said information. Traditionally, when describing security vulnerabilities, they are only considered serious if there is an active proof of concept that could allow an attacker to gain control of a host from another, remote host. We believe that in the case of web-based systems, vulnerabilities that allow an attacker to obtain any personal information about other users or any vulnerability that allows an attacker to change information about other users should be considered serious as these types of bugs allow access to protected resources. The following vulnerability ranking system was used for defining the severity of the bugs found at the application level of a web-based system.

> **Very Severe** – A vulnerability is considered very severe if it allows an attacker to access user credentials which could allow the attacker to compromise the account at any time. These types of vulnerabilities include any type of password disclosure or allowing a user to make changes to the longterm credentials (password) of other users. These types of vulnerabilities may require minimal-to-no user interaction and minimal-to-no prior information about the victim. A vulnerability is also considered very severe if it allows any user to write information to another user's protected storage space. This does not necessarily have to be a user credential. If an attacker is able to write data to another user's protected area, he/she may be able to insert a script that he/she knows will be executed while the victim is logged in or, at the very least, overwrite sensitive information which could cause problems for the victim.
>
> **Severe** – A vulnerability is considered severe if it allows that attacker to obtain temporary control of another user's account. In a web-based

systems, this type of vulnerability could result in disclosing the victim's session ID to an attacker. The attacker could then access information about the victim. One such technique is storing user names or other personal information on the client's browser. Some user interaction may be required to exploit this type of vulnerability.

Moderate – A vulnerability is considered moderate if a malicious user can exploit it and then convince the victim to disclose his/her sensitive information to the attacker directly based on the results of the attack. Examples of attacks of this severity level include those where a trap is set by the attacker (code injected into his/her account or session ID fixed) and then the attacker uses some sort of social engineering technique (contacting administrative personnel or e-mailing legitimate users) to entice the victim into accessing the malicious information. This could allow the attacker to steal temporary session ID information or fix temporary session ID information with the direct (and unknowing) assistance of the victim. User interaction is required to successfully exploit vulnerabilities of this severity level.

Low – A vulnerability is considered of low severity if it is not necessarily exploitable in the current system configuration but, should the configuration of the system change, the vulnerability could become more severe. Vulnerabilities of this level of severity could also be flawed assumptions made by developers that show up during the code review but do not directly affect the security of the system at the current time. These types of vulnerabilities could include specific configuration errors which make other, more severe, attacks more likely.

Application level analysis should continue until all use cases have been analyzed. Although it would be ideal to locate every flaw in the system, it is well known that this is not possible. However, by providing the client with the use case diagrams created during the evaluation, it will give them a better idea of the level of testing performed. In addition to the individual bug reports, an index containing the bug ID, the bug title and severity level should also be created. Since the bug reports will likely make up the majority of the report, it will make it easier for administrators to find what they are looking for faster and allow upper level managers to see at a glance the types of flaws that were found. After the analysis phase is complete, the testing team will move on to the final phase of the process, the post-audit phase.

3.4. The Post-Audit Phase

Once all of the testing activities have been completed, the testing team must provide the organization with a final presentation of their findings. Since all of the details have been produced during the testing phase, the goal in the post-audit phase is to summarize the results and provide an overall evaluation of the organization's security posture. The goal of the summary is to outline any general aspects about the system which could compromise its security as well as the potential consequences of those compromises. These could be flawed details or assumptions in the organization's security policy, broad insecure handling of potentially sensitive information, or poor software patching/updating practices. Since the details of specific vulnerabilities as well as potential attacks have already been compiled during the testing phase, the evaluation summary should focus on showing managers, who are presumably the decision makers that will enact the mitigation plan, the potential consequences of not mitigating potential threats to the system. Demonstrations of some of the potential attacks on the system can be a good way to convey this information. The summary can also include statistics regarding the number of bugs found as well as what proportion of those bugs the testing team considered to be serious (e.g. "severe" or "very severe"). This type of information is useful because it will give the administrative team a better idea of where to start to mitigate the potential threats.

In addition to summarizing the work that was performed during the evaluation, the post-audit summary should also include future considerations as well as recommended work to be done in the future. These recommendations should try to explain how the numerous types of flaws were introduced in the first place as well as recommendations as to where mitigation should start. Mitigation strategies should be descriptive enough so that the management team knows what the problem is and what needs to be done to solve it. They should not attempt to dictate how implementations should be done or dictate which technologies should be used. Since security is a never-ending battle, regular security evaluations should be scheduled into the organization's software development and network management activities. Future evaluations will allow the organization to determine whether previously discovered flaws have been adequately mitigated as well as increase the chances

of finding other, previously unknown bugs. The testing team should also make note of any limitations that were not worked out during the evaluation which may have hindered testing of certain systems. Some of the limitations encountered during the evaluation performed in this thesis are covered in section 5.3. The final report submitted should outline all of the hosts and applications covered as well as those not covered. This will allow future evaluation teams to focus on the parts of the system which may not have been tested as thoroughly.

Since the development team tasked with implementing mitigation solutions the problems found during the evaluation may not have the same in-depth security knowledge as the testing team, it is important to arrange a follow-up meeting with the development team. This should be done after the development team has had a chance to review the results of the evaluation. This will allow the development team to clarify any details in the report that may not be immediately understood. Additionally, the development team may have reasons why a particular mitigation solution suggested will not work. In this case, the development team and the testing team can work together to produce a viable solution that can help to mitigate the problem found while still being feasible in the specific context. This is beneficial because the development team will have a more detailed understanding of the system which can be combined with the in-depth security knowledge of the testing team.

3.5. Comparison To Previous Testing Methodologies

In order to compare our methodology to previous approaches (presented in section 2.1), we must consider the requirements the methodology must fulfill. The requirements that we considered to be of utmost importance are described in this section.

The first requirement that needs to be addressed when performing a security evaluation is the scope. It is important to determine the scope of the evaluation in order to appropriately allocate testing resources and provide the client with results that are tailored to their concerns. In defining the scope, the type of attacker and their initial level of access need to be considered. Therefore, it is important to be able to perform the evaluation from the perspective of an outsider with no access as well as a trusted user with some form of additional access as well. Related to scope is the ability to take into consideration any

restrictions placed on the testing team during the evaluation. The amount of resources in terms of available documentation, source code, and network diagrams will tend to vary among organizations. Therefore, it is important to be able to incorporate whatever resources are available and be able to adjust the testing strategy accordingly. Since the system being tested in this thesis is a web-based system, it would be beneficial to to have a methodology that specifically designed for testing web-based systems. Since we are performing a third-party evaluation, the methodology should be designed to be used by a third-party security team. In order to be generally useful, the evaluation methodology must be repeatable. Finally, the methodology should allow the testing team to provide potential solutions to help mitigate the problems found while testing the system.

Based on these requirements, a comparison of our methodology to previous work is provided below in table 3.2.

Table 3.2: A Comparison of Third-Party Testing Methodologies

Methodology	***Allows scope to be defined***	***Allows restrictions to be defined***	***Ability to handle varied resources***	***Ability to test from an outside perspective***	***Ability to test from varying levels of trust***	***Repeatable***	***Designed for Third-party security evaluations***
FHM				X	X	X	
OSSTMM		X	X	X		X	X
NIST 800-42	X			X		X	
TRIKE	X			X	X	X	
STRIDE/ DREAD	X					X	
Our Proposed Methodology	**X**	**X**	**X**	**X**	**X**	**X**	**X**

3.6. Summary

The security testing methodology used during the course of the evaluation performed during this thesis is made up of four phases: the initial scoping phase, the information gathering phase, the vulnerability testing and analysis phase, and the post audit phase. In the initial scoping phase, the testing team meets with the client in order to determine what

resources are available and to obtain the necessary resources and access to perform the evaluation. Once all of the necessary resources have been obtained, the testing team will need to work with the client to create the threat model (i.e. what assets is the organization trying to protect, what classes of users exist in the system, and what are those users allowed to do) as well as the adversary model (i.e. what resources does the attacker being considered have available to him/her). Finally, if there are any testing restrictions such as certain hosts that should not be tested or certain types of tests (such as DoS or social engineering attacks) that should not be performed, those should be identified at this point.

In the information gathering phase, the testing teams goal is to obtain as much information as possible about the target system. This is done using passive reconnaissance and active reconnaissance techniques. After performing active and passive reconnaissance, the testing team will have a detailed view of each host on the target network as well as the fairly detailed view of the target network as a whole. Using the test credentials given by the organization, the testing team should ensure that those credentials actually do provide access to the systems that the organization intended them to have access to. This will allow the testing team to become familiar with how the system functions. From the user credential test, the perceived security policy of the system can be generated which will provide details regarding allowed access and access controls in place as well as any discrepancies between what the organization claims a user class can do and what that user class can actually do.

The vulnerability testing and analysis phase consists of two parts: the configuration analysis and application analysis. The configuration analysis is used to uncover any vulnerabilities or configuration errors in services running widely used software. This is done using a variety of security scanners and the results of the scan are verified by the testing team in order to help rule out false positives. The application analysis is performed on all custom web-based applications. When design documentation or source code is available a white-box testing approach can be used where as if no documentation is available, a black-box testing approach will be used. For each vulnerability that is uncovered, a bug report is generated consisting of details about the vulnerability, explanation of how it can be exploited, consequences of not mitigating it, and some suggestions as to how to mitigate the vulnerability.

The final stage of our methodology is the post-audit phase. The goal of the post-audit phase is to provide a summary of the evaluation findings and an overall evaluation of the security posture of the systems tested. The testing team should provide suggestions for high-level mitigations solutions and address any follow up actions that should be undertaken. Follow up actions can include a follow up schedule with the organization, follow up evaluations in order to ensure that mitigation solutions put in place actually solve the problems found, address any problems that may not have been solved during the evaluation, or perform a similar evaluation based on different scope (different systems or attackers considered for example).

Since most web-based systems rely on some form of session management system to allow users to maintain their authenticated state after logging into a web-based system, the focus of the next chapter is on attacks which target session management system and the techniques used to help make these attacks harder to perform and less damaging if they are performed.

4. Session Management and Attacks

This chapter will cover a brief history of the use of sessions and session identifiers in contemporary web-applications. This includes a description of the original motivation behind this mechanism and the threats introduced when this mechanism is used for access control, authentication, and authorization purposes. Next a number of possible best practices that should help to mitigate some of the threats posed by the use of weak session identifiers will be covered. Finally, a more advanced solution is described which can be used to help make session hijacking more difficult should the identifier be disclosed.

4.1. History of Sessions and Cookies

HTTP [25], the protocol generally used by web-applications to communicate over a network, is a stateless protocol. This means that requests are made by a client application independent of one another and the responses to those requests sent by the server application are also independent of one another. Consequently there is no persistent connection between the server and the client and therefore, the server does not store any personal information about the client for later use. If the world-wide-web were simply used as a big lookup table for information this protocol would be fine. A user would request the information they are looking for and the appropriate server would send that user a static web-page containing the requested information. However, the world-wide-web has evolved into a global marketplace where users can make use of customized services such as e-mail and message boards and make purchases from retailers all over the world. It is easy to see that in an environment such as this, the applications providing these services need a way to retain a user's information between page requests, otherwise applications such as online shopping sites would be next to impossible to create or would be extremely user unfriendly. Therefore a mechanism was developed to help applications maintain user state information to make applications such as online shopping carts possible. This mechanism was fairly simple: allow the web server to store a small piece of information on the client's machine (i.e. a cookie) which can later be used to personalize that user's browsing experience.

The idea of using a cookie to store a session identifier was first introduced by Netscape Communications as a way to implement an online shopping cart application [38]. A formal specification produced by the IETF soon followed based on Netscape's original idea and was published as RFC 2109 in 1997 [53]. The specification was later updated in 2000 and can be found in RFC 2965 [54]. Essentially, a cookie is a piece of information stored by the client's browser that is passed back to the server which set it in order for the client application to maintain a desired state with the server. A simple example which illustrates this is a server which asks a user to enter his/her name and then addresses the user by name in all subsequent pages. Although a specification for cookies exists, this specification does not dictate what type of information is to be stored within a cookie. This means it is up to the application programmer to use this tool wisely. Unfortunately, there is nothing stopping a web-application developer from storing your personal information in your browser in the form of a cookie. This has been the center of many debates regarding Internet privacy. These debates are outside the scope of this thesis but information regarding some of the cases can be found in [38]. Although it is possible to store user information such as user ID and password in a cookie to allow a web-application to re-authenticate them at a later time, this is generally not done as this leaves the user open to a number of attacks. For example, if the site does not use an encrypted connection, the cookie names and values are sent in plain text format in every request the user makes to the corresponding server. This practice is even discouraged in section 7.4 of the HTTP state management specification shown in the following statement:

> *While it is common practice to use them in this way, cookies are not designed or intended to be used to hold authentication information, such as account names and passwords. Unless such cookies are exchanged over an encrypted path, the account information they contain is highly vulnerable to perusal and theft. [54]*

As an alternative, web-application developers set a cookie in the client's browser which stores a value known as a session identifier or session ID. This common practice makes web-application programming easier for developers and makes systems more usable

for end-users and provides a way to manage sensitive information on the server without involving the client.

Although there are a number of advantages to using this type of architecture when creating web-applications, there are a number of design considerations that need to be examined in order to ensure that the session ID cannot be compromised. If the session ID is compromised then a malicious user may be able to gain access to other sensitive information located on the host.

The session ID is only valid for the duration of the end-user's session. Once the user logs out of the system or closes his/her web browser, the session, as well as all information associated with it, should be destroyed. In theory, this seems like a reasonable solution; however, in practice there are a number of other factors that need to be considered if this mechanism is going to be used for secure applications (these will be explained in section 4.2). What should be noted when considering some of the common security problems associated with improperly used session IDs is that after the user logs into a secure section of a web-application, the user is assigned a session ID which is essentially a temporary password allowing the user to access certain restricted information and functionality. Attacks that can be performed on this session ID are summarized in the following section.

4.2. Common Attacks on Session IDs

Before examining the properties that make session IDs vulnerable to attack, we will describe the common classes of attacks that can be launched against session IDs. The end result of all the attacks considered is the same for all categories: the attacker taking on the identity of the victim from the point of view of the server. Once this identity is established, the attacker can potentially access information they should not be able to access and make requests to the system under the assumed identity of the victim.

4.2.1. Brute Force Attacks

In essence, a brute force attack is simply an attack where requests are made to a server with different session IDs until a correct ID is found. Attackers will use any information they can get regarding the session ID to help narrow down their search; however, a straight brute force attack assumes the attacker only has access to certain

properties of the session ID such as length and lifetime. This is the worst case scenario for an attacker; however, session IDs are rarely truly random. If an attacker is able to obtain or deduce any other information about how session IDs are generated, a brute force attack will be far more efficient. An excellent paper on the topic of brute force attacks was written by Endler [21] which explains how a number of production applications can be attacked because session IDs used are too predictable. This unsettling trend is further substantiated by a security advisory released by Netcraft in the same year [72]. The vendors of the products mentioned in the Netcraft advisory have released patches to help solve this problem; however, predictable session ID generation still exists. Information such as this allows an attacker to launch an educated brute force attack, and depending on how predictable the session IDs are, he may be able to guess the ID on the first try. Although some authors do classify brute force attacks and prediction as two different attacks, it is our opinion that prediction is simply a more specific brute force attack. The properties of weak session IDs that make brute force attacks possible are covered in the following sub-sections.

4.2.2. Weaknesses Enabling Brute Force Attacks

Session IDs can be predictable: In a number of applications, session IDs are created using simple pseudo-random number generation functions or, in some cases, the numbers are sequential or some combination of these two mechanisms. Some developers make the assumption that using random functions based on operating system random functions and seeding those functions with predictable values such as user IDs or system clock values is a secure means of assigning session IDs. If the session ID is only being used to keep track of the user's state and is not linked to any sensitive information than this scheme would be fine. In cases where sensitive information is associated with the session ID, it should be less predictable. If session IDs are predictable then an educated guessing attack could be used to guess a valid user's session ID and potentially gain access to sensitive information. This threat is compounded by the following two problems as well.

Session IDs can be too short: Some applications may select session IDs from a session ID space that is too small. A session ID space that is too small can be defined as a space that can be exhaustively searched efficiently in order to find a valid session ID. For

example, if a session ID is made up of alpha-numeric characters and is six characters in length, a brute force attack which tries every possible session ID without any other previous knowledge of the system is not impractical. If this weakness is combined with the predictable session ID selection previously mentioned, it makes guessing session IDs much easier.

Account lockout policies are generally not enforced on session IDs: This particular weakness is the main reason that brute force type attacks can be successful against session IDs. Most web-applications that have a secure section and rely on users entering credentials such as user name and password have some kind of policy in place which only allows the user a certain number of login attempts. If too many attempts on a single account are made then some action is taken to prevent more attempts to guess the user's password. In most cases the account will be locked out for a predetermined period of time or some action will need to be taken by the legitimate user to reactivate the account. This is done to prevent attackers from executing automated brute force attacks on login pages. Although this helps make brute force attacks on credentials such as user name/password combinations more difficult, attackers are still able to perform brute force attacks on session IDs unimpeded. If the session ID is not stored with the user name (i.e., the session ID is the only information submitted to the server to prove the user's identity), it is more difficult for an attacker to target a specific account on the system. However, if the goal of the attacker is to simply gain access to any account on the system, this attack is quite easy to execute and makes it harder for the system to determine which account to lockout when an attack is detected.

Session IDs have long expiration limits on servers: This weakness not only makes brute force attacks easier, it also makes other types of attacks such as session ID interception more successful as the attacker will have as long as the session is valid to access another user's information. Web application developers may offer the end-user a logout function which, when used, will delete the user's session and render the session ID useless. However, if the end-user simply closes their browser, in most cases, the session will persist on the server. The time an attacker has to launch an attack on the session varies based on the session timeout policy used by the web application. For example, if an application has a policy that a session can only last 60 minutes, the attacker has a maximum of 60 minutes to

compromise the session and access whatever information they are after. Some applications have an active session time out, meaning that if there is no activity on a particular session for a certain amount of time, the session is reset and the user will have to login again. If this is the method used to limit a user's session then the attacker has up to the predetermined time out duration to compromise the victim's session. For example, suppose a web application has a policy that states that if a session is inactive for more than 15 minutes, it is destroyed and the user must log in again. In this case the attacker has a maximum of 15 minutes to compromise the victim's session; however, if this is the only policy in place, once the attacker gains access to the victim's session, he just needs to keep the session active for as long as he wants to access the victim's information. This weakness is the main motivation for stealing a user's session ID, as improperly terminated sessions on the server are an easy target for attackers and can be quite common in web based applications.

4.2.3. Session IDs Transmitted in the Clear

In a number of online applications, when users log into the secure section of the application, their credentials are transmitted using a secure (encrypted) protocol such as SSL/TLS. This is done in order to protect the user's credentials from an eavesdropper along the transmission path. However, once the user has logged in, a number of applications will switch back to plaintext mode and transmit the user's session ID in plaintext. This session ID is then transmitted in plaintext with every request that the user makes to the server. This means that an eavesdropper along the transmission path would be able to sniff another user's valid session ID off of the network and replay that session ID to gain access to the user's sensitive information. Session IDs stored as cookies on the client can also be transmitted in the clear if the web-application has both secure and non-secure sections. If the session cookies are not marked as secure or if the domain path is too general then it is possible that secure information could be sent over a non-secure channel as cookies are always sent to servers within the specified domain. This is often overlooked by developers as they may make use of preexisting session management functionality provided by the technologies being used. These session IDs are generally stored in cookies on the client and are not marked as secure or have domain specifications that are too general. An unsuspecting

developer then may use the same technology (and the same server) to handle both secure and non-secure parts of applications. This means that even if SSL/TLS is used in the secure section of the web-application, the session ID will be sent in plaintext if the user navigates from the secure section to the non-secure section of the site.

4.2.4. XSS Vulnerabilities can be used to obtain session IDs

Cross-site scripting vulnerabilities are quite common in web-based applications and can be used to launch a number of different types of attacks. XSS attacks have been covered in detail in chapter 2. In this section, we are concerned with those types of attacks that allow a malicious user to obtain another user's credentials stored on the browser in the form of cookies. Generally an attacker will insert code to obtain the victim's cookie and transmit the cookie to the attacker somehow. This way the attacker knows that the session ID sent is immediately valid and can be used to access the victim's information. Therefore a number of attempts have been made to try and protect session ID cookies from scripting interfaces such as Javascript and VBScript. There are a number of techniques that can be used to help prevent XSS attacks. We will focus on those that can be used to help protect sensitive session IDs in the next section as well some best practices that can be included in the cookies themselves to help lessen the effectiveness of XSS attacks for this purpose.

4.2.5. Poor Implementation Practices Making Session Cookies Persistent

This type of weakness is not often mentioned and so it may be overlooked. A session ID is meant to be a temporary system access code or token good for the duration of the user's session. However, if the session IDs are not handled correctly, they may be allowed to persist longer than intended and leave legitimate users open to attack. To get a better understanding of the common practices that are used to transmit session ID information, the numerous methods of setting and passing session IDs should be described. There are three common means to transmit and store a user's session ID. The first is to set a cookie in the end-user's browser which stores the value of the session ID. This is arguably the most secure way to manage a user's session ID (although some arguments have been made which state otherwise); however, as will be described in later sections, there are a number of attacks that can be used to obtain this cookie value from the user. Cookies can be set in a number of

ways as well. The functionality to set a cookie on the client's machine is present in most popular server-side scripting and programming languages. However, if the web-application developer is not using one of these server-side technologies, there is a mechanism built into HTML that allows the site to set a cookie using META tags. For example, to set a cookie called Name to the value Joe, the following tag could be added to the HTML document sent to the user which will set the cookie in the end-user's browser:

<META HTTP-EQUIV= "Set-Cookie" CONTENT= "Name=Joe">

In this case, since no expiry date is specified, it is assumed that this is a session cookie and will be deleted by the browser when it is closed [39]. This seems reasonable as the end-user does not want the session cookie to persist past the end of the session. Ultimately it is up to the browser to actually delete the session cookie when the user is finished; however, through observation, we were able to determine that the Mozilla Firefox (version 1.5.0.2) and Microsoft Internet Explorer (version 6.0.2900 SP2) web-browsers handle the responsibility quite well. The problem that is overlooked when using the META tag to set a cookie in the user's browser is that, even though the cookie is deleted, the META tag continues to exist within the user's web cache. This means that an attacker can very easily gain access to the victim's computer (for example at an Internet cafe after the victim has left) and go into the browser's cache folder to retrieve the victim's session ID (or any other information that is stored as a cookie and set via a META tag). This implies that META tags, when used to set session cookies, have the unfortunate side effect of making a temporary session cookie a persistent cookie on the client side. If this is combined with an improperly terminated session as described above, this could lead to the session being hijacked.

The second mechanism used to store session IDs on the client side can also be found in HTML in the form of hidden fields. When creating online forms in HTML there is a hidden form element that can be used to store information (such as a session ID) but is not displayed to the user. Some online applications have made use of this mechanism to store session IDs as well as other personal information. This technique suffers from the same

problem as the aforementioned META tag; it makes the temporary session ID stored on the client side persistent. It can then be retrieved by another user at a later time. Although cookie management is a non-issue in this case, the hidden fields can be viewed quite easily by looking at the HTML code itself or by using a tool to display hidden fields inline. A number of web proxies and Firefox plugins have been developed which display hidden fields to the user as well as allow them to change the values of those fields on the fly(see [108], [107]). Applications that use this technique also suffer from a misplaced trust in the end-user. The web-application developer is trusting that the end user will not tamper with the information stored in their browser, which is a flawed assumption.

The third method of passing session IDs from the client to the server is to embed the user's session ID in the URL of the pages that the user has access to in the form of a request parameter. This is supported by the HTTP GET method. This means that the user's session ID will be located in all the URLs that the user can access. Here is an example to help illustrate this point. Suppose the user is accessing a page called info.htm and his session ID is 12345. Then a session ID would be included in the URL as follows:

http://someserver.com/info.htm?sessionid=12345

As is the case with a number of the aforementioned solutions, the user's session ID is stored in the browser's cache file and can therefore be extracted at a later time. Also, because the session ID is passed as an argument in the GET request, the session ID will also be persistently stored in the server's web logs. So if an attacker is able to access the server's web logs, the session IDs are available to the attacker through that channel as well. Similarly, an attacker could also obtain the session ID of another user from the referrer header, should the user access the attacker's site from the secure site. This is because the session ID will be embedded in the URL sent in the referrer header. This is also the most visible of the three methods. Hidden form fields and cookies are hidden from the user during normal use of the system but the URL parameters are displayed in the URL. This makes the session ID value more visible and possibly more prone to attack by curious or malicious users.

Finally, there is the case where the web-application stores the user's session ID in a persistent cookie. Instead of setting the cookie to expire at the end of the session, the system may set the cookie for a certain period of time which will cause the browser to store the cookie on the hard disk where it can be retrieved at a later time.

4.2.6. HTTP Response Splitting

Much like an XSS attack, an HTTP vulnerability in a web-application can be used to inject malicious code which allows an attacker to steal a user's session ID cookie. An HTTP response splitting attack is executed by getting a victim user to submit a malicious request to the vulnerable server. This request will be formatted in such a way that the response to the victim will be include two HTTP responses instead of one. This attack makes use of a poorly checked HTTP redirect (e.g. HTTP response code 302) page to send the victim to another page of the attacker's choice. If an unchecked parameter is present in the URL for the page the user should be redirected to, that can be used to inject a second HTTP response which will be sent and processed by the victim [51]. Although the attacker can perform a number of different attacks such as website defacement (both permanent and temporary) and a number of different types of cache poisoning attacks, we are primarily concerned with this attack being used to steal a user's session token. A more subtle way to use this type of attack is to use a response splitting vulnerability to set a session ID in the victim's browser which could later be used in a session fixation attack, which will be explained in the following section. This type of attack is less likely to be detected; however, it requires more effort on the part of the attacker.

4.2.7. Session Fixation Attacks

A session fixation attack is an interesting attack where the attacker is able to choose the session ID or establish a session, save the assigned ID and then coax another user into using that ID. A server that has a permissive session policy will check to see if the user has a valid session ID. If they do, the server will make use of it. If not, the server will issue another session ID. Contrary to permissive servers, a strict server will always overwrite the existing session ID regardless of whether a session ID is sent or not. Whether a strict or permissive server is used, a session fixation attack is still possible. The initialization is

slightly different. To perform a session fixation attack, the attacker visits the website and obtains a session ID in the case of a strict server, or in the case of a permissive server, the attacker can simply make up a session ID. Next, the attacker sends a link to the victim containing the newly acquired session ID (through e-mail or some other vulnerability present in the application such as an XSS). If the victim follows the link to the legitimate site, it will use the session ID passed by the victim to manage their session. In the case of a strict server, the session ID will be used as the session is valid since the attacker visited the site to create it. Next, the victim will need to authorize him/her self to the system. Upon a successful login, the system will associate the victim's sensitive information with the session ID set by the attacker. If any sensitive information is associated with the session ID from the victim's interaction, the attacker can then replay the session ID in order to gain access to the victim's session and access that information [52].

4.2.8. Session Riding or Cross-Site Request Forgery Attack

Although CSRF attacks (or session riding attacks as they are otherwise known) are a different type of attack, the end result is the same: the victim's identity is hijacked and used to make requests to a vulnerable web-application. The goal of the attacker, in this case, is to trick the victim into sending the request to the web-application from their own browser. If this is done then all cookies stored in the victim's browser for that application will be sent also. So the request will seem to come from the victim's browser and will contain all of the victim's authentication information. The attacker's objective is not to steal the victim's session ID in this case. This attack does not necessarily allow the attacker to gain access to another user's session; however, it does allow them to make requests under the guise of the victim, in much the same way as a session replay does. These attacks are performed by passing a preformed request (generally a URL which triggers a GET request, but it is possible to execute this attack using a POST request as well) to the victim where it is then executed by the victim on behalf of the attacker. The attack link does not even need to be clicked on by the user. If the attacker embeds the malicious request inside an IMG tag, a large number of browsers and e-mail clients will render the HTML in the email automatically which means the link in the IMG tag will execute automatically. To illustrate

this type of attack, here is a simplified example. Suppose an online bank makes use of the following page with the following parameters to make a money transfer from one account to another:

http://onlinebank.com/?amount=200&toaccount=348756

This type of system may be vulnerable to a CSRF attack if the only authentication done for this transaction is a session ID cookie check. To launch the attack, the attacker simply sends the victim a link containing their account number and the amount they wish to transfer to themselves. When the victim clicks on the link (either voluntarily or not), the transfer is made from the victim's account to the attacker's account. This is because, by clicking on the link, the victim's authentication cookie is sent along with the request. At no time does the attacker gain access to the victim's session ID, but the request is made under the assumed identity of the victim and comes from the victim's browser [85].

4.3. Session Management Best Practices

In this section we describe some best practices that, if followed, will help prevent certain session hijacking attacks. The first section describes the properties that the session IDs themselves should have to make them more secure. The next section describes implementation details that should be considered when using session IDs to maintain a user's sensitive information on the server. The "strong" session IDs used are only as strong as the underlying system using them. This section provides additional properties that should be considered in addition to the properties of a strong session ID.

4.3.1. Properties of Secure Session IDs

One main point that should be considered is that, for the duration of the user's session, the session ID is essentially a temporary password assigned by the server which allows the user to access sensitive information. That being said, the session ID should be as strong as (if not stronger than) the user's actual password. It is known that most human end-users choose relatively weak passwords (see [65] for an interesting study on this topic). End-

user passwords tend to be based on words found in a dictionary or some variation thereof. This is because human users generally have a hard time remembering long random strings of characters; therefore, they choose passwords that are easily remembered. The problem with this is that passwords based on words found in a dictionary can be found using a brute force attack much more easily than those that are truly random. Since a session ID is a temporary password handled by the server and client applications, it does not need to be remembered by the human end-user. This means that a session ID that is truly random (i.e., any session ID found in the session ID space is equiprobable) can be used without hindering the end-user. This makes the time required to guess the session ID using a brute force attack a maximum as all possible session IDs are equally likely [10].

> *Property 1: Session IDs should be truly random or as close to truly random as possible. This means that all session IDs should be equiprobable over the session ID space.*

This is not an easy feat to accomplish. A number of libraries have been created which try to offer cryptographically random numbers; however, the underlying problem is the fact the computers are deterministic machines. This means that to produce truly random numbers, a computer needs truly random data from a physical source. There are a number of cryptographically secure pseudo-random number generator algorithms such as Yarrow [49] and Blum-Blum-Shub [11] that can be implemented and used to generate random session IDs. However, some of these algorithms may impose performance penalties that are unacceptable in a web-based application. A number of web-application technologies offer a secure version of a random number generator that can be used to create cryptographically secure random numbers. In Java this can be done using the SecureRandom class which allows the programmer to seed the generator and choose the underlying algorithm or package to be used [86]. ASP.NET offers similar functionality through the RNGCryptoServiceProvider class [78]. In the Java implementation, if the random number generator is not seeded, the system seeds the generator using internal thread-timing information which should be sufficient for this type of application [62].

Simply making a session ID truly random is not the only measure that should be

taken to make session IDs more secure. If a session ID is truly random but only has a length of six and has a session ID space made up of only numeric characters, an adversary could try every possible session ID in the session ID space in a reasonable amount of time. So choosing truly random session IDs makes the amount of time to find a particular session ID a maximum; however, to increase that maximum value, session IDs must also have the following property:

> *Property 2: Session IDs should be long enough so that an exhaustive search of the session ID space is infeasible.*

Session IDs that are produced using functions based on cryptographic hash functions such as SHA-1 would be sufficient in length (digests are 160 bits in length). If session IDs of this length are used then, given the current state of computing technology, it becomes infeasible to exhaust the entire session ID space in general, let alone in the life time of the session on the server.

4.3.2. Implementation Considerations

> *Property 3: User sessions should have a limited lifetime on the server.*

This property is important for a number of reasons. First, if a session has a limited lifetime on the server then it limits the amount of time the attacker has to find a valid session ID. Second, if the attacker is able to find a valid session ID, limiting the session lifetime also limits the amount of time the attacker has to compromise the victim's information. There are a couple of ways that sessions are limited in many web-based systems. One way to limit a session is to have an absolute lifetime. After a predetermined amount of time, the server renders the session invalid and, if the legitimate user is not finished using the system, he/she is required to log in again. Some assumptions need to be made regarding the typical use of the system in order to determine a reasonable session lifetime. From a security perspective, the session duration should be as short as possible to limit the time frame an attacker has to compromise the system; however, from a usability perspective, the session duration needs to be long enough to allow the legitimate users to accomplish their tasks without having to log back into the system multiple times. This makes choosing an absolute session duration

difficult. Generally system designers will err on the longer side so as to not interfere with the end-users as much as possible. This means that the attacker may have more time to launch an attack.

Another way system designers try to limit session lifetime on the server is to take a more interactive approach. Instead of having an absolute session duration, the lifetime is determined by the actions of the end user. If a particular user has not performed any action in a given time period (generally much shorter than the duration of an absolute timeout), then the session is rendered invalid and the user is required to reauthenticate. This implies that as long as the user is actively using the system, his/her session will not timeout. Although this technique allots less time to an attacker to compromise the system, if the attacker manages to compromise the system within that limited time frame, he ultimately has a session of unlimited duration so long as he/she keeps the session active.

A better solution would be to combine the two aforementioned techniques. In combining an absolute timeout with a second interactive timeout, the session would be less likely to be compromised and if it is compromised, the attacker still has a limited time to access the system under the victim's identity. Although this measure alone does not prevent a session hijacking attack, the interactive timeout limits the amount of time the attacker has to launch the attack and, should the attacker successfully gain access to a valid session, the absolute timeout will limit the amount of time the attacker has to compromise the system further.

Property 4: Session IDs should always be transmitted over a secure channel.

Although this property seems self explanatory, there are some issues that are worth looking at in more detail. In general, web-applications that offer a secure section will allow the user to log in through a login page that is transmitted using SSL/TLS so that their credentials are not sent in plaintext across the network. This is done for most websites although there are still some sites that transmit login credentials in plaintext. In addition to sending credentials over an encrypted connection when the login form is sent to the server, the actual login form should also be sent over an encrypted connection to the client as well (although this is not always done. See [33] for examples of improperly used web-application

logins). However, once the user has logged in, a number of sites will stop transmitting information over the network using SSL/TLS and revert back to using plain HTTP. This can be problematic as the session ID can be sniffed off of the network using a network analyzer such as Ethereal [22]. There is plenty of opportunity to obtain the session ID in this way as the session ID is passed back to the server in every subsequent request. Although this problem is addressed by a large number of systems, there is another problem that is often overlooked which is explained in the following below.

Property 5: Ensure that the host and path assigned to the cookie are not too general.

In the case where a particular web-application has different functions running on a number of different servers, session ID cookies may inadvertently be sent in the clear when being transferred from one server to another within the same domain. For example, suppose a company named Acme has an online store. Acme has two servers, a secure server for processing orders named secure.acme.com and an insecure server to allow customers to browse Acme's merchandise named shop.acme.com. After a user logs into the secure server, they are assigned a session ID that is used to access their account and order information. If the Host property of the cookie is assigned a more general host value, in this case acme.com, when the user goes back to the insecure server, their session ID cookie will be transmitted in plain text since the shop.acme.com is in the acme.com domain. This type of vulnerability is sometimes overlooked as some applications have a large number of servers that may use the session ID to synchronize themselves and enhance the end-user experience. However, by setting the host and path values of the cookie appropriately in addition to setting the secure flag (covered below), the dissemination of the cookie can be controlled further. This will keep the session IDs from being transmitted in plaintext as well as limit the host the browser will send the cookie to. Although this mechanism protects the cookie from passive eavesdropping, it does not protect the session from being stolen by an XSS attack on the secure site, a CSRF attack, or an HTTP splitting attack.

Property 6: A new session ID should be assigned whenever the user logs in.

As mentioned in section 4.1.2.4, some web applications allow the client to pass a

previously used session ID which can then be used to manage the user's session data. This type of mechanism could be acceptable when non-sensitive information is being stored; however, when dealing with sensitive information, a new session ID should be assigned every time a user logs into the system. This will help to prevent session fixation attacks as an attacker will have a harder time setting a victim's session ID before they log into the system. Also, this session ID assignment should be done after the user successfully logs into the system and not when the user accesses the login page.

Property 7: Ensure that secure cookies are marked as Secure.

Marking a cookie as secure means that the cookie should not be transmitted over non-secure channels. In web-applications, this generally means that the cookie should not be transmitted over a non-SSL/TLS encrypted line. The Secure cookie parameter is part of the HTTP state management specification (see section 3.2.2. in [54]) and should therefore be supported by most legitimate browsers.

Property 8: Ensure that secure cookies are marked as HTTPOnly.

There is also a new flag that has been proposed by Microsoft to help mitigate XSS attacks called HTTPOnly [63]. What this flag will do is tell the browser that it should not allow the cookie to be accessed from a client side scripting interface such as Javascript or VBScript. This is a good idea; however, because it is not a standard, not all browsers support this property at this time. We conducted a brief test to determine which browsers support this property and of the six browsers we tested, only Internet Explorer (version 6.0.2900 SP 2) and the Konqueror (version 3.4.1.) Web Browser/file manager that is distributed with the KDE desktop environment for Linux actually enforced the flag. Details of the test can be found at [40]. What we also noticed is that setting the flag does not cause any other problems for browsers that do not support it. This means that the flag can be used as another line of defense, it just should not be solely relied on for protection at this time. Also, they do not protect against attacks where cookies are inadvertently stored persistently as described above, nor does this protect against session fixation attacks or CSRF attacks.

Property 9: Always provide a secure logout function for users.

Providing a secure logout function allows users to logout of their account and should remove the user's session from the server along with the data associated with the session. The logout button or link should be prominently displayed on the page so end-users are aware that it is there and a message could be placed on the page directing the user to make use of the logout function. This is to try and prevent users from simply closing the browser window when they are finished instead of using the logout button. Another solution could be to include a small client side script function which, if the user attempts to close their browser, reminds them to use the logout feature. This will bring the importance of using this function to the user's attention. The same type of function could also be written in order to inform the user that they should log out before they visit other sites as well. This may be considered an annoyance but it will significantly reduce the risk of session hijacking as the secure logout function would destroy the user's session on the server.

Property 10: Perform HTTP Referrer Header Filtering when users leave the site.

The HTTP specification does specify that the browser may send referrer information to the server [25]. The referrer information will contain the URL of the resource accessed prior to requesting the current page. If the URL contains credential information such as a session ID, this information could be leaked to an untrusted server. In order to better protect user credentials that are stored as URL parameters, the user should be redirected through a header filtering page by the server whenever he/she requests a page that is outside the application in question. This will force the browser to change the URL in the referrer field from one containing the user's credentials to a more benign resource name with no credential information. This technique can only be used to protect information stored as URL parameters [66].

Property 11: Use the HTTP POST method when performing transactions.

When processing transactions submitted by users over HTTP, only allow requests containing form data to be submitted using the POST method. This makes the data parameters sent to the server less visible to the end-user or attacker as well as helps to

mitigate certain attacks such as session riding, URL parameter tampering and referrer header information disclosure.

Property 12: Require additional authentication when performing sensitive transactions.

When users are performing an operation that views or makes changes to sensitive information, make the end user perform some sort of additional authorization to complete the operation. There are a number of ways this can be done. One is to require the user to enter his/her password at the same time as he/she is performing any sensitive operations. This solution can become annoying for end users if they are constantly required to enter their password to accomplish their desired actions. This also makes the end-user's password more vulnerable to interception as it is being transmitted over the network much more often.

Another solution which can be used to help prevent completely automated attacks is to introduce a challenge response type of system before any potentially harmful transactions are executed. One way to implement this type of system is to introduce a CAPTCHA that a human user must answer before the transaction can be executed. The acronym CAPTCHA stands for "Completely Automated Public Turing test to tell Computers and Humans Apart" and was first proposed by Moni Naor in 1996 [68]. A CAPTCHA is a challenge response system which presents the client with a challenge that is easy for a human user to solve but difficult for a machine to solve. They are usually in presented in the form of a picture containing scrambled letters which a human can read fairly easily but is not easily processed by a machine. This type of solution would help to prevent automated attacks such as session riding attacks described in section 4.2.5. It is applied in banking applications where e-mail money transfers are used in order to add an extra layer of security.

Property 13: Encrypt any data that must be stored within a cookie using a strong encryption algorithm.

Ideally, the only data that would be stored on the client is the session ID used to access information stored in a session on an application server. However, there are circumstances where storing identifiable data on the client may be required. For example, a large e-commerce system may have an architecture which makes use of a number of servers

all performing the same tasks to help with load balancing. In this case, storing all the user's information in a session on a single server becomes very difficult to manage and very resource intensive. In this case, storing some user data on the client is preferable as it will allow any of the servers to process transactions without having all of the user's data present in a session on the server. If user data must be stored on the client, it should be encrypted using a strong encryption algorithm like AES using a key stored on the server. Although it is possible that the key could be compromised if the server is compromised, this is also the case with a server side session. As an added measure of security, the encryption key used on the server should be changed periodically. This will help to lessen the impact of a compromised key as it will render previous encryption keys useless. If cookies are encrypted using a strong encryption algorithm then even if the cookie is compromised, it infeasible to extract the information from the cookie.

Property 14: Do not store or set session cookies in an HTML document.

As mentioned in section 4.2.3.3., storing temporary session cookies in a hidden form field or setting a cookie value using the HTML META tag has the undesirable side effect of making the temporary cookie persistent in the user's browser cache. In order to avoid making temporary cookie values persistent, session IDs should only by stored as session cookies whenever possible and should be set using the server-side application framework.

4.2.3. Summary of Session Management Best Practices

In this section, we summarize the session management best practices discussed in detail above. For each best practice we give a description, which attacks it used to help mitigate, and which attacks it will not mitigate.

Table 4.1: Session management best practices

Best Practice Description	*Helps to mitigate*
Session IDs should be truly random or as close to truly random as possible. This means that all session IDs should be equiprobable over the session ID space.	- session ID guessing
Session IDs should be sufficiently large.	- session ID brute force attacks
User sessions should have a limited lifetime on the server.	- used to limit the amount of time available to hijack the session
Session IDs should always be transferred over a secure channel.	- credential disclosure via passive network analysis
Ensure that the host and path assigned to the cookie are not too general.	- credential disclosure via passive network analysis - inadvertent credential disclosure to untrusted hosts within the same network
A new session ID should be assigned whenever the user logs in.	- session fixation attacks
Ensure that secure cookies are marked as Secure	- credential disclosure via passive network analysis
Ensure that secure cookies are marked as HTTPOnly.	- XSS attacks targeting cookies
Always provide a secure logout function for users.	- session hijacking after the legitimate user has finished using the system - credential disclosure after the legitimate user has finished using the system
Perform HTTP Referrer Header filtering when users leave the site.	- credential disclosure via URL parameters
Use the HTTP POST method when performing transactions.	- URL parameter tampering - session riding attacks - credential disclosure via URL parameters
Require additional authentication when performing sensitive transactions.	- session riding attacks - automated session hijacking attacks
Encrypt any data that must be stored within a cookie using a strong encryption algorithm.	- information disclosure via cookie dissemenation
Do not store or set session cookies in an HTML document.	- inadvertently making temporary session cookies permanent

These best practices will not completely prevent session hijacking attacks; however, if these practices are followed, it becomes significantly harder to obtain the necessary credential information needed to hijack a session.

4.4. Other Proposed Session Management Solutions

In this section we cover a number of proposed session management solutions which we don't consider to be best practices. A brief description of each of the proposed solutions is provided and an explanation of why these solutions are not feasible in practice.

4.4.1. IP and User-Agent String Restrictions

A number of researchers have suggested linking session IDs to some other information about the client such as their IP address or the user-agent string passed by the browser. There are a number of reasons this solution will not work to prevent session hijacking and even more reasons why this solution will cause problems for legitimate users. The reason this solution cannot be used to prevent session hijacking is that both of these parameters can be spoofed quite easily. And although it is quite difficult for an attacker to spoof the IP of another user and receive a response to the spoofed request, it does allow an attacker to launch a DoS attack on a specific user if they know the victim's IP if a lockout policy which uses the IP information is in place. Since user-agent strings are highly predictable, they do not provide any added security since they can be spoofed as well. Aside from being easy to spoof, there are a number of legitimate reasons that an end-user would change IP and user-agent strings in the course of a legitimate transaction. Taking AOL as an example for no other reason than information on their proxy system is publicly available and they are a well known ISP, we see that AOL users will frequently change IP addresses. AOL makes use of a number of proxy servers through which their users access the Internet. The end-users are also assigned a personal IP address that allows them to connect to Internet resources directly [5]. This means that an end-user who uses AOL as their ISP will likely have a number of different IP addresses during a session (both their own and those of the proxy servers) as well as different user-agent strings as the proxies may change the string. There is also the case where many users are connecting from a single IP address. This is the

case in a number of different network settings such as students working in a school network. All the machines on the network will send their requests via a small number of proxy servers so, from the point of view of the web-application server, all the users on that network will have the same IP address. Users on the network would then be able to steal each others sessions as the IP address provides no added security. Some sites offer this type of added security to the user as an option when they log in. The user can choose to "log in securely". The problem with this is that the user must understand the implications of tying their session ID to their IP and it is unlikely that the average user will have this knowledge.

4.4.2. Dynamic Session IDs

This technique focuses on repeatedly changing a client's session ID, either after every request or after a small number of requests, instead of using a single session ID for the duration of the user's session. This seems reasonable and so a number of variations of this type of scheme have been proposed. One scheme is to simply replace the user's session ID with every request that is made. This may help make a single user's session ID (retrieved via XSS or some other method) harder to use as it will not work if the legitimate user has already been assigned a new one. This does not protect the users against a brute force attack. It actually makes a successful brute force attack far more dangerous as, without proper session duration enforcements, an attacker can keep the session alive by periodically issuing requests and thus being assigned a new session ID for the victim's account. It is also harder to enforce session timeout policies in a scheme where the session ID is constantly changing.

Another solution along these lines proposes keeping a history of the session IDs used for a particular user for the purpose of detecting session hijacking attempts [66]. The author proposes that a system that keeps a session ID history will be able to detect a hijacking attack in the following way: if the legitimate user is assigned a new session ID on every request then a replayed session ID by an attacker signifies an attempted session hijacking. If a hijacking attempt is made, the session is reset and all users trying to use that session must reauthenticate. Barring the attacker knowing the victim's login credentials (in which case a session hijacking attack is pointless), the legitimate user will log in again and continue working. Even in the case where an attacker is able to replay the session ID before the

victim, once the victim replays the session ID, it will be detected as a hijack attempt and the session will be canceled. Now the author does acknowledge that this system is susceptible to DoS attacks and also acknowledges that this system does not prevent a session hijacking in the case where the attacker replays a valid session ID after the victim is finished using the system. In this case, the attacker is able to continue using the victim's account without the system knowing. This type of system may also incur a large amount of overhead in that if there are a lot of users and each user has a large number of session IDs stored by the server, there is a lot of information that is being stored and accessed to perform each operation [66].

4.4.3. Use a Keyed Hash Function to Create Session Tokens

This scheme proposed by Fu *et al.*, attempts to create unforgeable session tokens using a secure hash function. This is done by setting a cookie of the following format:

$$\text{cookie} = \text{exp}=t;\text{data}=s;\text{digest}=\text{MAC}_K(\text{exp}=t + \text{data}=s)$$

where t is the expiration time and s is some data used to identify the user. The MAC is created using a cryptographically secure keyed hash function using a key k stored on the server. This solution is meant to provide a single sign-on type of solution for web-applications that are running on numerous servers as the credentials and timestamp are stored on the client side. The idea is that an attacker will not be able to forge an authentication token for another user because they do not know the key on the server [26]. Although it is theoretically hard to determine the server's secret key, if the key is compromised, then an attacker is able to create valid tokens for any user in the system. This system also implicitly relies on the client sending valid data to the server. This means that credential revocation is not easily performed by the server as it must rely on the expiration date stored on the client. This system also tries to prevent brute force attacks; however, this scheme is not any more secure than using a random session ID to identify the user as long as the key and session ID are of equal length. If anything, this scheme is less secure because if an attacker can successfully find the key, as previously mentioned, they can create authentication tokens for any user in the system. This solution does not protect against session hijacking, session

riding, or HTTP response splitting attacks either.

4.5. Chapter Summary

The session hijacking problem is a difficult one to solve due to the fact that the HTTP protocol is stateless. This requires some sort of identifier to be stored on the client side so legitimate users can make use of the system efficiently. Another issue that makes session hijacking and credential disclosure a difficult problem to solve is the fact that modern web browsers are extremely powerful applications. Most notably, they are equipped with powerful scripting interfaces which allow hosts to access any part of the browser including any credential storage location. This makes XSS attacks extremely dangerous. Although the best practices presented in this chapter should help to make it harder for malicious users to obtain valid user credentials for other users, the issue of limiting the ability of the client application needs to be addressed in order to better protect end-users. The HTTPOnly feature described above is certainly a step in the right direction; however, more care should be taken to help limit the attack surface provided to malicious users from which to launch attacks. In the following chapter we discuss the details of the system tested using our methodology as well as the various vulnerabilities found and general difficulties encountered while performing the evaluation.

5. Test System Details and Problems Encountered

In this chapter we provide an overview of the system tested using our testing methodology. A summary of the various systems tested, how they are interconnected and an overview of the problems found during the evaluation are discussed. We also discuss some of the difficulties encountered during the course of the evaluation.

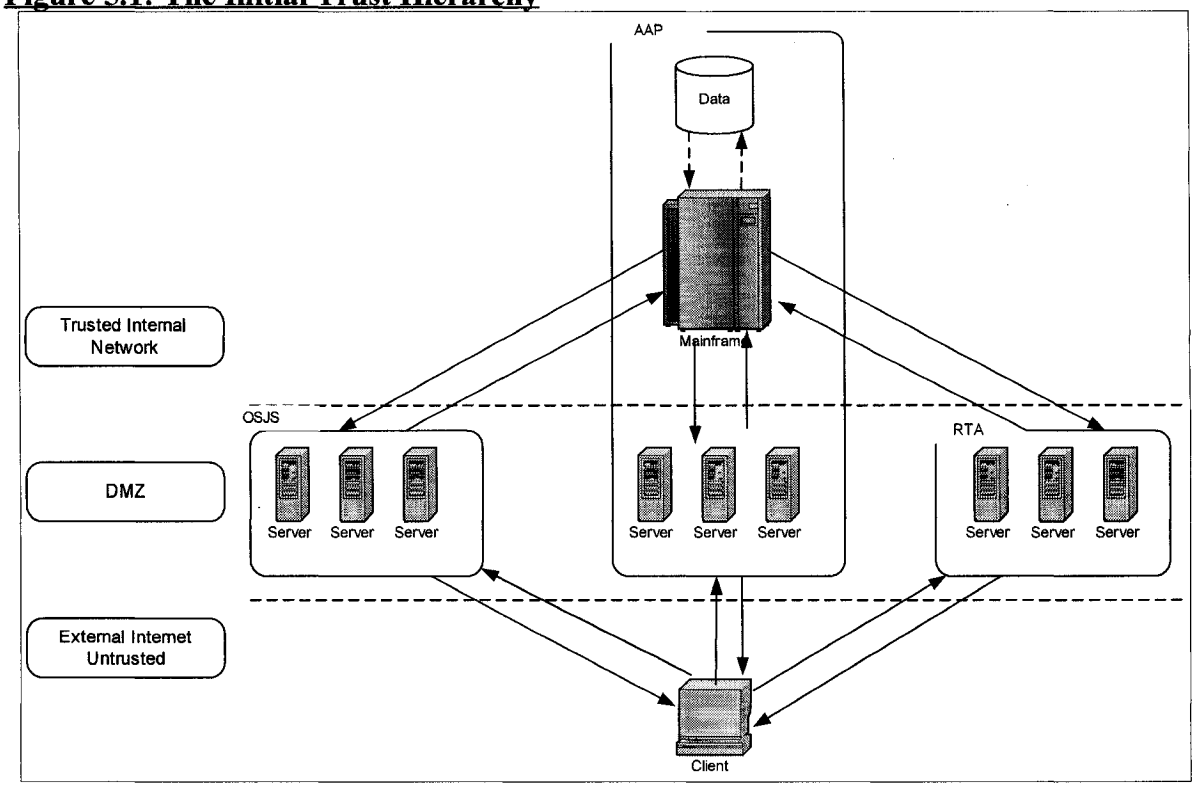
5.1. AAP System Details

When describing the sample AAP system evaluated, it may help to break down the application into layers based on the implicit trust level associated with each layer. This helps us to understand which parts of the system are accessible to which classes of the users in the system. The overall architecture of the AAP system will be built up throughout this section. Although this system is based on a collection of web-based applications deployed in a university setting, the goal here is to introduce a sample architecture which will be representative of a large number of deployed web-based systems in many different settings. For the remainder of this work we use the term credentials to describe authentication credentials used to grant access to resources in the web-based system. Credentials can either be username/password combinations or session IDs assigned by the system.

5.1.1. Trust Boundaries

The issue of trust in web-based applications is a large field of research. Therefore, it is not the goal of this work to delve into this area of research. Instead, the goal here is simply to describe the different areas within our sample application where access is considered more restricted. The following trust hierarchy (see Figure 5.1) shows the different levels of trust within the AAP system.

Figure 5.1: The Initial Trust Hierarchy



At the top of the hierarchy is the internal data store and the mainframe which contains the legacy applications which are used to access the data store. Since the mainframe and data store are located on the internal network, they are only accessible by the trusted hosts within the DMZ. The hosts in the DMZ are grouped together to show that they are managed by different groups within the organization. The group managed by the AAP administrative team includes the mainframe and data store because this is the group responsible for its management. The other groups in the DMZ managed by the OSJS and RTA groups also have trusted access to the AAP group's mainframe. Clients accessing any of the groups' systems are located on the external Internet and are considered untrusted but they are able to gain access to information in the data store that they are authorized to access through each of the given systems. Clients cannot access the mainframe and data store directly from the external network. The systems managed by each group which are accessible to the client are all hybrid web-based systems which implies there are also anonymous parts of each of their web applications. The dotted lines indicate the trust boundaries in the system. It should also

be noted that there are other web applications hosted within the DMZ that do not have any interaction with the AAP mainframe. These applications can be managed by other individuals or groups within the organization. Now that the initial trust boundaries have been determined, the actual system functionality can be described in terms of the groups of components present in Figure 5.1 above. Even though not all of the systems are managed by the same group, all of the systems interacting with the AAP mainframe also interact with the AAP authentication and session management system. This is done in order to offer the client a single sign-on type of solution so that they are able to login to the AAP system and are able to access the systems managed by the OSJS and RTA groups without having to login again. The systems offered by the RTA group have their own login screen to allow the user to solely login to their system; however, this will not grant access to the AAP system. The same is true for the OSJS system. The OSJS and RTA systems have access to the mainframe in order to verify credentials such as username/password combinations or session IDs assigned by the AAP system and to allow clients access based on those credentials. User credentials used to access the system as well as all the business information used by the system are stored in the mainframe data store.

5.1.2. AAP Mainframe and Data Store Components

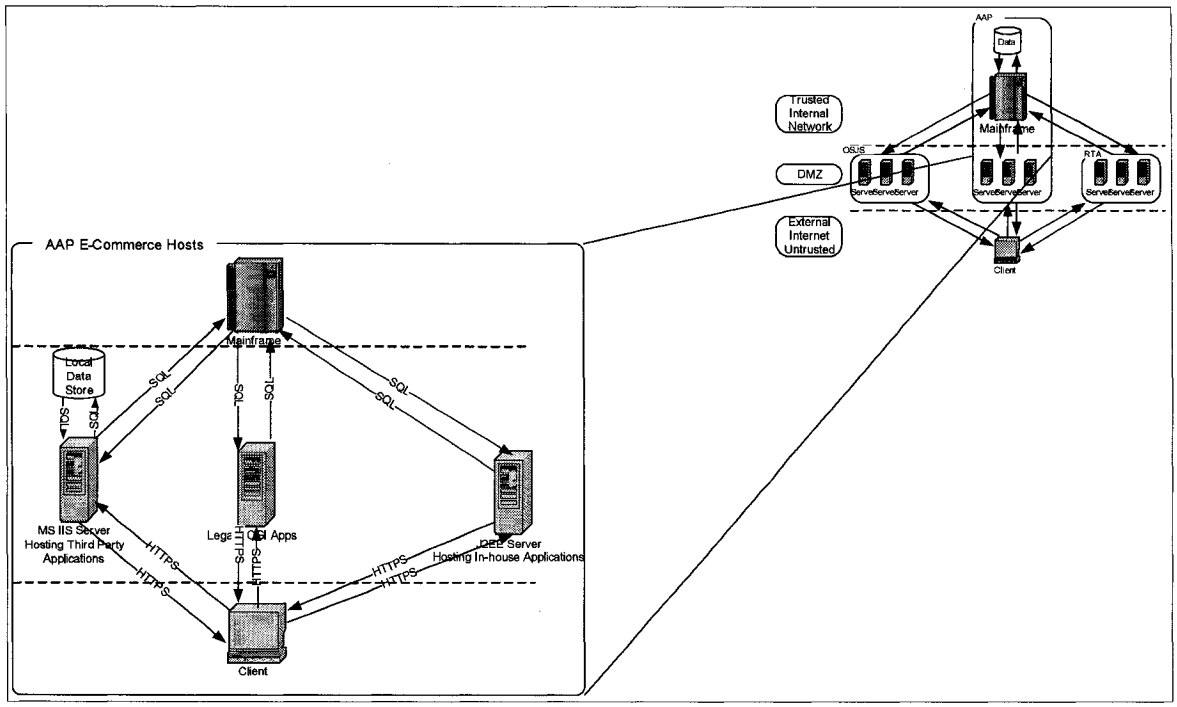
The AAP mainframe is a mainframe system that has been used by the organization for roughly 20 years and contains all of the business logic used to perform day-to-day operations. Before the idea of web-based systems and clients accessing their own information came up, the data used by the organization was managed through custom client applications which were used by staff within the organization. When the decision was made to offer some of these services to clients over the Internet, a number of CGI applications were created to provide a web-based interface to the legacy applications in place. The CGI applications are implemented using a legacy, COBOL-like language. The CGI applications are hosted on a intermediary host in the DMZ and managed by the AAP team. This is done because it is imperative that the mainframe and data store not be directly accessible from the external network. The mainframe contains a number of procedures to allow authenticated users to access their personal information stored on the mainframe. These procedures can be

used by other web-based systems using a number of different technologies through an RPC interface to the mainframe. There is a vast amount of sensitive, personal information stored in the mainframe data store such as names, mailing addresses, social insurance numbers, and credit card numbers, as well as business dependent information. In the case of the AAP application, business dependent information would be student's grades, financial information, and registration information.

5.1.3. AAP Modern Web-based Application Hosts

This component of the system includes all web-based hosts that are managed by the AAP team. This set of hosts is located in the DMZ and is used to offer numerous services to the clients over the external network. In Figure 5.2 below, a more in-depth view of the AAP managed hosts is provided. In this figure, there are three servers which are used to host the applications. One server is used to host the legacy CGI applications, a second to host Java applications created in-house by the AAP team, and a third used to host a third party ASP application purchased by the AAP team. Since these hosts are managed by the AAP team, they have direct control over how these hosts are configured and are able to deploy any security measure they wish in order to protect these hosts. The client communicates with all of these hosts using the standard HTTP protocol over an encrypted SSL connection. The third party applications also have a temporary storage which stores any transaction data created throughout a given period of time. This is done so that no changes are directly made to the mainframe data store by the third party application. Communications between the hosts in the DMZ and the mainframe are done using a number of stored procedures but ultimately, information in the data stores is accessed and modified using SQL. The same is true for the third party host accessing its own temporary storage. The user authentication is done using an in-house login application and user sessions with the system are managed by a system developed in-house which runs on the mainframe.

Figure 5.2: Detailed View of AAP Managed Web-based Hosts

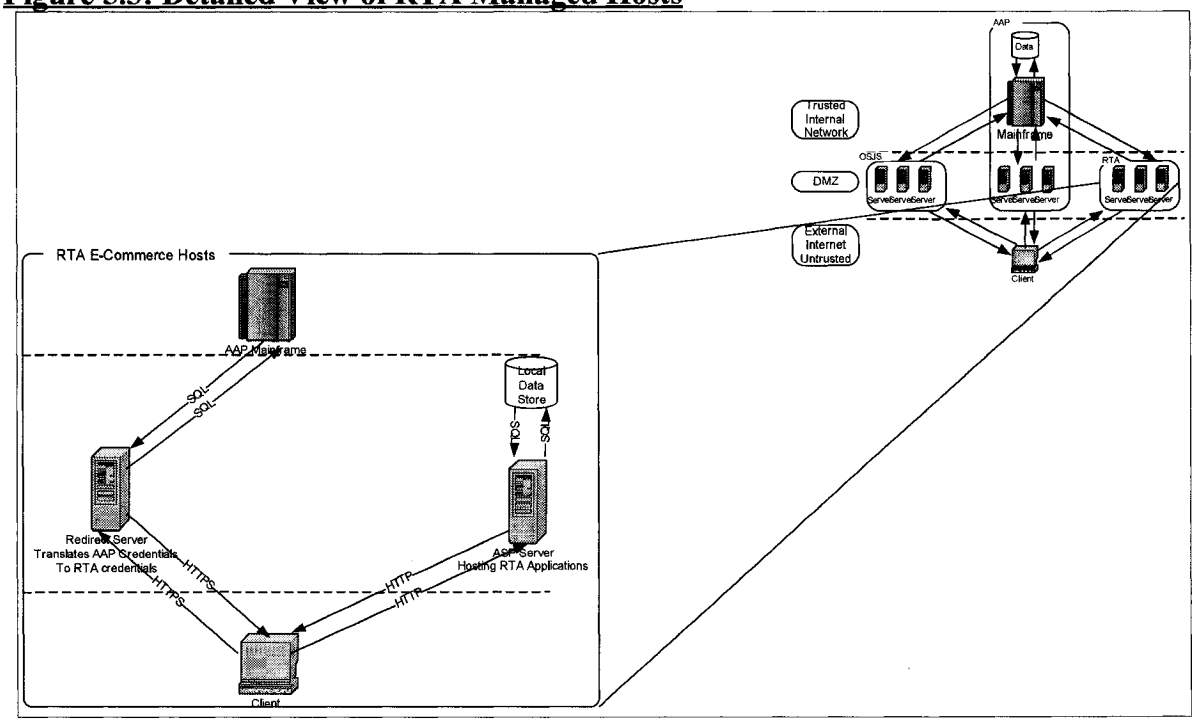


5.1.4. RTA Group Application Hosts

The RTA group is a separate group from the AAP group but they both belong to the same organization and, therefore, the hosts and applications run within the same domain. However, because they are owned and run by different groups, they do not necessarily have the same security policies in place. The RTA application has its own authentication system which allows users to login independent of the AAP applications. Because the authentication and session management system used in the RTA applications differs from those used by the AAP applications, a redirect server is placed between the RTA and AAP applications which is used to translate AAP credentials to RTA credentials when the user accesses the RTA system from the AAP system. Accessing the AAP system after logging into the RTA system is not supported; however, if the user has logged into the AAP system, accesses the RTA system, and wishes to return to the AAP system, this is allowed as the AAP credentials will still be stored on the client side. In order to convert the credentials, the redirect server retrieves the user's credentials and verifies them with the AAP mainframe. If

they are valid, RTA credentials are generated and the user is redirected to the RTA system with the new credentials. The data managed by the RTA system is stored independently of the AAP system and therefore, no interaction is needed between the main RTA server and the AAP mainframe. This includes the login information used to identify users. The RTA applications also do not support SSL encrypted connections so all transactions are performed using plaintext HTTP. The RTA applications are purchased third-party software which has been configured to suit the needs of the group. Figure 5.6 shows the details of the RTA system with respect to the overall system architecture.

Figure 5.3: Detailed View of RTA Managed Hosts

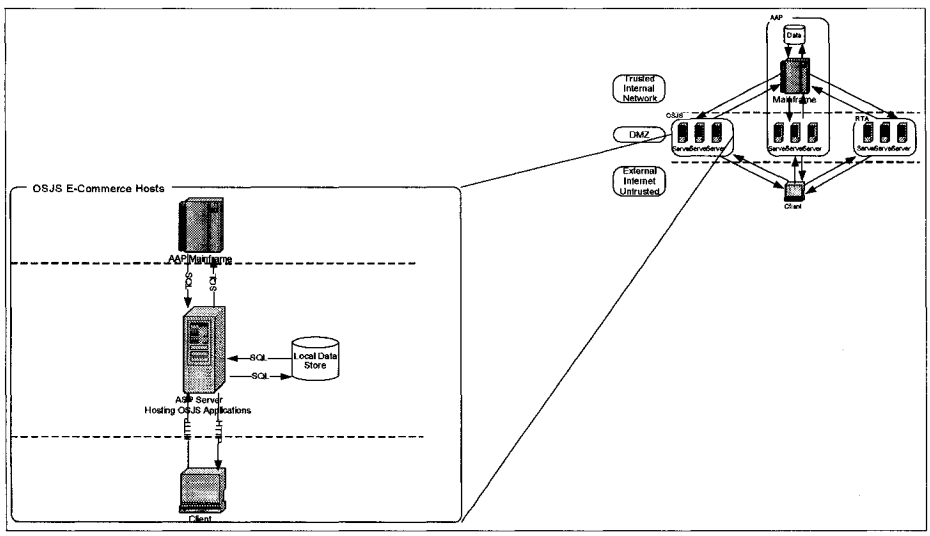


5.1.5. OSJS Group Application Hosts

The OSJS group, much like the RTA group, has a different management and administrative team than the AAP group. This implies that they may necessarily have different security policies regarding how their hosts are managed and how information resources are handled. Although the OSJS and RTA share this similarity, there are some key differences in how each of these group's systems function and interact with the AAP system. The OSJS does not have its own authentication system. It relies on the AAP mainframe to

authenticate users and makes use of the same authentication credentials. The OSJS system may be accessed directly from the AAP system by an authenticated user or the OSJS may be accessed by using its proprietary login screen. It should be noted that not all users of the AAP system will necessarily be users of the OSJS system. This means that the responsibility of managing who has access to the OSJS system has been assigned to the AAP system. The OSJS system only relies on the AAP system for the authentication service. All the information that is only relevant to the OSJS system is therefore managed and stored with that group's hosts. Once again, the management and administrative team is different for the OSJS which means that they may have different standards for information security. That said, the OSJS does not make use of an encrypted connection when performing transactions so all information is transmitted in the clear using HTTP. It should also be noted that simply because the OSJS is accessible from the AAP, the reverse is not true. A user cannot login to the OSJS and access the AAP without explicitly logging into the AAP. All of the applications managed by the OSJS team have been developed in-house. Figure 5.4 shows a detailed view of how the OSJS fits into the overall system.

Figure 5.4: Detailed View of OSJS Managed Hosts



5.1.6. Other Hosts in the Network

Although they are not illustrated in the diagrams above, there are a number of other

groups within the network who are also hosting an assortment of the web applications which have nothing to do with the AAP. These hosts may be managed by groups or individuals and they may or may not be trusted by the AAP. Even though these hosts do not have any direct interaction with the AAP system, when considering a secure web-based solution, these hosts need to be taken into consideration because they are operating inside the DMZ and may necessarily have more access to the internal network than an outside attacker.

5.1.7. Client Component

The client component in this system should be described in some detail. The users of the system make use of any number of client applications which support the HTTP and SSL protocols. This application will most likely be in the form of a web browser. Widely used web browsers are powerful, client-side applications that are highly configurable. A number of technologies can be used within the browser to make the user's experience more enjoyable as well as to help web application developers track users' identities and transactions. Most web clients allow the use of client side scripting technologies such as Javascript. These scripting interfaces have access to all resources that the browser has access to. The script code is downloaded from the host and is run locally on the user's client application. Since the client component is located in the external network, the application as well as any information sent from the application cannot be trusted by the hosts in the DMZ. Although the most likely client application will be one of the many popular web browsers such as Internet Explorer or Mozilla Firefox, there is nothing stopping a user from using other applications such as a web-proxy application like OWASP's WebScarab or the user could create their own client application. This is due to the fact that all of the protocols used to query the web-based systems are known.

5.1.8. Authentication Details

When users initially access the AAP system, they are presented with a login screen where they enter their assigned user name and password. That information is used to either allow or deny access by checking it against a user database stored on the mainframe. If the user name/password combination exists then the user is granted access to the AAP services

as well as the RTA and OSJS services by extension. The user is assigned a temporary session ID which is stored along with his/her user name and some other identifying information which is used later on by the various available applications. This information is stored on the client in the form of cookies and is configured in such a way that it is accessible to all the hosts within the domain. This will allow the user credentials to be available to all the AAP hosts as well as the RTA and OSJS hosts. Since some of the hosts do not use SSL, the cookies are not marked as secure so that they will be compatible with those hosts. The cookies are used in the RTA application to generate the necessary credentials to access their systems as described above. They are also used by the OSJS application to authenticate users who are coming from the AAP system.

5.2. Overview of Vulnerabilities Found

In this section, an overview of the vulnerabilities found in the AAP system is given in order to help better realize mitigation solutions in the next chapter.

5.2.1. Command Injection

As was described in detail in chapter 2, there are a number of different types of command injection that can be performed against a web application. This is due in part to the fact that there are a number of different command interfaces being used to query various systems. In the case of the AAP application, both SQL command injection as well as client side script command injection, both stored and reflected, were found to be possible. Although some basic validation was performed in the available HTML forms, this validation was only done on specific fields. For example, the fields that were expected to be numbers are checked to ensure the input from the client is in fact a number but no effort is made to detect command injection attacks in any of the HTML form inputs. In addition to form inputs, the information stored on the client in the form of cookies can also be used to inject commands into the system because these values are used as-is on the hosts. Some parts of the system also make extensive use of hidden form field elements to pass user credentials between hosts. Again these values are trusted and, therefore, not subject to any kind of scrutiny. These types of flaws could allow a malicious user to access information or

restricted parts of the system with out proper authentication or, even worse, allow a malicious user to obtain legitimate, longterm credentials so they can access the system under the victim's assumed identity whenever they wish. These types of vulnerabilities are compounded when considered with the arguably less secure policies enforced by groups outside the control of the AAP.

5.2.2. Broken Authentication

Once the user has authenticated him/her self to the system, as was previously mentioned, they are authenticated in all further transactions by the temporary credentials, in this case the username and session ID assigned by the AAP the server. However, since there are a number of application frameworks being used, user information is stored in a number of server sessions which are accessible using each given framework's own session ID. For example, the application used to change a user's password is implemented using a modern, web-application framework such as J2EE. When the user accesses the application to change his/her password using the HTML form provided, a second session ID is assigned by the J2EE server. The application consists of three steps: 1) the user verifies his/her identity by providing some piece of personal information (for example, his/her date of birth), 2) the user enters his/her old password and the new password, 3) a message is displayed in a third page which tells the user whether the action was successful or not. All information entered into the form is stored in a session on the J2EE server. The intention in the AAP system is to use the credentials assigned by the AAP after the user logs in as authentication credentials for any further actions taken by the user. However, do to the flawed assumption that the user will always access the pages for the application in order, it is possible to bypass the authentication of the AAP credentials. Additionally, when the user uses the current secure logout function (as they should), the AAP session is destroyed but the session of the application server continues to persist. This has lead to sensitive information being accessible without proper authentication. This is done using a session hijacking attack with the session ID assigned by the application server. If a malicious user can obtain a valid session ID assigned by the application server, he/she can request the page from the second step of the process described above (where the user enters his/her new password). This since

the session on the application server is not destroyed by the logout function, the form is populated with the information previously entered by the legitimate user, disclosing the victim's password. There are also instances where a malicious user can inject any username they wish into the userID cookie used by the AAP in order to retrieve information about that user. In this case, the affected applications do not check whether there is actually an open session for the user or if the valid AAP session ID has been supplied. This violates the organization's security policy as no student user should be able to access the information of any other user in the system.

5.2.3. Credential Disclosure

Since there is a large amount of identifiable user information stored on the client, the threat of command injection attacks becomes far more severe. Unfortunately, there are some more subtle and arguably more severe credential leaks present in the system simply due to how it is set up. Because the credentials are sometimes stored in hidden form fields, this has the effect of making temporary credentials more permanent. Another design issue which causes credential disclosure is the fact that the AAP user credentials are transmitted to all hosts within the domain. They are not marked as secure which means that any host that is accessed within the organization's domain, whether it is using an encrypted SSL line or not, will be sent the AAP credentials if they are stored on the client. This was done to allow single sign-on between groups but the inadvertent side effect is that all hosts, including those hosts which have nothing to do with the AAP system, will be sent these credentials. This could allow malicious hosts within the DMZ to harvest user credentials or harvest users' personal information. In addition, the varying security policies used by the various groups within the organization could also potentially disclose valid credentials to a malicious user without that user accessing the AAP run system. This is explained further in the following section.

5.2.4. Varying Configuration and Security Policies

As was previously mentioned, there are a number of groups outside the AAP group which have been granted trusted access the AAP group's systems. This can result in a breach

in the security of the AAP system through an outside source. For example, as mentioned above, the RTA and OSJS groups do not use an SSL encrypted connection to transmit potentially sensitive information. This, combined with the fact that AAP user credentials are sent in the form of domain cookies to these systems, can allow a malicious user to passively eavesdrop on the communications and extract the relevant user credentials. Since the AAP group has no control over the practices of the other two groups or any other group in the organization, they must take this into consideration when determining which parties should be trusted and which ones should not. The RTA and OSJS do not have the same security policy with respect to host protection. This was an assumption made during the security evaluation based on the number of accessible ports on these groups' hosts as well as which ports and services are offered to the external network. Some of the hosts belonging to groups outside the AAP group may be vulnerable to other attacks or may have already been compromised. This is especially dangerous if the hosts outside the AAP group make use of a local storage to manage user authentication information. For example, the RTA or OSJS system may manage their own database of user accounts which have been obtained from the AAP. This could be done to help improve performance. But if the credentials are stored outside the AAP system on a vulnerable host, a malicious user can simply target the weaker host in order to gain access to the AAP system.

5.3. Problems Encountered During the Evaluation

5.3.1. Resource Accessibility

One of the major problems encountered during the course of the security evaluation was obtaining access the various resources that were supposed to be made available. In terms of system artifacts, we were able to obtain source code for a number of systems run by the organization but there was no additional documentation available. The source code for any third-party software being used was not available. Additionally, access to the source code of all systems not run by the AAP department were also not available. Due to the varying resources available for the systems being evaluated, the testing procedure used needed to be adjusted. The black box testing approach discussed in section 3.3.2 was employed where system documentation and source code were not available. In the case

where source code was available, the white box testing approach described in section 3.3.2 was used instead.

5.3.2. Legal Issues

Understandably, the organization as well as the tester in this evaluation wanted to make sure the conditions of the evaluation were clear before commencing. This step is crucial yet very time consuming. Dealing with the rules of engagement for the evaluation are part of the first phase of our methodology and needs to be clear before moving forward. If the organization does not understand the implications and potential dangers of performing a security evaluation, problems between the organization and the testing team may arise as the evaluation proceeds. It took a substantial amount of time to agree on the exact details of the legal agreement between the testing team and the organization.

5.3.3. System Accessibility

At the onset of the evaluation, the organization agreed to provide valid test accounts which would allow the testing team to obtain a good understanding of how the system works as well as test the system from the perspective of a privileged user. Although test accounts were created at the beginning evaluation, the access provided by those test accounts was not equivalent to the access that would be provided to a legitimate user. In some cases, access to complete systems was not available for the duration of the evaluation which meant that the adversarial model needed to be altered when testing those systems to consider an attacker with no privileged access to those systems. This was the case with respect to systems not managed by the department in charge of the AAP system. In addition to the lack of resources available for non-AAP systems, there was also a lack of legitimate access to these systems. The consequence of this from a testing standpoint is that it reduces the coverage of the evaluation and those systems where access is not given will not be tested as thoroughly as those where access is granted.

5.4. Summary

In this chapter we described the sample web-based system that was the subject of our evaluation. The description included architectural and implementation considerations, as

well as an overview of the types of vulnerabilities found during the evaluation. The following chapter provides a detailed description of the proposed mitigation solutions to help make the AAP system more secure.

6. Proposed Solutions for the AAP System

When considering solutions for the problems described in section 5.2, it should be established that the solutions are for the AAP group to protect their assets. The ultimate goal is to protect the assets of the AAP group from malicious users while maintaining the desired functionality and making minimal changes to other groups outside the AAP since they are the actual client requesting the evaluation.

6.1. Re-evaluating the Trust Boundaries

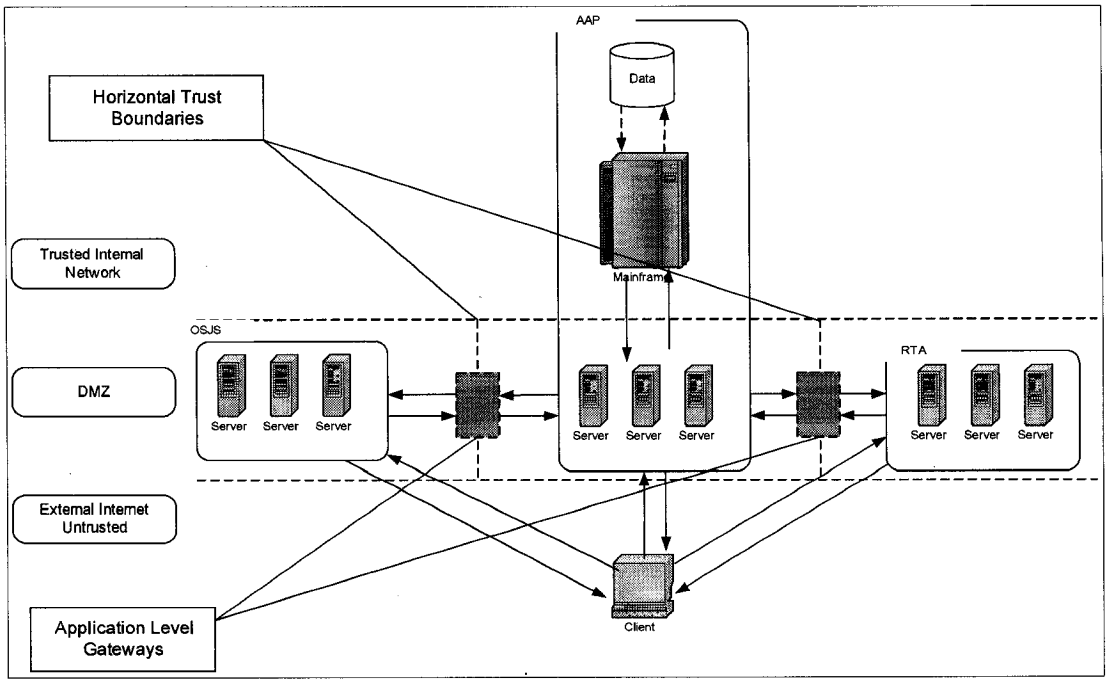
Consider the trust boundaries laid out in Figure 5.3 in chapter 5. The system is organized into what can be described as a vertical trust structure. At the top of the structure are the most trusted components of the system, those located on the internal network. Direct access to the internal network is restricted to hosts located in the middle layer of the trust hierarchy, the DMZ. Hosts located on the external network have limited access to the hosts in the DMZ. The first flawed assumption made by the administrators of the AAP system is that the other groups within the organization can be trusted to access their resources. This is a flawed assumption because not all groups within the organization are deploying their systems with the same standard of care as the AAP group. This has the potential to provide malicious users with a weaker target than those hosts managed by the AAP which can be used to later attack the AAP. These attacks include, but are not limited to, compromising the temporary storage of a host outside the AAP to gain credentials, sniffing credentials on an unencrypted connection, or setting up a host inside the DMZ in order to obtain credentials.

One possible solution for dealing with potentially untrusted hosts within the DMZ is to institute an information security policy which dictates how hosts within the DMZ are to be set up. This would essentially have the effect of hardening the DMZ in order to make those hosts more trusted. The policy would need to be adhered to by all groups within the organization. Although an organization-wide policy would dictate how information assets should be managed, a significant investment may be needed to ensure that any policy in place is actually being adhered to. It would also be difficult to introduce a policy that all

groups within an organization will agree on. This is due to the fact that the activities of the various groups within the organization will likely vary. In order to cope with these many groups of requirements, a global policy will either need to be very general in order to meet all of the requirements or it will need to be very strict with an allowance for exceptions. In either of these cases, it is quite likely that this will result in the same problems as those present in the original architecture.

A better solution for the AAP team in this case would be to reconsider the trust boundaries in and around their systems. During the evaluation, it was found that a number of hosts that are trusted by the AAP system within the DMZ could actually be malicious and compromise the AAP system's security. Therefore, as part of the proposed solution in this work, the vertical trust boundaries need to be reconsidered as well as horizontal trust boundaries introduced within the DMZ. By creating horizontal trust boundaries within the DMZ, AAP credentials can be managed better. This is due to the fact that the AAP system can limit the scope within which those credentials are used and disseminated. Although this may seem to limit the ability of the system administrators to integrate the various services offered outside the AAP group of applications, we will provide a satisfactory solution that should allow the continued use of the single sign-on functionality while maintaining the integrity of the AAP user credentials. In order to better understand this part of the solution, a high level view of the solutions should be considered. Working from the original architecture diagram presented in Figure 5.5, the horizontal trust boundaries are introduced in Figure 6.1 below.

Figure 6.1: Addition of Horizontal Trust Boundaries in the AAP System



The horizontal trust boundaries introduced will completely restrict the ability of groups outside the AAP system to access the mainframe directly. Instead, services offered by the AAP system to other groups in the network will be filtered through an application level gateway. Gateways will be explained further in subsequent sections but one of the goals of the gateway is to perform a credential conversion from the AAP credentials to the various other authentication credentials used by the various groups in the organization. By compartmentalizing the system and designing each component as though all clients using the services offered are malicious, it will make each component in the system more robust. Details on the various ways to implement the horizontal trust boundaries will be covered in detail in the following section.

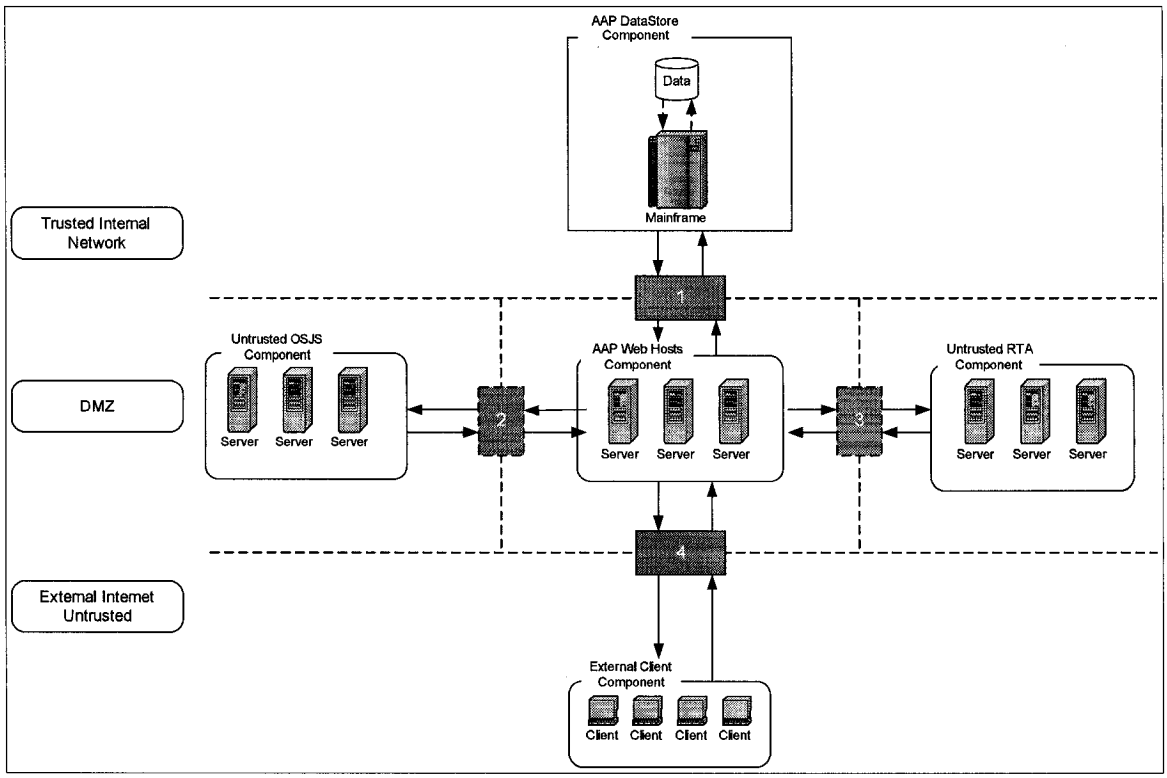
The same compartmentalized design that is used in the DMZ should also be applied to the vertical trust boundaries in the system. Command injection attacks, particularly SQL injection attacks, make use of the poorly enforced trust boundary between the AAP hosts in the DMZ and the mainframe on the internal network. The reason SQL injection attacks succeed is because untrusted data is used to construct a query or set of queries that is passed

from a trusted host to the internal DBMS. This is because many web applications construct database queries in the business logic portion of the application and then simply open a connection to the database to obtain the results of that query. This means that any query sent from the web hosts in the DMZ is trusted. The previously described scenario is basically equivalent to a malicious user gaining access to the a host through a Telnet session since the connected user is able to execute any command available on the target system. Also, most DBMSs require the web application to connect to the database server with a valid user name and password, in much the same way as a Telnet connection; however, the difference between these situations is that if a malicious user were to connect to a target host using Telnet, he/she would need to guess or obtain valid user credentials. In the web environment, the web host contains the necessary credentials to access the internal server. Therefore the malicious user simply needs to know how to manipulate the web application in order to inject their desired commands under the assumed identity of the application server. In the case of the AAP system, since there is minimal input validation in place, determining how to manipulate the queries in the web tier is not too difficult. In order to reinforce the trust boundary between the web applications and the database, a stricter access system needs to be designed in order to better protect the internal data store.

A similar issue exists in the trust boundary located between the DMZ and the external network. The clients interacting with the web hosts in the DMZ trust those hosts and therefore accept the information transmitted to them and, usually, execute any client-side script code that is transmitted as well. An attacker can exploit this trust by finding weaknesses in the system that allow him/her to insert client side code which will execute on the user's browser under the guise that it came from the trusted host. Again, in the case of the AAP system, user input is not scrutinized in any way so a malicious user will essentially be able to inject client side code directly into other clients' browsers either through reflective XSS or stored XSS. Since the code is perceived to have come from the trusted site, the scripting interface will have access to any credentials assigned by the AAP system. This is particularly dangerous in this case due to the large amount of identifiable information stored on the client side. Again, by compartmentalizing the web hosts and the client, the system can be designed in such a way that it is not as susceptible to code injection attacks.

Unfortunately the AAP system will not have complete control over all of the components in the system. Therefore, all of the components which interact with untrusted entities need to have carefully designed interfaces so as to minimize the risk of allowing malicious input to pass through. A compartmentalized view of the AAP system is given in Figure 6.2 in order to illustrate the possible interfaces or gateways which should be introduced between each component.

Figure 6.2: Compartmentalized View of AAP Managed Components



The component interfaces have been numbered in Figure 6.2 for reference purposes. Each of the interfaces has different properties which need to be considered when attempting to determine an appropriate solution for each. Some general assumptions which apply to all interfaces are as follows:

1. All inputs to the component interface are untrusted.
2. All responses from the components are untrusted.

This implies that all requests that are made to a component need to be analyzed and malicious inputs handled in such a way that they do not compromise the information resources to which they have access. Responses cannot be trusted either. If responses are trusted and used as-is, this can lead to client-side code insertion attacks. Now that the trust boundaries have been reevaluated at the architectural level; the next sections will focus on the design issues and recommendations for each of the of the components that the AAP team has control over as well as each of the application level gateways and credential management.

6.2. Credential Management

6.2.1. Cookie Dissemination

The way the AAP and associated systems handle credentials such as username/password combinations and session IDs currently is described in detail in section 5.2.3. along with the potential problems associated with this system. To briefly review, one of the main flaws found to exist in the system is that credentials used to access the AAP are disseminated in such a way that they become available to potentially untrusted hosts in the system. The goal of the solution is to limit the dissemination of these credentials while still providing all of the intended functionality. In this case, the solution needs to allow the user to access all associated systems managed by various groups outside of the AAP group while allowing the AAP group to maintain control of the credentials assigned by their systems.

The credentials used by the AAP system include the user's login name and password and the HTTP cookies containing the user's temporary session key as well as their user ID. The user name and password combination in this case are submitted using a web form over an encrypted SSL connection and are considered fairly secure since this is the only place where these credentials are used. The problem with this system is how and what information is stored on the client after the user has logged in. The system currently assigns the cookies a general domain (such as .uofm.edu) which causes the browser to send these cookies to all hosts within the domain. A more secure solution needs to restrict which hosts the user's cookies are sent to so that only AAP hosts have access to AAP credentials. The implication

of this, since these same credentials are used by other systems such as the OSJS systems, is that some alterations may need to be made to external systems which rely on this information. This particular design issue will be covered in the next section.

There are a number of ways to restrict the number of hosts which are sent the user's AAP credentials. The first method of restricting cookie dissemination is to mark cookies containing credential information as secure. By marking a cookie as secure, it tells the browser that the cookie should only be transmitted over an SSL encrypted connection. This solution will not affect the AAP system in any way as it already uses SSL encrypted connections for all transactions that contain sensitive information. One drawback to this solution is that any host within the DMZ that uses an encrypted connection will be sent the credentials. Since there are a number of untrusted hosts within the DMZ and there is nothing stopping them from setting up an SSL server, this solution will not protect the user's credentials from malicious hosts within the DMZ.

A second approach that can be used to restrict cookie dissemination is to change the path assigned to the cookie. The default path assigned to a cookie is the root path of the application. This tells the browser that the cookies should be sent to all applications located in the root directory of the server as well as any subdirectories. If the path of the cookie is changed so that it is only sent to those applications in some subdirectory of the root, this could help to limit when the cookies are sent by the browser as well. For example, if the cookie is assigned the path value /AAP/, the designers could then move all of the AAP applications into a directory on the web server called /AAP/. Then, when those applications are accessed, the browser will send the credentials because they are located in the appropriate path. The problem with this solution is that the domain is still set in a general way. This means that any host within the domain, in this case ufom.edu, simply needs to run their applications from a directory called /AAP/ and that host will be sent the credentials. The reason this occurs is that the browser decides to send a cookies based on a number of attributes. The main attributes used are the host, the path, and whether it is marked as secure. In the aforementioned case, the cookies will be sent to all hosts *.uofm.edu/AAP/. Again, there is nothing keeping a malicious party from setting up an application within the DMZ with a URL like attacker.uofm.edu/AAP/stealcreds.php. An application in this location will

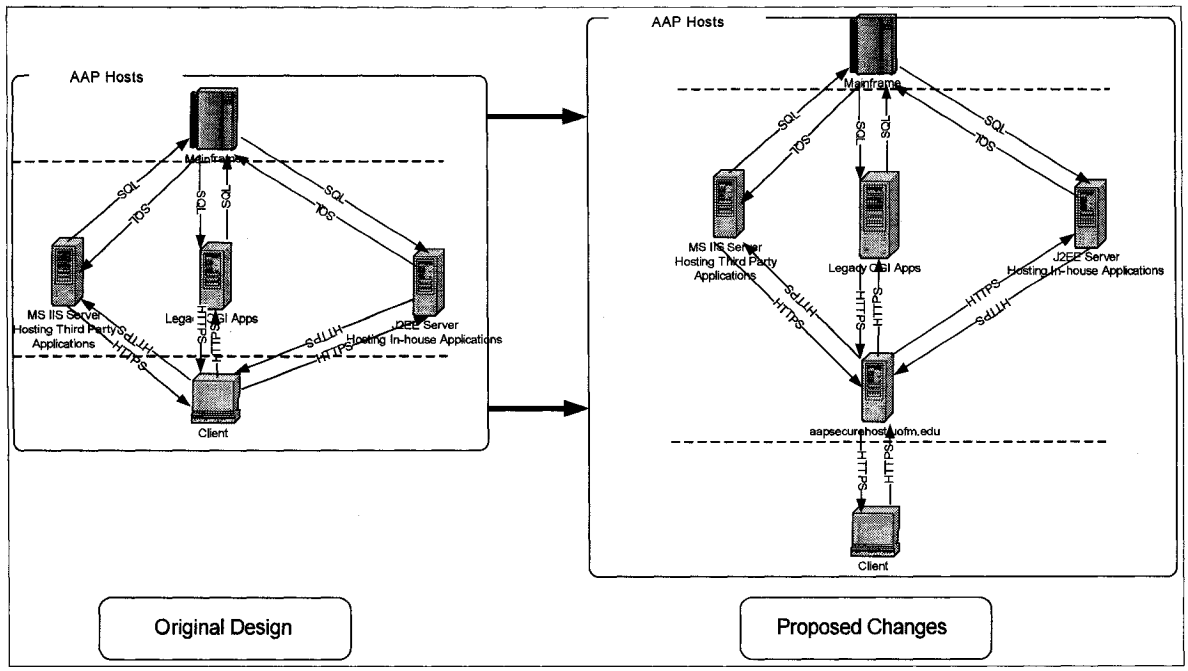
have the same access to the user's credentials as the legitimate AAP hosts.

The main reason the previous two solutions are not viable is that they rely on conditions that are not under the complete control of the AAP group. Malicious users can create new directories within their own web root directory and they can also set up their hosts to use SSL encrypted communications. The AAP team must make use of the one attribute that they have complete control over, and that is the host attribute. One of the fundamental properties of cookies is that a site can only set cookies for itself. This means that site A cannot set a cookie with a domain value for site B and expect that cookie to be sent to site B. Therefore a third solution could be to assign the AAP cookie to a single host controlled by the AAP group. In order to use this type of solution, the multiple AAP hosts would need to perform their transactions through a single host within the domain. This solution would help prevent malicious hosts within the DMZ from stealing credentials with rogue sites since the AAP has control of the DNS server used to refer to all hosts within the domain. Although it is not impossible for an attacker to compromise the DNS server, it makes it harder for the attacker to force the browser to divulge the cookie information in this way. The new AAP credential cookie could be set in the following way:

set-cookie:
sessionkey=kfj3e8djfe8de9djke83;domain=AAPsecurehost.uofm.edu;path=/;secure;

In the above example, the cookie will only be sent to the AAPsecurehost.uofm.edu host and will only be sent over SSL as an added protective measure. In order for this solution to be feasible, the AAP web hosts component will need to be restructured as illustrated in Figure 6.3 below.

Figure 6.3: Possible Restructuring of AAP Web Hosts Component

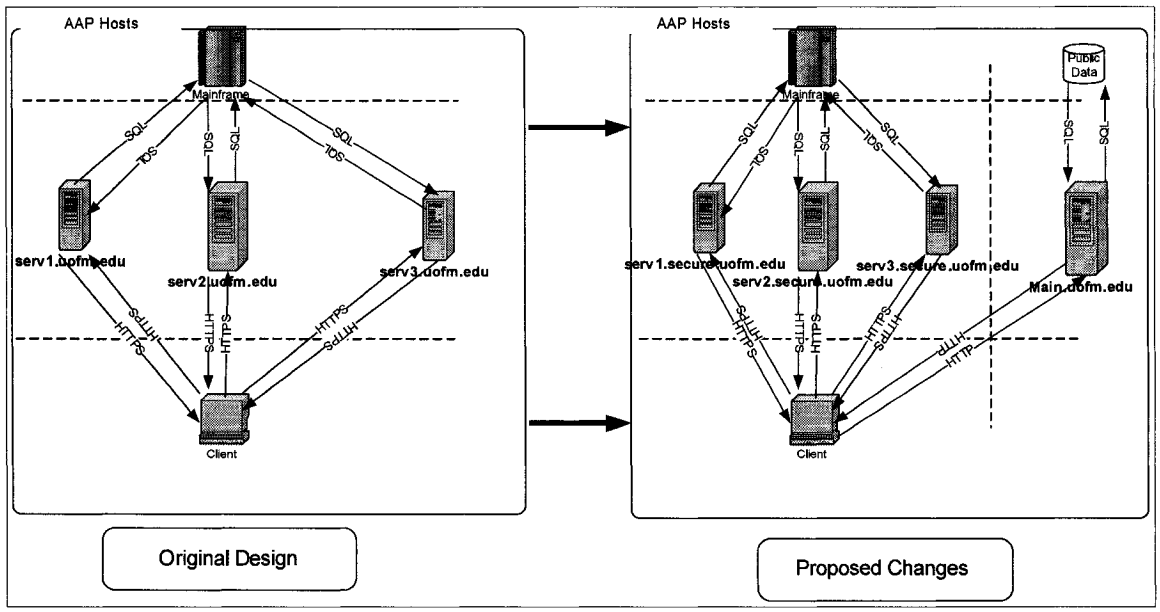


This solution is successful in limiting scope in which the user's AAP credentials are sent to the host; however, it creates a bottleneck at the AAPsecurehost host as well as a single point of failure. If this system were used to help control user credentials, there may be a significant performance penalty since a single host is being used to process all user requests. The processing power of the multiple hosts in the DMZ is no longer being used to process client requests directly. Another disadvantage of this design is that if the AAPsecurehost host goes down, the entire system becomes unavailable. In the original design, if one of the hosts goes down, only the services offered on that host are rendered unavailable (unless the host running the login application goes down). The final disadvantage of this proposed design change is the fact that a significant amount of development may be necessary to implement the single point of access. A more ideal solution would allow the multiple servers managed by the AAP group to continue to process user requests while minimizing the development effort to implement the changes while still restricting the availability of AAP credentials to AAP hosts.

The final solution presented in this section attempts to provide the most ideal solution in terms of security as well as ease of deployment. Instead of limiting access to user

credentials to a single host managed by the AAP group, the credentials can be made available to all hosts managed by the AAP group by creating a subnetwork within the overall domain. Since the AAP group manages the DNS server, a virtual subnet can be created that only contains hosts managed by the AAP group. Figure 6.4 below shows a possible renaming of the AAP hosts within the DNS server.

Figure 6.4: Better Restructuring of AAP Web Hosts Component



Once the DNS server is updated, further changes to the system are minimal. All of the links within the AAP system need to be updated in order to reflect the naming changes. This needs to be done so that the user's credentials are sent to the appropriate hosts with each request. The second change to the system is to the code which sets the cookies. Since the there is now a sub domain that is under complete control of the AAP group, the cookies should be set so that only those hosts located within the sub domain have access to them. The new cookie format will be as follows:

set-cookie:
sessionkey=kfj3e8djfe8de9djke83;domain=secure.uofm.edu;path=/;secure;

Although there are still untrusted hosts within the DMZ, the AAP group can now maintain a part of the network that untrusted hosts will not be able to become a part of (barring a compromise in the DNS server as was mentioned previously). In addition, this solution requires minimal changes to the existing system architecture and minimal changes to the actual implementation. Changes may also be necessary elsewhere in the organization and so the rest of the organization should be notified of the changes so the various groups can make the appropriate changes as well.

One final change illustrated in Figure 6.4 that should be considered is the AAP team should host their onymous applications on separate hosts from their anonymous applications. The AAP system currently hosts their anonymous and onymous sites on the same server. If the content for the anonymous site is stored in the same database as the sensitive data used by the onymous site, it increases the surface of attack on the onymous site. This is due to the fact that a flaw in the anonymous site may be exploited by a malicious user in order to access the back-end data store using a SQL injection attack or, since the anonymous site is hosted within the secure subnet, it may be possible to exploit an XSS flaw in the anonymous portion of the site in order to gain access to user credentials. By separating the anonymous and onymous web applications as well as public and confidential information in storage, the surface of attack is reduced even further. The compartmentalization of the data store is covered in a later section.

6.2.2. Information Stored on the Client

In order to further protect information resources, the amount of information stored on the client side should be minimal. In the case of the AAP system, a temporary session ID as well as the user ID are stored on the client. The problem is that by storing the user ID on the client side, the attacker may be able to target specific users within the system. This may be possible by simply replacing the attacker's own user ID with that of another user in the system in the HTTP request. If the user ID value is trusted by the application, it may be used to gain unauthorized access to sensitive information. A better solution, if the user ID is needed by the application, would be to store the ID as well as any other information needed by the application in a session on the server. This allows the AAP applications to maintain

complete control over all sensitive information used while limiting the surface of attack. Therefore the session ID should be the only credential stored on the client and the user ID cookie should be removed.

6.2.3. Internal Data Storage Considerations

In "the current system", all the information used by the AAP system is stored in a single, relational database. This includes all information used in the public, anonymous portion of the site as well as confidential user information and user credential information such as user names and password hashes. By storing all of this information in the same data store, the surface of attack on the more sensitive information is opened up significantly. This is due to the fact that any SQL injection flaw in either the anonymous or onymous portion of the system can lead to the compromise of personal data or user credentials. In order to better protect sensitive information as well as credential information, the data stores used need to be separated. To separate the data in the most efficient and secure manner, each class of data needs to be considered in terms of what types of accesses are needed, the frequency of those accesses, and the relative importance that the information not be compromised. In the AAP system, the information can be separated into three categories as follows:

> User credentials: User credentials in this case are made up of user names and passwords. They are accessed when the a user attempts to login to their account in order to verify that the user should be granted access to confidential information. This information is not updated often and is only read when a login request is made. This information should be kept as secure as possible since it provides indefinite access to personal information. The user credential storage also contains access control information associated with the user. This access control information is used to grant access to certain parts of the system based on the user's role.

> Personal Information: Personal information is any information that is accessible to an authenticated user. In the case of the AAP system, it would be information such as grades, mailing address, and financial information. Temporary session IDs are also stored in the section of the system for the duration of the user's session. This information is accessed through a number of applications within the AAP and may

be subject to frequent updates. Access is only granted if a valid session ID is associated with the specific user which implies that the user has already authenticated him/her self using the afore mentioned user credentials.

Public Information: Public information includes any information that is accessible via the anonymous portion of the web site. This information may contain relevant information regarding events at the organization or descriptions of services offered as well as any other information the organization wishes to share with the general public. This information is not sensitive at all and is updated and read frequently.

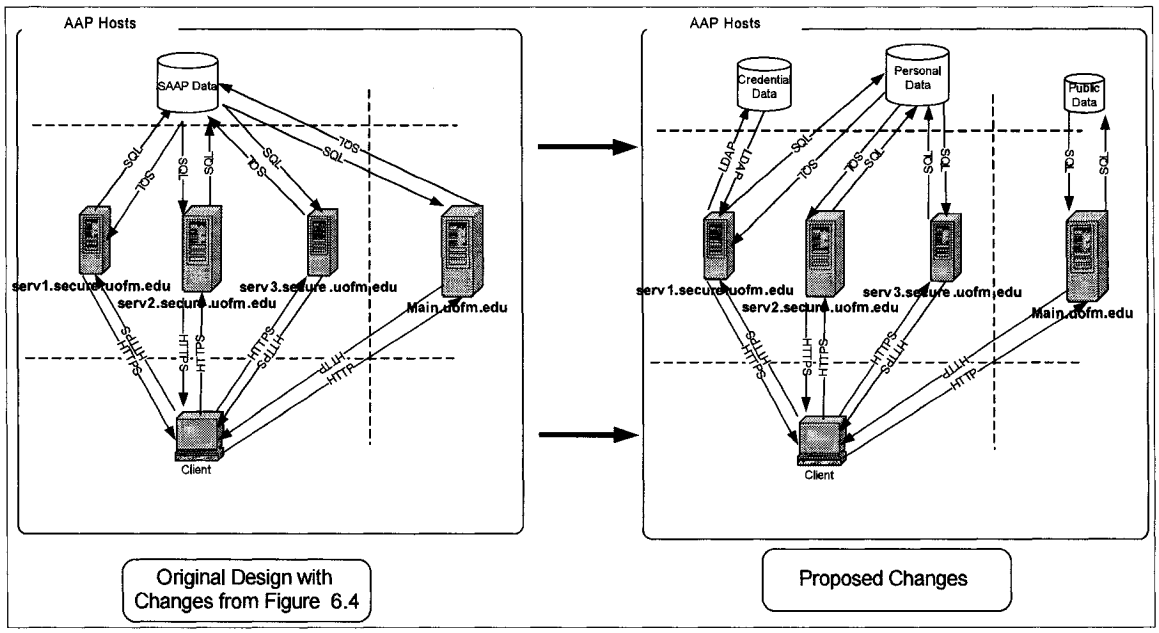
By separating the system data into these groups, it reduces the surface of attack for each group. The designers also need to consider how this information is to be stored. All the information is currently stored in a relational database and queried using SQL. A web account was created on the server in order to provide access to the web server to make queries on behalf of the user. Since the system data is going to be separated, alternative storage mediums may be considered for each group of data. Relational databases are an excellent means of representing large data models with many inter-related object types and complex relations. In addition to this flexibility, the SQL query language provides a simple means of obtaining, adding, and updating information in the database. This is an ideal means of storing and managing personal user information as well as public information. Although these two data groups can be managed by a similar system, they should still be separated either physically by hosting the databases on two different database servers or logically by creating two different accounts within the DBMS, one to access public information and one to access personal information. The physical separation technique is more secure since personal data is not physically on the same host as the public data. Therefore, if an attacker is able to exploit a flaw in the public web applications and perform a SQL injection attack, they are only able to gain access to the information present on that data host. This solution also requires more resources as a dedicated host is needed for each of the data stores and these data stores will need to be managed independently. The logical separation technique may provide a sufficient solution since both the public and personal level accounts can be restricted by the DBMS. As long as the public account is only given

access to the relevant tables in the DBMS and not the entire database, the DBMS will look after the access control of the information. This way, even if there is an exploitable SQL injection flaw in the public system, the DBMS will not allow queries to access personal information. This is a more efficient solution in terms of resources as well since it only requires the accounts used by the public and personal web servers to be reconfigured. All of the data can still be managed in a central location.

When dealing with user credential information, the relational database model may be used but it may not be the best solution. Since an RDBMS schema is not bound by any standard, there is no common way to represent users, roles, and permissions in the database. This can make it difficult to integrate a large number of distributed applications which rely on user credentials. If a common location is used to store all user credentials and permissions, a database schema which represents this information will need to be agreed upon. An alternative to this is to have each application manage its own user database. The problem with this that it makes user management much harder as there may be a number of different ways to represent credential information and any changes to this information will need to be propagated through all of the credential stores. This is the problem that LDAP directory servers were created to solve. The LDAP protocol and directory servers based on this technology provide a standards based approach to managing user information that can be used by any client which implements the LDAP protocol [105]. In addition to a standard protocol used to access the user base, the structure of the directory as well as identifying information for specific entries are also specified by a number of standards [103, 104, 36, 37, 102] which ensures that the user directory has a standard base structure. Managing user credentials in this way has the advantage of providing a single credential store which allows new applications to be added with minimal effort. Modern application development frameworks such as J2EE and ASP.NET offer rich application programming interfaces which can be used to develop applications which make use of an LDAP service. Also, using an LDAP to store user credentials can also allow credentials for other, non-web-based systems to be changed from the web interface. For example, a user can change his/her user credentials for a particular workstation if the underlying operating system of that workstation uses an LDAP directory to perform authentication. Figure 6.5 below reflects the proposed

architectural changes to the back-end data store component.

Figure 6.5: Proposed Changes to Internal Data Store Component



Now that the system has been recompartmentalized to help manage user information and credentials better, the communication gateways between components needs to be considered in greater detail. This is covered in the following section.

6.3. Application Gateway Details

6.3.1. Client/DMZ Gateway

The application gateway between the client and the DMZ (denoted as box "4" in Figure 6.2) is probably the most important gateway in the system since it is the point of contact with potentially malicious users. It is also likely the hardest gateway to design because of all of the types of attacks which target the web tier. This gateway helps to mitigate the command injection attacks described in section 5.2.1. The main type of attack that should be addressed at this level is the threat of client-side code insertion or XSS attacks. XSS attacks are particularly dangerous in an onymous system because they can allow a malicious user to hijack other user's accounts as was previously mentioned. Preventing XSS

attacks is a difficult problem to solve and a number of different approaches have been proposed to help mitigate this threat. These are described in detail in chapter 2. One of the reasons XSS attacks are so difficult to prevent is that there are many ways of encoding data and client-side code used by a web browser which make it extremely difficult to filter. Developers may attempt to filter client-side code out of user supplied values by searching for known keywords or symbols, a black-listing approach. However, since there are so many ways to encode Javascript code due in part to the fact that whitespace in keywords is ignored as well as the many different character encoding schemes supported by most modern web browsers, maintaining accurate lists of which content to filter out can be time consuming and error prone. A somewhat easier and more efficient way to help make XSS attacks more difficult is to create strict rules regarding what is considered to be valid data and reject any data that is not valid, a white list approach. This task is made easier by making use of strongly typed variables to store any values coming from the client. Many XSS and other code injection vulnerabilities are made possible by the fact that most client side inputs are stored and used as strings which can contain any information. If a particular input is supposed to be an integer, make sure the value is checked to ensure it is an integer. The same types of rules can be created when considering string inputs coming from the client. A simple example is a check to ensure a valid postal code was entered into the appropriate field. A simple way to check this is to ensure that each part of the code is three characters long and that each part includes only alphanumeric characters. Other similar validation rules can be created for any other type of data input needed by the application.

Although the white-listing approach may make it easier to detect malicious input, there are some rules which are not as clear. For example, if a rule is created which specifies that a last name will only contain alphabetic characters, this will not allow users that legitimately have punctuation characters in their names (such as the last name O'Connor) to enter their names. Therefore, a better solution for performing user validation should perhaps be made up of both a white-listing and black-listing approach. When coding validation rules, a description of the data that should be accepted needs to be produced (the white-list). The description should include information such as length requirements (exact, max, and min length if applicable), data type, and characters allowed. By specifying which characters are

allowed in a data field, it will be easier to detect and discard potential script injection attacks since most of these types of attacks will contain a number of special characters. In addition to specifying an acceptable character set, other rules may be coded which restrict how certain special characters may be used in a particular field. For example, when developing a validation function for a last name field described above, special characters used in certain script injection attacks such as ' and – need to be allowed in some cases because they may legitimately appear in a last name. In this case, a rule could be coded which enforces a policy where only a single dash preceded and followed by a letter may be allowed and any other instance is rejected (black-list). By combining the two approaches, it should allow developers to provide strict, white-list style rules for input validation but also allow them to specify black-list style exceptions to the rules.

Another consideration that needs to be considered when performing data validation is where the validation will be done. Since web applications can potentially have a large number of forms and therefore a large surface of attack, some researchers have proposed the idea of creating a separate validation module that all data flows are directed through. A solution similar to this was proposed by Scott and Sharp [84]. They propose an architecture where a security gateway is set up between the web hosts and the outside network which takes a defined security policy and checks all untrusted user input to ensure it conforms to that policy. This allows security policy to be abstracted away from the actual web applications. Although this type of solution has the potential to lessen the impact of flawed application code, it also introduces a bottle neck in the system since all traffic needs to be processed by the security gateway. This type of solution also does not help to enforce better programming practices among development teams so they may continue to produce faulty software. A variation of this type of solution would be to create a group of data validation classes in the organization's desired platform. The validation package would provide an API that can be used to validate data sent by the client to ensure it meets the desired format requirements. Existing forms and applications would need to be updated so that they make use of the validation module; however, in the case of the AAP system, these updates would need to be done regardless since the applications do not perform any validation at all. An advantage of using a centralized package to perform validation is that if any updates are

made to the package, those updates will be made to all the applications which make use of the package as well. It also provides a means of centralizing a data validation policy without introducing a bottleneck into the system.

In addition to strict validation rules for untrusted data, in the case where special characters are necessary and the information being entered may eventually be displayed by the client, all special characters should be replaced by their HTML encoded equivalents before being displayed to the user. Again this can be coded and added to the validation module so that the functionality is available for all of the application developers in the group. This implies that, in addition to having validation routines to check data received from the client, global routines will also be available to render any potentially unsafe data so that it is not interpreted by the browser as code but simply as text containing special characters. This will help to mitigate the threat of XSS attacks even further because even if a malicious user is able to inject client-side script into a data field that will later be displayed to the user, the output checking routine will render the text in such a way that it will either be displayed literally or, in some cases, the special characters will be removed. Strict validation rules before data input and output will not completely prevent script injection attacks. But it will raise the bar considerably for a malicious user to launch a successful attack.

6.3.2. Inter-DMZ Gateway

These gateway help to mitigate the problems described in section 5.2.3. The inter-DMZ gateway also helps to mitigate the threat of other systems run by divisions which have security policies that are less strict as described in section 5.2.4. In the current design, the AAP cookies are sent to all hosts within the organization's domain. If the changes proposed above are made to the system, those cookies will no longer be available to hosts outside the AAP secure subnet. The main design goals of the application gateway between the secure AAP subnet and other group's hosts located in the DMZ (denoted as "2" and "3" in Figure 6.2) are the following:

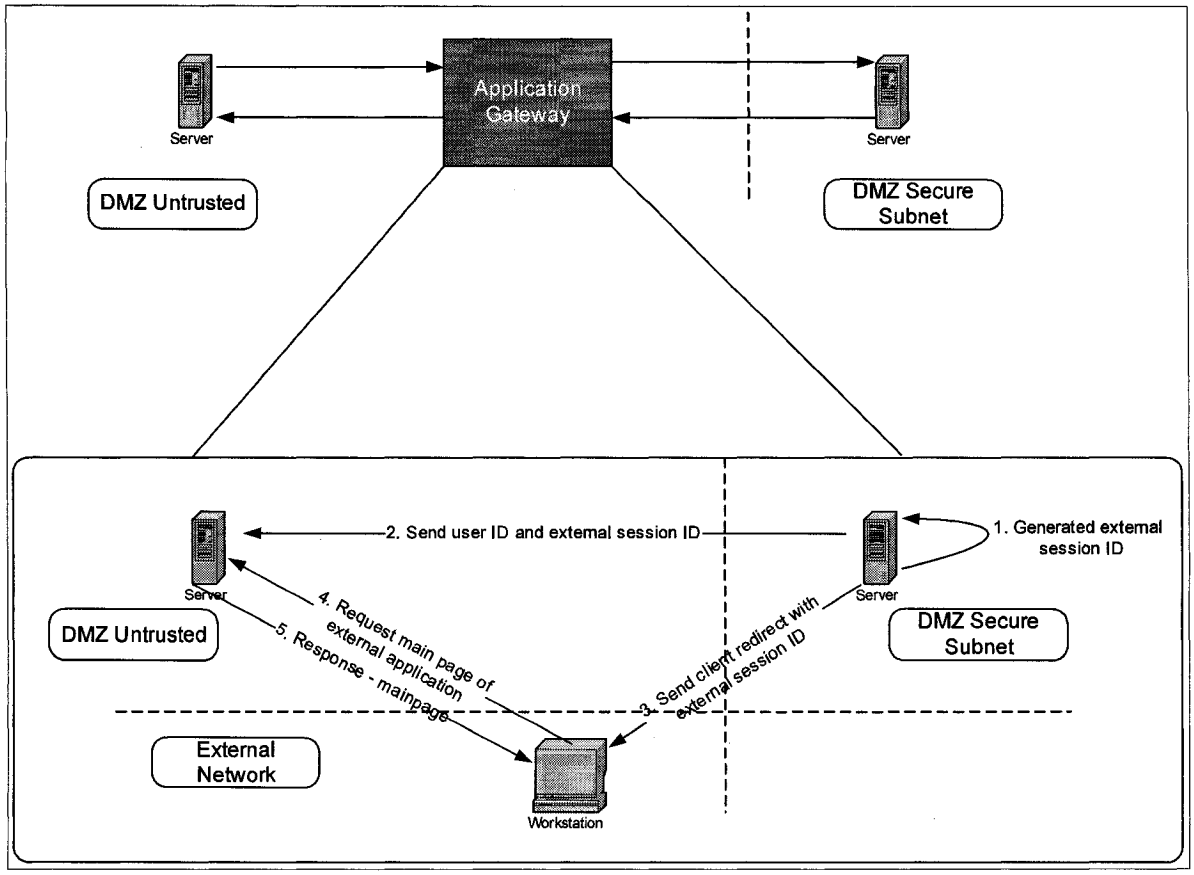
1. The client should be able to access systems related to the AAP located outside the AAP subnet as long as he/she is authenticated to the AAP system.

2. If the client has already logged into the AAP system and accessed a system located outside the AAP subnet, he/she should be able to return to the AAP system without re-authenticating.
3. If the client logs into a system outside the AAP subnet, he/she should not be granted access to the AAP automatically.

In addition to these general design objectives, there are some additional security considerations that should be addressed from the point of view of the AAP system. The main consideration is that the application gateway should not provide a weaker point of entry to the AAP system than any application inside the secure subnet. Therefore, the gateways need to be designed in such a way that a malicious user will be just as likely to guess a valid credential for a specific user in the system as he/she would had he/she queried some part of the AAP system directly.

Even though the specific gateways to the OSJS system and the RTA system are slightly different, the proposed design is general enough that it should be easy to add both of these services as well as any additional services in the future. In this solution, when the user requests the service outside the AAP subnet, a connection between the host inside the subnet and the service provider outside the subnet is established directly within the DMZ. This creates an out-of-band connection with respect to the client which will help to protect the user's credentials. In this out-of-band connection, the AAP generates a valid session ID token for the external system and sends it directly to the external host with the user's ID so the external system will know which user is accessing the system. Once this has occurred, the client will be redirected to the external system with the external session ID present as a URL parameter.

Figure 6.6: Detailed Overview of Inter-DMZ Gateway



This design provides a standard interface for all external applications to be linked to the AAP. The session ID generation module can be developed independently of the rest of the systems involved. This allows the AAP to interact with external systems which may have session IDs which are different from those used in the AAP. This solution fulfills all of the design objectives presented above. The client can access the external system from the AAP as shown in Figure 6.6 above. In this solution, the user is given an ID to access the external system which is different from the credentials used to access the AAP system. Since the AAP systems have been reorganized into a subnet in order to maintain control of their credentials, the AAP credentials are not sent to the external system but they are still stored on the client's browser. This allows the user to return to the AAP through a simple link provided in the external application which directs the user to the AAP main page. This fulfills the second design objective since the client, if they have already been authenticated in

the AAP, will be able to return to the AAP from the external application. Finally, the third design objective is also fulfilled since the user will be able to login to the external application but will not have access to the AAP from the external application without reauthenticating. This is due to the fact that the client will not have the necessary AAP credentials to access the AAP and they will be redirected to the AAP login screen. In addition to fulfilling all of the proposed design objectives, this system also ensures that the AAP system cannot be attacked from a potentially weaker external system. This is due to the fact that external systems would no longer make direct calls to the AAP system in order to authenticate users. All information that could be used to identify the client is sent via an out-of-band connection so it cannot be compromised by a malicious client.

6.3.3. DMZ/Internal Network Gateway

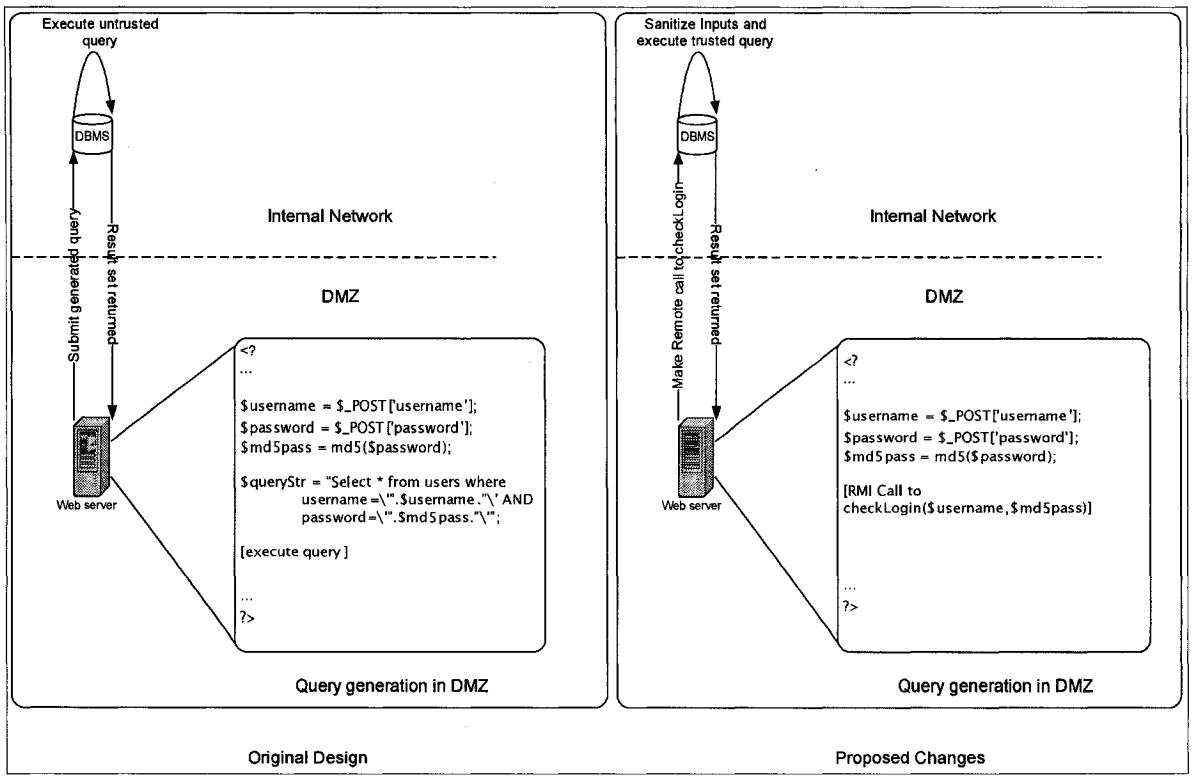
The main threats to the internal network from the hosts in the DMZ will come from the ability of those hosts to create and execute any type of query they wish. This gateway is used to help mitigate the vulnerabilities described in section 5.2.1. In the current AAP system, client input is used to create SQL queries in the web tier which is then sent to the database server as a trusted command. The side effect of that is if the user input is not sanitized to remove potentially malicious data, it can lead to a SQL injection attack. A SQL injection vulnerability can essentially provide the client with a direct link to the internal database and allow him/her to execute any SQL command he/she wishes which violates the trust structure previously proposed. The main design considerations when developing the application gateway between the DMZ and the internal network (denoted as "4" in Figure 6.2) are as follows:

1. The application servers need to be granted access to the relevant information in the database.
2. The application server should not be able to execute any queries in the database outside those used to fulfill legitimate requests.

Part of the problem with how the current system accesses the internal database is that the internal database completely trusts the hosts in the DMZ. The queries are performed by

simply opening a connection to the database server and passing any arbitrary query. The connection from the DMZ hosts and the database server needs to be limited so that only authorized queries are performed. Therefore, the actual query construction needs to be moved inside the internal network and out of the web applications running in the DMZ. This is illustrated in Figure 6.7 below.

Figure 6.7: Proposed Restructuring of DMZ/Internal Network Gateway



In the figure above, the proposed solution shows a remote method invocation (RMI) call to the internal database. The goal here is not to specify a specific technological way of creating the solution but to illustrate that the query executed by the DBMS should be generated by the DBMS and not in the DMZ. The goal here is to limit the attack surface of the client by hardening all parts of the system even if the client is not directly interacting with those components. In Figure 6.7, the database is accessed by a remote function call of some kind as opposed to running the query with a wide open command shell. This way the trust boundary between the DMZ and the internal network is hardened because the internal

database server is longer giving the web application server full access. As far as the actual technology used to create the remote function calls, there are a number of options. Most DBMS applications offer the developers a way to write stored procedures which can be executed by outside applications or by users directly through a command prompt. The stored procedure can be written in such a way that only the required information is passed and that information can be scrutinized by the procedure before executing any queries and returning any data. It is still possible to inject SQL code into the stored procedure if the inputs are not sanitized but this design offers the internal database a way to protect itself from potentially malicious users. Data sanitation can and should be performed in the web tier as well but with this design, if a section of the web tier does not have any sanitation, the DBMS is still able to resist attack. In the case where the stored procedure functionality is not offered, there are a number of efficient distributed programming platforms that can be used to create a layer of abstraction between the web tier and the data store. An example of this type of data store could be the LDAP service. In this case, an interface is needed to allow the web tier to make requests to the LDAP server while maintaining the query generation functionality on the internal network and not in the DMZ. Since the server does not have a feature which allows stored procedures to be created, a web service interface using the Simple Object Access Protocol (SOAP) is used to create a web service that is accessible to the hosts in the DMZ. This solution is similar to that illustrated in above in Figure 6.7. This same type of solution can be used to limit access to any other type of existing data store, such as an XML data store accessed through XPATH, or future data stores that allow queries to be generated on the fly. One final solution that should also be considered is the use of prepared queries or prepared statements. Prepared statements are a way of creating dynamic SQL queries within applications that will maintain the integrity of the desired query. Prepared statements are supported by most modern DBMS applications as they are part of the SQL-92 ANSI standard [45]. They are prepared in a similar way as the flawed example shown in Figure 6.7 but with one main difference. When the prepared statement is created, a template of the desired query is passed to the database where it is parsed or "prepared". A place holder character, in this case the "?", is used to mark where the dynamic inputs are to be placed when executing the query. Parsing the query before the parameters are known has the effect

of predetermining the syntax of the query. That means that even if potentially harmful inputs are passed to the statement, the DBMS will interpret them as data and not as SQL commands. Then the statement is executed by setting the parameters to the values passed to the web application. Although it may seem that the internal database is once again trusting the web-tier to create the query with the appropriate data, this is not the case. With a prepared statement, the query definition and the user supplied arguments are passed separately. This means that the query is actually created by the DBMS and malicious input is rendered completely benign since the statement structure is determined before the input is known. Prepared statements are not much more difficult to use from a development point of view since most modern web-application technologies have the required functionality built in. In the case that a particular framework does not have this functionality, prepared statements can be done entirely in SQL since it is part of the standard.

6.4. Summary

In section 5.2, we described the vulnerabilities found in the AAP system during our security evaluation. In this chapter, we presented possible mitigation solutions to the problems found with the AAP system. First we reexamine the trust boundaries currently in the system and offer a stricter set of trust boundaries. This will help to compartmentalize the system in such a way that each component can be designed so that no input is trusted. The system is divided into compartments which communicate through various application gateways. Specifically, we addressed the issue of untrusted hosts located within the DMZ of the organization. In the following chapter, we give our closing remarks as well as discuss topics related to this thesis that provide interesting challenges for the research community.

7. Conclusions and Future Considerations

7.1. Conclusions

In this work, the goal was to perform a study of third party security evaluations of web-based systems. The testing methodology presented can be used as a guideline for performing future application evaluations. A second goal of this portion of the work was to provide information regarding a number of non-technical issues that testing teams will need to deal with when performing security evaluations. Third party evaluations are an important part of the software development life cycle. The emphasis should be placed on the term "third party". It is important that experienced security professionals be consulted since they will have an extensive knowledge of security issues related to the system but they also have the ability to look at the system from a completely objective point of view.

In addition to providing guidelines for performing a third-party security evaluation, another goal of this work was to determine the root architectural causes of the most common web application flaws in the AAP system. Many flaws found in web applications can be traced back to improper design consideration and assumptions. The issue of dealing with inter-organizational hosts that are not necessarily trusted or run by multiple groups with different policies is addressed in some detail. This particular scenario may be overlooked because of the assumption that if the host is in the organization's trusted network that they can be trusted. We present an example where that is not the case as well as provide a potential solution to address this issue.

Finally, we described various attacks which focus on the session management systems used by web applications. Due to the many types of vulnerabilities that affect web-based systems, it is important to be aware of best practices for how to properly manage user sessions. And as more transactions are performed over the Internet, the issue of authentication and authorization will become even more important.

7.2. Future Considerations

Although the work done in the area of risk management was described in some detail, one issue that could be addressed further is the idea of presenting the potential risk to non-technical management teams. Security evaluations are in large part a technical exercise. In addition to this, it is difficult, if not impossible, to really quantify the value of performing regular risk assessments of information systems since there is no tangible return on investment to present to management personnel who make financial decisions. If a new feature is be added to a software package, it can be seen and used. If an advertising campaign is launched, the ads can be seen and revenue changes monitored to determine the return on investment. Security improvements are somewhat intangible and success is relatively immeasurable since success is based on the absence of events. And although statistics can be provided after an evaluation is performed to show the number of bugs found or any other information the evaluation team wishes to present, convincing organizations to agree to third party audits in the first place can be quite difficult because of the intangible nature of IT security. An interesting study related to this may be to test whether there is a correlation between the amount of money spent on third party evaluations and the perceived losses due to information security breaches. Such a study could provide added incentive to non-technical decision makers to have their IT systems evaluated in order to better protect their assets and clients.

The severity ranking of flaws found during security evaluations is relatively subjective. Although some work has been done from a risk management point of view, there is room to expand in the area of vulnerability severity metrics. Simply stating that a vulnerability is severe because a successful compromise can lead to remote control of a host is not enough information to allow the mitigation team to prioritize which flaws to deal with first. In addition to considering the potential consequences of a successful exploit, other metrics such as cost of mitigation, number of bugs concurrently mitigated by a particular solution, and ease of exploitation are just some examples that would be useful to consider in future severity ranking solutions.

Related to the notion of metrics relating security vulnerabilities is the subject of

comparative analysis of various web-based solutions. In a situation where an organization wishes to deploy a web-based application and has a number of options, there needs to be a way to objectively rank the vulnerabilities in each of the solutions and then determine which application provides the more secure solution. This type of analysis is difficult because not all vulnerabilities will provide the same level of access when exploited and the amount of effort to mitigate vulnerabilities in a system also varies. For example, suppose two systems have been evaluated. The first system has five severe vulnerabilities which would give an attacker full access to all data managed by the system. Additionally, these vulnerabilities are not easy to exploit and two relatively easy solutions can be used to fix all five vulnerabilities. The second system has 20 vulnerabilities which would give an attacker partial access to some data managed by the system. These vulnerabilities are easy to exploit and would require 18 solutions to mitigate all 20 vulnerabilities. If one had to choose on of these two systems to put into production, which one is the most secure (or poses the least risk to the organization)? This is not a trivial problem to solve and so further research in this area would be beneficial.

Another interesting issue to consider in future research is the issue of regression testing with respect to security evaluations. Once an evaluation has been completed and mitigation solutions suggested and implemented, how can the organization test their system to ensure that the solutions used actually solve the problems. Additionally, how can the organization be assured that new (potentially more serious) security vulnerabilities are introduced when solutions to previously found bugs are deployed? Although mitigation solutions can be verified by having another third-party security evaluation done on the system. However, it would be more beneficial to the organization if there was a way for them to verify that mitigation solutions deployed actually fix the flaws found by the testing team without having to involve the testing team.

The issue of session management, specifically session hijacking prevention, is another area of research that is important to the web application security field. Although more steps are being taken by application developers to help lessen the effect of session hijacking or user credential disclosure, these problems are still quite difficult to solve. Although proper credential management and generation as well as good design practices

such as those described in this work will help to lessen the attack surface, if a solution could be developed so that this problem were completely eliminated, it would make modern web-based systems far more secure and may offer consumers more piece of mind.

8. Bibliography

[1] "A Guide to Building Secure Web Applications and Web Services", OWASP Foundation, 2006.

[2] "ACK Scan [-sA]", http://www.nmap-tutorial.com/html/node13.html, accessed September 16th, 2006.

[3] Alberts, C., Dorofee, A., "Managing Information Security Risks: The OCTAVE Approach", Addison Wesley, 2003.

[4] Ammann, P., Wijesekera, D., Kaushik, S., "Scalable, Graph-Based Network Vulnerability Analysis", *Proceedings of the 9th ACM conference on Computer and Communications Security,* November 2002.

[5] "AOL Proxy Info", http://webmaster.info.aol.com/proxyinfo.html, accessed April 28, 2006.

[6] "ARIN: WHOIS Database Search", http://ws.arin.net/whois, accessed April 30, 2006.

[7] Baharin, K., Din, N., Jamaludin, Z., Nooritawati, T., "Third Party Security Audit Procedure for Network Environment", *Proceedings of the 4th National Conference on Telecommunications Technology, Malaysia*, 2003.

[8] Bergman, M., "The Deep Web: Surfacing Hidden Value", *Deep Content White Paper*, 2001.

[9] Bertino, E., Bruschi, D., Franzoni, S., Nai-Fovino, I., Valtolina, S., "Threat Modeling For SQL Servers", *Conference on Communications and Multimedia Security*, September 2004.

[10] Bishop, M, "Computer Security: Art and Science", pg. 314, theorem 12-2, Addison Wesley, 2003.

[11] Blum, L., Blum, M., Shub, M., "A Simple Unpredictable Pseudo Random Number Generator", *SIAM Journal on Computing*, Vol. 15, Issue 2, pg. 364-383, May 1986.

[12] "Bugtraq – SecurityFocus", http://www.securityfocus.com/archive/1, accessed August 8th, 2006.

[13] "CERT Advisories", http://www.cert.org/advisories/, accessed August 8th, 2006.

[14] CERT Coordination Center, http://www.cert.org, accessed July 28th, 2006.

[15] Chadwick, D., "Threat Modelling for Active Directory", *Conference on*

Communications and Multimedia Security, September 2004.

[16] "Common Vulnerabilities and Exposures", http://cve.mitre.org, accessed August 8th, 2006.

[17] Dalton, G., Mills, R., Colombi, J., Raines, R., "Analyzing Attack Trees Using Generalized Stochastic Petri Nets", *Proceedings of the 2006 IEEE Workshop on Information Assurance*, West Point, NY, 2006.

[18] De Cock, D., Wouters, K., Schellekens, D., Singelee, D., Preneel, B., "Threat Modelling for Security Tokens in Web Applications", *Conference on Communications and Multimedia Security*, September 2004.

[19] de Vivo, M., Carrasco, E., Isern, G., de Vivo, G., "A Review of Port Scanning Techniques", *ACM SIGCOMM Computer Communication Review, Vol. 29, Issue 2,* April 1999.

[20] Desmet, L., Jacobs, B., Piessens, F., Joosen, W., "Threat Modelling for Web Services Based Web Applications", *Conference on Communications and Multimedia Security*, September 2004.

[21] Endler, D, "Brute-Force Exploitation of Web Application Session IDs", iAlert White Paper, 2001.

[22] "Ethereal: A Network Protocol Analyzer", http://www.ethereal.com, accessed April 30, 2006.

[23] Farmer, D., Spafford, E., "The COPS Security Checker System", *Proceedings of the Summer USENIX Conference*, 1990.

[24] Farmer, D., Venema, W., "Improving the Security of Your Site By Breaking Into It", Usenet posting in comp.security.unix, December 3rd, 1993.

[25] Fielding, R, Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616, 1999.

[26] Fu, K., Sit, E., Smith, K., Feamster, N., "Dos and Don'ts of Client Authentication on the Web", Proceedings of the 10th USENIX Security Symposium, August 2001.

[27] Fydor, "Remote OS Detection via TCP/IP Stack FingerPrinting", http://insecure.org/nmap/nmap-fingerprinting-article.txt, October 18th, 1998.

[28] Fydor, "The Art of Port Scanning", *Phrack Magazine, Vol. 7, Issue 51,* September 1st, 1997.

[29] Gane, C., Sarson, T., "Structured Systems Analysis: Tools and Techniques", Prentice-Hall, 1979.

[30] Geer, D., Harthorne, J., "Penetration Testing: A Duet", *Proceedings of the 18th Annual Computer Security Applications Conference*, 2002.

[31] Grimm, R., Eichstadt, H., "Threat Modeling for ASP.NET", *Conference on Communications and Multimedia Security*, September 2004.

[32] Hasan, R., Myagmar, S., Lee, A. J., Yurcik, W., "Toward a Threat Model for Storage Systems", *International Workshop on Storage Security and Survivability (StorageSS)*, 2005.

[33] Herzberg, A., "The Unprotected Login Hall of Shame", http://www.cs.biu.ac.il/~herzbea//shame/index.html, accessed on September 16th, 2006.

[34] Herzog, P., "OSSTMM 2.1.1.: Open-Source Security Testing Methodology Manual", *Institute for Security and Open Methodologies*, September 2005.

[35] Howard, M., LeBlanc, D., "Writing Secure Code 2nd Edition", pg. 69 – 124, Microsoft Press, 2003.

[36] Howes, T., "The String Representation of LDAP Search Filters", RFC 2254, December 1997.

[37] Howes, T., Smith, M., "The LDAP URL Format", RFC 2255, December 1997.

[38] "HTTP cookie – Wikipedia, the free encyclopedia", http://en.wikipedia.org/wiki/HTTP_cookie, accessed on March 30th, 2006.

[39] "HTML tag reference guide - <META http-equiv set-cookie>", http://www.html-reference.com/META_httpequiv_setcookie.htm, accessed April 15th , 2006.

[40] "HTTPOnly Test Script and Browser Support Info", http://bradmetz.zapto.org/httponlytest.html, created April 28. 2006.

[41] Huang, Y., Huang, S., Lin, T., Tsai, C., "Web Application Security Assessment by Fault Injection and Behavior Monitoring", *Proceedings of the 12th international conference on World Wide Web,* May 2003.

[42] Huang, Y., Tsai, C., Lee, D., Kuo, S., "Non-Detrimental Web Application Security Scanning", *Proceedings of the 15th International Symposium on Software Reliability Engineering*, 2004.

[43] Huang, Y., Yu, F., Hang, C., Tsai, C., Lee, D., Kuo, S., "Securing Web Application Code by Static Analysis and Runtime Protection", *Proceedings of the 13th International*

Conference on World Wide Web, May 2004.

[44] "IOActive", http://ioactive.com, accessed on July 26th, 2006.

[45] "ISO/IEC 9075:1992, Database Language SQL" ANSI SQL:92 Standard, July 1992.

[46] Jovanovic, N., Kruegel, C., Kirda, E., "Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper)", *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, 2006.

[47] Jovanovic, N., Kruegel, C., Kirda, E., "Precise Alias Analysis for Static Detection of Web Application Vulnerabilities", *ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, June 2006.

[48] Kals, S., Kirda, E., Kruegal, C., Jovanovic, N., "SecuBat: A Web Vulnerability Scanner", *Proceedings of the 15th international conference on World Wide Web,* May 2006.

[49] Kelsey, J., Schneier, B., Ferguson, N., "Yarrow 160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator", *Sixth Annual Workshop on Selected Areas in Cryptography*, August 1999.

[50] Klein, A., "Blind XPATH Injection", Sanctum White Paper, 2004.

[51] Klein, A, "'Divide and Conquer' HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics", Sanctum Inc. White Paper, 2004.

[52] Kolsek, M., "Session Fixation Vulnerability in Web-Based Applications", ACROS Security White Paper, December 2002.

[53] Kristol, D., Montulli, L., "HTTP State Management Mechanism", RFC 2109, 1997.

[54] Kristol, D., Montulli, L., "HTTP State Management Mechanism", RFC 2965, 2000.

[55] Lerida, J., Grackzy, S., Vina, A., Andujar, J., "Detecting Security Vulnerabilities in Remote TCP/IP Networks: An Approach Using Security Scanners", *Proceedings of the IEEE 33rd Annual International Carnahan Conference on Security Technology*, October 1999.

[56] Lethbridge, T., Laganiere, R., "Object-Oriented Software Engineering: Practical Software Development Using UML and Java 1st Edition", Chapter 10: Testing and Inspecting to Ensure High Quality, pg. 349, *McGraw-Hill Education*, 2001.

[57] Linde, R., "Operating System Penetration", *AFIPS National Computer Conference Proceedings Vol. 44 pg. 361 – 368,* 1975.

[58] Lo, E., Marchand, M., "Security Audit: A case study", *Canadian Conference on Electrical and Computer Engineering*, 2004.

[59] Martin, M., Livshits, B., Lam, M., "Finding Application Errors and Security Flaws Using PQL: A Program Query Language", *Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications*, October 2005.

[60] Mauw, S., Oostdijk, M., "Foundations of Attack Trees", *Eighth Annual International Conference on Information Security and Cryptology*, Seoul, South Korea, 2005.

[61] McDermott, J., "Attack Net Penetration Testing", *Proceedings of the 2000 workshop on New security paradigms*, 2001.

[62] McGraw, G, Viega, J, "Building Secure Software" Chapter 10: Randomness and Determinism, Pg. 263, Addison Wesley, Sept. 2003.

[63] "Mitigating Cross-site Scripting With HTTPOnly Cookies", http://msdn.microsoft.com/workshop/author/dhtml/httponly_cookies.asp, accessed April 28, 2006.

[64] Mohamed, A., "An Effective Modified Security Auditing Tool (SAT)", *23rd International Conference on Information Technology Interfaces*, 2001.

[65] Morris, R., Thompson, K., "Password Security: A Case Study", Communications of the ACM, November 1979.

[66] Murphey, L, "Secure Session Management", GSEC Pratical Assignment Paper, January 2005.

[67] Myagmar, S., "Threat Modeling as a Basis for Security Requirements", *Proceedings of the IEEE Symposium on Requirements Engineering for Information Security*, August 2005.

[68] Naor, M., "Verification of a Human in the Loop or Identification Via the Turing Test", Unpublished Manuscript, 1996.

[69] "National Vulnerability Database Statistics", http://nvd.nist.gov/statistics.cfm?results=1, accessed September 16th, 2006.

[70] "Nessus", http://www.nessus.org, accessed August 8th, 2006.

[71] "Netcraft", http://www.netcraft.com, accessed August 15th, 2006.

[72] "Netcraft: Security Advisory 2001-01.1 – Predictable Session IDs", http://news.netcraft.com/archives/2003/01/01/security_advisory_2001011_predictable_sessio

n_ids.html, accessed on April 22nd , 2006.

[73] "Nikto", http://www.cirt.net/code/nikto.shtml, accessed August 16th, 2006.

[74] Pettit, S., "Anatomy of a Web Application: Security Considerations", Sanctum Inc. white paper, 2001.

[75] Phillips, C., Swiler, L., "A Graph-Based System for Network Vulnerability Analysis", *Proceedings of the 1998 Workshop on New Security Paradigms,* 1998.

[76] "phpBB.com::Creating Communities", http://www.phpBB.com, accessed August 7th, 2006.

[77] "Pyramid BenHur Default Firewall Weakness", http://www.securityfocus.com/bid/5279, accessed September 16th, 2006.

[78] "RNGCryptoServiceProvider Class", http://msdn2.microsoft.com/en-US/library/system.security.cryptography.rngcryptoserviceprovider(VS.80).aspx, accessed April 19th, 2006.

[79] Saitta, P., Larcom, B., Eddington, M., "Trike v.1 Methodology Document", *ToorCon 2005*, 2005.

[80] Salter, C., Saydjari, O., Schneier, B., Wallner, J., "Toward a Secure System Engineering Methodology", *Proceedings of the 1998 Workshop on New Security Paradigms,* September 1998.

[81] "SamSpade.org", http://samspade.org/, accessed August 15th, 2006.

[82] Schneier, B., "Attack Trees: Modeling Security Threats", *Dr. Dobb's Journal*, Dec. 1999.

[83] Schneier, B., "Secrets and Lies: Digital Security in a Networked World", pg. 318 – 333, John Wiley and Sons, 2000.

[84] Scott, D., Sharp, R., "Abstracting Application-Level Web Security", *Proceedings of the 11th International Conference on the World Wide Web (WWW2002),* May 2002.

[85] Schreiber, T, "Session Riding: A Widespread Vulnerability in Today's Web Applications", SecureNet Whitepaper, Dec. 2004.

[86] "SecureRandom (Java 2 Platform SE v1.4.2), http://java.sun.com/j2se/1.4.2/docs/api/java/security/SecureRandom.html , accessed April 19th, 2006.

[87] SecurityFocus Vulnerability Database, http://www.securityfocus.com/vulnerabilities, accessed July 28th, 2006.

[88] Seo, J., Kim, H., Cho, S., Cha, S., "Web Server Attack Categorization based on Root Causes and Their Locations", *Proceedings of the International Conference on information Technology: Coding and Computing*, 2004.

[89] Sharma, A., Martin, J., Anand, N., Cukier, M., Sanders, W., "Ferret: A Host Vulnerability Checking Tool", *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing*, 2004.

[90] Sheyer, O., Haines, J., Jha, S., Lippmann, R., Wing, J., "Automated Generation and Analysis of Attack Graphs", *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2002.

[91] Sheyner, O., Wing, J., "Tools for Generating and Analyzing Attack Graphs", *Proceedings of Formal Methods for Components and Objects*, 2005.

[92] Singh, S., Lyons, J., Nicol, D., "Fast Model-Based Penetration Testing", *Proceedings of the 2004 Winter Simulation Conference,* 2004.

[93] Skaggs, B., Blackburn, B., Manes, G., Shenoi, S., "Network Vulnerability Analysis", *Midwest Symposium on Circuits and Systems,* August 2002.

[94] Spett, K., "SQL Injection: Are Your Web Applications Vulnerable?", SPI Dynamics White Paper, 2005.

[95] "SquirrelMail – Webmail for Nuts!", http://www.squirrelmail.org, accessed August 7th, 2006.

[96] Stefan, J., Schumacher, M., "Collaborative Attack Modeling", *Proceedings of the 2002 ACM symposium on Applied computing,* 2002.

[97] The Open Web Application Security Project, "OWASP Top Ten Project", http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project, accessed August 7th, 2006.

[98] Tidwell, T., Larson, R., Fitch, K., Hale, J., "Modeling Internet Attacks", *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, June 2001.

[99] Tripunitara, M., Dutta, P., "Security Assessment of IP-based Networks: A Holistic Approach", *Proceedings of INET'99,* June 1999.

[100] "Vulnerability Scanners", http://www.cotse.com/tools/vuln.htm, accessed August 8th, 2006.

[101] Wack, J., Tracy, M., Souppaya, M., "Guideline on Network Security Testing", *NIST Special Publication 800-42, US Department of Commerce*, October 2003.

[102] Wahl, M., "A Summary of the X.500(96) User Schema for use with LDAPv3", RFC 2256, December 1997.

[103] Wahl, M., Coulbeck, A., Howes, T., Kille, S., "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.

[104] Wahl, M., Howes, T., Kille, S., "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997.

[105] Wahl, M., Kille, S., Howes, T., "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

[106] "Web Application Security Consortium: Threat Classification", http://www.webappsec.org/projects/threat, accessed on March 30th , 2006.

[107] "Web Developer Extension", http://chrispederick.com/work/webdeveloper/, accessed on April 30, 2006.

[108] "Webscarab", http://www.owasp.org/software/webscarab.html, accessed on April 30, 2006.

[109] "Welcome! - The Apache HTTP Server Project" http://httpd.apache.org, accessed August 16th, 2006.

[110] Whitaker, A., Newman, D., "Penetration Testing and Network Defense", Chapter 5: Performing Host Reconnaissance, pg. 96 – 105, *Cisco Press,* November 2005.

[111] Wikipedia, http://wikipedia.org, accessed July 28th, 2006.

[112] Xu, D., Nygard, K., "A Threat-Driven Approach to Modeling and Verifying Secure Software", *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005.

[113] Yourdon, E., Constantine, L., "Structured Design", Yourdon Press, 1975.

[114] Zimmer, D., "Real World XSS", Venture Digital White Paper, 2003.

Appendix A: Sample Security Policy Description

The security policy here is defined in terms of the student user group. Since the main method a student accesses the AAP is through the web-based interface, we will describe the security policy in place from the information gathered and observed while performing the security evaluation. It should be noted that no explicit policy document was obtained from the organization or any other source so the policy described here is that observed while performing the audit along with certain assumptions made regarding access to certain parts of the system.

A.1. Access Rights to Student Users

Students are able to access their own personal information but should not have any type of access to the information of other users in the system. Students can view their academic information including grades, course timetables, exam timetables, contact information, and password management information. Students can also access applications such as the Online Student Job Search (OSJS) and the Remote Teaching assistant applications if they are eligible to do so. Since not all students require access to the OSJS, access to this application is only presented on the AAP main page of those students who have access to that system.

In addition to viewing personal information, students are also able to make changes to some of the information stored in the AAP. This information includes contact information (mailing address, phone number, e-mail address), and password management information (actual passwords as well as information used to restore passwords). Students should not be able to modify information about themselves regarding grades or financial information. Students should not have access to services reserved for other user groups.

Once a valid student user has logged into the AAP system, he/she should have access to any system associated with the AAP system that he/she should have access to regardless of which host the service is offered on. In other words, AAP access should provide a single-sign-on solution that allows verified users to access services hosted on a number of different servers within the organization's domain without having to repeat the initial login procedure for each server accessed. This implies that user credentials must be transmitted between servers without any direct end-user interaction.

Users should be given a finite number of chances to log into a particular account after which the account will be disabled. The organization has stated that a user have 5 attempts to log into a given account after which the account is locked out. To enable a locked account, the user must contact the organization to restore the account and reset the password. This can be done through the website or by telephone. A locked account will not be usable until it is reset through one of these two methods. After a user successfully logs into an account he/she will only have a finite amount of time to use the AAP system. Once that time limit has expired the user will be asked to repeat the login process where he/she will be assigned a new session key and the session will be reset.

Although students can access systems outside the AAP hosts described above through

the AAP system, the reverse should not be true. That is to say, for example, that if a student logs into the RTA service, the user will not be able to access the AAP without first authenticating to the AAP system. The same is true for all services run outside the servers administered by the AAP group. These include servers used to host RTA and OSJS applications.

A.2. Mechanisms Used to Enforce the Security Policy

A.2.1. Authentication Mechanism

Since the AAP service provides access to a large amount of sensitive information, it must ensure that only authorized users are able to access the system. This is done by requiring the user to enter his/her userID and password into the HTML form provided. Once the user has successfully authenticated himself/herself, the system will assign a temporary session ID which is stored on the user's browser in the form of a cookie along with the user's ID. This session ID along with the user ID are to be used by all other AAP applications to determine whether the user has access to the requested service or not. The cookies containing the user credentials are configured in such a way that they are always transmitted to hosts within the organization's domain therefore they are always available to hosts within the domain. The applications being accessed then obtain those credentials from the cookies and verify the user with the AAP. All the session IDs are stored and managed by the AAP system.

Session IDs and, consequently, user sessions are subject to a timeout period of 45 minutes. This means that any session ID assigned by the AAP will only give access to the system (for the associated account) for a maximum of 45 minutes. Once a session ID has expired, the user presenting that ID will be redirected back to the login screen where he/she will be asked to reenter his/her user credentials.

A.2.2. Account Lockout and Password Recovery

The account lockout policy described by the organization is one where a given account is locked out after 5 invalid login attempts are made. It was found through testing that the actual number of invalid tries allowed is 8 after which the account will be locked out. When an account is locked out, the owner of the account must contact the organization's help desk either by telephone or online by submitting a form requesting assistance. If a password is reset because of an account lockout, the password is reset to something that (presumably) only the legitimate user knows. In this case, the password is reset to the user's mother's maiden name. The user can then login to his/her account using this password.

The AAP system also offers an "I forgot my password" option. This allows a user to reset his/her password by correctly answering a secret question. Upon correctly answering the secret password, a random password is created and is e-mailed to the user's e-mail account. The user can then login to the e-mail account and retrieve the new password and use it to log into the AAP. To use this service the user will have to have entered a secret question and answer to be used by this application and the user must have provided a valid e-

mail account. If one of these two conditions is not met then the user must contact the organization and the password is reset in the same way as though the account were locked out. It should be noted that the password used to reset the account in the case where the account is locked out differs from the case where the user resets his/her password using the secret question.

A.2.3. Single Sign On

The services offered through by the AAP are hosted on a number of different servers. One of the goals of the designers of this system is to offer a single-sign-on solution, meaning the user only needs to log in at one location in order to access all of the services he/she is authorized to use. When accessing services hosted on machines outside those administered by the AAP group, a login bypass page is generally used to authenticate the user. This bypasses login screens that are used exclusively by the services themselves (not the AAP login screen). Examples of these applications are the RTA and the OSJS previously described. Both of these services allow the user to login outside of the AAP but, as stated above, do not provide access to the AAP system without prior authentication to said system. The login bypass page takes the user's AAP credentials and presumably checks their validity. If they are still valid then the user is granted access to the service.

A.2.4. Access Rights Enforcement

As mentioned above, there is certain information in the AAP that students are allowed to edit and information that they can only view and not edit. In the case of information that is read-only, access is granted based on the user ID/session ID combination sent by the browser. The credentials are then checked in the AAP to ensure they are still valid and the information is presented to the user. There are no obvious locations where the user could enter or change information that they are not supposed to be able to change. In the case where information is both viewable and editable, the user is presented with a form where he/she can view the information currently stored in the system and also make and submit any changes to that information.

When the student logs into the AAP, the main page menu is generated dynamically which will present the user with links to all the systems he/she has access to. This ensures that users are not presented with links to applications that they do not have access to. For example, if a user should have access to the RTA application but not OSJS application, he/she will be presented with a link to the RTA and not the OSJS.

Appendix B: Sample Vulnerability Report

Title:	**Student Bulletin Board XSS Vulnerability**	*ID No:*	**BUG054**
Type:	XSS resulting in client-side code insertion	*Severity:*	Severe
Impact:	This vulnerability allows an attacker to insert malicious script code into the student bulletin board which could allow the attacker to do numerous things. This provides the attacker with an attack vector which could be used to steal a victim's credentials or present a fraudulent web page under the guise that it came from the legitimate site. This is a fairly severe bug due to the fact that this code is entered into the bulletin board that is accessible to the public and because it resides on the same domain as every other online service offered by the organization. This means that cookies assigned by the organization's websites can then be accessed and potentially stolen using an exploit which makes use of this vulnerability.		
Cause (Details):	Cross-Site scripting (XSS) vulnerabilities are caused by improper (or absent) user input validation. In this case, the assumption seems to be that the end-user is a trusted part of the system so there are no checks in place to ensure that he/she is entering valid information. This allows an attacker to create or update his/her bulletin board entries through the "Add or Update Your Entries" function where he/she can insert script code used to steal another user's credentials at a later time. This code will then be executed whenever anyone views the attacker's entry. This is what is known as a stored XSS attack since the malicious script code can be stored on the vulnerable server to accessed at a later time. This assumes that the victim has client-side scripting enabled but that seems to be a reasonable assumption given the heavy reliance on Javascript code elsewhere on the organization's site. This is also a default setting in most browsers. The insertion and modification form input should be sanitized as described in the following section.		
Possible Solution:	Although XSS attacks are quite common in web applications, they are ways to help mitigate the problem. One way to mitigate this type of attack is to sanitize the input provided by the user in order to remove any malicious script code. It is recommended that a white-list approach be used when developing the input sanitizer. This means that a detailed specification of what values should be allowed as input for each form element should be compiled and then the user input should be sanitized according to these criteria. Examples of this technique are, for a field where the user enters their name, only alpha-numeric characters should be allowed. In the case of a user's homepage URL being taken as input, the filter will be a little more complicated as certain special characters will need to be allowed such as the : and / characters. Characters that should be		

	filtered out of any user input are special characters such as <, >, (,), [,], ', “, ;, :, /, #, *, &, %. These characters are commonly used to inject code into a vulnerable web application. In this case, a form validation module could be developed which could then be used to validate all other forms used within the organization's site. There are a number of techniques and resources outlined by the Open Web Application Security Project which address this issue. These resources are available at www.owasp.org. There are a number of ways to exploit XSS vulnerabilities but in this case, the input space is fairly limited and so the input should be restricted using the information presented above. The database used to store this information will also need to be checked for any previously stored code as other users in the system may have entered malicious code into this application in the past. This can be done by finding all records in the database that contain the special characters mentioned above and removing them from those records.
Date Found:	March 3, 2010