2006-12

A secure alert system

Chew, Heng Hui.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/2392

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A SECURE ALERT SYSTEM

by

Heng Hui Chew

December 2006

Thesis Advisor:	Gurminder Singh
Co-Advisor:	Karen Burke

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2006	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE A Secure Alert System	**5. FUNDING NUMBERS**
6. AUTHOR(S) Heng Hui Chew	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The integrated mobile alert system (IMAS) is a mobile device messaging system that provides a means for people to stay connected and receive information in a modality that is constantly available to them. It was developed and built into a proof-of-concept (PoC) system at the Naval Postgraduate School (NPS) from commercial off the shelf (COTS) products. Like other systems, this system suffers from vulnerabilities because of bugs. These bugs come from (1) COTS products, (2) design of the system and (3) developed processes/applications. The study will review the design of IMAS, its processes and the COTS products. The focus of the study is to review these components and identify potential vulnerabilities. Furthermore, it will explain how these vulnerabilities may be exploited by probable threats. It will recommend solutions that can correct or prevent vulnerabilities. Lastly, the thesis will propose other measures that would make the system more secure.

14. SUBJECT TERMS IMAS, Integrated Mobile Alert System, Identify Potential Vulnerabilities, Mobile Device Messaging System, Secure Alert System	**15. NUMBER OF PAGES** 83
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

A SECURE ALERT SYSTEM

Heng Hui Chew
Engineer, Defence Science Technology Agency (DSTA)
B.Eng., Nanyang Technological University, 2000

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2006

Author: Heng Hui Chew

Approved by: Gurminder Singh
Thesis Advisor

Karen Burke
Co-Advisor

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The integrated mobile alert system (IMAS) is a mobile device messaging system that provides a means for people to stay connected and receive information in a modality that is constantly available to them. It was developed and built into a proof-of-concept (PoC) system at the Naval Postgraduate School (NPS) from commercial off the shelf (COTS) products. Like other systems, this system suffers from vulnerabilities because of bugs. These bugs come from (1) COTS products, (2) design of the system and (3) developed processes/applications. The study will review the design of IMAS, its processes and the COTS products. The focus of the study is to review these components and identify potential vulnerabilities. Furthermore, it will explain how these vulnerabilities may be exploited by probable threats. It will recommend solutions that can correct or prevent vulnerabilities. Lastly, the thesis will propose other measures that would make the system more secure.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Prof Gurminder Singh and Prof Karen Burke for their support and guidance towards this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

In today’s modern society, mobile devices are getting smaller, more powerful and packed with many functionalities that people use to surf the web, listen to their favorite songs or even watch a movie. It is no wonder that they have become an ubiquitous facet of our daily lives. Carrying a mobile device allows individuals to stay connected with their family, friends or business associates. People today carry more than one mobile device with them, e.g., cellular phones, smart-phones, BlackBerry, Portable Digital Assistant (PDA), pagers and laptops

A mobile alert is a mode of push communication whereby users are notified or informed of a piece of information/event by their mobile devices. This could be in the form of a page by a pager, simple text message (SMS), multimedia message (MMS), e-mail, voice-mail or instant message (IM) that is readily available to the mobile devices. Hence, with mobile alerts, people are able to stay connected to the world they live in. They may receive the latest stock price information from the stock market, a reminder of an appointment, a situational awareness report of the battleground by troops, an emergency alert from authorities and many others.

However, there are times when the mode of reaching a person is inappropriate, i.e., it is context-insensitive to the environment in which the person is located. A person in a meeting, watching a movie, or attending a concert, would prohibit the ring of a call. Therefore, it would be nice if an integrated mobile alert system (IMAS) was available - a system that could transform the mobile alerts into a mode that is appropriate and non-interruptive to the environment that person is in. The person may specify the environment and the mode of alerts they prefer to receive. That way, an alert could always reach the person regardless of the environment they are in. Two students, Hsu and Le [1], took this idea and built a proof-of-concept (PoC) system using commercial-off-the-shelf (COTS) products to illustrate the advantages of this system.

B. PURPOSE

The purpose of this thesis is to study and evaluate IMAS and identify vulnerabilities in the system so that solutions or mitigation methods can be proposed to secure the system. The vulnerabilities come from all components of IMAS. These components include:

- COTS products
- Design and implementation of IMAS
- Applications/functions

Figure 1 shows the software components that compose IMAS. It consists of in-house developed web applications and COTS products. The COTS products used in IMAS include (1) the operating system (Windows 2003 Server Standard Edition), (2) the web server (IIS6.0), (3) the scripting language (PHP) and (4) the database server (MySQL 5.0.18).The focus of this thesis is to review these components and identify potential vulnerabilities. Furthermore, it will explain how these vulnerabilities may be exploited by probable threats. It will recommend solutions that can correct or prevent vulnerabilities. Lastly, the thesis will propose other measures that would make the system more secure.

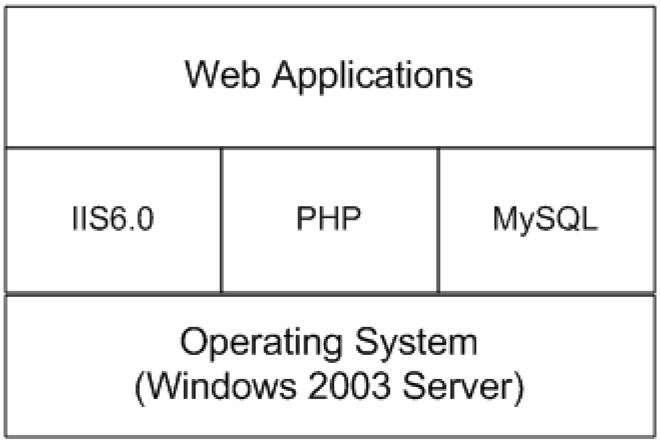


Figure 1. IMAS supporting components

C. ASSUMPTIONS

In this thesis, certain assumptions are being made so that the evaluation, solutions and mitigation methods proposed are valid in the assumed context and environment.

1. Personnel

It is assumed that all personnel accessing the system are trustworthy, i.e., their security clearances have been verified and they have received appropriate security training. This will prevent them from jeopardizing the system and putting the security of the system at risk. It will also be assumed that all personnel will follow all requisite procedures for system access.

2. Deployment of System

It is assumed that the system is deployed in a secure room where physical access to the room is controlled either by a personal identification number (PIN)-based entry system or a smart-card/magnetic-striped entry system. This allows only authorized personnel to access the room after they have cleared the identification and authentication process. In addition, this type of entry system provides an entry record for audit to be conducted and reviewed. The walls and ceiling of the room are assumed to be hardened to prevent easy penetration.

3. Physical Computer Security

The system is assumed to be protected by a common access card (CAC) system to prevent un-authorized personnel from accessing the system physically. Only authorized personnel with a valid CAC card authenticated by the system can access the applications. Procedures will be in-place requiring a person to either log off or lock their computers via “screen lock” when leaving the system unattended for a pre-determined period of time.

4. Limitations

Due to limited time available, the software codes of in-house developed processes/applications and the database design will not be analyzed in this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

II. COTS PRODUCTS AND THEIR VULNERABILITIES

A. INTRODUCTION TO IMAS

IMAS is created to provide a common medium for people to stay connected by receiving alerts across a wide variety of platforms. It allows users to configure their daily schedules and mode of alerts they wish to receive. The system can then coordinate the sources of alerts and reconstruct them to reach the users in the most appropriate and non-interruptive way.

Unfortunately, like many systems, IMAS suffers from security vulnerabilities that make it an easy target for attacks. The sources of these vulnerabilities include components that are used to develop the system and system design and implementation bugs. The components include COTS products like the hardware, operating systems, commercial applications and in-house developed applications. The attacks often lead to (1) system crashes, (2) corruption of data, (3) theft of information, or (4) sending of alerts to a wrong party. These attacks are disastrous and undesirable to both the system and the people involved. Hence, to ensure that IMAS is not subjected to subversion and exploitation, these vulnerabilities must be removed or prevented.

IMAS is composed of COTS products and in-house developed applications. The former provides the infrastructure and general services while the latter provides specific services pertaining to IMAS. COTS products are often vulnerable because of bugs in (1) software codes, (2) design errors in the applications and (3) mis-configurations in the installations. This chapter will review the COTS products used in IMAS and highlight common vulnerabilities reported in each component. Furthermore, it will recommend best-practiced methods to reduce and mitigate vulnerabilities. It will cover: (1) the operating system (Windows 2003 server standard edition) that manages the system, (2) the web server (IIS6.0) that hosts the web pages, (3) the scripting language (PHP) that is used to create the applications and (4) the database (MySQL) that stores and retrieves data.

B. WINDOWS 2003 SERVER

The operating system, Windows 2003 server standard edition in the case of IMAS, is the core of any system. It manages the resources of the system and ensures that all requests, locally or remotely, from the applications run timely and successfully. Windows 2003 server is considered the most secure and reliable operating system to date. It is built by Microsoft under the Trustworthy Computing initiative. Under this initiative, the programmers were trained to write secure codes. These codes are inspected and scrutinized for vulnerabilities. In addition, Windows 2003 server simplifies the usage and management of its suite of security functionalities, e.g., public key infrastructure (PKI) authentication services, single-sign-on (SSO) session and identity management control. The security functions prevent or minimize users from making mistakes that might violate the security of the system. A special tool, security configuration wizard (SCW), is also incorporated into the operating system to help users configure the system in the most secure mode. In this mode, all unused services are turned off by the operating system by default and no administrator is allowed to logon remotely using a null password. [2]

Despite this, Windows 2003 server still suffers from vulnerability attacks. Figure 2 shows a vulnerability report on Windows 2003 server standard edition published by Secunia [3], from January to November 2006. There were vulnerabilities reported every month except for March.

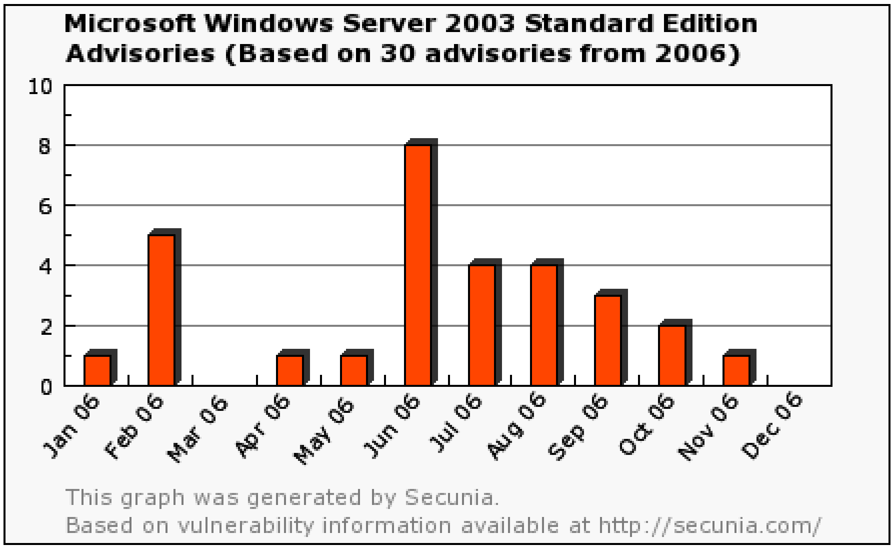


Figure 2. Number of vulnerabilities reported

Figure 3 shows that among the vulnerabilities reported, only 93% of them could be fixed with patches and no patches were available for the remaining 7%.

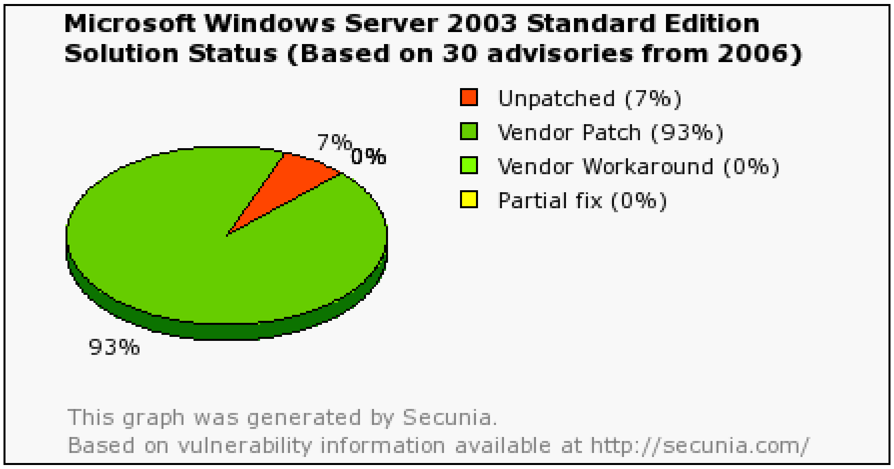
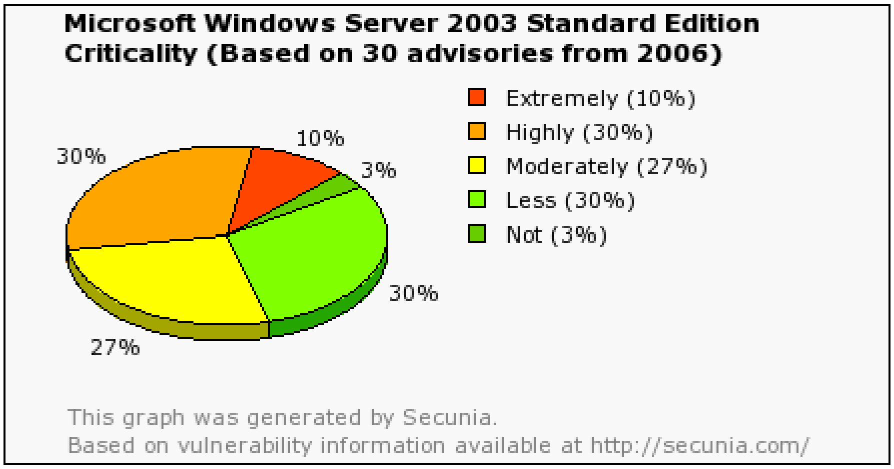


Figure 3. Solution Status

Figure 4 shows that among the vulnerabilities found, 10% were extremely critical and 30% were rated highly critical. This 40% requires immediate remedy actions to be taken.

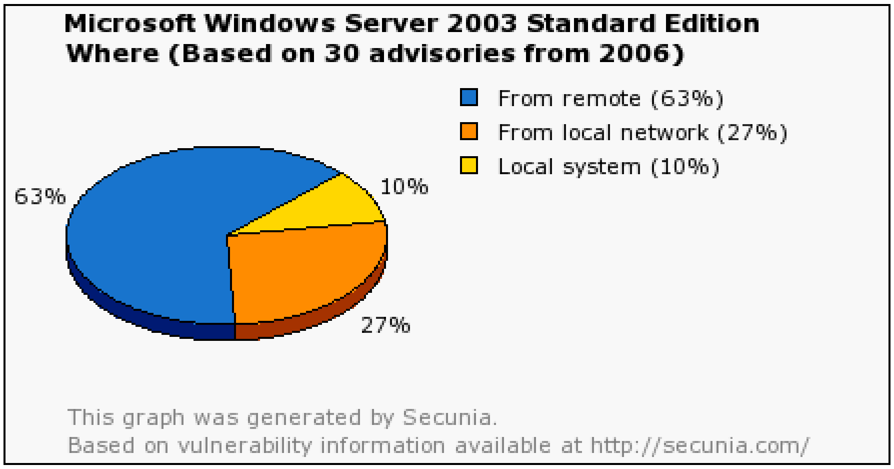


Microsoft Windows Server 2003 Standard Edition
Criticality (Based on 30 advisories from 2006)

This graph was generated by Secunia.
Based on vulnerability information available at http://secunia.com/

Figure 4. Criticality report of the vulnerability

Figure 5 shows that on vulnerability-based attacks, 90% were launched from network-based systems. Of these attacks, 63% were from remote systems and 27% from local network systems. The remaining 10% came from local systems.



Microsoft Windows Server 2003 Standard Edition
Where (Based on 30 advisories from 2006)

This graph was generated by Secunia.
Based on vulnerability information available at http://secunia.com/

Figure 5. Location of attack

Figure 6 shows that among the vulnerability-based attacks, 58% were system-accessed in nature while the other attacks such as denial-of-service (DoS) and privilege-escalation took up 18% and 13%, respectively.

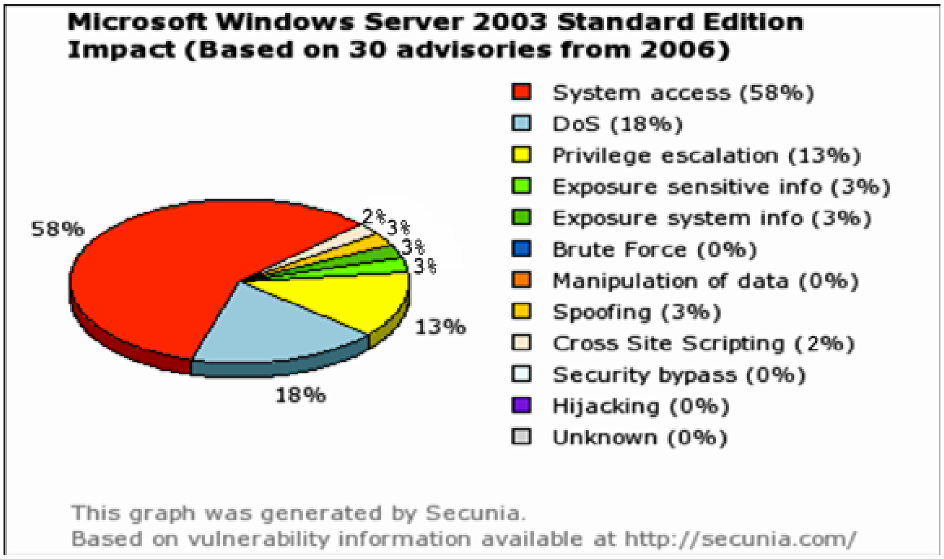


Figure 6. Types of attacks

1. Common Vulnerabilities

Among the vulnerabilities reported on Windows 2003 server standard edition that surfaced from 2005 to 2006 by SecurityFocus [4], it was discovered that most of the reported vulnerabilities come from bugs or design errors in the software code. A majority of the vulnerabilities reported fell into three primary areas: (1) boundary condition error, (2) failure to handle exceptional conditions and (3) input validation error. These vulnerabilities were found in (1) Internet Explorer (IE) 6 (the web browser that comes bundled with Windows 2003 server), (2) window libraries in the likes of ".dll" files and (3) windows explorer. These vulnerabilities could lead to many potential remote attacks. For instance, malicious media files, web pages, or images could be used to crash the system and prevent it from delivering service. Hence, to prevent the system from succumbing to these types of vulnerability exploits, it is necessary to determine the presence of these vulnerabilities and fix them.

2. How to Determine if the System Is at Risk

The common practice to determine if the system is at risk is to use a vulnerability scanner such as Nessus or Microsoft Baseline Security Analyzer (MBSA) .These tools allow the users to determine the system vulnerabilities' presence so that actions can be taken to resolve or mitigate them. Nessus is a security tool that scans all the ports used by the services running on the system(s) and also tries to exploit these ports. MBSA is a vulnerability tool offered by Microsoft that checks the configuration of the operating system for missing patches, user accounts without passwords and available updates. It will highlight any vulnerabilities that do not conform to its security checklist. Both scanner programs provide users with a report that highlights vulnerabilities detected.

3. How to Protect against Vulnerabilities

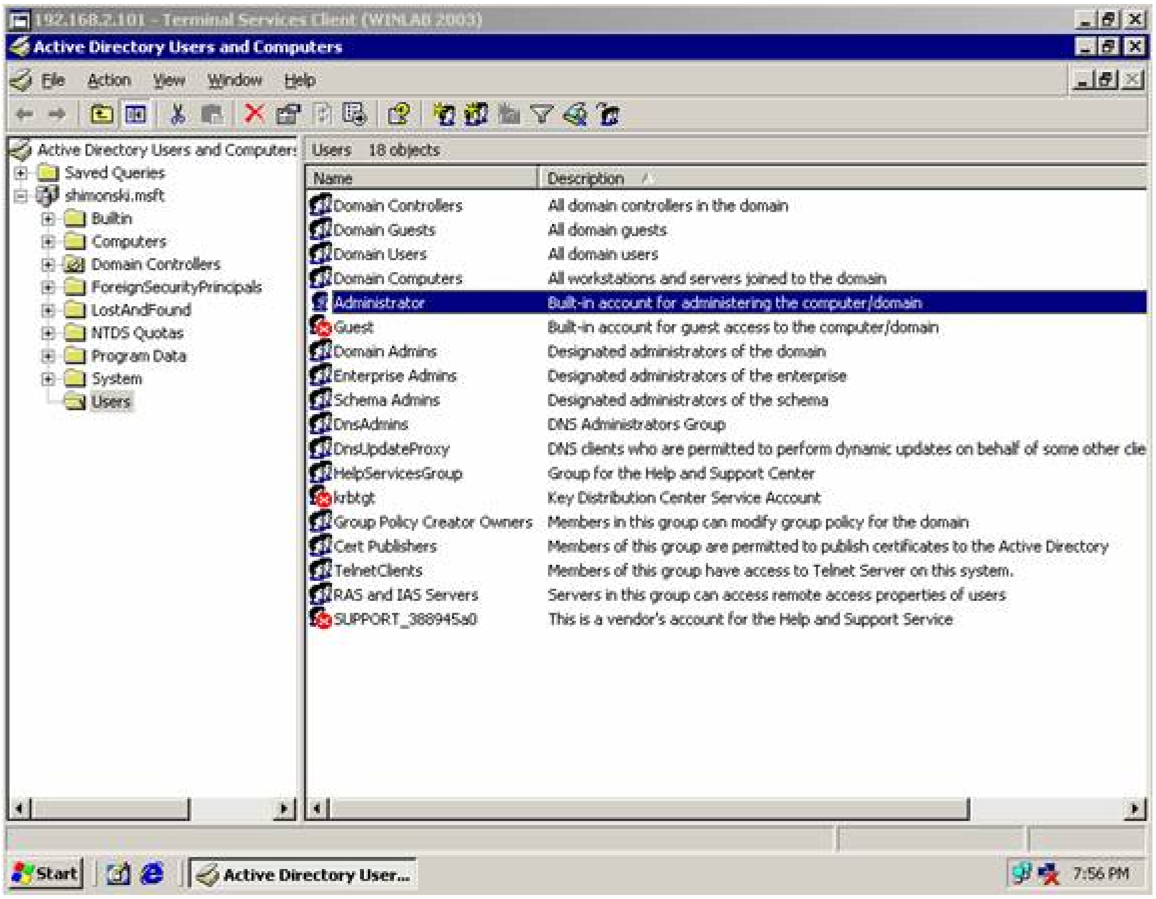
Once the vulnerabilities have been determined, necessary steps must be taken to reduce them and protect the system. Some of the best practices adopted are as follows.

a. Windows Updates and Patches

Ensure that patches and service packs (hot-fixes) are always up-to-date to prevent any exploitation of vulnerabilities. Windows 2003 server provides the user with options to update the patches and service packs automatically. However, as the patches might introduce additional bugs and vulnerabilities, it is advisable to update patches only when a common consensus is reached among all security advisories.

b. Disable/Rename Default Accounts

As part of the process of securing the server, all un-used default accounts must be disabled or renamed. In Windows 2003 server, there are numerous built-in accounts that can be used by default as shown in Figure 7. This is a common thread for attack.



Users 18 objects

Name	Description
Domain Controllers	All domain controllers in the domain
Domain Guests	All domain guests
Domain Users	All domain users
Domain Computers	All workstations and servers joined to the domain
Administrator	Built-in account for administering the computer/domain
Guest	Built-in account for guest access to the computer/domain
Domain Admins	Designated administrators of the domain
Enterprise Admins	Designated administrators of the enterprise
Schema Admins	Designated administrators of the schema
DnsAdmins	DNS Administrators Group
DnsUpdateProxy	DNS clients who are permitted to perform dynamic updates on behalf of some other clie
HelpServicesGroup	Group for the Help and Support Center
krbtgt	Key Distribution Center Service Account
Group Policy Creator Owners	Members in this group can modify group policy for the domain
Cert Publishers	Members of this group are permitted to publish certificates to the Active Directory
TelnetClients	Members of this group have access to Telnet Server on this system.
RAS and IAS Servers	Servers in this group can access remote access properties of users
SUPPORT_388945a0	This is a vendor's account for the Help and Support Service

Figure 7. Default accounts

To prevent this type of attack, it is mandatory to always change the name or disable default accounts. Accounts in Windows 2003 server that are highly common (and built-in) are the Guest and Administrator accounts. It is recommended to rename the default Administrator account to enhance system security. It is also advisable to create a backup Administrator account to protect primary Administrator account log-out. [5]

c. Install Anti-Virus Software

Currently, most viruses are created to attack Windows systems. To prevent such attacks, anti-virus software must be installed and signatures must be updated regularly. In addition, the system must be scanned regularly to ensure that it is free of viruses. Norton and MacAfee are two popular choices for anti-virus software.

d. Other Protection Measures

Other security measures recommended to protect the system include the following:

- ***Implement NTFS*** (New Technology File System) – NTFS is a windows file system developed for Windows NT that provides a good method to implement security on the file system. It provides the capability to impose an access-control list (ACL) to files and folders, thereby permitting or disallowing peoples' access. In the newest NTFS, i.e., NTFS5, it provides additional capability to encrypt files and folders. This will keep files safe from intruders who might try to gain unauthorized physical access to sensitive and stored data.
- ***Disk Segmentation*** – it is always advisable to separate specific data, e.g., sensitive and non-sensitive data, on separate physical disks. This prevents attackers from traversing the web site directory trees inside the web server. This in turn provides another security layer to deter attackers from gaining access to the cmd.exe utility remotely.
- ***Install Host-based Intrusion protection System (HIPS) /Host-based Intrusion detection system (HIDS)*** – this agent, which is installed on the system, monitors activities in the system against a set of specific patterns or signatures. If an anomaly activity is detected, it will either block the activity by the HIPS or inform the user through HIDS.

C. IIS6.0

IIS6.0, which comes bundled with Windows 2003 server, was designed to be a secure product. It was re-architected and built on a Windows Server 2003 platform exclusively. Most of the services and features were turned off by default to prevent any possible attacks. In the new architecture, all web applications are serviced individually by a dedicated worker process in an application pool. Therefore, if one process fails, it will not affect other processes. In addition, the anonymous account, which comes with IIS6.0 by default no longer, has write permissions on the default web root folder. This prevents any attacker from taking advantage of this vulnerability. [6] However, like Windows 2003 server, it lends itself to vulnerabilities because of bugs and design errors in the software code.

1. Reported Vulnerabilities

Among the vulnerabilities reported by SecurityFocus [7], the most noted vulnerabilities are those pertaining to (1) remote buffer overflow, (2) web admin console, (2) disclosure of IP and (4) internal address. All of these vulnerabilities stem from the application's inability to validate input or design errors in the software code.

2. How to Determine the Vulnerabilities

The vulnerabilities of IIS6.0 can be determined by the use of a vulnerability scanner such as Nessus or MBSA. All services and ports identified by Nessus can be checked against IIS6.0 to determine if they are needed. Any un-used ports and services should be turned off. MBSA would determine the latest patches and fixes that should be patched to prevent IIS6.0 from possible vulnerabilities.

3. How to Protect against Vulnerabilities

a. Windows Updates and Patches

Ensure patches and service packs (hot-fixes) are always up to date to prevent any exploitation of vulnerabilities. However, as patches might introduce bugs, it is advisable to update patches only when a common consensus is reached among all the security advisories.

b. Auditing and Logging

Logging must always be enabled to keep track of all accesses and requests made at the web server. The auditing provides valuable information about possible attacks on the web server, which is useful during incident handling. To prevent log files from being accessed by un-authorized personnel, it should be secured with NTFS permissions. Log files should also be audited on a routine basis to detect any anomaly activity.

c. Access-Control

Access-control should always be in place to deter or defend against attackers. It can be applied to (1) control applications running on the server, (2) restrict files served by the server and (3) limit access to the web site hosted by the server. To ensure access-control security, the following recommendations are made [8]:

- Applications - only enable those web service extensions that have a corresponding application or web service hosted on the web server. By default, IIS6.0 enables ASP, ASP.NET, Front Page Server extensions, Internet Data Connector (IDC), Server Side Include (SSI). It is recommended to disable all unknown ISAPI extensions or all unknown CGI extensions**.** This will reduce the probability of vulnerability attacks associated with these applications.
- Files – only enable those files that are served by the server. It is recommended never to enable wildcard "*" as this will permit IIS6.0 to serve any file type (including .ini, .bat) to the client. This will increase the probability of handling any malicious files.
- Website permissions – website permissions should always be exercised with caution and be granted on an as-needed basis. By default, all users from the Internet are granted read permission that allows them to view the contents hosted by the server. Write and executable permissions are disabled by default. This prevents any un-authorized person from the Internet to upload malicious files to the server or execute malicious scripts on the server. Hence, it is recommended to grant write and executable permissions on an as-needed basis. It is also recommended to put all executable and script files in one directory so that it is easy to configure access permissions and audit special access permissions.
- User accounts – IIS6.0 provides four built-in accounts: (1) LocalSystem account that is part of the administrators group which has administrator rights, (2) Network Service account that has fewer permissions on the system, (3) Local Service account that has the same privileges as the Network Service account on the local machine but cannot access resources across the network and (4) IIS_WPG group account that is assigned minimum permissions. This account is also used to start any worker process. It is recommended to assign users to either the Network Service account or the Local Service account depending on the services and resources they required. No users should be assigned to LocalSystem account because this account has a high level of access rights and has been a target for many exploits.

D. PHP SCRIPTING LANGUAGE

PHP is the most widely used scripting language for the web. Based on some reports, 50% of the Apache servers world-wide have PHP installed. A large number of Content Management Systems (CMS), portals and bulletin boards (BB) are written in PHP. Despite its popularity and widespread-usage, PHP is plagued with vulnerabilities. It was reported by SANS that in 2005, a problem was reported at least every single week in some software using PHP. [9]

1. Common Vulnerabilities

Some of the common vulnerabilities reported include the following.

a. PHP Package Vulnerability

This vulnerability is found in the PHP package itself and is caused by flaws in the way the PHP package software handles POST requests. It was reported that each of these flaws could allow an attacker to execute arbitrary code on the remote system.

b. Remote File Vulnerability

This vulnerability is found inside the application using PHP and is due to the application’s failure to handle maliciously crafted file properly. This allows an attacker to run malicious code on the vulnerable web server.

c. Remote Command Execution Vulnerability

This vulnerability is found in applications using PHP and is due to the application’s failure to process data that contains commands properly. This allows attackers to execute commands remotely that may compromise the application and underlying system.

d. SQL Injection Vulnerability

This vulnerability is found in applications using PHP and is due to the application’s failure to sanitize variables used to construct SQL queries. This permits attackers to inject SQL code into the variables. This attack is commonly used to recover password hashes for administrators of the PHP applications.

e. Remote Code Execution in Library Vulnerability

This vulnerability is found in libraries using PHP and is due to a failure in the library to properly conduct a sanity check on the input. This permits attackers to execute code remotely in libraries. The “Lupper worm” is a worm that was created to exploit remote execution vulnerabilities in these libraries.

2. How to Determine the Vulnerabilities

The vulnerabilities of PHP can be determined by scanning the web servers periodically with a vulnerability scanner. Common PHP vulnerability scanners are (1) Acunetix web vulnerability scanner [10], (2) remote PHP vulnerability scanner [11] and (3) File inclusion scanner [12].

3. How to Protect against Vulnerabilities

The best practices to protect against PHP vulnerabilities are listed as follows.

a. Update Patches

Always apply all vendor patches for PHP and PHP-based applications.

b. Web Scanning

Always run a vulnerability scanner on the web site frequently or periodically.

c. Secure PHP Configuration

Always secure PHP configuration by disabling the parameters “register_globals”, “allow_url_fopen” and “magic_gpc_quotes” and enabling the parameters “safe_mode” and “open_basedir”.

d. Run SQL Injection Test

Always conduct automated SQL injection tests against PHP applications hosted by the web server using web application security assessment tools like Paros Proxy.

e. Adopt PoLP (Principle of Least Privilege)

Always adopt POLP for running PHP using tools such as PHPsuExec, and php_suexec. These are modules/software that allow users to impose ACL on the PHP files. This allows only limited/authorized persons to access and operate PHP files.

f. Upgrade

Always upgrade to the latest version of PHP as it will eliminate many existing PHP security issues. At the time of this writing, the latest version is PHP 5.

g. External Protection Mechanisms

Always use an intrusion prevention/detection system that will block/alert any malicious HTTP request(s).

E. SQL

Databases are a key element for data storage, searching or manipulation in many systems. They can be found virtually everywhere, from businesses, financial and banking institutions, to schools and anywhere that requires manipulation of large data. Due to the valuable information they store such as passwords and financial details, they are often a target of attacks. [13] To make matters worse, these attacks could be easily conducted by anybody who has a query tool because most modern relational databases are port addressable. The attackers could connect directly to the database using the query tool and bypass security mechanisms used by the operating system.

1. Common Vulnerabilities

Among vulnerabilities reported in databases, the most common vulnerabilities can be classified into the following categories:

- Buffer overflows in processes that listen to well-known TCP/UDP ports
- SQL injection via web entry
- Databases running in default configuration with default usernames and passwords
- Databases running with weak passwords for privileged accounts

2. How to Determine Vulnerabilities

The vulnerabilities of the database could be determined by using a vulnerability scanner or tools offered from database vendors such as MySQL network scanner and Microsoft SQL server tool.

3. How to Protect against Vulnerabilities

Some of the recommended practices to protect against database vulnerabilities are listed as follows.

a. Update Patches

Always ensure all DBMS have up-to-date patches because any un-patched or outdated versions are likely to contain vulnerabilities.

Always check vendor sites for patch information and subscribe to vulnerabilities alerts offered by the vendor. For MySQL, the web site is: http://lists.mysql.com/.

b. Secure DBMS and Applications

Always secure DBMS and its applications. This includes:

- Adopt least privilege principle for DBMS access
- Remove/change default passwords on the database’s privilege and systems accounts
- Use store procedures where possible
- Remove/disable unnecessary procedures
- Set length limits on form fields

c. Other Security Mechanisms

Always use firewalls or other security devices to control network access to ports associated with database services.

d. Data Validation

Always ensure that the application performs data validation and sanitation at the server end to prevent attacks such as SQL injection or buffer overflow.

F. CHAPTER SUMMARY

This chapter observed that COTS products, typically software components, are plagued with vulnerabilities that stem from bugs or design errors in the software code. These vulnerabilities are exploited by attackers that often lead to (1) system crashes, (2) theft of information or (3) corruption of data. These attacks are disastrous to both the system and the people involved, including IMAS. Although patches and mitigation methods are always available to reduce and prevent against such vulnerabilities, they are reactive in nature. They are always available after the vulnerability has been reported. This opens up a vulnerability window where systems are susceptible to attacks. In addition, since most, if not all software codes contain bugs, users are expected to patch or apply mitigation methods whenever a vulnerability is reported.

The next chapter reviews the framework of IMAS. It will analyze its design and processes to uncover and identify possible vulnerabilities. Furthermore, it will discuss how these vulnerabilities are exploited by well-known attacks. It will also provide solutions or mitigation techniques to help reduce and prevent these vulnerabilities.

THIS PAGE INTENTIONALLY LEFT BLANK

III. IMAS SECURITY ANALYSIS AND RECOMMENDATIONS

A. INTRODUCTION

This chapter studies the framework of IMAS and reviews two areas: (1) its design architecture and (2) how in-house developed applications handle input, to determine probable vulnerabilities. It will step through the flow of IMAS processes to determine probable vulnerabilities in each of them. The study will focus specifically on identifying vulnerabilities of in-house developed applications that are frequently exploited by common attacks and understand why they succeed. Furthermore, it will cover wrong-input-entry vulnerabilities. It will then propose various countermeasures to show how these vulnerabilities can be mitigated. In addition, the study will suggest security methods to protect the data from unauthorized access and modification. Other security measures are recommended to make IMAS more secure.

B. IMAS FRAMEWORK OVERVIEW

The IMAS system was constructed based on a framework shown in Figure 8. In this figure, an IMAS user enters the web portal system through a *profile interface.* However, before the user is able to use the system, he/she must create an account in the system. Here, the user enters the following information:

- first name
- last name
- desired user name
- password
- e-mail address
- e-mail password
- IM screen-name
- IM provider
- cell-phone number
- cell-phone provider
- cell-phone model

This information, less the cell phone model, is stored in the *user profile*. The cell phone model is stored in the *device profile*. After registering, the user logs into IMAS to enter his/her schedule. Here, they will be presented with a calendar menu from which they will select the date of the event. They will then enter the (1) description, (2) time, and duration, (3) context and (4) types of alert they want to receive. This information is stored in the *user profile*. IMAS currently offers three context types: (1) general, (2) meeting and (3) do not disturb. In the general context, the user will receive one of the following: (1) SMS message, (2) IM, (3) e-mail, or (4) cell-phone call. In the meeting context, a SMS message, IM, or e-mail will be sent. In the do not disturb context, an e-mail is the only alert disseminated. According to the user-set parameters in the *user profile,* when an alert comes in through the *media storage*, IMAS will perform an alert-transformation in the *alert construction*. Here, it transforms the incoming alert to an alert specified by the user. After the alert is made, the system sends it to the user thorough the *alert retrieval and dissemination* stage. [14]

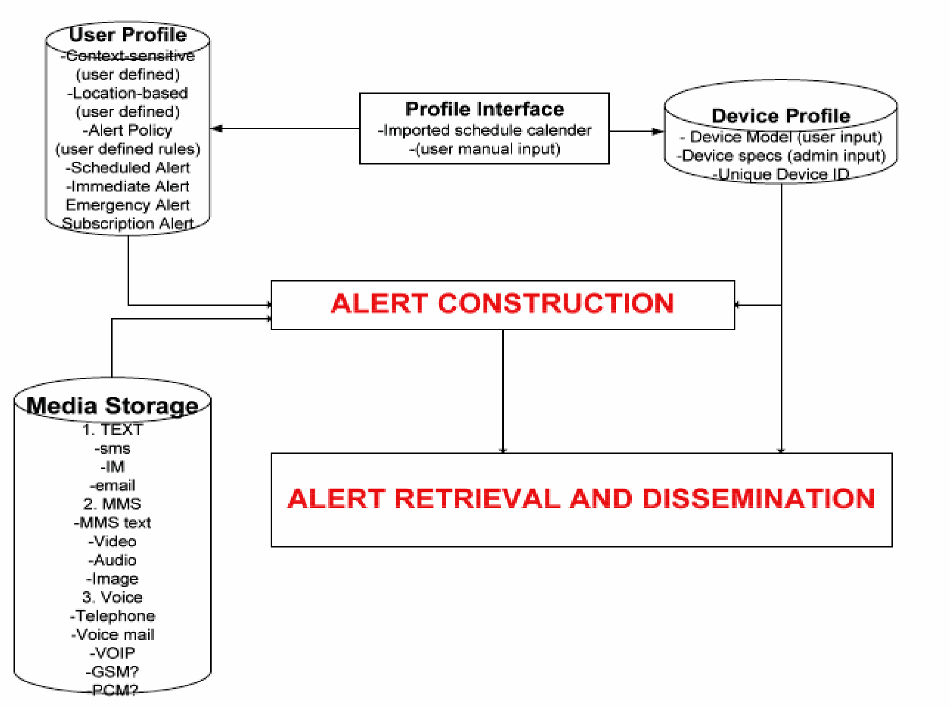


Figure 8. IMAS Framework

C. DESIGN AND IMPLEMENTATION

The IMAS was built on a single computer (a Dell Optiplex GX270 3.2GHz, 1GB RAM and 30GB HDD) running all its services/applications and data. Although this design and implementation is a cheap and a quick means to build a system, it is considered a bad implementation because it is a single point of failure. If the box is taken down, all the services will be lost, i.e., loss of availability.

1. Vulnerabilities

The single-box design implementation suffers vulnerabilities in the following areas.

a. Security

Since the data is housed in a single box with the services/applications, all the data could be compromised if the web server is exploited. An attacker could gain access to all the data, even though they are housed in different directories. This is known as the directory traversal attack. This box could also be used as a launching pad against other corporate computer systems if the computer is on a corporate local area network. In addition, the attackers could use services not required by IMAS for an exploit. This implementation denies the flexibility of deploying other security options. It weakens the security of IMAS due to the difficulty of applying appropriate security options on different applications and data residing inside the same box.

b. Management Control

By incorporating all the services and applications within a single box, the management control of people’s access will be complicated. People from various departments managing different services or applications are required to access the system to maintain and support numerous operations. If the management control of personnel access is not handled properly, system degradation increases.

c. Scalability

A single-box implementation does not scale well because upgrades to applications or the physical box will bring the system down. In addition, it will be taxing on computing resources if traffic is heavy. This would lead to a potential loss of service when traffic is beyond the threshold of the system.

2. Tier-Architecture

The common practice to overcome issues discussed previously is to separate applications into tiers and implement them into a 3-tier architecture. [15, 16] Figure 9 shows an overview of the 3-tier architecture. This architectural tier consists of the: (1) web, (2) application and (3) database.

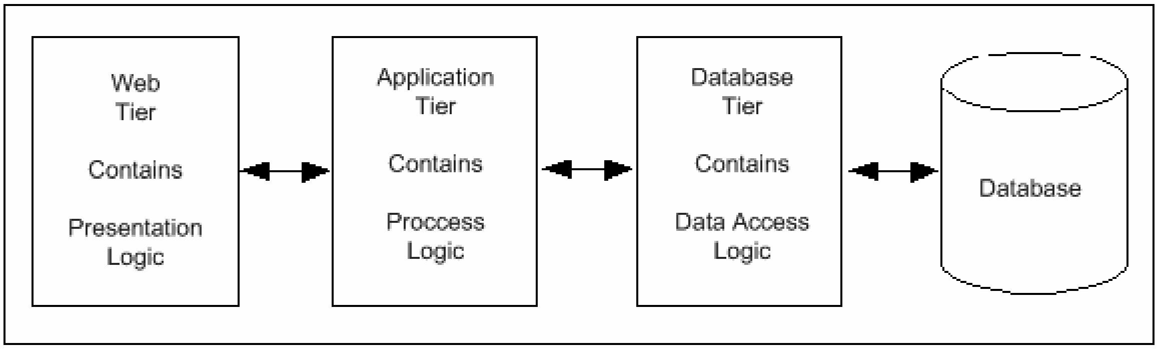


Figure 9. 3-tier architecture

The web tier provides static content, e.g., simple HTML pages to users with web applications. The application tier provides dynamic content, e.g., applications and processing logic, while the database tier provides access to the database for data storage and retrieval.

a. Physical Implementation

The 3-tier architecture should be implemented using either a 2-box or 3-box approach. In the 2-box implementation, the web and application tier are co-located and run within a single box while the database tier is in a separate box. In the 3-box implementation, each tier is run on a single box. Figure 10 and Figure 11 show the 2-box and 3-box physical implementation, respectively.

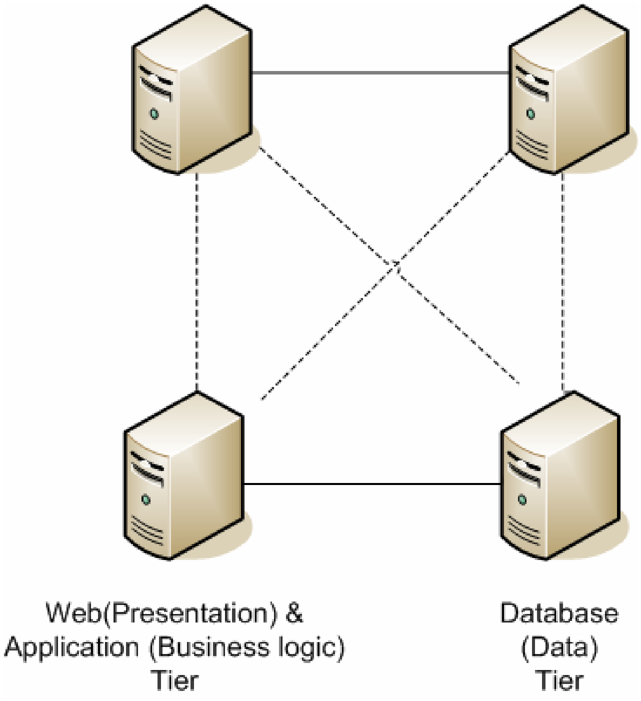


Figure 10. 2-box implementation

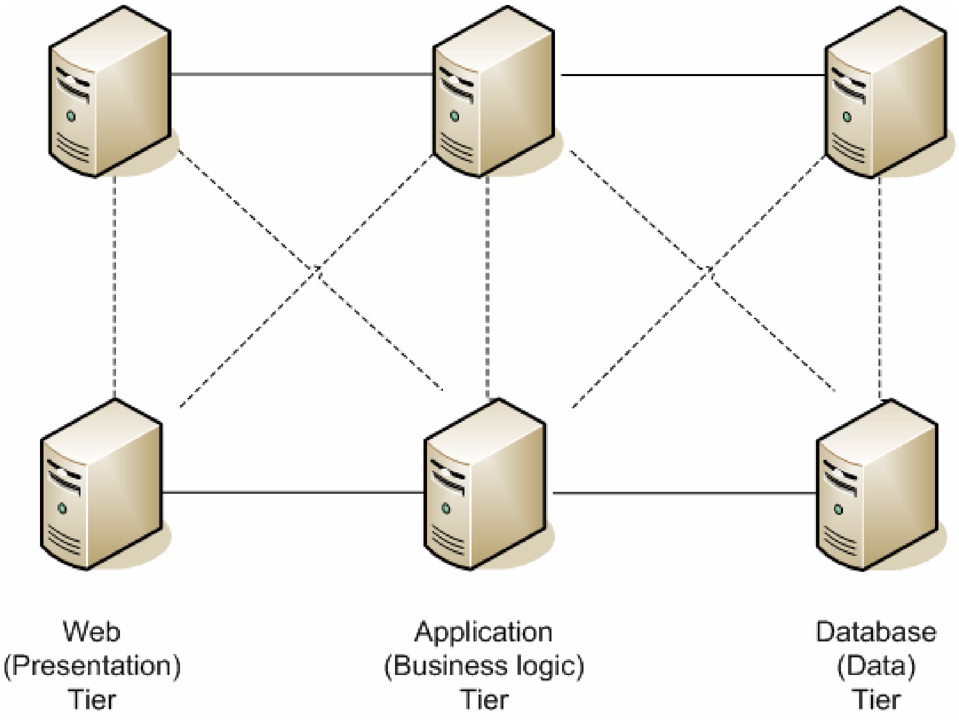


Figure 11. 3-box implementation

b. Benefits of 3-Tier Architecture

The 3-tier architecture is recommended over the single-box architecture due to numerous benefits it provides. The benefits are discussed as follows.

- Security – The 3-tier architecture offers several security benefits. They are listed as follows.
    - Data is better protected - With the separation of applications and data into tiers, logically and physically, data does not reside in the same box as applications. This provides another layer of defense against data access. Hence, if the web server is compromised, attackers will not be able to access data directly. This prevents information leakage.
    - Impact is minimized – With separation, only the box that is compromised is affected. It will not affect other systems on the network, i.e., impact is contained within a single box. Hence, the impact is minimized.
    - Deployment of appropriate security options – With separation, it allows the deployment of appropriate security options on different boxes and applications. For example, ACL could be applied to the database to control the access. The application proxy/FW or host based IPS/IDS/FW could also be installed on each box to filter out any unwanted/suspicious traffic.
    - Control of unwanted services – With separation, services/applications running on each box are better controlled. Unwanted services are turned off to prevent attackers from exploiting them.
- Higher Availability – The 3-tier architecture offers a higher availability. Each tier can deploy multiple boxes offering the same service simultaneously. Therefore, if a single box goes down, service will not be disrupted. Instead, the service can be provided by the other boxes in the same tier.
- Stronger Management Control – The 3-tier architecture provides stronger management control in two aspects: (1) applications running in the system and (2) personnel access. With applications de-lineated and separated into tiers, the system can control the types of applications running in each box. Applications not belonging to the tier will not be allowed to run in the box. This in turn, limits the access of personnel supporting the applications. Personnel supporting one tier will not be allowed to access the box belonging to the other two tiers.

- Better Scalability – The 3-tier architecture offers better scalability because it allows upgrades (hardware and software) without loss of service. Since this architecture offers HA, hardware and software upgrades could be implemented incrementally i.e., one box at a time without disrupting the service.

3. Recommended Design and Implementation

The 2-box approach is recommended for IMAS. This approach is cheaper and consumes less bandwidth (as there are fewer tiers). Figure 12 shows the proposed 3-tier architecture based on a 2-box approach.

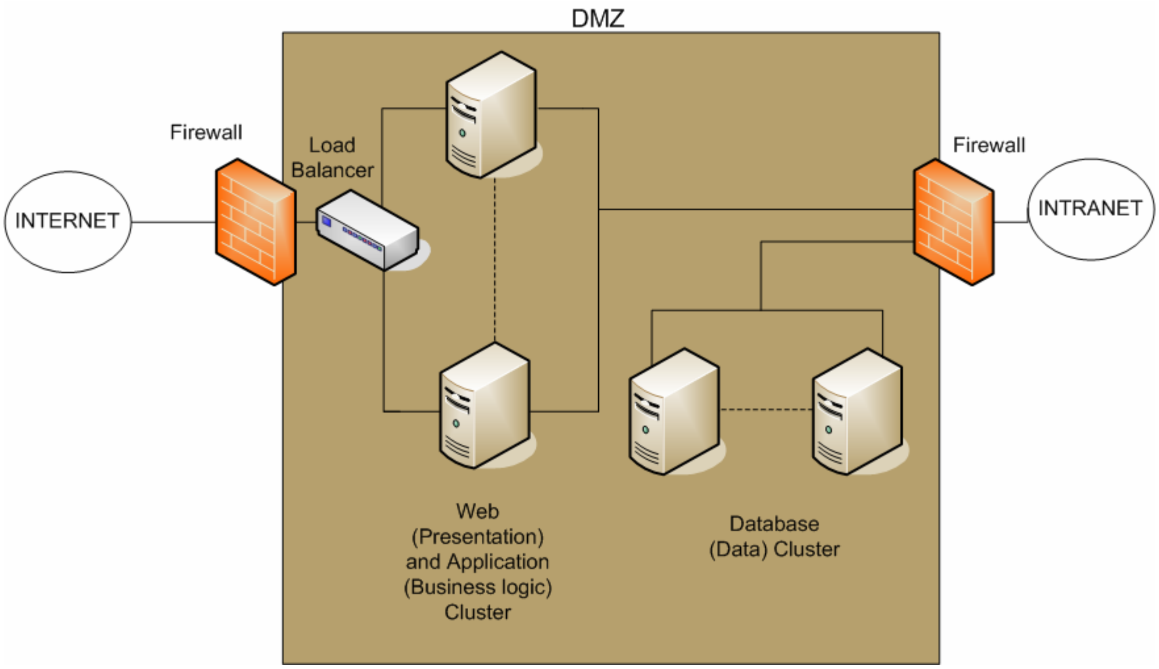


Figure 12. Proposed 3-tier architecture

The servers are deployed inside a demilitarized zone (DMZ) for enhanced security. DMZ is an isolated network that sits between the Internet and the Intranet. It is protected by two firewalls. The firewall facing the Internet controls access between the Internet and DMZ. The firewall facing the Intranet limits access from the Intranet to DMZ. It does not allow hosts from DMZ to access the Intranet. This prevents attackers from using IMAS as a launching pad to attack the Intranet if it is compromised. The database server is placed in a different segment in the DMZ, separated from the web and application server. This provides an additional layer of defense, which deters attackers from accessing data easily.

D. PROCESSES

This section will step through the IMAS processes and identify probable vulnerabilities.

1. Registration

Registration is the first process a user interacts with IMAS before he/she can use the system. In this process, the user registers an account and inputs his/her personal and device profile information. However, prior to registration, the user must verify that they are gaining access to IMAS and not a bogus server hosted by fraudsters.

a. Vulnerability - Phishing

“Phishing is a criminal activity using social engineering techniques. In this act, phishers attempt to fraudulently acquire sensitive information, such as passwords and credit card details, by masquerading as a trustworthy person or business in an electronic communication.” [17] In the context of this thesis, phishing refers to users being led or directed to a masqueraded web site that looks identical to the lMAS. However, it is hosted by attackers without the knowledge of end users or IMAS’s owner. When users are on this masqueraded web site, they log in and unknowingly release their personal information.

b. How to Detect Phishing

There are various ways to determine phishing. The most common and effective way is to use anti-phishing software. This software validates if the website is a phishing website by checking against a list of reported or blacklisted phishing websites. The following is a list of anti-phishing software recommended: [18]

- Earthlink Tool Scamblocker – This software offers a check against a list of blacklisted phishing website. It checks the owner and location of the web site. In addition, it prevents against phishing and pop-ups.

- Corestreet’s SpoofStick – This software validates the web site by using the browser's internal read-only variables to display the top and second level domain name that the user is browsing.
- TrustWatch Toolbar – This software checks in real-time if the web site has been verified by a trusted third party.

c. How to Prevent Phishing

Based on studies conducted on anti-phishing [19, 20], it is recommended that IMAS implement the following measures:

- Distinguish the IMAS site from the malicious site - Always use a single domain name that matches the system, e.g., www.imas.com rather than using an IP address or multiple domain names for the server. This enables users to better distinguish the web site from malicious web sites. IMAS should not be sharing the web server with other web applications to prevent confusion.
- Public Certificate – IMAS should implement public key infrastructure (PKI) technology. PKI is an arrangement that provides a trusted third party who completes a validation check and vouches for user identities. IMAS should generate a public/private key pair and register it with a trusted certificate authority (CA), e.g., Verizon, RSA Security Inc., to obtain its digital certificate. CA is a trusted third party entity that vets and certifies the identities of the users or systems. After certifying, it binds their public keys and their identities into a digital certificate signed by the CA. Users may then verify the web server using their own web browser. If the server uses a certificate that is not signed by a trusted CA, the browser will issue a warning alerting the user.
- Encrypt every page – All the pages in IMAS should be encrypted using secure socket layer (SSL) /transport layer security (TLS). This will allow the web browser used by the user to authenticate that the page comes from IMAS. SSL/TLS are cryptographic protocols which provide secure communications on the Internet for things such as web browsing, e-mail, Internet faxing, and other data transfers.

2. Establish Secure Communication Channel

All registrations should be conducted in a secure mode, i.e., all data input by users should be encrypted. This is because data submitted by users is confidential and sensitive.

a. Vulnerability – Internet Snooping

By default, communications over the web are conducted in clear text using http protocol. If an attacker is eavesdropping or snooping on the

Internet, the attacker will know what is being communicated. The attacker may then use this information to impersonate the user and gain access to the user’s bank account or credit card information. This is disastrous and undesirable.

Likewise, if the registration for IMAS is conducted in clear text using http protocol, it is likely that data will be eavesdropped or intercepted by attackers snooping on the Internet. This results in a loss of confidentiality. Thus, to prevent this form of vulnerability, it is necessary to establish a secure communication channel/session between users and IMAS before and during the registration process. This ensures that all data transmitted are secure. This secure communication channel must always be established whenever any confidential information needs to be communicated between users and IMAS.

b. How to Establish Secure Communication Channel/Session

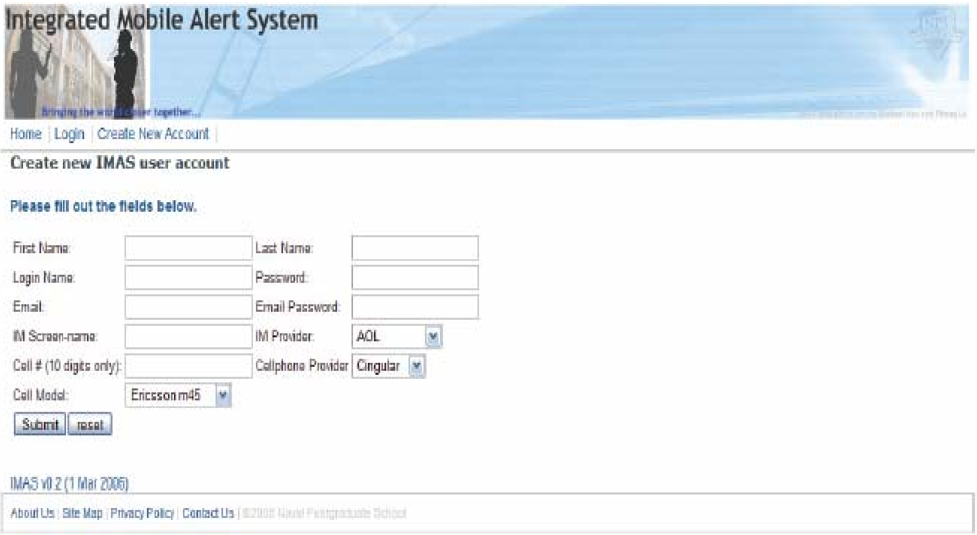
IMAS should use SSL/TLS to establish a secure session between the web server and the user. SSL/TLS supports two modes of authentication to establish a secure communication session: (1) server and (2) server and client. In the server-authentication mode, only the server is authenticated by the client. Here, the web browser will check the certificate used by the web server. If it is not signed by a trusted CA, the web browser will give a warning informing the user. In the second mode, the server and client are mutually authenticated. Once the authentication is completed, a session key will be generated to encrypt the session between the server and the user. This session key is only valid during this session. If a new session is established, a new session key is used.

c. Deployment in Secure Environment

If IMAS is used in a corporate or secure environment, it is recommended to include SSL/TLS server-client authentication together with the standard username and password authentication implemented by the system. This provides a more restricted access. If the user does not have their CAC that stores the private key, then the system will deny the user from establishing a session to IMAS.

3. Data Process

In the registration process, an IMAS user will enter the following: (1) e-mail address, (2) e-mail password, (3) IM username, (4) IM provider, (5) cell phone number, (6) cellular provider and (7) cell phone model. Figure 13 shows the registration menu of IMAS. Once the user has input the data and clicked on the submit button, the data will then be sent to IMAS for processing. However, before data can be processed by any application(s), it must be validated at the server-end for correct and non-malicious entry.


Integrated Mobile Alert System
Bringing the world closer together...
Home | Login | Create New Account |
Submit
reset
About Us | Site Map | Privacy Policy | Contact Us | ©2006 Naval Postgraduate School

Create new IMAS user account

Please fill out the fields below.

First Name:		Last Name:	
Login Name:		Password:	
Email:		Email Password:	
IM Screen-name:		IM Provider:	AOL
Cell # (10 digits only):		Cellphone Provider	Cingular
Cell Model:	Ericsson m45		

IMAS v0.2 (1 Mar 2006)

Figure 13. IMAS registration menu

a. *Vulnerabilities*

The input submitted by users is likely to contain errors due to typing errors or "fat finger" mistakes. This leads to incorrect information being captured by IMAS. As a result, the system is unable to deliver alerts to the user (causing a loss of service) which is detrimental to IMAS effectiveness. Hence, it is vital to ensure that data is submitted correctly.

In addition, data type validation is imperative. Based on a report by Gartner, most web application attacks in 2004 [21] were a result of web applications' failure to check and validate input entry before any application execution. This creates an opportunity for the attacker to sneak in malicious characters or scripts that exploit the vulnerability of the underlying protocols supporting applications such as PHP, MySQL and the operating system. Hence, it is also necessary to perform data validation at the trusted server-end, before it is being used by the application.

b. How to Correct Input Entry Errors

The way to ensure data is free of input entry errors is to perform data-correction checks. This section proposes several data-correction check methods for each field in the registration menu. These methods leverage existing services/programs/scripts available to perform the check to minimize the programmer's effort. The programmer just needs to write a script to use these services. However, it should be noted that these services are written by un-trusted parties. Therefore, it is necessary that these programs be reviewed by a group of security programmers to ensure they do not contain malicious code. In addition, these programs must be tested rigorously to ensure they do not contain malicious codes. The validation check for each field is listed as follows.

(1) E-mail Address and Password Validation Check. There are two ways to validate an e-mail address and password: (1) script or (2) external web mail retrieval service. IMAS could leverage these two methods as follows:

- Script – The script polls the mail server with an e-mail address and password entry provided by the user. If the e-mail address and password entry are valid entries, the mail server will return a successful reply. If an unsuccessful reply is received, IMAS can provide feedback to the user that the entries submitted are invalid and ask him/her to verify the entries. "Pop3 account polling", is a software written in perl script that works on this principle. [22] It polls an external POP3 account on a regular basis and retrieves any e-mail messages. It then executes an e-mail processing program, e.g., Outlook, and feeds the message text to the program's standard input.
- External web-mail retrieval service - This type of service is available free on the Internet [23]. It allows users to retrieve e-mail from any POP3 accounts from the web using the user's e-mail address and password entries. If the e-mail and/or password is incorrect, it will generate an error message. IMAS would then provide feedback to the user. "Mail2web," [24] is a free service available on the Internet that verifies the e-mail address and password. Furthermore, it allows for both secure and non-secure log-in.

(2) IM Name and IM Provider Validation Check. This check leverages the search feature offered by IM client software. This feature provides direct access to the provider’s database. Before IMAS can use this feature, it must download the IM client software from IM providers e.g., MSN, ICQ, Yahoo Messenger, Skype or Jabber into IMAS. IMAS will have a registered account for each IM provider utilized by the various users. IMAS can then use the search engine of the IM-client software to verify the user’s username. If the username is valid, IMAS will add this username to the contact list of the IM client software in IMAS. If the user’s entry is invalid, the search will return a null result. IMAS can then provide feedback to the user and request a re-verification.

However, a drawback is that the search might also return a list of other users sharing the same username. To refine the search, additional fields such as country and e-mail address are required. In the current IMAS implementation, the menu only allows the user to input the IM screen-name and the IM provider. This limitation prevents the system from verifying the entry effectively. In addition, IMAS is inflexible as it only allows the user to input one IM account. Nowadays, users typically have more than one IM account. Users may need various IM accounts in order to communicate with different groups of people, e.g., business associates, friends and family; or for different environments. Some IM services are blocked in certain environments.

Hence, to resolve these issues, the registration menu needs to be flexible. It must allow users to add/select more than one IM account. Each account should have additional fields to provide a better granularity search. Although IM service provides a convenient and almost real-time alert notification to reach the user, it should be noted that some IM software is insecure. Messages are transmitted in clear text and can be read by anyone with the sniffing tools and network access. In addition, messages are vulnerable during processing on the server. Messages can also be intercepted, altered and re-transmitted. Therefore, it is advisable not to send any confidential message or alerts via this mode.

Fortunately, many popular IM providers (AOL, ICQ, MSN messenger and Yahoo messenger), recognize this drawback and have since launched a secure-version client software. This software encrypts the message between the users to promote a more secure session. For better management of different IM-client-software installed on the system, it is recommended that IMAS uses unified IM software. This software collates all users' contacts from the various IM providers and presents it as a single IM view. Miranda, is an IM software that offers both secure and normal messaging services. [25, 26]

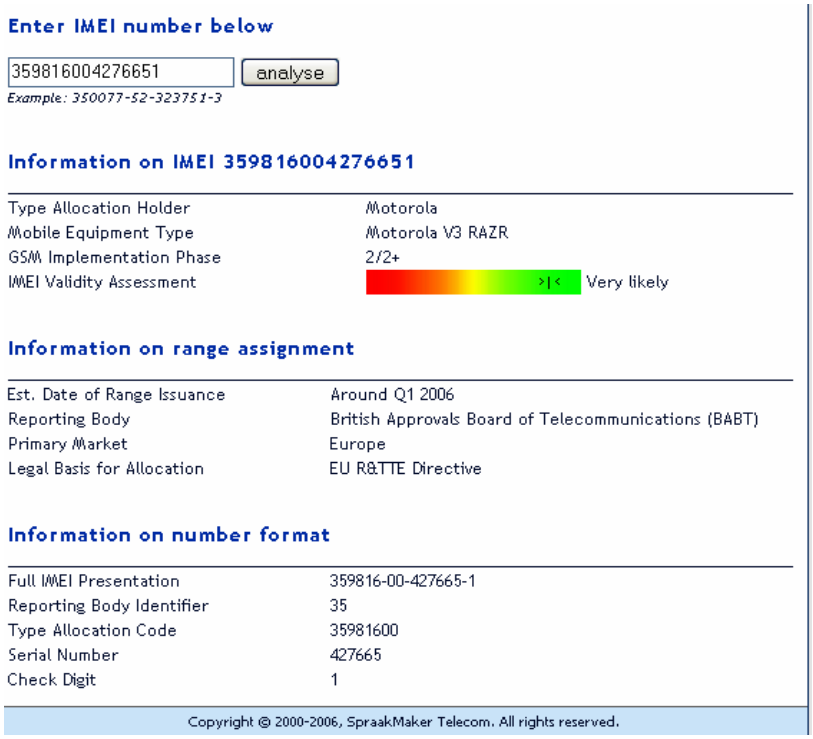
(3) Cell Phone Number and Service Provider Check. There are multiple web sites that provide free services that allow individuals to verify the service provider for a given cell phone number. One such service is the phone number analysis service provided by International Number Plans [27]. It accepts a cell phone number and then returns a service provider. If the service provider returned does not match the entry provided, IMAS will provide feedback to the user.

However, most of these services do not support local number portability (LNP) at this moment because LNP database is not available to the public. LNP is a service that allows users to switch service provider(s) while keeping the original number. In this case, the service will not provide a correct verification. A workaround for a ported cell phone number is for IMAS to send a verification SMS to the user. If he/she receives the SMS, then the service provider is a correct entry. If they do not receive it, they must check their entry again to ensure they input the correct service provider.

(4) Cell Phone Model Validation Check. The model of a cell phone can be validated by checking the unique identification number embedded on the cell phone (by the manufacturer). The International Mobile Equipment Identity (IMEI) embedded on every GSM/UMTS cell phone is a 15 or 17 digit number that provides information on the origin, model, and serial number of the device. It is usually found printed on the phone underneath the battery. Alternatively, it can be found by dialing the sequence *#06# on the cell phone

[28]. The mobile equipment identifier (MEID), found on CDMA cell phones, is 56-bit long (14 hex or 18 decimal digits), containing the 8-bit regional code (RR), a 24-bit manufacturer code, and a 24-bit manufacturer-assigned serial number [29].

(5) To Validate GSM Cell Phones. IMAS could leverage an online service, the "IMEI number analysis" offered by International Numbering Plans. This service analyzes the IMEI and identifies the model and issuing authority [30]. Figure 14 shows the output of this service. For CDMA phones, as MEID does not contain information on the model, it is impossible to identify the model. The only way to identify the model is to query the manufacturer's database which might not be available. Nevertheless, IMAS could use the serial number and verify the manufacturer with the Telecommunication Industry Association (TIA) website [31]. TIA is responsible for managing and issuing MEIDs to manufacturers. If the selected input model does not match the manufacturer, IMAS will provide feedback to the user for re-verification.



Enter IMEI number below

359816004276651

Example: 350077-52-323751-3

Information on IMEI 359816004276651

Type Allocation Holder	Motorola
Mobile Equipment Type	Motorola V3 RAZR
GSM Implementation Phase	2/2+
IMEI Validity Assessment	>|< Very likely

Information on range assignment

Est. Date of Range Issuance	Around Q1 2006
Reporting Body	British Approvals Board of Telecommunications (BABT)
Primary Market	Europe
Legal Basis for Allocation	EU R&TTE Directive

Information on number format

Full IMEI Presentation	359816-00-427665-1
Reporting Body Identifier	35
Type Allocation Code	35981600
Serial Number	427665
Check Digit	1

Copyright © 2000-2006, SpraakMaker Telecom. All rights reserved.

Figure 14. IMEI analysis output

c. How to Correct Data-Type Errors

The data-type errors introduced by attackers are often attacks that exploit applications' failure to validate input entry. These errors could lead to system crashes, theft of information or corruption of data. This vulnerability can be corrected or mitigated by data-type validation methods. These methods rely on three data-type validation models; (1) Accept only known valid data, (2) Reject known bad data, and (3) Sanitize bad data. These three data-type models check on (1) data type, (2) syntax and (3) length of data entry [32]. They are listed as follows:

- ***Accept only known valid data model*** – This is the preferred model to validate data whereby applications should only accept input that is known to be safe and expected. For example, a valid username (defined as ASCII A-Z and 0-9) should be type-checked as a string and must be comprised of A-Z and 0-9. It must also be bounded by the valid length of the application.
- **Reject known bad data model** – This model relies on the application knowing about specific malicious payloads, e.g., specimens or signatures. Although this strategy can limit exposure, it is difficult for any application to maintain an updated database of web applications against signatures because signatures change very frequently.
- ***Sanitize bad data model*** – This model attempts to make bad data harmless, i.e., converting bad data into a common character set. It is effective against bad data input but is extremely difficult to implement. Hence, it should only be used as a second line of defense.
- This part of the section looks at several well-known web attacks that leverage applications' failure to validate input for exploitation. It will show how these attacks are prevented by validation and other techniques.
- ***Cross Site Scripting (XSS)*** – This type of attack lies in the vulnerability of the application's script. Often, the victim is tricked into making a specific and carefully crafted HTTP request using traps. These traps may masquerade in the form of a (1) link in a HTML aware e-mail, (2) web based bulletin board or (3) link or image embedded in a malicious web site. The traps will then direct the malicious request to the web server. Here, the script will read just part of the HTTP request and echo it back in full or in part, without first sanitizing it. It does not ensure that the entry is free from any java script code or HTML tags.

Hence, if the request contains a java script, it will be echoed back to the user as a response page. The user's browser will interpret it as a java script used by the application to verify its inputs. It will then allow the script to access the cookies on the client end. This exposure of all cookies' allows for information to be collected by the attacker. As most identification/authentication/authorization information is maintained inside the web applications cookies, this will allow the attacker to impersonate the users and access their accounts, which is disastrous [33].

d. Mitigation Techniques

There are various mitigation techniques available to prevent XSS. Common techniques are as follows.

- Perform input filtering/data sanitation that looks into the user's inputs such as parameters and HTTP headers and filter against HTML tags including java script. The input includes all possible sources, e.g., query parameters, body parameters of POST request and HTTP headers.
- Perform correct encoding of dynamic output data. This can prevent malicious scripts from being passed to the user. The application could make an explicit decision to encode un-trusted data and leave trusted data untouched, thus preserving the integrity of the mark-up content.
- Enforce fixed character set on web pages and ensure that the data inserted by the system is free from byte sequences containing special characters such as "<". Web pages must be configured to inform the browser of what character set to use when submitting the form data back to the server. The web pages must also be configured to inform the servers' applications as to what character set they need to use internally. This deters any unspecified character-encoding entry that might be misinterpreted by the web server resulting in a probable attack.
- Install a third party application firewall, which intercepts XSS attacks and blocks them before they reach the web server and script. The application firewall validates and inspects the data against various HTML tag patterns and rejects the request if there are any matches. Thus, it ensures that the malicious input does not reach the server and script.
- Run an automated web application vulnerability tool to inspect the web site and launch all variants of attacks it recognizes against all scripts. The attacks include trying different parameters, headers

and paths. Each input to the application e.g., parameters of all scripts, HTTP headers, and paths, is checked with as many variations as possible. If the response page contains a java script code that the browser can execute, then an XSS vulnerability is exposed.

e. Direct SQL Injection

This type of attack occurs when the application does not validate user entry. This allows attackers to make direct database calls into the database. Attackers will typically add a malicious database function in the request and cause the database to process and execute their desired function. The consequences can be devastating, e.g., the attacker can change the administrative password of the system or the value of money deposited into his bank account. Thus, it can be seen that a poorly designed web application without (or with minimum) a validation function can allow an attacker to retrieve and place data in any system at will.

Direct SQL injection can be used to (1) change SQL values, (2) concatenate SQL statements, (3) add function calls and stored-procedures to a statement and, (4) typecast and concatenate retrieved data. The following exemplifies how each of these attacks can be executed in the request: [34]

- Changing SQL Values

  ```
  UPDATE usertable SET pwd='$INPUT[pwd]' WHERE uid='$INPUT[uid]';
  ```

  Malicious HTTP request

  ```
  http://www.none.to/script?pwd=ngomo&uid=1'+or+uid+like'%25admin%25';--%00
  ```

- Concatenating SQL Statements

  ```
  SELECT id,name FROM products WHERE id LIKE '%$INPUT[prod]%';
  ```

  Malicious HTTP request

  ```
  http://www.none.to/script?0';insert+into+pg_shadow+usename+values+('hoschi')
  ```

- Adding function calls and stored-procedures to a star statement

  ```
  SELECT id,name FROM products WHERE id LIKE '%$INPUT[prod]%';
  ```

  Malicious HTTP request

  ```
  http://www.none.to/script?0';EXEC+master..xp_cmdshell(cmd.exe+/c)
  ```

- Typecast and concatenate retrieved data

```
SELECT id,t_nr,x_nr,i_name,last_update,size FROM p_table WHERE size = '$INPUT[size]';
```

Malicious HTTP request

```
http://www.none.to/script?size=0'+union+select+'1','1','1',concat(uname||'-'||passwd)+as+i_name+'1'+'1'+from+usertable+where+uname+like+'25
```

f. Mitigation Techniques

There are various mitigation techniques available to prevent Direct SQL injection. Two techniques are as follows:

- Perform in-house filtering that looks at the user’s input for SQL commands. SQL queries should be built from data values and never from other SQL queries or parts thereof. Therefore, the filtering will remove special characters used in SQL statements, e.g., “+”, “,”, “’” (single quote) and “=”. However, this approach will not be able to stop all SQL injection attacks. It can be difficult to implement if the input filtering algorithm has to decide whether the data, that contains special characters, is destined to become part of a SQL query. In addition, as different databases handle special characters differently, the filtering algorithm has to know which database this query might run against and filter off these special characters.
- Construct all queries with prepared statements and/or parameterized stored procedures – A prepared statement or parameterized stored procedure is one that encapsulates variables and will escape/not encapsulate special characters within them automatically. Therefore, any value for the parameters that contain SQL commands will be interpreted as special characters and will be rejected by the statement.

These two techniques complement each other and provide the defense in depth needed to mitigate SQL injection attacks.

(1) Direct Operating System (OS) Commands Attack. This type of attack focuses on the system commands that are available to almost every programming and scripting language. These languages use system commands to utilize the services provided by the OS. File-handling, sending e-mail or invoking the OS tools are some common usage commands by web applications. However, if the input entry is not validated, the attack will result an (1) alteration of system commands, (2) alteration of parameters passed to

system commands, (3) execution of additional commands and OS command line tools or (4) execution of additional commands within the executed command. All of these actions could result in the corruption of data or loss of service. PHP is used to develop applications in IMAS. It is advisable to avoid using the following commands [35]:

- Require ()
- Include ()
- Eval()
- Preg_replace() (with /e modifier)
- Exec()
- Passthru()
- ‘’ (backticks)

One common mitigation technique used is:

- Perform an in-house filtering that looks into the user’s input for OS commands. The set of commands to filter depends on the scripting or programming language.

(2) Path Traversal Attack. This attack exploits the file system of the web server commonly used as a storage repository by many web applications. The file system stores their information, which includes images and static HTML pages. Web servers are easily accessible by anybody who has access to the Internet. An attacker can easily launch an attack on the web server by constructing a malicious request using meta-characters such as “../.” These meta-characters can be used to describe the path of a location or to return data about the physical file location if they are not properly checked or validated. This vulnerability is referred to as “file disclosure vulnerability.” The attacker may also combine this technique with direct OS command attack or direct SQL injection attack to launch a specially crafted URL to path traversal attack. This special attack allows the attacker to traverse system directories and execute system commands. [36]

There are two mitigation techniques available to prevent path traversal attack. They are as follows:

- Use path normalization provided by the development language e.g., PHP, Perl.
- Validate the input entry to remove any offending path strings such as "../" as well as their Unicode variants from the system input.

(3) Null Bytes Attack. This type of attack exploits the vulnerability of the underlying programming language that treats null byte "\0" as the termination of a string. If the input is not validated, a malicious entry containing the null byte will be accepted as a valid entry by the application. This, in turn, will be passed to the underlying applications, which interpret it differently. For example, "AAA\0BBB" is accepted by the applications as a valid entry but is interpreted as "AAA" by the lower-layer applications. This attack can be used to (1) disclose physical paths, files and OS-information, (2) truncate strings, (3) bypass validity checks that look for sub-strings in parameters and (4) cut off strings passed to SQL queries.

Perl, Java and PHP are some of the most popular scripting and programming languages that have this vulnerability. Since PHP is used to develop applications in IMAS, it is pertinent to rectify or mitigate this vulnerability. [37]

One common mitigation technique used is:

- Validate all input entries by checking on all null bytes before passing to underlying applications.

(4) Parameter Manipulation Attack. This type of attack changes the value of parameters sent from the user's browser to the web application to perform a task that is not originally intended. Under this form of attack, the attacker will always target parameters originated from the (1) cookies, (2) form fields, (3) URL query strings or (4) HTTP headers. [38]

(5) Cookies Manipulation Attack. Cookies have always been the preferred method to maintain state in a stateless HTTP protocol because they are a convenient mechanism to store information. Furthermore,

they are used by many web applications to store user preference and session tokens. However, the value could easily be changed by a user and sent to the server with the URL request regardless if it is a persistent cookie, non-persistent cookie, secure cookie or insecure cookie.

Persistent cookies enable a web application to remember the user by establishing an information file on the user's system. A non-persistent cookie is stored in the RAM on the user's system and is destroyed when the browser is closed. A secure cookie is used in secure communication sessions only, e.g., HTTPS. It provides three types of services: (1) authentication, (2) integrity and (3) confidentiality. An insecure cookie is used to manage session cookies and can be used in both secure (HTTPS) and normal connections (HTTP).

In addition, most cookies do not offer any encryption protection. This process creates an opportunity for any attacker to modify the cookie content at will. Typically, most attackers will launch this type of attack on session tokens that make authorization decisions. [39]

There are various mitigation techniques available to prevent a cookies manipulation attack. Three techniques used are as follows:

- The most reliable way to ensure that the data is returned un-modified is to implement a session token. A session token is used to associate the session to the userid. At the server-end, a session table is set up to map the user's session to the user's data variables in the cache/database. When an application needs to check a user's property, it will check the userid with its session table and refer to the user's data variables in the cache/database.
- Build detection hooks to evaluate the cookie for any infeasible combinations that would indicate tampering, e.g., a "detection" flag embedded in the cookie.
- Encrypt the cookie to prevent it from being tampered. The cookie should be hashed and the hash is compared when it is returned to the application or it could be encrypted using symmetric encryption. However, the latter approach will not work if the system is penetrated or compromised. In this situation, a new key will need to be generated.

(6) HTML Form Field Manipulation Attack. Most of the web applications used provide a mechanism that allows users to input information. This information is often stored as form field values and sent to the web application as an HTTP request (GET or POST). HTML can also store field values as hidden fields, which are not presented to the screen by the browser. These hidden fields are collected and submitted as parameters during form submission. However, these form fields, even though they are pre-selected (such as drop-down/checkboxes, free form or hidden), can easily be manipulated by an attacker. The attacker can save the source of the web page, edit it and reload the page in the web browser before submitting the value back to the server. This poses a serious threat as the attacker may change many form fields, e.g., the value, the maximum length tag of the input entry and the visibility of the display. This results in changing the original and truthful values of the form fields.[40]

There are various mitigation techniques available to prevent HTML form field manipulation attack. Two techniques used are as follows:

- Instead of using hidden form fields that reference the properties of a field, which is subjected to changes, it is advisable to use session token to reference properties stored in a server-side cache. Thus, when an application needs to check a user property, it checks the session cookie with its session table and points to the user's data variable in the cache or database.
- Detect the changes in HTML form field manipulation - In this approach, the name and value pair of the hidden field in the form is concatenated together into a single string. A secret key is appended to this concatenated string to form what is known as an outgoing form message (OFM). The secret key will not be sent with the form but is kept at the server-end. The OFM is then hashed to generate an outgoing form digest (OFD) that is added to the form as an additional hidden field. When the form is submitted back to the server, the incoming name and value pair are concatenated along with the secret key into an incoming form message (IFM). Next, a hash is computed to generate the incoming form digest (IFD). If the IFD does not match the OFD that was sent along with the form, then the hidden field is detected as altered and the applications can choose to ignore this submission. This same technique can be used to prevent URL parameter tampering as seen in the next section.

(7) URL Manipulation Attack. In many web applications, HTML forms are normally submitted using one of two methods: GET or POST. In the GET method, all the element names and their values will appear in the query string of the next URL the user sees. Tampering with query strings is very easy; all the user has to do is look at the address bar of the web browser and modify the values of the URL. For instance, this attack can be carried out by a user requesting the server credit $100.00 into his account numbered 12345 instead of a debit transaction of $100.00. This simple manipulation technique can have devastating effects on financial institutions. [41] An example of this command is as follows:

```
http://www.victim.com/example?accountnumber=12345&debitamount=100
http://www.victim.com/example?accountnumber=12345&creditamount=100
```

Unfortunately, HTML form is not the only mechanism that has these vulnerabilities. Almost all navigation done on the Internet is via hyperlinks. When a user clicks on a hyperlink to navigate from one site to the other, or within an application, he is sending a GET request. This GET request has a query string with parameters similar to the form. This allows any attacker to look at the URL window of the browser and change the parameters.

There are several techniques used to solve URL manipulation problems and each technique can be used in different situations. The best solution is to avoid putting parameters into a query string (or hidden form string).

- POST instead of the GET method - The POST method puts parameters to be passed in the body of the request which is not visible within the URL. This eliminates the opportunity for the user to make changes to the URL so that it may bypass the application. However, it does not prevent advanced hackers that typically have the tools from seeing and manipulating the POST parameters.
- Encrypting or signing the entire query string or all hidden form field values - This technique prevents a user from setting the value or seeing the value. This technique is applicable to situations where the parameter value is confidential such as a password.

- Replace variables in URL with page token – This technique replaces all variables in the URL with a page token. A page token is a random number or “nonce” that is only valid with a particular page. It is not valid in another page. At the server end, a page table is created to map the variables to the page token. Thus, whenever a request is made by the user and sent to the application, the application will check the value of the page token. If the page token does not match the value in the table, it indicates that the user has manipulated the variable and the request is rejected. In addition, because the variable is now a random number, the user will not know the actual parameter, therefore, deterring the user from making any changes.
- Hashing the value of the URL query string (or hidden form fields) and append it to the URL – This approach can detect manipulations. If the URL value is changed, the hash value will not match and the application is able to detect it. However, this method does not prevent the user from seeing a value; it only prevents the attacker from changing the value.

(8) HTTP Header. HTTP headers are control information passed from web clients to web servers based on HTTP requests and from web servers to HTTP clients based on HTTP responses. These headers are sometimes used by web applications to determine the URL of the web page from which the request was originated, e.g., a Referer header. This check is to prevent attackers from saving the web pages, modifying them and re-posting them from their own computer. Although most web browsers do not allow header modification, an attacker is still able to modify the header with his own script or by using freely available proxies that allow easy modification. This will lead to SQL injection or a falsified URL. [42]

There are various mitigation techniques available to prevent HTTP header attack. One technique used is as follows:

- Sign and preferably encrypt the header, such as in the case of cookies. However, it is key to note that a Referer header should never be used as any security decision.

Two well known attacks include (1) HTTP request smuggling (HRS) and (2) HTTP response splitting (HRSP). These attacks exploit the web server and intermediate devices such as a (1) cache server, (2) proxy server or

(3) web application firewall. Failures to properly handle malformed inbound HTTP requests, i.e., they read the request differently, allows the attacker to achieve malicious intent.

(9) HTTP Request Smuggling Attack. In this attack, the attacker sends multiple specially crafted HTTP requests that cause the two attacked entities (web server and intermediate devices) to see different sets of requests. This type of attack allows the attacker to smuggle a request to one device without the other device being aware of it. Various types of smuggling attacks are: (1) bypassing the application firewall protection, (2) web cache poisoning, (3) session hijacking and (4) cross site attacking. Noteworthy, the most vital attack is the attacker's ability to bypass the application firewall protection which is deployed in many web sites' protections. In the web cache poisoning attack, the smuggled request tricks the cache server into unintentionally associating a URL to another URL's page (content) and directs the content for the URL. In the web application firewall attack, the smuggled request can be a worm (like Nimda or Code Red) or buffer overflow attack, targeting the server. In this type of attack, the attacker can manipulate the web server's request/response sequencing, allowing for credential hijacking and other malicious outcomes, making it one of the most severe attacks. [43]

There are various mitigation techniques available to prevent HRS attack. These techniques are as follows:

- Application firewall - Although this form of attack can bypass application firewalls, an application firewall is still recommended to be installed to detect and deter this attack in deployment where both cache servers and web servers are deployed. To ensure the firewall is not vulnerable to this form of attack, it needs to be assured that it is secure against this type of attack either through server patches or algorithm.
- Deploy a web server that employs a stricter HTTP procedure such as an Apache server.
- Allow only secure communications, i.e., SSL communication (HTTPS instead HTTP) and terminate the client session after each request or switch all pages to non-cacheable. This process will prevent poisoning in the cache-web server situation.

(10) HTTP Response Splitting Attack. This type of attack exploits the application's failure to reject illegal user input; input that contains malicious or unexpected characters, specifically the CR (carriage return) and LF (line-feed) characters. Here, the attacker sends two requests through the intermediate device. The first request is a single specially crafted request that forces the web server to generate an output stream composed of two responses. These two responses are then interpreted by the intermediate device as two HTTP responses instead of one response, as done in the normal case. The second request would typically ask for some resource on the web server. However, the second request would be matched by the intermediate device to the second HTTP request response, which is fully controlled by the attacker. The attacker, therefore, tricks the intermediate device into believing the server resource is the second HTTP response, while it is, in fact, data forged by the attacker through the web server in the first request.

This form of attack can lead to proxy cache poisoning known as "classic defacement" where the user is directed to a web site controlled by the attacker. Furthermore, it can result in browser cache poisoning where the poison page stays in the cache, waiting for the user to load it. Lastly, this form of attack can be used to attack a script created to handle HTTP errors or to defeat a character-based input filter. [44]

There are various mitigation techniques available to prevent HRSP attack. These techniques are as follows:

- Perform input filtering that validates all input entries and removes the CRs and LFs (and all other hazardous characters) before embedding data into any HTTP response headers, particularly when setting the cookies and re-directing. It is possible to use third party tools (e.g., Sanctum's Appshield and AppScan) to defend against CR/LF injection and to test for the existence of holes before application deployment.
- Always patch the application engine to ensure it is up-to-date.
- Ensure the application is accessed through a unique IP address (i.e., the same IP address is not used for other applications, as it is with virtual hosting).

4. Database Process

Once the data has been validated and sanitized by the applications, the data will be inserted into the database for storage. The stored data can also be retrieved from the database for output response.

a. Vulnerability

Although the database is physically separated from the web and application server and protected by ACLs and other security mechanisms, e.g firewalls, the data is still subjected to subversion if the database system is compromised and the data is being snooped on the Internet. Hence, besides physical access controls, other security mechanisms must be in place to encrypt the data. These security measures must be in place whether data are in the database or in transit from the application server to the database. The data will be viewed as “garbage” by others if it is taken out of the context of the database or if it is snooped by others while it is in transit from the application server to the database server.

b. How to Protect the Data

MySQL, the database used by IMAS, offers several methods to protect the data in transit or in storage. IMAS must use these methods to protect the data. The methods are as follows.

(1) Data Encryption into Database. MySQL provides several methods to encrypt the data in the database. Among these methods, one method is recommended because of its strong encryption algorithms, the Advanced Encryption Standard (AES) encryption and decryption. It is recommended that all of the data or at a minimum, the most sensitive data, e.g., passwords and cell phones/models in IMAS be encrypted using these methods. [45]

- ***AES_ENCRYPT ( ) and AES_DECRYPT ( )*** - these functions encrypt and decrypt data using the AES algorithm. It is encoded with a 128-bit key length, but can be encoded with a 256-bit key length by modifying the source. By far, AES_ENCRYPT() and AES_DECRYPT() are considered to be the most cryptographically secure encryption functions currently available in MySQL. To store the data in encrypted form inside the database, the user uses the following command:

```
INSERT INTO t VALUES (1,AES_ENCRYPT('text','password'));
```

(2) Hashing. MySQL also supports several methods to generate hashes for its data to be stored inside the database. These hash-generating functions are based on well known algorithms e.g., SHA (secure hash algorithm) or variant SHA1 and MD5 (message digest 5). To protect the integrity of the data, especially sensitive data (passwords, cell phone numbers or cell phone models, and encryption keys), it is recommended to hash these sensitive data and store the hashes in a different storage area. The storage area needs to be physically separated from the database so that it will not be compromised/ tampered if the database system is compromised. A script should be written to periodically hash these sensitive data in the database. These hashes should then be checked against the stored-away-hashes to ensure none of the data was tampered with or modified.

- ***SHA( ) or SHA1( )*** – this function calculates and computes a SHA-1 160-bit checksum for a given input based on SHA. SHA1() or SHA() is considered cryptographically more secure than MD5(). The following command shows how to hash a string using SHA1.

```
SELECT SHA1(‘password’)
```

- ***MD5 ( )*** – this function calculates and computes a MD5 128-bit checksum for a given input. The following command shows how to hash a string using MD5.

```
SELECT MD5 (‘password’)
```

(3) Secure Connection between Database and Client. MySQL supports a secure connection using a SSL between the database server and its clients. In the context of this thesis, it refers to the application server which sends the validated and sanitized data to the database. This prevents any attacker snooping on the network from seeing the data in clear-text. The secure connections can be set up using the bundled yaSSL that comes with MySQL or OpenSSL. *OpenSSL* is open-source software *that* generates *digital* certificates and the keys (public and private) to support SSL. The secure connections in MySQL are based on OpenSSL application interface (API) and are available through MySQL C API. [46]

(4) Data Backup. Data backup is a critical process that protects data from any data loss events. IMAS must backup the database (a full backup or an incremental backup) periodically for speedy recovery against server crash, power failure, file system crash, hardware failure or system exploitation. MySQL offers both full and incremental backup options. A user can perform a one-time full-backup to capture the entire database and then use incremental backups to capture subsequent changes. This backup plan optimizes the long (full backup) and short (incremental) backup times for the backup operation. In addition, MySQL allows a user to recover data by specifying the time (starting time or stopping time) or position (in the database) of recovery to recover the data. MySQL also offers error-check capability on the tables and table recovery capability that handles data corruption issues. [47]

To better safeguard the data, all backups should be copied to a media that is not housed in the same server as the database. Additionally, it should be kept in a safe place, e.g., tape or DVD in a safe or off-site.

5. Session Login and Profile Updates

After the user has completed the registration, the user will then log in again to either re-confirm the account or make changes to the profile. Before being allowed access into the system and make any changes, it is necessary to login and authenticate to the system with the username and password. As mentioned earlier, the IMAS system will implement either a one-factor I&A

(identification and authentication) or a two factor I&A. A one-factor I&A uses only the IMAS username and password. However, the two-factor I&A requires the user to authenticate using the CAC card (containing the user's private key) prior to authentication to the IMAS system. Upon successful authentication, the system will create a session and issue a session token to the user. This session token will allow the system to associate the identity of the user to the application for accountability. Furthermore, it provides the context in which all their interactions with the server will be evaluated. [48]

Session tokens are often stored in cookies (created by the web application) inside the user's stations. However, they may also be stored in a static URL, dynamically written URL, hidden in the HTML of a web page or a combination of these methods. Regardless of the form used, a secure session management scheme must be implemented because a poorly designed/ implemented scheme can lead to compromised user accounts. The following section discusses how to implement a secure session management scheme.

a. Secure Authentication

Authentication is the first stage in creating a session. This stage is especially prone to security attacks because once a session is created; all further actions associated with this "legal" session need no further authentication. Hence, a strong authentication mechanism is required. To prevent any subversion, all authentication information should always be passed over a secure medium. In the context of this thesis, the secure medium is established over SSL. In addition, to increase the resistance against dictionary and brute force attack, a well-built password (requiring a minimum number of characters, consisting of a combination of letters, numbers and other special characters), is needed.

Authentication is also subject to severe enumeration attacks from attackers in the hope of accessing the system. Therefore, it is important to ensure the system contains a delay to slow down the enumeration. A threshold should also be set to count the multiple failed authentications for every account in the system so that if the threshold is reached, a warning is generated. This will prompt the user to take necessary actions.

Both of these techniques serve as a measure to monitor consecutive attempts and number of failed attempts over a specified period of time.

b. Session Maintenance

Once the user has successfully authenticated, all actions taken will not be authenticated and will only be identified by a session token/identifier. Thus, it is crucial to create a secure session identifier that is user unique, non-predictable, and resistant to reverse-engineering. The identifier must be cryptographically strong, signed and time-stamped. Additionally, the token/identifier should always be created from a trusted random generator, e.g., Yarrow, EGADS, and encrypted using a well known encryption algorithm, e.g., AES or 3DES. To further enhance the security, all session tokens should be associated with an HTTP instance to prevent hijacking and replay attacks. One such mechanism involves using page tokens, which are unique for any generated page and can be tied to session tokens on the servers.

Hijacking an existing session is a common technique employed by attackers. To prevent this form of attack when the user switches from secure to non-secure pages, the system should always use a different session id. If the common session id must be used for secure and non-secure pages, then the secure pages must always authenticate the user before negotiating new keys. This step will lessen the vulnerability of an attacker using the hijacked session token to access a secure page. If the user is not re-authenticated, the attacker with the hijacked session token/id, could access the secure page and obtain sensitive information. This leads to a confidentiality breach.

Other measures include using long key space for token/identifier creation and managing the time-window of the session.

c. Session Management

Session termination is an important aspect of session management. Any unattended sessions left open over a period of time open a security gap for potential breaches. Similarly, any non-terminated sessions permit attacker to gain access to specific sessions which can lead to identity theft. Hence, it is vital to reduce the vulnerability window of the user(s) session. The session should be terminated in the following conditions:

- Inactive session – A session that has been inactive for a predetermined period of time should be terminated to reduce the window of attack. Typically, this time ranges from 15-30 minutes, depending on the preference of the system.
- Long session – To prevent session hijacking and a brute force attack on existing active session(s), the active session will be terminated after it has reached a prefixed maximum time. Thereafter, the user is required to re-authenticate so that a new session token/id is created. This reduces the window of attack on a replay attack.
- Security Error Logoff Session – Any security error encountered in the application should result in an immediate termination of the session. This prevents any orphaned session from being taken over by an attacker.
- Logoff/Logout Session – A session should always enable the user to logoff/logout the session as the most secure section.
- In addition to a session termination scheme, other session management schemes include [49]:
- Session Forging/Brute Force Detection ("booby trap" session tokens) – This scheme measures against a brute force attack where the attacker tries hundreds to thousands of session tokens embedded in a legitimate URL or cookie. These tokens are never actually assigned to the users but reside on the server end. They will detect this anomaly and ban the originating IP address or lock out the account.
- Session Re-Authentication – This scheme prompts the user for a re-authentication and is applicable when a critical user action such as a money transfer or the purchase of a big-ticket item has to be made. In the IMAS context, re-authentication may be activated if the user tries to change the e-mail address or cell phone number that impacts the delivery of the alerts.
- Session Tokens on Logout – These days, it is common practice for users on the move to use widespread terminals in Internet kiosks, e.g., airports and cybercafés, to access the Internet. This provides a new set of risks because a browser only removes session cookies when a browser thread is closed and most Internet kiosks

maintain the same browser and do not close it. Thus, it is necessary to overwrite session cookies when the user logs off the application.

d. User Profile Update

Once the user logs into IMAS and is issued a session token/id, a default calendar view of the current week will be presented to the user. An example of this is shown in Figure 15.

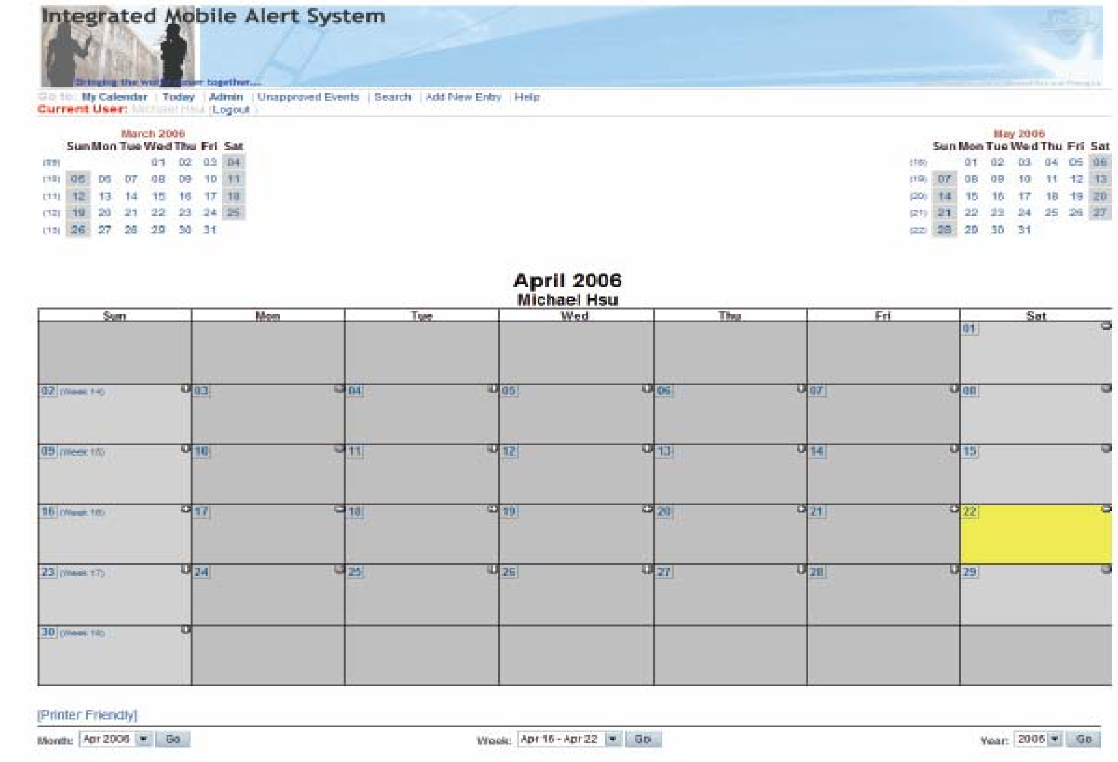

Integrated Mobile Alert System
Bringing the world closer together...
Go to: My Calendar | Today | Admin | Unapproved Events | Search | Add New Entry | Help
Current User: Michael Hsu [Logout]
March 2006
Sun Mon Tue Wed Thu Fri Sat
May 2006
Sun Mon Tue Wed Thu Fri Sat
April 2006
Michael Hsu
Sun
Mon
Tue
Wed
Thu
Fri
Sat
[Printer Friendly]
Month: Apr 2006 Go
Week: Apr 16 - Apr 22 Go
Year: 2006 Go

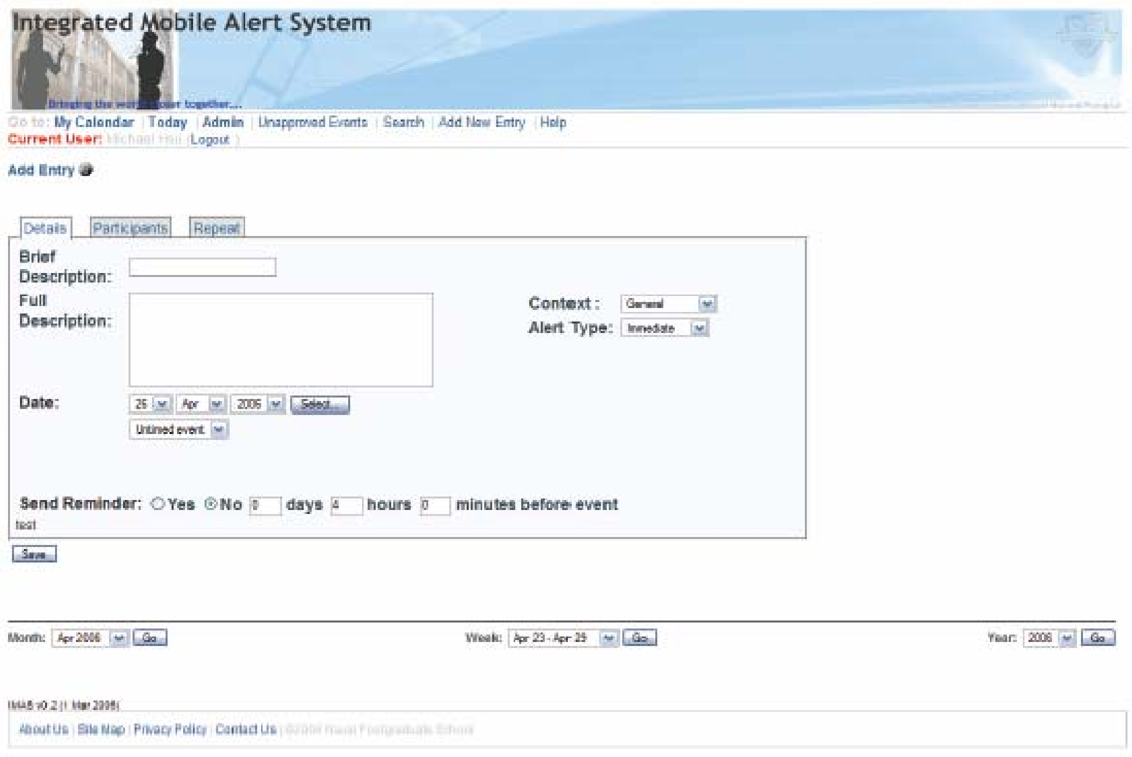
Figure 15. Calendar view

From this point, the user can select a date and enter the following information in the event entry menu:

- Description of event
- Duration of event

- Context of event
- Types of alerts they prefer

Figure 16 illustrates this process.


Integrated Mobile Alert System
Bringing the world closer together...
Go to: My Calendar | Today | Admin | Unapproved Events | Search | Add New Entry | Help
Current User: Michael Hui (Logout)
Add Entry
Details
Participants
Repeat
Select...
Save
Go
About Us | Site Map | Privacy Policy | Contact Us | ©2006 Naval Postgraduate School

Brief Description:

Full Description:

Context : General

Alert Type: Immediate

Date: 25 Apr 2006

Untimed event

Send Reminder: ○ Yes ● No 0 days 4 hours 0 minutes before event

test

Month: Apr 2006 Week: Apr 23 - Apr 29 Year: 2006

IMAS v0.2 (1 Mar 2006)

Figure 16. Event Entry Menu

When the user submits preferences, IMAS will perform the data-correction and data-type check. In the data-correction check, IMAS will verify two fields: (1) date and (2) context along with type of alerts, to prevent user mistakes.

The date entered must not be earlier than the current date because any event before the present date and time is obsolete. Additionally, the date must not be a date too distant (one year later) as the system does not recognize this as a realistic event. The context selection will determine the type of alerts. Thus, a binding algorithm should be built into the software to ensure the alert types correspond to the context selected. The data type validation shall follow the recommendations made in earlier section.

D. ADDITIONAL SECURITY MEASURES

In addition to the recommendations already discussed, there are several other measures that need to be implemented to strengthen the security of the web applications. This section covers some of these measures.

1. Privacy Consideration

In some countries, it is mandated by law to safeguard a user's privacy. Hence, systems that deal with private and sensitive information must take additional steps to ensure a user's privacy is maintained. The following list provides recommendations to protect a user's privacy [50]:

a. Warnings

The system should warn users of the danger of sharing common PCs such as those in Internet kiosks or libraries. Warnings, at a minimum, should include:

- Do not access private data from common PCs because of the possibility of pages being retained in the browser cache
- Recommendations/steps to clear the cache in the browser after accessing private data
- Fact that "temp" files can still remain in the PC after use
- Fact that the proxy server and other LAN users may be able to intercept traffic

b. Page Parameters

The system needs to ensure that personal data is displayed only when necessary. All private data must be masked or partially-masked to prevent people from seeing it. In addition, if possible, the data should not be stored in the cache of the PC. All data should be cleared and removed when the user ends the session. This process can be achieved by setting the (1) expire time, (2) no-cache tags (headers) and (3) no-pragma-cache tags (headers), of the web pages when designing the pages. [51]

2. Event Logging

Event logging is essential to providing key security information about a web application and its associated processes and technologies. It captures the following crucial information:

- Suspicious behavior taking place in the system that can be fed to an intrusion detection system in real-time for root-cause analysis.
- User's actions to provide individual accountability in the web application system.
- Events that are helpful for the reconstruction of events after a problem has occurred, be it security related or not. Event reconstruction permits system administrators or forensic investigators to determine the extent of an intruder's activities and expedite the recovery process.

Failure to enable proper event logging mechanisms in the web applications may result in the system being unable to detect unauthorized access attempts. Hence, it is strongly recommended that logging be enabled in IMAS.

3. Compartmentalization of Data

Data is the most critical part of any system and information leakage is deemed disastrous. Hence, sufficient effort and measures must be taken to protect data.

One such approach is accomplished through compartmentalization. Typically, data can be decomposed as sensitive (private) and non-sensitive (public). The first thing a programmer needs to do in the design of the IMAS system is to identify the sensitivity or classification of the data so that it is possible to separate them either physically (separate hard-drive) or logically (in different folders). Putting both sensitive and non-sensitive data together, creates a higher risk of compromising the confidentiality of the data, even though access control might be in place on each file. This is because an authorized person who has access to the directory and/or files could work around such controls and gain access to the files he/she is not authorized.

In addition, by compartmentalizing the data, it offers an additional layer of defense and the flexibility to better control access to the data. The data entry points,, points where users access the data from the outside, can also be identified and various access control mechanisms, such as mandatory access control (MAC), discretionary access control (DAC) or even role-based access control (RBAC), can be implemented. This strengthens the access control to the data.

MAC is implemented by the system based on an access policy established by the data owner. It will decide on the resources or files the users are allowed to access and what privileges they have based on the login credentials they provide.

DAC is an access policy determined by the owner of the file or other resources. The owner decides who is allowed access to the file and what type of privileges.

RBAC is an access policy based on the roles the person is tasked to perform. The permissions and privileges are tied directly to the role.

By compartmentalizing, the system can encrypt and hash the data to provide an additional level of confidentiality and integrity protection. Thus, it is recommended that IMAS compartmentalize the data.

4. Encryption

Encryption is a key aspect in providing protection to the web applications. It protects the data from being modified and viewed by any un-authorized user(s). However, it should be noted that there are many types of encryptions, each serving a specific layer. SSL is a form of encryption that only encrypts data in the transport layer, protecting the data in transit. It is not specific to the application and will not provide protection to the application layer. Thus, other types of encryption are needed to encrypt the specific fields in the HTML forms and mask off the links within the pages. It is also necessary to obscure the content and structure of the application to avoid probing. All fields and links used by the server should also be signed, encrypted or compared with values stored in the backend to avoid/detect manipulation. Likewise, data stored in the database should be encrypted and hashes generated to protect the data and detect un-authorized modification.

5. Caching

Caching in the form of a cache server is commonly used in many web applications. It is used to enhance performance because it buffers/caches the most frequently-used content and serves it on behalf of the web server. This eliminates the need to request it from the web server on every request. However, this poses a security hazard because if any content pages, i.e., non-multimedia, are put on external unprotected cache servers, these pages (which are part of the application) will expose the application to changes in the pages. If this occurs, it will cause a loss of control over the application flow and increase the complexity in the security design.

In addition, to prevent private records from remaining on the user's computers, especially in a common-use terminal, after the web page is served, it is recommended to use a "no-cache" HTTP header on the web page. Removal of the page by setting this indicator not only adds to the obscurity, but also protects the user's private data from being copied or manipulated.

6. Production Site

The production server used to host the web application should always be separated from the internal servers, servers that belong to the Intranet. This process is needed so that the impact on the servers can be contained. Additionally, the server should not run other software that might disrupt the web application. Furthermore, the server should never be used as a development server to develop an application. This process could leave temporary, old and saved files that were created during the development phase on the server. These files might provide "holes" and traces for the attacker to exploit.

The production server should maintain a sterile environment and all new development shall be done and verified on a development server (which is on a separate domain from the production server) before porting it over to the production server. The production server should never be administered from outside the domain. It needs to be administered within the domain to prevent any abuse. All administration should be executed locally on the production server to prevent any attacks (from inside and outside).

7. DMZ

DMZ is a crucial component of the periphery and network defense for any organization. It provides not only network level protection but also creates a safe environment for web applications. The DMZ segregates the external-facing machines from the internal machines, thereby separating the web applications from each other. By using the DMZ, it permits building differential access for external users. Additionally, it prevents internal DMZ users from accessing the applications through the Intranet or other access methods.

E. CHAPTER SUMMARY

This chapter, discussed how vulnerabilities can affect the system through the design of the system and its processes. In the system design, it was noted that a single box implementation suffers from a single-point of failure and does not offer the flexibility to be scalable and provide high availability. Therefore, it was recommended to separate the applications into different tiers for better management control. Other security mechanisms were also proposed to be applied at each tier to secure the system.

In the design of the applications/processes, it was observed that most well known web attacks exploit the application’s failure to validate the input entry. This in turn allows attackers to introduce malicious codes into the system. It was recommended to use various data validation techniques to prevent these vulnerabilities.

This chapter also proposed implementing a secure session management scheme and other security mechanisms to secure IMAS.

IV. CONCLUSIONS

All systems suffer from vulnerabilities due to bugs found in components supporting the system. These bugs include (1) errors in software code, (2) software design, and (3) mis-configurations in installations.

IMAS, like other systems, also suffers from vulnerabilities. In this thesis, vulnerabilities were identified in the following areas: (1) design and implementation of IMAS, (2) COTS products used and (3) in-house developed processes/applications.

The single-box implementation of IMAS suffers from a number of vulnerabilities. These are: (1) it is a single point of failure, (2) data might be leaked if the box is exploited, (3) it can be used as a launching pad against other corporate systems if the system is on a corporate network and (4) it does not offer the flexibility to apply appropriate security options on different applications. In addition, the management control of system access and services running on the box is complicated. This implementation does not scale well.

COTS products used in IMAS contain vulnerabilities that are due to errors in software code, software design and configuration. Examples include (1) failure to check and sanitize malicious input entry by applications, (2) enable un-used services and accounts and (3) weak passwords.

The in-house developed processes/applications contain vulnerabilities due to their failure to inspect and sanitize inputs. This opens a security gap which many well-known web attacks can exploit. This results in system crashes, theft of information and corruption of data.

This thesis recommended several solutions to correct and prevent noted vulnerabilities. .A 3-tier architecture is recommended for the design and implementation of IMAS. This architecture separates applications and data into different tiers and resolves the issues encountered in a single-box implementation. COTS software used in IMAS are recommended to keep the software version up-to-date so that they are free from reported vulnerabilities. This is achieved by subscribing to vulnerability alert reports and continuously applying patches and fixes to the software whenever vulnerabilities are reported. In addition, recommendations were made to disable un-used accounts and services in order to keep the system secure.

The in-house developed applications/processes are recommended to incorporate numerous validation and filtering techniques to deter common web attacks (that leverage on applications' failure to validate input entry). This includes proposing various types of data-type and data-correction validation techniques for different fields used in IMAS. Furthermore, ways to implement secure session management and data protection inside the database are also recommended to enhance the process security of IMAS.

Other security measures such as (1) privacy consideration, (2) event logging, (3) compartmentalization of data, (4) cache implementation, (5) segregating the roles of the production server and (5) deployment of IMAS in DMZ, are recommended to enhance the security of IMAS.

Lastly, it should be noted that vulnerabilities come from all components supporting IMAS. Hence, efforts must be taken to ensure each component is continuously analyzed to decrease vulnerabilities. This includes in-house developed applications and the design of the database. As these two areas were not covered in this thesis, it is recommended that future studies be made in these areas to identify probable vulnerabilities. The code of the application should be reviewed by a panel of security programmers to ensure it is free from design and code errors. Likewise, the design and implementation of the database should also be reviewed to ensure it does not contain errors. The removal of errors will make IMAS more secure.

LIST OF REFERENCES

[1] The Design and Implementation of a Prototype Web Portal for the Integrated Mobile Alert System (IMAS) by Pong De Le & Michael Hsu, Naval Postgraduate School, June 2006.

[2] ENT News, Windows 2003 Server, http://www.entmag.com/reports/article.asp?EditorialsID=42, accessed 1 October 2006.

[3] Secuna, Vulnerability Report: Microsoft Windows Server 2003 Standard Edition, http://secunia.com/product/1173/?task=statistics_2006, accessed 1 December 2006.

[4] SecurityFocus, Vulnerabilities: Microsoft Windows 2003 Server Standard Edition SP1, http://www.securityfocus.com/bid, accessed 1 December 2006.

[5] Windows Security.com, Windows Server 2003 Hardening List (Part 1), http://www.windowsecurity.com/articles/Windows-Server-2003-Hardening-List-Part1.html, accessed 15 November 2006.

[6] Security Elements of IIS 6.0, Anthony DeVoto, SANS Institute 2003.

[7] SecurityFocus, http://www.securityfocus.com/bid, Vulnerabilities: Microsoft IIS 6.0, accessed 15 November 2006.

[8] Paladion Networks Pvt Ltd., Securing IIS Web Servers, http://palisade.plynt.com/issues/2006Sep/secure-iis-severs/, accessed 15 October 2006.

[9] The SANS Institute, PHP-based Applications. pp. 14-15, http://www.sans.org/top20, accessed 5 September 2006.

[10] Aucentix, http://www.acunetix.com/vulnerability-scanner/, Audit your website with Acunetix Web Vulnerability Scanner, accessed 25 November 2006.

[11] Insecure.org, RPVS-Remote PHP Vulnerability Scanner (open source), http://seclists.org/fulldisclosure/2005/Jan/0552.html, accessed 25 November 2006.

[12] Insecure.org, FIS [File Inclusion Scanner] v0.1, http://seclists.org/webappsec/2006/q3/0304.html, accessed 25 November 2006.

[13] The SANS Institute, Database Software, pp. 16-17, http://www.sans.org/top20, accessed 5 September 2006.

[14] The Design and Implementation of a Prototype Web Portal for the Integrated Mobile Alert System (IMAS), p. 51, by Pong De Le & Michael Hsu, Naval Postgraduate School, June 2006.

[15] Tony Marston, April 24, 2002, The 3-Tier Architecture – is it hardware or software?, http://www.marston-home.demon.co.uk/Tony/uniface/3tierhardsoft.html, accessed 28 October 2006.

[16] ThinkQuest® New York City, Three Tier Web Applications, http://www.tqnyc.org/tutorial/three_tier/index.php?s=T, accessed 28 October 2006.

[17] Wikipedia, Phishing, http://en.wikipedia.org/wiki/Phishing, accessed 29 October 2006.

[18] Anti-phising.info, Download Anti-Phishing freeware, web browser, toolbar and scam blocker, http://www.anti-phishing.info/anti-phishing-freeware.htm, accessed 15 November 2006.

[19] Paladion Networks Pvt Ltd., Anti-Phishing Techniques – Protection Measures, http://palisade.plynt.com/issues/2006Aug/phishing-protection/, accessed 30 September 2006.

[20] Do Security Toolbars Actually Prevent Phishing Attacks? Min Wu, Robert C. Miller, Simson L. Garfinkel, MIT Computer Science and Artificial Intelligence Lab.

[21] “Airline Web Sites seen As Riddled with Security Holes “, Computerworld, February 4, 2002, Gartner analyst John Pescatore.

[22] e-mailsoftwaretools.com, E-mail Account Polling Software – Pop3 polling and e-mail piping script, http://www.e-mailsoftwaretools.com/e-mailaccount.html, accessed 15 October 2006.

[23] e-mailaddresses.com, , Check Your E-mail, http://www.e-mailaddresses.com/e-mail_checkpop.htm, accessed 15 October 2006.

[24] mail2web.com, Pick Up Your E-mail, http://www.mail2web.com/, accessed 15 October 2006.

[25] Miranda, Instant Messenger, About Miranda IM, http://www.miranda-im.org/about/, accessed 17 October 2006.

[26] Miranda Instant Messenger, http://addons.miranda-im.org/details.php?action=viewfile&id=2445, accessed 17 October 2006.

[27] International Numbering Plan, Analysis of telephone numbers, http://www.numberingplans.com/?page=analysis&sub=phonenr, accessed 18 October 2006.

[28] Wikipedia, International Mobile Equipment Identity, http://en.wikipedia.org/wiki/IMEI, accessed 20 October 2006.

[29] Wikipedia, MEID, http://en.wikipedia.org/wiki/MEID, accessed 20 October 2006.

[30] International Numbering Plans, Analysis of IMEI numbers, http://www.numberingplans.com/?page=analysis&sub=imeinr, accessed 20 October 2006.

[31] Telecommunications Industry Association (TIA), Electronic Serial Numbers (ESN) and MEID, http://www.tiaonline.org/standards/resources/esn/codes.cfm#14bit, accessed 21 October 2006.

[32] CIGSecurity.com, Chapter 10: Date Validation, http://www.cgisecurity.com/owasp/html/ch10.html, accessed 23 October 2006.

[33] Cross Site Scripting by Amit Klein, Former Director of Security and Research, a whitepaper from Watchfire.

[34] CIGSecurity.com, Attacks on the System - Direct SQL Commands, http://www.cgisecurity.com/owasp/html/ch11s03.html, accessed 23 October 2006.

[35] CIGSecurity.com, Attacks on the System - Direct OS Commands, http://www.cgisecurity.com/owasp/html/ch11s03.html, accessed 23 October 2006.

[36] CIGSecurity.com, Attacks on the System - Path Traversal and Path Disclosure, http://www.cgisecurity.com/owasp/html/ch11s03.html, accessed 23 October 2006.

[37] CIGSecurity.com, Attacks on the System -Null Bytes, http://www.cgisecurity.com/owasp/html/ch11s03.html, accessed 23 October 2006.

[38] CIGSecurity.com, Parameter Manipulation, http://www.cgisecurity.com/owasp/html/ch11s04.html, accessed 23 October 2006.

[39] CIGSecurity.com, Cookie Manipulation, http://www.cgisecurity.com/owasp/html/ch11s04.html, accessed 23 October 2006.

[40] CIGSecurity.com, HTML Form Field Manipulation, http://www.cgisecurity.com/owasp/html/ch11s04.html, accessed 23 October 2006.

[41] CIGSecurity.com, URL Manipulation, http://www.cgisecurity.com/owasp/html/ch11s04.html, accessed 23 October 2006.

[42] CIGSecurity.com, HTTP Header, http://www.cgisecurity.com/owasp/html/ch11s04.html, accessed 23 October 2006.

[43] HTTP Request Smuggling, Chiam Lin Hart, Amit Klein, Ronen Heled & Steven Orrin, a whitepaper from Watchfire.

[44] HTTP Response Splitting, Web Cache Poisoning Attacks & Related Topics, Amit Klein, Director of Security & Research, Sanctum Inc., a whitepaper from Watchfire.

[45] MySQL AB, 12.10.2 Encryption and Compression Functions, http://dev.mysql.com/doc/refman/5.1/en/encryption-functions.html#function_md5,accessed 24 October 2006.

[46] MySQL AB, 5.9.7 Using Secure Connections, http://dev.mysql.com/doc/refman/5.1/en/secure-connections.html, accessed 25 October 2006.

[47] MySQL AB, 5.10 Backup and Recovery, http://dev.mysql.com/doc/refman/5.1/en/disaster-prevention.html,accessed 27 October 2006.

[48] Developing and Deploying Secure Web Applications, a whitepaper from Watchfire, pp. 6-7.

[49] CIGSecurity.com, Session Management Schemes, http://www.cgisecurity.com/owasp/html/ch07s03.html, accessed 23 October 2006.

[50] CIGSecurity.com, Privacy Considerations, http://www.cgisecurity.com/owasp/html/ch12.html, accessed 23 October 2006.

[51] Developing and Deploying Secure Web Applications, a whitepaper from Watchfire, pp. 8-10.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Yeo Tat Soon
   Director, Temasek Defence Systems Institute (TDSI)
   National University of Singapore
   Singapore