Global Information Assurance Certification Paper

Vilas L Ankolekar
GIAC Security Essentials Certification (GSEC) Practical Assignment,
Version 1.4b, Option 1
Application Development Technology and Tools: Vulnerabilities and threat management with secure programming practices, a defense in-depth approach
10 November 2003

Abstract

This paper addresses the security challenges that exist due to programming flaws, and explains how simple programming practices can reduce the risks. The paper starts with a description of common application vulnerabilities and risks. The vulnerabilities that are discussed include Buffer overflows, SQL Injection, Script Injection, XML injection and others. The application development platforms, technologies and tools that are widely used in the industry and the vulnerabilities that exist in them are discussed next. The technology and tools discussed include Web Services, Wireless, JAVA, C/C++, Web and Database. Further, the secure programming practices that can be used to avoid the vulnerabilities are presented. Since more and more organizations are embracing the outsourcing business model, the importance of having good security practices in such an environment is briefly touched upon. In the end, case study examples have been provided to illustrate the use of secure coding principles.

A Firewall provides security to the Organization's infrastructure to a great extent, but it is not adequate. By identifying vulnerabilities and taking steps to reduce risks during application development stage, organizations can add one more layer of defense against threats. This illustrates the defense in-depth approach to securing applications from real world threats.

1.0 Introduction

The Internet has made inroads into every corner of our lives. Web applications and Web services are the major forces behind the Internet. For the past few years, the pitch over security in applications has reached a new crescendo. New attacks and vulnerabilities are being reported on almost daily basis. "Bad guys" are looking for every opportunity to exploit these vulnerabilities and cause damage on every front. In these circumstances application developers cannot take security for granted. The security in Applications has to be "by design" and not by chance.

When discussing security, it is often emphasized that configuring routers, installing firewalls, securing application servers or providing secure applications access will make applications secure. But security has many aspects and dimensions. Designing applications so that they are secure is a neglected but critical aspect of a defense in-depth approach to securing applications.

The reasons to either ignore or not to spend enough time on securing Applications during development may include:

- Non-existence of secure programming standards: Many organizations think a Firewall is all it takes to secure the applications. Security design during application design is given least priority and hence no standards exist. Although non-existence of standards should not be an excuse, unless organizations create standards and enforce them, security will not be considered a priority.
- Due to tight schedules and market pressures, and in order to release applications to production as quickly as possible, project managers do not plan enough time for security during coding and testing.
- Over-reliance on Network security, Operating System (OS) security and other application security: Many developers think that Network security and OS Security will provide all the protection needed. For many, the meaning of secure applications is to use Secure Socket Layer (SSL) communications. This complacency and over reliance on other security solutions means they give no considerations to secure coding.
- Many developers do not understand security implications in the tools and technologies they use. Application developers spend great deal of time making Applications work, but may compromise on security unknowingly due to their lack of knowledge. Together with lack of written security standards, the problems multiply.
- Many developers tend to believe that “Obscurity is security”: who would want to attack my application? But before they realize it, somebody will find the vulnerability in the code. Easy access to Internet resources has turned even script kiddies into dangerous hackers.
- The application developed today may not be web enabled and

hence security is not considered a requirement. But developers do not realize that eventually most applications will be interfaced to the Web. Also, even though not web enabled, these applications can still be exploited through other web-enabled applications.

- Outsourcing of development: Development Outsourcing changes the definition of perimeter and brings a new dimension to the whole security issue. Security has to be thought in a totally different perspective since a third party has the access to internal network that may not follow security best practices. In order to insure that the code delivered by outsourcing companies is not malicious or does not have security flaws, it is essential that these companies also follow secure programming practices.

Designing secure applications is not very difficult. General approach to securing an application program should include:

- Understand the application development environment in which the application program is developed and deployed, and identify the security risks associated with it.

- Apply secure coding principles to application programs, while making the best use of security infrastructure that is built into the application development and deployment environment.

This paper addresses the security challenges that exist due to programming flaws, and explains how simple programming practices can be used to overcome the risks. Various sections included in this paper are as follows:

- **Common application Vulnerabilities**: This section gives an overview of most of the vulnerabilities that are being exploited by the hackers, which include Buffer Overflow, Format strings, Script Injection and others.
- **Application Development Platform, Technology and Tools**: This section gives an overview of the application development tools, technologies and vulnerabilities around them.

After reading the “Common application Vulnerabilities“ and “Application Development Platform, Technology and Tools“ sections, it is intended that the reader will be able to appreciate the relation among various

technologies, tools and the common application vulnerabilities.

- **Secure programming practices:** The secure programming practices are presented in this section using a model that represents an abstract view of a program.
- **Security issues in the outsourcing model:** This section includes discussion on security considerations in an outsourcing environment.
- **Code reviews, source code analyzers and tools:** In this section the importance of code review is discussed. Also, various source code analyzers and tools are briefly described.
- **Case study examples:** In this section a few examples of programming flaws and means to avoid them are discussed.
- **Conclusion**

It cannot be emphasized enough that a firewall provides security to the Organization's infrastructure to a great extent, but it is not adequate. Security experts agree that best way to deal with the security is to adopt a defense in-depth strategy. One simple security flaw, apart from causing increased application maintenance cost, also can cause considerable public relations damage to an organization. Documenting secure programming practices, making them as mandatory standards and auditing the application code to check for compliance, organizations can mitigate risks of attacks and can reduce the application maintenance costs and prevent public relations disasters.

By identifying vulnerabilities and taking steps to reduce risks during application development stage, organizations can add one more layer of defense against threats. The addition of multiple layers of defense illustrates the defense in-depth approach to securing applications from real world threats. While this paper may not address all the issues in all the tools and technology, the techniques presented in this paper can be extended to other technology and tools as well. If more information is needed the reader is encouraged to go through the references provided at end of the paper.

2.0 Common application Vulnerabilities

The following are some of the vulnerabilities that are commonly exploited by hackers. These vulnerabilities are caused mainly due to bad programming practices.

2.1.1 Access Control flaws

Access control is a mechanism through which applications control data and functions access, allowing only authorized users to perform allowed functions. Access control issues include un-authorized access to the system, functions and access to sensitive data.

While some Access Control flaws may be the result of lack of a clear security policy, improper application design can also cause access control issues. One of the main reasons that can cause Access Control issues is the inadequate input validation. As described in the “Secure programming practices” section, Input Validation is one of the most significant sources of vulnerabilities in applications.

2.1.2 Buffer Overflows

The buffer overflow, one of the common security vulnerabilities, occurs when the application does not perform adequate size checking on the input data. This programming flaw can be used to overwrite memory contents. When the data written to the buffer exceeds the allocated buffer length, the excess data spills over to adjacent memory space. This memory space is normally the application’s program stack that is used to store the address of next piece of code that it will execute. Through Buffer Overflow attack, this memory space can be overwritten causing the application to lose control of its execution.

Under buffer overflow conditions programs may behave in a very strange manner. The results can be unpredictable. In many instances they may not respond or in other words they may hang. This “hang” situation of the application program can turn into a “Denial of Service (DoS)” attack thus making program inaccessible.

A clever attacker can also exploit the buffer overflow conditions to run arbitrary OS commands. By initiating a Buffer Overflow attack, the attacker will insert the address of her piece of code into the memory, causing attacker’s code to run when the application finishes the current execution. When an application runs in a context of a privileged user such as root on UNIX and Admin on Windows, the Buffer Overflow attack can have dangerous consequences. The arbitrary command executed can be a Trojan horse, and the attacker can hijack the host machine.

Buffer Overflow attacks are not easy to initiate. But if the attacker is successful, it

can have very devastating effect on the host machine. The application developer needs to exercise every caution during applications design and development so that these vulnerabilities do not exist. As such, it is not difficult to avoid Buffer Overflow vulnerability. As described in the "Secure programming practices " section, simple techniques such as Input Validation can be used to avoid this vulnerability.

2.1.3 SQL Injection

In a SQL Injection attack, the attacker is able to modify the SQL command that is being executed at the backend database to read, delete, or insert data. The SQL command injection is carried out by variety of means, including form inputs in a HTML page. Many applications create dynamic SQL queries using the input data and these queries are run against backend databases. The application becomes vulnerable to SQL Injection in situations where the dynamic SQL is created without data validation. To illustrate an example consider the following code segment:

String sql = "Select a from b where name =" + in_name;

In the above code if the ***in_name*** variable is not validated there is likelihood that this code is susceptible to the SQL Injection attack.

Using this vulnerability, it is possible to crash a smooth running database. It is also possible to access sensitive data. If the attacker is able modify the data using this vulnerability, there is every chance that the confidentiality, integrity and availability of data will be greatly compromised.

The SQL injection vulnerability is normally caused by poor design, such as improper data validation.

2.1.4 Script Injection and Cross Site Script (XSS) Injection

When an attacker is able to insert scripting commands into the client's web requests it is called a "Script Injection" attack. A method in which a user is tricked in to clicking on an URL link embedded with script command is termed as Cross Site Script (XSS) injection attack. Attackers employ social engineering techniques to trick the user and to carry out XSS attacks.

XSS vulnerable links are sent to the unsuspecting users via email or they are

displayed at several places, including at discussion forums and banners on a web site. When a user clicks on such an URL, through URL redirection the information about that user is sent back to the attacker. The information that is sent might include the session cookies generated during login to a web site. With this stolen cookie, without needing user name and password, the attacker can login to the web site as the user who had clicked on the malicious URL link. In this way the attacker can hijack the session and may manage to take full control of the user's session.

Apart from session hijacking, through XSS, the attacker can also cause other serious problems: steal files from the victim's machine, install a Trojan horse and even modify the data being displayed on the user's browser. XSS attacks can have serious consequences when an attacker is able to modify information presented to the user.

Exploiting XSS vulnerability is easy and almost all the web pages exhibit some kind of XSS vulnerability. It is possible to mitigate this risk to a great extent by employing good design principles as detailed in the "Secure programming Principles" section.

The following paper provides a good discussion on XSS vulnerability: [xss1]

2.1.5 Format strings

In programming languages like 'C', printing functions such as printf () and sprintf () can be called without specifying a format parameter. That means if an application is coded to print without specifying the formats, the application will accept any input, including the strings that are used to control the output format. In such a case there is a possibility that the application has format string vulnerability.

Using this vulnerability it is possible to carry out a Buffer Overflow attack by providing various format strings as input. Strings like "%n","%d","%s" etc, when provided as inputs to a print function that has no format specification, it can corrupt the memory addresses. A crafty attacker can use this vulnerability to modify memory addresses, insert her code in that memory space and thus carry out a Buffer Overflow attack.

For a good discussion on Format String vulnerability following paper is recommended: [format-string]

2.1.6 Command Injection/Shell escape

A Command Injection attack occurs when the attacker is able to specify and execute commands on the OS shell. This attack is more prevalent in shell scripts, such as PERL/CGI programs. Since these scripts are run mostly under elevated set of privileges, the exploit can do a considerable amount of harm. Even a Java program may be subject to an attack if it has been coded to use System.RunTime class.

2.1.7 Race Conditions

When an application is executing discrete steps, it is possible to intercept the application between various steps and execute a piece of code. For example, consider a case where an application writes data to a file. Imagine that it has two steps, the first one is file security check and the second step is writing actual contents to that file. It is possible, as soon as the file check is complete and before the write step, to substitute a new file. If a hacker can exploit this vulnerability, security can easily be compromised.

For more on race conditions please refer to the following:[race]

In the above section, we discussed most of the common application programming vulnerabilities. If we carefully analyze the vulnerabilities discussed so far, we see common threads: almost all of them are caused by simple programming flaws like Input Validation errors, Output Validation errors and other flaws in programming logic.

In the following section we will discuss various application development platform, technology and tools, and understand how they influence the security environment.

3.0 Application Development Platform, Technology and Tools

Although application developers may not get to choose technology or tools for development, organizations do have array of tools and technologies to choose from; from emerging technologies like Web services to age-old languages like C. Many are proprietary and many are open source. When choosing a particular technology many factors affect the decision, but often security is not given consideration. It is good practice to consider security even while choosing a

technology. Since the platforms on which the technologies are implemented also influence the security, it would certainly help thinking about security in the platform decisions too. In other words, security must be thought from the word "go" and it should be given consideration throughout the application development life cycle. Security design should not be an afterthought.

As for the technology and tools, the security environment will also depend on how the developers use them. To better understand the technology impact on security, the developer also needs to understand the technology. The following is a brief introduction to technology and tools. Also, an attempt has been made to list the possible vulnerabilities along with the technology overviews.

3.1.1 Web Applications

The Internet has become the backbone of the new economy and web applications are the enabler of this great industrial revolution. Web applications facilitate presenting the personalized content to the user. They are dynamic and in almost all cases access a back-end database. HTML is the *lingua franca* of the web applications and HTTP is the protocol used for web communication.

A variety of technology and tools are used to generate the contents and provide a user interface and process the page after submitted by the user. One of the oldest server side technologies is CGI (Common Gateway Interface). Newer tools and technologies include Java server pages, Active server pages, PHP (hypertext processor), Java Script and Perl etc.

CGI is one of the most popular technologies that are used for processing the page on the server. Apart from being popular, it is also one of weakest links as far as the security is concerned. Since "Shell escape" is a commonly used feature in CGI, there is a chance that this is exploited for attacks.

Since wide variety of tools and technologies are used in the web development, there is a possibility most of the vulnerabilities described in the "Common application Vulnerabilities " section exist in the web applications.

3.1.2 Web Services

Web Services, meaning a service offered on the web, is a revolutionary technology that enables application-to-application communication. One of greatest benefit of Web Services is that applications can be interfaced without

having to know the underlying mechanics of implementation. The following is a link to a tutorial on web services: [web-services-tutorial]

Typically, in order to communicate with a Web Service, a business application sends a request to a given URL using the SOAP (Simple Object Access Protocol) over HTTP. The Web Service, after receiving the request, processes it, and returns a response. The classic examples include getting driving directions, getting a stock quote etc.

In simple terms, a Web Service is a discrete business process that performs following functions:

- Through the Web Service Description Language (WSDL), a Web Service exposes and describes itself. In order that other applications can understand and access it, a Web Service defines its functionality and attributes [wsdl]
- Through Universal Description, Discovery, and Integration (UDDI), a Web Service gets published into electronic Yellow Pages, so that applications can easily locate it [uddi]
- Using industry standard protocols, SOAP and HTTP, Web Services can easily be invoked.
- When a Web Service is invoked, the results are passed back to the requesting application over the same Internet standard protocol that is used to invoke the service.

Currently there are two major competing Web Services platforms that are available to the industry: JAVA and .NET. Essentially these two platforms use similar underlying technology: XML. In the following web site you can learn lot about web services: [webservices]

3.1.2.1 Simple Object Access Protocol (SOAP)

The Simple Object Access Protocol (SOAP) is a lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment [soap]

SOAP supports different styles of information exchange, including Remote Procedure Call style (RPC) and Message-oriented exchange. RPC style information exchange allows for request-response processing. In the RPC style an endpoint receives a procedure-oriented message and replies with a correlated response message. In the Message-oriented information exchange a message is

sent but the sender may not expect or wait for an immediate response. Message-oriented information exchange is also called Document style exchange.

In summary, Web Services are based on XML and HTTP and there are two competing platforms: JAVA and NET.

3.1.2.2 Security issues in Web Services related to programming

The possible programming-related security issues in Web Services are as follows:

- Buffer Overflow: Since Web Services are exposed as APIs, the data parameters are well known to other applications. The openness of Web Service, which is supposed to be a boon for application integration, can also be a bane. The attacker can send an input parameter larger than the actual size, and if the application is not able to handle the parameter, this might result into a Buffer overflow attack. Also, the attacker could send malformed data that might crash the system.
- XML Injection attack: This is due to the fact the backbone of Web Service is XML, and any inputs passed in XML can be the subject of injection attack.
- SQL Injection: Web Services are also interfaced with the back-end database, which increases vulnerability to a SQL injection attack.

3.1.3 Wireless Applications

Wireless applications provide “Anytime, Anywhere” access to the web applications, making them great productivity enablers. Wireless applications are designed to be easy to use and they are intelligent enough to run on many devices by providing appropriate content delivery and user interfaces. With the convergence of cell phones and PDAs, the user interface is no longer a major issue. Various statistics indicate that the users base of mobile networking is increasing at a fast rate and it is going to reach the same users base as wired users. With the growing user base, organizations are finding greater need to wireless enable their applications and almost all the industries are embracing the Wireless technology.

Several topologies and techniques are used for wireless communication. Wireless technologies use one of these protocols for communications: WAP (Wireless Access Protocol), Bluetooth or 802.11b/a/g. Each of these protocols

has built-in security to avoid break-ins from hackers, but they are not sufficient. Attackers use “Eavesdrop” techniques to hack into network and if the data is not encrypted properly, the data can be stolen. The other common security issue faced in wireless is losing the device. Since the devices are small and portable it is easy to lose. So, it is essential to secure the device access by using strong passwords or using authentication mechanisms like Biometrics. As a means to provide one more layer in the defense, it is important that the Wireless applications should also be coded using “Secure Programming Practices” in order to provide secure access.

Since Wireless applications also use same underlying tools as in the case of Web applications, if not coded correctly they are likely to have following vulnerabilities: Buffer Overflow, Cross-site scripting, Script Injection, SQL-Injection, format strings, shell escape.

3.1.4 XML

XML is the EXtensible Markup Language. It is called extensible because it is actually a “metalanguage”. XML is used for describing other language, which lets you design customized markup languages for limitless different types of documents. XML can do this because it is written in SGML, which is the Standard Generalized Markup Language. XML defines new a way of communication across the web.

XML is also the backbone of Web services technology. For more info on XML follow this link: [xml]. Since XML constitutes an important part in web communication, the security in XML is very crucial. On general XML security issues follow this link: [xml-secure].

A bad design in XML can result in injection attacks. Avoiding this vulnerability should be the primary objective during design time.

3.1.5 C/C++

C/C++ still remains one of the most popular languages for programming for CGI. One of most common vulnerabilities in C is the “Format Strings”. Also, improper memory management can cause vulnerabilities that can result in buffer-overflow attacks. Developers need to be extra cautious when developing web applications in C by following strict programming guidelines.

With regards to vulnerabilities, it is common to have Buffer Overflow, Cross-site scripting, Script Injection, SQL-Injection, format strings, and shell escape vulnerabilities, if secure programming techniques are not used.

3.1.6 JAVA

The program language that was designed to write portable code for the Internet, JAVA by design is inherently one of the most secure programming languages. As in the case of other tools, what makes it vulnerable is insecure programming. It may not be uncommon to have a “Shell Escape” attack in JAVA if System.RunTime is used.

SUN’s security guideline on java [sun-java1] is recommended reading on Java security.

3.1.7 Database Servers

Present day database servers are driving forces behind the IT infrastructure. With the provision of native support to JAVA, XML and other leading edge technologies inside the database, they are no longer just database servers. Most of the databases, including Oracle, support terra bytes of data and at the same time they act as application servers. They support latest technology like Web services and also facilitate exchanging of information in a decentralized, distributed environment. They are the power horse of IT departments with almost all the features of an Operating system built right into them.

Databases also have language extensions inside them along with Structured Query Language (SQL) support. PL/SQL in Oracle is one such example.

With such complexity, the management of database is a Herculean task, which is made easier by the management tools provided by the database vendors. But, managing security in database is a major issue, which has to be carefully planned and executed. This difficulty is not due to security flaws in the database, but due to bad or non-existent security policies and poor application design. As a matter of fact, the databases have the infrastructure built in for the highest level of security.

Since databases support almost all the tools natively and as external procedures, application developers favor using them because it is easier to access and manipulate data with the supported tools. Again, security is a major concern

since the application resides very near to the data. Almost all attacks are possible in a database environment. Any small flaw in the application can easily compromise Confidentiality, Integrity and Availability.

3.1.8 Other Application tools

Other application design tools and technologies include ActiveX, VBScript, Shockwave, Flash, NET platform and more. All of these tools can have one or more vulnerability discussed in the previous sections.

As we have seen so far, the application development tools and technologies greatly influence the security environment. The security approach will also differ based on the application tools and technologies that we choose. The technology like JAVA has built-in security infrastructure that make security implementation easy. Also, some technologies have more security issues than others. Implementing a sound security solution becomes easy if the developer understands the risks involved in a development and deployment environment.

4.0 Secure Programming Practices

While going through various vulnerabilities in the previous sections, one common thread is that most of these flaws are easily avoidable. The threats due to these vulnerabilities can be easily managed by proper discipline during the design and developmental stages. Assuming that the proper design methodology is used in development, in this section we will discuss some simple programming principles that can be used to prevent flaws due to programming errors.

4.1.1 Programming Model

If we draw a model of application program it will look as shown in Fig: 1. It is a simple model, but illustrates the essential aspects of any application. This model is adapted from [david2]. The common characteristics of an application program are:

- Inputs
- Program Logic
- Call outs to external routines
- Access to data (might include read, write or both)

- Outputs

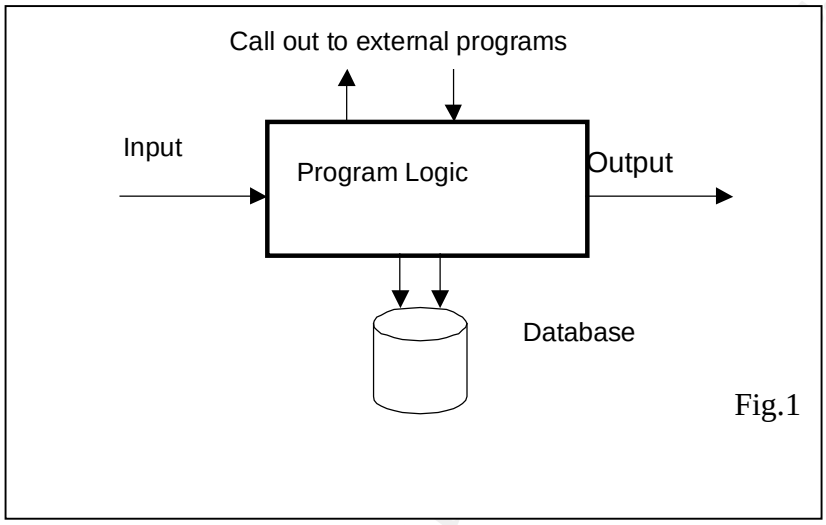

Fig.1

Each of these characteristics creates a possible threat vector and correlates to one of the vulnerabilities discussed in the previous sections. In order that the threats are managed well, the expectations from good programming practices are:

- The application should know what is coming in. All the input data should be validated and all the unnecessary input data should be discarded. The Benefits: Avoiding Buffer flow, Script injection, SQL Injection, Format Strings vulnerabilities and Countering SPAM etc.
- The application should be structured and written with good flow and controls. This includes program flow, data handling, memory handling, error handling etc. The benefits: Avoiding Buffer Flow, Race conditions, Script Injection vulnerabilities.
- The application should only call other external resources that it knows, in other words, should call only the trusted resources. Also, it should make sure that only valid data is passed to and received from the external resources. The benefits: Avoiding Command Injection/ Shell escape, Cross-Site scripting vulnerabilities.

- The application should limit access to only the data as needed for the program logic and processing. If using databases for data access, the application should insure that only validated queries are passed. The "Principle of least privileges" should be strictly followed. The benefits: Avoid SQL Injection, DATA theft etc.
- The application should guard what is being sent out; the output should be properly formatted and data should sent only as per the user's privileges without disclosing too much of program information. The benefits: Avoiding Script Injection, Format Strings, Buffer overflow etc.

Each of the characteristics should be carefully managed in order to design secure application. There are excellent references ([david1], [david2], [owasp1], [owasp2]) on the web that explain how these can be managed to design a secure program. The following sections include secure programming principles from these references.

4.1.2 Input Validation

Un-validated inputs pose the greatest threat among all the types of programming flaws. As per the "The Ten Most Critical Web Application Security Vulnerabilities" security paper published by Open Web Application Security Project [owasp2], "Un-validated parameter" is one of the top vulnerabilities. This proves the importance of validating input before accepting and processing it. In general following principles should be adopted in input validations:

- Data type validations: Check what type of data is input and validate against the data type the application is expecting. Proper error handling should be part of the program flow and control design in case of data type mismatch. Examples of data types include string, integer, URL, user defined data types etc. Although we could rely on programming tools to catch such mismatches, since some tools are forgiving in the matters of data type mismatches, this may not be reliable. Code reviews and peer reviews can help discover such errors. If an URL is used as data type, it is also important to validate the existence of such an URL.
- Check if the parameter is necessary. It is good practice to remove all the unnecessary parameters.
- Enforce number validation, including specifying ranges.
- Check for the data length and enforce minimum and maximum lengths: Hackers can initiate Buffer overflow attack as a direct result of data length

validation errors that occur when input data is specified more than data length.

- Check for Nulls and check if it is allowed as input, since many programming languages do not handle “null” values in an efficient manner.
- Validate the character sets that are being input. It is easy to bypass validations if input is provided in a different character set.
- Check for specific patterns and legal values.
- If the program is validating for a call-out to an external application, it is very important to validate in the context of the security environment of called applications.
- Do not rely on client side validation. It is very important to realize that attackers even use applications like “Telnet” to access web applications by totally bypassing the client. Also, it is possible to make client validation ineffective by using proxy attack, popularly known as man-in-the-middle-attack. If the application depends on the client validation, it may eventually turn into a hacker’s tool.

In short, all the input data should be validated before processing and if data does not pass validations, the errors should be handled properly.

4.1.3 Program control and logic flow

One of the important goals to remember while designing an application is that it should have a good internal structure. Using sound “Software Engineering Principles” should help in achieving such goals. The program flow should be such that the program easily recovers from any erroneous conditions and does not lose control of its execution. Hackers rely on such flaws in the application to take control of the application and effectively take control of the system on which the application is running.

Points to consider while designing are:

- Principles of least privilege: Operate only at a privilege that is required to run the application. In this way even if the application loses control of its execution the damages will be limited.
- Do not combine data and program control.
- Handle errors properly: Many applications, when an error is encountered, do not fail safely. Instead of stopping the processing of the request any further, in order to make the applications “dumb user proof”, they continue

execution by assuming certain conditions. Although this makes application very user friendly, it may also turn it into a hacker's tool. User friendliness should not compromise security. It is also a good practice not to fail completely, but to fail-safe. Failing completely might give an opportunity for "Denial Of Service" attacks.

- Carefully handle errors: When an error is encountered it is important to display proper messages, but limit the messages only to the extent to indicate that there was a problem. It is good practice to log errors into a separate log file and display only cryptic messages to the user.
- Do not allow race conditions: A good example of a race condition can be illustrated when an application creates a temporary file. These are traditionally created in a shared directory, e.g. /tmp or /var/tmp. The trick commonly used by the attacker is to create a symbolic between some important file and the temporary files and cause the secure program to modify the important file. Precaution needs to be taken while dealing with temporary files, including not re-using the same temporary file, getting a random file names, clearing the temporary files etc.

4.1.4 Calling external routines

Almost all programs call external routines. Examples include library files, shell escapes to perform OS operations, call-outs to URLs, web services etc. Calling an external routine is tantamount to handing over the execution of the program to that routine. Great care needs to be exercised while calling out to external routines, including validating input parameters being passed, validating the output parameters etc. But, the first and foremost thing to determine is whether the external routine that is being called is safe. It is very important to understand the security model of the external resource that is being used in a program, and failing to do so may compromise security. Things to consider include:

- Validate inputs and outputs
- Use Application Programming Interfaces while calling out external routines
- Understand the security model of the application being called and decide if it is safe to call that routine.
- Handle the returns carefully
- Have ways to know if the external resource is responding or not. The application should time-out and appropriate actions should be taken if the external resource does not respond within a reasonable time limit.
- Do not pass sensitive data to the external routine. If there is a need to

send such data use secure channels and if possible use encryption. There may be a need to send sensitive data to a library routine where it is not possible to use secure channels or encryption. In such a situation it is very important understand the security environment. If the environment is risky it is recommended that such library routine should not be used.

4.1.5 Database access

Applications normally access data through interfaces provided by databases, except in the cases where the language used for programming is native to the database. One such example of language that is native to the database is PL/SQL in Oracle Database. JAVA is also supported in many databases, thus giving more choice to developers.

The main objective of secure data access is to limit the data access on a needed basis. The best approach is to use the security model built in the database itself instead of implementing other custom approaches. Points to consider are:

- Use the security model built inside the database. Modern databases have excellent security infrastructure built for security and it is a waste of time to implement a custom solution. There is an also risk that the custom security solution is not efficient.
- Check what is passed as input while querying the database. If building a dynamic query utilizing user inputs, check carefully for SQL Injection attacks.
- Use “Principles of least privileges”. By doing so, in case of an attack, the damage will be minimal.

Avoiding data thefts should be the most important goal while protecting the data access through applications.

4.1.6 Handling Output

Handling output is as important as handling input. After all, one program’s output is another program’s input. As a responsible citizen, any program should judiciously send back the information. Good principles in handling outputs include:

- Avoid format string error: A common mistake in languages like “C” is to

take the user's input as a format parameter to control the program's output. This amounts to giving control of programs to the end users by allowing them to run their programs through format strings. Developers do not realize that the formatting programs are very powerful. If used craftily, a Format String attack can easily be initiated using this vulnerability. The application should not format output based on user's input.

- Don't give out too much information: While handling exceptions do not give too much information. If the user base is not trusted, it is often enough to say that the application has failed, instead of saying why the application has failed. If more information is required, it is a good practice to log exceptions as errors in the error files.
- Don't include comments in the outputs. The case in point is web applications; the HTML file can be easily viewed and this may provide online education for the hackers about the application, its versions etc.
- Do not display configuration information. It may be possible to set up application servers or firewalls not to display configuration information. If possible do not allow access to configuration files.
- Have a way to handle unresponsive outputs. Make provisions for time-outs.

So far we have seen a few common approaches that can be applied during application program design. Although it is not possible to guarantee that the programs will be 100% secure, adopting these recommendations should at least help in avoiding most security vulnerabilities.

Having discussed programming principles, let us briefly discuss the security issues involved in an outsourcing development model.

5. 0 Security issues in the outsourcing model

Information Technology Outsourcing refers to employing external service agencies to provide services for various IT functions. The services thus outsourced may include entire IT functions or partial functions like Software Development. There is plenty of literature available on the benefits of outsourcing from top research organizations and as well on the Internet. The economical benefits of outsourcing outweigh many disadvantages that exist in the outsourcing model and hence there is a tendency to overlook these pitfalls during outsourcing decisions. As in the case of internal development, security in the outsourced development is also often neglected. Still, lack of secure

programming practices can have a significant impact. The risks involved are many. In addition to potentially containing any of the vulnerabilities discussed so far, the code delivered by an outsourcing company may also contain hidden routines with malicious purposes. In order to insure that the code delivered by outsourcing companies is not malicious or does not have security flaws, it is essential that these companies also follow secure programming practices

There are many reasons why code delivered by outsourcing companies may pose dangerous security threats. They include:

- The very important benefit that outsourcing provides, the low cost, may be a de-motivating factor to the outsourcing company; to keep the cost down, the outsourcing company may not employ good security controls.
- Lack of control over the developers employed by the outsourcing company.
- Developers may lack skills to understand security implications and employ enough security controls.

In a paper titled "Outsourcing impact on security issues", by Malgorzata Pankowska discusses these and other issues. The following is the link for this paper: [malgorzata]. The paper emphasizes how important it is to have security controls for an outsourcing project.

The outsourcing agencies should be subject to stringent security controls and should be audited to make sure that security standards are adopted by outsourcing agencies as well.

6.0 Code reviews and source code scanners

A developer has just finished coding an application. She has reviewed the code to her satisfaction and finds no flaws in her program. Now the question is: can she certify that the program is without any vulnerability? As per the application design and development best practices, the answer should be "NO". Because, there are chances that the developer has overlooked some part of her application code. Even in cases where the developer is 100% sure about her application program, it is a good practice to employ code reviews and peer-reviews to make sure that the application program has been coded according to the best established practices. Assuming that the peer-review is a part of application design and development process, reviewing code for Security flaws also should

be added to that process.

In the cases where it is humanly impossible to scan the code for flaws, source code scanners and testing tools can be used. The popular code scanners include RATS (Rough Auditing Tool for Security), Flawfinder, Pscan, Splint (Secure Programming Lint), ESC/JAVA (Extended Static Checking for Java).

There are also other tools like AtStake WebProxy, SPIKE Proxy, WebserverFP, KSES, Mieliekoek.pl, Sleuth, Webgoat and AppScan, which are useful in testing the vulnerabilities in the application. Among these tools, Webgoat is an open source tool that is aimed at educating developers on the most common web application security and design flaws using practical exercises [pen-test3]. The following paper is a good reference on Source code analyzers: [giac].

As for the database related scanning tools, please refer to [NGSSoftware] website.

Although these tools cannot point out 100% of the flaws, they will be helpful in identifying the flaws those would have missed otherwise.

7.0 Case study examples

In this section we will see few cases of programming flaws that can cause vulnerabilities and which can be easily exploited. The examples have been simplified to make it easy to follow.

(The case study examples are adapted from these references: [david1], [david2], [owasp1], [pen-test2], [sql-inject1], [sql-inject2], [sql-inject3], [sql-inject4])

7.1.1 Input validation case

As we had discussed in earlier sections, applications can be subjected to attacks if the inputs are not validated. Let's consider following code segment:

***<html><head><title> Order Entry System</title></head><body> ***
<h1>Order Entry Form</h1>
<FORM METHOD="POST" ACTION="/submitorder.jsp">
<table><tr><td> Item </td><td> <INPUT TYPE="text"NAME="itemnumber">
</td>
<td><INPUT TYPE="HIDDEN" name="itemprice" value="4.25">

<td> Quantity: </td><td><INPUT TYPE="text"NAME="totalnum">
***</td>
</tr>***
<tr><td><INPUT TYPE="submit" VALUE="Checkout"></td></tr>
</table></FORM></body></html>

The above code will render a screen shown in Fig.2 to the user. This screen is a front end for hypothetical Order Entry system.

After entering required information, when the user submits the request, "submitorder.jsp" a Java Sever Page is called to process the order.

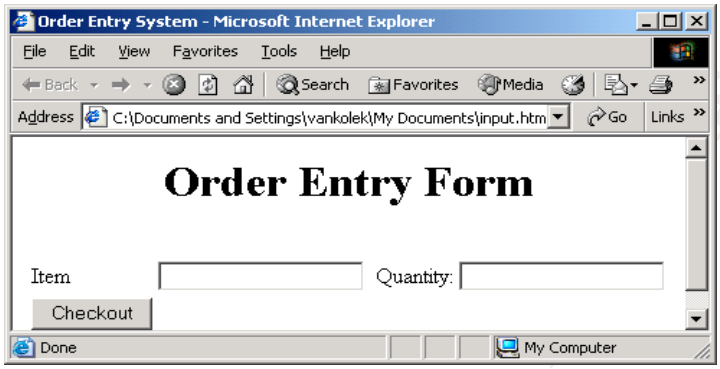

Fig.2

Although the user is presented with only two of the input parameters, the third parameter "itemprice" is hidden within the form. This hidden parameter can easily be seen using "View/Source" options available in the browser. If the user edits the "itemprice" using her favorite text editor, saves the page to her local drive, and submits the form, if the "submitorder.jsp" page does not validate the inputs, there is a good chance that the user gets a discounted price!

This case, apart from illustrating the importance of input validation, also brings out other pertinent points, including:

- Use caution when using hidden variables in a web page. Hidden fields should not store sensitive information. If there is a need to use hidden fields to store critical information, consider using encryption.
- Pages are subject to source code disclosure issues. Examine the page source to check if any sensitive information is disclosed.

7.1.2 SQL Injection case

SQL injection attacks are great security threats to the information that is stored in the databases. It is easy to carry out a SQL injection attack if the application does not perform basic data validations. In the code snippet below we will see how a SQL injection can be carried out:

```
<html>
<head>
<title> Quote Information</title>
</head>
<body>
<center>
<h1>Quote Information</h1>
<FORM METHOD="POST" ACTION="/getquotes.jsp">
<table><tr><td> User Name: </td><td> <INPUT
TYPE="text"NAME="inusername"> </td>
<td>Password: </td>
<td><INPUT TYPE="password"NAME="password"> </td><br></tr>
<tr>
<td><INPUT TYPE="submit" VALUE="GetquotesInfo"></td>
</tr>
</table>
</FORM>
</body>
</html>
```

The above code segment renders a screen with input fields for entering user name and password for a hypothetical Quote Information system. This is a typical system used by the Sales Managers/Agents of an organization to see their quote information, commission information and other pertinent data.

After filling out the information, on submission the web page calls a Java Server Page: "getquotes.jsp". The "getquotes.jsp", has an embedded JDBC statement to access the database and to get the sales commission information from quotes table. Following is a typical dynamic query that is used to get the information based on the user name provided:

```
" Select * " +
" From quotes where user_name = "+ inusername;
```

If the application is coded such that the “inusername” is substituted in the above query without validation, it is very easy to carry out SQL injection attack. We need to simply provide user name as below:

username union select * where 1 = 1

When submitted, the dynamic query will look like as shown below:

Select *
From quotes where user_name = username union select * where 1 = 1

The above query will return all the values from the database even though the user name supplied is not valid and this will make hackers job easy.

Hackers use wide variation of SQL injection attacks. The following papers provide more information on various SQL Injection techniques: [sql-inject1], [sql-inject2], [sql-inject3], [sql-inject4]

The database is the heart of most enterprises. If the application has SQL injection vulnerability it is easy to compromise the very heart of enterprise. So it is important that every effort is made to avoid this vulnerability.

7.1.3 Command injection case

JAVA is a powerful language. When it is provided natively within the database, it can be used to do more sophisticated programming, right within the database.

One of the features that a developer needs most is the ability to call external programs, including host commands, from within the database. With the availability of JAVA within the database, the task has become very easy.

For example, there is need to use the Operating System email utility to send an email based on certain data conditions. Although there are other ways to accomplish this, one of the ways is:

- Create a Java class to execute a host command. Load that Java class in to the database using database vendor supplied routine.
- Define a wrapper database function to call the Java class.
- Use the function to call the mailing program.

In an Oracle Database environment a sample Java class to call host command would look like this:

```
CREATE OR REPLACE AND COMPILE JAVA SOURCE NAMED
"CallHostMail" AS
import java.io.*;
public class CallHostMail {
  public static String sendMail(String MailArgs){
    try{
//call host mail routine by passing the arguments
      Runtime.getRuntime().exec(MailArgs);
      return("0");
    }
    catch (Exception e){
      System.out.println("Error running mail: " + MailArgs +
                         "\n" + e.getMessage());
      return(e.getMessage());
    }
  }
}
```

A wrapper function in Oracle can be created like this:

```
CREATE or REPLACE FUNCTION Call_Host_Mail_Program_Func(MailArgs
bIN STRING)
RETURN VARCHAR2 IS
LANGUAGE JAVA
NAME 'CallHostMail.sendMail(java.lang.String) return String';
```

Although the developer's intention was to send mails using this program, but this program can also be used to run any host command, e.g:

Call_Host_Mail_Program_Func ('/usr/bin/cp /home/abc/a.txt /home/abc/b.txt').

The above execution will copy files, from a.txt to b.txt.

This function can be very useful since it allows calling any host command from database, but there is a danger in it too. A malicious user can insert commands

like this***: Call_Host_Mail_Program_Func ('/usr/bin/rm *.*)*** and wipe out all the data files from the host directory!.

How do we avoid this type of injection? Again, the answer is input validation. This example makes another strong case for input validation, which is a common thread in all the cases of programming related vulnerabilities. Also, it is very important that the developer understands the technology that she is using and the security implications of using such a technology.

The following references illustrate examples for Buffer Overflow, Format strings vulnerabilities: [format-string], [buf-overflow] (please check under "Papers written by NISR team members before NGSSoftware", for these two references)

8.0 Conclusion

As mentioned earlier, firewalls provide security to the Organization's infrastructure to a great extent, but it is not adequate. Security experts agree that best way to deal with the security is to adopt defense in-depth strategy.

Application developers have a responsibility in securing their organization's infrastructure. They have to follow certain guidelines and programming standards while designing the code to prevent their code being used as a hacker's tool. Following the practices discussed in this paper and making them as a standard, application developers can secure their code. Secure coding alone may not wipe out all vulnerabilities, but application developers must at least do their part.

In summary, the secure programming practices include:

- Understand the application development environment in which the application program is developed and deployed, and identify the security risks associated with them.

- Apply basic secure coding principles to application programs, while making the best use of security infrastructure that is built into the application development and deployment environment. The coding principles include:

    - The application should know what is coming in. This helps avoid Buffer flow, Script injection, SQL Injection, Format Strings vulnerabilities and Counter SPAM etc.

- The application should be structured and written with good flow and controls. This helps to avoid Buffer Flow, Race conditions, and Script Injection vulnerabilities.
- The Application should only call the trusted resources. Also, they should make sure that only valid data is passed to and received from external resources. This helps to avoid Command Injection/ Shell escape, and Cross-Site scripting vulnerabilities.
- The application should limit access to the data only as needed for the program logic and processing. This helps to avoid SQL Injection, DATA theft etc.
- The application should guard what is being sent out: This helps to avoid Script Injection, Format Strings, Buffer overflow etc.

As we have discussed in the paper, the application development tools and technologies greatly influence the security environment. The security approach will also differ based on the application tools and technologies that are chosen. The technologies like JAVA, NET have built-in security infrastructure that make security implementation easy. Also, some technologies have more security issues than others. Understanding the tools and technologies and identifying the security risks associated with them is a major step towards achieving a good security solution.

By identifying vulnerabilities and taking steps to reduce risks at the application development stage, organizations can add one more layer in their defense against threats. This strategy of adding more and more layers of defense, illustrate the principles behind the defense in-depth approach of securing the applications from real world threats.

References

[david1] David A. Wheeler , Secure Programming for Linux and Unix HOWTO, v3.010 Edition, 3 March 2003, http://www.dwheeler.com/secure-programs

[david2] David A. Wheeler , Programming Secure Applications for UNIX-Like Systems, 30 March 2003, http://www.dwheeler.com/secure-programs/secure-programming.pdf

[owasp1] A guide to building secure web applications, The Open Web Applications Project, Version 1.1, Sep 22, 2002, http://belnet.dl.sourceforge.net/sourceforge/owasp/OWASPGuideV1.1.pdf

[owasp1-guide-page] "The OWASP Guide to Building Secure Web Application and Web Services" Web Page, http://www.owasp.org/documentation/guide

[owasp2] Ten most critical Web Applications Securities, January 13, 2003, http://umn.dl.sourceforge.net/sourceforge/owasp/OWASPWebApplicationSecurityTopTen-Version1.pdf

[razvan] Razvan's Application Security Page http://members.rogers.com/razvan.peteanu

[sans] Eric Cole, Jason Fossen, Stephen Northcutt, Hal Pomeranz, SANS Security Essentials with CISSP CBK, 2003, Version 2.1, SANS Press

[cert] The CERT® Coordination Center (CERT/CC) Home Page, www.cert.org

[secure-code1] Mark.G.Graff, Kenneth R. van Wyk, Secure Coding, Principles and Practices, 1st edition (July 2003), O'Reilly & Associates

[sun-java1] Security Code Guidelines, 2 February, 2000, http://java.sun.com/security/seccodeguide.html

[sun-java2] Security, Learn how built-in Java security features and tools protect your program and system from invaders. http://developer.java.sun.com/developer/technicalArticles/Security/

[web-services-tutorial] Eric Armstrong, Stephanie Bodoff, Debbie Carson et al,The Java Web Services Tutorial, et al, February 19, 2003,

http://java.sun.com/webservices/docs/1.1/tutorial/doc/

[xml-secure] XML Security Home Page, http://www.xml.org/xml/resources_focus_security.shtml

[malgorzata] Malgorzata Pankowska, “Outsourcing impact on security issues”, http://figaro.ae.katowice.pl/~pank/secout2.htm

[wsdl] Web Services Description Language (WSDL) Home Page, 1.1, http://www.w3.org/TR/wsdl

[uddi] Universal Description Discovery Integration (UDDI) Home Page, http://www.uddi.org

[xml] Extensible Markup Language (XML) Home Page, www.xml.org

[webservices] Web Services Home Page, http://www.webservices.org

[owasp3] The Open Web Application Security Project Home Page, http://www.owasp.org

[hackexposed] Joel Scambray, Stuart McClure, George Kurtz, Hacking Exposed (Second Edition) 2001, Osborne/McGraw-Hill

[oracle-security] Marlene Theriault, Aaron Newman, Oracle Security Handbook, 2001, Osborne/McGraw-Hill

[cgi-security] Cgisecurity Home Page, http://www.cgisecurity.com/

[webservice-oreilley] O’Reilley’s Webservices.xml.org Home Page, http://webservices.xml.com (Nov 10,2003)

[webproxy] AtStake WebProxy Home Page http://www.atstake.com/webproxy

[spike] SPIKE Proxy Home Page http://www.immunitysec.com/spike.html

[kses] KSES Home Page http://sourceforge.net/projects/kses

[mieliekoek] Mieliekoek.pl SQL Insertion Crawler, http://www.securityfocus.com/archive/101/257713

[sleuth] Sleuth http://www.sandsprite.com/Sleuth

[webgoat] Webgoat Home Page, http://www.owasp.org/development/webgoat

[appscan] AppScan Home Page,
http://www.sanctuminc.com/solutions/appscan/index.html

[ngsssoftware] NGSSoftware Home Page, http://www.nextgenss.com

[buf-overflow] “Buffer Overflow for beginners” Web Page,
http://www.nextgenss.com/papers.html ((please check under “Papers written by NISR team members before NGSSoftware” for this reference)

[esc/java] Compaq Systems Research Center Home Page, “Extended Static Checking for Java”, http://www.research.compaq.com/SRC/esc/Esc.html
2000.

[format-string] David Litchfield, “Windows 2000 Format String Vulnerabilities” Web Page, http://www.nextgenss.com/papers.html (please check under “Papers written by NISR team members before NGSSoftware” for this reference)

[flawfinder] “’Flawfinder” Homepage,
http://www.dwheeler.com/flawfinder/

[giac] Thien La, Secure Software Development and Code Analysis Tools, GIAC Practical Paper, September 30th, 2002, http://www.sans.org/rr/papers/46/389.pdf

[oracle-java-9i] Oracle9*i* Java Developer's Guide
Release 2 (9.2), March 2002,
http://download-west.oracle.com/docs/cd/B10501_01/java.920/a96656/toc.htm, =

[pen-test1] Jody Melbourne and David Jorm, Penetration Testing for Web Applications, Part 1, June 16, 2003, http://www.securityfocus.com/infocus/1704

[pen-test2] Jody Melbourne and David Jorm, Penetration Testing for Web Applications, Part 2, July 3, 2003, http://www.securityfocus.com/infocus/1709

[pen-test3] Jody Melbourne and David Jorm, Penetration Testing for Web Applications, Part 3, August 20, 2003, http://www.securityfocus.com/infocus/1722

[pscan] DeKok, Alan. “PScan: A limited problem scanner”, July 7th, 2000. http://www.striker.ottawa.on.ca/~aland/pscan/

[splint] University of Virginia, Department of Computer Science, Splint - Secure Programming Lint, 2002, http://splint.org/

[sql-inject1] Pete Finniga, SQL Injection and Oracle, Part One, November 21, 2002, http://www.securityfocus.com/infocus/1644

[sql-inject2] Pete Finniga, SQL Injection and Oracle, Part Two, November 28, 2002, http://www.securityfocus.com/infocus/1646

[sql-inject3] Chris Anley, (more) Advanced SQL Injection, 18/06/2002, http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf

[sql-inject4] Chris Anley, Advanced SQL Injection In SQL Server Applications, 2002, http://www.nextgenss.com/papers/advanced_sql_injection.pdf

[rats] “RATS”, Rough Auditing Tool for Security Web Page, http://www.securesoftware.com/rats.php

[race] Raynal, Frederic. “Avoiding security holes when developing an application - 5: race conditions”, January 27, 2003, http://www.security-labs.org/index.php3?page=122

[xss1] David Endler, The Evolution of Cross-Scripting attacks, May 20, 2002, http://www.idefense.com/idpapers/XSS.pdf

[xss2] Paul Lindner, Preventing Cross-site Scripting Attacks, Feb 20, 2002, http://www.perl.com/pub/a/2002/02/20/css.html