Prevention of Code Injection Attacks Based on Shell Injection

G.Anil Babu [1], V.Srinivas [2]

[1]Student, AL-AMMER College of Engineering & Information Technology,Anandapuram,Visakapatnam.

[2]Associate Professor, AL-AMMER College of Engineering & Information Technology,Anandapuram,Visakapatnam

Abstract: The existing protection scheme offers no protection against attacks which do not rely on executing code injected by the attacker. The existing system follows von Neumann architecture. Where the memory cannot split into several segments. To forestall the code injection attack, the memory architecture is changed by virtually Splitting it into two segments i.e. code segment and data segment. The change in architecture does not allow the intruder to take charge of the injected code, as the injected code remains no executable. The split memory technique follows Harvard Architecture. Also, Address space layout randomization is followed, where the data are stored in various locations and not as whole in a single memory location. The intruder or an attacker can be tracked by knowing their location, IP address, date and time of the attack etc, that are not available in the existing system. To display the contents of users is the disadvantages in the architectural approach. In this paper we introduce the Shell Injection technique for displaying the user content in the memory according to the content split into number of intruder's information. Our proposed technique also implements URL based attacks in the memory content of the users.

Index Terms: *URL based attacks, Memory split, Code injection Attack, Randomization.*

I. INTRODUCTION

Code injection is the application of a computer bug that is caused by processing unspecified or continent data. Code injection can be used by an attacker to describe code into a computer program to change the course of code injection of execution. The results of a code injection attack can be disastrous. For instance, code injection is used by some computer worms to propagate.

Remote File Inclusion (RFI) is a type of vulnerability most often found on websites. It allows an attacker to include a remote file, usually through a script on the web server. The vulnerability occurs due to the use of user-supplied input without proper validation. This can lead to something as minimal as outputting the contents of the file, but depending on the severity. The existing system

follows von Neumann architecture. Where the memory cannot split into several segments. This type of technique allows the intruder to inject the code in the single segment and executes it. The intruder takes control of the entire code running in the system and it grants access to modify the data and perform activities without the knowledge of the authorized users. Also address space layout randomization is not possible, the entire data are stored in the single address space, and it allows the third party member to obtain all the valuable information, which is being stored in the database.

Most of the web applications are addicted towards code injection attacks. Data is injected by an intruder or an attacker and that third person takes control of the entire system thus leading to loss of secured data and also malfunctioning of the entire system. If the system is attacked, the attacker is not known by the administrator and the person remains invincible. This leads to many disorders in the web applications.

Code injection attacks can be prevented by virtual splitting of memory i.e. code segment and the data segment. It is based on Harvard Architecture. The memory space is allocated in such a way that the code and data segment of the system are stored separately. The injected code remains in the data segment and it will not be executed as it makes unavailable for the processor during the instruction fetch from the memory. Also, the tracking facility enables the administrator to detect the intruder with their IP address, system name, path, location etc. It allows the administrator to take necessary action on the intruder.

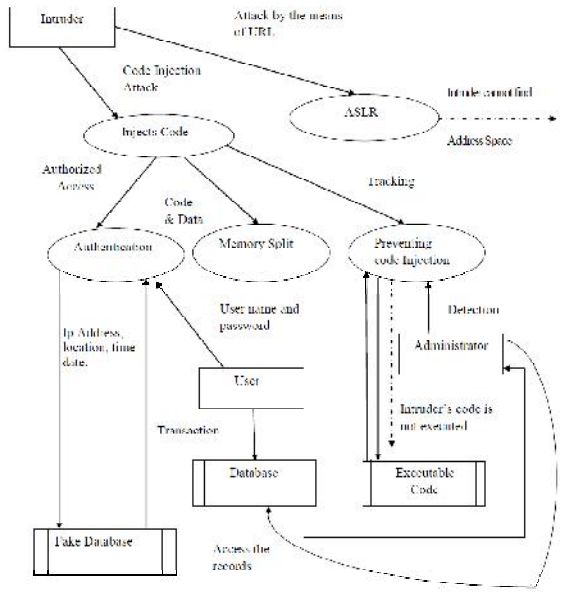

Fig 1: Code Injection Process.

Code injection can be prevented with Address space layout randomization phase, preventing code injection phase. In this intruders attack by means of URL is prevented by ASLR phase. The intruders can't guess by means of the URL displayed in the Address bar. In the code injection prevention, the system will run code only when it is inserted from the Administrators IP address and host name. Thus intruder's code is not executed and prevented from huge disaster to the website.

II. RELATED WORK

Research on code injection attacks has been ongoing for a number of years now, and a large number of protection methods have been researched and tested. There are two classes of techniques that have become widely supported in modern hardware and operating systems; one is concerned with preventing the execution of malicious code after control flow hijacking, while the other is concerned with preventing an attacker from hijacking control flow.

The first class of technique is concerned with preventing an attacker from executing injected code using no executable memory pages, but does not prevent the attacker from impacting program control flow. This protection comes in the form of hardware support or a software only patch. Hardware support has been put forth by both Intel and AMD that extends the page-level protections of the virtual memory subsystem to allow for non-executable pages. (Intel refers to this as the "execute-disable bit". The usage of this technique is fairly simple: Program information is separated into code pages and data pages. The data pages (stack, heap, bss, etc) are all marked no executable. At the same time, code pages are all marked read-only. In the event an attacker exploits a vulnerability to inject code, it is guaranteed to be injected on a page that is non-executable and therefore the injected code is n ever run. Microsoft makes use of this protection mechanism in its latest operating systems, calling the feature Data Execution Protection (DEP). This mediation method is very effective for traditional code injection attacks, however it requires hardware support in order to be of use. Legacy x86 hardware does not support this feature. This technique is also available as a software-only patch to the operating system that allows it to simulate the execute-disable bit through careful mediation of certain memory accesses. PAX PAGEEXEC is an open source implementation of this technique that is applied to the Linux kernel. It functions identically to the hardware supported version, however it also supports legacy x86 hardware due to being a software only patch.

III. EXISTING SYSTEM

To forestall the code injection attack, the memory architecture is changed by virtually Splitting it into two segments i.e. code segment and data segment. The change in architecture does not allow the intruder to take charge of the injected code, as the injected code remains no executable. The split memory technique follows Harvard Architecture. Also, Address space layout randomization is followed, where the data are stored in various locations and not as whole in a single memory location. The intruder or an attacker can be tracked by knowing their location, IP address, date and time of the attack etc, that are not available in the existing system.

IV. PROPOSED SYSTEM

To understand how the most basic shell injection might work, imagine a simple case. A custom script is needed to display file contents to users, but the development team doesn't want to spend time writing a procedure to read the files. Instead, they decide to allow users to specify a file, then use the Unix command cat to display the results.

4.1 Authentication Phase

This is the first module of all applications which contains the user registration and login and administrator's login. In the previous stages, an unknown user also can block the valid user account without knowing the password of the account holder. This is one type of intruder. In the first phase if the user wrongly types the password simultaneously (more than 3 times) then the login will be transferred to a temporary (fake) account page. The intruder does not know that he is in a fake page as it resembles original page.

4.2 Address space Intrusion Avoidance phase

Address space layout randomization could be combined with this phase to prevent the URL based attacks. Even if the intruder attacks the system through URL the control will not be granted to the intruder. If the intruder wants to move to next page after the authentication through URL the user remains in the same address, but the page that is being displayed will be different.

4.3 Preventing Code Injection phase

When the intruder tries to modify any data or create any malicious event, the intruder is not permitted to perform the activities since intrusion is done with unauthorized user name and password. If the changes are done with unauthorized access then the information of the intruder are gathered and it is being sent to the administrator in the secure manner.

4.4 Users Content Information

We saw a basic example of how command injection might work. In this section, we will talk about the varieties of command injections and how they can be executed. Assuming some analysis has found a website function which is likely to be vulnerable to shell injection.

V. EMPIRICAL RESULTS

In this section we describe the two consecutive terms of URL based attack detection. Input design is the process of converting the user oriented input to the computer oriented format. Authentication module is used to log in to the system and perform the operations. Split memory module is used to separate the code and the data segment. Preventing code injection module helps the administrator to know the details of the intruder. The details are collected and it is stored in the database.

Output design generally refers to the results and information that are generated by the system for many end-users; output is the main reason for developing the system and the basis on which they evaluate the usefulness of the application. In this system, with the authenticated user name and password, the user can perform the operations without any restrictions. If the users want to update the data or transfer the amount, the action can be done successfully, where as if the intruder logs in without knowing the password or user name there by giving false details more than three times, the intruder is redirected towards a fake page where the details of the intruder can be tracked. Also when an attacker wants to update any data, the updation is done only temporarily and it is not stored or updated in the database. To the third person the transaction is restricted and on clicking the 'view details' only fake details are displayed. Thus any attack performed by the third person is blocked or restricted.

The attacker may probably corrupt various parts of a program's memory space. Due to the fact that the operating system doesn't understand the working of the running program, it would be infeasible for it to attempt any sort of recovery that would permit the application to continue running. It may be much more feasible, for the application itself to register a call-back function or a special signal handler that the operating system could transfer execution to in the event an attack is detected.

In our approach above mentioned problems can be discussed in the malicious attackers. Assume for a moment that you have found the previous examples page, which takes as an argument a filename as input and executes the shell command "cat" against that file. In the previous example, a semicolon was used to separate out one command form another, to indicate that after the cat command completed, another function should be called in the same line. It is reasonable to assume that a more advanced developer might have filtered out some forms of shell injection, such as by removing semicolons, rendering the previous attack ineffective. There are a number of ways to string shell commands together to create new commands. Here are the common operators you can use, as well as examples of how they might be used in an attack:

Redirection Operators

Examples: <, >>, >

These operators redirect either input or output somewhere else on the server. < will make whatever comes after it standard input. Replacing the filename with < filename will not change the output, but could be used to avoid some filters. > redirects command output, and can be used to modify files on the server, or create new ones altogether. Combined with the cat command, it could easily be used to add unix users to the system, or deface the website. Finally, >> appends text to a file and is not much different from the original output modifier, but again can be used to avoid some simplistic detection schemes.

Pipes

Examples: |

Pipes allow the user to chain multiple commands. It will redirect the output of one command into the next. So you can run unlimited commands by chaining them with multiple pipes, such as cat file1 | grep "string".

Inline commands

Examples: ;, $

This is the original example. Putting a semicolon asks the command line to execute everything before the semicolon, then execute everything else as if on a fresh command line.

Logical Operators

Examples: $, &&, ||

These operators perform some logical operation against the data before and after them on the command line.

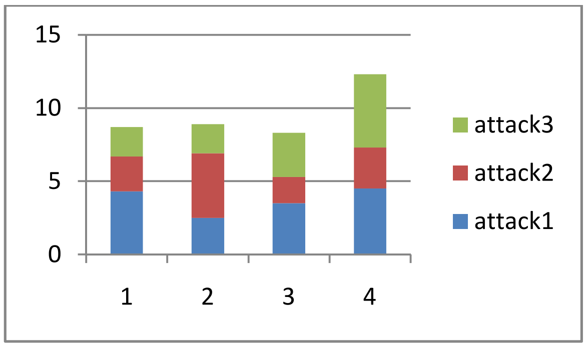

Fig 2: Performance analysis with existing attack detection.

- \`shell_command\` - executes the command
- $(shell_command) - executes the command
- | shell_command - executes the command and returns the output of the command
- || shell_command - executes the command and returns the output of the command
- ; shell_command - executes the command and returns the output of the command
- && shell_command executes the command and returns the output of the command
- > target_file - overwrites the target file with the output of the previous command
- >> target_file - appends the target file with the output of the previous command
- < target_file - send contents of target_file to the previous command
- - operator - Add additional operations to target command.

VI. CONCLUSION

The split memory technique follows Harvard Architecture. Also, Address space layout randomization is followed, where the data are stored in various locations and not as whole in a single memory location. The intruder or an attacker can be tracked by knowing their location, IP address, date and time of the attack etc, that are not available in the existing system. Due to the fact that the operating system doesn't understand the working of the running program, it would be infeasible for it to attempt any sort of recovery that would permit the application to continue running. It may be much more feasible, for the application itself to register a call-back function or a special signal handler that the operating system could transfer execution to in the event an attack is detected.

VII. REFERENCES

1) https://www.golemtechnologies.com/articles/shell-injection
2) Technet.microsoft.com/emus/library/cc723564.aspx,Feb 1, 2001
3) https://www.golemtechnologies.com/articles/shell-injection Oct 9, 2011.
4) doi.ieeecomputersociety.org/10.1109/TDSC.2010.1 by R Riley - 2010 - Cited by 26 - Related articles Dec 15, 2010.
5) Ryan Riley, Xuxian Jiang, Dongyan Xu," An Architectural Approach to Preventing Code Injection Attacks", Nov 2010.