Journal of Engineering Science and Technology
Vol. 17, No. 1 (2022) 0075 - 0084

SURVEY OF OS COMMAND INJECTION WEB APPLICATION VULNERABILITY ATTACK

MOHAMMAD ALAHMAD*,
ABDULRAHMAN ALKANDARI, NAYEF ALAWADHI

Computer Science Department, College of Basic Education,
Public Authority of Applied Education and Training, Ardiya Campus, Kuwait City, Kuwait
Senior Technical Support Engineer (MIS Department) Employment Communication & Information Technology Regulatory Authority (CITRA): Kuwait City, Kuwait
*Corresponding Author: malahmads@yahoo.com

Abstract

Web applications plays a major role in our daily life. Web applications are used in our daily shopping, socializing, and banking. OWASP 2017 report categorized top 10 of web application vulnerability attacks. This paper discusses one of these vulnerability attacks which is OS command injection attack. Based on OWASP 2017 report, OS command injection considered one of the highest threats facing web applications security. We have discussed in detail the steps, methods, some examples, the impact of such an attack and the prevention of OS command injection web application vulnerability attacks.

Keywords: Attacks, OS command injection, Web application security, Web application, Web Vulnerabilities.

1. Introduction

Web application is an application software that run on a webserver unlike computer-based software applications that only runs on a computer's operating system locally. The rapid spread of software developments around the globe is increasing exponentially. Insecure software can ruin our financial, healthcare, and other critical information stored on its server. Developers, organizations, and communities need to raise security awareness that jeopardize their businesses. More precisely, they need to share the security vulnerabilities prevalence data and attacks facing their businesses. OWASP is a non-profit organization founded in 2001 to provide experts, designers, and researchers to security protection to their web applications.

OWASP community consist of corporations, members, developers, and volunteers that unite to share the most web application security risks to increase the security level of their businesses and create a new application security standard. To achieve a basic security, web applications require the three-security triangle (CIA) which are: data Confidentiality, data Integrity and Availability of web application. Data confidentiality refers of protecting information from an unauthorized party. This can be accomplished by encrypting all user information stored on web applications. Data integrity refers to the accuracy and consistency of information of an entire life cycle of a web application. Where the life cycle of a web application consists of the information stored in database of a web application, processing and retrieving data. Thirdly, availability of web application refers to the ability of users to access information of a web application.

OWASP released three reports in the last decade that summarize the most critical web application security risks. In OWASP's 2010 top 10 report [1], the top web application vulnerability risks were injection, cross-site scripting followed by broken authentication and session management. However, in OWASP's 2013 top 10 report [2], the top web application vulnerability risks were injection, broken authentication and session management, and then cross-site scripting, respectively. That means, broken authentication and session management became number 2 top of web application vulnerability risks instead of number 3 as was in OWASP 2010 report. That is because researchers have paid more attention to such attack and explored more. Their risk methodology is based on evaluating the threat agents, attack vectors, weakness prevalence, weakness detectability, technical impacts, and business impacts of the vulnerability [3].

This paper discusses a comprehensive detail of one the first OWASP web application attacks which is OS command injection attack. These details are the types of OS command injections, examples of OS command injections and how to prevent them. In 2009, the rise of the first cryptocurrency called Bitcoin which operated at the top of blockchain decentralized public ledger technology (DLT) changed the future shape of the existing global finance system, for ex. Visa payment system. Blockchain's DLT network is a peer to peer network which eliminates the third party, for ex. banks, from being the median of exchange between the two end parties, hence, blockchain try to ends the domination of the federal banks. Which means, the power of decision making is distributed between nodes among the network equally and that is called decentralized network "peer to peer traditional network". So, blockchain revolution created a ruthless battle between Visa centralized existing system and cryptocurrencies that built at the top of blockchain decentralized distributed public ledger network which become fiercer since the Bitcoin launch [4].

2. OS Command Injections

Description. sometimes, web applications need to communicate with the operating system commands to execute the underlying host operating system and the file system. This can be performed to run system commands and launch applications using different programming languages or scripts such as shell, python, or PHP. Operating system (OS) command injection (also known as shell injection or OS injection) is a web vulnerability attack that allow the attacker to inject unwanted user data in the form of cookies, forms, HTTP headers, etc. [5]. to operating system commands on an application's server to compromise the application and its data. This attack is possible when a user supplies a non-valid information before accessing of the targeted vulnerable web application. The user data supplied could be in a form of forms, cookies, HTTP headers etc. This attack is performed in a system level commands to enable the attacker to gain an unauthorized access or to retrieve information from a vulnerable web application server. These are the steps of how OS commands inject works:

i. Attackers must identify vulnerabilities of the targeted web application; this allow attackers to insert a malicious code into the OS and gain (some) functionality of the web application without having direct access to the OS.

ii. Attackers alters the website content using HTML language by inserting the malicious code in a form of cookies or HTTP headers.

iii. Once the malicious code is inserted into the web application server and become as an origin code of the infected web application, now, browsers interpret this code and allow attacker to execute specific commands across victim's computers.

It is clearly that OS command injection vulnerability occurs when the user input is not validated properly, instead, a malicious code inserted by an attacker is passed to the web server without any sanitization. For example, a web application provides a ping functionality for any IP address through his web application interface to any user to confirm a host connection. That means, the ping command is passed directly to the web server. At this stage, attacker can inject unwanted system command along with ping command using metacharacters. Thus, the web application passes it to the web server for execution and allow a gain control of that particular web server. Fig. 1 illustrates the OS command injection.

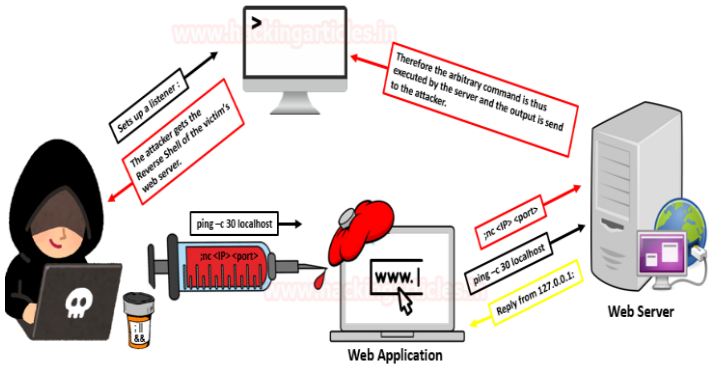

Fig. 1. OS command injection [5].

Metacharacters. A character that has a special meaning to a computer program. They are used to separate the actual input user to the web application from the malicious unwanted system commands inserted by an attacker. The commonly used metacharacters are [6-8]:

Table 1. Methods. OS command injections consists of two types:

Operators	Description
;	The semicolon is the most common metacharacter used to test an injection flaw. The shell would run all the commands in sequence separated by the semicolon.
&	It separates multiple commands on one command line. It runs the first command then the second one.
&&	If the preceding command to && is successful then only it runs the successive command.
|| (windows)	The || runs the next command to it only if the preceding command fails i.e., initially it runs the first command, if it doesn't complete then it runs up the second one.
|| (Linux)	Redirects standard outputs of the first command to standard input of the second command.
‘	The unquoting metacharacter is used to force the shell to interpret and run the command between the back ticks. Following is an example of this command: Variable = “OS version uname-a” && echo$variable
()	It is used to nest commands.
#	It is used as a command line comment.

2.1. Results-based command injections

The attacker inserts a malicious code into the vulnerable application and infer back by this application by an output implying its vulnerability. Based on that output, the attacker can modify the inserted input to obtain a specific information from such a server.

2.2. Blind Command Injections

The attacker inserts a malicious code into the vulnerable application and this application does not deliver back any output. That means, the results are not shown on the screen. This can be categorized as:

2.2.1. Time based technique

The attacker assumes and guess the results of the injected OS command and based on that, he can decide whether the application is vulnerable to that attack based on time blind command.

2.2.2. File based technique

The attacker writes the results of the execution of the injected OS command.

Basic OS command injection. Chandel [5] tried to ping a local IP address (172.0.0.1) using his own website after degrading its security level using Damn Vulnerable Web App (DVWA) software. DVWA is a PHP/MySQL web

application that intended to help students, teachers, and professionals to test their security skills and increase the security levels of their designed web applications in a legal environment. From Fig. 2 image, we can see the output result of the ping from [8-10] website using DVWA software.

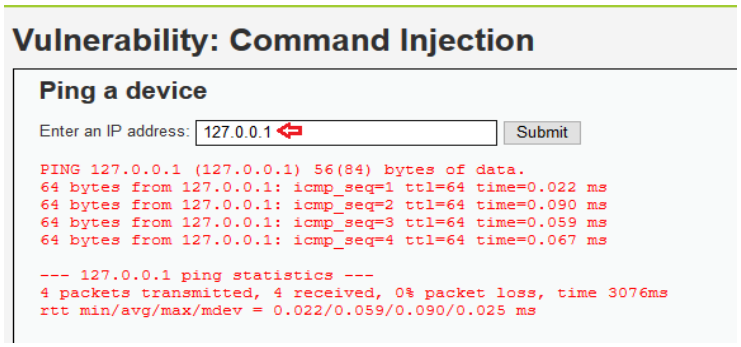

Fig. 2. Ping test of a local host

The output result imply that the web site is vulnerable to OS command attack. Then, he injected a metacharacter after the local IP address and waited for the response from the website server as shown in Fig. 3.

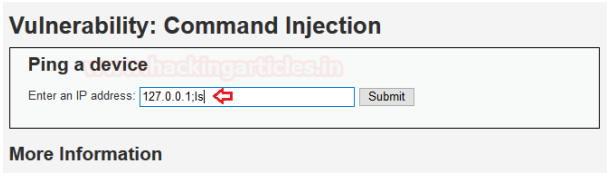

Fig. 3. Inject of metacharacter.

As Fig. 3 shown, he entered a metacharacter semicolon (;) and then an arbitrary letter (ls). The output result of this test is shown in Fig. 4.

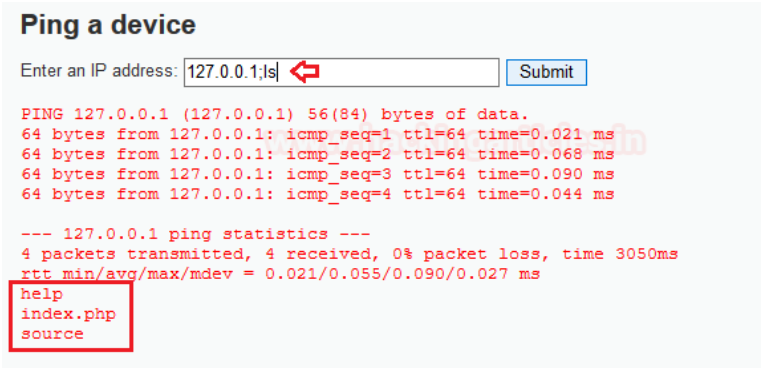

Fig. 4. Output result of OS attack.

As shown from Fig. 4, the output result listed the directory of where the main page is saved on the website server. To thwart such metacharacters attack as shown earlier, developers' setup a blacklist which contain different commonly combinations of metacharacters to protect their web applications from OS command attack. As a result, OS command attack using metacharacters only work when developers forget to add a specific metacharacter combination to his blacklist. Figure. 5 shows a researcher [11-15] captured a password file after injecting all the combination possibilities of metacharacters along with a local IP address.

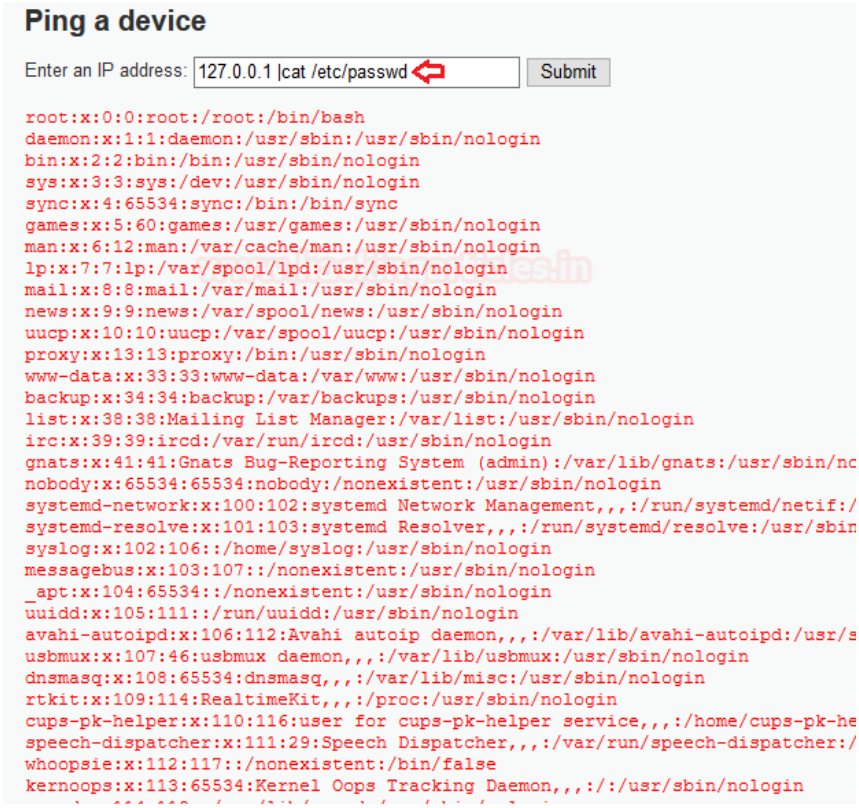

Fig. 5. Capturing a password file using the metacharacter "|".

Examples. To understand more about OS command attack, some examples are described below by OWASP.

Example 1. PHP is an acronym stands for Hypertext Pre-processor. It is an open source general scripting language frequently used in web development. In the following code, PHP will attempt to execute the contents within the backtick quotes as an OS command [16].

```
system("cat /etc/passwd");
exit();
could also be programmed in below manner :
'cat /etc/passwd`
exit();
```

Example 2: UNIX

UNIX is an operating system that is developed in the Bell Laboratories of AT&T. The following code is a bind with UNIX command [17].

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv) {
char cat[] = "cat ";
char *command;
size_t commandLength;

commandLength = strlen(cat) + strlen(argv[1])+ 1;
command = (char *) malloc(commandLength);
strncpy(command, cat, commandLength);
strncat(command, argv[1], (commandLength -
strlen(cat)) );

system(command);
return (0);
}
```

By using the above code, we simply obtained the file requested. However, if we inject another command to the end of next line, the command will be executed by cat Wrapper.

```
$ ./catWrapper Story.txt
When last we left our heroes...
```

After writing the command cat Wrapper, a semicolon and an arbitrary command are added to execute other commands.

```
$ ./catWrapper "Story.txt; ls"
When last we left our heroes...
Story.txt               doubFree.c              nullpointer.c
unstosig.c              www*                    a.out*
format.c                strlen.c                useFree*
catWrapper*             misnull.c               strlength.c
useFree.c
commandinjection.c      nodefault.c             trunc.c
writeWhatWhere.c
```

Example 3

Sulaiman and Rahi [18] wrote a simple program as showing below, that accepts the filename as an OS command and return the contents back to the user of that file. The program is installed setuid root to enable administrators to inspect their systems for training purposes and without damaging their main systems.

```
int main(char* argc, char** argv) {
char cmd[CMD_MAX] = "/usr/bin/cat ";
strcat(cmd, argv[1]);
system(cmd);

}
```

Since the written program is not linked to the system root, then, the call system () cannot be performed or executed.

The impact of the attack. Problems resulting from OS command injection attacks varies from minors to majors based on the security mechanisms provided by the application server and how it is vulnerable to such attack. These results could be corrupting, manipulating, or stealing of a data, launching distributed denial of service (DDoS) attack, controlling a server and etc.

Prevention. It is vital to address the security vulnerabilities that leads to OS command injection attack of the web application server. When developing web applications, developers should be careful how to pass user input to functions of a web application's operating system. Simply, we can mitigate and prevent OS command injection attack by disabling calls out to OS command from application layer code. But in some cases, calling out to OS command from an application code is needed, therefore, we concluded such prevention methods to stop such an attack as follows.

3. Preventing Manipulation at The Source

Any application that has access to a web server, it is vital to stop accessing and controlling the OS commands from that application.

4. Rejecting unacceptable code

User input validation and sanitizing to the web server is required by the server to avoid injecting unwanted code by an attacker and that can be achieved by creating:

4.1. Blacklist

Which contain different commonly combinations of metacharacters to protect their web applications from OS command attack. As a result, OS command attack using metacharacters only work when developers forget to add a specific metacharacter combination to his blacklist.

4.2. Whitelist

Which contain only the values list that are accepted by the web server and reject all other values.

5. Controls Over APIs

When an application needs to pass parameters to OS, we can only launch a specific process based on a specific name and command line to be passed to OS instead of enabling API of passing any command string to OS shell interpreter.

6. Conclusion and Future Works

In the last decade, web application security attracted researchers to explore more practical solutions to prevent vulnerabilities attacks happing in our daily life transactions. OS command injection web application vulnerability attacks is one of the top 10 vulnerability attacks classified by OWASP 2017 report. This paper discussed in details OS command injection web application vulnerability attacks.

Precisely, we discussed the steps, methods, some examples, the impact of such an attack and the prevention of OS command injection web application vulnerability attacks. As a result, web developers must be aware of preventing OS command attacks by using the proper countermeasure protection method that suits their web application to stop or at least mitigate suck attack.

Acknowledgements

This work supported and funded by the public authority of applied education and training, research project no (BE-21-04).

References

1. Open Web Application Security Project (OWASP). (2017). OWASP Top Ten Project. Retrieved 15, February 2021, from https://www.owasp.org.
2. Open Web Application Security Project (OWASP). (2010). OWASP Top Ten Project. Retrieved 15, February 2021, from https://www.owasp.org.
3. Open Web Application Security Project (OWASP). (2013). OWASP Top Ten Project. Retrieved 15, February 2021, from https://www.owasp.org.
4. Al-Khurafi, O.B.; and Al-Ahmad, M.A. (2015). Survey of web application vulnerability attacks. 4*th International Conference on Advanced Computer Science Applications and Technologies* (*ACSAT*). Kuala Lumpur, Malaysia, 154-158.
5. Chandel, R. (2021). Comprehensive guide on OS command injection. Retrieved 15, February 2021, from https://www.hackingarticles.in/comprehensive-guide-on-os-command-injection/
6. Barracuda Networks, I. (2021). OS-command injection. Retrieved 18 February 2021, from https://campus.barracuda.com.
7. Makanadar S.; Solankurkar V. (2013). An approach to detect and prevent SQL injection attacks using web service. *International Journal of Science and Research*, 2(4), 242-245.
8. Choudhary A.; and Dhore, M. (2012). CIDT: detection of malicious code injection attacks on web applications. *International Journal of Computer Applications*, 52(2), 19-26.
9. SQL injection inference attacks. Retrieved 15, February 2021, from http://www.sqlinjection.net/inference.
10. Jones M. (2014). Top ten security risks: broken authentication and session management. Retrieved 15, February 2021, from https://www.credera.com.
11. Doupe, A. (2014). *Advanced automated web application vulnerability analysis*. Ph.D. Thesis, University of California Santa Barbara, California, United States.
12. What the Verizon Report (2015). Tells us about web app attacks. Retrieved 15, February 2021, from https://www.acunetix.com.
13. Banach, Z. (2021). What is session hijacking: your quick guide to session hijacking attacks. Retrieved 13 April 2021, from https://www.netsparker.com.
14. Crick, T.; Davenport, J.H.; Irons, A.; and Prickett, T. (2019). A UK case study on cybersecurity education and accreditation. 49*th Annual Frontiers in Education Conference* (*FIE*),1-16.

15. Hamam, H.; and Derhab, A. (2021). An OWASP top ten driven survey on web application protection methods. in risks and security of internet and systems. 15*th International Conference* (*CRiSIS*), Paris, France, 235.
16. Malallah, H.; Zeebaree, S.R.; Zebari, R.R.; Sadeeq, M.A.; Ageed, Z. S.; Ibrahim, I. M.; and Merceedi, K.J. (2021). A comprehensive study of kernel (issues and concepts) in different operating systems. *Asian Journal of Research in Computer Science*, 8(3), 16-31.
17. Maalouf, A.; and Lu, L. (2021). Precise command injection analysis in android applications. 5*the International Conference on Management Engineering, Software Engineering and Service Sciences*, New York, United States, 1-7.
18. Sulaiman, R.B.; and Rahi, M.A. (2021). A framework to mitigate attacks in web applications. *Journal of Computer Sciences* (*IUP*), 15(1), 22-59.