

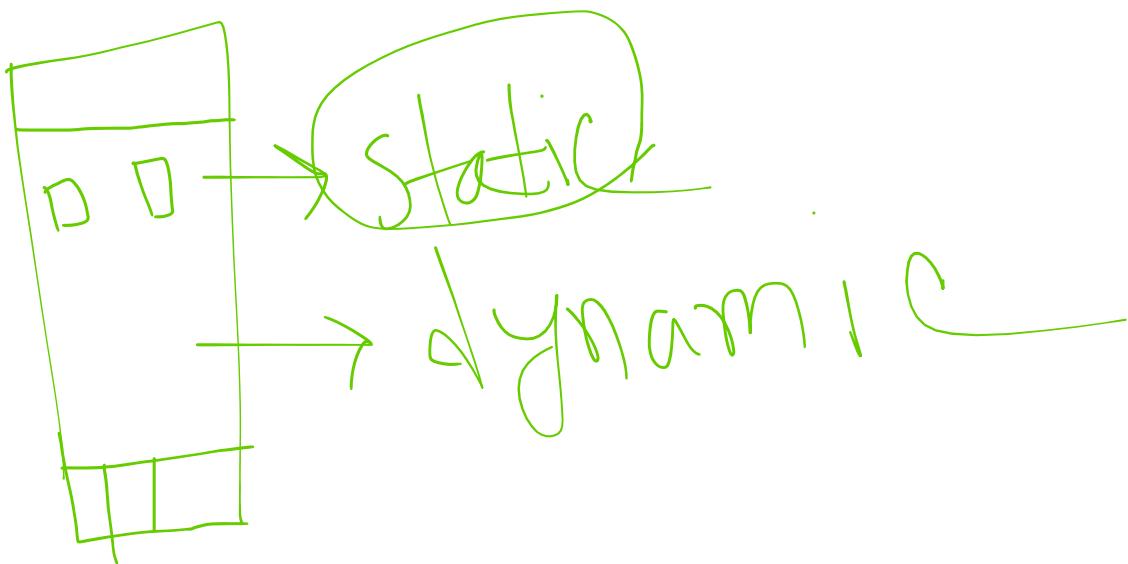
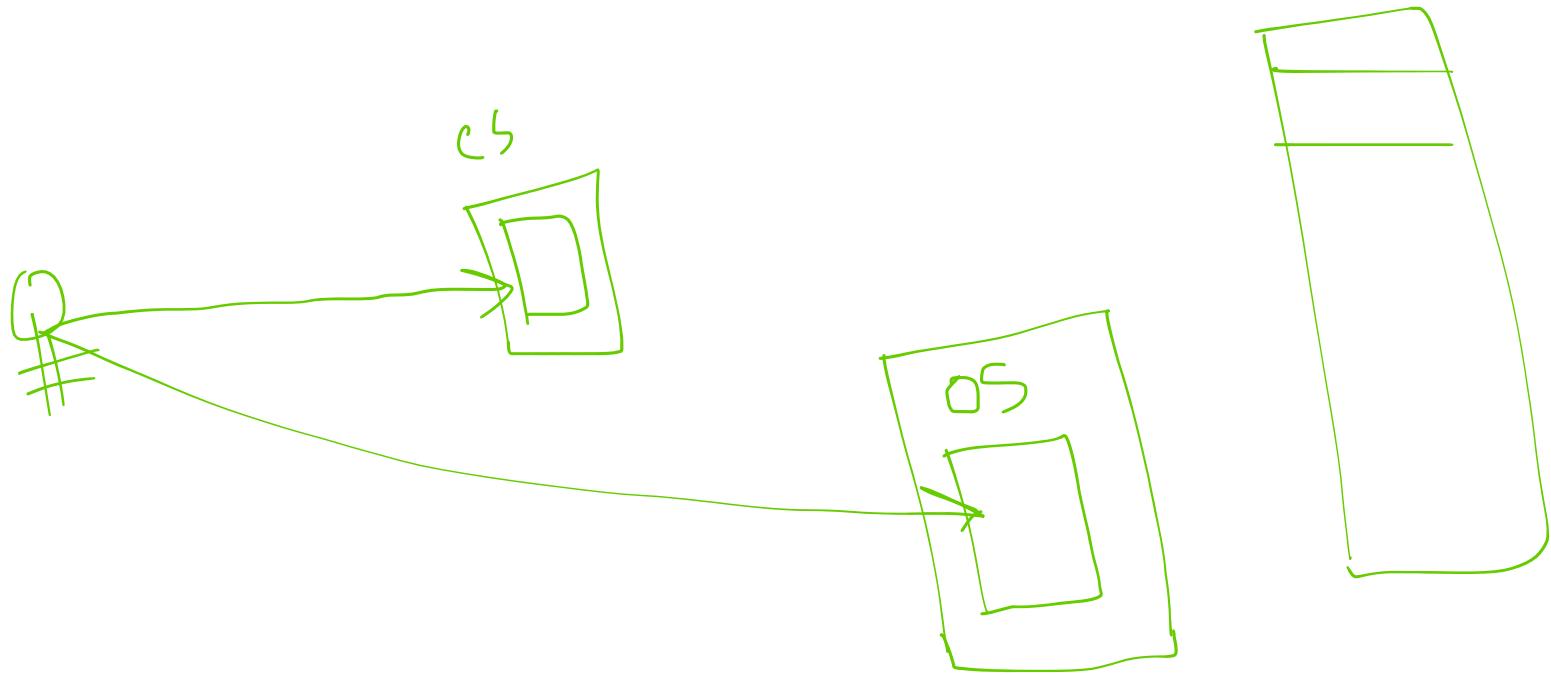
Web Cache Deception

when an attacker tricks a web cache into storing private user data, making it accessible to others.

Web cache

is a technology that stores copies of web content (such as HTML pages, images, videos, and scripts) to reduce load times and improve website performance.

cache server
origin server



What Should Be Cached Safely

Static Assets (Unchanging Resources): These are files that rarely change and are shared across all users.

CSS files (.css): Stylesheets used to define the appearance of the website.

JavaScript files (.js): JavaScript files that are used for functionality but are typically **not user-specific**.

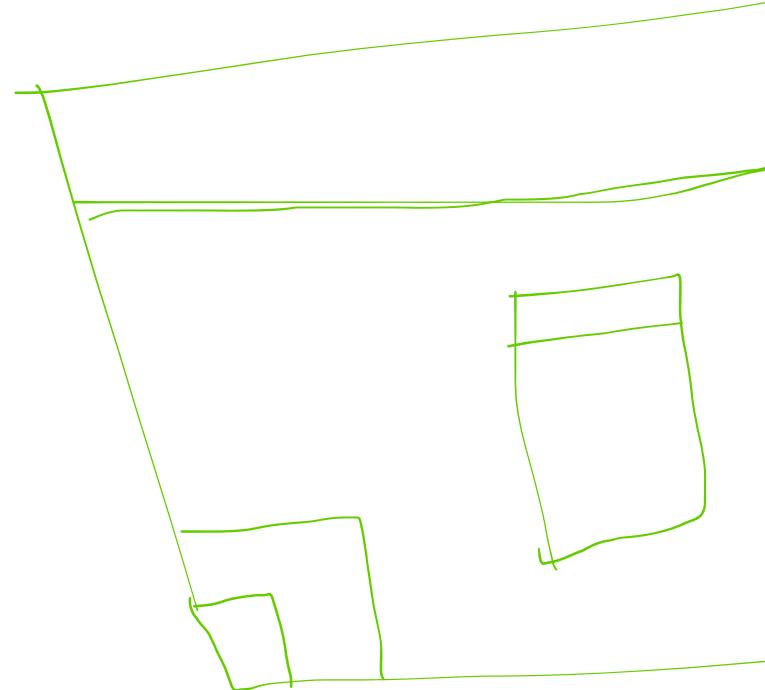
Image files (.jpg, .jpeg, .png, .gif, .svg, .webp): These files are often used across multiple pages and are static in nature.

Font files (.woff, .woff2, .ttf, .otf): Web fonts are generally static and used across various pages.

Video files (.mp4, .webm): Video content that doesn't change frequently.

Audio files (.mp3, .ogg): Like videos, audio files are **static** and usually **don't** change unless updated.

Certain infrequently changing HTML pages (e.g., "About Us," "Contact Us")

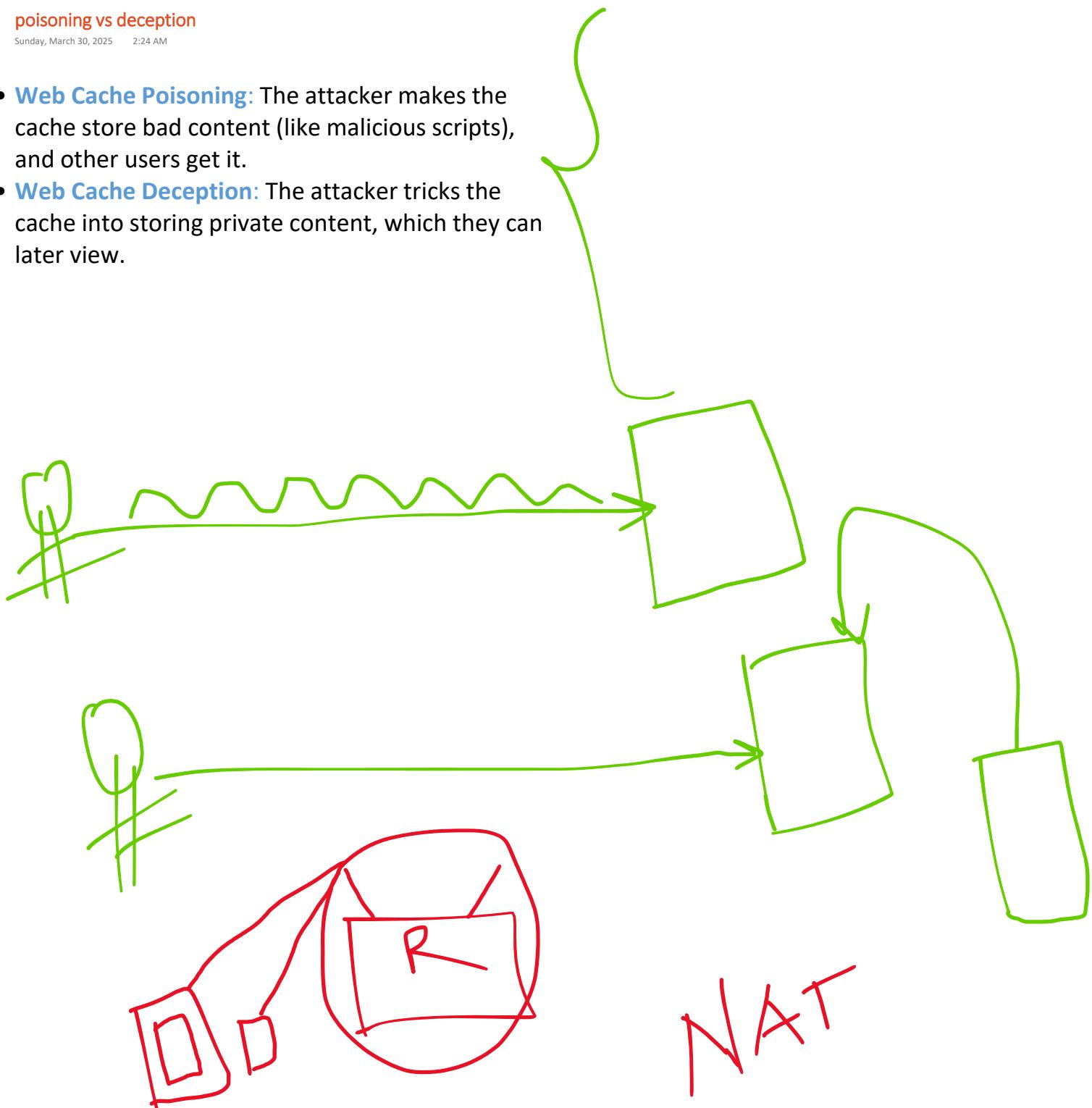


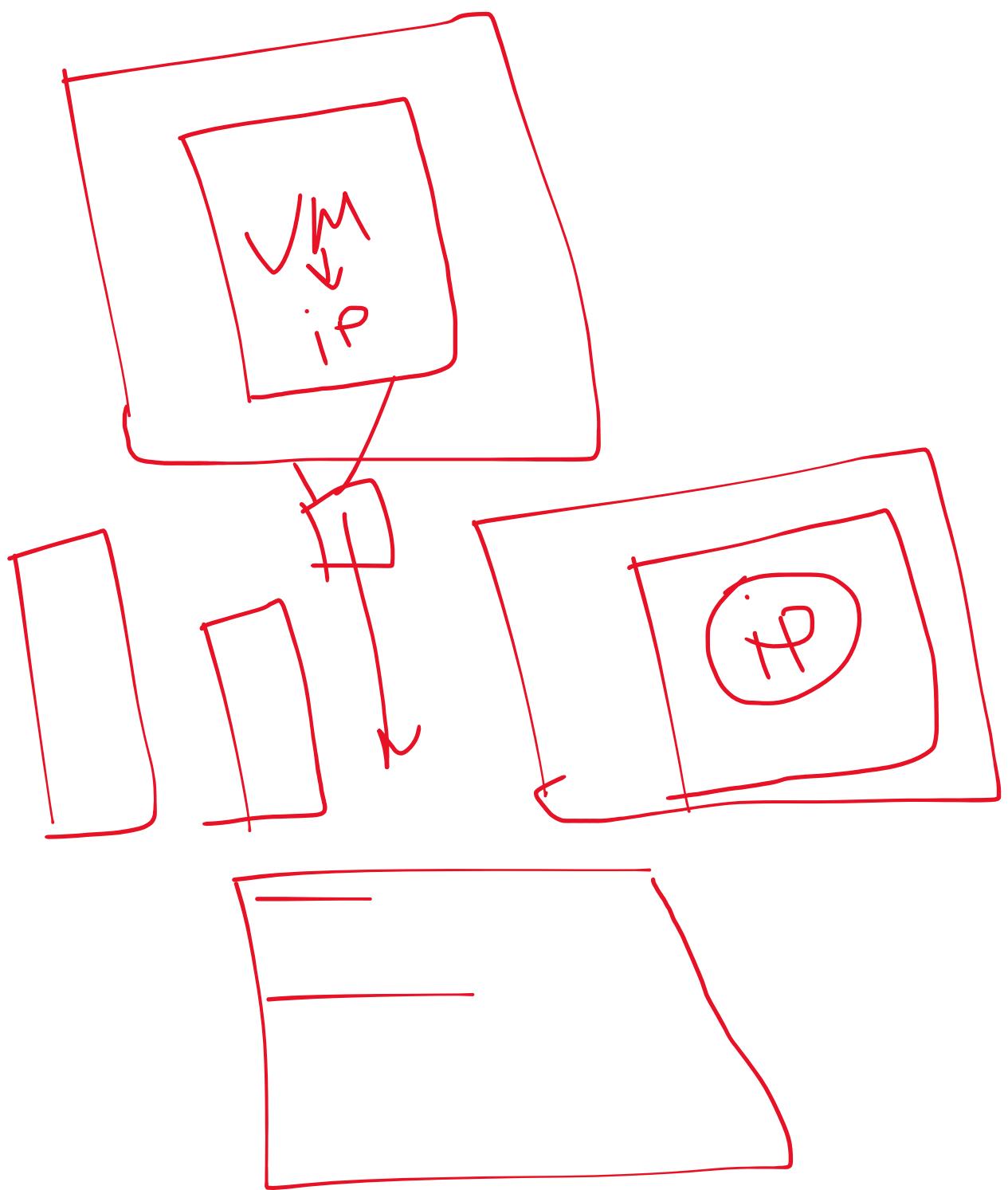


poisoning vs deception

Sunday, March 30, 2025 2:24 AM

- **Web Cache Poisoning:** The attacker makes the cache store bad content (like malicious scripts), and other users get it.
- **Web Cache Deception:** The attacker tricks the cache into storing private content, which they can later view.





Step 1: Identifying a Target Endpoint<https://example.com/user/profile>**Step 2: Identifying a Discrepancy in URL Parsing**

Testing with Burp Suite:

1. Step 1: Identify a Non-Cached Endpoint

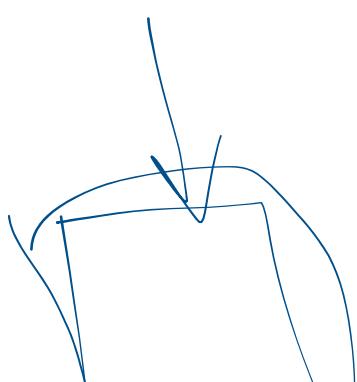
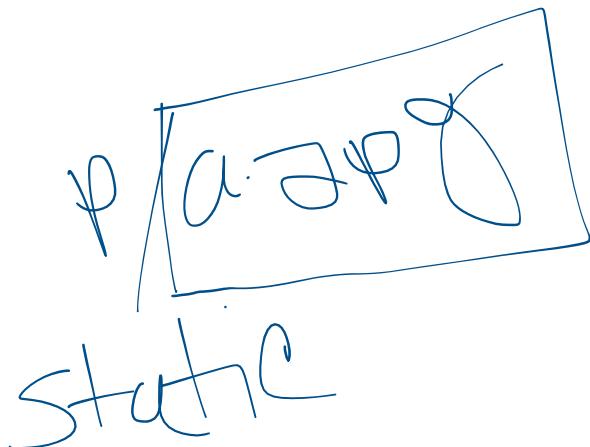
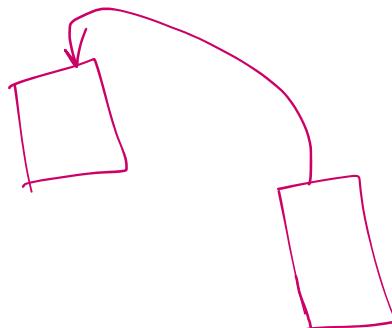
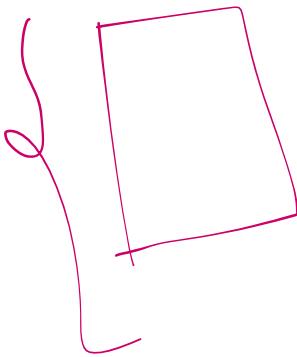
- o Send a request to:
GET /user/profile HTTP/1.1
Host: example.com
 - o Check the Cache-Control header in the response:
Cache-Control: private, no-store
- This means the page is not cached.

2. Step 2: Modify the URL to Trick the Cache

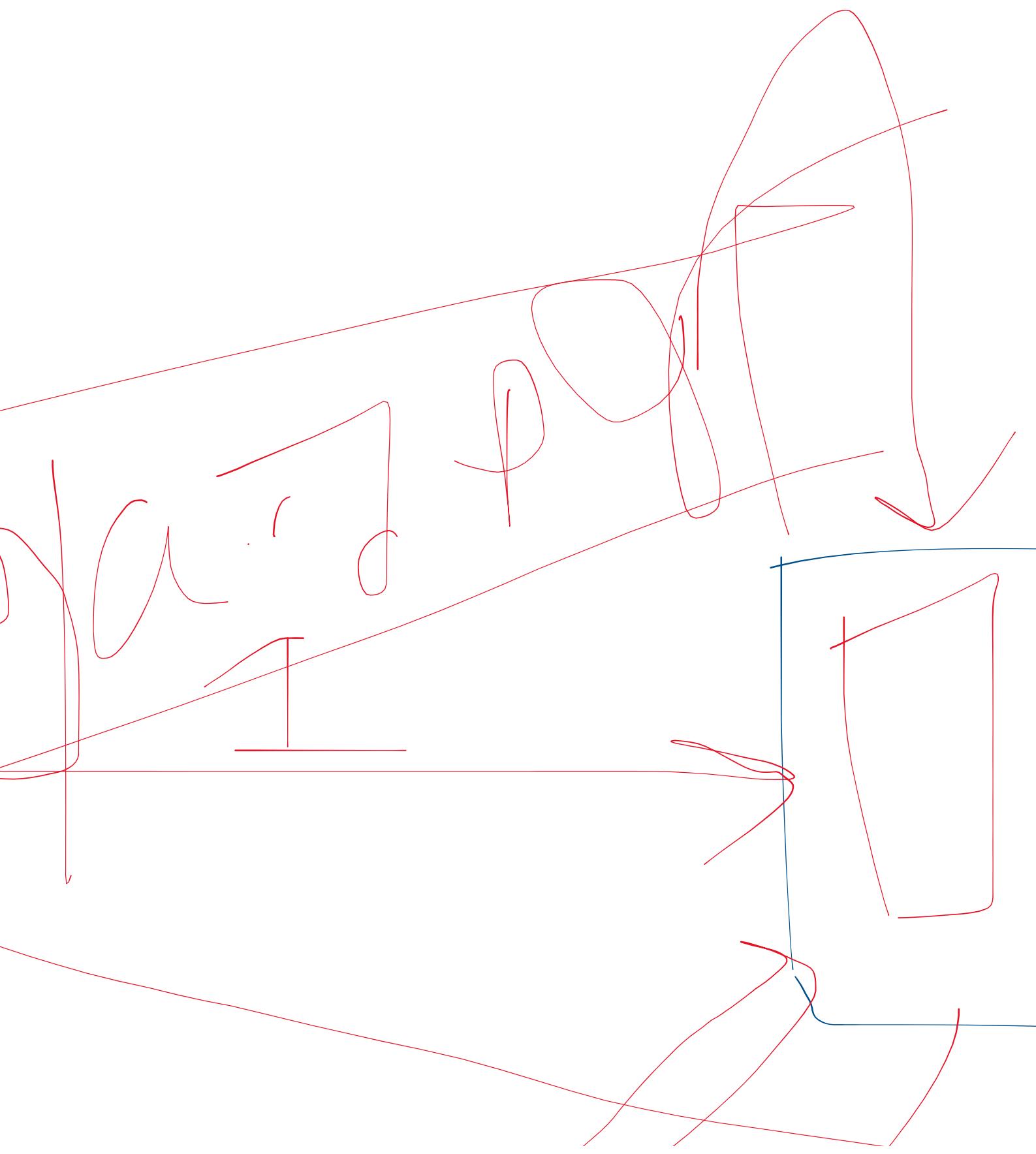
- o Change the request in Burp Suite to:
GET /user/profile/logo.jpg HTTP/1.1
Host: example.com
- o If the server does not validate the extension, it may still return user-specific content.
- o However, the cache sees .jpg and stores it as a static file.

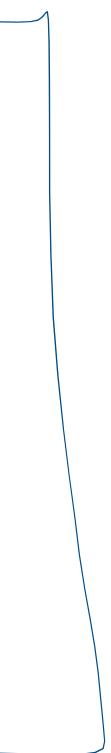
3. Step 3: Verify if the Content is Cached

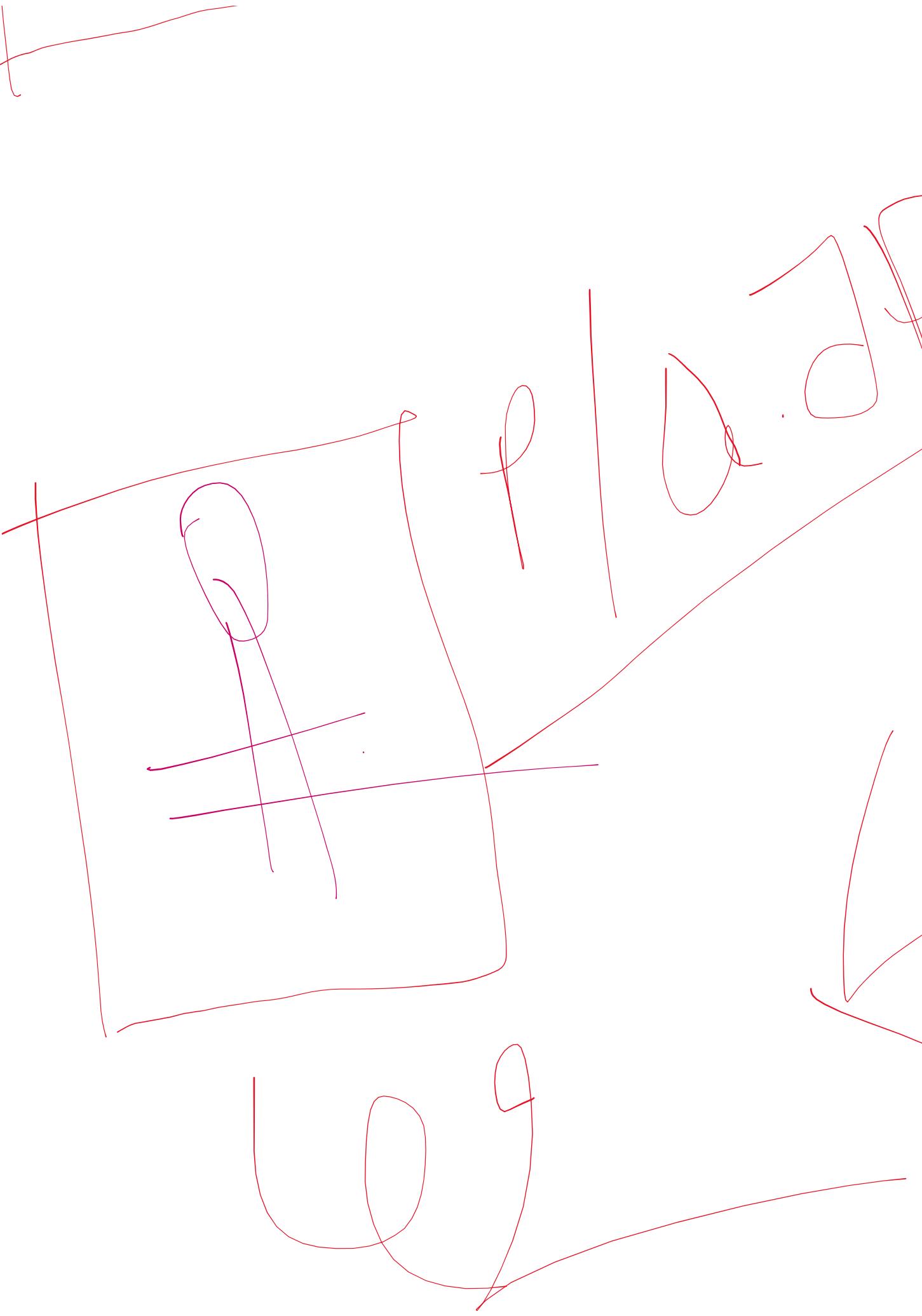
- o Wait for another user to visit
<https://example.com/user/profile/logo.jpg>.
- o The attacker can now send:
GET /user/profile/logo.jpg HTTP/1.1
Host: example.com
- o If the response contains the previous user's profile data, the site is vulnerable.



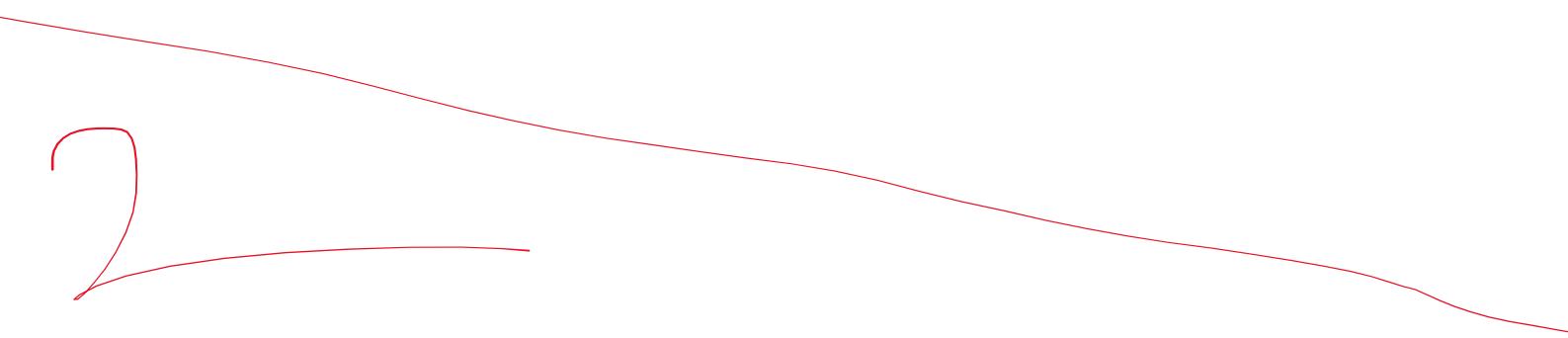


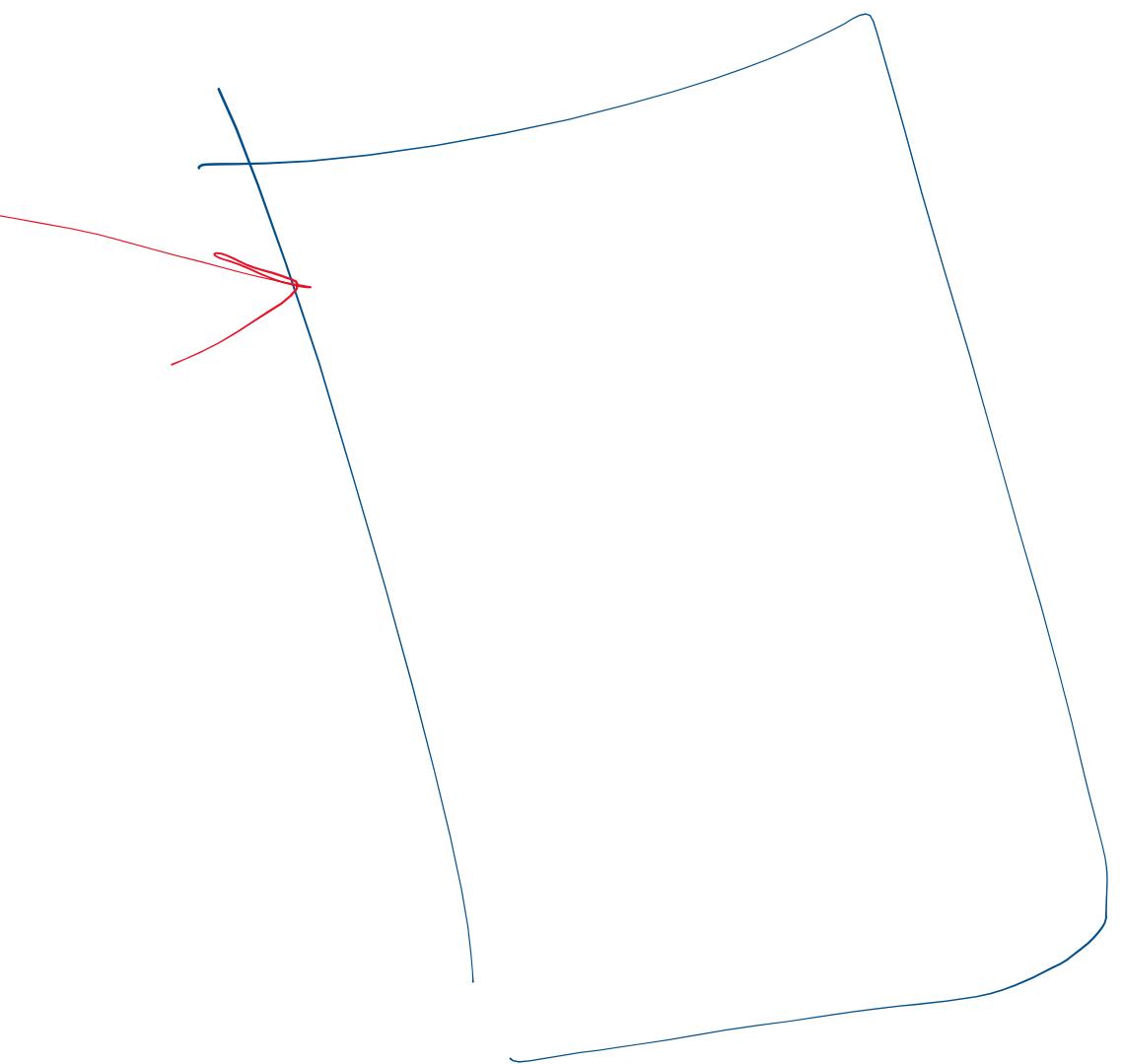


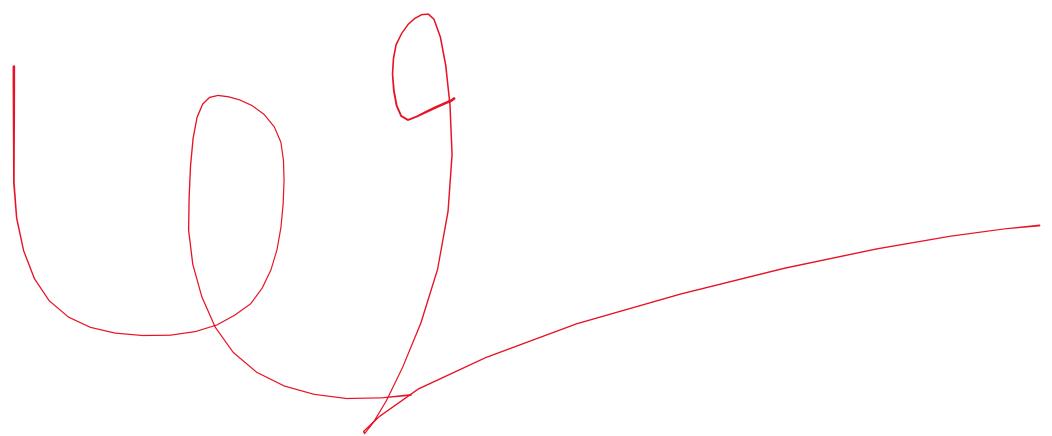


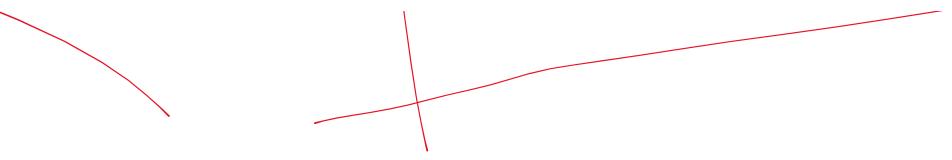














hit vs miss

Sunday, March 30, 2025 2:39 AM

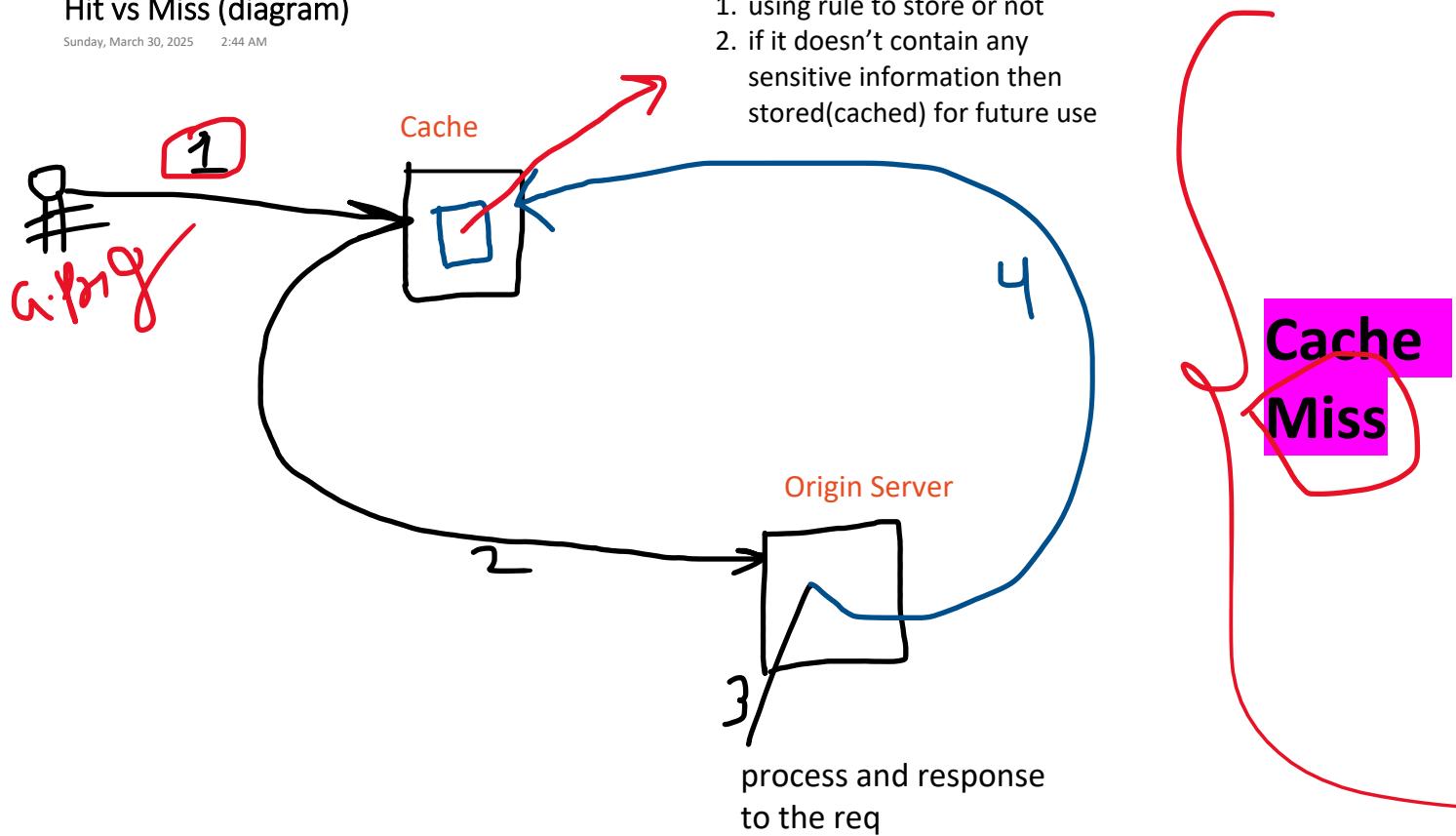
A web cache is a system that sits between the origin server and the user.

When a client requests a static resource, the request is first directed to the cache. If the cache doesn't contain a copy of the resource (known as a cache miss), the request is forwarded to the origin server, which processes and responds to the request. The response is then sent to the cache before being sent to the user. The cache uses a preconfigured set of rules to determine whether to store the response.

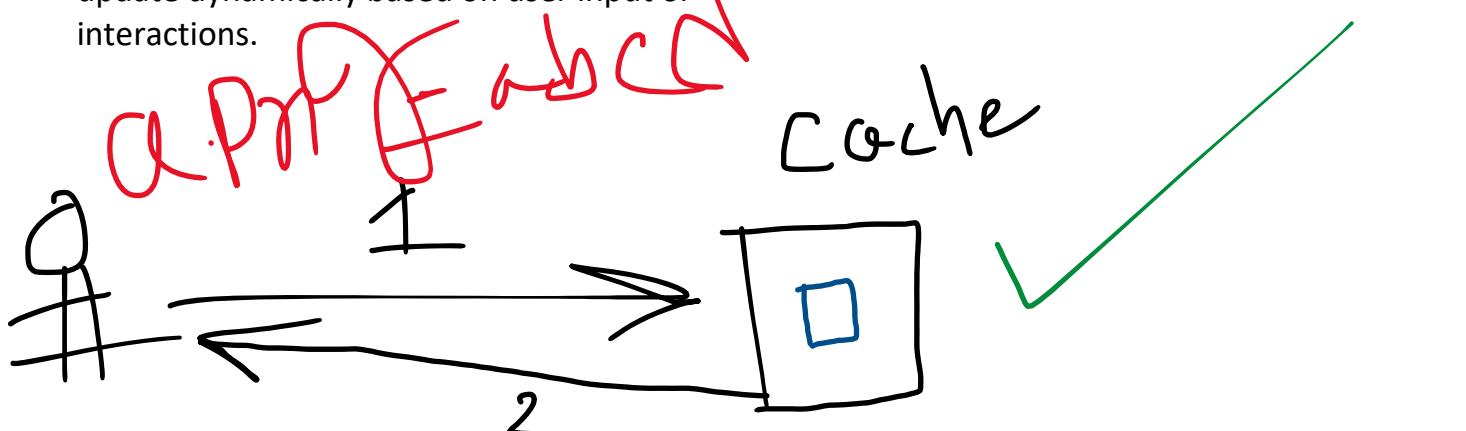
When a request for the same static resource is made in the future, the cache serves the stored copy of the response directly to the user (known as a cache hit).

Hit vs Miss (diagram)

Sunday, March 30, 2025 2:44 AM

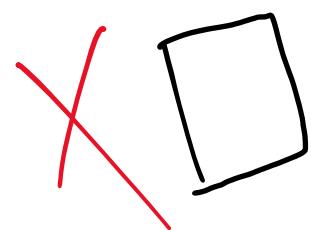


A **static resource** refers to any file or content on a website that doesn't change or update dynamically based on user input or interactions.



Cache
Hit

origin



Cache Keys

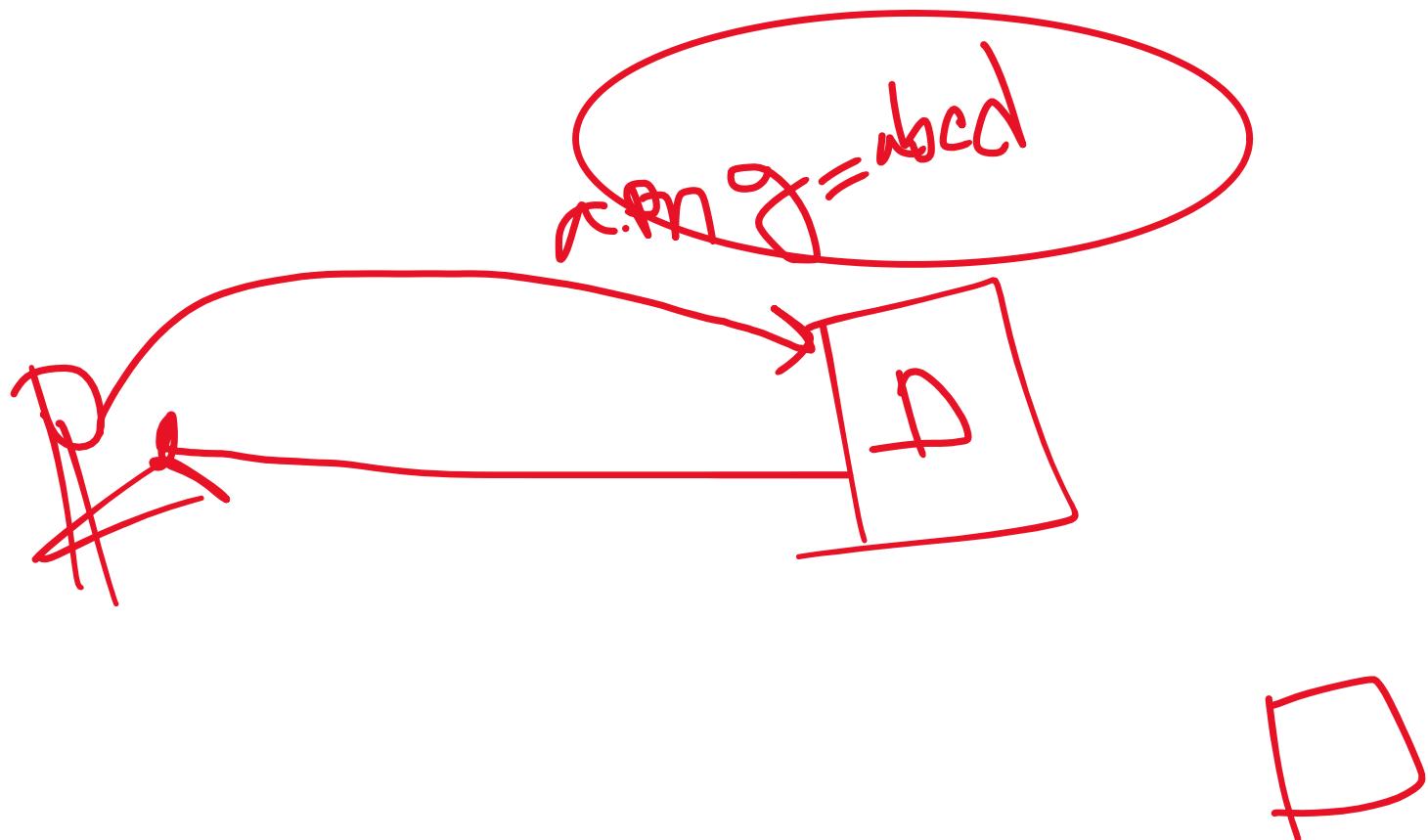
Sunday, March 30, 2025 3:11 AM

When the cache receives an HTTP request,

- it must decide whether there is a cached response that it can serve directly, or
- whether it has to forward the request to the origin server.

The cache makes this decision by generating a '**cache key**' from elements of the HTTP request. Typically, this includes the URL path and query parameters, but it can also include a variety of other elements like headers and content type.

If the incoming request's cache key matches that of a previous request, the cache considers them to be equivalent and serves a copy of the cached response.



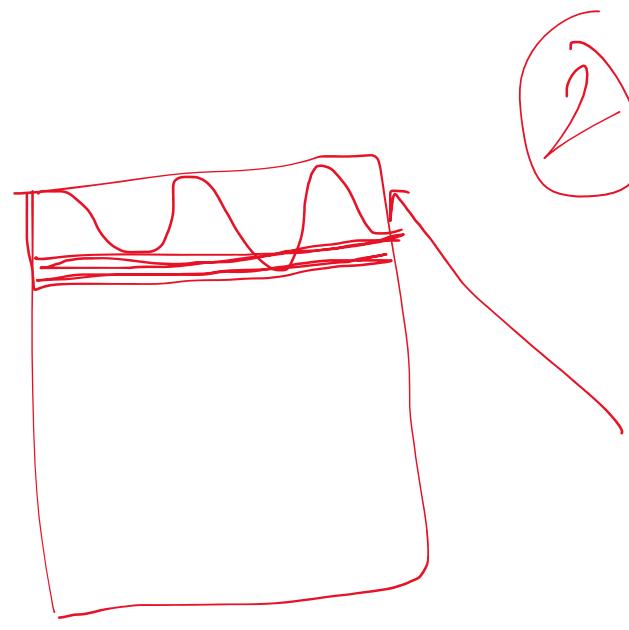
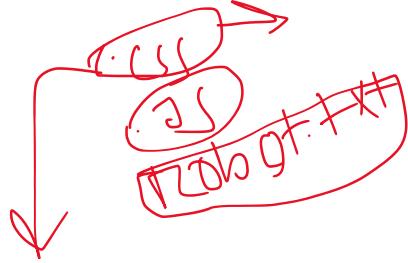
Cache rules

Cache rules determine what can be cached and for how long. Cache rules are often set up to store static resources, which generally don't change frequently and are reused across multiple pages.

Dynamic content is not cached as it's more likely to contain sensitive information, ensuring users get the latest data directly from the server.

Web cache deception attacks exploit how cache rules are applied, so it's important to know about some different types of rules, particularly those based on defined strings in the URL path of the request. For example:

- **Static file extension rules** - These rules match the file extension of the requested resource, for example `.css` for styleheets or `.js` for JavaScript files.
- **Static directory rules** - Static directory rules are rules that apply to all files inside a specific folder, usually containing unchanging (static) resources like images, CSS, or JavaScript files.
- **File name rules** - These rules match specific file names to target files that are universally required for web operations and change rarely, such as `robots.txt` and `favicon.ico`.



)

1 Find a Target Page

🔍 **Goal:** Look for a page that shows **private user data** (like account info, messages, or settings).

🛠️ **Tool:** Use **Burp Suite** to inspect HTTP responses for sensitive information.

✓ Example:

- A website has a user profile page:

👉 <https://example.com/user/profile>

- This page contains **private** user details like name, email, and balance.

2 Find a Cache Weakness

🔍 **Goal:** Check if the website and cache **treat URLs differently**.

🛠️ **How?** Look for inconsistencies in:

- **Path handling** (/profile vs. /profile/)
- **File extensions** (/profile vs. /profile.jpg)
- **Special characters** (/profile?nocache=1)

✓ Example:

- The server only serves /user/profile dynamically (per user).
- But if you add .jpg at the end (/user/profile.jpg), the **cache thinks it's a static file** and stores it!

3 Trick the Cache

🔍 **Goal: Modify the URL** so the cache stores sensitive data as a public file.

✓ Example:

1. Original dynamic page:

👉 <https://example.com/user/profile> (private)

2. Tricked URL:

👉 <https://example.com/user/profile.jpg>

3. If the caching system **treats it as a static file**, it **stores** the response

(which contains user info).

4 Steal the Cached Data

🔍 **Goal:** Wait for another user to access the **malicious URL**, causing their private data to be cached.

🛠 **Tool:** Use Burp Suite to fetch the cached version of the URL.

✓ Example:

- Victim logs in and visits: <https://example.com/user/profile>
- Cache mistakenly **stores** their profile info under /user/profile.jpg
- Attacker requests /user/profile.jpg and **sees the victim's private data!**

Why Use a Cache Buster?

When testing for **cache weaknesses**, you must ensure that each request gets a fresh response. If the server **caches** previous responses, your tests might not work correctly.

How to Use a Cache Buster?

1. **Modify the URL** by adding a unique query string every time you send a request.

Example:

- First request: <https://example.com/user/profile?nocache=1234>
- Second request: <https://example.com/user/profile?nocache=5678>

2. **Automate with Burp Suite's Param Miner**

- Install **Param Miner** extension.
- Go to **Param Miner > Settings > Add dynamic cache buster** in Burp.
- Burp now **automatically** adds a unique query parameter to every request.
- Check the added query strings in the **Logger** tab.

1 Check Response Headers

Look at **HTTP response headers** to see if the response came from a cache.

Common Headers:

- **X-Cache: hit** → Response served from cache.
- **X-Cache: miss** → Response **not in cache**, so it was fetched from the server.
- **X-Cache: dynamic** → Response generated dynamically, not cached.
- **X-Cache: refresh** → means that the server checked (revalidated) the cached content to ensure it's still up-to-date before serving it.

Instead of fetching new data, it confirmed the cached version was still valid.

Example Response:

HTTP/1.1 200 OK

X-Cache: hit

Cache-Control: public, max-age=3600

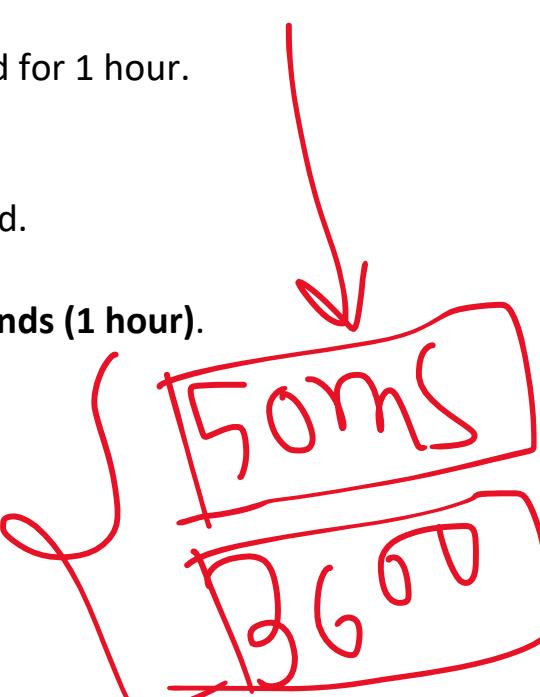
- ◊ This means the response **was cached** and will be valid for 1 hour.

2 Check Cache-Control Header

The **Cache-Control** header shows how caching is managed.

What to Look For?

- **public, max-age=3600** → Can be cached for **3600 seconds (1 hour)**.
- **private** → Cached **only for the user**, not shared.
- **no-cache** → Must be **revalidated** before serving.
- **no-store** → **Not cached** at all.



Example:

Cache-Control: public, max-age=86400

- ◊ The page can be **cached for 24 hours**.

3 Compare Response Times

 Trick: A **cached response** is usually **faster** than a fresh response.

How to Test?

1. Send a request and note the **response time** (e.g., 500ms).
2. Send the **same request again**.
 - If response time **drops significantly** (e.g., 50ms), it's likely **cached**.
 - If the response time **stays the same**, it's likely **not cached**.

Example:

- First request: **500ms** (Fetched from origin)
- Second request: **50ms** (Served from cache)

Step 1: Open Burp Suite and Intercept a Request

1. Open **Burp Suite** and ensure **Intercept** is **ON**.
2. Browse to a target website (**e.g.**, <https://example.com/profile>).
3. Capture the request in **Burp Proxy**.
4. Send the request to **Repeater** (Right-click → **Send to Repeater**).

Step 2: Analyze the First Response

1. In **Repeater**, click **Send**.
2. Look at the **Response Headers**.

Example Response (First Request):

HTTP/1.1 200 OK

Date: Sun, 30 Mar 2025 12:00:00 GMT

Content-Type: text/html; charset=UTF-8

X-Cache: MISS

Cache-Control: public, max-age=3600

- ◊ **X-Cache: MISS** → The cache **did not contain** the response, so it was fetched from the origin server.
- ◊ **Cache-Control: public, max-age=3600** → This tells us the response is **cacheable** for 1 hour.

Step 3: Send the Same Request Again

1. Click **Send** again in **Repeater**.
2. Look at the new response headers.

Example Response (Second Request):

yaml

CopyEdit

HTTP/1.1 200 OK

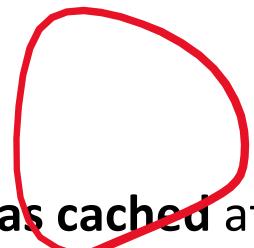
Date: Sun, 30 Mar 2025 12:00:10 GMT

Content-Type: text/html; charset=UTF-8

X-Cache: HIT

Cache-Control: public, max-age=3600

- ◊ **X-Cache: HIT** → This means the response was cached after the first request and is now being served from the cache.
- ◊ **Response time is now much lower** → Indicates the request was **served from cache** instead of hitting the origin server.



🛠 Step 4: Compare Response Times

You can compare the **response time** from the first and second requests:

- **First request:** 450ms (Fetched from origin)
- **Second request:** 50ms (Served from cache)

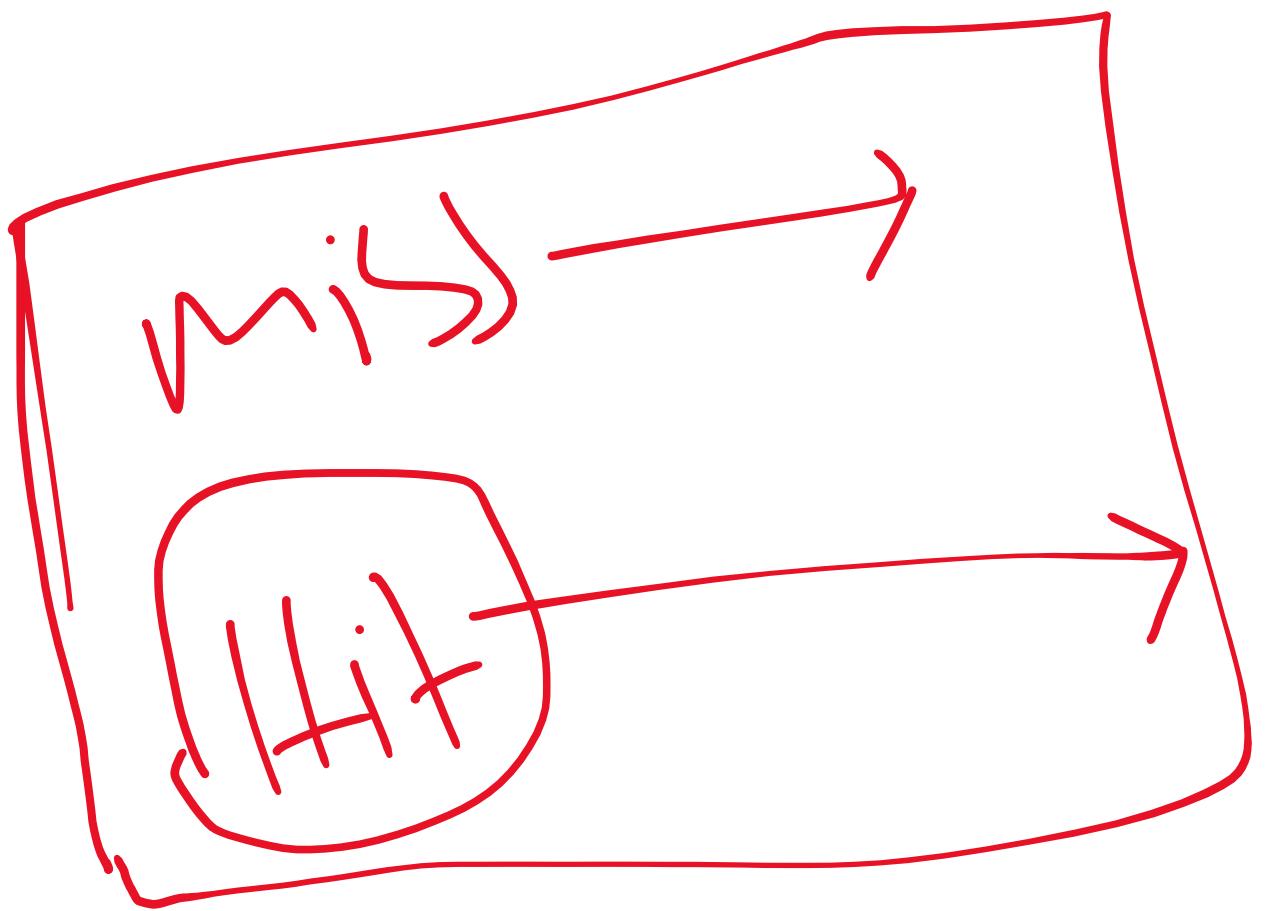
If the second response is **faster** and X-Cache: HIT, then caching is working.

💡 Bonus: Automate Cache Testing with Burp Extensions

To test caching behavior automatically, you can use:

- Param Miner** (to add cache-busting query parameters)
- Turbo Intruder** (to send multiple requests rapidly)

Would you like help setting up an **automated script** for this in Burp?



💡 Concept Summary

- Caches **store static files** (like `.css, .js, .jpg`) to speed up websites.
- **Dynamic pages** (like `/profile`) should **not** be cached.
- If a website **mismatches how URLs are processed**, an attacker can **trick the cache** into storing private data.

🛠️ How Attackers Exploit This

Step 1: Find a Sensitive Page

- Example:
[`https://example.com/user/settings`](https://example.com/user/settings)
 - Shows **personal details** of a logged-in user.
 - Normally **not cached** because it's dynamic.

Step 2: Add a Fake Static Extension

- Example:

ruby

CopyEdit

[`https://example.com/user/settings/style.css`](https://example.com/user/settings/style.css)

- The **server** still gives the **user's settings page**.
- The **cache sees .css and stores it** (thinking it's static).

Step 3: Steal Cached Data

- Later, the attacker visits:

[`https://example.com/user/settings/style.css`](https://example.com/user/settings/style.css)

- The cache **serves the stored response**.
- If the previous user visited this page, their **settings are now visible** to the attacker!

💧 Simple Example

1. Regular Request:

`GET /user/settings HTTP/1.1`

Host: example.com

- **Cache-Control:** private, no-store (Not cached).

2. Modified Request:

GET /user/settings/style.css HTTP/1.1

Host: example.com

- If the **cache stores this response**, it's a **security risk**.

3. Attacker Retrieves Cached Data:

GET /user/settings/style.css HTTP/1.1

Host: example.com

- If the attacker sees **another user's settings**, the attack worked!

Path mapping discrepancies

Sunday, March 30, 2025 9:20 PM

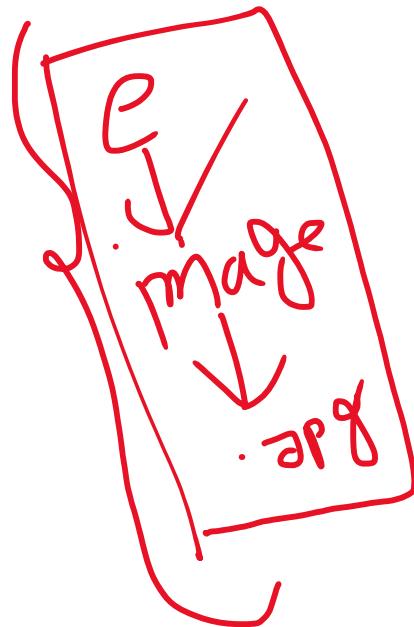
Two Types of URL Mapping:

1. Traditional URL Mapping:

- This is when the URL directly maps to a **file on the server**.
- Example: <http://example.com/images/logo.jpg>
 - The server looks for the file logo.jpg in the images folder and shows it to you.

2. RESTful URL Mapping:

- This is when the URL **doesn't directly map to a file** but instead points to a **resource** on the server.
- Example: <http://example.com/user/123/profile>
 - This is asking the server to show **user 123's profile**. The server generates the profile dynamically based on the user ID (**it's not a file, but dynamic content**).



Discrepancy Example (How Cache and Server Can Interpret URLs Differently):

Imagine a URL: <http://example.com/user/123/profile/wcd.css>

- **Server's Role:**

The server sees this URL and thinks, "This is a request for **user 123's profile**." The server will **ignore the .css part** because it's just a part of the URL, and it will return **the profile information** for user 123.

- **Cache's Role:**

The cache sees the URL and thinks, "This URL ends with .css, so it must be a CSS file." The cache assumes it's a request for a **CSS file** and tries to **store the profile data as a static file**. This can cause **problems** because the profile data is **cached incorrectly**.

What Happens Next?

- **What an Attacker Can Do:**

The attacker can send a link like

<http://example.com/user/123/profile/wcd.css> to another user. When

the second user visits this URL:

- The cache will serve the **profile page** of user 123 (thinking it's a static CSS file).
- The second user **sees someone else's profile**, which is a **privacy issue**.

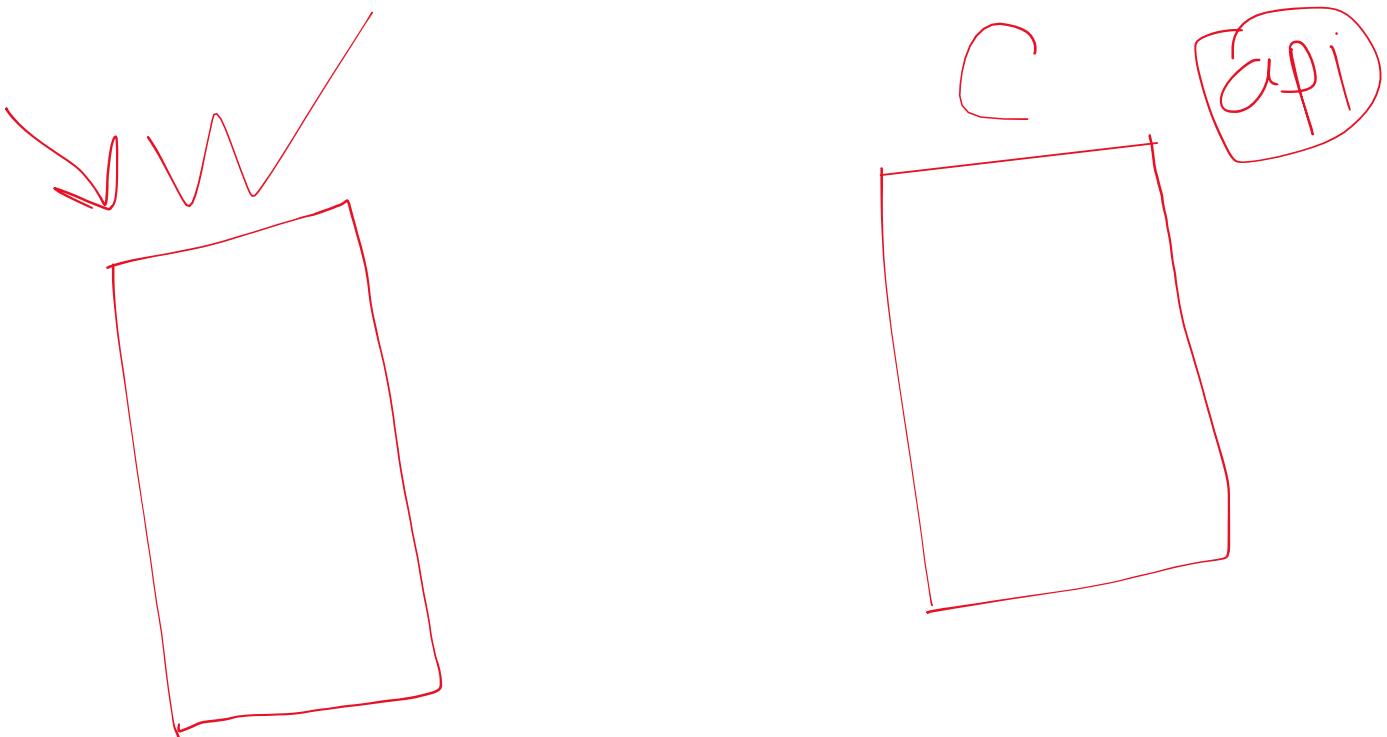
extension list:

.js
.CSS

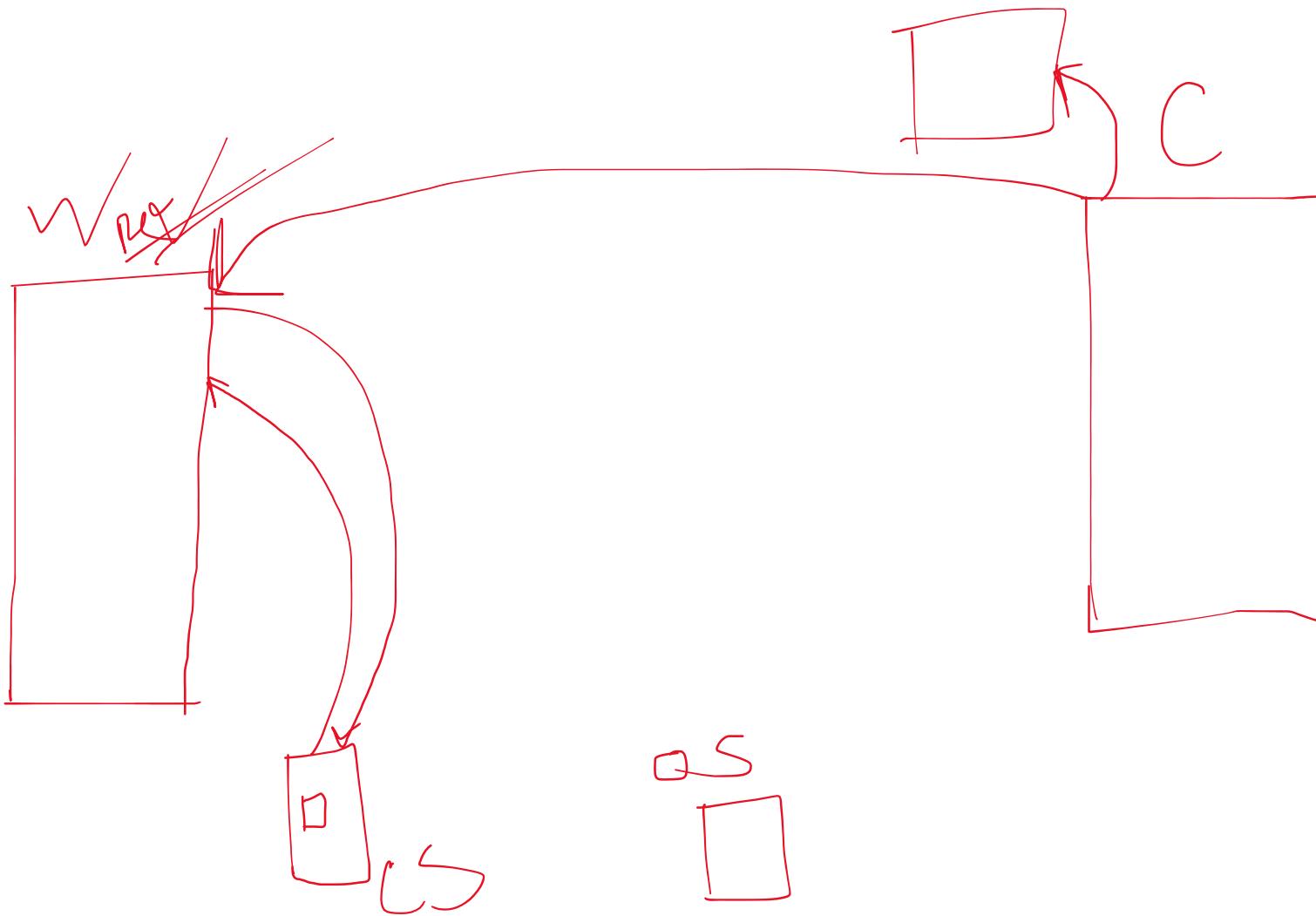
.ico
.png
.jpg
.gif
.svg
.webp
.woff
.woff2
.ttf
.eot
.mp3
.mp4
.wav
.ogg
.exe
.sh
.bat
.cmd
.php
.asp
.aspx
.jsp
.json
.xml
.csv
.log
.txt
.zip
.tar
.gz
.swf

Carlos-এর API কী আমাদের কাছে এল কিভাবে?

- Carlos যখন /my-account/wcd.js ইন্ট করল, তার API কী সহ রেসপন্সটি ক্যাশ হয়ে গেল।
- এরপর আমরা যখন একই URL-টি ইন্ট করলাম, ক্যাশ আমাদের সেই একই রেসপন্স ফেরত দিল—যেটা হিলে Carlos-এর API কী।
- অর্থাৎ, সার্ভার বুবাতেই পারেনি যে এটি একটি সেনসিটিভ রিসোর্স এবং আলাদা ইউজারের জন্য আলাদা হওয়া উচিত ছিল।





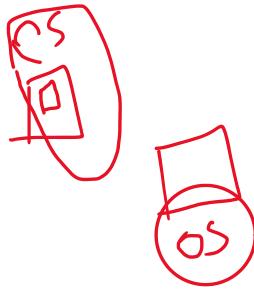




Delimiter Discrepancies

Thursday, April 3, 2025 9:23 PM

- Delimiter = বিভাজক বা সীমারেখা নির্ধারণকারী চিহ্ন।
 - এটি কোনো কিছু আলাদা করার জন্য ব্যবহৃত হয়, যেমন
 - > ;
 - > /
 - > ?
 - > .



- Discrepancies = অমিল বা পার্থক্য।
 - যখন দুই জায়গায় একই জিনিস ভিন্নভাবে ব্যাখ্যা করা হয়, তখন তাকে discrepancy বলা হয়।



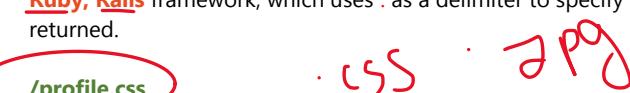
ফলাফল:

Java Spring; কে Matrix Variable মনে করে এবং পরের অংশ বাদ দিয়ে দেয়।

কিন্তু Web Cache; সহ পুরো URL কে সংরক্ষণ করতে পারে।

- সার্ভার: /profile;style.css কে /profile হিসেবে পড়ে।
- ক্যাশ: /profile;style.css কে আলাদা CSS ফাইল হিসেবে ধরে এবং ক্যাশ করে।

Ruby, Rails framework, which uses . as a delimiter to specify the response format. There isn't a CSS formatter, so the request isn't accepted and an error is returned.

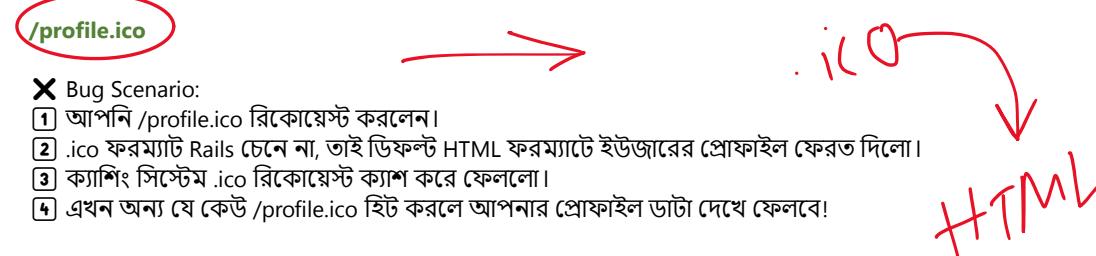


তবে, Rails-এর কোনো CSS ফরম্যাট নেই, তাই এটি বুঝতে পারে না কিভাবে /profile রুটকে CSS হিসেবে রিটুর্ন করতে হবে।

ফলাফল:

সার্ভার এই রিকোয়েস্ট গ্রহণ করবে না।

পরিবর্তে এটি একটি ত্রুটি (error) ফেরত পাঠাবে, যেমন 404 Not Found বা 406 Not Acceptable।



X Bug Scenario:

- আপনি /profile.ico রিকোয়েস্ট করলেন।
- .ico ফরম্যাট Rails চেনে না, তাই ডিফল্ট HTML ফরম্যাটে ইউজারের প্রোফাইল ফেললো।
- ক্যাশিং সিস্টেম .ico রিকোয়েস্ট ক্যাশ করে ফেললো।
- এখন অন্য যে কেউ /profile.ico হিট করলে আপনার প্রোফাইল ডাটা দেখে ফেলবে!

Encoded characters may also sometimes be used as delimiters.

/profile%00foo.js

OpenLiteSpeed Server

- Scenario: আপনি /profile%00foo.js রিকোয়েস্ট করলেন।
- OpenLiteSpeed কীভাবে এটিকে ব্যাখ্যা করবে?

%00 হলো Null Character, যা OpenLiteSpeed ডিলিমিটার হিসেবে ধরে।

তাই, সার্ভার শুধুমাত্র /profile হিসেবে রিকোয়েস্ট প্রসেস করবে।

Akamai বা Fastly

- বেশিরভাগ ওয়েব ফ্রেমওয়ার্ক (Laravel, Django, Express.js) ইত্যাদি

%00 (Null Byte) থাকলে রিকোয়েস্ট রিজেক্ট করবে এবং Error দিবে (যেমন: 400 Bad Request বা 404 Not Found)।

কারণ, এটি একটি অস্বাভাবিক বা ম্যালিসিয়াস ইনপুট।

- কিন্তু যদি Akamai বা Fastly ক্যাশিং ব্যবহার করা হয়

তাহলে %00 কে পাথের অংশ হিসেবে ব্যাখ্যা করবে।

অর্থাৎ /profile%00foo.js রিকোয়েস্ট দিলে, এটি পুরো URL (/profile%00foo.js) ক্যাশে স্টোর করবে।

ফলে, ভুল ডাটা ক্যাশ হয়ে গিয়ে Cache Poisoning হতে পারে!



- Akamai (আকামেই)
- ⦿ সবচেয়ে পুরাতন ও বড় CDN প্রদানকারী
- ⦿ এন্টারপ্রাইজ-লেভেল নিরাপত্তা (WAF, DDoS Protection, Bot Management)
- ⦿ গ্লোবাল কভারেজ (Edge Computing, Cloud Security)
- ⦿ বড় বড় প্রতিষ্ঠান ব্যবহার করে (Microsoft, Apple, Amazon)

☞ মূলত এন্টারপ্রাইজ গ্রেড নিরাপত্তা ও পারফরম্যান্স অপ্টিমাইজেশনের জন্য জনপ্রিয়।

- Fastly (ফাস্টলি)
- ⦿ লো-লেটেন্সি (Low Latency) ও ফাস্ট ক্যাশিং ফোকাসড CDN
- ⦿ Real-time লেভেলে API-driven কনফিগারেশন
- ⦿ Edge Computing (ফাস্ট ও আর্ট ডাটা প্রসেসিং)
- ⦿ Cloudflare-এর প্রতিদ্বন্দ্বী হিসেবে পরিচিত

☞ Fastly মূলত ডেভেলপারদের জন্য বেশি ফ্রেক্সিবল এবং API-কেন্দ্রিক সলিউশন দেয়।

example

Friday, April 4, 2025 3:27 AM

Real-Life Example (Bug Hunting):
Initial URL: /settings/users/list



Add arbitrary string: /settings/users/listaaa

সার্ভার যদি একই প্রতিক্রিয়া দেয়, তাহলে এটা আমাদের বেস রেফারেন্স হয়ে যাবে।

Add possible delimiter: /settings/users/list;aaa



যদি সার্ভার একই রকম প্রতিক্রিয়া দেয়, তবে ; হলো ডেলিমিটার।

যদি সার্ভার আলাদা প্রতিক্রিয়া দেয়, তবে ; চারিত্ব সার্ভার দ্বারা ডেলিমিটার হিসেবে ব্যবহৃত হচ্ছে না।

Add static extension: /settings/users/list;aaa.jpg

যদি ক্যাশ .jpg এক্সটেনশনটিকে গ্রহণ করে কিন্তু সার্ভারে এটি না থাকে, তবে আপনি এই দুর্বলতা ব্যবহার করতে পারেন।

➄ Real-Life Example

আপনার টাগেটি সাইটের পাথ ধরুন:

↗ <https://example.com/settings/users/list>

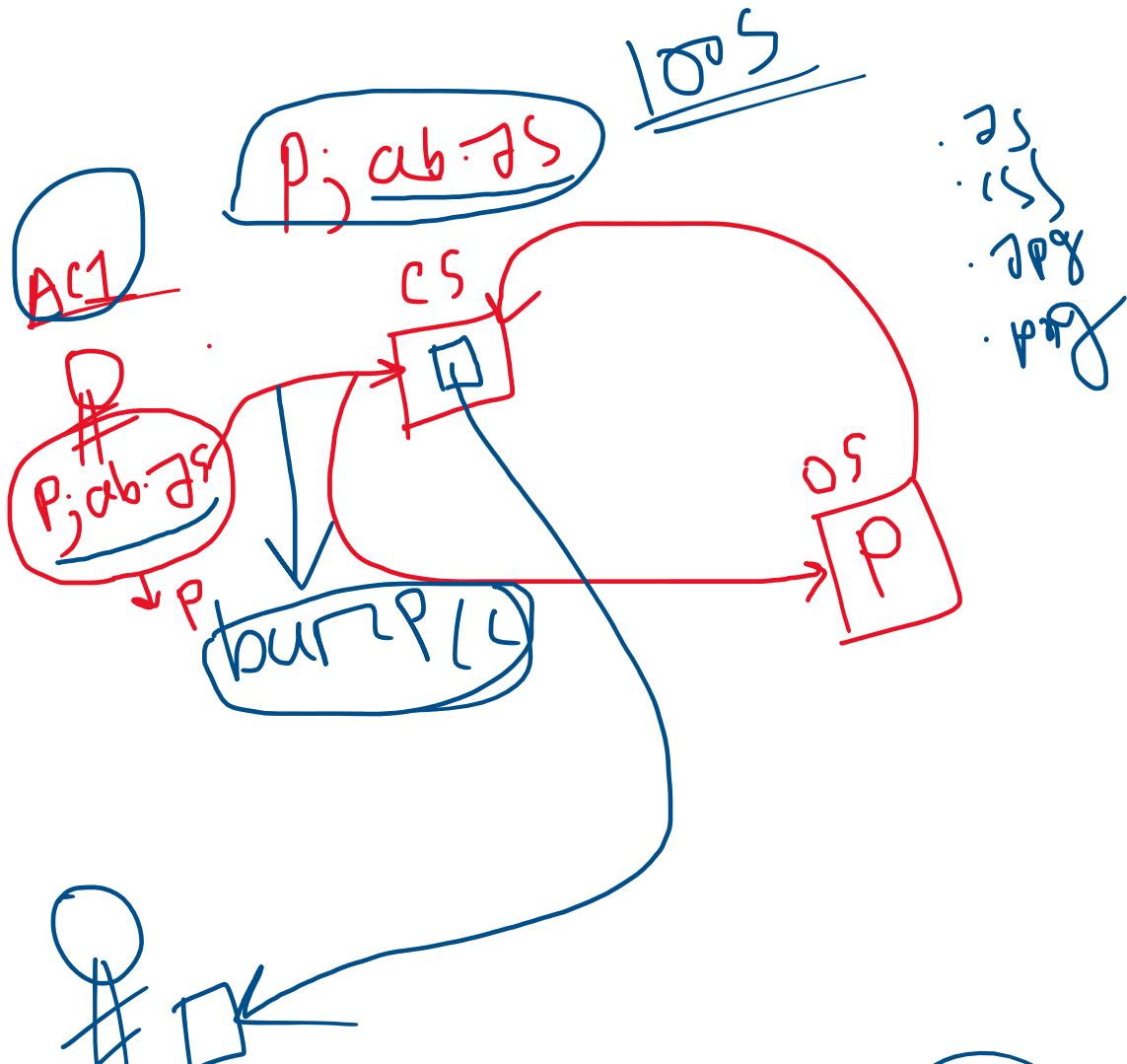
আপনি এই URL টেস্ট করলেন:

↗ <https://example.com/settings/users/list:hacked.js>

Possible Outcomes:

ক্যাশ যদি ; কে ডেলিমিটার হিসেবে না ধরে, তাহলে এটি ক্যাশ হয়ে যেতে পারে।

যে কেউ <https://example.com/settings/users/list:hacked.js> ভিজিট করলে, সে ডায়নামিক ইউজার প্রোফাইল ডেটা পেয়ে যেতে পারে!



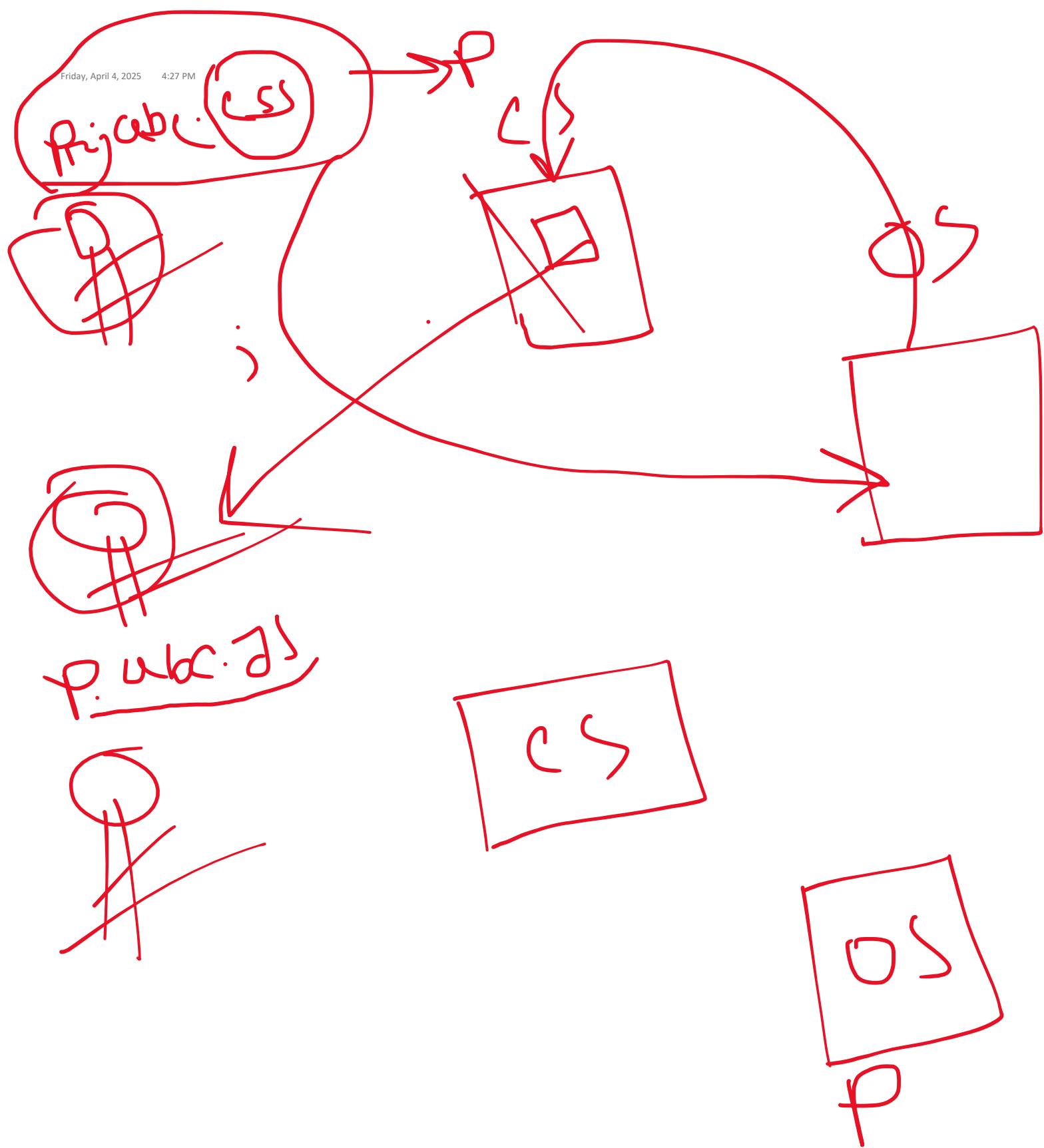
DK
AC2

WC

AC1

CL

AC2



This indicates that the origin server uses ; and ? as path delimiters.

✍ এখন যদি origin server ; বা ? কে path delimiter হিসেবে ধরে? ধরি আমরা নিচের মতো একটা রিকুয়েস্ট পাঠাই:

GET /my-account;admin=true HTTP/1.1

Origin server বুঝে নেয়: ওহ, এটা /my-account এরই request, সাথে একটা extra flag admin=true

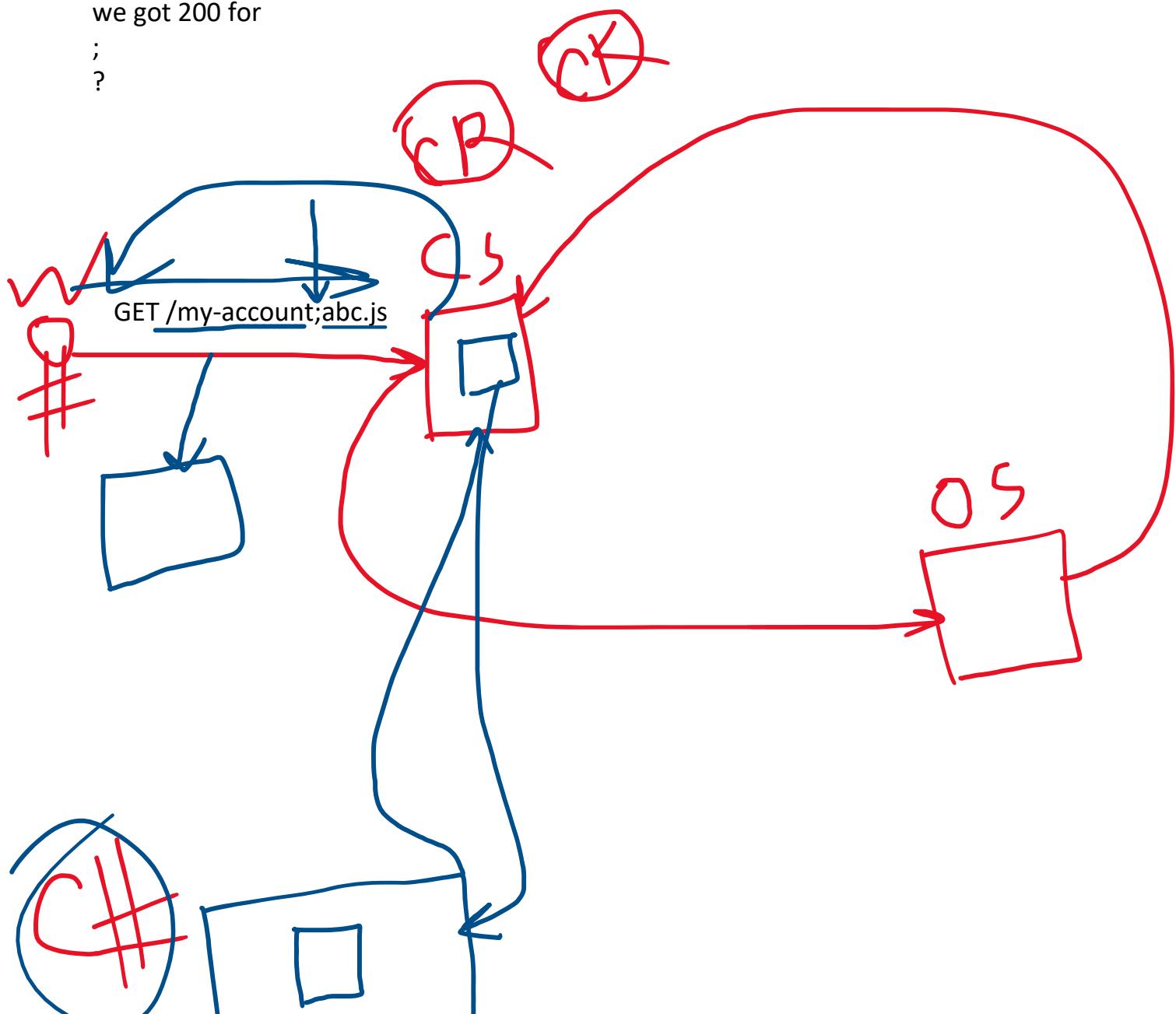
✗ কিন্তু cache ভাবে: এটা /my-account;admin=true নামে আলাদা একটা পেইজ

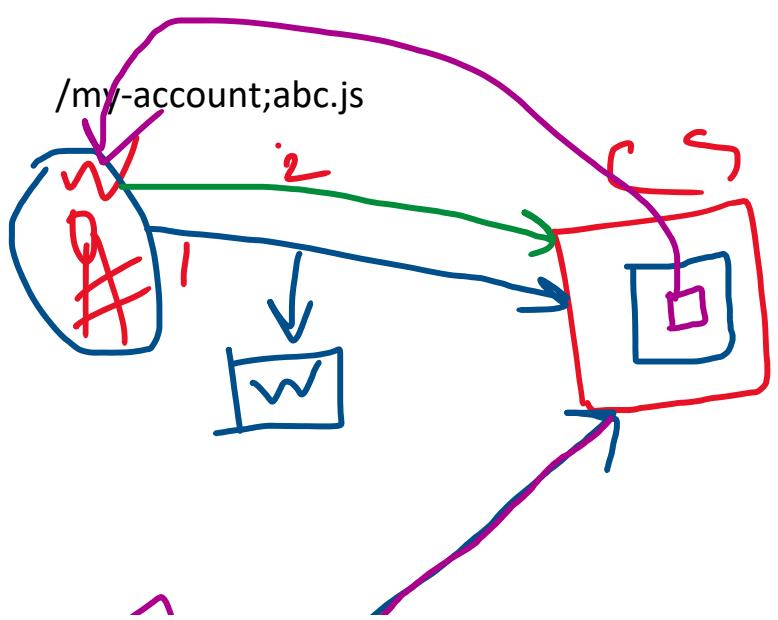
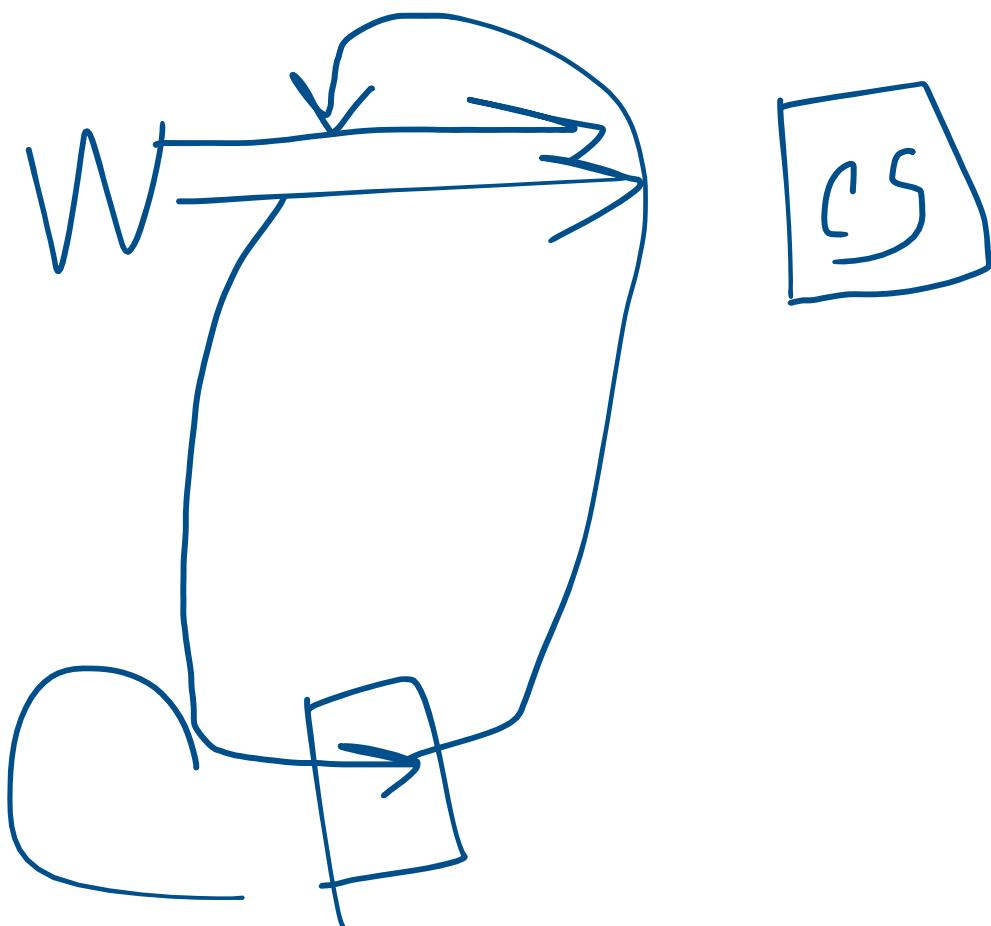
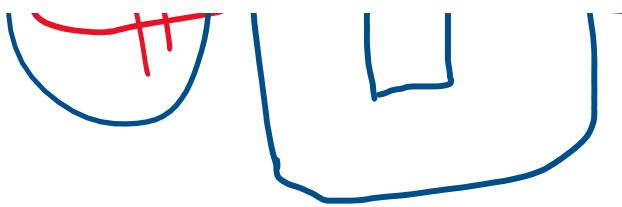
তাহলে দুইজনের বোঝার মধ্যে mismatch হয়ে গেল।

we got 200 for

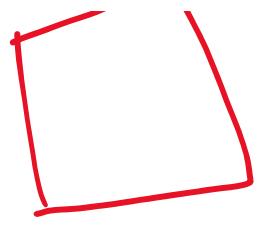
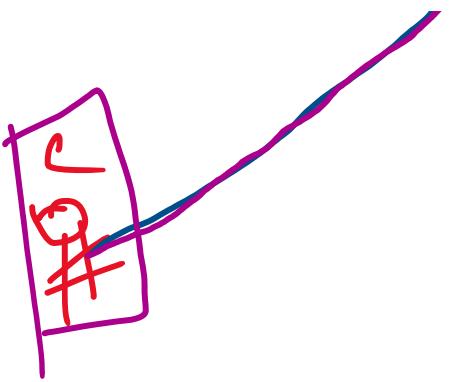
;

?





OS



Delimiter Decoding Discrepancies

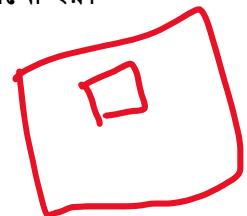
Sunday, April 6, 2025 6:13 PM

অনেক সময় ওয়েবসাইটগুলো URL-এর মধ্যে এমন কিছু তথ্য পাঠায়, যেখানে কিছু বিশেষ চিহ্ন (যেমন: ?, # ইত্যাদি) থাকে।

এই বিশেষ ক্যারেক্টারগুলো যেন ডেটা হিসেবে বিবেচিত হয়, তাই সাধারণত সেগুলো encode করে পাঠানো হয়।
(উদাহরণ: # → %23, ? → %3f)

- Example:
◦ প্রথম URL:
/profile?logout
◦ দ্বিতীয় URL:
/profile%3Flogout

? → .1.3F → ?



ক'বল দেখতে অনেকটা একরকম মনে হলেও, cache system এগুলাকে আলাদা URL ধরে, কারণ:

cache system সাধারণত URL decode করে না!

অর্থাৎ %3F ওর কাছে %3F ই থাকে, ? না।

তাই Cache System ভাবে:

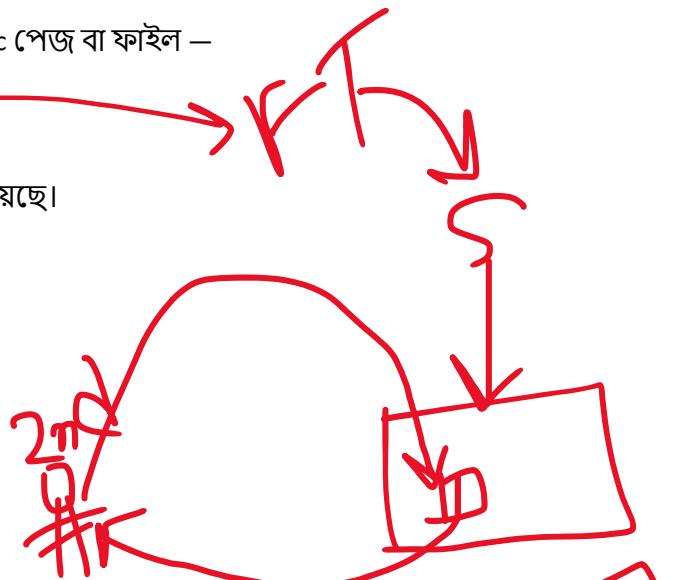
/profile%3Flogout ≠ /profile?logout

ক'বল ওর কাছে /profile%3Flogout মানে একদম আলাদা একটা static পেজ বা ফাইল –
যেমন /about.html, /logo.png এর মতো।

তাই Cache system ভাবে:

“এই /profile%3Flogout নামে এক্সেলি একটা পেজ রিকোয়েস্ট হয়েছে।
এটা কেবল একবার এসেছে – মনে হয় এটা cache করা যায়।”

তাই সেটা cache করে রাখে।



কিন্তু Backend করে ডিকোড...

Backend %3F কে ? বানিয়ে ফেলে

Backend তখন বুঝে **/profile?logout**

আর যেহেতু ইউজার logged-in ছিল, তাই ডেটা দিয়ে দেয়

ক'বল Cache system জানেই না যে ওটা ছিল একটা প্রাইভেট page!

উপসংহার:

Cache system URL ডিকোড না করেই যেটা দেখছে সেটাই ধরে নেয়।

তাই /profile%3Flogout কে দেখে ও ভাবে এটা আলাদা একটা static ফাইল – আর সেটা cache করে ফেলে।

এটাকেই বলে parser behavior mismatch – আর এখান থেকেই শুরু হয় Web Cache Deception attack.

Example:

/profile%23wcd.css

profileAbc.wcd

- The origin server decodes %23 to #. It uses # as a delimiter, so it interprets the path as /profile and returns profile information.
- The cache also uses the # character as a delimiter, but doesn't decode %23. It interprets the path as /profile%23wcd.css. If there is a cache rule for the .css extension it will store the response.

✍ কেন Encoded Non-Printable Characters দিয়ে URL Truncation টেস্ট করা উচিত?

আপনি যখন Web Cache Deception বা অন্য কোনো URL-based vulnerability খুঁজে দেখেন, তখন কিছু non-printable character থাকে যেগুলো দেখতে পাওয়া যায় না, কিন্তু ডিকোড হলে তারা URL-এর আচরণ পরিবর্তন করে ফেলতে পারে।

Non-printable characters হল এমন ক্যারেক্টারগুলো, যেগুলো সাধারণভাবে ক্লীনে দৃশ্যমান হয় না বা প্রিন্ট করা যায় না। এগুলো সাধারণত বিশেষ কাজের জন্য ব্যবহৃত হয়, যেমন ডেটা ফরম্যাটিং বা কন্ট্রোলিং।

উদাহরণ:

NULL character (%00):

এটা URL বা টেক্স্টের শেষ নির্দেশ করে। অনেক সময় এটি URL-এর ট্রান্সমিশন শেষ করে দেয়, যার ফলে বাকি অংশ দেখা যায় না।

Line Feed (%0A):

এটা একটি new line নির্দেশ করে, অর্থাৎ পরবর্তী টেক্স্ট বা ইনপুটকে আলাদা লাইন হিসেবে বিবেচনা করা হয়।

Tab (%09):

এটা একটি tab সূচির জন্য ব্যবহৃত হয়, যা টেক্স্ট ফরম্যাটিংয়ে ব্যবহৃত হয়, তবে এটি URL-এর ক্ষেত্রে বিভ্রান্তি তৈরি করতে পারে।

Carriage Return (%0D):

এটা সাধারণত পুরনো সিস্টেমে new line হিসেবে ব্যবহৃত হতো, তবে এখনো কিছু জায়গায় এটি ফাংশনাল থাকতে পারে।

❖ URL Truncation মানে কী?

URL Truncation মানে হচ্ছে:

ক্ষেত্রে ব্রাউজার বা সার্ভার যদি কোনো কারণে একটি বড় URL-এর শেষের অংশ কেটে ফেলে (truncate করে), তাহলে মূল URL-এর behavior বদলে যেতে পারে।

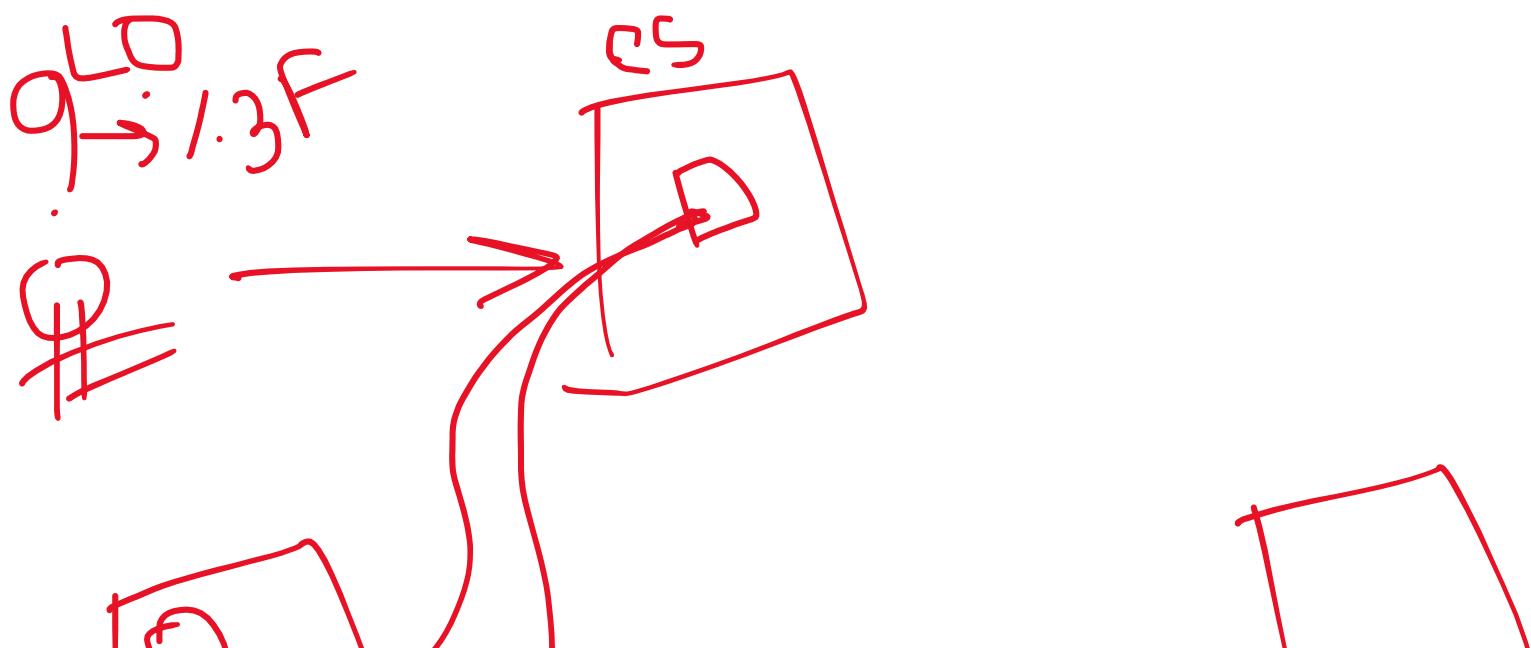
🔍 উদাহরণ:

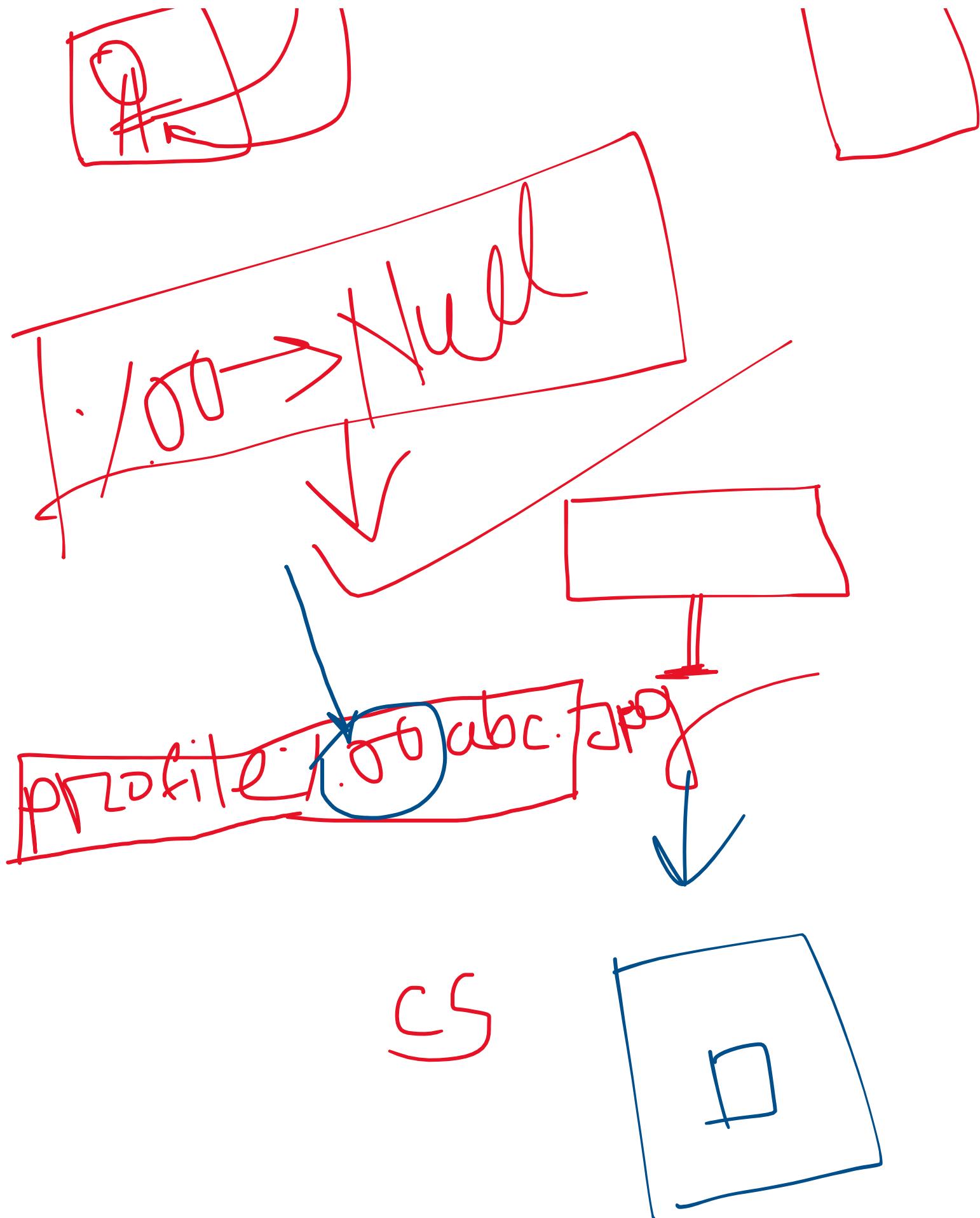
<https://example.com/admin/dashboard?user=admin>

! যদি কোনো কারণে সার্ভার শুধু এটুকু পর্যন্ত দেখে:

<https://example.com/admin/dashboard>

তাহলে query parameter ?user=admin হারিয়ে গেল — অর্থাৎ এটা authentication বা authorization এর জন্য গুরুত্বপূর্ণ ছিল।

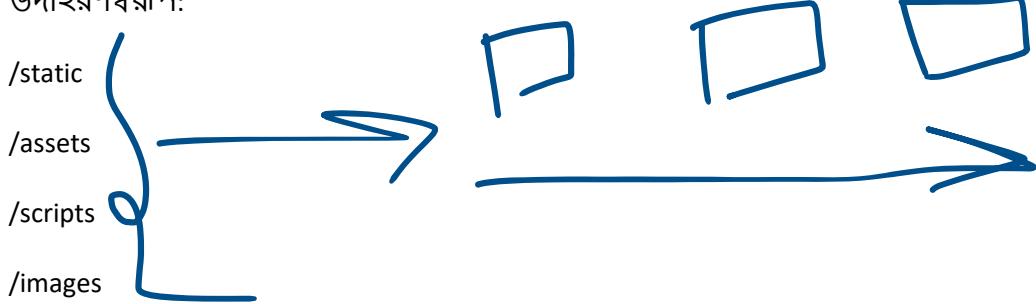




Exploiting static directory cache rules

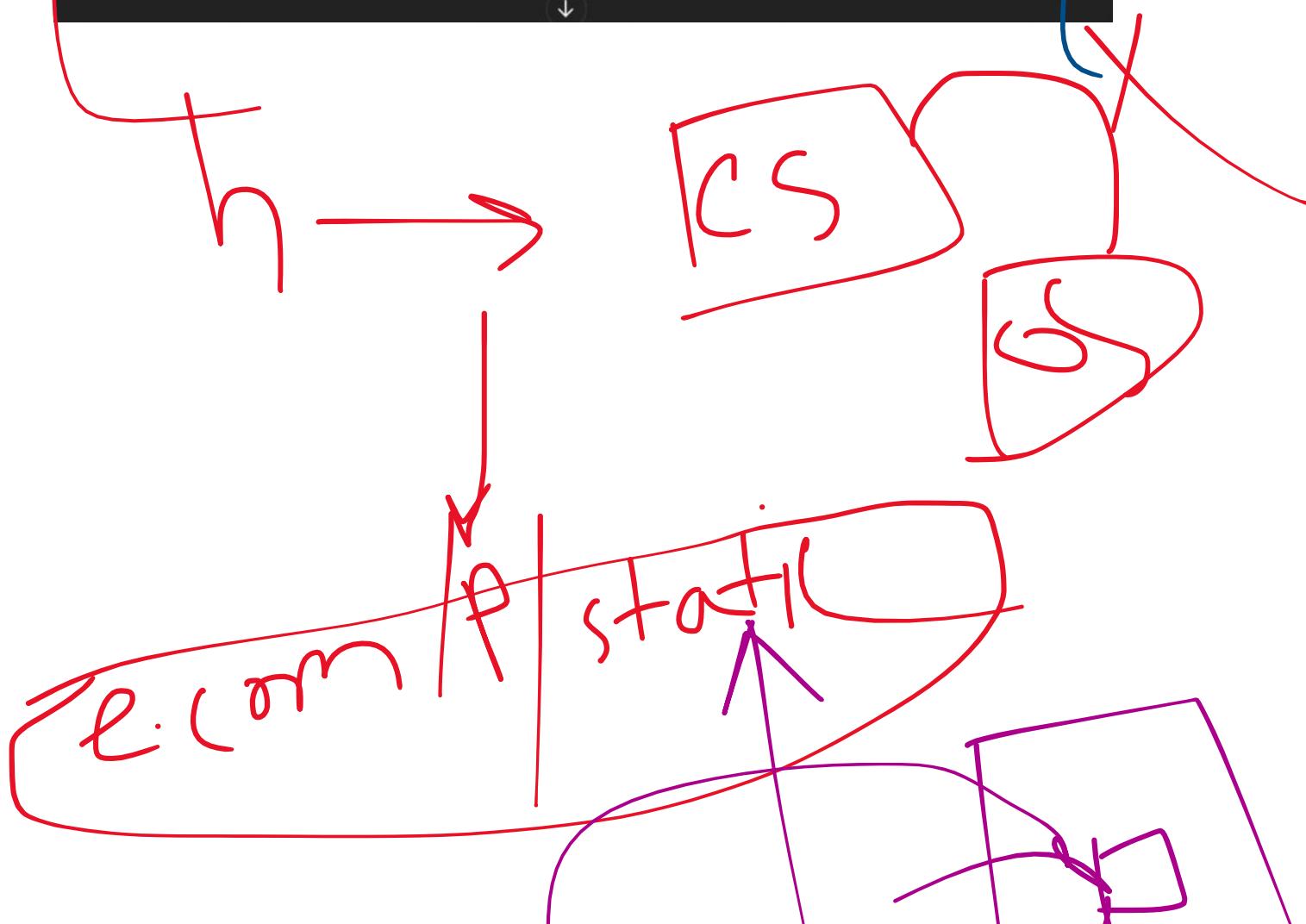
Monday, April 7, 2025 12:55 AM

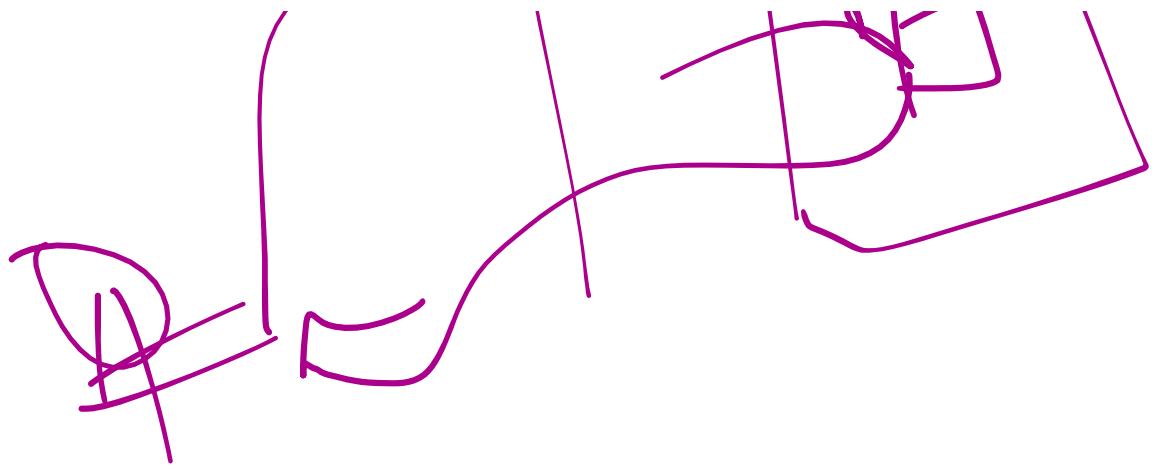
ওয়েব সার্ভারগুলো সাধারণত static resources (যেমন CSS, JavaScript, image) নির্দিষ্ট ডিরেক্টরিতে সংরক্ষণ করে।
উদাহরণস্বরূপ:



এখন, ক্যাশ সিস্টেমের জন্য কিছু নির্দিষ্ট cache rules থাকে, যা এই ডিরেক্টরির URL path এর prefix দেখে ক্যাশ করে।
যেমন, /static/ যদি থাকে, তবে ক্যাশ সিস্টেম জানে এটি static resource, এবং এটি ক্যাশে রাখবে।

| ফাইল টাইপ | URL | কোথায় আছে | ক্যাশ হবে? |
|-----------|---|-----------------------------|---|
| CSS ফাইল | https://example.com/static/style.css | /static/ ফোল্ডারে | <input checked="" type="checkbox"/> হ্যাঁ |
| JS ফাইল | https://example.com/scripts/app.js | /scripts/ ফোল্ডারে | <input checked="" type="checkbox"/> হ্যাঁ |
| ইমেজ | https://example.com/images/logo.png | /images/ ফোল্ডারে | <input checked="" type="checkbox"/> হ্যাঁ |
| পেজ | https://example.com/login | /login (কোনো static dir না) | <input type="checkbox"/> না |





Normalization involves converting various representations of URL paths into a standardized format. This sometimes includes decoding encoded characters and resolving dot-segments, but this varies significantly from parser to parser.

Parser মূলত ইনপুট ডেটাকে সঠিকভাবে ব্যাখ্যা (interpret) করে এবং সিস্টেম বা প্রোগ্রামকে সেই ডেটা নিয়ে কাজ করার সুযোগ দেয়।

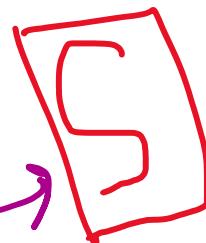
৭

- একটি **URL parser** URL-এর পাথ, কোয়েরি স্ট্রিং, এবং অন্যান্য অংশগুলো আলাদা করে চিনে এবং তাদের বিশ্লেষণ করে।
- একটি **HTML parser** HTML ডকুমেন্টে লেখা কোডগুলোকে বোঝে এবং ওয়েব পেজের উপাদানগুলো তৈরি করে।

Example:

◦ **Original Request:**

ধরা যাক, অ্যাপ্লিকেশনে এমন একটা রিকোয়েস্ট যায়:



/profile

ক্ষেত্রে এটা হলো সেই পাথ যেটা প্রোফাইল পেজ রিটার্ন করে — এবং এটা সাধারণত authentication প্রয়োজন করে।

◦ **কিন্তু Web Cache Deception** এর জন্য, আপনি রিকোয়েস্টটাকে একটু চালাকিভাবে পরিবর্তন করেন:

/static/..%2fprofile

E

ক্ষেত্রে এখানে:

- /static/ দেখে cache ধরে এটা একটা স্ট্যাটিক ফাইলের পাথ।
- ..%2f মানে ../, যেটা এক ধাপ উপরের ডিরেক্টরি বোঝায়।
- %2f হলো / এর URL encoded ফর্ম।



/static/..profile

.. মানে: "static থেকে এক ধাপ উপরে যাও" → আপনি / root ডিরেক্টরিতে ফিরলেন

তারপর /profile → অর্থাৎ, root থেকে profile ডিরেক্টরিতে চুকলেন

🔍 এখন কী হয়:

Cache Server ভাবে:

Cache Server ভাবে:

- এটা `/static/..%2fprofile` → মানে স্ট্যাটিক ফাইল পাথ।
- সে কোনো ডিকোড বা নরমালাইজ করে না।
- তাই সে cache করে ফেলে প্রোফাইল পেজটা।

Origin Server (মানে মূল অ্যাপ্লিকেশন সার্ভার) এটা ডিকোড করে:

`/static/..profile` → `/profile`
অর্থাৎ, আসল প্রোফাইল পেজ রিটার্ন করে।

ক্ষেত্রে যখন attacker path traversal ব্যবহার করে (যেমন: `../profile`), তখন সে চাই যে cache বা origin server যেন সেই path-টা ঠিকভাবে decode করে ও resolve করে।

কিন্তু সমস্যা হলো:

⚠ Victim-এর ব্রাউজার নিজেই যদি `../` resolve করে ফেলে...

তাহলে:

Cache server বা origin server-এর কাছে আসবে একদম ক্লিন URL (যেমন `/profile`)

অর্থাৎ attacker-এর crafted traversal path cache পর্যন্ত পৌঁছাবেই না!

Each dot-segment in the path traversal sequence needs to be encoded.

❖ Example:

1 সাধারণভাবে:

<http://victim.com/images/../../secret.txt>

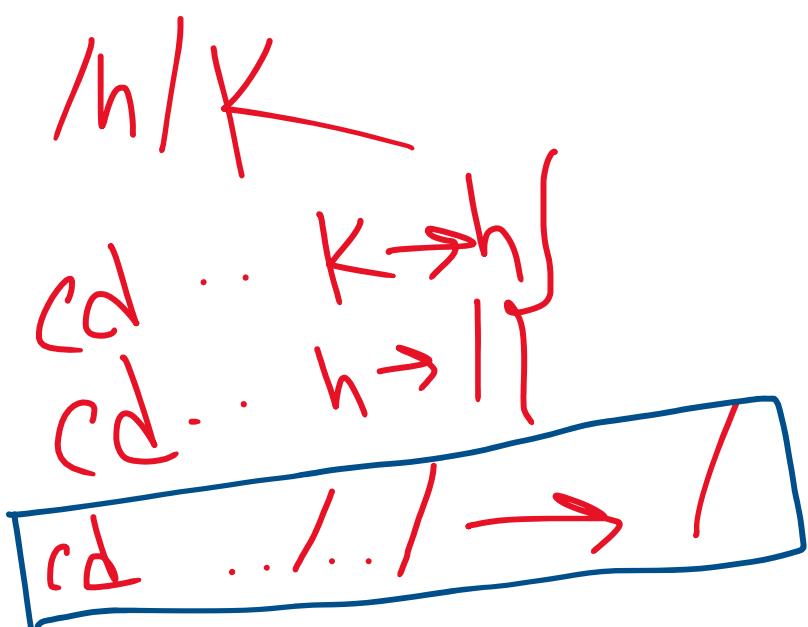
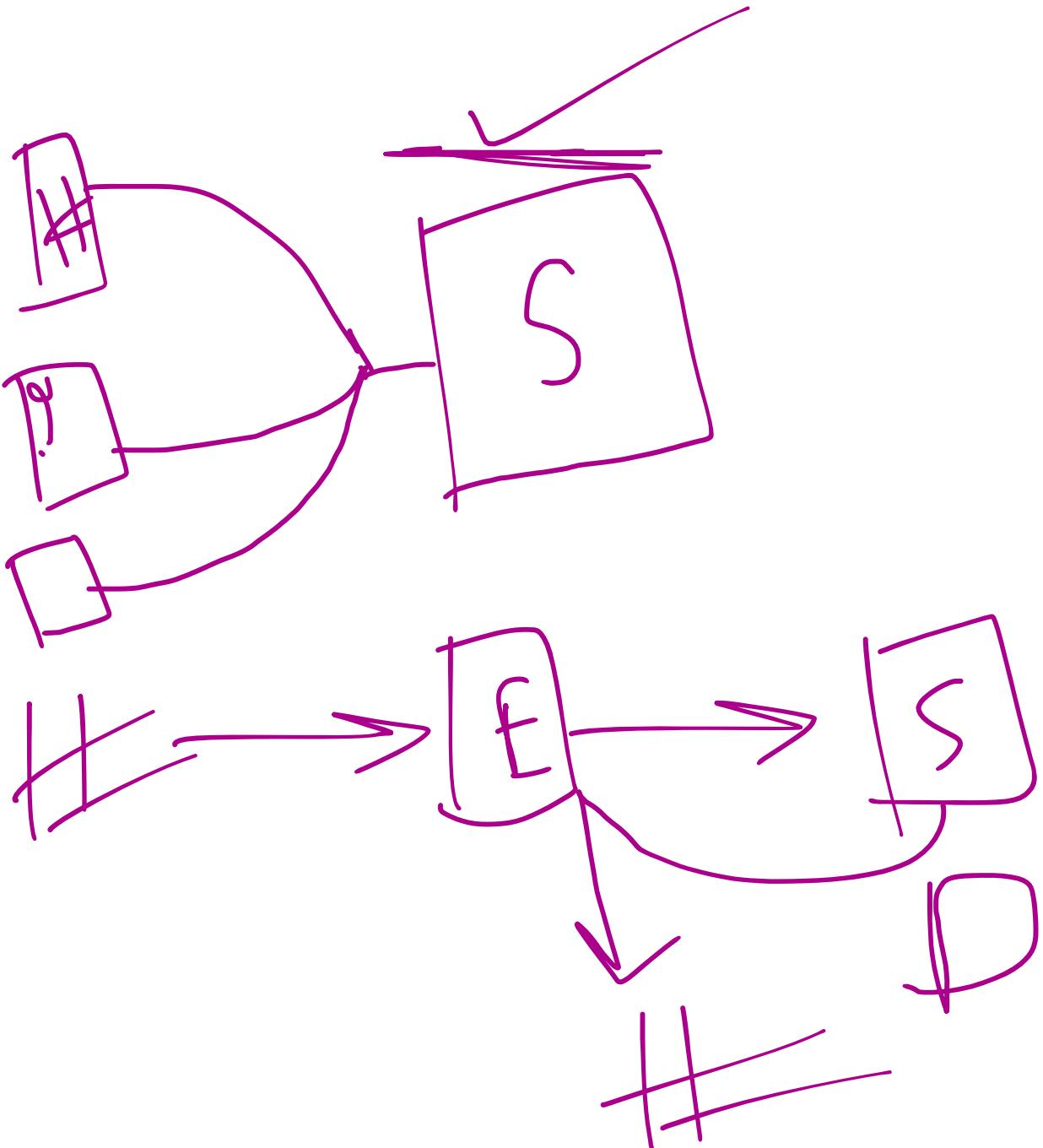
● ব্রাউজার বুঝে ফেলবে এটা path traversal → নিজে resolve করে ফেলবে।

2 Encoding দিয়ে:

<http://victim.com/images/%2e%2e%2e%2e/secret.txt>

❖ এখানে `%2e%2e/` মানে `../`

এখন ব্রাউজার আর resolve করবে না — পুরোটা সরাসরি সার্ভারে পাঠাবে।



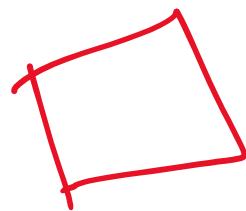
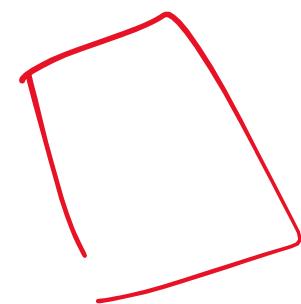
| CD ... / ... |

... → .1.2e

1 → .1.2f

20 → ↗ ↘ ↙ ↖

↓
RT
→ .1.2e



X

T

. → 1.2e

Detecting normalization by the origin server

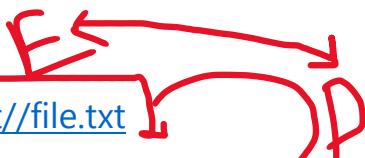
Tuesday, April 8, 2025 10:39 PM

10

Normalization কী বোঝায়?

Normalize একটা সাধারণ/স্ট্যান্ডার্ড ফরম্যাটে রূপান্তর করা, যাতে সার্ভার বুঝতে পারে ইউজার আসলে ঠিক কোন জায়গায় ষেতে চাইছে।

<https://example.com/app/..%2e%2e/secret//file.txt>
<https://example.com/secret/file.txt>



পরীক্ষা করার পদ্ধতি কী?

ধরুন, আপনার একটা ইউআরএল আছে /profile — যেটা প্রোফাইল দেখায়। এখন আপনি এটা একটু ঘুরিয়ে /aaa/..%2fprofile করে পাঠালেন।

এখানে,

/aaa/ হলো একটা মিথ্যে ফোল্ডার,

.. মানে এক ধাপ পেছনে যাওয়া (মানে /aaa/ থেকে বের হয়ে আসা),

%2f মানে / — এটা URL encode করা ছিল।

curl -i -H "Host: example.com" https://example.com/aaa/..%2fprofile

This option tells curl to include the HTTP headers in the output. Normally, curl only displays the body of the server's response.

To test how the origin server normalizes the URL path, send a request to a non-cacheable resource with a path traversal sequence and an arbitrary directory at the start of the path.

Non-cacheable resource মানে কী?

Non-cacheable resource হলো এমন কোনো ওয়েব রিসোর্স (যেমন API, user-specific page, private data) যেটা ক্যাশে রাখা উচিত না।

অর্থাৎ, যখনই কেউ ওই রিসোর্সে অনুরোধ পাঠাবে, সার্ভার প্রতি বার নতুন করে সেই ডেটা পাঠাবে। এটি কখনোই ব্রাউজার বা CDN-এর ক্যাশে জমা থাকবে না।

To choose a non-cacheable resource, look for a non-idempotent method like POST

Idempotent মানে এমন কোনো অনুরোধ (request) যেটা আপনি একবার, দশবার বা একশোবার করলেও সার্ভারে একই প্রভাব ফেলে। (GET)

Non-idempotent মানে — বারবার req করলে ভিন্ন ভিন্ন ফল হতে পারে। (POST)

```
POST /aaa/..%2fprofile HTTP/1.1
Host: example.com
Content-Type: application/json
Cache-Control: no-cache
Content-Length: 17

{"test":"data"}
```

যদি রেসপন্সে profile info আসে →

সার্ভার slash decode করে এবং path normalize করে

যদি 404 বা অন্য error আসে →

সার্ভার slash decode করেনি বা .. resolve করেনি

When testing for normalization, start by encoding **only the second** slash in the dot-segment. This is important because some CDNs match the slash following the static directory prefix.

/aaa/..profile

/aaa/..1.%2fP

CDN sees ../ and blocks it.

/static/ → হচ্ছে static directory prefix

Original: /aaa/..profile

Encoded: /aaa/..%2fprofile ← the '%2f' is the encoded second slash

It's important because **some CDNs block ..** but ignore ..%2f, so by encoding the second slash, you can bypass the CDN's check and still reach the protected path on the origin server.

↗ .. / → E

You can also try encoding the full path traversal sequence, or encoding a dot instead of the slash. This can sometimes impact whether the parser decodes the sequence.

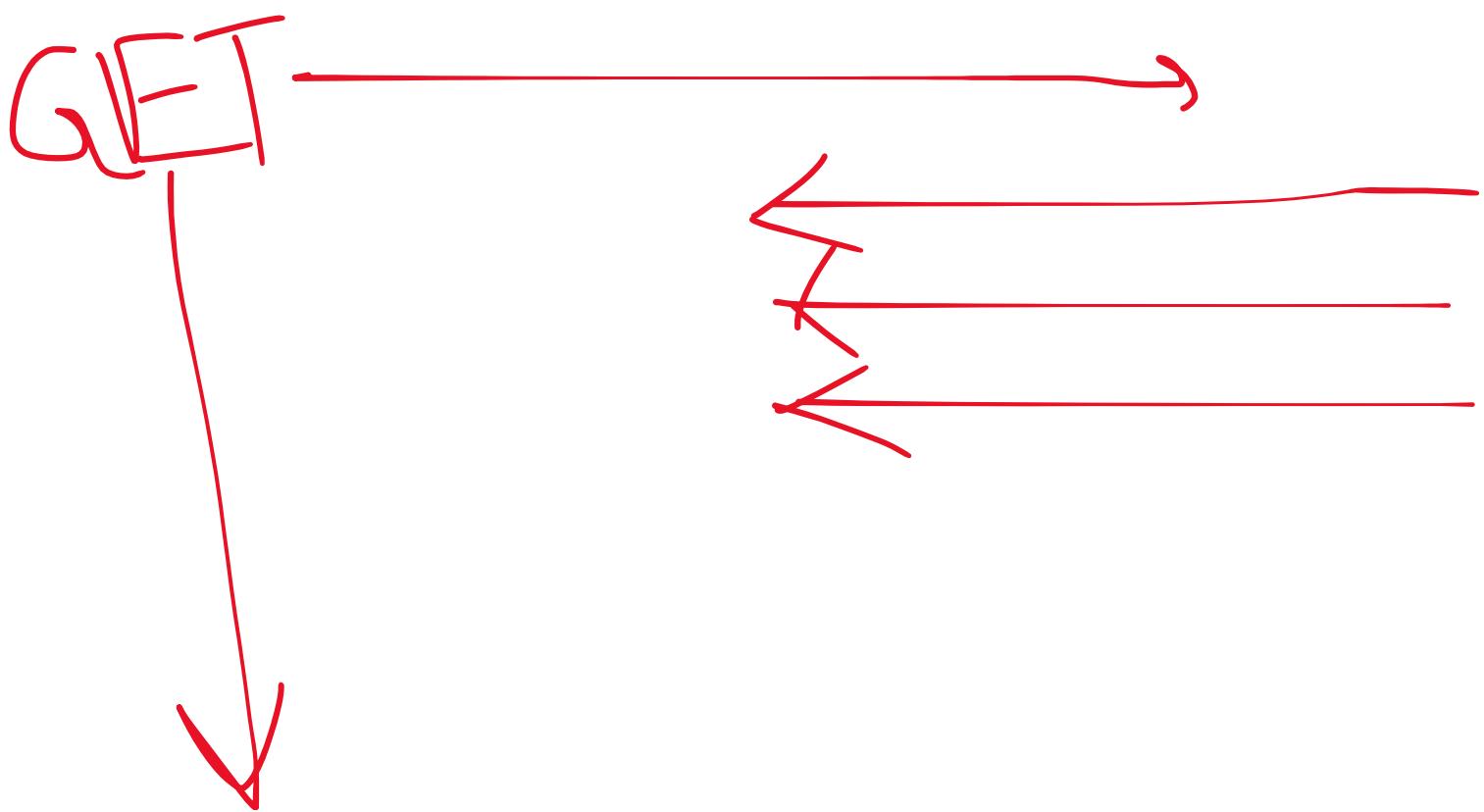
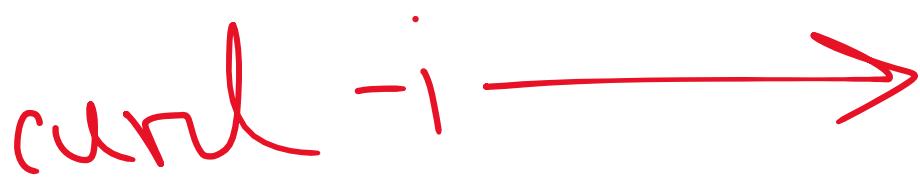
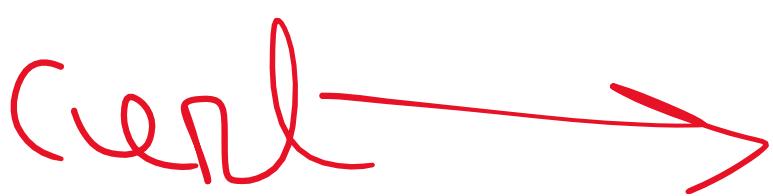
Example variations:

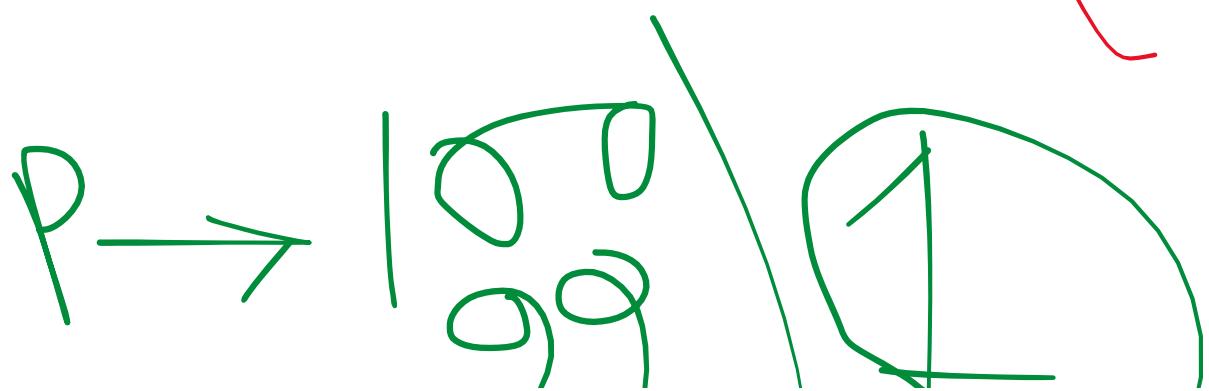
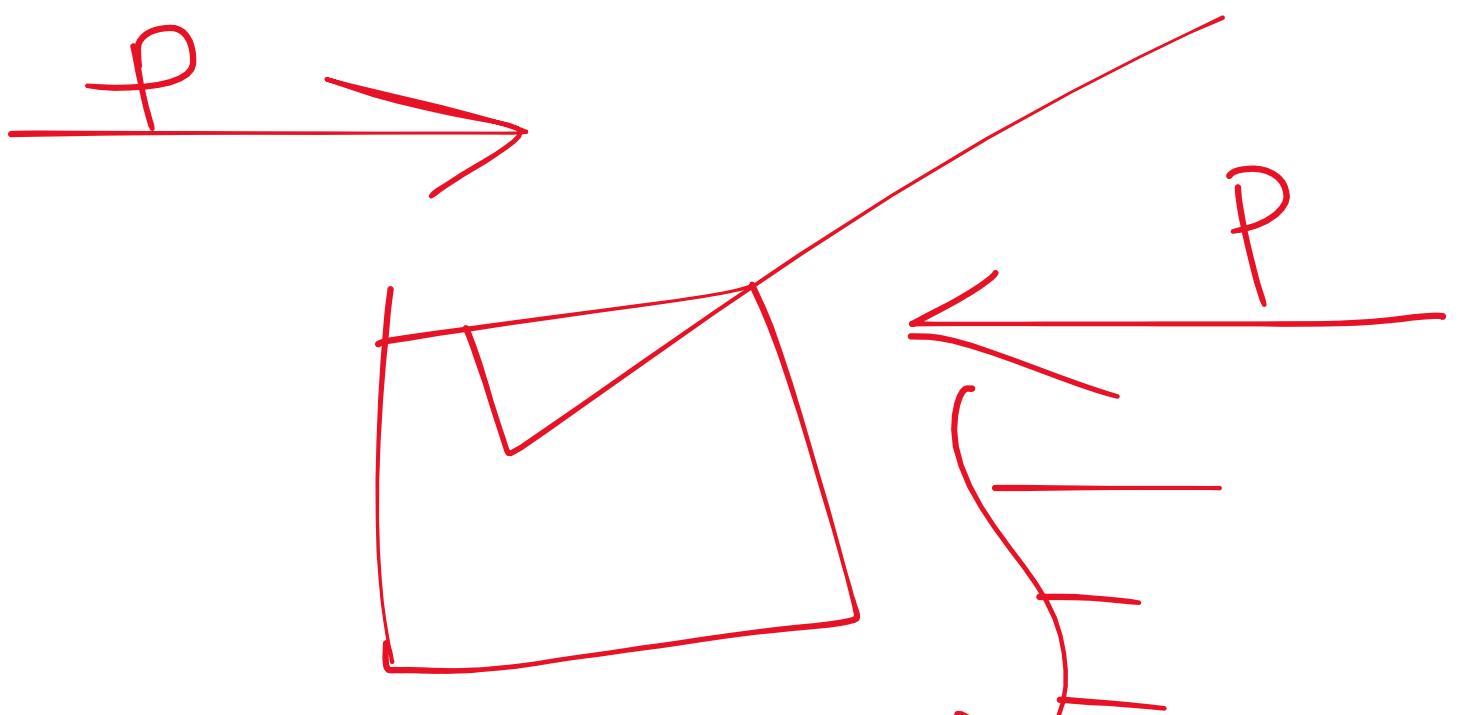
- /static/..%2fadmin
- /static/%2e%2e/admin
- /static/%2e./admin

might block one form but allow another



might block one form but allow another







You can try to bypass the cache and reach a sensitive dynamic page by crafting a URL like this:

/<static-directory-prefix>/..%2f<dynamic-path>

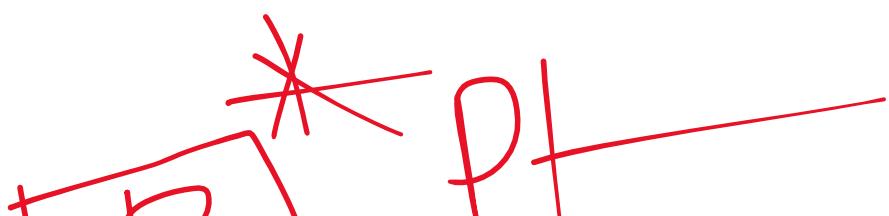
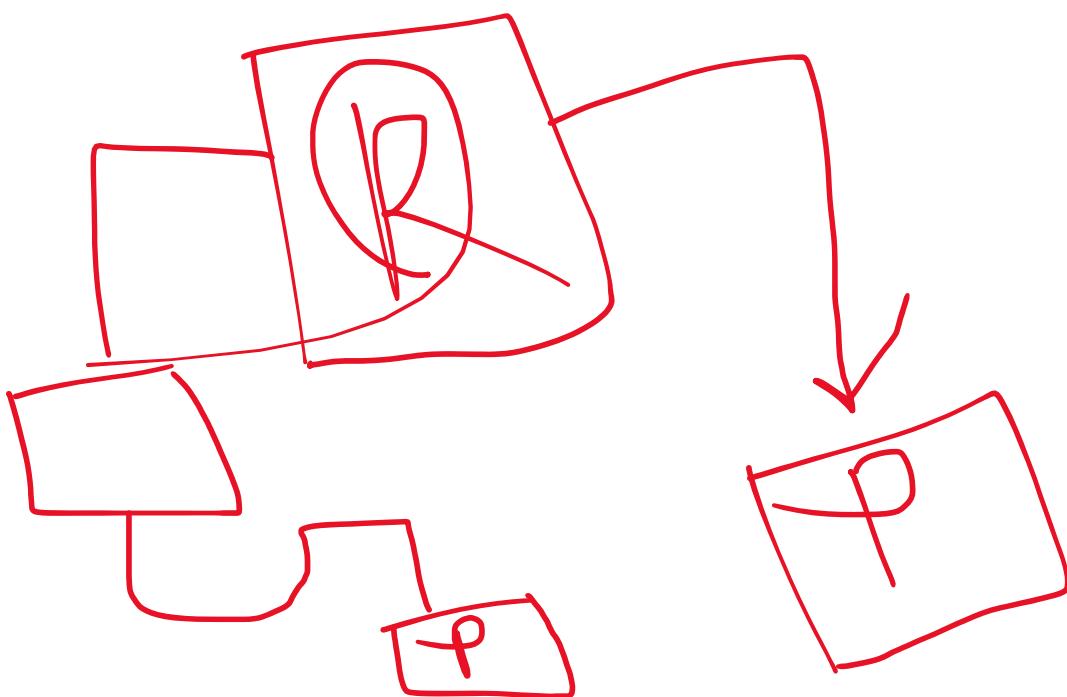
🔍 Break it down:

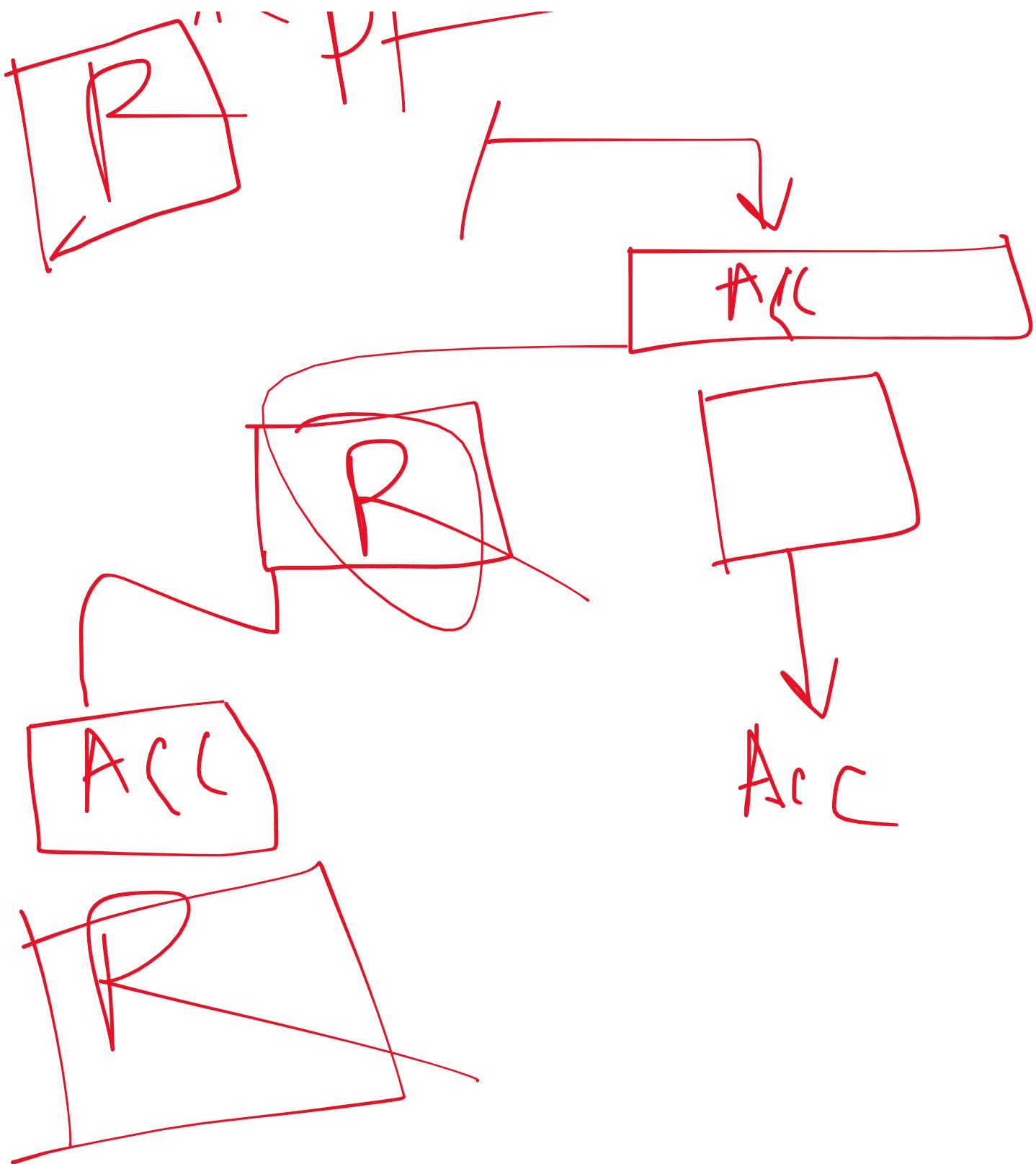
<static-directory-prefix> → something like /assets, /static, /images

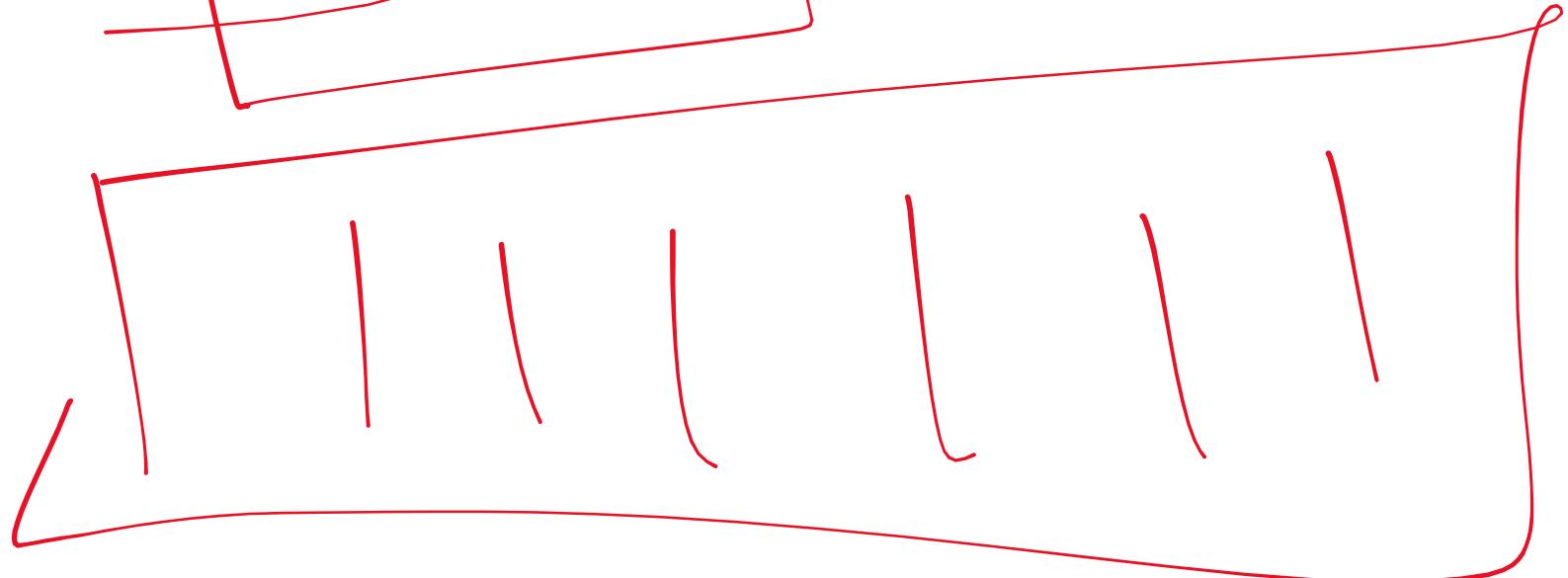
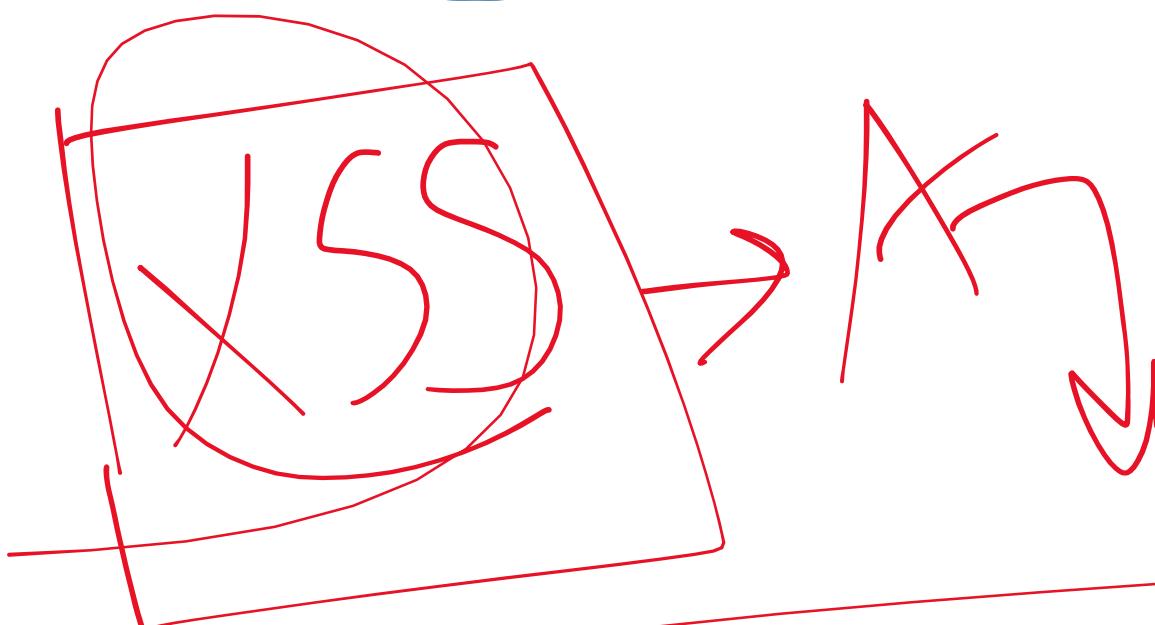
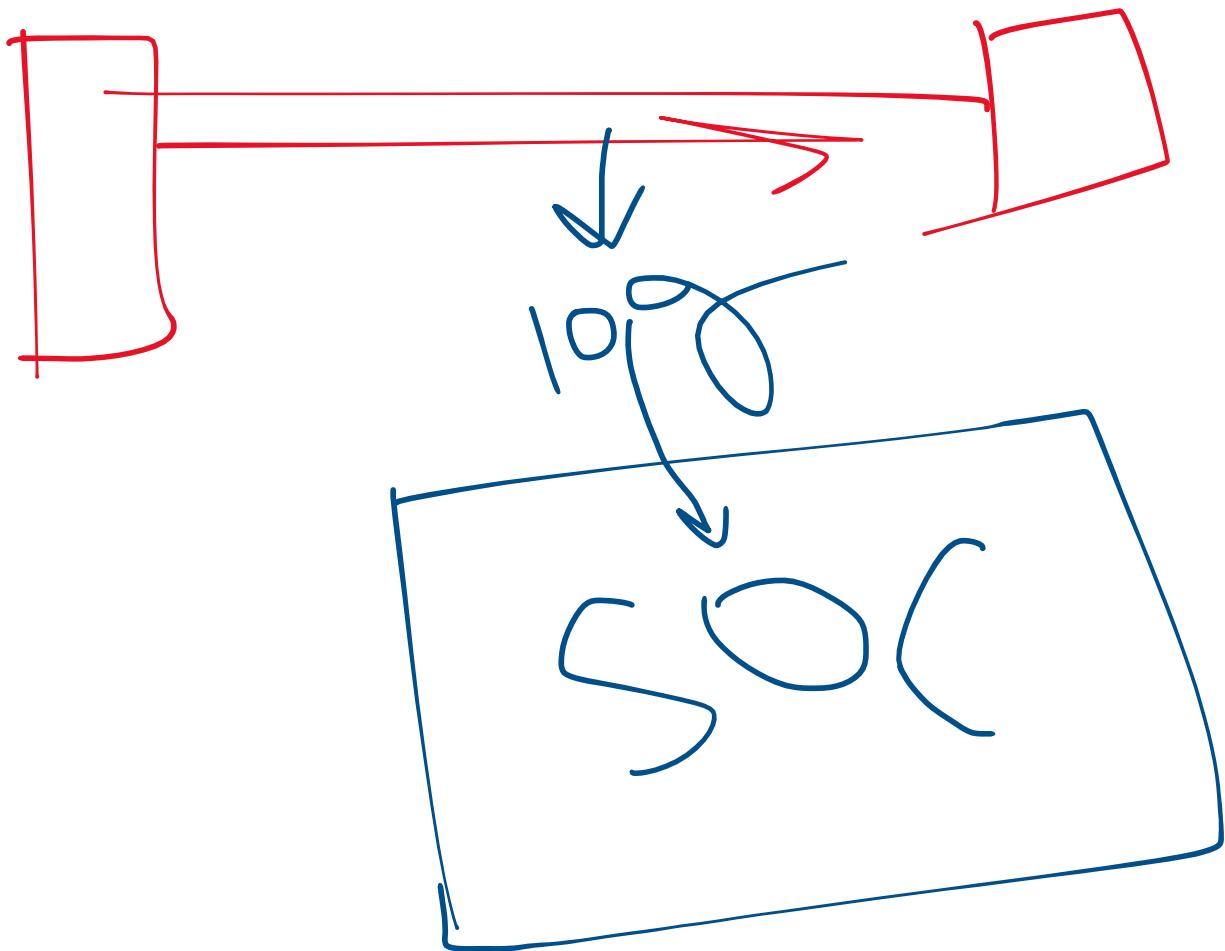
..%2f → encoded ../ (dot-segment to go up a directory)

<dynamic-path> → something private like /profile, /account, /admin

SP









Detecting normalization by the cache server

Saturday, April 12, 2025 1:55 AM

A **static directory** refers to a folder on a web server where static files (such as images, CSS, JavaScript, videos, and other assets) are stored. These files are not dynamically generated but are served as-is to users when requested. **Static files are typically used for the layout, design, and media content of a website.**

For example:

/static/images/logo.png

/static/css/styles.css

/static/js/app.js

These files remain the same unless manually updated and don't change based on user interaction or other server-side logic. Static directories are often used for content that doesn't require frequent updates or server-side processing.

Look in **Burp Suite**'s HTTP history for URLs that point to static files (like in /static/, /images/, /css/), and check if the responses are cached.

📁 Common Static Directories:

/static/

/assets/

/images/

/img/

/css/

/js/

/media/

/fonts/

/scripts/

/files/

/public/

💡 How to Do It in Burp:

1. Go to Proxy → HTTP history
2. Click the **filter bar** (top right of the history panel)
3. Set filters like:
 - **Show only** → Status code: 2xx
 - **MIME types** → Check Script, Image, and CSS

➤ Replace part of the URL (after the static directory) with random text (like /assets/aaa).

➤ If the response is still cached, it means the cache is likely based on the /assets directory.

404 error পেজগুলো সাধারণত ক্যাশ হয় না, কিন্তু যদি একই ডিরেক্টরির অন্য পেজ ক্যাশ হয়ে যায়, তাহলে বুঝবেন যে ডিরেক্টরি
ভিত্তিক ক্যাশিং হচ্ছে।



ভিত্তিক ক্যাশিং হচ্ছে।

<dynamic-path>%2f%2e%2e%2f<static-directory-prefix>

... |

<dynamic-path> এটা একটি ডাইনামিক পাথ (যেমন /user/123/profile)।

%2f%2e%2e%2f: এই অংশটি ../ এর এনকোডেড ফর্ম, যা path traversal এর জন্য ব্যবহৃত হয়। এটা ব্যবহারকারীর পাথ থেকে parent directory তে যেতে সাহায্য করে।

<static-directory-prefix>: এটা হলো static ডিরেক্টরি (যেমন /assets/) যেখানে আপনি file অ্যাক্সেস করতে চান।

/user/123/profile%2f%2e%2e%2fassets/js/script.js

... |

result:

Cache server: এটি %2e%2e%2f কে .. হিসেবে decode করে এবং /assets/js/script.js কে cache করে রাখে।

Origin server: এটি হয়তো path traversal স্থিতিভাবে handle করবে না, এবং 404 error দেখাবে।

উদাহরণ:

যদি আপনি /assets/ ডিরেক্টরি অ্যাক্সেস করতে চান, তাহলে এভাবে লিখবেন:

/assets%2e%2e%2fstatic/js/script.js

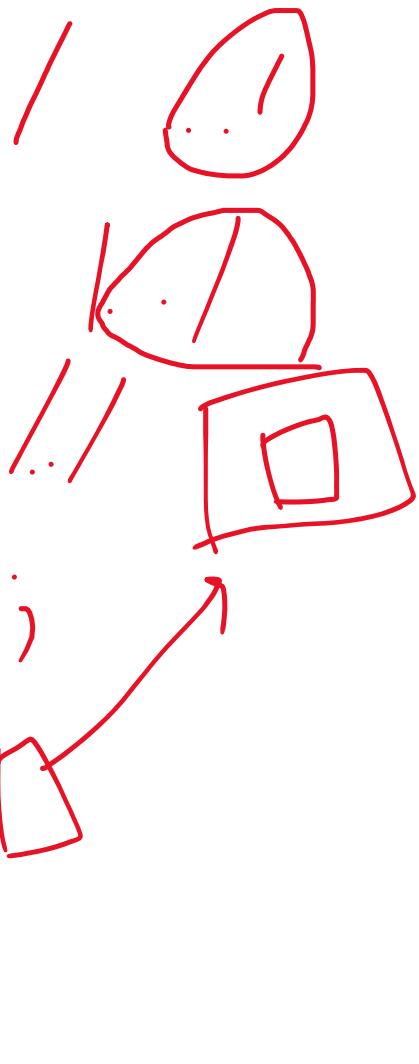
এখানে unencoded slash (/) দেয়ার দরকার নেই, কারণ cache server %2f কে সঠিকভাবে decode করে নেবে।

কেন এটা জরুরি:

Cache server এভাবে path-কে সঠিকভাবে handle করবে, যা আক্রমণকারীর জন্য cache poisoning বা unauthorized access এর সুযোগ তৈরি করতে পারে।

Note

When exploiting normalization by the cache server, encode all characters in the path traversal sequence. Using encoded characters helps avoid unexpected behavior when using delimiters, and there's no need to have an unencoded slash following the static directory prefix since the cache will handle the decoding.



In this situation, path traversal alone isn't sufficient for an exploit. For example, consider how the cache and origin server interpret the payload /profile%2f%2e%2e%2fstatic:

- The cache interprets the path as: /static
- The origin server interprets the path as: /profile%2f%2e%2e%2fstatic

The origin server is likely to return an error message instead of profile information.

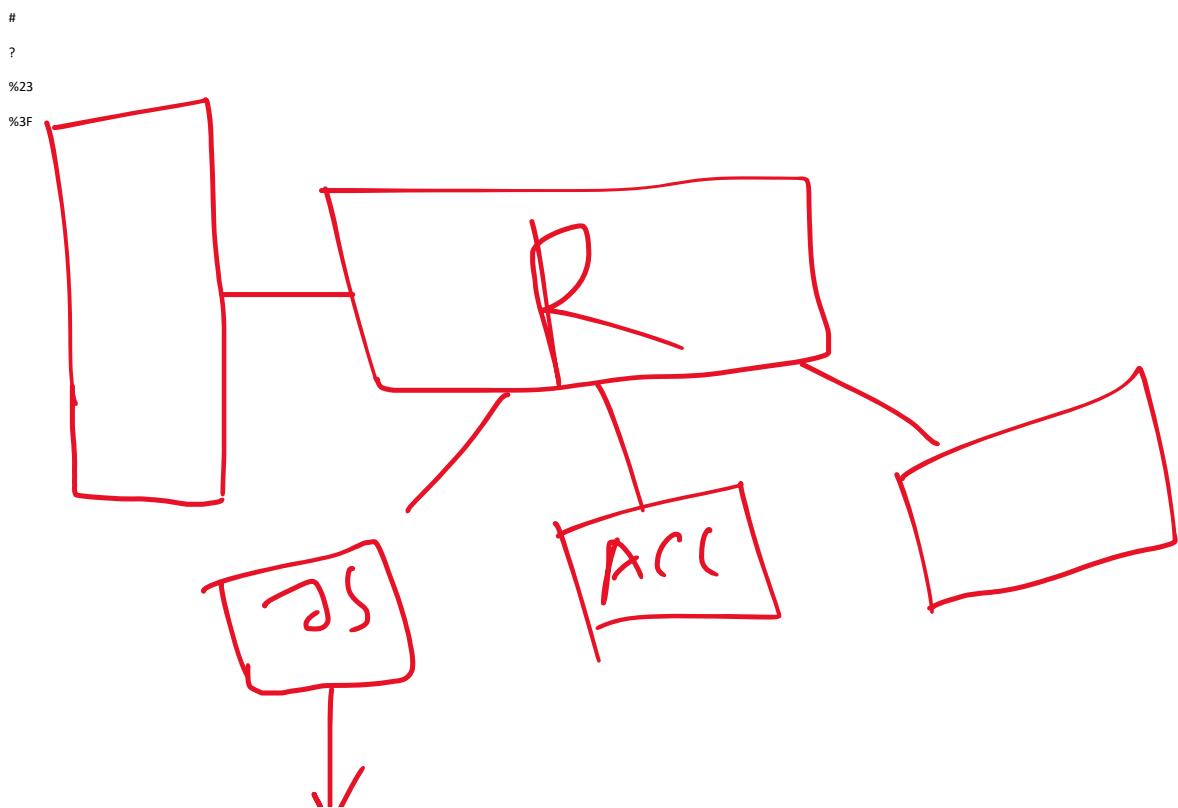
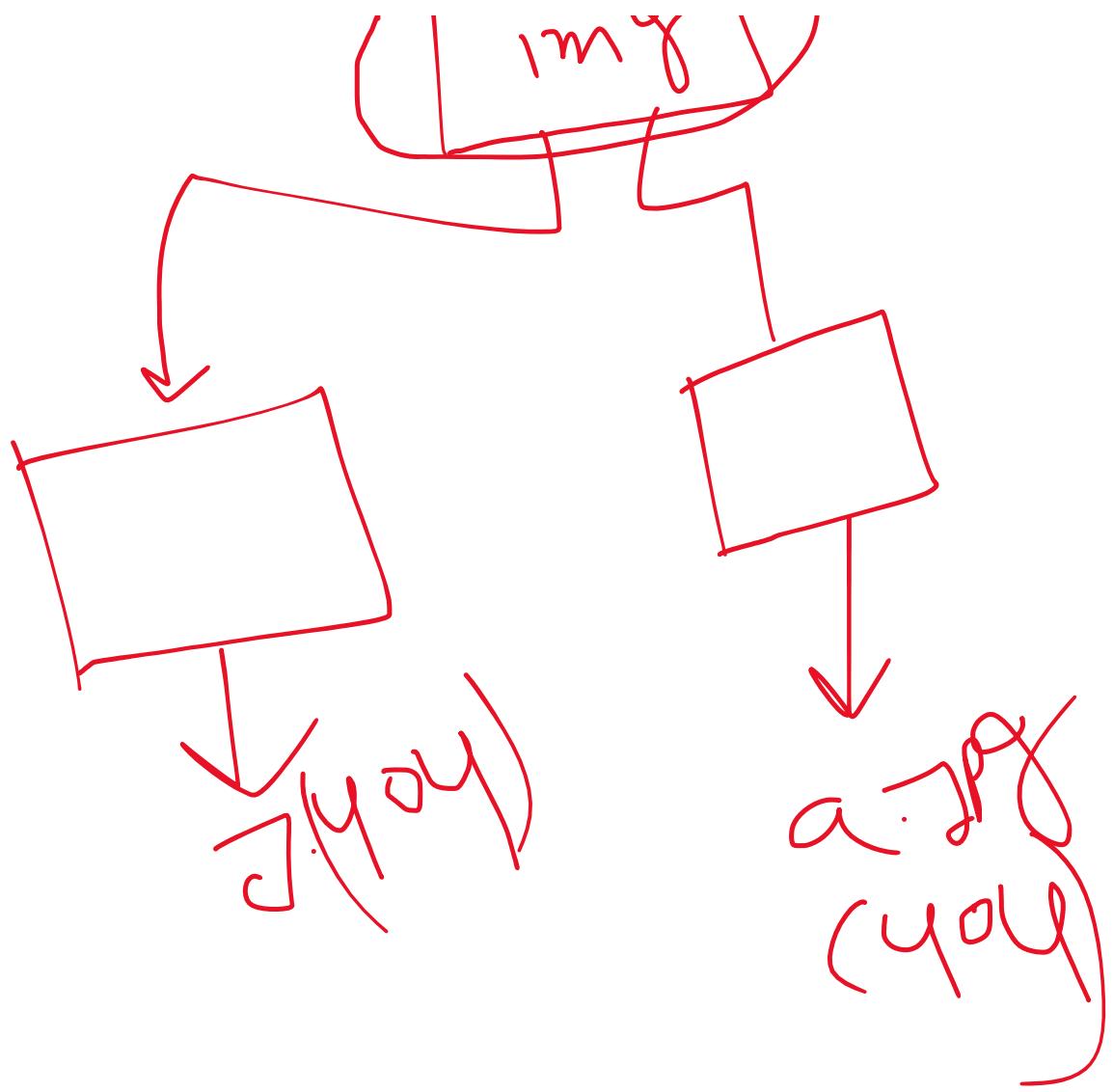
To exploit this discrepancy, you'll need to also identify a delimiter that is used by the origin server but not the cache. Test possible delimiters by adding them to the payload after the dynamic path:

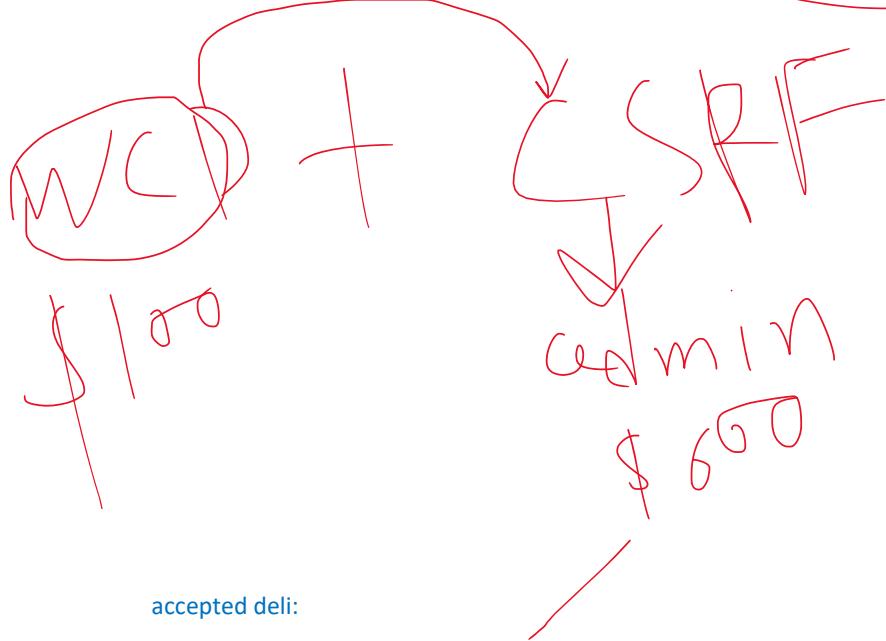
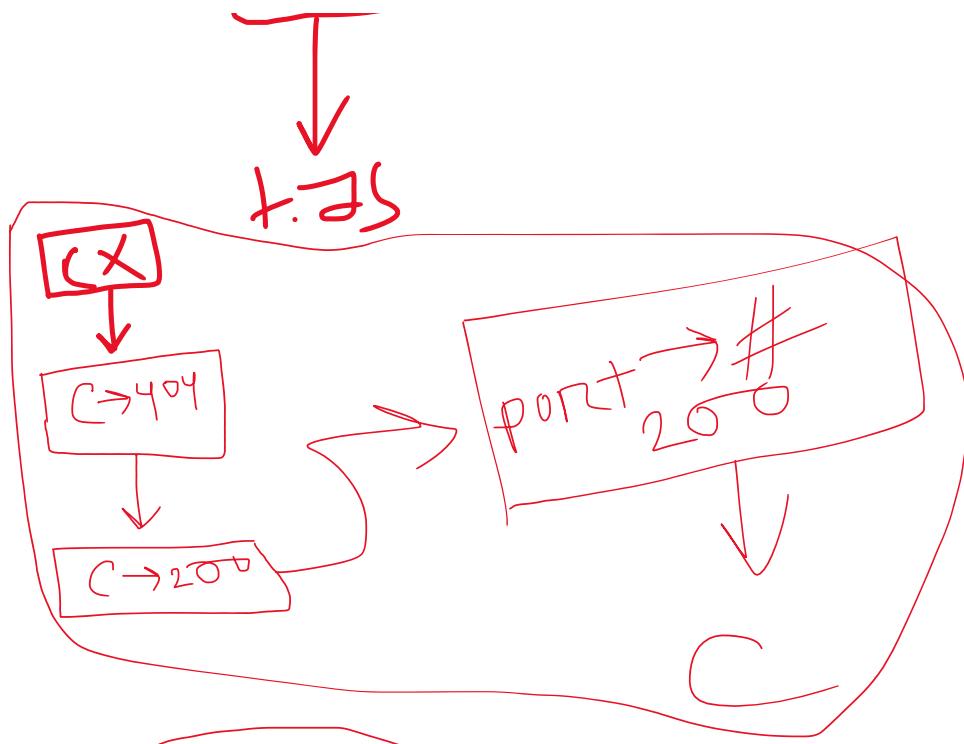
- If the origin server uses a delimiter, it will truncate the URL path and return the dynamic information.
- If the cache doesn't use the delimiter, it will resolve the path and cache the response.

For example, consider the payload /profile;%2f%2e%2e%2fstatic. The origin server uses ; as a delimiter:

- The cache interprets the path as: /static
- The origin server interprets the path as: /profile

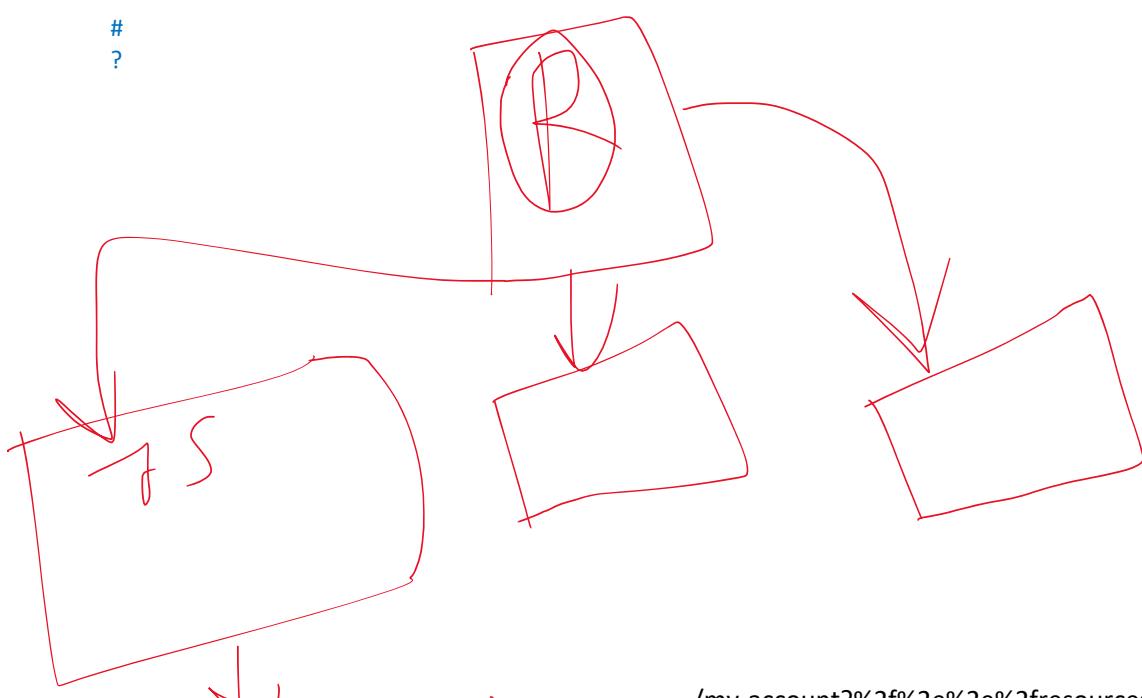
The origin server returns the dynamic profile information, which is stored in the cache. You can therefore use this payload for an exploit.

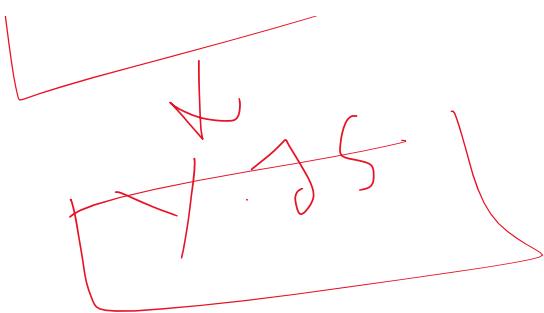




accepted deli:

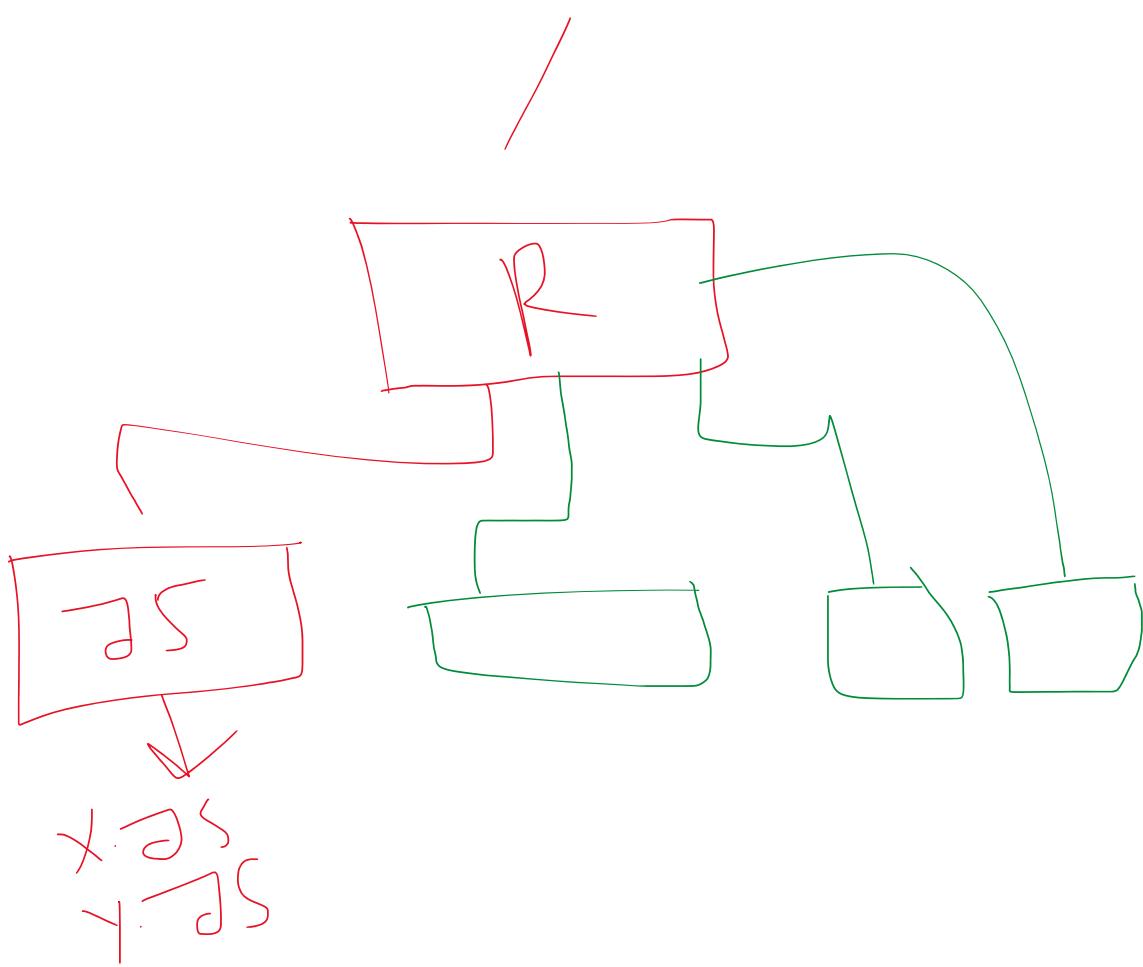
?

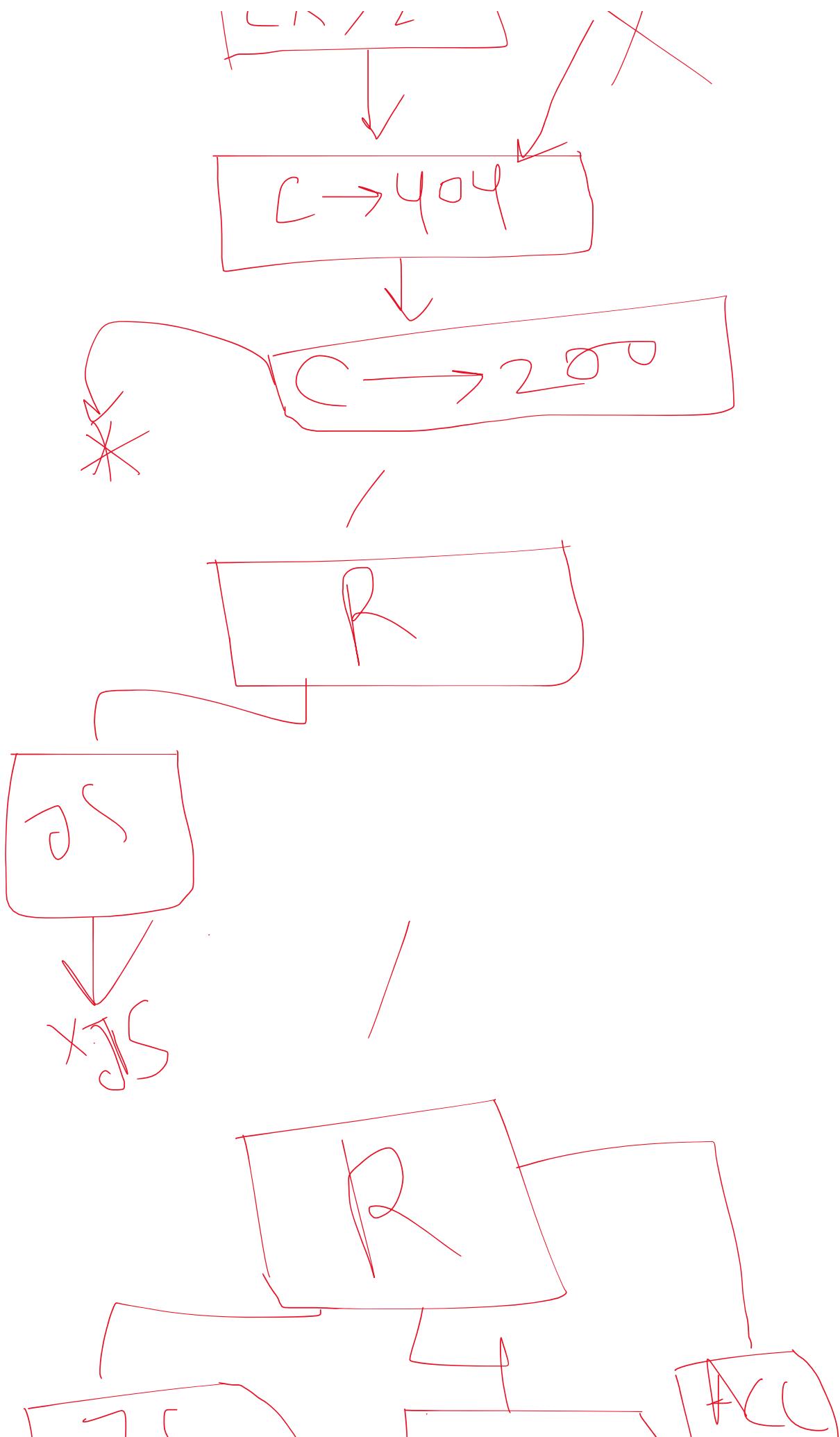


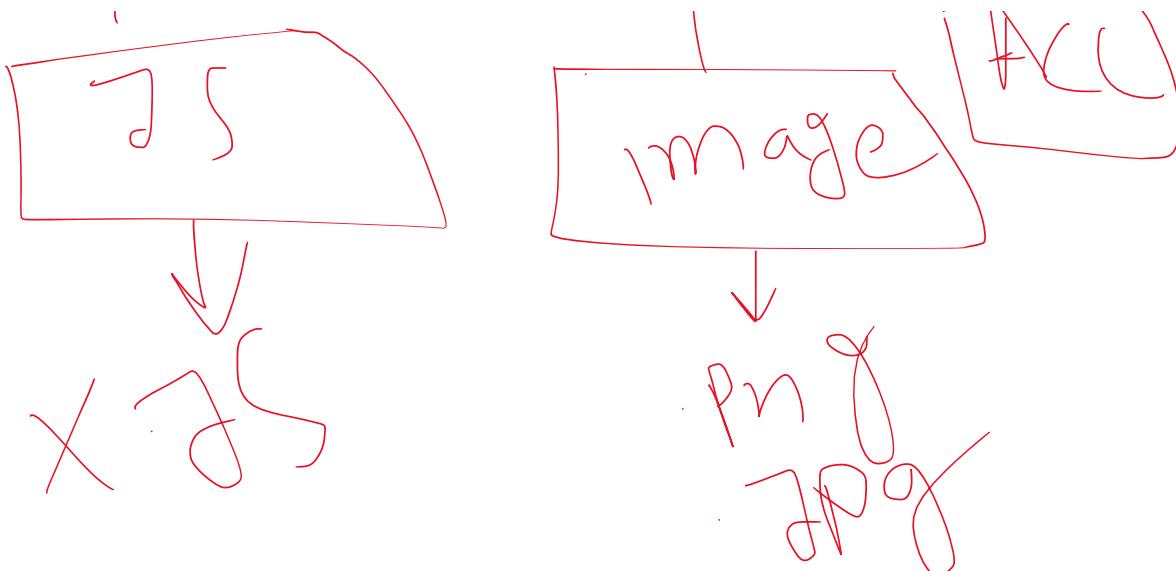


/my-account?%2f%2e%2e%2fresources

/
?







Job description
[SMP CyberSecurity](#)

SMP CyberSecurity

Follow

Summary

- Vacancy: **02**
- Age: **20 to 45** years
- Location: Dhaka (GULSHAN 2)
- Salary: Tk. **30000 – 50000** (Monthly)
- Published: 2 Sep 2024

Requirements

Additional Requirements

- Age 20 to 45 years
- Experience: Proven experience in ethical hacking, penetration testing, and vulnerability assessment. Relevant certifications, such as **CEH (Certified Ethical Hacker)**, **OSCP (Offensive Security Certified Professional)**, or equivalent, are highly desirable. While these certifications are a significant plus, we welcome applications from both freshers eager to start their careers and experienced professionals seeking to leverage their expertise.
- Technical Skills: Strong understanding of networking protocols, operating systems, web technologies, and security tools. Proficiency in scripting languages (e.g., Python, Bash) is a plus.
- Analytical Skills: Excellent problem-solving abilities, with a keen eye for detail in identifying and assessing security vulnerabilities.
- Communication: Strong written and verbal communication skills, with the ability to convey complex technical information to non-technical stakeholders.
- Ethical Standards: High level of integrity and adherence to ethical standards in conducting security assessments and handling sensitive information.
- Desired Attributes: Team Player: Ability to work collaboratively with cross-functional teams and contribute to a positive team environment. Adaptability: Capacity to adapt to evolving security challenges and technologies.
- Proactiveness: Proactive approach to identifying and addressing potential security issues before they become threats.

Responsibilities & Context

- Penetration Testing: Perform thorough penetration testing on web applications, networks, and systems to identify vulnerabilities and security weaknesses.
- Ethical Hacking: Conduct ethical hacking exercises to simulate real-world attacks and assess the effectiveness of security measures.
- Vulnerability Assessment: Analyze and evaluate security vulnerabilities, provide detailed reports, and recommend remediation strategies.
- Security Audits: Execute comprehensive security audits and assessments to ensure compliance with industry standards and best practices.
- Reporting: Document findings in detailed reports, providing actionable insights and recommendations for improving security posture.
- Collaboration: Work closely with IT and security teams to address vulnerabilities, implement security improvements, and enhance overall security strategies.
- Continuous Learning: Stay updated on the latest hacking techniques, security threats, and industry trends to apply cutting-edge solutions.

Skills & Expertise

CyberSecurity

Information Security

IT Security

Network Security

Penetration testing

Compensation & Other Benefits

- Tour allowance, Performance bonus
- Lunch Facilities: Partially Subsidize
- **Salary Review: Yearly**
- **Festival Bonus: 2**

Workplace

Work from home, Work at office (hybrid)

Employment Status

• Contractual

Job Location

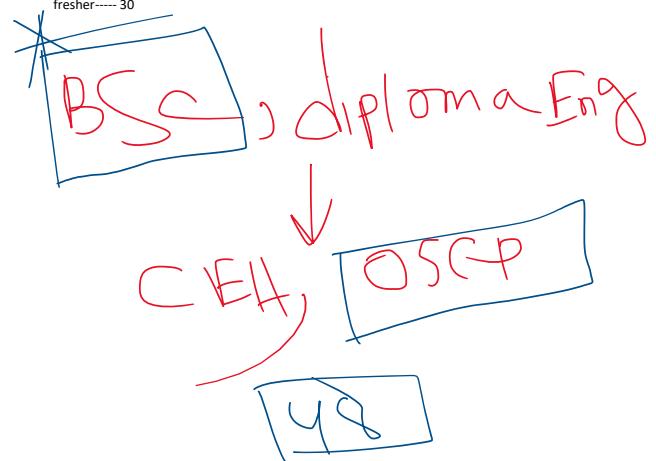
Dhaka (GULSHAN 2)

Source link

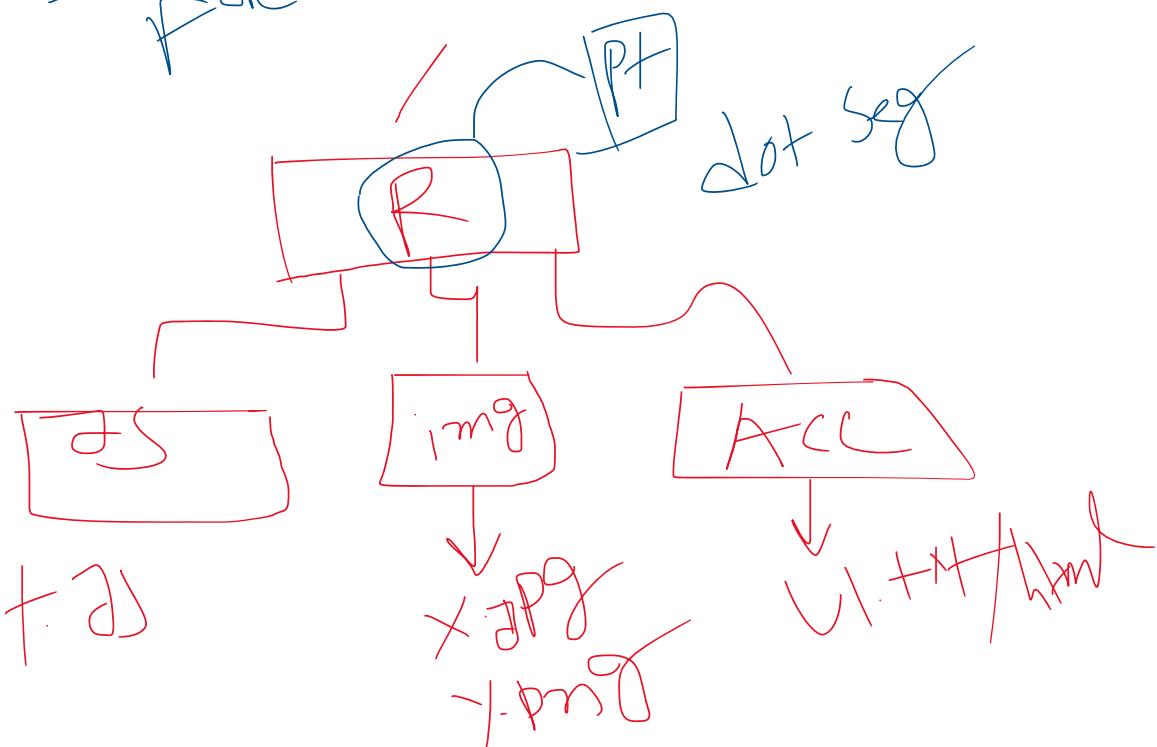
25 ---- 30
26---- 35

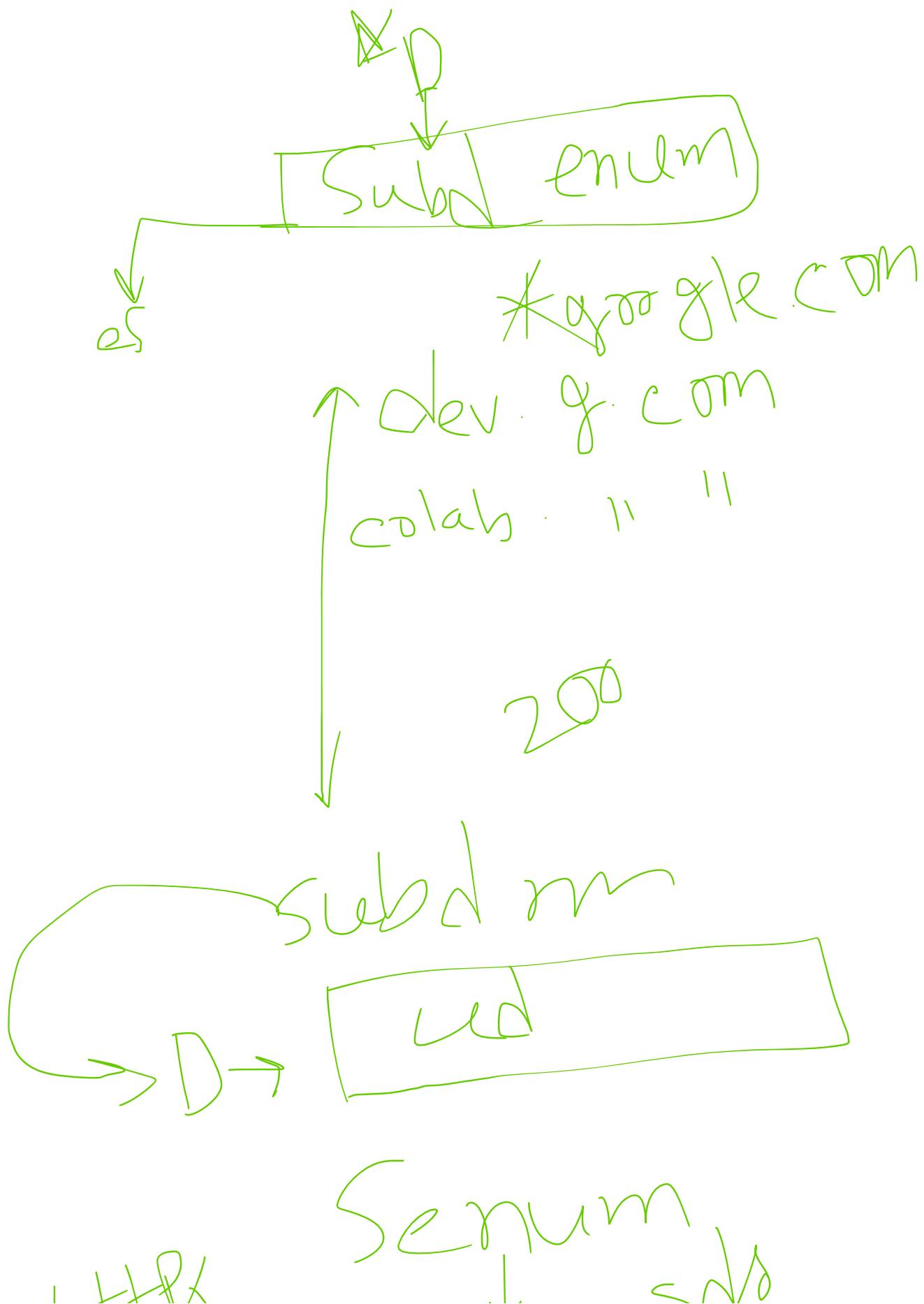
27---- 40

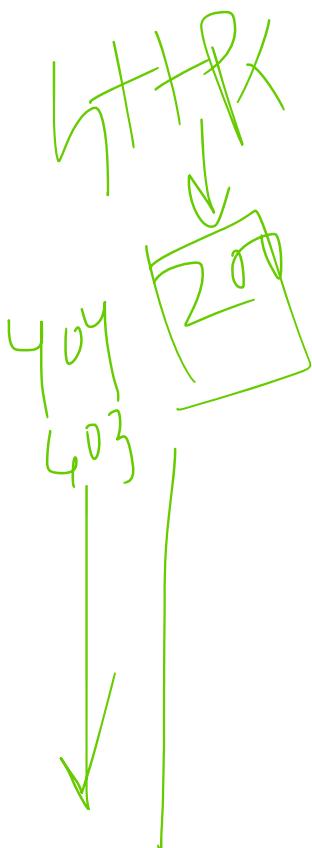
3 years---- 40
 fresher---- 30



~~SOS~~
CCHAT Microtik
✓ E Fortinet



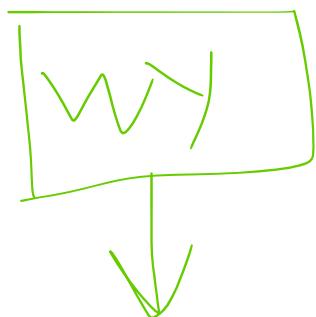




111
SOM SDS

301

Sd



100

3
2
25
111
!



◦ **/blob/ দিয়ে যেটা হয়:**

আপনি GitHub-এর ওয়েবসাইটে কোনো ফাইল দেখছেন। যেমন:

<https://github.com/user/repo/blob/main/file.yaml>

এখানে /blob/ আছে মানে GitHub আপনাকে ওয়েব ব্রাউজারে সুন্দরভাবে ফাইল দেখাচ্ছে, কিন্তু এটা আসলে ওই ফাইলের আসল ডেটা নয়। আপনি যদি এটা wget, curl, বা স্ক্রিপ্ট দিয়ে নামাতে চান, তাহলে নামবে না ঠিকমতো, কারণ এটা HTML পেজ।

◦ **raw.githubusercontent.com দিয়ে যেটা হয়:**

আপনি ওই একই ফাইলের "raw content" বা কাঁচা ডেটা পাচ্ছেন — যেটা আপনি নামাতে পারবেন সহজে।

<https://raw.githubusercontent.com/user/repo/main/file.yaml>

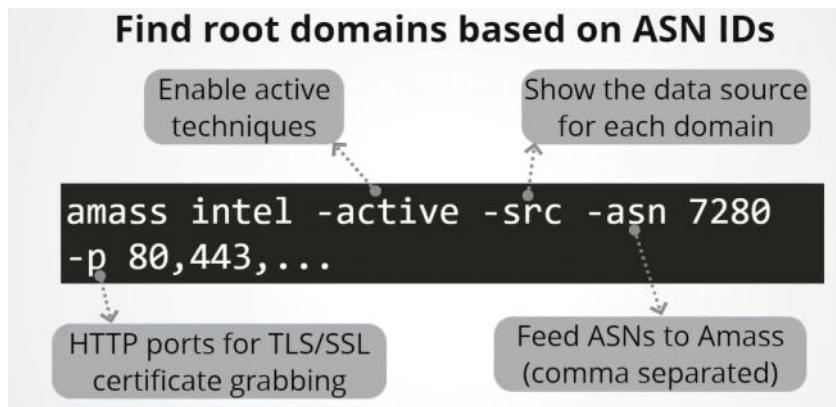
এখানে /blob/ নেই, এবং এটা সরাসরি ফাইলের কনটেন্ট ফেরত দেয় — কোনো ওয়েব পেজ নয়। এটা ব্যবহার হয় wget, curl, বা প্রোগ্রামে ডাউনলোড করার সময়।

amass intel -org Yahoo

🔍 **What Does amass intel -org Yahoo Do?**

The intel mode in Amass is employed for passive information gathering. When you specify -org Yahoo, Amass searches various public data sources to collect details associated with the organization "Yahoo." This can include:

- **Autonomous System Numbers (ASNs):** Identifying ASNs registered to Yahoo.
- **Domain Names:** Discovering domains and subdomains owned by Yahoo.
- **IP Address Ranges:** Finding IP ranges allocated to Yahoo.
- **Network Infrastructure:** Mapping out Yahoo's internet-facing infrastructure



🔍 **amass intel**

এটি Amass টুলের "intel" মোড চালায়। এই মোডের মূল উদ্দেশ্য হলো root domain খুঁজে বের করা বিভিন্ন উৎস থেকে, যেমন: ASN, CIDR, organization name ইত্যাদি।

↳ **-active**

এই অপশনটি দিলে Amass active reconnaissance করে।

অর্থাৎ এটি actively scan করে, যেমন:

ওয়েব সার্ভারে গিয়ে তাদের TLS/SSL certificate থেকে ডোমেইন নাম বের করে।

Public IP-তে সংযুক্ত সার্ভার স্ক্যান করে নতুন root domain বের করতে পারে।

Active scan মানে হলো—এই প্রক্রিয়ায় আপনি লক্ষ্যমাত্রার সার্ভারে সরাসরি রিকোয়েস্ট পাঠাচ্ছেন।

এর ফলে অনেক সময় নতুন ও লুকানো ডোমেইন পাওয়া যায়, যেগুলো শুধু passive ভাবে ধরা পড়ে না।

⚠ Active scan করলে লক্ষ্যমাত্রা aware হতে পারে, তাই এটি সতর্কতার সাথে ব্যবহার করুন।

☛ -src

এই flag দিলে, Amass প্রতিটি পাওয়া domain এর সোর্স (উৎস) দেখায়।

উদাহরণ:

certspotter → যদি domain টি TLS certificate থেকে পাওয়া যায়

whois → যদি WHOIS রেকর্ড থেকে পাওয়া যায়

dnsdb, securitytrails, shodan ইত্যাদি সোর্সগুলো দেখাবে

এটা সাহায্য করে বুঝাতে—কোন সোর্স থেকে কোন ইনফরমেশন এসেছে, যা পরবর্তী বিশ্লেষণে গুরুত্বপূর্ণ।

☛ -asn 7280

এই flag দিয়ে আপনি একটি ASN (Autonomous System Number) উল্লেখ করছেন।

ASN হচ্ছে ইন্টারনেটের বড় বড় নেটওয়ার্কের জন্য নির্ধারিত নম্বর। প্রতিটি ISP, কোম্পানি, ডাটা সেন্টারের নিজস্ব ASN থাকে।

ASN 7280 হলো উদাহরণস্বরূপ, এটি কোন একটি কোম্পানি বা সার্ভিস প্রোভাইডারের IP range প্রতিনিধিত্ব করছে।

Amass এই ASN এর অধীনে থাকা IP range থেকে root domains বের করার চেষ্টা করবে।

◦ আপনি একাধিক ASN ও দিতে পারেন: -asn 7280,13335,32934 (Cloudflare, Facebook ইত্যাদি)

☛ -p 80,443

এই অপশনটি বলে দেয়, Amass কোন কোন পোর্টে স্ক্যান করবে TLS সার্টিফিকেট বা অন্যান্য তথ্য বের করতে।

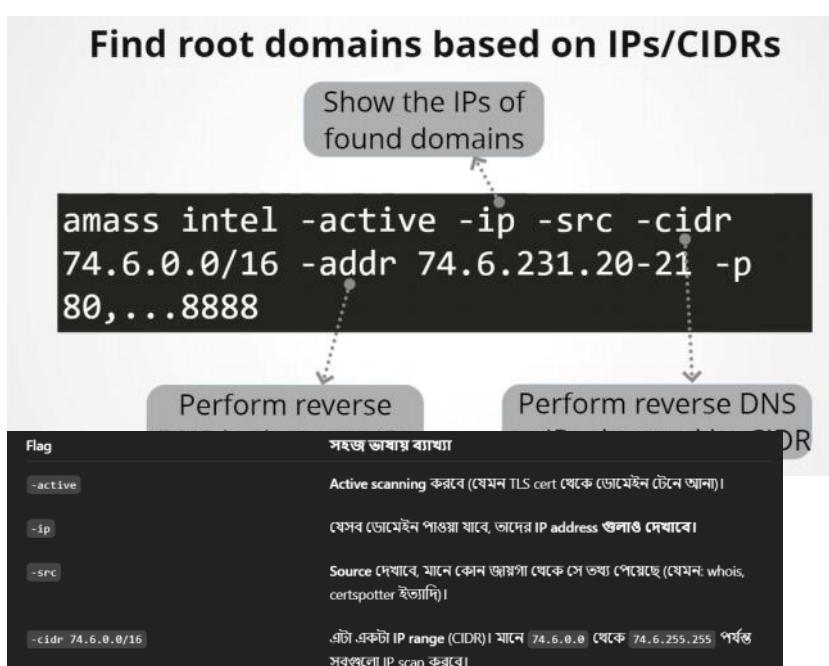
80 → HTTP

443 → HTTPS

8080, 8443, 10443 → অন্যান্য কমন web service পোর্ট

Amass এই পোর্টগুলোতে সংযোগ দিয়ে TLS/SSL সার্টিফিকেট সংগ্রহ করে তাতে থাকা subject alternative name (SAN) থেকে ডোমেইন বা সাবডোমেইন খুঁজে বের করে।

⚠ যদি আপনি শুধু passive দেখতে চান, এই -p ফ্ল্যাগ প্রয়োজন নেই। কিন্তু active recon করতে চাইলে এটা অনেক বেশি কার্যকর।



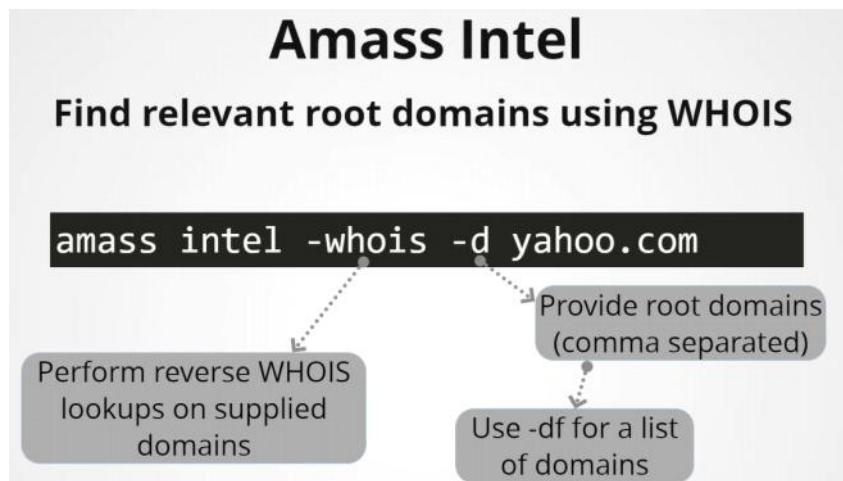
| Flag | সহজ ভাষায় ব্যাখ্যা | OR |
|----------------------|---|----|
| -active | Active scanning করবে (যেমন TLS cert থেকে ডোমেইন টেনে আনা)। | |
| -ip | যেসব ডোমেইন পাওয়া যাবে, তাদের IP address শুলাও দেখাবে। | |
| -src | Source দেখাবে, যানে কোন জয়গা থেকে সে তথ্য পেয়েছে (যেমন: whois, certyspotter ইত্যাদি)। | |
| -cidr 74.6.0.0/16 | এটা একটা IP range (CIDR)। মানে 74.6.0.0 থেকে 74.6.255.255 পর্যন্ত সবগুলো IP scan করবে। | |
| -addr 74.6.231.20-21 | এই দুইটা specific IP address দিলেও খুঁজবে (20 & 21)। | |

এখানে 20-21 মানে একটা ছোট range:

74.6.231.20

74.6.231.21

এগুলোতেই reverse DNS ও TLS probing করবে।



কমাণ্ডটি কী করছে:

amass intel -org "Yahoo! Inc." | cut -d',' -f1 | paste -sd ''
এই পুরো লাইনট মূলত Yahoo! Inc. এর সাথে সম্পর্কিত ASNs (Autonomous System Numbers) বের করে, এবং সেগুলিকে comma-separated list হিসেবে সজার্য।

ডাপে ধাপে ব্যাখ্যা:

◦ amass intel -org "Yahoo! Inc."
এটি Amass কে instruct করছে যে "Yahoo! Inc." নামক organization-এর জন্য Intel gather করো।

এখানে \। ব্যবহার করা হয়েছে কারণ। bash shell-এ বিশেষ চিহ্ন, তাই escape করা হয়েছে।

◦ cut -d',' -f1
amass যেসব তথ্য দেয় সেগুলো comma-separated হয়।

এই অংশ বলে: কমার আগে যেটা আছে(মানে, প্রথম field) শুধু সেটা রাখো — যা হলো ASN।

◦ paste -sd ''
এটি সব ASN গুলোকে এক লাইনে নিয়ে আসে, আর প্রতিটির মধ্যে কমা দিয়ে join করে দেয়।

⚠ সমস্যা কী?

যখন আপনি এইভাবে লিখেন:

amass intel -org "Yahoo! Inc."

তখন bash ! দেখে ভাবে আপনি history থেকে কিছু recall করতে চাচ্ছেন।
যদি সে বুঝতে না পারে, তখন error দেয়:

bash: !: event not found

সমাধান:

1. Escape করুন!

"Yahoo\! Inc."

এখানে \। মানে — “এই ! কে normal character হিসেবে ধরো।”

2. অথবা single quotes ব্যবহার করুন:

'Yahoo! Inc.'

Single quotes দিলে bash ভেতরের special character গুলাকে evaluate করে না।