Bypassing CSRF Protections

A Double Defeat of the Double-Submit Cookie Pattern
• 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)
Cross-site Request Forgery

- Attacker sends payload via victim’s browser
- Browser automatically includes user’s identity
• Simple CSRF protection – no server-side state
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. 

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...
• 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
• 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]

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
• Attacker now controls cookies sent to https://www.example.com/submit

• Attacker’s CSRF cookie sent first due to longer path
• 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">
MAN-IN-THE-MIDDLE ATTACKS
• 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

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

Set-Cookie: _csrf=a; Path=/
+ CSRF attack payload

Browser stores new _csrf cookie and sends CSRF payload

/submit?_csrf=a-GePpmIVNI39L6UB5Hh4t5-mj0
(Cookie: _csrf=a; session=abc...)

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

https://www.example.com

Victim's browser

MiTM

Any Site (HTTP)

HTTP Request

Intercept response and inject plain HTTP request to www.example.com

Modified HTTP Response

Request image from http://www.example.com

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.

Request accepted as csrf token matches cookie

Submission Accepted

https://www.example.com

Victim's browser

MiTM

Any Site (HTTP)

Intercept HTTP request and respond with Set-Cookie header to store long-lived _csrf cookie

While user logged in, attacker forces victim's browser to submit CSRF payload

Attacker has successfully issued CSRF request to https://www.example.com on behalf of victim
• After fixating CSRF cookie, attacker can create successful CSRF payload
• 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

• 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

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

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

// load csrf in cookie mode
app.use(csrf({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});
});
```
Body and query parameters checked first!
Exploit Default Value Function

OWASP
The Open Web Application Security Project

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

+ 

CSRF form sent from attacker, with pre-calculate CSRF token for the secret 'a'!

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

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

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

//load csrf in cookie mode - with cookie signing and custom value function
app.use(csrf({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

• 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
Questions?

@securitybits