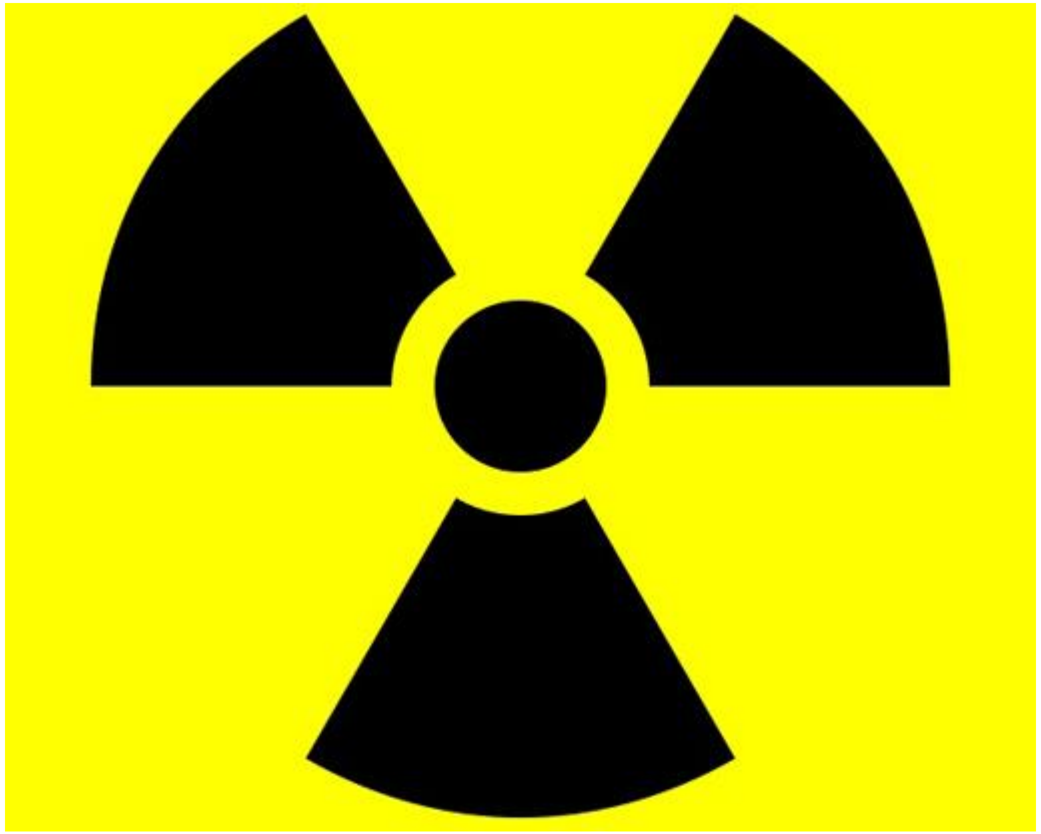
Client-Side JavaScript Security

Null Bachaav Session

Ahamed Nafeez
@skeptic_fx



The contents of this slides doesn’t cover everything that we are gonna do today.

Let us set some goals for today

- Stick with everything client side.
- Talk less, server side stuff.
- Juicy EcmaScript primer.
- DOM XSS and defenses.
- Don’t talk about Nodejs security.
- Don’t even go near regular XSS, CSP, postMessage, IFrame sandbox etc.
- See funny and scary things in browser parsing.
- We will try to switch gears and randomly visit cool issues.

ECMAScript

- No. Lets not talk about JavaScript history, ES3, Netscape again.
- Insanely Dynamic.
- Functional.
- Prototypal inheritance.
- Everyone can write JavaScript.
- Very few understand and know what JavaScript really is.
- Native vs Host Objects.

Insanely Dynamic

- Dynamically typed.

  ```
  var a = 1; // Integer
  a = “abc”; // String
  ```

- Object base – Think everything is an Object. Like completely everything.
- The evil eval and its run-time evaluation.

Functional

- Functions are first class citizens.

```
var a = function(){
        doStuff();
} // run-time, anonymous function

function a(){

        doStuff();

} // parse-time, named function


// Now you their difference as well.
```

Prototypal Inheritance

- Classical inheritance - Classes inherit from other classes.
- Prototype-based inheritance – Object inherits from other objects.
- __proto__

Lets see some example.

Native vs Host Objects

- Native Object:
  object in an ECMAScript implementation whose semantics are fully defined by this specification rather than by the host environment.

Date, Math, parseInt, eval

- Host Object:

object supplied by the host environment to complete the execution environment of ECMAScript.

window, document, location, history, XMLHttpRequest, setTimeout etc.

Well, some ECMAScript history won’t hurt anyway

- First Edition, 1997
- Lots of political and bull* reasons happened.
- June 2011, ES 5.1
- ES 6 – Awesome features from python, proxies, collections etc.
  Named ES6 Harmony
- There is some form of ES7 in the works as well.

Since we won’t talk about traditional XSS.

- So you must be knowing what a context is, in XSS.
- HTML, Attribute, Href, JavaScript etc.

- Let us take a certain JS injection example.

```
<script> var a = “injected-data”; </script>
```

- Stripping “ should save us right?

Exercise 1

Did you know? Parsing is buggy

- a = “**</script>**<img src=x onerror=alert(1)”

Browser Parsing

HTML Parser / Renderer vs JavaScript Parser

Global Namespaces

- Closures and variable hoisting in JavaScript are important concepts.
- Lets see a few examples.
- Globally assigned variables (without var statement) leaks in to the global window Object in case of the browser DOM.
- Examples again.

Exercise 2

Solution

```
name=‘youWon’;
```

DOM XSS

- Switching gear to another variant of XSS.

Classic Twitter URL:

https://twitter.com/#!/hasegawa

Becomes:

https://twitter.com/hasegawa

```
( function(g){
var a=location.href.split("#!")[1];
if(a){
g.location=g.HBR=a;
}
}
)(window);
```

Credits to Stefano Di Paolo for this find on Twitter.

DOM XSS Fix 1

- http://twitter.com/#!javascript:alert(1)
- Will be executed since javascript: is a pseudo-schema
- Fix:

```
(function(g){
var a=location.href.split("#!")[1];
if(a){
g.location=g.HBR=a.replace(“:”,"",”g”);
}
}
)(window);
```

DOM XSS Fix 2

- http://twitter.com/#!javascript::alert(1)
- Pretty easy. Wasn’t it?
- Fix:

```
(function(g){
var a=location.href.split("#!")[1];
if(a){
g.location=g.HBR=a.replace(/:/gi,"");
}
}
)(window);
```

DOM XSS Fix 3

- **Open Redirect:** http://twitter.com/#!//www.lol.com
- **JS Exec on IE:** http://twitter.com/#!javascript&x58;alert(1)
- **Final Fix:**

```
(function(g){
var a=location.href.split("#!")[1];
if(a){
g.location.pathname=g.HBR=a;
}
}
)(window);
```

Sources & Sinks

- Go to the DOMXSS Wiki,

Am not gonna talk about them in detail now.
https://code.google.com/p/domxsswiki/wiki

- Sources: location et al, . .

- Sinks: innerHTML, eval, $(), . . .

DOM XSS Filters

- Not our grandpa's old XSS filters we see in WAFs and server side logic.
- These are things which sits in between the DOM sources and sinks.
- Whitelisting, Encoding, Not using unsafe functions in between the sources and sinks.
- Key point: Any source which almost directly enters an eval is exploitable most of the times.

Whitelisting

- Most of the times, when we deal with a source like location. We are talking about a direct injection point. And if our logic needs to make use of that value, make sure we have a safe whitelist of values that we accept.

- We should forget the term, blacklisting. Its better for me, you and the rest of the humanity.

Safe filter functions

- The context in which you are trying to filter the values from the source matters.
- We are talking about the contexts within JavaScript itself.
- Bad Filters – innerHTML, textContent, createTextNode. Don’t misuse them.
- You’re Probably Misusing DOM text methods:

http://benv.ca/2012/10/4/you-are-probably-misusing-DOM-text-methods/

Encode them all well.

```
function escapeHtml(str) {
    return String(str)
        .replace(/&/g, "&amp;")
        .replace(/</g, "&lt;")
        .replace(/>/g, "&gt;")
        .replace(/"/g, "&quot;")
        .replace(/'/g, "&#039;")
        .replace(/\//g, "&#x2F;")
}
```

That’s it for now

Thanks

D