JavaScript, security, and the case for feature simplicity
About Me

- Natalie Silvanovich AKA natashenka
- Project Zero member
- Previously did mobile security on Android and BlackBerry
The Problem

Firefox 0-day in the wild is being used to attack Tor users

Sundown exploit kit weaves Edge hack hole
What is a JS vulnerability?

Developer writes code in JavaScript

JavaScript is hosted on a website

Browser downloads JavaScript

Script engine executes JavaScript

User visits website

User has wonderful Web experience

Google
What is a JS vulnerability?

Malicious developer writes code in JavaScript, including malicious native code.

JavaScript is hosted on a (malicious?) website.

User visits website.

Browser downloads JavaScript.

Script engine hits a bug processing the JS and executes malicious native code.

User's computer is compromised.
The (Technical) Problem

- JavaScript engines contain memory corruption bugs that malicious developers can hit on purpose.
- These bugs can be used to execute native code in the browser.
- Fixing and preventing these bugs is one part of improving browser security.
My Work

- Reported 20+ exploitable bugs in browsers
- Tracked public JavaScript vulnerabilities
- Most vulnerabilities involved one of two features
Array[@@species]

class MyArray extends Array {
    static get [Symbol.species]() { return Array; }
}

var m = new MyArray(1, 2, 3);
var s = m.slice(1);

s instanceof MyArray; // false
Array[@@species] Vulnerabilities

- **CVE-2017-5030**: Out-of-bounds read in V8 Array.concat (Chrome)
- **CVE-2017-8634**: Overflow in Array.concat (Edge)
- **CVE-2017-7064**: appendMemcpy uninitialized memory copy (Safari)
- **CVE-2016-7190**: Heap Overflow in Array.map (Edge)
- **CVE-2016-7200**: Heap Overflow in Array.filter (Edge)
- **CVE-2017-0134**: Overflow in Array.concat (Edge)
- **Bug 725865**: Array Species Optimization Issue (Chrome)
Sample Array[@@species] Bug (CVE-2016-7200)

class dummy{
    constructor(){ return [1, 2, 3]; }
}
class MyArray extends Array {
    static get [Symbol.species](){ return dummy; }
}
var a = new MyArray({}, [], "natalie", 7, 7, 7, 7, 7);
function test(i){ return true; }
var o = a.filter(test);
Why does Array[@@species] have so many bugs?

- Violates developer expectation by adding call to user code in new location
  - Has side effects
- Modified functions were generally written years ago
Array[@@species] modification rate
Array Index Accessor

```javascript
var o = {};

function f(){ return 10; }

Object.defineProperty(o, "0", { get: f, set : f });

0[0]; // 10
```
Sample Array Index Accessor Bug (CVE-2016-7189)

```javascript
var t = new Array(1,2,3);
    Object.defineProperty(t, '2', {
        get: function() {
            t[0] = {};
            for(var i = 0; i < 100; i++){
                t[i] = {a : i};
            }
            return 7;
        }
    });
var s = [].join.call(t);
```

Sample Inheritance Bug (CVE-2017-2447)

```javascript
var ba;
function s(){
    ba = this;
}
function dummy(){
}
Object.defineProperty(Array.prototype, "0", {set : s });
var f = dummy.bind({}, 1, 2, 3, 4);
ba.length = 100000;
f(1, 2, 3);
```
Why Do Array Index Accessors Have So Many Bugs?

- Violates developer expectations by adding call to user code in new location
- Affects methods without code changes
- Can affect Array.prototype and newly created Arrays
  - Increased JavaScript implementation of host functions is a factor
- Requires a lot of code to implement
Usage

Percentage of page views that use this feature

% page loads

Date

11/17  12/17  12/17  01/18  01/18  01/18
“We designed everything with the assumption that we could implement it perfectly”

-- TC39 committee member
Change is hard

- Logically, the specification is very reliant on these features
- Would require large code changes
- Would break websites
It’s not just JavaScript

- Almost every piece of software Project Zero has looked at has unnecessary attack surface
- Reducing attack surface is the most effective way of reducing vulnerabilities
  - If it doesn’t exist, it doesn’t have any bugs
Avoiding complexity in feature design

- Consider complexity and attack surface versus usage in design
- Be willing to deprecate features
- Have a deprecation plan
- Make sure all code is still relevant
Avoiding complexity in feature design

- Extend analysis to third-party components
- Consider complexity in branching, build environment in patch cycles
- Reduce the privileges of error-prone code
Conclusions

Only you can reduce attack surface