Google

A Story About JavaScript

Natalie Silvanovich
May 21, 2020

About Me

- Natalie Silvanovich AKA natashenka
- Project Zero member
- Reported 100+ vulnerabilities in JavaScript and Flash over the past 5 years

LangSec

Ambiguity of message/protocol specification is insecurity; ad hoc parsing is an engine of exploitation; overly complex syntax can make judging security properties of input impractical or even undecidable.

The State of JavaScript

- In 2020, there have been:
  - 7 serious vulnerabilities in V8, one used in the wild
  - 3 serious vulnerabilities in SpiderMonkey, 2 exploited in the wild
  - 4 serious vulnerabilities in JSC
  - Does not include internally discovered bugs, or bugs in other features where JavaScript contributed
  - **It is currently May**

The State of JavaScript

- Also
  - JavaScript engines have millions of lines of code
  - Dozens of commits per day
  - Performance is a challenge

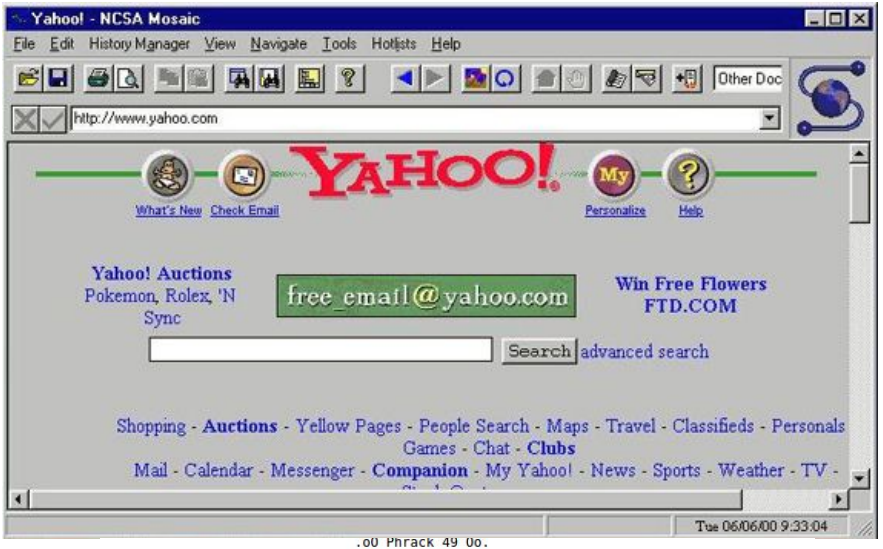
Why are there so many bugs in JavaScript?

What went wrong?

How can we do better?

“The story so far: In the beginning the JavaScript was created. This has made a lot of people very angry and been widely regarded as a bad move.”

-- Douglas Adams



.oO Phrack 49 Oo.

Volume Seven, Issue Forty-Nine

File 14 of 16

BugTraq, r00t, and Underground.Org
bring you

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Smashing The Stack For Fun And Profit
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

by Aleph One
aleph1@underground.org

`smash the stack` [C programming] n. On many C implementations
it is possible to corrupt the execution stack by writing past
the end of an array declared auto in a routine. Code that does
this is said to smash the stack, and can cause return from the
routine to jump to a random address. This can produce some of
the most insidious data-dependent bugs known to mankind.
Variants include trash the stack, scribble the stack, mangle
the stack; the term mung the stack is not used, as this is
never done intentionally. See spam; see also alias bug,
fandango on core, memory leak, precedence lossage, overrun screw.

Introduction
~~~~~~~~~~~~
~~~~~~~~~~~~

JavaScript History

1995 -- Brendan Eich creates JavaScript (originally Mocha and then LiveScript) and it is released in Netscape

1996 -- IE implements JScript, an implementation of JavaScript

1997 -- ECMAScript 1 released

1998 -- ECMAScript 2 released

1999 -- ECMAScript 3 released

ECMAScript History

2008 -- ECMAScript 4 abandoned

2009 -- ECMAScript 5 released

2011 -- ECMAScript 5.1 released

2015 -- ECMAScript 6 released

2016 -- ECMAScript 7 released

Weak Typing

- Strong typing was rejected in ECMA 4
    - Consequences for security and performance

Weak Typing

JavaScript

```
var a = “hello”;
var s = a.concat(b);
```

C++

```
void str_concat(Obj this, Obj a){
    IsString(this);
    IsString(obj);
        ...
}
```

Weak Typing

- Type confusion occurs when a type is not checked correctly
  - Highly exploitable bug type
- For vulnerabilities reported in 2020:
  - 3/7 V8 bugs are type confusion
  - 2/3 SpiderMonkey bugs are type confusion
  - 2/4 JSC bugs are type confusion
- ~5% of Flash vulnerabilities were in ES4 engine

Weak Typing

- Affects performance and maintainability
  - Fundamentally, weak typing requires extra checks
  - Browser JIT engines reduce checks at the cost of development time, code complexity and risk of introducing bugs

ECMAScript 6

- ES6 introduced features that caused a disproportionate number of bugs

Array.species

“But what if I subclass an array and slice it, and I want the thing I get back to be a regular Array and not the subclass?”

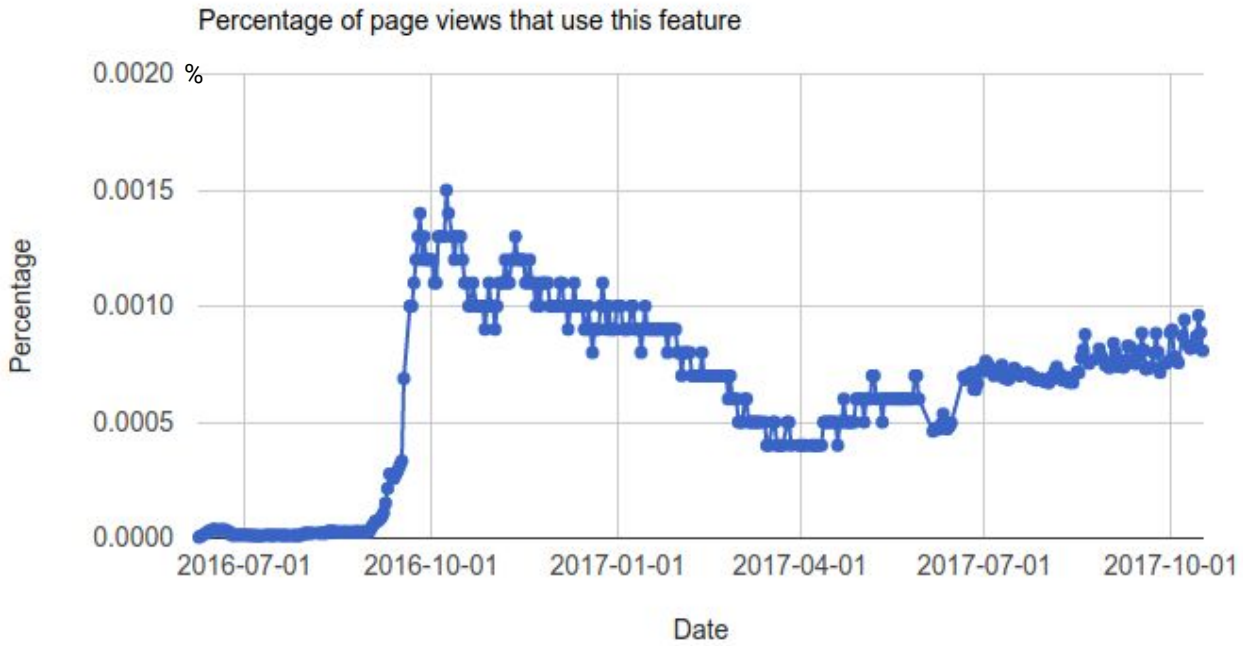
```
class MyArray extends Array {
   static get [Symbol.species]() { return Array;}
}
```

- Easily implemented by inserting a call to script into *every single* Array native call

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)

Array[@@species] modification rate



Array Index Accessors

```
var t = [1, 2, 3];
  Object.defineProperty(t, '2', {
    get: function() {
      return 7;
    }
  });
```

Array Index Accessor Bugs

- **Bug 386988:** Out-of-bounds access vulnerability in Array.concat() (Chrome)
- **CVE-2016-5129:** V8 OOB Read in GC with Array Object (Chrome)
- **CVE-2016-3386:** Stack Overflow in Spread Operator (Edge)
- **CVE-2016-7202:** Overflow in Array.reverse (Edge)
- **CVE-2016-7194:** Info Leak in Function.apply (Edge)
- **CVE-2016-7194:** Proxy Memory Corruption (Edge)
- **CVE-2016-7189:** Info Leak in Array.join (Edge)
- **PZ 1230:** Uninitialized memory reference in arrayProtoFuncSplice (Safari)
- **CVE-2016-7203:** Heap Overflow in Array.splice (Edge)

Array Index Accessor Bugs requiring Array Inheritance

- **PZ 1230:** uninitialized memory reference in arrayProtoFuncSplice (Safari)
- **CVE-2016-1646:** v8 Array.concat OOB access (Chrome)
- **CVE-2016-1677**: type confusion lead to information leak in decodeURI (Chrome)
- **CVE-2017-0141**: memory corruption in Array.reverse (Edge)
- **CVE-2017-2447**: Out-of-bounds read when calling bound function (Safari)
- **CVE-2017-6980**: arrayProtoFuncSplice doesn't initialize all indices (Safari)
- **CVE-2017-7005**: JSGlobalObject::haveABadTime causes type confusion (Safari)
- **CVE-2017-6984:** heap buffer overflow in Intl.getCanonicalLocales (Safari)

Array Index Accessor usage

- ~10% of webpages use array index accessors, the majority due to jQuery

What makes JSC have a bad time?

```
void JSGlobalObject::haveABadTime(VM& vm)
{
    ASSERT(&vm == &this->vm());

    if (isHavingABadTime())
        return;
```

What makes JSC have a bad time?

```
var t = Array.prototype;
  Object.defineProperty(t, '2', {
    get: function() {
      return 7;
    }
  });

var a = [];
```

Why did these features cause so many bugs?

- Violates developer expectations by adding call to user code in new location
- Affects methods without code changes
- Requires a lot of code to implement
- Vastly increases the code’s range of behavior

I guess we created these features without thinking of how we were going to implement these features

-- ES Committee member

Conclusions

- JavaScript is an excellent example of how failing to design with implementation in mind leads to security and other problems
- It is probably too late to fix JavaScript, but …
  - What ‘JavaScripts’ are we creating today?
  - How can we make incremental progress on software that is already implemented?

Questions and Discussion

http://googleprojectzero.blogspot.com/

@natashenka
natashenka@google.com