Comparing Javascript Engines

Xiang Pan, Shaker Islam, Connor Schnaith

Background: Drive-by Downloads

1. Visiting a malicious website
2. Executing malicious javascript
3. Spraying the heap
4. Exploiting a certain vulnerability
5. Downloading malware
6. Executing malware

Background: Drive-by Downloads

1. Visiting a malicious website
2. Executing malicious javascript
3. Spraying the heap
4. Exploiting a certain vulnerability
5. Downloading malware
6. Executing malware

Background: Drive-by Downloads

```
var obj = new Object();
obj.__proto__.__defineGetter__("a", function () {
                                     this.__proto__ = null;
                                     gc();
                                     return 0;
                                    });
obj.a;
```

Figure 2: CVE-2009-1833: Malicious JavaScript Codes that can Trigger a Mozilla Firefox JavaScript Engine Vulnerability

Background: Drive-by Downloads

```
var obj = new Object();
obj.__proto__.__defineGetter__("a", function () {
                                      this.__proto__ = null;
                                      gc();
                                      return 0;
                                    });
obj.a;
```

Setup: Making the prototype null while in the prototype creates a pointer to something random in the heap.

Background: Drive-by Downloads

```
var obj = new Object();
obj.__proto__.__defineGetter__("a", function () {
                                      this.__proto__ = null;
                                      gc();
                                      return 0;
                                    });
obj.a;
```

Environment: gc( ) is a function call specific to Firefox, so the attacker would want to spray the heap with an exploit specific to firefox.

Background: Drive-by Downloads

```
var obj = new Object();
obj.__proto__.__defineGetter__("a", function () {
                                     this.__proto__ = null;
                                     gc();
                                     return 0;
                                    });
obj.a;
```

Obfuscation: If the browser executing the javascript it firefox,the code will proceed to the return statement. Any other browser will exit with an error due to an unrecognized call to gc( ).

Background: Drive-by Downloads

```
var obj = new Object();
obj.__proto__.__defineGetter__("a", function () {
                                     this.__proto__ = null;
                                     gc();
                                     return 0;
                                    });
obj.a;
```

Download: The return will be to a random location in the heap and due to heap-spraying it will cause shell code to be executed.

Background: Goal of Our Project

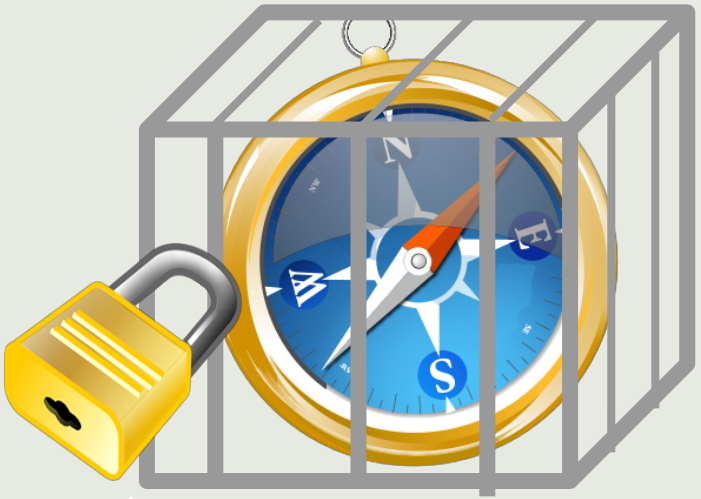
- The goal is to decode obfuscated scripts by triggering javascript events
- The problem is when triggering events, some errors, resulting from disparity of different engines or some other reasons, may occur and terminate the progress
- We need to find ways to eliminate the errors and therefore generate more de-obfuscated scripts

Ex 1

```
<script>
    function f(){
        //some codes
        gc();
        var x=unescape('%u4149%u1982%u90 [...]'));
    eval(x);
    }
</script>
```

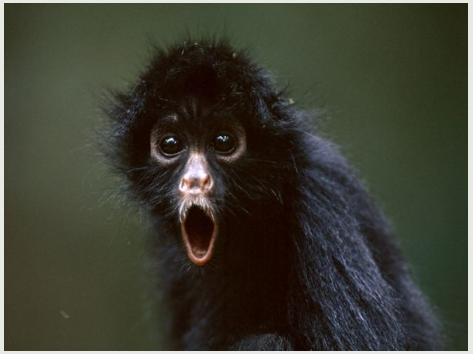
Project Overview - Part One

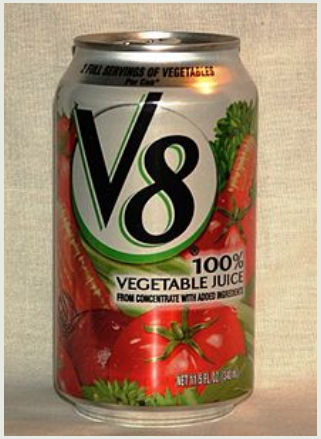
- Modify WebKit engine so that it can generate error informations.
- Modify WebKit engine so that it can pre-define any functions or pre-include any libraries.
- Analyze the errors resulting from executing more than 140,000 malicious scripts.



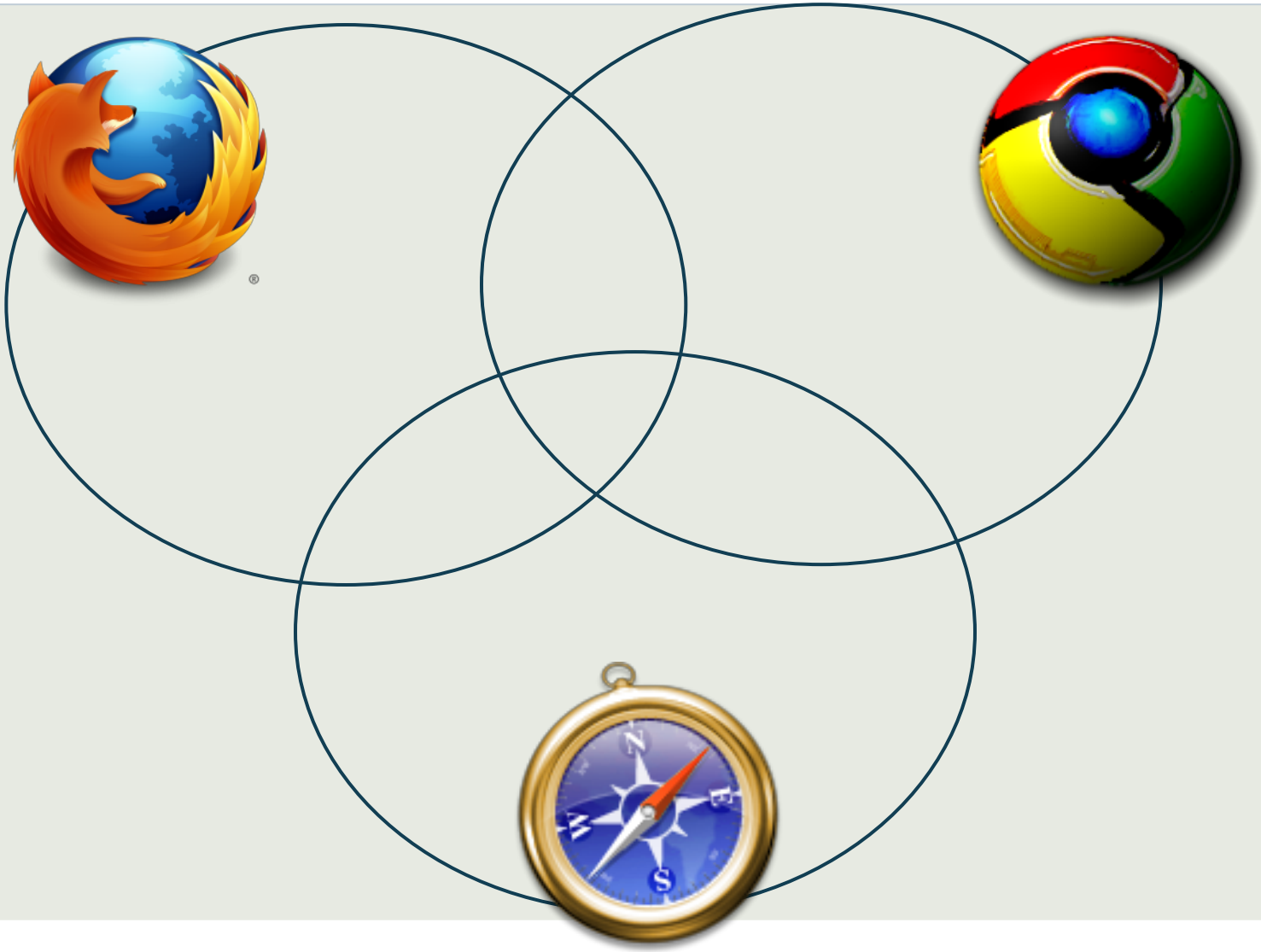
Project Overview - Part Two

- Create Spidermonkey (Firefox javascript engine) and V8 (Chrome javscript engine) test suites and run them in Webkit
- Determine the cross-compatibility of Spidermonkey, V8 and Webkit
- Compare the results to the javascript in the malicious files





Project Overview - results



Part One - malicious file analysis

1. Modify webkit to make it output error informations:

```
fff6609a91bc854fa07d30268385ed12.html.err
fff731237ff61a9d95b92328a95d44c4.html.err
fff750d1a82b3ad5f2ed32b6466376ff.html.err
fff7b12b055c8c05616b1749022bc40a.html.err
fff7e48ffdac996bee146675119cb88c.net.err
fff84f0baae52d41205ad0d73ec065d9.html.err
fff88ca64dcf42801343e24d29be2d59.html.err
fff8b66ea72d0b73ea2a1504f172a468.html.err
fff97e2e119e62537895b05e9abc31f6.html.err
fffa6290aea5deb0c3b3df55c56cdfe6.html.err
fffabe00a636a491588beefe6ba53034.html.err
```

```
Syntax Error: JSON Parse error: Unexpected identifier "cb"
line number: 10
{return JSON.parse(a);}

Syntax Error: JSON Parse error: Unexpected identifier "cb"
line number: 151
{try{return window.JSON.parse(a)}catch(c){return _.q}}

Syntax Error: JSON Parse error: Unexpected identifier "cb"
line number: 151
{try{return window.JSON.parse(a)}catch(c){return _.q}}
```

Part One - malicious file analysis

2. Modify webkit to make it output can pre-define the functions/objects or pre-include relevant libraries.

Before loading any pages, WebKit will read an assigned file and execute the scripts in it. This file may include the libraries and objects/functions we want webkit to define or include.

Part One - malicious file analysis

2. Analyze the errors from executing JavaScript events of 142338 malicious pages.

Syntax Error	Unexpected token '<'	Possible due to nested script tag
	JSON Parse error: Expected '}'	Possible due to nested script tag
Type Error	undefined' is not an object/function	perhaps is the consequence of other reference error
	null' is not an object	
	qwtqwt'/a/b is not a function	Perhaps the author try to invoke some undefined function
Reference Error	Can't find variable: loadComments	It is possible this function is defined in "components/comments/js/comments.js", but this file has not been successfully imported
	Can't find variable: addComments	It is possible this function is defined in "components/comments/js/comments.js", but this file has not been successfully imported
	Can't find variable: $	JQuery has not been successfully downloaded
	Can't find variable: ActiveXObject	ActiveXObject is only defined in IE
	Can't find variable: CollectGarbage	This is a JScript global object
	Can't find variable: chgBg	DHTML Menu Studio library has not been successfully imported
	Can't find variable: SWFObject	FLASH plugin has not been installed
	Can't find variable: asdas/qwtqwt/FB/a2/b/twttr/asdvds	
	Can't find variable: MM_preloadImages	These function seems to be defined by Macromedia
	Can't find variable: MM_swapImgRestore	
	Can't find variable: MM_swapImage	
	Can't find variable: write_ref	
	Can't find variable: check_colors_picked	This function seems to emerge only in malicious pages. It's definition ma be obfuscated when triggering the events

Part two - test suite

- Automatically put contents of .js files into .html files
- Modify webkit scripts to simplify testing process and print error messages
- Run scripts that take all .js.html files as input and output error messages for each file

Part Two - test suites

```
// Flags: --expose-gc

// Test that safepoint tables are correctly generated for apply with
// arguments in the case where arguments adaptation is needed.

function f(x, y) {
  if (x == 149999) gc();
  return x + y;
}

function g() {
  f.apply(this, arguments);
}

for (var i = 0; i < 150000; i++) {
  g(i);
}
```

Part Two - test suites

QtTestBrowser

Starting webkit launcher, running against the built WebKit in /home/xpan/WebKit-r10xxxx/WebKitBuild/Release/lib...

size0

Reference Error: Can't find variable: gc

throw exception

ReferenceError: Can't find variable: gc

line number: 34

```
{
  if (x == 149999) gc();
  return x + y;
}
```

Current State

- SpiderMonkey
  - Installed SpiderMonkey engine with javascript shell
  - Gathered test suites from various sources specific to SpiderMonkey compatibility
  - Ran tests manually in shell and automatically in SpiderMonkey to verify compatibility
  - Next up… run test suites in Webkit

Current State

- V8
  - Installed V8 engine with javascript shell
  - Modified scripts in order to test effectively and print detailed error messages
  - Modified test suites from javascript to html so they can be run by WebKit
  - Next up… run modified scripts and analyze error messages

Next Steps

- More testing still needs to be done to get a comprehensive comparison of Webkit to Spidermonkey and V8
- Due to a large amount of malicious files (over a million!), we will search for specific terms to try and find javascript code of interest
- After complete analysis of the malicious scripts, we will comprise a venn-diagram type comparison to display the differences in each engine's vulnerabilities
- Finally, we will hypothesize as to the defensive techniques unique to each engine based on their vulnerability differences