Defending Computer Networks

Lecture 17: Javascript/Web Drive-bys

Stuart Staniford
Adjunct Professor of Computer Science
Logistics

• Quiz 2: Tuesday, November 4th
• Project Milestone 1: Friday, November 7th
Vulnerability in widely used 'strings' utility could spell trouble for malware analysts

One of the first things a malware analyst does when encountering a suspicious executable file is to extract the text strings found inside it, because they can provide immediate clues about its purpose. This operation has long been considered safe, but it can actually lead to a system compromise, a security researcher found.

String extraction is typically done using a Linux command-line tool called strings that's part of GNU Binutils, a collection of tools for binary file analysis and manipulation available by default in most Linux distributions.

Google security engineer Michal Zalewski was recently running a type of vulnerability testing known as fuzzing against a library called libbfd (the Binary File Descriptor library) that sits at the core of GNU Binutils and is used for file format parsing. Fuzzing is the act of providing unexpected input to an application like libbfd in order to trigger potentially exploitable behavior.

What Zalewski found was, in his own words, "a range of troubling and likely exploitable out-of-bounds crashes due to very limited range checking." These are the kinds of errors that can lead to arbitrary code execution.

Assigned Reading

Main Goals for Today

• Micro-tour of Javascript
• Web-client attacks
HTML

• Main markup language of the web
• text/html is most common type over HTTP
• Tag based formatting

<!DOCTYPE html>
<html>
<head>
  <title>HTML Example</title>
</head>
<body>
  <p><b>Hello...</b> <a href="http://www.cnn.com/">world!</a></p>
</body>
</html>

Try it – load into browser
Some other tags of interest

• `<table>
  – `<tr>`<th>Column 1</th><th>Column 2</th>`</tr>
  – `<tr>`<td>Data A</td><td>Data B</td>`</td>`
• `</table>`

• `<ul>`<li>Bullet 1</li><li>Bullet 2</li>`</ul>`

• `<img src = “http://foo.com/bar.jpg”>`

• `<iframe src = “http://foo.com/bar.html” width =20 height=40/>`
JS Inclusion in HTML

- `<script> js – blah - blah </script>`
  - Technically should be
  - `<script language = “javascript”>`
- `<script src = “foo.js”>`
- These are interpreted/run at page load time
- In tag attributes:
  - `<button type="button" onclick=“myJSFunc()">Button Name</button>`
  - onmouseover, onkeypress, dozens more events that can trigger interpretation/execution of additional js
Let’s have a look

• Developer view of www.cnn.com
Some basics of syntax

• Variable declarations
  – var x; // Now x is undefined
  – var x = 5; // Now x is a Number
  – var x = "John"; // Now x is a String

• Loose dynamic typing a la Perl etc

• All the usual C operators: +, -, ++, &&, ...

• + on strings is concatenation
  – “foo” + “bar” == “foobar”
  – “foo”+5 == “foo5”
Scoping Example

<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Example</title>
</head>
<body>
  <script>
    var foo = "Hello world";
  </script>
  <p><b>Hello...</b> <a href="http://www.cnn.com/" onmouseover="alert(foo)" onmouseout="alert('Javascript Example')">world!</a></p>
</body>
</html>
JavaScript Arrays

• var cars=["Saab","Volvo","BMW"];
  – cars[0] == “Saab”
  – cars.length == 3

• Arrays can be returned from functions and passed to functions
Control Structures

• if(i<5) {foo code} else {bar code}
• for (var i=0;i<N;i++) { blah; blah;}
• while (i < 5) {blah; blah;}
• switch(n) {
  – case 1: blah;break;
  – case 2: blah; break;
  – default: blah}
Object Orientation in JS

- Objects are like hashes/dictionaries

```javascript
var person = {
  firstname: "John",
  lastname: "Doe",
  id: 5566
};

- person.id == 5566

- Everything is an object, and many standard methods available
  - var foo = "bar";
  - foo.length == 3
  - foo.substring(0, 1) == "ba"
```
Accessing the DOM from JS

• Given `<p id="intro">Hello world.</p>`
  – var x=document.getElementById("intro");
  – var y = document.getElementsByTagName("p")
    • y is now an array of all the `<p>` elements
    • for(var i=0; i<y.length; i++)...
  – x.innerHTML = “Goodbye.”
    • Will replace “Hello world” with “Goodbye”
  – document.createElement("p");
function spray_heap() {
    var chunk_size, payload, nopsled;

    chunk_size = 0x80000;
    payload = unescape("<PAYLOAD>");
    nopsled = unescape("<NOP>");
    while (nopsled.length < chunk_size)
        nopsled += nopsled;
    nopsled_len = chunk_size - (payload.length + 20);
    nopsled = nopsled.substring(0, nopsled_len);
    heap_chunks = new Array();
    for (var i = 0 ; i < 200 ; i++)
        heap_chunks[i] = nopsled + payload;
}
Heap Sprays

Before spray

Free

300 Mb

200 Mb

100 Mb

0 Mb

fragmentation

After spray

Free

Consecutive chunks of nops + shellcode

Spray until you reach predictable location

© 2011 - www.corelan.be

Used

Sprayed (Used)

Used

Used
Sample Browser Exploit

• This is a famous IE exploit used as 0day
  – To compromise Google and many others
  – By Chinese PLA

• We will walk through

• http://www.exploit-db.com/exploits/11167/
Protecting Yourself

• Up-to-date
  – OS
  – Browser
  – Plugins
• *BSD > Linux > Mac OS > Windows
  – Not inherently more secure, just less attacked
• Click-to-play
• AV (sort of)
Javascript Obfuscation

- Javascript has things like
  - `eval()`
  - `document.write()`
- Can create code on the fly and execute it
- So initial appearance of code and what finally executes may be very very very different
Sample Obfuscated Javascript

```javascript
var k="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";
function se97a(s){var o="";var c1,c2,c3;var e1,e2,e3,e4;var i=0;s=s.replace(/[^A-Za-z0-9\+/\=]/g,""');do{e1=k.indexOf(s.charAt(i++));e2=k.indexOf(s.charAt(i++));e3=k.indexOf(s.charAt(i++));e4=k.indexOf(s.charAt(i++));c1=(e1<<2)|(e2>>4);c2=((e2&15)<<4)|(e3>>2);c3=((e3&3)<<6)|e4;o=o+String.fromCharCode(c1);if(e3!=64){o=o+String.fromCharCode(c2);}if(e4!=64){o=o+String.fromCharCode(c3);}}while(i<s.length);return o;}

eval(se97a("ZnVuY3Rpb24gYXNhcyhzZGFzKSB7dmFyIG9zPSI0Zhcibjcz1NYXRoLmNiaWwoc2Rhcy5sZW5ndGvMik7Zm9yKGk9MDtpPHNzO2krKyl7dmFyIGNrPXNkYXMuc3Vic3RyaW5nKGkqMiwoaSsxKSojKXtvArPSTdHJpbmccZnJvbUNoYXJDb2RlMDM3KStjazt9cmV0dXJulHVuZXNjYXBIKG9zKTt9");document.write(se97a("asas"));

(3 more pages)
It’s actually even worse

• Polymorphism
  – Servers can generate different obfuscation of underlying exploit with every HTTP response

• Obfuscation widely used legitimately
  – Intellectual property protection

• So how to detect on wire?
  – Snort-style signatures need not apply...
Process Caveats

- This is an account of work done for a commercial vendor (FireEye, SV startup).
  - Was Chief Scientist until Feb.

- Some restrictions apply.
Pre-Existing Product

- Designed to detect zero-day worms (internal spread)
- Phase I heuristics: port-scan detection
- Worked technically, but not as a value proposition
- Plug into core vs edge network
Problem Statement (I)

- Typical enterprise egress speed is 100Mbps - 10Gbps
Problem Statement (II)

- Heuristics must run fast (line rate)
  - Taken to mean must be single-pass
  - Multithreaded
- 1 in $10^6$-$10^7$ http responses is bad.
- VM bandwidth limited – can only afford to run 1 in $10^3$-$10^4$ responses in VM.
  - This sets FP rate allowed in heuristics
  - FN rate is as little as possible.
  - So have to be fairly discriminating
  - VM gets us the other $10^3$-$10^4$ factor of discrimination