Manipulating The Manipulator

destroying browser-based memory corruption exploits

Tomer Teller
Adi Hayon

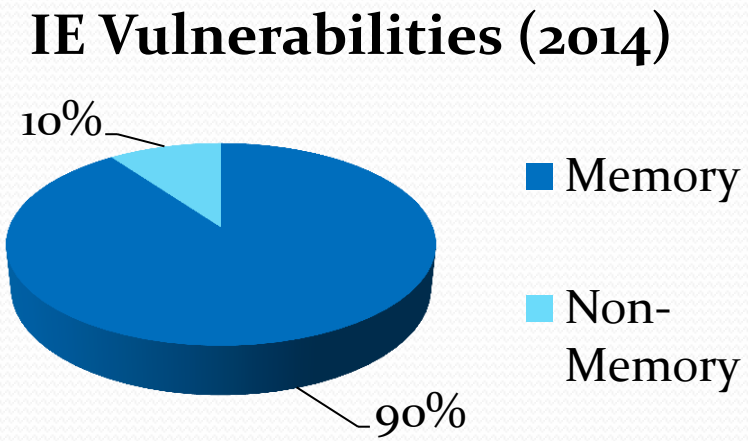
Vulnerability Statistics for 2014*

- 167 IE vulnerabilities
- 87 Chrome vulnerabilities
- 79 Firefox vulnerabilities
- ?? Opera

* Gathered on 18/8/2014 from NVD

Memory Corruption Vulnerabilities

- Stack/Heap buffer Overflows
- Integer Overflow
- Pointer Vulnerabilities (UAF/Double free)
- Format Strings

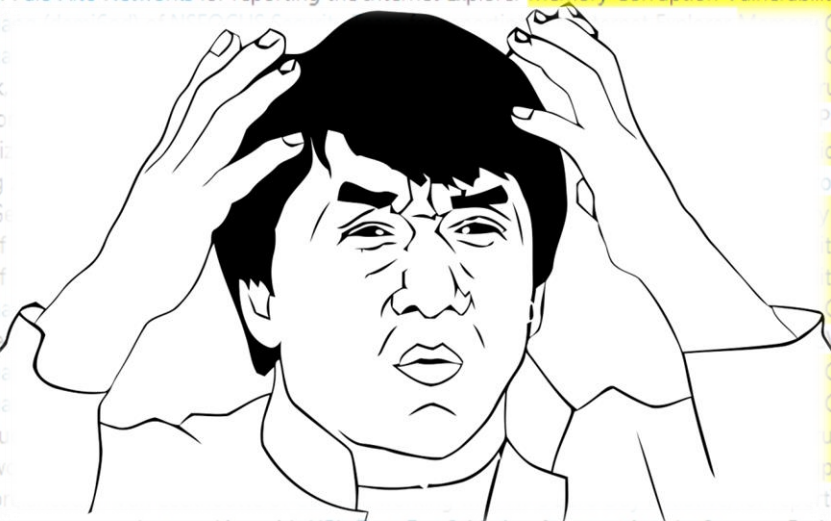


Microsoft Security Bulletin MS14-051

(August, 2014)

- What's wrong here?!

- AbdulAziz Hariri of HP's Zero Day Initiative for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-2774)
- Yujie Wen of Qihoo 360 for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-2784)
- Bo Qu of Palo Alto Networks for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-2796)
- Chen Zh… Corruption Vulnerability (CVE-2014-2808)
- Chen Zh… Corruption Vulnerability (CVE-2014-2810)
- IronRock… uption Vulnerability (CVE-2014-2811)
- James Fo… rivilege Vulnerability (CVE-2014-2817)
- AbdulAzi… n Vulnerability (CVE-2014-2818)
- Zeguang… Day Initiative, for reporting the Internet Explorer Elevation of Privilege Vulnerability (CVE-2014-2819)
- Arthur G… Corruption Vulnerability (CVE-2014-2820)
- Bo Qu of… y (CVE-2014-2821)
- Bo Qu of… y (CVE-2014-2822)
- Chen Zh… orruption Vulnerability (CVE-2014-2823)
- Yuki Che… E-2014-2824)
- Chen Zh… orruption Vulnerability (CVE-2014-2825)
- Chen Zh… Corruption Vulnerability (CVE-2014-2826)
- Simon Zu… uption Vulnerability (CVE-2014-2827)
- Omair, w… tion Vulnerability (CVE-2014-4050)
- Peter 'co… ing the Internet Explorer Memory Corruption Vulnerability (CVE-2014-4051)
- An anonymous researcher, working with HP's Zero Day Initiative, for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-4052)
- Simon Zuckerbraun of HP's Zero Day Initiative for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-4055)
- Peter 'corelanc0d3r' Van Eeckhoutte of Corelan, working with HP's Zero Day Initiative, for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-4056)
- Yuki Chen of Trend Micro, working with HP's Zero Day Initiative, for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-4057)
- Sky, working with HP's Zero Day Initiative, for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-4058)
- An anonymous researcher, working with HP's Zero Day Initiative, for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-4063)
- Wei Wang of VulnHunt for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-4067)



Case Study: CVE-2013-3897

- Use-After-Free vulnerability in IE (MS13-080)
- ROP chain to defeat DEP
- Using target-based non-ASLR modules
  - Office 2007/2010 - hxds.dll (location.href = 'ms-help:‘)
  - Msvcrt.dll
  - JAVA
- Heap Spray to allocate ROP chain around 0x14141414

Memory Layout Importance

- Shellcode should be placed in a predictable address
- Allocations should be adjacent
- Front-End Managers
  - Look-Aside List (LAL)
  - Low-fragmentation Heap (LFH)
- Predictable/controllable allocations
  - Heaplib 1.0 by Alexander Sotirov (Heap Feng Shui)
  - Heaplib 2.0 by Chris Valasek

CVE-2013-3897 Exploitation

- Inspect memory layout during exploitation

Endpoint-based exploit mitigation solution

(one of not so many)

- EMET by Microsoft (latest version 5.0)
- Free exploit mitigation application
- Thwart memory corruptions exploits (among other things)
- oxdabbadoo excellent EMET v4.1 Report
  http://oxdabbadoo.com/wp-content/uploads/2013/11/emet_4_1_uncovered.pdf

App Name	DEP	SEHOP	NullPage	HeapS...	EAF	EAF+	Mand...	Botto...	LoadLib	MemProt	Caller	SimEx...	Stack...	ASR
Acrobat.exe	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☐
AcroRd32.exe	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☐
EXCEL.EXE	☑	☑	☑	☑	☑	☐	☑	☑	☑	☑	☑	☑	☑	☑
iexplore.exe	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
INFOPATH.EXE	☑	☑	☑	☑	☑	☐	☑	☑	☑	☑	☑	☑	☑	☐

Endpoint-based solution disadvantages

- Kernel based vulnerabilities can evade it
- Invasive - application compatibility issues
- Enterprise maintenance (install, manage, update, etc.)
- Can be detected

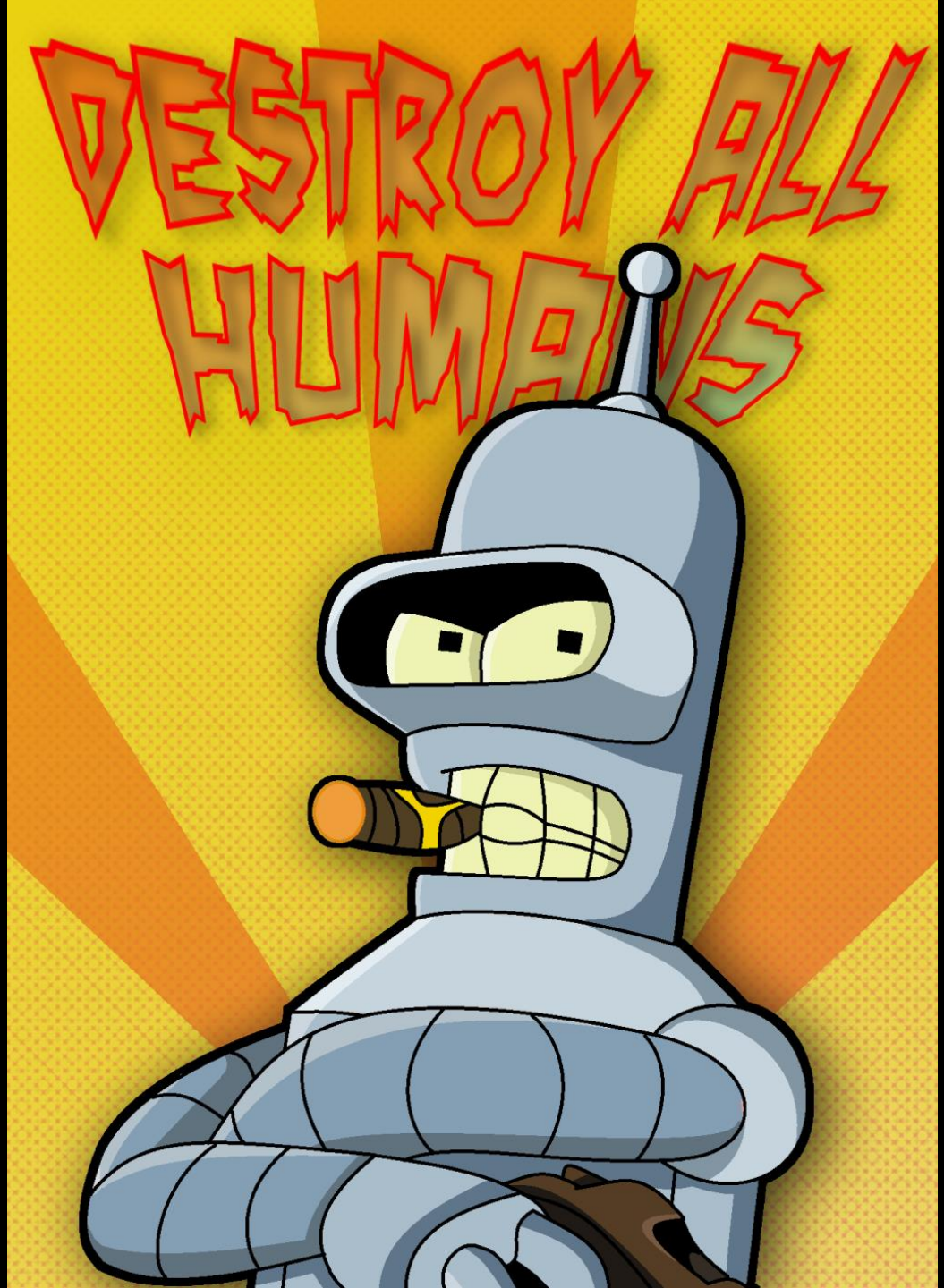
Lockheed Martin "Kill Chain"

- Describes phases of intrusions
- Mapping Indicators to defender courses of actions

Table 1: Courses of Action Matrix

Phase	Detect	Deny	Disrupt	Degrade	Deceive	Destroy
Reconnaissance	Web analytics	Firewall ACL				
Weaponization	NIDS	NIPS				
Delivery	Vigilant user	Proxy filter	In-line AV	Queuing		
Exploitation	HIDS	Patch	DEP			
Installation	HIDS	"chroot" jail	AV			
C2	NIDS	Firewall ACL	NIPS	Tarpit	DNS redirect	
Actions on Objectives	Audit log			Quality of Service	Honeypot	

source: http://www.lockheedmartin.com/content/dam/lockheed/data/corporate/documents/LM-White-Paper-Intel-Driven-Defense.pdf


DESTROY ALL
HUMANS

The Idea

- Exploiting memory corruption vulnerabilities requires a certain memory state

Manipulating the memory state

=

Destroying the exploit

(making it less reliable)

The Approach

- Assume: All websites are guilty until proven innocent
- Install a network proxy that monitors HTTP(S)
- Rewrite responses to include a JS library

- The library desired effect:
  - **Destroy** exploits memory layout
  - **Preserve** user experience and performance

Think “Anti-Heaplib” or “JavaScript ASLR“

Before

HTTP Request

HTTP Response

Code exploiting some client-side memory corruption vulnerability

After

HJsPI2Ky[1] - Notepad

File Edit Format View Help

```
<!DOCTYPE HTML PUBLIC " //W3C//DTD HTML 4.0//EN">
<html><script src="http://192.168.200.111:8000/myjs.js"></script>
<head>
<script>var KLUfefSOwlwRiFeyCXEflFVEtovWNuN = '1b193d6d2c213e052b220c003a3d342
a286c37131e1c0614251c22030f283f221339330O2e0535292b11180503102l0c393f3e0918203
5480d7b7a42706911386534 5d5d3a7f41762f412f63355a4c6a38417f7a537f2464581a2d68007
375026e6373181a7e7f166339506d6332480d2a7c457469116a3264595d3a28477e7d412f63645
d1e7a6800727e536374230b1c7c7450337b016d6573184a7b7c4063395739613548 0d767c4c75e
a412f33635e4d6a38477e2d5c7f2467581c7a680027755d6974230f4b2b7b50332e0538617318 4
```

Rewrite HTTP/S response to include the JS library

Code exploiting some client-side memory corruption vulnerability

How does it work?

- Hooks JavaScript elements that are used in exploits
  - Array (push/pop/..)
  - String (insert/remove/..)

- Manipulates hooked functions
  - “Setters” **destroy** layout
  - “Getters” **restore** layout

The Manipulations (partial list)

- Bit flipping
  - Switch between bits
- String reordering
  - e.g. Reversing Strings: “SHELLCODE” -> “EDOCLLEHS”
- Array swapping
  - Swap(Arr[i],Arr[j])
- Asynchronous defragmentation
  - setInterval() + Dummy Allocations + Garbage Collection

The proxy **randomizes** the manipulation technique on each iteration to avoid attackers adjustments

Example:
Before Array Manipulation

```
var arr = new Array();
arr[0] =
"nop+shellcode";
```

How it looks in JS

```
0:006> s 0c0c0c0c L1000 24 4e 9f 22
0c0c104e  24 4e 9f 22 fd b0 25 35-92 7f 15 87
0:006> dd 0c0c104e-16 L100
0c0c1038  90909090 90909090 90909090 90909090
0c0c1048  90909090 4e249090 b0fd229f 7f923525
0c0c1058  e3f68715 b34f767b 4870bb2c 37737e77
0c0c1068  7804e230 a843d539 e0889098 d4137205
0c0c1078  02461499 6734e1d0 66a9b13f 0c78f884
0c0c1088  197df586 9bb6b5f9 32721d77 40e0d1d2
0c0c1098  97427cb4 c128e381 794976d6 b94a7b3c
0c0c10a8  b7932d75 9bb81c74 3a7f4a7a 127e37e2
0c0c10b8  98b543eb 2b73bfd5 b6bb48f5 05464b8d
0c0c10c8  15b03570 34b42496 b314ba40 91fd6b66
0c0c10d8  d62ab199 90a9d485 beb8922d 1c49b73c
0c0c10e8  b90c7141 4e1d67f9 b23d4725 04e11193
0c0c10f8  422f4f9f a8f83f2c dadafc97 1aac1cbb
0c0c1108  2474d9e5 c92b5ef4 5e314ab1 fcee8318
0c0c1118  4e085e03 daf464ef 1e0b98e2 dff3e1ea
0c0c1128  3a7a8deb 4e189fda 026b104f b739db7c
0c0c1138  b895a9f7 f7c304b0 54cba941 a6b7ab81
0c0c1148  68860bd6 95cf4d2b d2981fc4 a7adb077
0c0c1158  37fd3b4b 36b7d8cc 60cc4efd 190171dd
0c0c1168  22466a54 d0bc012e 198dc3b1 24412b80
0c0c1178  6098a62c 9aef598b 58f7e4ef 7d72328d
0c0c1188  a524b035 2eb215c7 69b1d2cb 0216e5c8
```

How it looks in memory

Example:
After Array Manipulation

```
var arr = new Array();
nset(arr,0,
"nop+shellcode")
```

```
0:006> s 0c0c0c0c L1000 24 4e 9f 22
0c0c104e  24 4e 9f 22 fd b0 25 35-92 7f 15 87
0:006> dd 0c0c104e-16 L100
0c0c1038  15b03570 34b42496 b314ba40 91fd6b66
0c0c1048  d62ab199 90a9d485 beb8922d 1c49b73c
0c0c1058  b90c7141 4e1d67f9 b23d4725 04e11193
0c0c1068  422f4f9f a8f83f2c dadafc97 1aac1cbb
0c0c1078  2474d9e5 c92b5ef4 5e314ab1 fcee8318
0c0c1088  4e085e03 daf464ef 1e0b98e2 dff3e1ea
0c0c1098  3a7a8deb 4e189fda 026b104f b739db7c
0c0c10a8  b895a9f7 f7c304b0 54cba941 a6b7ab81
0c0c10b8  68860bd6 95cf4d2b d2981fc4 a7adb077
0c0c10c8  37fd3b4b 36b7d8cc 60cc4efd 190171dd
0c0c10d8  22466a54 d0bc012e 198dc3b1 24412b80
0c0c10e8  6098a62c 9aef598b 58f7e4ef 7d72328d
0c0c10f8  a524b035 2eb215c7 69b1d2cb 0216e5c8
0c0c1108  90909090 90909090 90909090 90909090
0c0c1118  90909090 4e249090 b0fd229f 7f923525
0c0c1128  e3f68715 b34f767b 4870bb2c 37737e77
0c0c1138  7804e230 a843d539 e0889098 d4137205
0c0c1148  02461499 6734e1d0 66a9b13f 0c78f884
0c0c1158  197df586 9bb6b5f9 32721d77 40e0d1d2
0c0c1168  97427cb4 c128e381 794976d6 b94a7b3c
0c0c1178  b7932d75 9bb81c74 3a7f4a7a 127e37e2
0c0c1188  98b543eb 2b73bfd5 b6bb48f5 05464b8d
```

How it looks in JS

How it looks in memory

Why does it work?

JS doesn't care about the memory layout

(so we hook and manipulate it, keeping it transparent)

Exploitation does

(layout manipulation breaks attacker assumptions)

Introducing Amnesia

- JS library to manipulate browser memory layout
- MiTM proxy which injects the lib to HTTP/S traffic

Check Point
SOFTWARE TECHNOLOGIES LTD.

Breaking CVE-2013-3897 with Amnesia

The Challenges

- Engineering Challenges:
  - User Experience (not breaking 'good' websites)
  - Multi-Browser Support
  - Performance
- Security Challenges:
  - Multi-layer obfuscation
  - Multi-Stage exploits

Flash is the new black

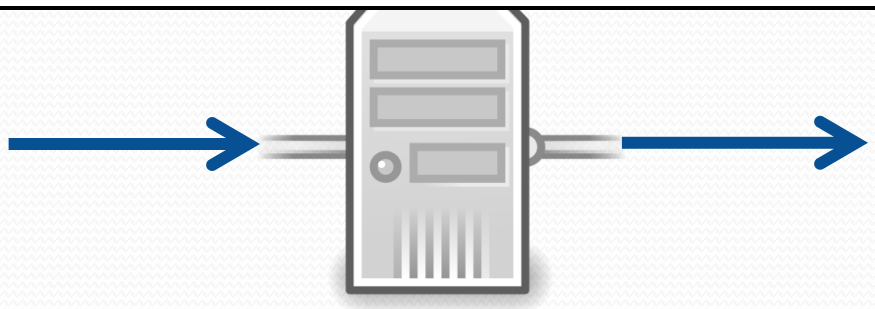
- Layout manipulation via Flash become popular
  - HTML+JS to setup the ground for exploitation
  - Flash object to setup the memory layout
  - Trigger the attack from JS or Flash
  - CVE-2014-1776, CVE-2014-0322, CVE-2013-3163, …
- Automation is harder
  - Environment needs all the elements to reproduce
  - Evasion tricks

SWF Wrapping

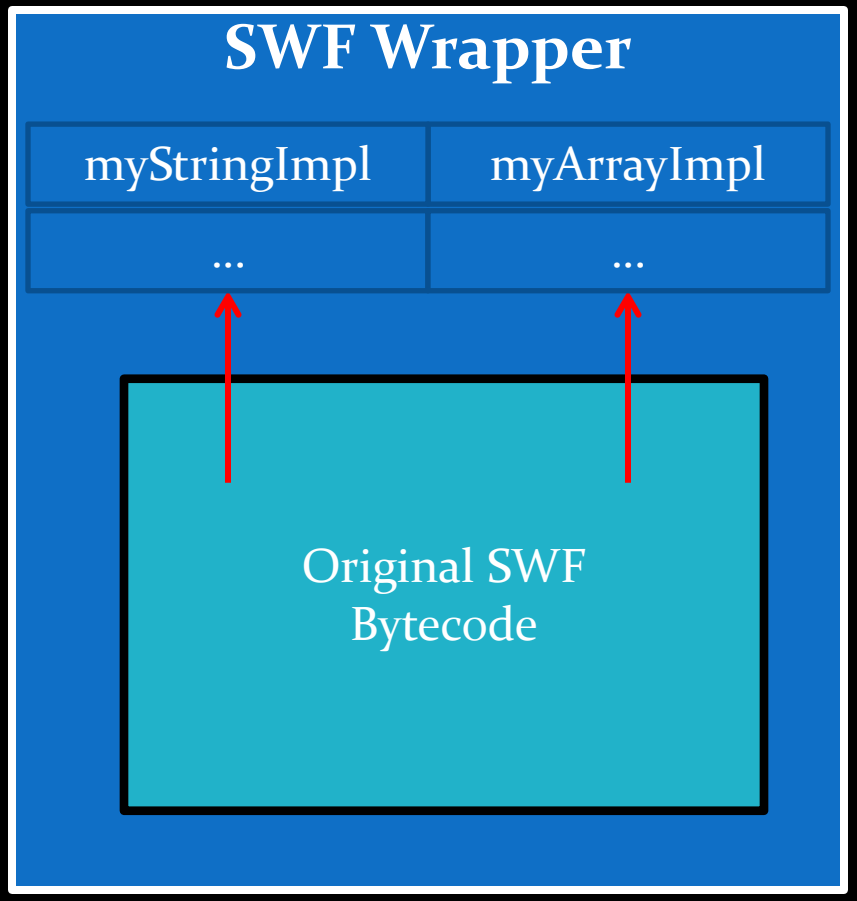
- Network Side (Proxy):
  - Replace the original SWF link with a link of a wrapper SWF
  - Pass the original SWF to the wrapper as an argument
- Client-Side (Browser):
  - Download & Execute the wrapper SWF file
- Client Side (Flash):
  - Download the original SWF file
  - Bytecode reflection
- Manipulate
  - Direct bytecode manipulation
  - Decompile -> Manipulate -> Recompile

SWF Wrapping

```
<HTML>
.
.
<object data="wrapper.swf" />
    <param type="application/x-shockwave-flash"/>
    <param name="FlashVars" value="exploit.swf/>
</object>
.
.
</HTML>
```



Proxy + Amensia


SWF Wrapper
myStringImpl
myArrayImpl
...
...
Original SWF
Bytecode

Check Point
SOFTWARE TECHNOLOGIES LTD.

SWF Wrapping In Action

Future Work

- PDF JavaScript
- JIT spraying
- Forced HeapSpray
- Asynchronous defragmentation improvements
- Shellcode scrubbing

Summary

- Exploiting memory corruptions is hard but popular
- End-Point solutions work but come with a price
- Network-based exploit mitigation alternatives exist

Amnesia

Open Source JS library to
destroy memory corruption exploits

https://github.com/djteller/Amnesia

Check Point
SOFTWARE TECHNOLOGIES LTD.

Thank You

Check out our projects

@

https://github.com/djteller/

@djteller  @adihayon1

Security Innovation Group