Introduction to Windows Kernel Exploitation

SANS SEC760 Extract

SANS Orlando, 2013

Stephen Sims

stephen@deadlisting.com

Kernel Exploitation…
…for Honey Badgers

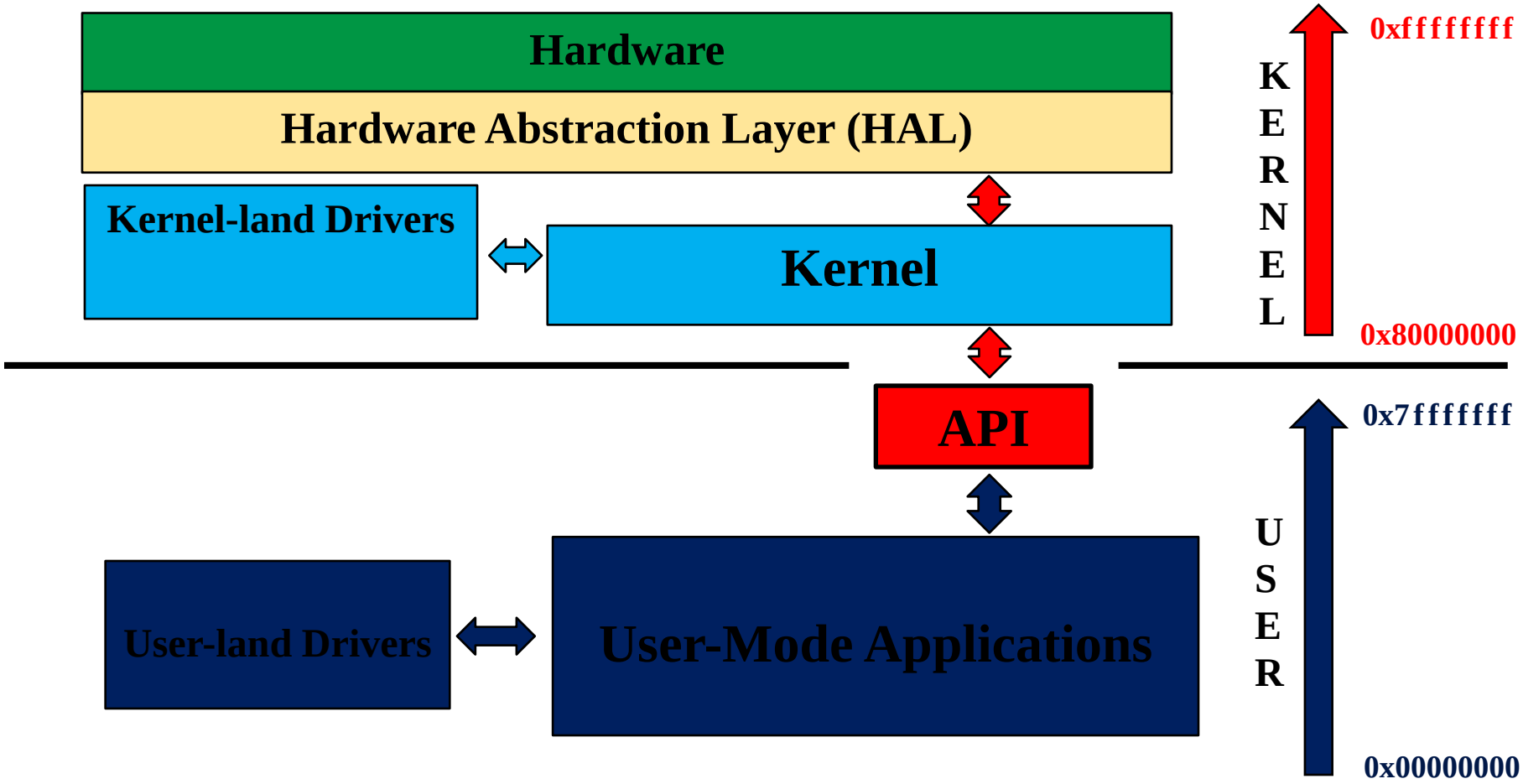
The Windows Kernel

- Software layer between applications and hardware
- Manages virtual memory, physical memory, drivers, input/output, hard disk access, prioritization, etc…
- Allows developers and applications transparent access to hardware through system calls
- Runs in a privileged mode to protect itself from application errors
- Windows uses Ring 0 for the Kernel and Kernel drivers, and Ring 3 for user applications
- A crash in Ring 0 causes the whole system to fail
- Has access to all memory in all processes

CPU Modes / Processor Access Modes

- Windows has two access modes:
  - Kernel Mode – Core Operating System Components, Drivers
  - User Mode – Application Code, Drivers
- Kernel memory is shared between processes
- 32-bit Windows provides 2GB of virtual memory to the kernel and 2GB to the user; however, there is an optional /3GB flag to give 3GB to the user
- 64-bit Windows provides 7TB or 8TB to the kernel and 7TB or 8TB to the user
  - Depends on the architecture: x64 or IA-64
  - This does not exhaust 2 ** 64, but is plenty for now

High-Level Layout – 32-bit



Debugging the Windows Kernel

- Common Windows debuggers such as Ollydbg and Immunity Debugger are Ring 3 debuggers
- Visibility is lost once crossing over to Ring 0
- Kernel debugging can be performed with WinDbg from the Microsoft SDK, IDA Pro with remote debugging, or other methods
- The target system being debugged is most commonly in a VM, while the host connects with the debugger
- You must enable debugging on the target system in its BIOS

Setting Up Kernel Debugging on Windows

- Most common configuration is to have two systems:
  - Host system performing the debugging
  - Target system being debugged
- Multiple ways to connect to the target:
  - Null modem cable, IEEE 1394 cable, or USB 2.0 cable
  - Virtualization
- Local Kernel Debugging
  - Enable the host system for debugging at boot-up
  - Not the preferred option due to limitations
- This information is documented in dozens of books and articles

Setting Up a Kernel Debugger with Virtualization

- Windows 7 Example
  - Power down the virtual machine
  - Add a new serial port under the hardware tab in VMware Settings
  - Select, "Output to a named pipe"
  - Default on VMware will set it to \\.\pipe\com_1
    - This must correlate to the serial port number
    - e.g. "Serial Port 2" should use the name \\.\pipe\com_2
  - Select, "This end is the server," and "The other end is an application"

Setting Up a Kernel Debugger with Virtualization

- Windows 7 Example – cont.
  - Under I/O mode, check the box that says, "Yield CPU on poll"
    - Per VMware, *"This configuration option forces the affected virtual machine to yield processor time if the only task it is trying to do is poll the virtual serial port."*
  - Open an Administrative command shell
    - *bcdedit /set {current} debug yes*
    - *bcdedit /set {current} debugtype serial*
    - *bcdedit /set {current} debugport <serial port assigned>*
    - *bcdedit /set {current} baudrate 115200* ← Default
    - Reboot the system

WinDbg

- Ring 0 debugger for Windows
- Part of the Microsoft Software Developer Kit (SDK) and the Windows Driver Kit (WDK)
- Available at http://msdn.microsoft.com/en-us/windows/hardware/gg463009/
- Can install as a standalone; however, you will have less functionality
- It's a good idea to grab a cheat sheet on the commands from Google… non-intuitive

Boot Up the Debugging-Enabled Virtual Machine

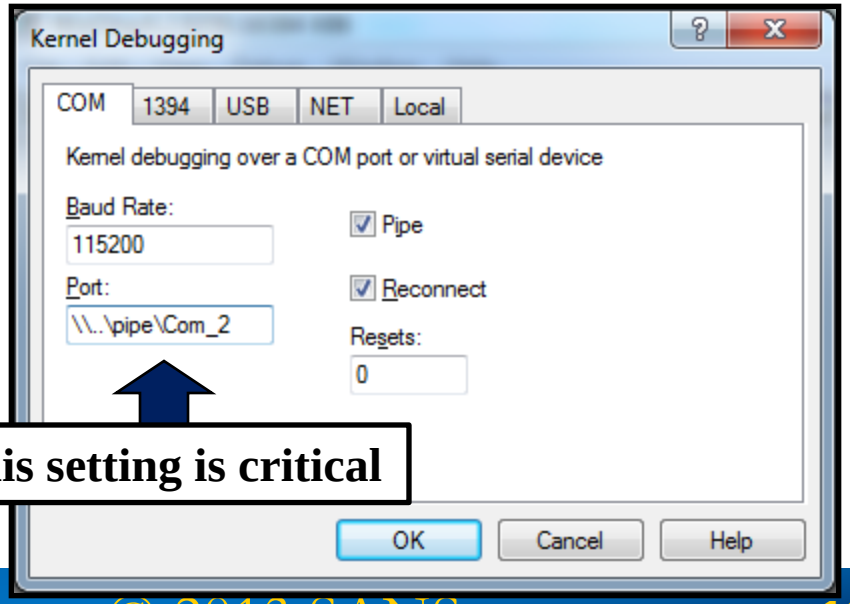
- If you have properly set up the target system, the following should appear at boot

```
                    Windows Boot Manager

Choose an operating system to start, or press TAB to select a tool:
(Use the arrow keys to highlight your choice, then press ENTER.)

    Windows 7 [debugger enabled]                                    >
```

- You should now be able to connect with WinDbg

Open Up WinDbg

- Under File, Symbol File Path, enter in:
  - srv*c:\<folder for symbols>*http://msdl.microsoft.com/download/symbols
- We're setting two paths:
  - The local path "c:\" copies necessary symbols to that location from MS and "http" to the symbol store
- Click File, Kernel Debug
- Set the right COM port
- \\.\pipe\Com_2



Connect to the Target

- Once the COM port opens, click Debug, Break

Command - Kernel 'com:port=\\.\pipe\com_2,baud=115200,pipe,reconnect' - WinDbg:6.2.9200.16384...

```
Built by: 7600.16385.x86fre.win7_rtm.090713-1255
Machine Name:
Kernel base = 0x82808000 PsLoadedModuleList = 0x82950810
Debug session time: Fri Aug 24 14:13:56.728 2012 (UTC - 7:00)
System Uptime: 0 days 0:06:02.078
Break instruction exception - code 80000003 (first chance)
*******************************************************************************
*                                                                             *
*   You are seeing this message because you pressed either                    *
*       CTRL+C (if you run console kernel debugger) or,                       *
*       CTRL+BREAK (if you run GUI kernel debugger),                          *
*   on your debugger machine's keyboard.                                      *
*                                                                             *
*                    THIS IS NOT A BUG OR A SYSTEM CRASH                      *
*                                                                             *
* If you did not intend to break into the debugger, press the "g" key, then   *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                          *
*                                                                             *
*******************************************************************************
nt!RtlpBreakWithStatusInstruction:
82873394 cc              int     3
```

Kernel Memory!

```
kd>
```

Debugging the Windows Kernel - Demo

- Demonstration of setting up a remote kernel connection with WinDbg on Windows 7 and causing a crash
- SMBv2 bug discovered by Laurent Gaffié
- Crash in KeAccumulateTicks() due to NT_ASSERT()/DbgRaiseAssertionFailure()

Crash!

Command - Kernel 'com:port=\\.\pipe\com_2,baud=115200,pipe,reconnect' - WinDbg:6.2.9200.16384...

```
*       CTRL+BREAK (if you run GUI kernel debugger),                    *
*   on your debugger machine's keyboard.                                *
*                                                                       *
*                    THIS IS NOT A BUG OR A SYSTEM CRASH                *
*                                                                       *
* If you did not intend to break into the debugger, press the "g" key, then   *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                    *
*                                                                       *
*************************************************************************
nt!RtlpBreakWithStatusInstruction:
82873394 cc              int     3
kd> g
Assertion: *** DPC execution time exceeds system limit
    This is NOT a break in update time
    This is a BUG in a DPC routine
    Perform a stack trace to find the culprit
    The time out limit will be doubled on continuation
    Use gh to continue!!

nt!KeAccumulateTicks+0x316:
82873dc8 cd2c            int     2Ch
```

kd>

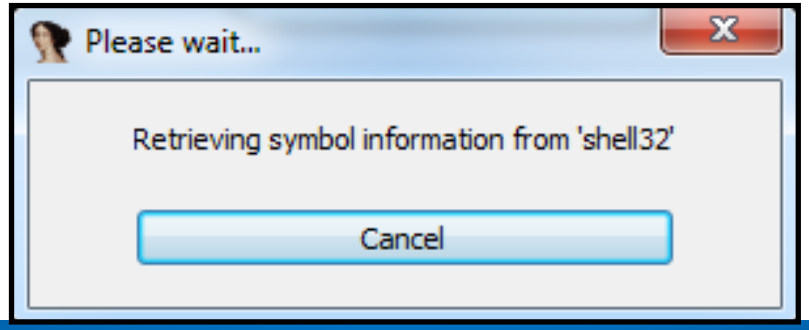
WinDbg with IDA

- One of the debugging options supported by IDA is WinDbg
- Before starting with this debugging option you must have installed Debugging Tools for Windows
- It is also beneficial to create the following environment variable for debugging symbols
  - Variable Name: _NT_SYMBOL_PATH
  - Value: srv*C:\WIN7\Symbols*http://msdl.microsoft.com/download/symbols
- Enter a PATH environment variable so IDA can find WinDbg

WinDbg with IDA (2)

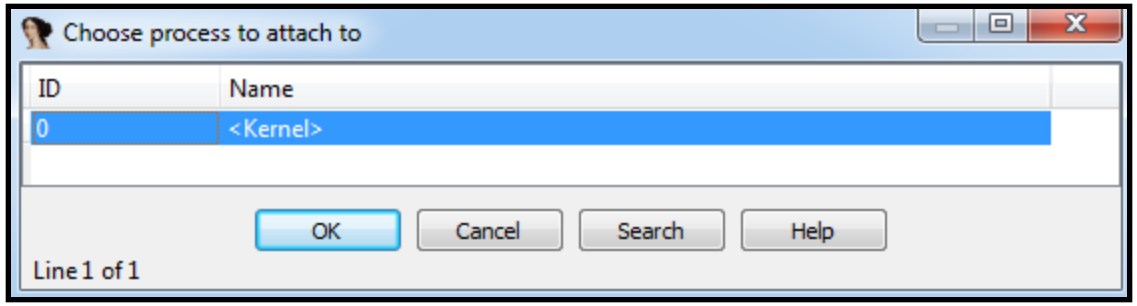
- You must set the IDA debugger to "Windbg debugger"
- Unless you are attaching to a running process, load a file into IDA and press F9 to start
- WINDBG is now showing in IDA: 

- When pausing the process, if debugging symbols are set up right, you will see this box as symbols are being retrieved:



Kernel Debugging with WinDbg through IDA Pro

- Under the Debugger menu option, select Attach, Windbg debugger
- For the connection string, use: `com:port=\\.\pipe\Com_2,pipe`
- Click on Debug Options, followed by Set specific options, and choose Kernel mode debugging
- You should get:

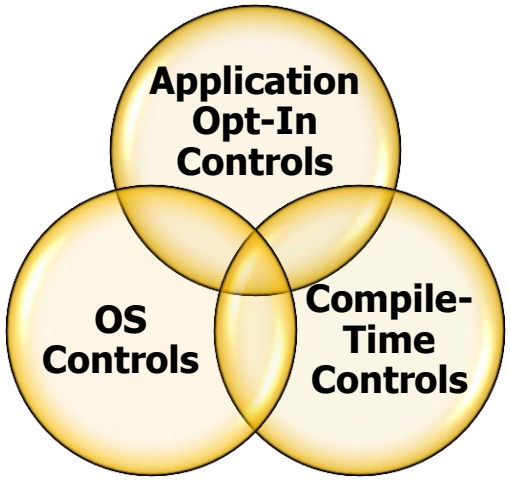


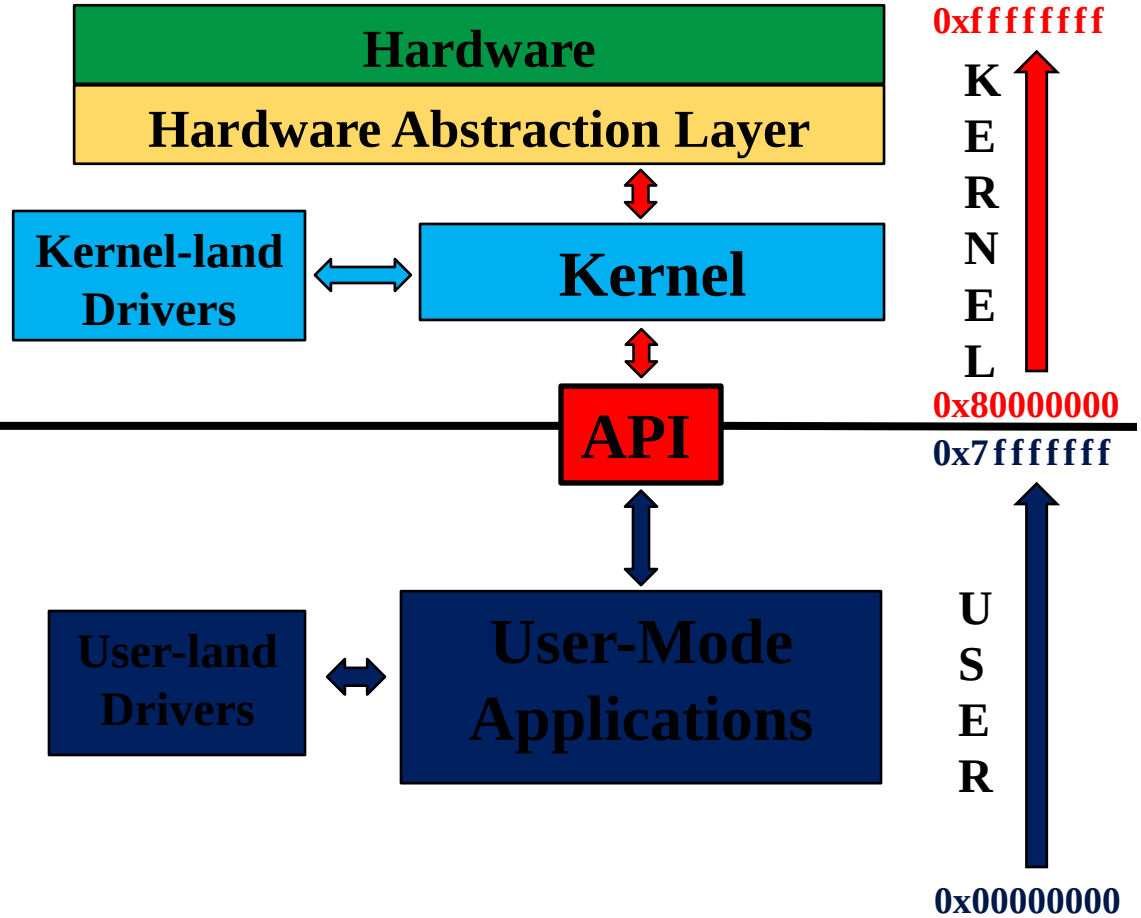
Windows User-Mode Vulnerabilities

- User-Mode vulnerabilities still often exist but are increasingly difficult to exploit due to exploit mitigation controls
  - Data Execution Prevention (DEP)
  - SafeSEH
  - Address Space Layout Randomization (ASLR)
  - Safe Unlink
  - Low Fragmentation Heap (LFH)
  - Security Cookies
  - Many more….

So Why Attack the Kernel?

Less Exploit Mitigation Controls Historically





Kernel Hacking Considerations

- Kernel hacking requires advanced knowledge of C++ development on Windows, memory, architecture, etc
- Very large learning curve between user-mode and kernel-mode hacking
- Per Microsoft:
  - *"x64 versions of Windows Vista and Windows Server 2008 require Kernel Mode Code Signing (KMCS) in order to load kernel-mode software"*
  - *http://msdn.microsoft.com/en-us/library/windows/hardware/gg487317.aspx*
- Lots of user functionality was moved to the kernel
- Mistakes will likely crash the whole system

Common Windows Kernel Exploitation Techniques

- Stack and heap overflows
  - Kernel pool is a shared resource amongst all Ring 0 functionality
- Null pointer dereferencing
  - e.g. Uninitialized pointers
  - Attacker must load payload to 0x00000000
- Prior to Windows 7 & Server 2008, Safe Unlink was not performed in the kernel pool
- Lookaside Lists still used in Kernel memory on Win-7
  - Singly-Linked with no validation
- Input validation errors

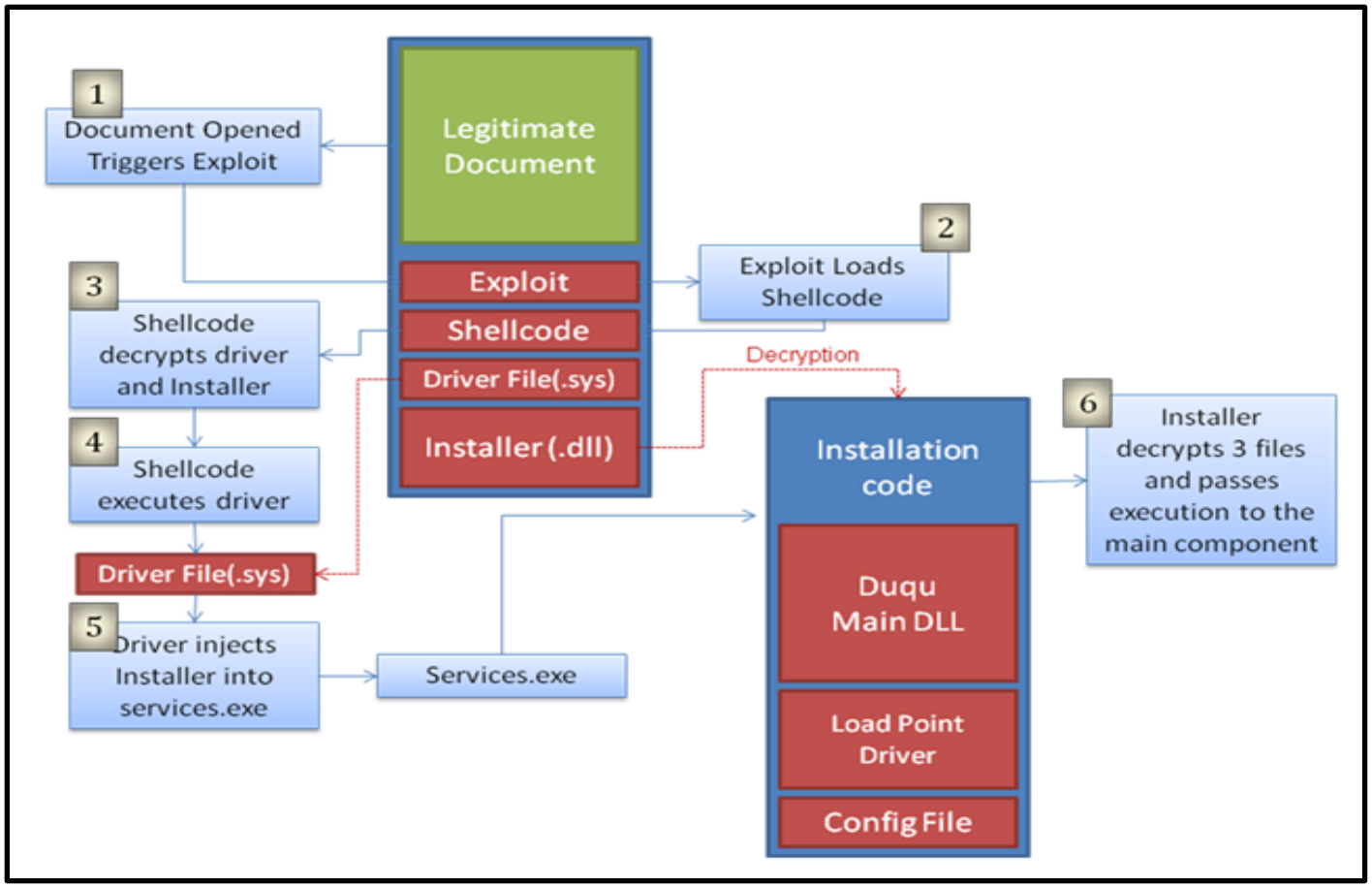
Windows Kernel Hardening

- On Windows 8 and Server 2012
  - First 64KB of memory cannot be mapped, so no more null pointer dereferencing
  - Guard pages added to the kernel pool
  - Improved ASLR
  - Kernel pool cookies
- General protection enhancements
  - C++ vtable protection for Internet Explorer
  - ROP/JOP protection
  - ForceASLR, sehop, more aggressive cookies

The Duqu Trojan

- Malware that exploited a 0-day Windows Kernel flaw and contained a command & control channel
- Shared much of the same code as Stuxnet
- Initially sent via malformed MS Word documents to targeted organizations
- Communicated directly to a C&C server in various countries, or if lacking Internet access, bridging through other infected systems
- Lots of great research by CrySyS Lab, Kaspersky, Symantec, and other organizations

Symantec's Duqu Diagram



http://www.symantec.com/connect/w32-duqu_status-updates_installer-zero-day-exploit

The Exploit

- Duqu exploited a kernel bug:
  - Vulnerability in TrueType font parsing could allow elevation of privileges
  - http://support.microsoft.com/kb/2639658
  - http://technet.microsoft.com/en-us/security/bulletin/ms11-087
- Vulnerability is exploited by opening a malicious document or web page that embeds TrueType font files
- Sign extension error occurs when glyph contours is > 0x7FFF
  - Sign extension increases the number of bits to the MSB side of a + or – value
  - 0x7FFF in Base2 is 111111111111111 | 0x8000 is 1000000000000000
- Allows for the corruption of kernel heap memory
- Must have advanced knowledge of MS GDI to walk through

Demonstration

- Triggering MS11-087
- GDI Font Fuzzer built by Lee Ling Chuan & Chan Lee Yee

References

- .NET Obfuscation - http://msdn.microsoft.com/en-us/magazine/cc164058.aspx
- Overload Induction - http://msdn.microsoft.com/en-us/library/ms227210%28v=vs.80%29.aspx
- Kasperskey – Duqu Steal Everything - http://www.kaspersky.com/about/press/duqu
- MS11-087 Advisory - http://technet.microsoft.com/en-us/security/bulletin/ms11-087
- Windows Internals Part 1 – Sixth Edition "Russinovich, Mark; Solomon, David A., Ionescu, Alex" Microsoft Press, 2012
- A Guide to Kernel Exploitation – Attacking the Core "Perla, Enrico; Oldani , Massimiliano" Syngress, 2011
- Enhanced Memory Protections in IE10 – Higman, Forbes http://blogs.msdn.com/b/ie/archive/2012/03/12/enhanced-memory-protections-in-ie10.aspx
- OpenRCE Anti Reverse Engineering Database http://www.openrce.org/reference_library/anti_reversing