RSA Conference2016
San Francisco | February 29 – March 4 | Moscone Center

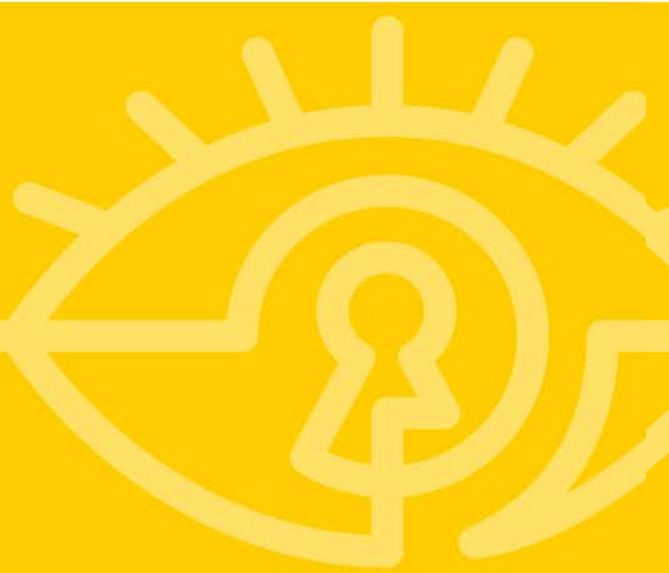
Connect to Protect

SESSION ID: HT-T10

# Bruh! Do you even diff?

## Diffing Microsoft Patches to Find Vulnerabilities

**Stephen Sims**
Security Researcher
SANS Institute
@Steph3nSims



# Part I – Binary Diffing

# Binary Diffing



- Security patches are often made to applications, libraries, driver files, etc.
- When a new version is released it can be difficult to locate what changes were made
  - Some are new features or general application changes
  - Some are security fixes
  - Some changes are intentional to thwart reversing
- Some vendors make it clear as to reasoning for the update to the binary
- Binary diffing tools can help us locate the changes

# Can you spot the difference?



# Oh, there it is!



# How do they Work?



- Diffing tools determine matched and unmatched blocks and functions using various techniques
  - Same name matching
  - Same assembly, same decompiled code, same edges
  - Equal calls to and from functions
  - Block count from within a function and lines of disassembly
  - Many, many other techniques…

# A Basic Example of a Diff



- Below is an example of a code block within a function from two different versions of the same file

Unpatched

```
0006F947    _TcpDeliverReceive@28
0006FB23    and             ds:[esi+0x10], 0
0006FB27    mov             eax, 0xFFF7
0006FB2C    and             b2 ds:[ebx+0x40], b2 ax
0006FB30    mov             eax, ss:[ebp+var_1C]
0006FB33    add             ds:[esi+0xC], eax
0006FB36    test            ds:[edi+0x48], 0x10000
0006FB3D    jz              loc_6FB4F
```

Patched

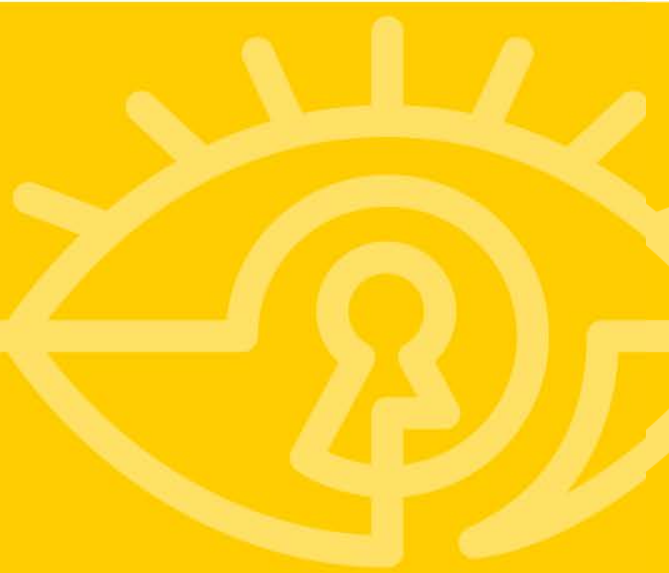
```
0006F977    _TcpDeliverReceive@28
0006FB53    and             ds:[esi+0x10], 0
0006FB57    mov             eax, 0xFFF7
0006FB5C    and             b2 ds:[ebx+0x40], b2 ax
0006FB60    mov             eax, ss:[ebp+var_1C]
0006FB63    add             ds:[esi+0xC], eax
0006FB66    test            eax, eax

0006FB68    jbe             loc_6FB71
```

# Why Diff Binaries and Patches?

- 0-day exploit sales and bug bounties are very popular and profitable
  - In early 2014, Yang Yu earned $100K disclosing 3 exploit mitigation bypass techniques to MS
  - At CanSecWest Pwn2Own 2014 Vupen took home $400K and in 2015 Jung Hoon Lee took home $225K!
  - Google paid over $1.5M in 2014 in bug bounties
- 1-day exploit pricing depends on the severity of the bug, the of affected users, and how quickly it is developed af patch is released
- Exploit writing is *very* competitive

http://www.forbes.com/sites/andygreenberg/2012/03/23/shopping-for-zero-days-an-price-list-for-hackers-secret-software-exploits/#535dbe366033



# Part II – Binary Diffing Tools

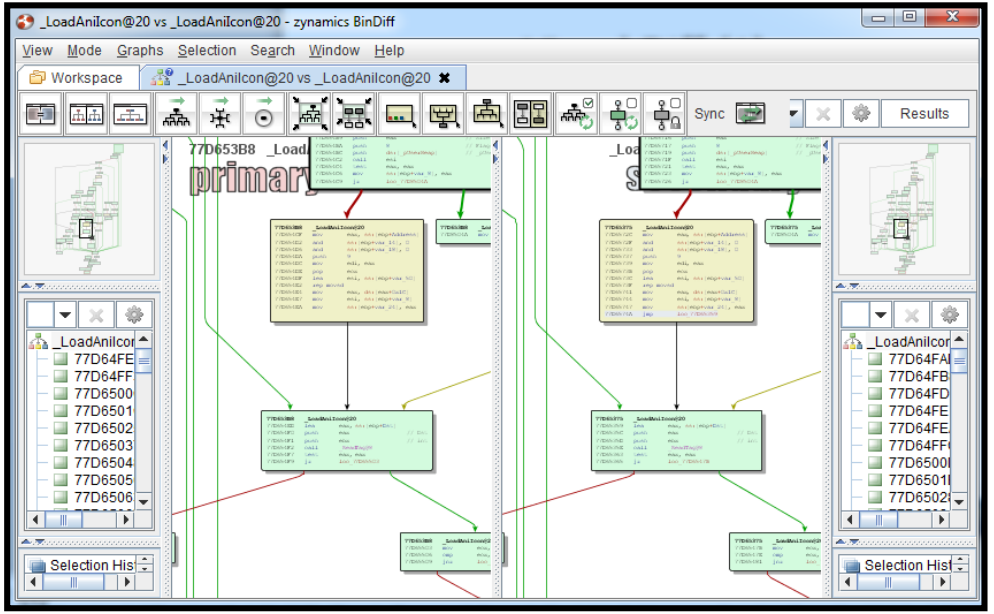
# Popular Binary Diffing Tools



- The following is a list of well-known binary diffing tools:
  - Zynamics/Google's BinDiff
  - Joxean Koret's Diaphora
  - Core Security's turbodiff
  - DarunGrim 4 by Jeongwook Oh
  - patchdiff2 by Nicolas Pouvesle
- There are others, but we will focus on BinDiff and Diaphora

# BinDiff

- Originally from Zynamics prior to being acquired by in 2011
- Plugin to IDA
- Not currently maintained
- Available for $200*

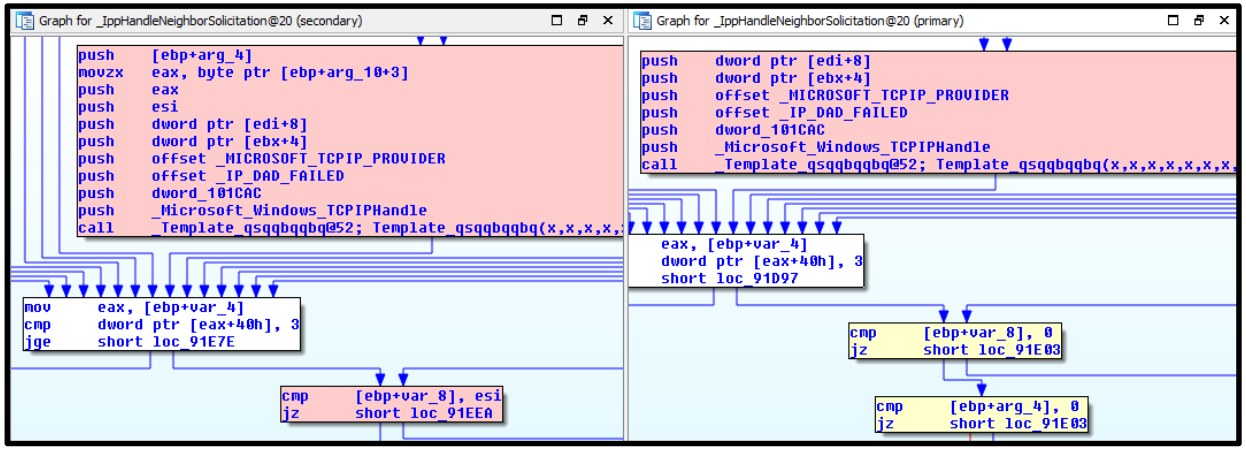



* The source code was released in 2015 for some of the Zynamics products, and some of it pulled back. It is sometimes easy to find for purchasing and then other times seems to be unavailable.

# Diaphora



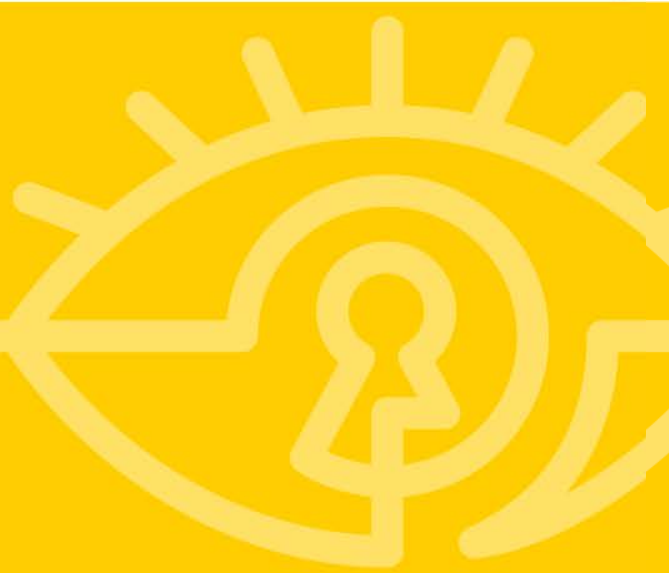
- A free binary diffing tool written and actively maintained by Joxean Koret at: https://github.com/joxeankoret/diaphora/
- Uses a large number of heuristics to determine differences
- Can take time
  run on large



# Basic Binary Diffing Demonstration



- Demonstration of a diff between two C programs using different functions to display “Hello World!”
  - HelloWorld1
    - Uses the printf() function; however, GCC optimizations convert it to a puts() call
  - HelloWorld2
    - Uses the fprintf() function which is not changed by GCC



# Part III – Microsoft Patches

# Patch Tuesday

- Microsoft releases patches on the second Tuesday of each month, for now…
- An effort to help simplify the patching process
  - Random patch releases caused many users to miss patches
  - However, waiting up to 30 days for the next patch has security concerns
- Emergency patches are released out-of-cycle
- Many exploits released in the days following
- Analyzing Microsoft packages can help you understand how they fix bugs, helping with finding  unknown bugs

# Patch Distribution



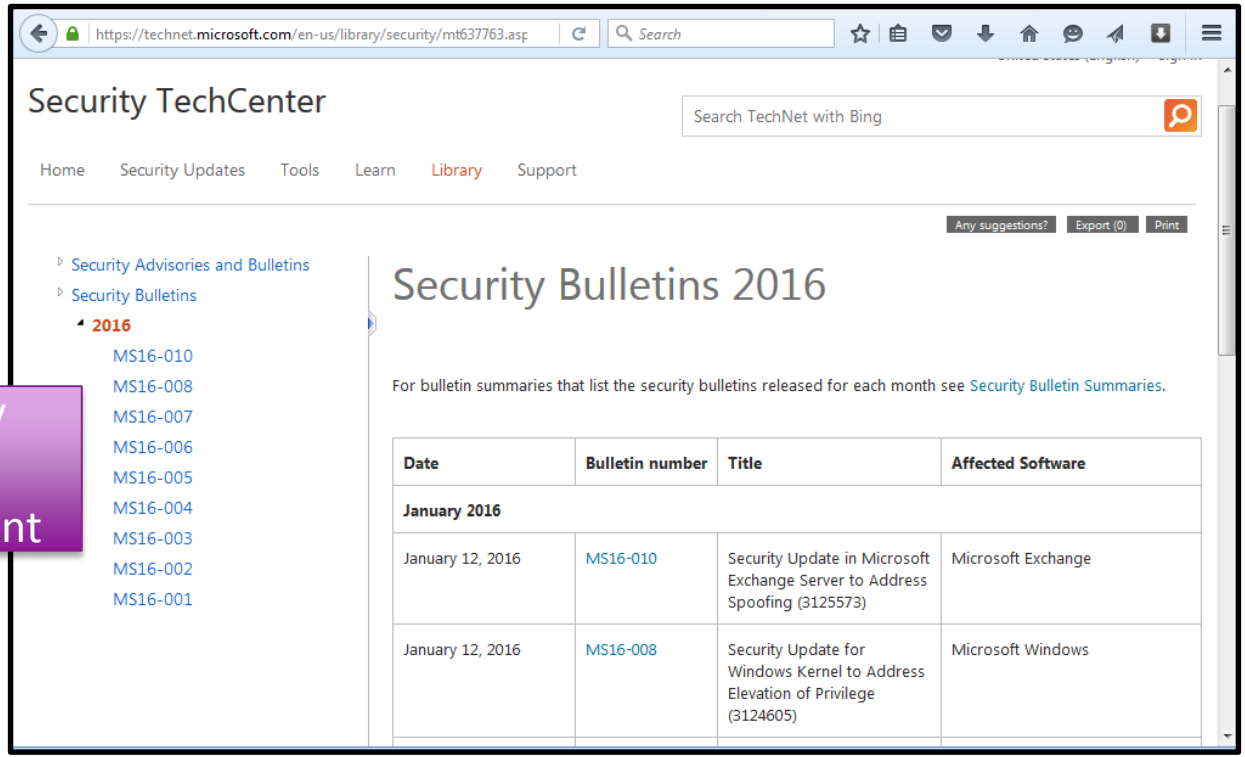
- Windows Update
  - Website available at http://update.microsoft.com
  - Automatic Updates
- Vista, 7, 8, 10 & Server 2008/2012/2016
  - Automatic Updates has expanded functionality
- Windows Server Update Service (WSUS)
  - Enterprise patch management solution
  - Control over patch distribution
- Third-party Patch Management Solutions

Deprecated

# Obtaining Patches



Security Bulletin Assignment



# Patch Extraction

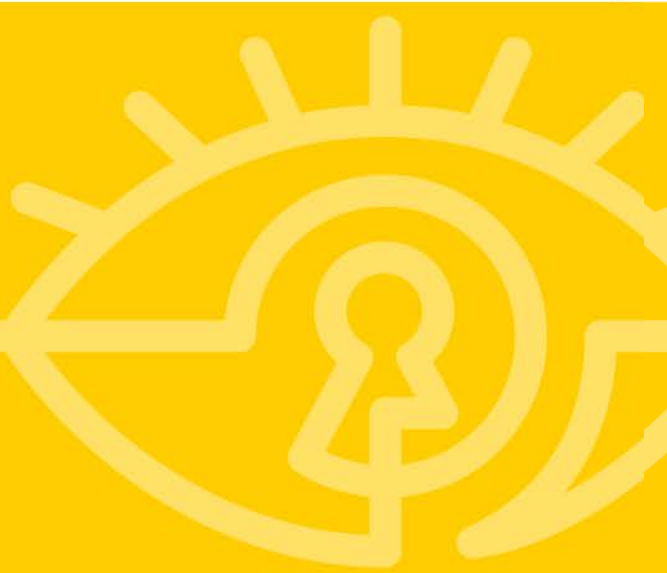


- Once a patch has been downloaded…
  - Download the previous most recent update to that same file (e.g. Updates replaced)
  - Use the “expand” tool at command line to extract both patch files
    - e.g. expand –F:* <.msu file> <dest>
    - Do the same for the extracted .cab file
    - Do not double-click or it will attempt to install
  - Analyze the General Distribution Release (GDR) file as it is limited to security fixes | QFE files may included unwanted changes

# Windows 10 Patches



- Microsoft says, *"Windows 10 updates are cumulative. Therefore, this package contains all previously released fixes."* e.g. https://support.microsoft.com/en-us/kb/3135174
  - You can get stand-alone packages from Microsoft's Update Catalog website: http://catalog.update.microsoft.com/v7/site/home.aspx
  - …but, they still contain all fixes, which makes determining what change corresponds to a particular CVE more difficult
  - For now, it's easiest just to use updates for Windows, Vista, 7, and 8



# Part IV – Diffing a Microsoft Patch

# MS16-014 & MS16-009

- Microsoft Security Bulletin MS16-014 – Important
  - Security Update for Microsoft Windows to Address Remote Code Execution (3134228) – Private Disclosure - Discovered by Greg Linares
- Microsoft Security Bulletin MS16-009 – Critical
  - Cumulative Security Update for Internet Explorer (3134220)
- Both have fixes that apply in relation to CVE-2016-0041 – DLL Loading Remote Code Execution Vulnerability
  - Corrects how Windows and IE validates input before loading DLL files
  - Affected Windows Vista through Windows 10 and IE 9/10/11
  - The file urlmon.dll is patched as part of MS16-009

# Diffing urlmon.dll



- When diffing urlmon.dll, only one function has a change
  - BuildUserAgentStringMobileHelper()

Matched Functions

| similarity | confide | change | EA primary | name primary |
|---|---|---|---|---|
| 0.99 | 0.99 | -I----- | 000000018003B2A0 | BuildUserAgentStringMobileHelper(UACOMPATMODE,c... |
| 1.00 | 0.99 | ------- | 0000000180001000 | _dynamic_initializer_for__g_OleAutDll__ |
| 1.00 | 0.99 | ------- | 0000000180001010 | _dynamic_initializer_for__g_mxsMedia__ |
| 1.00 | 0.99 | ------- | 0000000180001040 | _dynamic_initializer_for__g_mxsSession__ |

  - It is nearly identical at 99% similarity

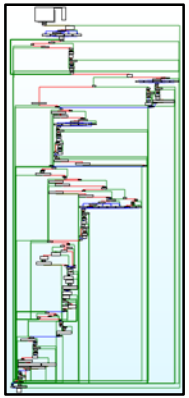
# Visual Diff of Function Changes



```
0000000180003B2A0   ?BuildUserAgentStringMobileHelper@@YAPEADW4UACOMPATMODE@@PEADW4USERAGENT_TYPE@@H@Z
0000000180003BDA1   xor         r8d, r8d                    // dwFlags
0000000180003BDA4   lea         b8 rcx, b8 cs:[LibFileName]  // LibFileName

0000000180003BDAB   xor         edx, edx                    // hFile
0000000180003BDAD   call        b8 cs:[__imp_LoadLibraryExW] // __imp_LoadLibraryExW
0000000180003BDB3   test        b8 rax, b8 rax
0000000180003BDB6   jz          b8 loc_18003BE7B
```

Unpatched



Patched

```
0000000180003B2A0   ?BuildUserAgentStringMobileHelper@@YAPEADW4UACOMPATMODE@@PEADW4USERAGENT_TYPE@@H@Z
0000000180003BCB1   xor         edx, edx                    // hFile
0000000180003BCB3   lea         b8 rcx, b8 cs:[LibFileName]  // LibFileName
0000000180003BCBA   mov         r8d, 0x800                  // dwFlags

0000000180003BCC0   call        b8 cs:[__imp_LoadLibraryExW] // __imp_LoadLibraryExW
0000000180003BCC6   test        b8 rax, b8 rax
0000000180003BCC9   jz          b8 loc_18003BD8C
```

# LoadLibraryExW()



- LoadLibraryExW() is the Unicode name for LoadLibraryEx(), and is the function called where the arguments were changed by the patch

```
HMODULE WINAPI LoadLibraryEx(
  _In_        LPCTSTR   lpFileName,
  _Reserved_  HANDLE    hFile,
  _In_        DWORD     dwFlags
);
```

- Per MSDN, *"This function loads the specified module into the address space of the calling process."*
- One argument is dwFlags which can be used to specify how and from where DLL's can be loaded

# The Vulnerability



- In the unpatched function, a value of 0 is being passed as the dwFlags argument
  - A value of 0 will cause the behavior of LoadLibraryEx() to model that of the LoadLibrary() function
  - This may allow for the loading of malicious DLL's

```
000000018003B2A0    ?BuildUserAgentStringMobileHelper@@YAPEADW4UACOMPATMODE@@PEADW4USERAGENT_TYPE@@H@Z
000000018003BDA1    xor         r8d, r8d                    // dwFlags
000000018003BDA4    lea         b8 rcx, b8 cs:[LibFileName] // LibFileName

000000018003BDAB    xor         edx, edx                    // hFile
000000018003BDAD    call        b8 cs:[__imp_LoadLibraryExW] // __imp_LoadLibraryExW
000000018003BDB3    test        b8 rax, b8 rax
000000018003BDB6    jz          b8 loc_18003BE7B
```

dwFlags value is 0

# The Fix

- In the patched function, a value of 0x800 is being passed as the dwFlags argument
  - This value's name is "LOAD_LIBRARY_SEARCH_SYSTEM32"
  - This ensures that only "%windows%\system32" is searched, preventing the loading of malicious DLL's

```
000000018003B2A0   ?BuildUserAgentStringMobileHelper@@YAPEADW4UACOMPATMODE@@PEADW4USERAGENT_TYPE@@H@Z
000000018003BCB1   xor         edx, edx                    // hFile
000000018003BCB3   lea         b8 rax, b8 cs:[LibFileName] // LibFileName
000000018003BCBA   mov         r8d, 0x800                  // dwFlags

000000018003BCC0   call        b8 cs:[__imp_LoadLibraryExW]
000000018003BCC6   test        b8 rax, b8 rax
000000018003BCC9   jz          b8 loc_18003BD8C
```

dwFlags value is 0x800

# Part V – Demonstration and Exploitation

# How can we exploit this bug?



- The DLL urlmon.dll is loaded automatically when you start IE and many other Microsoft applications
- We need urlmon.dll to attempt to load PhoneInfo.dll
  - ****PhoneInfo.dll does not come with windows 10****
  - BuildUserAgentStringMobileHelper() from within urlmon.dll attempts to load PhoneInfo.dll where we saw the patch applied
  - If we can get this line executed we can put a malicious DLL somewhere late in the SafeDLLSearchMode order
  - A simple text search in IDA shows you the locations where PhoneInfo.dll is passed as an argument to LoadLibraryExW()

# SafeDLLSearchMode Order



- Per MSDN

If **SafeDllSearchMode** is enabled, the search order is as follows:

1) The directory from which the application loaded.
2) The system directory. Use the **GetSystemDirectory** function to get the path of this directory.
3) The 16-bit system directory. There is no function that obtains the path of this directory, but it is searched.
4) The Windows directory. Use the **GetWindowsDirectory** function to get the path of this directory.
5) The current directory.
6) The directories that are listed in the PATH environment variable. Note that this does not include the per-application path specified by the **App Paths** registry key. The **App Paths** key is not used when computing the DLL search path.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682586%28v=vs.85%29.aspx

# Demonstration



- In this demo we will place our own PhoneInfo.dll file into a location listed in the PATH environment variable and get it loaded into IE11

# Apply What You Have Learned Today

- Defense – Understand the threat…
  - Audit the patch management process in your organization
    - How long are your systems unpatched after patches are released?
    - When patches are released, is someone prioritizing and determining whether or not a vulnerability affects your organization?
- Offense – Diff to generate exploits and profit…
  - Use the aforementioned tools and steps to diff Microsoft patches to quickly develop an exploit from private disclosures
  - Practice, Practice, Practice!

# Questions?



- Thank you…
  - Stephen Sims
  - stephen@deadlisting.com
  - @Steph3nSims
- …and remember