Gray Hat Hacking

The Ethical Hacker's Handbook

Sixth Edition

Dr. Allen Harper, Ryan Linn, Stephen Sims,
Michael Baucom, Daniel Fernandez,
Huáscar Tejeda, Moses Frost

Mc Graw Hill

New York Chicago San Francisco
Athens London Madrid Mexico City
Milan New Delhi Singapore Sydney Toronto

Gray Hat Hacking: The Ethical Hacker's Handbook, Sixth Edition

1 2 3 4 5 6 7 8 9   LCR   26 25 24 23 22

Library of Congress Control Number: 2021925396

ISBN   978-1-264-26894-8
MHID   1-264-26894-7

Sponsoring Editor Wendy Rinaldi	**Technical Editor** Heather Linn	**Production Supervisor** Thomas Somers
Editorial Supervisor Janet Walden	**Copy Editor** Bart Reed	**Composition** KnowledgeWorks Global Ltd.
Project Manager Warishree Pant, KnowledgeWorks Global Ltd.	**Proofreader** Rachel Fogelberg	**Illustration** KnowledgeWorks Global Ltd.
Acquisitions Coordinator Emily Walters	**Indexer** Ted Laux	**Art Director, Cover** Jeff Weeks

ABOUT THE AUTHORS

Dr. Allen Harper, CISSP, retired in 2007 from the military as a Marine Corps Officer after a tour in Iraq. He has more than 30 years of IT/security experience. He holds a PhD in IT with a focus on information assurance and security from Capella, an MS in computer science from the Naval Postgraduate School, and a BS in computer engineering from North Carolina State University. In 2004, Allen led the development of the GEN III Honeywall CD-ROM, called roo, for the Honeynet Project. Since then, he has worked as a security consultant for many Fortune 500 and government entities. His interests include the Internet of Things, reverse engineering, vulnerability discovery, and all forms of ethical hacking. Allen was the founder of N2NetSecurity, Inc., served as the EVP and chief hacker at Tangible Security, program director at Liberty University, and now serves as EVP of cybersecurity at T-Rex Solutions, LLC, in Greenbelt, Maryland.

Ryan Linn, CISSP, CSSLP, OSCP, OSCE, GREM, has over 20 years in the security industry, ranging from systems programmer to corporate security to leading a global cybersecurity consultancy. Ryan has contributed to a number of open source projects, including Metasploit, the Browser Exploitation Framework (BeEF), and Ettercap. Ryan participates in Twitter as @sussurro, and he has presented his research at numerous security conferences, including Black Hat, DEF CON, Thotcon, and Derbycon, and has provided training in attack techniques and forensics worldwide.

Stephen Sims is an industry expert with over 15 years of experience in information technology and security. Stephen currently works out of the San Francisco Bay Area as a consultant. He has spent many years performing security architecture, exploit development, reverse engineering, and penetration testing for various Fortune 500 companies, and he has discovered and responsibly disclosed a wide range of vulnerabilities in commercial products. Stephen has an MS in information assurance from Norwich University and currently leads the Offensive Operations curriculum at the SANS Institute. He is the author of the SANS Institute's only 700-level course, SEC760: Advanced Exploit Development for Penetration Testers, which concentrates on complex heap overflows, patch diffing, and client-side exploits. He holds the GIAC Security Expert (GSE) certification as well as the CISA, Immunity NOP, and many others. In his spare time, Stephen enjoys snowboarding and writing music.

Michael Baucom has over 25 years of industry experience, ranging from embedded systems development to leading the product security and research division at Tangible Security. With more than 15 years of security experience, he has performed security assessments of countless systems across a multitude of areas, including medical, industrial, networking, and consumer electronics. Michael has been a trainer at Black Hat, speaker at several conferences, and both an author and technical editor for *Gray Hat Hacking: The Ethical Hacker's Handbook*. His current interests are in embedded system security and development.

Huáscar Tejeda is the co-founder and CEO of F2TC Cyber Security. He is a seasoned, thoroughly experienced cybersecurity professional, with more than 20 years and notable achievements in IT and telecommunications, developing carrier-grade security solutions and business-critical components for multiple broadband providers. He is highly skilled in security research, penetration testing, Linux kernel hacking, software development, and embedded hardware design. Huáscar is also a member of the SANS Latin America Advisory Group, SANS Purple Team Summit Advisory Board, and contributing author of the SANS Institute's most advanced course, SEC760: Advanced Exploit Development for Penetration Testers.

Daniel Fernandez is a security researcher with over 15 years of industry experience. Over his career, he has discovered and exploited vulnerabilities in a vast number of targets. During the last years, his focus had shifted to hypervisors, where he has found and reported bugs in products such as Microsoft Hyper-V. He has worked at several information security companies, including Blue Frost Security GmbH and Immunity, Inc. Recently, he co-founded TACITO Security. When not breaking software, Daniel enjoys training his working dogs.

Moses Frost started his career in designing and implementing large-scale networks around the year 2000. He has worked with computers in some form or another since the early 1990s. His past employers include TLO, Cisco Systems, and McAfee. At Cisco Systems, he was a lead architect for its Cyber Defense Clinics. This free information security dojo was used in educating individuals from the high school and university levels as well as in many enterprises. At Cisco, he was asked to work on crucial security projects such as industry certifications. Moses is an author and senior instructor at the SANS Institute. His technology interests include web app penetration testing, cloud penetration testing, and red team operations. He currently works as a red team operator at GRIMM.

Disclaimer: The views expressed in this book are those of the authors and not of the U.S. government or any company mentioned herein.

About the Contributor

Jaime Geiger currently works for GRIMM Cyber as a senior software vulnerability research engineer and for SANS as a certified instructor. He is also an avid snowboarder, climber, sailor, and skateboarder.

About the Technical Editor

Heather Linn is a red teamer, penetration tester, threat hunter, and cybersecurity strategist with more than 20 years of experience in the security industry. During her career, she has consulted as a penetration tester and digital forensics investigator and has operated as a senior red team engineer inside Fortune 50 environments. In addition to being an accomplished technical editor, Heather has written and delivered training for multiple security conferences and organizations, including Black Hat USA and Girls Who Code, and she has published exam guides for the CompTIA Pentest+ certification. She holds or has held various certifications, including OSCP, CISSP, GREM, GCFA, GNFA, and CompTIA Pentest+.

CHAPTER 18

Next-Generation Patch Exploitation

In this chapter, we cover the following topics:

- Application and patch diffing
- Binary diffing tools
- Patch management process
- Real-world diffing

In response to the lucrative growth of vulnerability research, the interest level in the binary diffing of patched vulnerabilities continues to rise. Privately disclosed and internally discovered vulnerabilities typically offer limited technical details publicly. The more details released, the easier it is for others to locate the vulnerability. Without these details, patch diffing allows a researcher to quickly identify the code changes related to the mitigation of a vulnerability, which can sometimes lead to successful weaponization. The failure to patch quickly in many organizations presents a lucrative opportunity for offensive security practitioners.

Introduction to Binary Diffing

When changes are made to compiled code such as libraries, applications, and drivers, the delta between the patched and unpatched versions can offer an opportunity to discover vulnerabilities. At its most basic level, binary diffing is the process of identifying the differences between two versions of the same file, such as version 1.2 and 1.3. Arguably, the most common target of binary diffs are Microsoft patches; however, this can be applied to many different types of compiled code. Various tools are available to simplify the process of binary diffing, thus quickly allowing an examiner to identify code changes between versions of a disassembled file.

Application Diffing

New versions of applications are commonly released in an ongoing manner. The reasoning behind the release can include the introduction of new features, code changes to support new platforms or kernel versions, leveraging new compile-time security controls such as canaries or Control Flow Guard (CFG), and the fixing of vulnerabilities. Often, the new version can include a combination of the aforementioned reasoning. The more changes to the application code, the more difficult it can be to identify those related to a patched vulnerability. Much of the success in identifying code changes related to vulnerability fixes is dependent on limited disclosures. Many organizations choose to release minimal information as to the nature of a security patch. The more clues we can obtain from this information, the more likely we are to discover the vulnerability. If a disclosure announcement states that there is a vulnerability in the handling and processing of JPEG files, and we identify a changed function named **RenderJpegHeaderType**, we can infer it is related to the patch. These types of clues will be shown in real-world scenarios later in the chapter.

A simple example of a C code snippet that includes a vulnerability is shown here:

```
/*Unpatched code that includes the unsafe gets() function. */
int get_Name(){
    char name[20];
          printf("\nPlease state your name: ");
          gets(name);
          printf("\nYour name is %s.\n\n", name);
          return 0;
}
```

And here's the patched code:

```
/*Patched code that includes the safer fgets() function. */
int get_Name(){
    char name[20];
          printf("\nPlease state your name: ");
          fgets(name, sizeof(name), stdin);
          printf("\nYour name is %s.\n\n", name);
          return 0;
}
```

The problem with the first snippet is the use of the **gets()** function, which offers no bounds checking, resulting in a buffer overflow opportunity. In the patched code, the function **fgets()** is used, which requires a size argument, thus helping to prevent a buffer overflow. The **fgets()** function is considered deprecated and is likely not the best choice due to its inability to properly handle null bytes, such as in binary data; however, it is a better choice than **gets()** if used properly. We will take a look at this simple example later on through the use of a binary diffing tool.

Patch Diffing

Security patches, such as those from Microsoft and Oracle, are some of the most lucrative targets for binary diffing. Microsoft has historically had a well-planned patch management process that follows a monthly schedule, where patches are released on the second Tuesday of each month. The files patched are most often dynamic link libraries (DLLs) and driver files, though plenty of other file types also receive updates, such as .exe files. Many organizations do not patch their systems quickly, leaving open an opportunity for attackers and penetration testers to compromise these systems with publicly disclosed or privately developed exploits through the aid of patch diffing. Starting with Windows 10, Microsoft is much more aggressive with patching requirements, making the deferral of updates challenging. Depending on the complexity of the patched vulnerability, and the difficulty in locating the relevant code, a working exploit can sometimes be developed quickly in the days or weeks following the release of the patch. Exploits developed after reverse-engineering security patches are commonly referred to as *1-day or* n*-day exploits.* This is different from 0-day exploits, where a patch is unavailable at the time it is discovered in the wild.

As we move through this chapter, you will quickly see the benefits of diffing code changes to drivers, libraries, and applications. Though not a new discipline, binary diffing has only continued to gain the attention of security researchers, hackers, and vendors as a viable technique to discover vulnerabilities and profit. The price tag on a 1-day exploit is not typically as high as a 0-day exploit; however, it is not uncommon to see attractive payouts for highly sought-after exploits. As most vulnerabilities are privately disclosed with no publicly available exploit, exploitation framework vendors desire to have more exploits tied to these privately disclosed vulnerabilities than their competitors.

Binary Diffing Tools

Manually analyzing the compiled code of large binaries through the use of a disassembler such as the Interactive Disassembler (IDA) Pro or Ghidra can be a daunting task to even the most skilled researcher. Through the use of freely available and commercially available binary diffing tools, the process of zeroing in on code of interest related to a patched vulnerability can be simplified. Such tools can save hundreds of hours of time spent reversing code that may have no relation to a sought-after vulnerability. Here are some of the most widely known binary diffing tools:

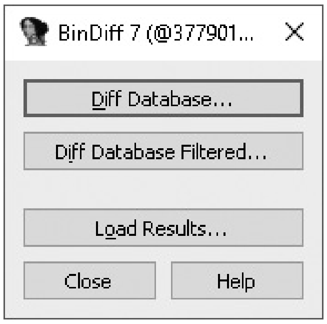
- **Zynamics BinDiff (free)** Acquired by Google in early 2011, Zynamics BinDiff is available at www.zynamics.com/bindiff.html. It requires a licensed version of IDA (or Ghidra).
- **turbodiff (free)** Developed by Nicolas Economou of Core Security, turbodiff is available at https://www.coresecurity.com/core-labs/open-source-tools/turbodiff-cs. It can be used with the free version of IDA 4.9 or 5.0. If the links are not working, try here: https://github.com/nihilus/turbodiff.
- **DarunGrim/binkit (free)** Developed by Jeong Wook Oh (Matt Oh), DarunGrim is available at https://github.com/ohjeongwook/binkit. It requires a recent licensed version of IDA.
- **Diaphora (free)** Developed by Joxean Koret. Diaphora is available at https://github.com/joxeankoret/diaphora. Only the most recent versions of IDA are officially supported.

Each of these tools works as a plug-in to IDA (or Ghidra if noted), using various techniques and heuristics to determine the code changes between two versions of the same file. You may experience different results when using each tool against the same input files. Each of the tools requires the ability to access IDA Database (.idb) files, hence the requirement for a licensed version of IDA, or the free version with turbodiff. For the examples in this chapter, we will use the commercial BinDiff tool as well as turbodiff because it works with the free version of IDA 5.0 that can still be found online at various sites, such as at https://www.scummvm.org/news/20180331/. This allows those without a commercial version of IDA to be able to complete the exercises. The only tools from the list that are actively maintained are Diaphora and BinDiff. The authors of each of these should be highly praised for providing such great tools that save us countless hours trying to find code changes.

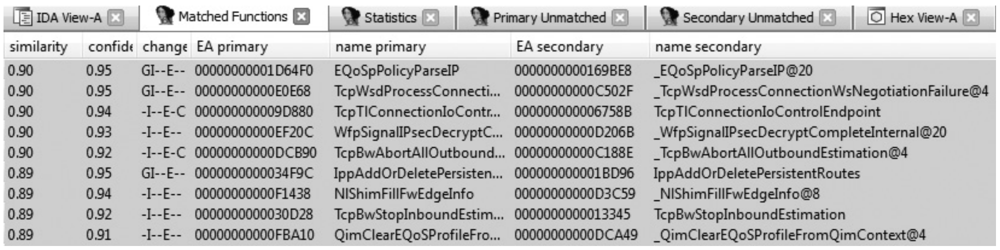
BinDiff

As previously mentioned, in early 2011 Google acquired the German software company Zynamics, with well-known researcher Thomas Dullien, also known as Halvar Flake, who served as the head of research. Zynamics was widely known for the tools BinDiff and BinNavi, both of which aid in reverse engineering. After the acquisition, Google greatly reduced the price of these tools to one-tenth their original price, making them much more accessible. In March 2016, Google announced that, going forward, BinDiff would be free. The project is actively maintained by Christian Blichmann, with BinDiff 7 being the most recent version at the time of this writing. BinDiff is often praised as one of the best tools of its kind, providing deep analysis of block and code changes. As of mid-2021, BinDiff support for Ghidra and Binary Ninja, another great disassembler, was in beta.

BinDiff 7 is delivered as a Windows Installer Package (.msi), Debian Software Package file (.deb), or a Mac OS X Disk Image file (.dmg). Installation requires nothing more than a few clicks, a licensed copy of IDA Pro, and the required version of the Java Runtime Environment. To use BinDiff, you must allow IDA to perform its auto-analysis on the two files you would like to compare and save the IDB files. Once this is complete, and with one of the files open inside of IDA, you press CTRL-6 to bring up the BinDiff GUI, as shown here:

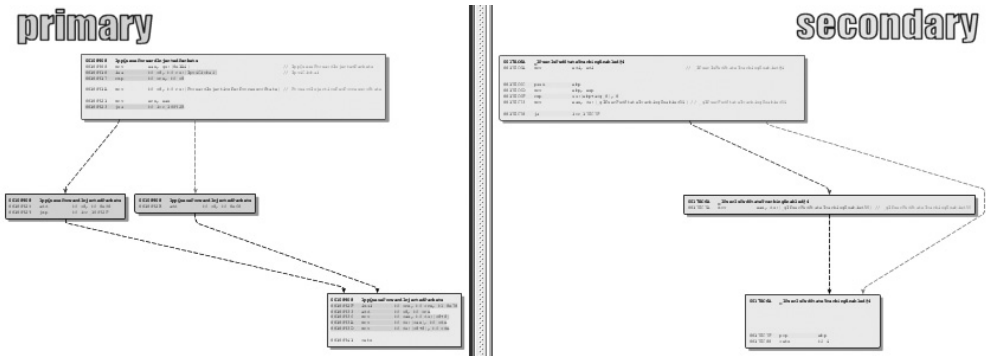


The next step is to click the Diff Database button and select the other IDB file for the diff. Depending on the size of the files, it may take a minute or two to finish. Once the diff is complete, some new tabs will appear in IDA, including Matched Functions, Primary Unmatched, and Secondary Unmatched. The Matched Functions tab contains functions that exist in both files, which may or may not include changes. Each function is scored with a value between 0 and 1.0 in the Similarity column, as shown next. The lower the value, the more the function has changed between the two files. As stated by Zynamics/Google in relation to the Primary Unmatched and Secondary Unmatched tabs, "The first one displays functions that are contained in the currently opened database and were not associated to any function of the diffed database, while the Secondary Unmatched subview contains functions that are in the diffed database but were not associated to any functions in the first."[1]



similarity	confide	change	EA primary	name primary	EA secondary	name secondary
0.90	0.95	GI--E--	00000000001D64F0	EQoSpPolicyParseIP	0000000000169BE8	_EQoSpPolicyParseIP@20
0.90	0.95	GI--E--	00000000000E0E68	TcpWsdProcessConnecti...	00000000000C502F	_TcpWsdProcessConnectionWsNegotiationFailure@4
0.90	0.94	-I--E-C	000000000009D880	TcpTIConnectionIoContr...	000000000006758B	TcpTIConnectionIoControlEndpoint
0.90	0.93	-I--E--	00000000000EF20C	WfpSignalIPsecDecryptC...	00000000000D206B	_WfpSignalIPsecDecryptCompleteInternal@20
0.90	0.92	-I--E-C	00000000000DCB90	TcpBwAbortAllOutbound...	00000000000C188E	_TcpBwAbortAllOutboundEstimation@4
0.89	0.95	GI--E--	000000000034F9C	IppAddOrDeletePersisten...	000000000001BD96	IppAddOrDeletePersistentRoutes
0.89	0.94	-I--E--	00000000000F1438	NlShimFillFwEdgeInfo	00000000000D3C59	_NlShimFillFwEdgeInfo@8
0.89	0.92	-I--E--	000000000030D28	TcpBwStopInboundEstim...	0000000000013345	TcpBwStopInboundEstimation
0.89	0.91	-I--E--	00000000000FBA10	QimClearEQoSProfileFro...	00000000000DCA49	_QimClearEQoSProfileFromQimContext@4

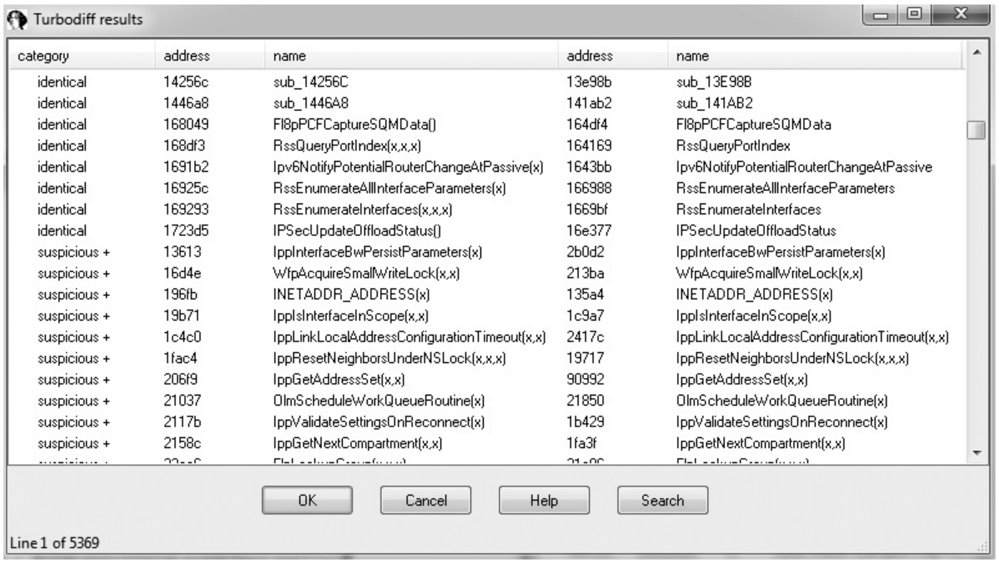
It is important to diff the correct versions of the file to get the most accurate results. When going to Microsoft TechNet to acquire patches published before April 2017, you'll see a column on the far right titled "Updates Replaced." The process of acquiring patches starting in April 2017 is addressed shortly. Going to the URL at that location (Updates Replaced) takes you to the previous most recent update to the file being patched. A file such as jscript9.dll is patched almost every month. If you diff a version of the file from several months earlier with a patch that just came out, the number of differences between the two files will make analysis very difficult. Other files are not patched very often, so clicking the aforementioned Updates Replaced link will take you to the last update to the file in question so you can diff the proper versions. Once a function of interest is identified with BinDiff, a visual diff can be generated either by right-clicking the desired function from the Matched Functions tab and selecting View Flowgraphs or by clicking the desired function and pressing CTRL-E. The following is an example of a visual diff. Note that it is not expected that you can read the disassembly because it is zoomed out to fit onto the page.



turbodiff

The other tool we will cover in this chapter is turbodiff. This tool was selected due to its ability to run with the free version of IDA 5.0. DarunGrim and Diaphora are also great tools; however, a licensed copy of IDA is required to use them, making it impossible for those reading along to complete the exercises in this chapter without already owning or purchasing a licensed copy. DarunGrim and Diaphora are both user friendly and easy to set up with IDA. Literature is available to assist with installation and usage (see the "For Further Reading" section at the end of this chapter). Diffing tools that work with other disassemblers, such as Ghidra, are another alternative.

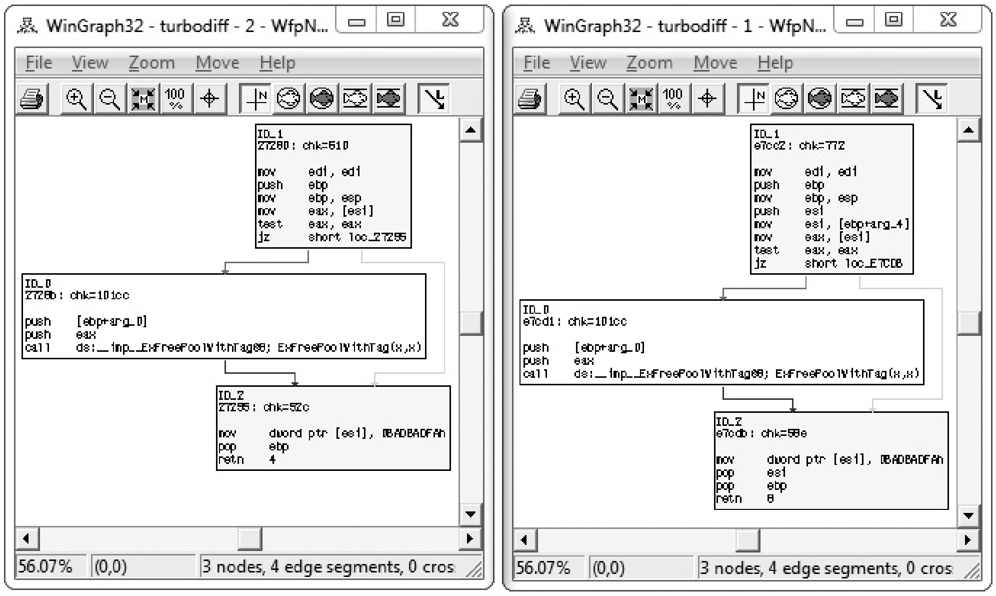
As previously mentioned, the turbodiff plug-in can be acquired from the http://corelabs.coresecurity.com/ website and is free to download and use under the GPLv2 license. The latest stable release is Version 1.01b_r2, released on December 19, 2011. To use turbodiff, you must load the two files to be diffed one at a time into IDA. Once IDA has completed its auto-analysis of the first file, you press CTRL-F11 to bring up the turbodiff pop-up menu. From the options when you're first analyzing a file, choose "take info from this idb" and click OK. Repeat the same steps against the other file to be included in the diff. Once this has been completed against both files to be diffed, press CTRL-F11 again, select the option "compare with…," and then select the other IDB file. The following window should appear.



category	address	name	address	name
identical	14256c	sub_14256C	13e98b	sub_13E98B
identical	1446a8	sub_1446A8	141ab2	sub_141AB2
identical	168049	Fl8pPCFCaptureSQMData()	164df4	Fl8pPCFCaptureSQMData
identical	168df3	RssQueryPortIndex(x,x,x)	164169	RssQueryPortIndex
identical	1691b2	Ipv6NotifyPotentialRouterChangeAtPassive(x)	1643bb	Ipv6NotifyPotentialRouterChangeAtPassive
identical	16925c	RssEnumerateAllInterfaceParameters(x)	166988	RssEnumerateAllInterfaceParameters
identical	169293	RssEnumerateInterfaces(x,x,x)	1669bf	RssEnumerateInterfaces
identical	1723d5	IPSecUpdateOffloadStatus()	16e377	IPSecUpdateOffloadStatus
suspicious +	13613	IppInterfaceBwPersistParameters(x)	2b0d2	IppInterfaceBwPersistParameters(x)
suspicious +	16d4e	WfpAcquireSmallWriteLock(x,x)	213ba	WfpAcquireSmallWriteLock(x,x)
suspicious +	196fb	INETADDR_ADDRESS(x)	135a4	INETADDR_ADDRESS(x)
suspicious +	19b71	IppIsInterfaceInScope(x,x)	1c9a7	IppIsInterfaceInScope(x,x)
suspicious +	1c4c0	IppLinkLocalAddressConfigurationTimeout(x,x)	2417c	IppLinkLocalAddressConfigurationTimeout(x,x)
suspicious +	1fac4	IppResetNeighborsUnderNSLock(x,x,x)	19717	IppResetNeighborsUnderNSLock(x,x,x)
suspicious +	206f9	IppGetAddressSet(x,x)	90992	IppGetAddressSet(x,x)
suspicious +	21037	OlmScheduleWorkQueueRoutine(x)	21850	OlmScheduleWorkQueueRoutine(x)
suspicious +	2117b	IppValidateSettingsOnReconnect(x)	1b429	IppValidateSettingsOnReconnect(x)
suspicious +	2158c	IppGetNextCompartment(x,x)	1fa3f	IppGetNextCompartment(x,x)

Line 1 of 5369

In the category column you can see labels such as identical, suspicious +, suspicious ++, and changed. Each label has a meaning and can help the examiner zoom in on the most interesting functions, primarily the ones labeled suspicious + and suspicious ++. These labels indicate that the checksums in one or more of the blocks within the selected function are mismatched, as well as whether or not the number of instructions has changed. When you double-click a desired function name, a visual diff is presented, with each function appearing in its own window, as shown here:


WinGraph32 - turbodiff - 2 - WfpN...
File
View
Zoom
Move
Help
56.07%
(0,0)
3 nodes, 4 edge segments, 0 cros
WinGraph32 - turbodiff - 1 - WfpN...
File
View
Zoom
Move
Help
56.07%
(0,0)
3 nodes, 4 edge segments, 0 cros

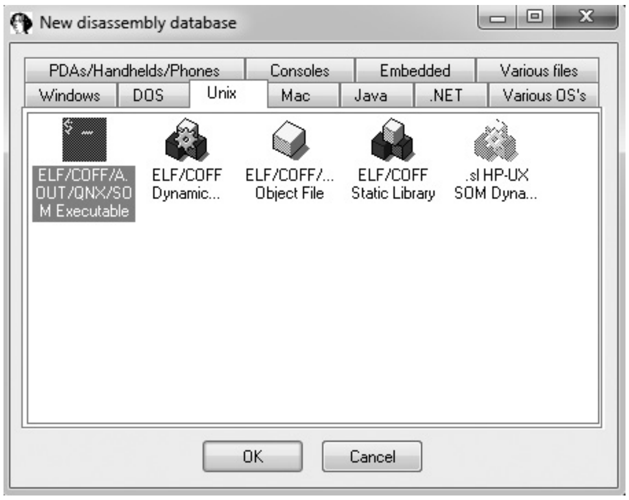
```
ID_1
27280 : chk=510

mov     edi, edi
push    ebp
mov     ebp, esp
mov     eax, [esi]
test    eax, eax
jz      short loc_27255

ID_0
2728b : chk=101cc

push    [ebp+arg_0]
push    eax
call    ds:__imp__ExFreePoolWithTag@8; ExFreePoolWithTag(x,x)

ID_2
27255 : chk=52c

mov     dword ptr [esi], 0BADBADFAh
pop     ebp
retn    4
```

```
ID_1
e7cc2 : chk=772

mov     edi, edi
push    ebp
mov     ebp, esp
push    esi
mov     esi, [ebp+arg_4]
mov     eax, [esi]
test    eax, eax
jz      short loc_E7CDB

ID_0
e7cd1 : chk=101cc

push    [ebp+arg_0]
push    eax
call    ds:__imp__ExFreePoolWithTag@8; ExFreePoolWithTag(x,x)

ID_2
e7cdb : chk=58e

mov     dword ptr [esi], 0BADBADFAh
pop     esi
pop     ebp
retn    8
```

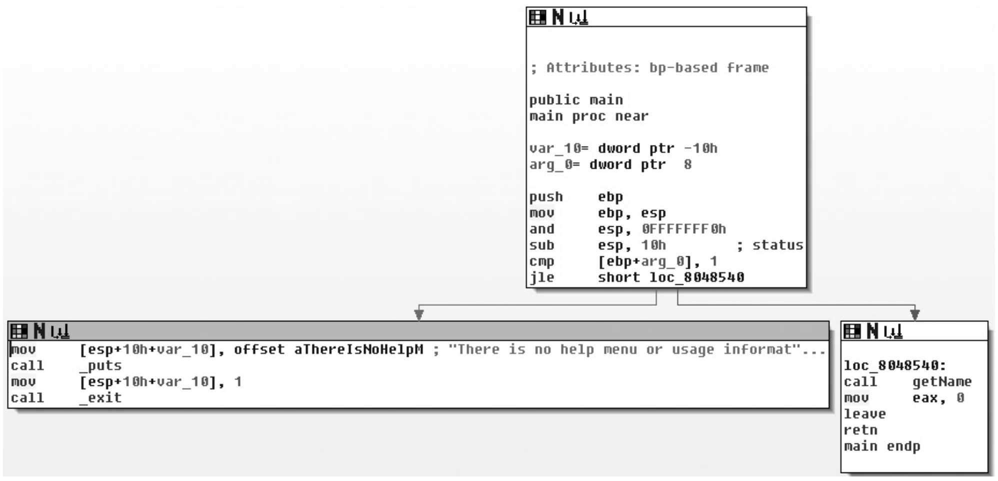
Lab 18-1: Our First Diff

NOTE Copy the two ELF binary files name and name2 from Lab1 of the book's repository and place them in the folder C:\grayhat\app_diff\. You will need to create the app_diff subfolder. If you do not have a C:\grayhat folder, you can create one now, or you can use a different location.

In this lab, you will perform a simple diff against the code previously shown in the "Application Diffing" section. The ELF binary files name and name2 are to be compared. The name file is the unpatched one, and name2 is the patched one. You must first start up the free IDA 5.0 application you previously installed. Once it is up and running, go to File | New, select the Unix tab from the pop-up, and click the ELF option on the left, as shown here, and then click OK.



Navigate to your C:\grayhat\app_diff\ folder and select the file "name." Accept the default options that appear. IDA should quickly complete its auto-analysis, defaulting to the **main()** function in the disassembly window, as shown next.



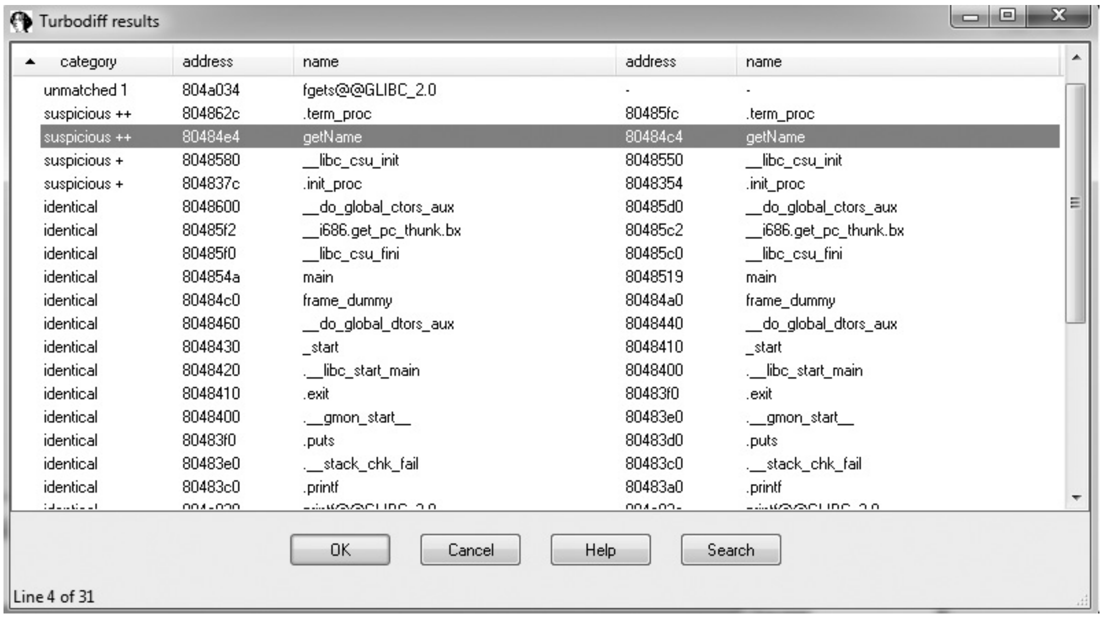
```
; Attributes: bp-based frame

public main
main proc near

var_10= dword ptr -10h
arg_0= dword ptr  8

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 10h        ; status
cmp     [ebp+arg_0], 1
jle     short loc_8048540
```

```
mov     [esp+10h+var_10], offset aThereIsNoHelpM ; "There is no help menu or usage informat"...
call    _puts
mov     [esp+10h+var_10], 1
call    _exit
```

```
loc_8048540:
call    getName
mov     eax, 0
leave
retn
main endp
```

Press CTRL-F11 to bring up the turbodiff pop-up. If it does not appear, go back and ensure you properly copied over the necessary files for turbodiff. With the turbodiff window on the screen, select the option "take info from this idb" and click OK, followed by another OK. Next, go to File | New, and you will get a pop-up box asking if you would like to save the database. Accept the defaults and click OK. Repeat the steps of selecting the Unix tab | ELF Executable, and then click OK. Open up the name2 ELF binary file and accept the defaults. Repeat the steps of bringing up the turbodiff pop-up and choosing the option "take info from this idb."

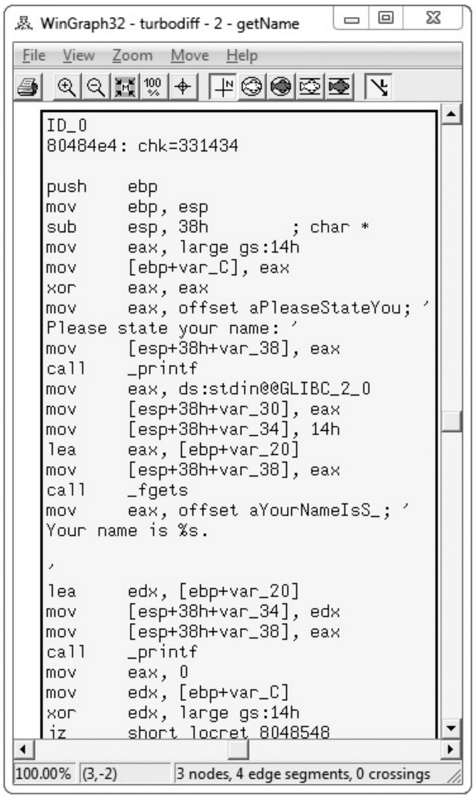
Now that you have completed this for both files, press CTRL-F11 again, with the name2 file still open in IDA. Select the option "compare with…" and click OK. Select the name .idb file and click OK, followed by another OK. The following box should appear (you may have to sort by category to replicate the exact image).



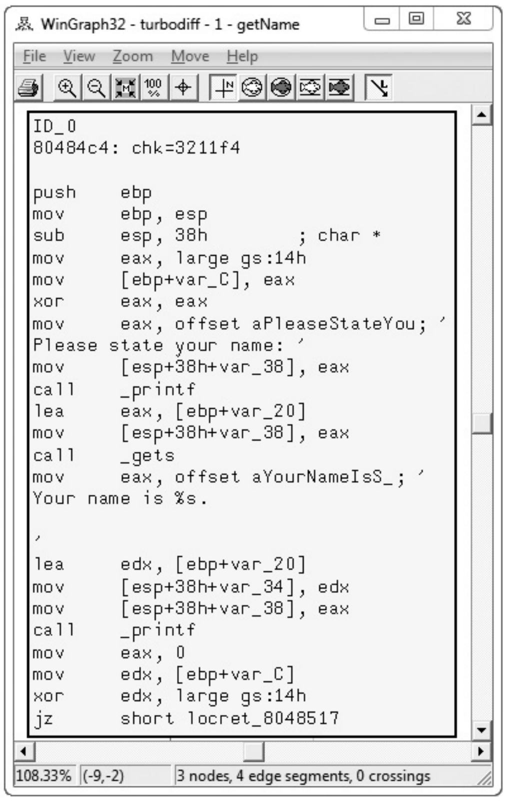
category	address	name	address	name
unmatched 1	804a034	fgets@@GLIBC_2.0	-	-
suspicious ++	804862c	.term_proc	80485fc	.term_proc
suspicious ++	80484e4	getName	80484c4	getName
suspicious +	8048580	__libc_csu_init	8048550	__libc_csu_init
suspicious +	804837c	.init_proc	8048354	.init_proc
identical	8048600	__do_global_ctors_aux	80485d0	__do_global_ctors_aux
identical	80485f2	__i686.get_pc_thunk.bx	80485c2	__i686.get_pc_thunk.bx
identical	80485f0	__libc_csu_fini	80485c0	__libc_csu_fini
identical	804854a	main	8048519	main
identical	80484c0	frame_dummy	80484a0	frame_dummy
identical	8048460	__do_global_dtors_aux	8048440	__do_global_dtors_aux
identical	8048430	_start	8048410	_start
identical	8048420	.__libc_start_main	8048400	.__libc_start_main
identical	8048410	.exit	80483f0	.exit
identical	8048400	.__gmon_start__	80483e0	.__gmon_start__
identical	80483f0	.puts	80483d0	.puts
identical	80483e0	.__stack_chk_fail	80483c0	.__stack_chk_fail
identical	80483c0	.printf	80483a0	.printf

Line 4 of 31

Note that the **getName()** function is labeled "suspicious ++." Double-click the **getName()** function to get the following window:


WinGraph32 - turbodiff - 2 - getName
File View Zoom Move Help
100.00% (3,-2) 3 nodes, 4 edge segments, 0 crossings

```
ID_0
80484e4: chk=331434

push    ebp
mov     ebp, esp
sub     esp, 38h        ; char *
mov     eax, large gs:14h
mov     [ebp+var_C], eax
xor     eax, eax
mov     eax, offset aPleaseStateYou; '
Please state your name: '
mov     [esp+38h+var_38], eax
call    _printf
mov     eax, ds:stdin@@GLIBC_2_0
mov     [esp+38h+var_30], eax
mov     [esp+38h+var_34], 14h
lea     eax, [ebp+var_20]
mov     [esp+38h+var_38], eax
call    _fgets
mov     eax, offset aYourNameIsS_; '
Your name is %s.

'
lea     edx, [ebp+var_20]
mov     [esp+38h+var_34], edx
mov     [esp+38h+var_38], eax
call    _printf
mov     eax, 0
mov     edx, [ebp+var_C]
xor     edx, large gs:14h
jz      short locret_8048548
```


WinGraph32 - turbodiff - 1 - getName
File View Zoom Move Help
108.33% (-9,-2) 3 nodes, 4 edge segments, 0 crossings

```
ID_0
80484c4: chk=3211f4

push    ebp
mov     ebp, esp
sub     esp, 38h        ; char *
mov     eax, large gs:14h
mov     [ebp+var_C], eax
xor     eax, eax
mov     eax, offset aPleaseStateYou; '
Please state your name: '
mov     [esp+38h+var_38], eax
call    _printf
lea     eax, [ebp+var_20]
mov     [esp+38h+var_38], eax
call    _gets
mov     eax, offset aYourNameIsS_; '
Your name is %s.

'
lea     edx, [ebp+var_20]
mov     [esp+38h+var_34], edx
mov     [esp+38h+var_38], eax
call    _printf
mov     eax, 0
mov     edx, [ebp+var_C]
xor     edx, large gs:14h
jz      short locret_8048517
```

In this image, the left window shows the patched function and the right window shows the unpatched function. The unpatched block uses the **gets()** function, which provides no bounds checking. The patched block uses the **fgets()** function, which requires a size argument to help prevent buffer overflows. The patched disassembly is shown here:

```
mov     eax, ds:stdin@@GLIBC_2_0
mov     [esp+38h+var_30], eax
mov     [esp+38h+var_34], 14h
lea     eax, [ebp+var_20]
mov     [esp+38h+var_38], eax
call    _fgets
```

There were a couple of additional blocks of code within the two functions, but they are white and include no changed code. They are simply the stack-smashing protector code, which validates stack canaries, followed by the function epilog. At this point, you have completed the lab. Moving forward, we will look at real-world diffs.

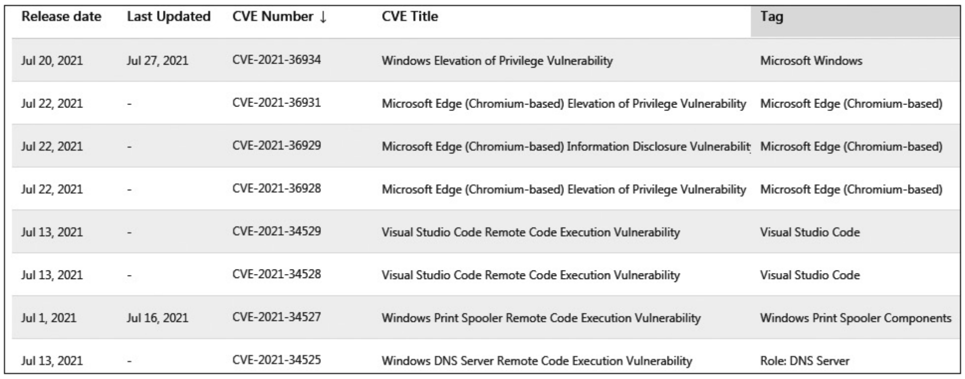
Patch Management Process

Each vendor has its own process for distributing patches, including Oracle, Microsoft, and Apple. Some vendors have a set schedule as to when patches are released, whereas others have no set schedule. Having an ongoing patch release cycle, such as that used by Microsoft, allows for those responsible for managing a large number of systems to plan accordingly. Out-of-band patches can be problematic for organizations because there may not be resources readily available to roll out the updates. We will focus primarily on the Microsoft patch management process because it is a mature process that is often targeted for the purpose of diffing to discover vulnerabilities for profit.

Microsoft Patch Tuesday

The second Tuesday of each month is Microsoft's monthly patch cycle, with the occasional out-of-band patch due to a critical update. The process significantly changed with the introduction of Windows 10 cumulative updates, taking effect on Windows 8 as of October 2016, as well as a change in the way patches are downloaded. Up until April 2017, a summary and security patches for each update could be found at https://technet.microsoft.com/en-us/security/bulletin. Starting in April 2017, patches are acquired from the Microsoft Security TechCenter site at https://www.catalog.update.microsoft.com/Home.aspx, with summary information at https://msrc.microsoft.com/update-guide/releaseNote/. Patches are commonly obtained by using the Windows Update tool from the Windows Control Panel or managed centrally by a product such as Windows Server Update Services (WSUS) or Windows Update for Business (WUB). When patches are desired for diffing, they can be obtained from the aforementioned TechNet link, using the search syntax of (YYYY-MM Build_Number Architecture), such as "2021-07 21H1 x64."

Each patch bulletin is linked to more information about the update. Some updates are the result of a publicly discovered vulnerability, whereas the majority are through some form of coordinated private disclosure. The following link lists the CVE numbers associated with the patched updates: https://msrc.microsoft.com/update-guide/vulnerability.



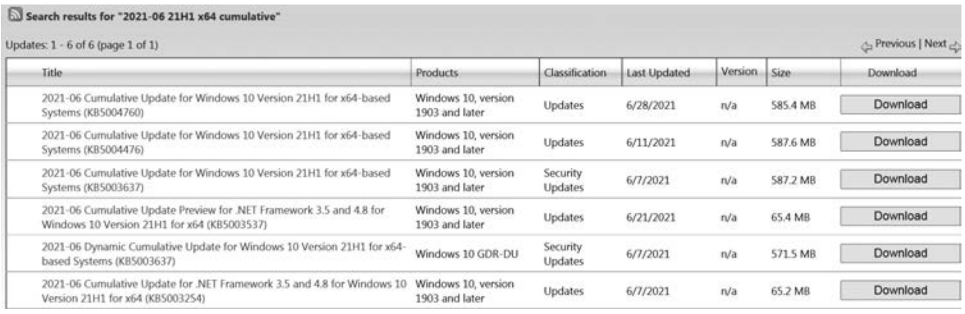
Release date	Last Updated	CVE Number ↓	CVE Title	Tag
Jul 20, 2021	Jul 27, 2021	CVE-2021-36934	Windows Elevation of Privilege Vulnerability	Microsoft Windows
Jul 22, 2021	-	CVE-2021-36931	Microsoft Edge (Chromium-based) Elevation of Privilege Vulnerability	Microsoft Edge (Chromium-based)
Jul 22, 2021	-	CVE-2021-36929	Microsoft Edge (Chromium-based) Information Disclosure Vulnerabilit	Microsoft Edge (Chromium-based)
Jul 22, 2021	-	CVE-2021-36928	Microsoft Edge (Chromium-based) Elevation of Privilege Vulnerability	Microsoft Edge (Chromium-based)
Jul 13, 2021	-	CVE-2021-34529	Visual Studio Code Remote Code Execution Vulnerability	Visual Studio Code
Jul 13, 2021	-	CVE-2021-34528	Visual Studio Code Remote Code Execution Vulnerability	Visual Studio Code
Jul 1, 2021	Jul 16, 2021	CVE-2021-34527	Windows Print Spooler Remote Code Execution Vulnerability	Windows Print Spooler Components
Jul 13, 2021	-	CVE-2021-34525	Windows DNS Server Remote Code Execution Vulnerability	Role: DNS Server

When you click the associated links, only limited information is provided about the vulnerability. The more information provided, the more likely someone is quickly able to locate the patched code and produce a working exploit. Depending on the size of the update and the complexity of the vulnerability, the discovery of the patched code alone can be challenging. Often, a vulnerable condition is only theoretical, or can only be triggered under very specific conditions. This can increase the difficulty in determining the root cause and producing proof-of-concept code that successfully triggers the bug. Once the root cause is determined and the vulnerable code is reached and available for analysis in a debugger, it must be determined how difficult it will be to gain code execution, if applicable.

Obtaining and Extracting Microsoft Patches

Let's look at an example of acquiring and extracting a cumulative update for Windows 10. When we look at the prior list of CVEs for July 2021, we see that CVE-2021-34527 says, "Windows Print Spooler Remote Code Execution Vulnerability." This is the vulnerability named "PrintNightmare," as can be seen in the Microsoft announcement at https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-34527. There were various patches released between June 2021 and August 2021 and beyond. For this walk-through, we will download the June 2021 and July 2021 cumulative update for Windows 10 21H1 x64. Our goal is to locate the vulnerable and patched file associated with PrintNightmare and get some initial information as to how it was corrected.

We must first go to https://www.catalog.update.microsoft.com/Home.aspx and enter the search criteria of **2021-06 21H1 x64 cumulative**. When doing this we get the following results:

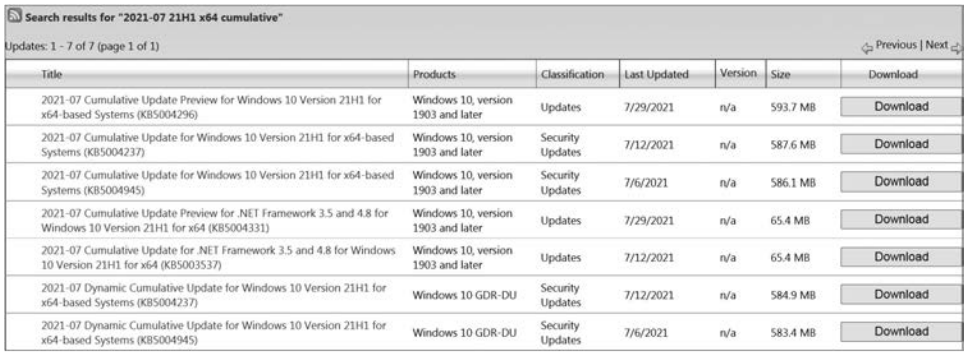


Search results for "2021-06 21H1 x64 cumulative"

Updates: 1 - 6 of 6 (page 1 of 1)

Title	Products	Classification	Last Updated	Version	Size	Download
2021-06 Cumulative Update for Windows 10 Version 21H1 for x64-based Systems (KB5004760)	Windows 10, version 1903 and later	Updates	6/28/2021	n/a	585.4 MB	Download
2021-06 Cumulative Update for Windows 10 Version 21H1 for x64-based Systems (KB5004476)	Windows 10, version 1903 and later	Updates	6/11/2021	n/a	587.6 MB	Download
2021-06 Cumulative Update for Windows 10 Version 21H1 for x64-based Systems (KB5003637)	Windows 10, version 1903 and later	Security Updates	6/7/2021	n/a	587.2 MB	Download
2021-06 Cumulative Update Preview for .NET Framework 3.5 and 4.8 for Windows 10 Version 21H1 for x64 (KB5003537)	Windows 10, version 1903 and later	Updates	6/21/2021	n/a	65.4 MB	Download
2021-06 Dynamic Cumulative Update for Windows 10 Version 21H1 for x64-based Systems (KB5003637)	Windows 10 GDR-DU	Security Updates	6/7/2021	n/a	571.5 MB	Download
2021-06 Cumulative Update for .NET Framework 3.5 and 4.8 for Windows 10 Version 21H1 for x64 (KB5003254)	Windows 10, version 1903 and later	Updates	6/7/2021	n/a	65.2 MB	Download

We will download the file "2021-06 Cumulative Update for Windows 10 Version 21H1 for x64-based Systems (KB5004476)." Next, we will change the search criteria to **2021-07 21H1 x64 cumulative**. The results are shown next.



Search results for "2021-07 21H1 x64 cumulative"

Updates: 1 - 7 of 7 (page 1 of 1)

Title	Products	Classification	Last Updated	Version	Size	Download
2021-07 Cumulative Update Preview for Windows 10 Version 21H1 for x64-based Systems (KB5004296)	Windows 10, version 1903 and later	Updates	7/29/2021	n/a	593.7 MB	Download
2021-07 Cumulative Update for Windows 10 Version 21H1 for x64-based Systems (KB5004237)	Windows 10, version 1903 and later	Security Updates	7/12/2021	n/a	587.6 MB	Download
2021-07 Cumulative Update for Windows 10 Version 21H1 for x64-based Systems (KB5004945)	Windows 10, version 1903 and later	Security Updates	7/6/2021	n/a	586.1 MB	Download
2021-07 Cumulative Update Preview for .NET Framework 3.5 and 4.8 for Windows 10 Version 21H1 for x64 (KB5004331)	Windows 10, version 1903 and later	Updates	7/29/2021	n/a	65.4 MB	Download
2021-07 Cumulative Update for .NET Framework 3.5 and 4.8 for Windows 10 Version 21H1 for x64 (KB5003537)	Windows 10, version 1903 and later	Updates	7/12/2021	n/a	65.4 MB	Download
2021-07 Dynamic Cumulative Update for Windows 10 Version 21H1 for x64-based Systems (KB5004237)	Windows 10 GDR-DU	Security Updates	7/12/2021	n/a	584.9 MB	Download
2021-07 Dynamic Cumulative Update for Windows 10 Version 21H1 for x64-based Systems (KB5004945)	Windows 10 GDR-DU	Security Updates	7/6/2021	n/a	583.4 MB	Download

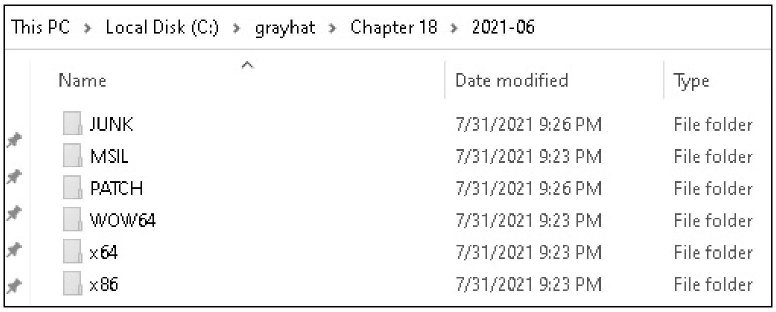
We will download the file "2021-07 Cumulative Update for Windows 10 Version 21H1 for x64-based Systems (KB5004237)." We now have both cumulative updates, which should include the files needed to look at CVE-2021-34527, but they must be extracted.

Patches can be manually extracted using the **expand** tool from Microsoft, included on most versions of Windows. The tool expands files from a compressed format, such as a cabinet file or Microsoft Standalone Update package (MSU). When the **-F:** argument is used to specify a file, wildcards are supported with the * character. The command would look something like **expand.exe -F:* <file to extract> <destination>**. When you run this command against a downloaded cumulative update, a Patch Storage File (PSF) with a .cab extension is quickly extracted. The same **expand** command must be applied to this file in order to extract the contents. This will take some time to run (likely more than 10 minutes), as there are typically tens of thousands of folders and files. For the sake of brevity, we will not dive into the associated internal structure and hierarchy associated with patch file internals, except for those necessary to quickly get into patch diffing. To help speed things up, we will use the PatchExtract tool from Greg Linares, which makes use of the **expand** tool. An updated version from Jaime Geiger is listed in the "For Further Reading" section and is the version used in this chapter.

The PatchExtract tool is a PowerShell script that both extracts the patch file contents and neatly organizes the files into various folders. In order to use the tool, it is a good idea to create a destination folder as to where you want the extracted files to be placed. For our purposes, we will name one folder "2021-06" and a second folder "2021-07." We will extract the contents of the June 2021 update to the "2021-06" folder and the contents of the July 2021 update to the "2021-07" folder. With the June 2021 .msu cumulative update file copied into the "2021-06" folder, we run the following command (entered all on one line) using a PowerShell ISE session:

```
PS C:\grayhat\Chapter 18> ..\PatchExtract.ps1 -PATCH .\windows10.0-kb5004296-
x64_1d54ad8c53ce045b7ad48b0cdb05d618c06198d9.msu -PATH . | Out-Null
```

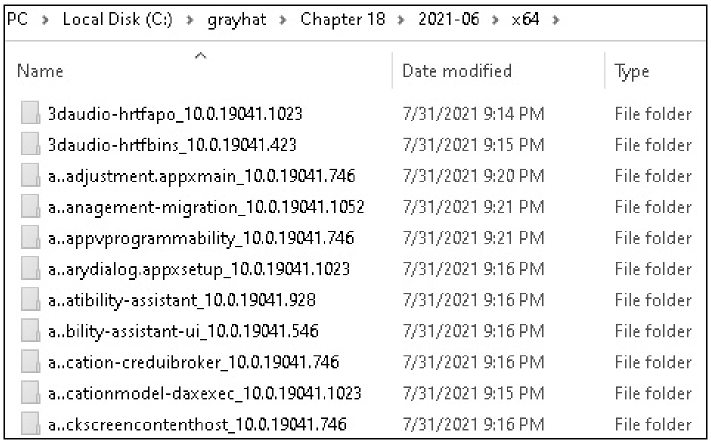
After this command was executed, it took about 20 minutes for the files to be extracted. There were also a few PowerShell messages about names already existing, but nothing preventing the patch from being fully extracted. Upon completion, we are left with various folders, including JUNK, MSIL, PATCH, WOW64, x64, and x86. The JUNK folder contains files that we are not interested in, such as manifest files and security catalog files. The PATCH folder contains the larger nested cabinet files we just extracted. The MSIL, WOW64, x64, and x86 folders contain the bulk of the platform data and patch files in which we are interested.



This PC > Local Disk (C:) > grayhat > Chapter 18 > 2021-06

Name	Date modified	Type
JUNK	7/31/2021 9:26 PM	File folder
MSIL	7/31/2021 9:23 PM	File folder
PATCH	7/31/2021 9:26 PM	File folder
WOW64	7/31/2021 9:23 PM	File folder
x64	7/31/2021 9:23 PM	File folder
x86	7/31/2021 9:23 PM	File folder

Inside the x64 folder are over 2,900 subfolders, all with different descriptive names, as seen here:



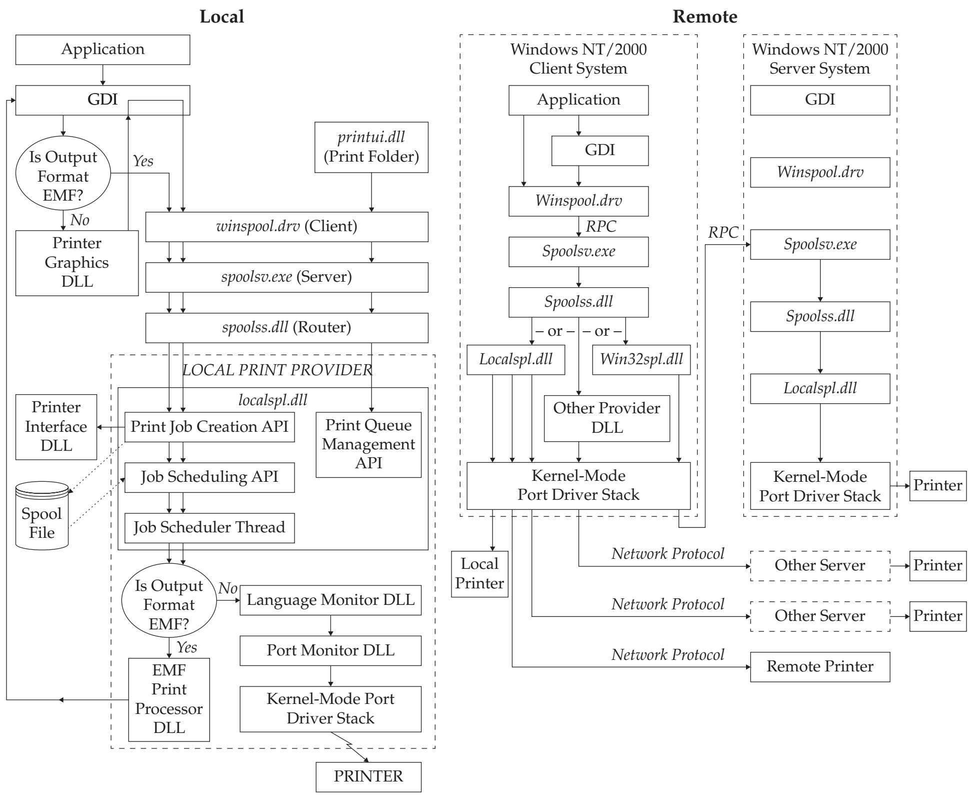
PC > Local Disk (C:) > grayhat > Chapter 18 > 2021-06 > x64 >

Name	Date modified	Type
3daudio-hrtfapo_10.0.19041.1023	7/31/2021 9:14 PM	File folder
3daudio-hrtfbins_10.0.19041.423	7/31/2021 9:15 PM	File folder
a..adjustment.appxmain_10.0.19041.746	7/31/2021 9:20 PM	File folder
a..anagement-migration_10.0.19041.1052	7/31/2021 9:21 PM	File folder
a..appvprogrammability_10.0.19041.746	7/31/2021 9:21 PM	File folder
a..arydialog.appxsetup_10.0.19041.1023	7/31/2021 9:16 PM	File folder
a..atibility-assistant_10.0.19041.928	7/31/2021 9:16 PM	File folder
a..bility-assistant-ui_10.0.19041.546	7/31/2021 9:16 PM	File folder
a..cation-creduibroker_10.0.19041.746	7/31/2021 9:16 PM	File folder
a..cationmodel-daxexec_10.0.19041.1023	7/31/2021 9:15 PM	File folder
a..ckscreencontenthost_10.0.19041.746	7/31/2021 9:16 PM	File folder

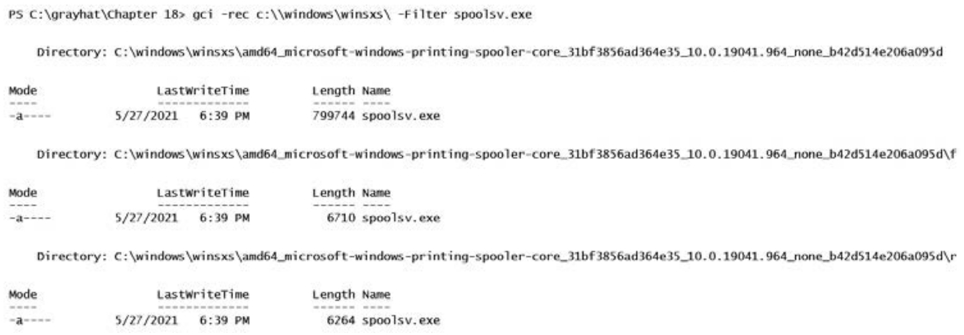
Inside each of these folders are typically two subfolders, called "f" and "r," which stand for forward and reverse, respectively. Another subfolder name you may come across is "n," which stands for null. These folders include the delta patch files. The "r" folder contains the reverse differential files, the "f" folder contains the forward differential files, and the "n" folder contains new files to be added. It used to be the case where the patch included the entire file to be replaced, such as a DLL or driver. Microsoft changed to the delta format in which the reverse differential file takes the updated file, once installed, back to the Release To Manufacturing (RTM) version, and the forward differential takes the file from RTM to where it needs to be for the current update.[2] If a new file is added to the system on Patch Tuesday, via the null folder, it could be considered the RTM version. Once that file is patched during a subsequent Patch Tuesday update, a forward differential can be applied to make it current. This update will also come with a reverse differential file that can be applied to take the file back to the RTM version so that a future forward differential can be applied to continue to make it current.

As mentioned, once upon a time, Microsoft patches would include the entire files to replace the ones being patched; however, if you take a look at the patch files within the f and r folders, you will quickly notice that the file size of the supposed DLLs or drivers is far too small to be the entire file. A number of years ago, Microsoft created a set of patch delta APIs. The current API is the MSDELTA API.[3] It includes a set of functions to perform actions, such as applying a patch delta. Jaime Geiger created a script called "delta_patch.py" to utilize the API in order to apply reverse and forward deltas, which we will use shortly. The delta patch files include a 4-byte CRC32 checksum at the beginning of the file, followed by a magic number of PA30.[3, 4]

Before we move onto applying patch deltas, we need to identify a file related to a patch in which we are interested. CVE-2021-34527 is related to the "PrintNightmare" vulnerability. In order to determine which files we are interested in diffing, we need to understand a bit more about spooling services on Windows. Take a look at the following image, from Microsoft, which shows both local and remote printer provider components:[5]



We can see a few candidates to diff in the images, including winspool.drv, spoolsv.exe, spools.dll, and localspl.dll. The vulnerability associated with PrintNightmare indicated the potential for remote code execution (RCE). In the image on the right, we can see an RPC call to spoolsv.exe. In our preliminary analysis, it was determined that spoolsv.exe, winspool.drv, and localspl.dll are the most interesting targets. We will start with analyzing spoolsv.exe. Our next step is to apply the patch deltas for the June 2021 and July 2021 updates. We must identify a copy of spoolsv.exe from our Windows 10 WinSxS folder, apply the associated reverse delta, and then apply the forward delta for each of the two months. WinSxS is the Windows side-by-side assembly technology. In short, it is a way for Windows to manage various versions of DLLs and other types of files. Windows needs a way to replace updated files, while also having a way to revert back to older versions if an update is uninstalled. The large number of DLLs and system files can become complex to manage. We will look through the WinSxS folder to find a copy of spoolsv .exe, and its associated reverse delta patch, in order to take it back to RTM. Take a look at the following PowerShell command and associated results:



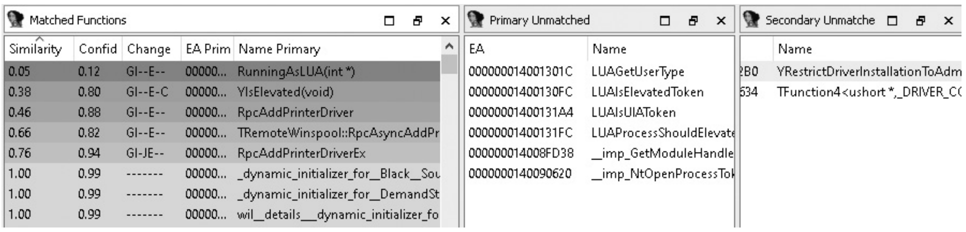
```
PS C:\grayhat\Chapter 18> gci -rec c:\\windows\winsxs\ -Filter spoolsv.exe

    Directory: C:\windows\winsxs\amd64_microsoft-windows-printing-spooler-core_31bf3856ad364e35_10.0.19041.964_none_b42d514e206a095d

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----         5/27/2021   6:39 PM        799744 spoolsv.exe

    Directory: C:\windows\winsxs\amd64_microsoft-windows-printing-spooler-core_31bf3856ad364e35_10.0.19041.964_none_b42d514e206a095d\f

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----         5/27/2021   6:39 PM          6710 spoolsv.exe

    Directory: C:\windows\winsxs\amd64_microsoft-windows-printing-spooler-core_31bf3856ad364e35_10.0.19041.964_none_b42d514e206a095d\r

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----         5/27/2021   6:39 PM          6264 spoolsv.exe
```

We can see a spoolsv.exe file from May 2021, along with an r folder and an f folder, which includes the delta patch files. We will create a spoolsv folder in our C:\grayhat\ Chapter 18\ folder and then copy the full spoolsv.exe file, along with the r folder and its contents. This will allow us to apply the reverse delta patch, followed by using the forward delta patch from the June 2021 and July 2021 updates to the file, using the delta_patch.py tool.

```
PS C:\grayhat\Chapter 18> .\delta_patch.py -i .\spoolsv\spoolsv.exe -o .\
spoolsv.2021-06.exe .\spoolsv\r\spoolsv.exe .\2021-06\x64\printing-spooler-
core_10.0.19041.1052\f\spoolsv.exe
Applied 2 patches successfully
Final hash: kXOpI3uCt6K/gNfXD/ZfCaiQl8sy8EcluGHY+vZRX5o=

PS C:\grayhat\Chapter 18> .\delta_patch.py -i .\spoolsv\spoolsv.exe -o .\
spoolsv.2021-07.exe .\spoolsv\r\spoolsv.exe .\2021-07\x64\printing-spooler-
core_10.0.19041.1083\f\spoolsv.exe
Applied 2 patches successfully
Final hash: 0+G8zsSJmi5O1RIHgwYYSA9qNUSc+lFjgcCxryrt7Dg=
```

As you can see, the reverse and forward delta patches were applied successfully. We now have the spoolsv.exe file versions for both June and July. We will use the BinDiff plug-in for IDA Pro to compare the differences between the two versions. To do so, we will need to perform the following actions:

- Have IDA perform its auto-analysis against both files.
- Load the June version into IDA and press CTRL-6 to bring up the BinDiff menu.
- Perform the diff and analyze the results.


Matched Functions

Similarity	Confid	Change	EA Prim	Name Primary
0.05	0.12	GI--E--	00000...	RunningAsLUA(int *)
0.38	0.80	GI--E-C	00000...	YIsElevated(void)
0.46	0.88	GI--E--	00000...	RpcAddPrinterDriver
0.66	0.82	GI--E--	00000...	TRemoteWinspool::RpcAsyncAddPr
0.76	0.94	GI-JE--	00000...	RpcAddPrinterDriverEx
1.00	0.99	-------	00000...	_dynamic_initializer_for__Black__Sou
1.00	0.99	-------	00000...	_dynamic_initializer_for__DemandSt
1.00	0.99	-------	00000...	wil__details___dynamic_initializer_fo

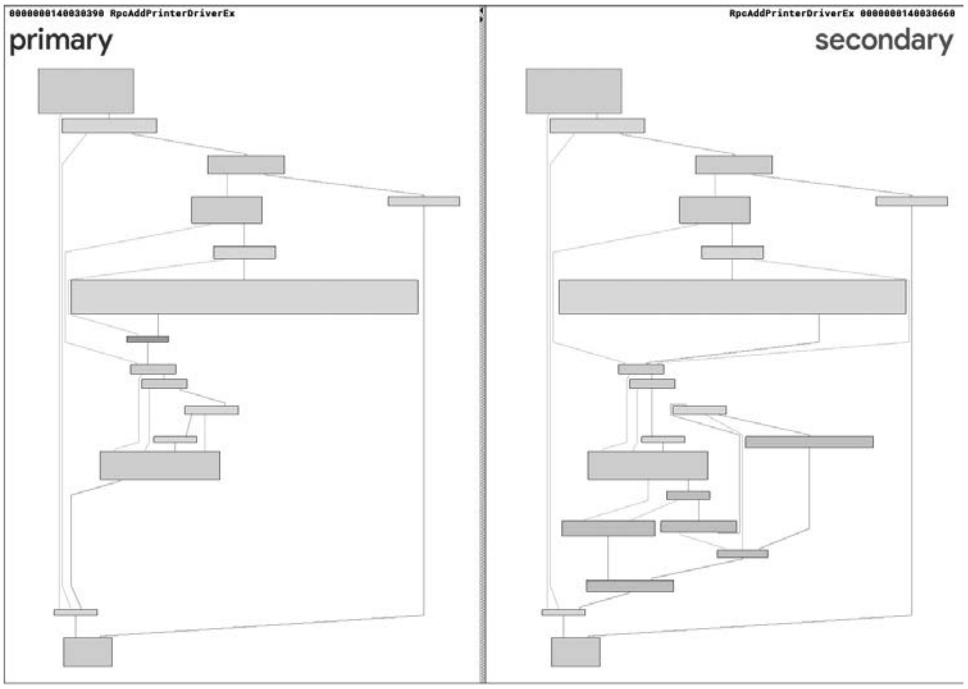
Primary Unmatched

EA	Name
000000014001301C	LUAGetUserType
00000001400130FC	LUAIsElevatedToken
00000001400131A4	LUAIsUIAToken
00000001400131FC	LUAProcessShouldElevate
000000014008FD38	__imp_GetModuleHandle
0000000140090620	__imp_NtOpenProcessTok

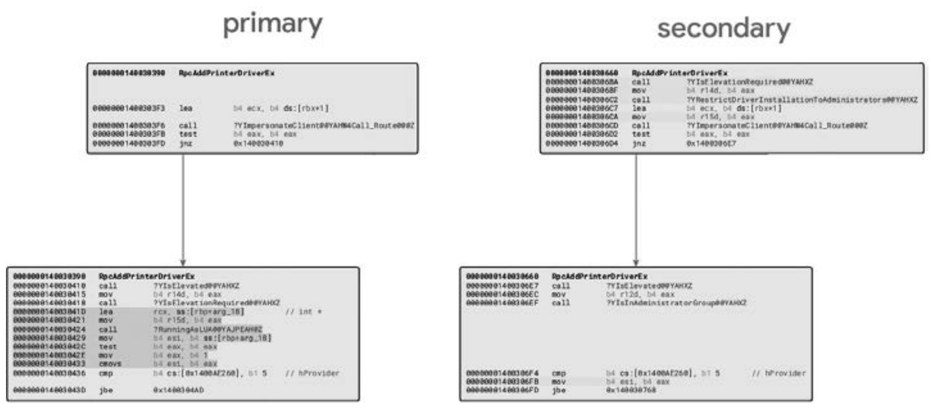
Secondary Unmatche

	Name
2B0	YRestrictDriverInstallationToAdm
534	TFunction4<ushort *,_DRIVER_C(

In the results we can see changes to five functions, the removal of four functions and two imports, and the addition of two new functions in the patched version of spoolsv.exe, as seen in the Secondary Unmatched tab. The function name **YRestrictDriverInstallationToAdministrators** sounds like an obvious function of interest. Let's perform a visual diff of the function **RpcAddPrinterDriverEx**.


0000000140030390 RpcAddPrinterDriverEx
primary
RpcAddPrinterDriverEx 0000000140030660
secondary

We can see a large number of differences between the versions of the function. When zooming into the area towards the top center, we see the following:



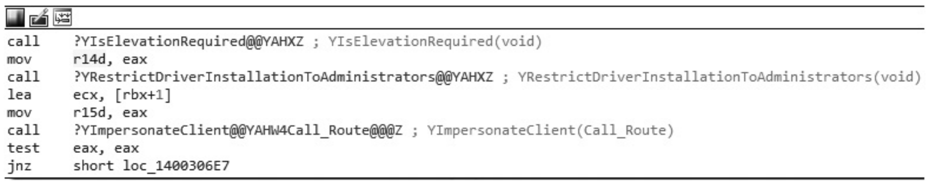
```
0000000140030390   RpcAddPrinterDriverEx

00000001400303F3   lea      b4 ecx, b4 ds:[rbx+1]

00000001400303F6   call     ?YImpersonateClient@@YAHW4Call_Route@@@Z
00000001400303FB   test     b4 eax, b4 eax
00000001400303FD   jnz      0x140030410
```

```
0000000140030390   RpcAddPrinterDriverEx
0000000140030410   call     ?YIsElevated@@YAHXZ
0000000140030415   mov      b4 r14d, b4 eax
0000000140030418   call     ?YIsElevationRequired@@YAHXZ
000000014003041D   lea      rcx, ss:[rbp+arg_18]          // int *
0000000140030421   mov      b4 r15d, b4 eax
0000000140030424   call     ?RunningAsLUA@@YAJPEAH@Z
0000000140030429   mov      b4 esi, b4 ss:[rbp+arg_18]
000000014003042C   test     b4 eax, b4 eax
000000014003042F   mov      b4 eax, b4 1
0000000140030433   cmovs    b4 esi, b4 eax
0000000140030436   cmp      b4 cs:[0x1400AE260], b1 5     // hProvider

000000014003043D   jbe      0x1400304AD
```

```
0000000140030660   RpcAddPrinterDriverEx
000000014003068A   call     ?YIsElevationRequired@@YAHXZ
000000014003068F   mov      b4 r14d, b4 eax
00000001400306C2   call     ?YRestrictDriverInstallationToAdministrators@@YAHXZ
00000001400306C7   lea      b4 ecx, b4 ds:[rbx+1]
00000001400306CA   mov      b4 r15d, b4 eax
00000001400306CD   call     ?YImpersonateClient@@YAHW4Call_Route@@@Z
00000001400306D2   test     b4 eax, b4 eax
00000001400306D4   jnz      0x1400306E7
```

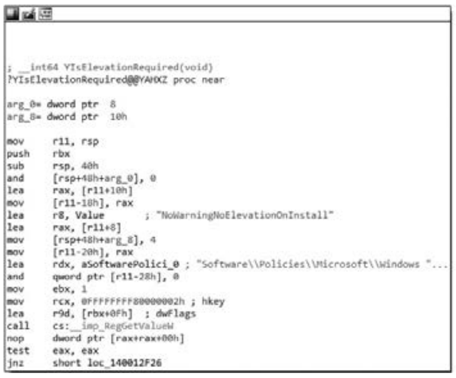
```
0000000140030660   RpcAddPrinterDriverEx
00000001400306E7   call     ?YIsElevated@@YAHXZ
00000001400306EC   mov      b4 r12d, b4 eax
00000001400306EF   call     ?YIsInAdministratorGroup@@YAHXZ




00000001400306F4   cmp      b4 cs:[0x1400AE260], b1 5     // hProvider
00000001400306FB   mov      b4 esi, b4 eax
00000001400306FD   jbe      0x140030768
```

On the primary (unpatched) side is a call to **RunningAsLUA**, which is removed from the secondary (patched) side. There is a new call to the function **YRestrictDriverInstallationToAdministrators** in the patched version. When examining the cross-references to this new function, we see two calls. One call is from **RpcAddPrinterDriver**, and the other is from **RpcAddPrinterDriverEx**. Both of these functions were identified as having changes. The following illustration shows the block of code within **RpcAddPrinterDriverEx** where there is a call to **YIsElevationRequired** and **YImpersonateClient**.

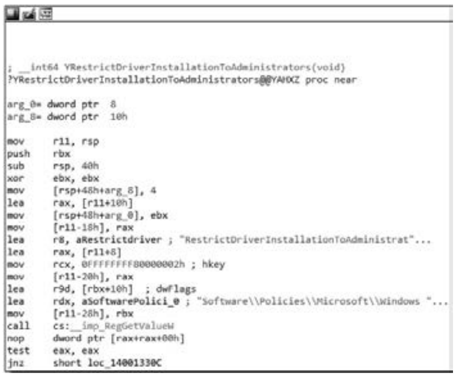


```
call    ?YIsElevationRequired@@YAHXZ ; YIsElevationRequired(void)
mov     r14d, eax
call    ?YRestrictDriverInstallationToAdministrators@@YAHXZ ; YRestrictDriverInstallationToAdministrators(void)
lea     ecx, [rbx+1]
mov     r15d, eax
call    ?YImpersonateClient@@YAHW4Call_Route@@@Z ; YImpersonateClient(Call_Route)
test    eax, eax
jnz     short loc_1400306E7
```

When looking at each of these functions, we see a unique registry key being accessed, as shown here:

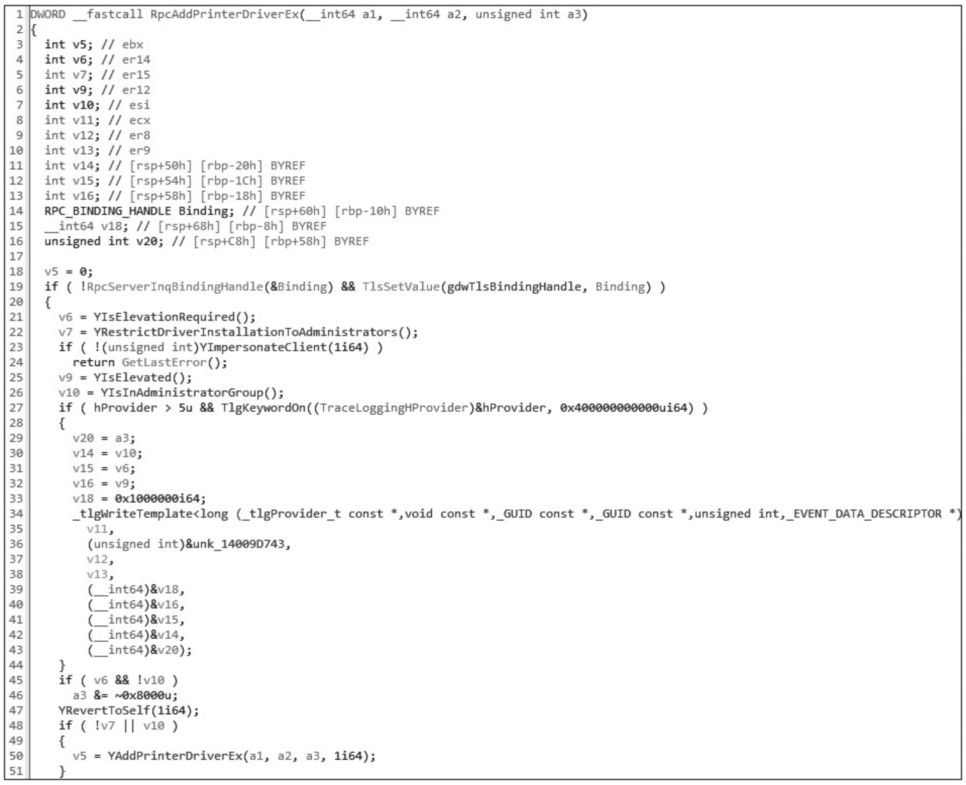


```
; __int64 YIsElevationRequired(void)
?YIsElevationRequired@@YAHXZ proc near

arg_0= dword ptr  8
arg_8= dword ptr  10h

mov     r11, rsp
push    rbx
sub     rsp, 40h
and     [rsp+48h+arg_0], 0
lea     rax, [r11+10h]
mov     [r11-18h], rax
lea     r8, Value       ; "NoWarningNoElevationOnInstall"
lea     rax, [r11+8]
mov     [rsp+48h+arg_8], 4
mov     [r11-20h], rax
lea     rdx, aSoftwarePolici_0 ; "Software\\Policies\\Microsoft\\Windows "...
and     qword ptr [r11-28h], 0
mov     ebx, 1
mov     rcx, 0FFFFFFFF80000002h ; hkey
lea     r9d, [rbx+0Fh]  ; dwFlags
call    cs:__imp_RegGetValueW
nop     dword ptr [rax+rax+00h]
test    eax, eax
jnz     short loc_140012F26
```



```
; __int64 YRestrictDriverInstallationToAdministrators(void)
?YRestrictDriverInstallationToAdministrators@@YAHXZ proc near

arg_0= dword ptr  8
arg_8= dword ptr  10h

mov     r11, rsp
push    rbx
sub     rsp, 40h
xor     ebx, ebx
mov     [rsp+48h+arg_8], 4
lea     rax, [r11+10h]
mov     [rsp+48h+arg_0], ebx
mov     [r11-18h], rax
lea     r8, aRestrictdriver ; "RestrictDriverInstallationToAdministrat"...
lea     rax, [r11+8]
mov     rcx, 0FFFFFFFF80000002h ; hkey
mov     [r11-20h], rax
lea     r9d, [rbx+10h]  ; dwFlags
lea     rdx, aSoftwarePolici_0 ; "Software\\Policies\\Microsoft\\Windows "...
mov     [r11-28h], rbx
call    cs:__imp_RegGetValueW
nop     dword ptr [rax+rax+00h]
test    eax, eax
jnz     short loc_14001330C
```

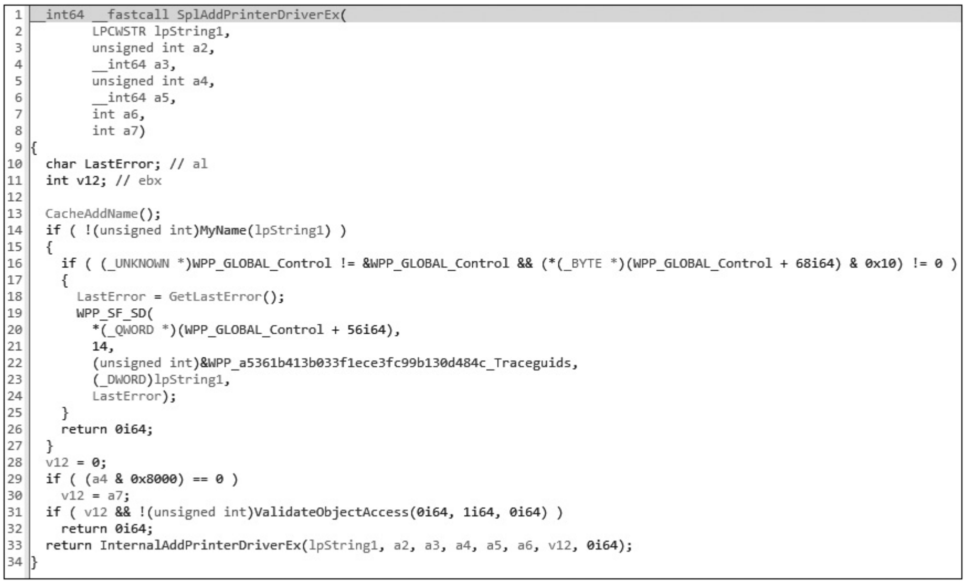
The **YIsElevationRequired** function checks a key called **NoWarning-NoElevationOnInstall**, and **YRestrictDriverInstallationToAdministrators** checks a key called **RestrictDriverInstallationToAdministrators**. The return from **YIsElevationRequired** is recorded in **r14** and the return from **RestrictDriverInstallationToAdministrators** is recorded in **r15**. Let's take a look at the pseudocode of the **RpcAddPrinterDriverEx** function to get a better understanding of the flow. We are using the Hex-Rays decompiler, but you could also use Ghidra or another tool.



```
 1 DWORD __fastcall RpcAddPrinterDriverEx(__int64 a1, __int64 a2, unsigned int a3)
 2 {
 3   int v5; // ebx
 4   int v6; // er14
 5   int v7; // er15
 6   int v9; // er12
 7   int v10; // esi
 8   int v11; // ecx
 9   int v12; // er8
10   int v13; // er9
11   int v14; // [rsp+50h] [rbp-20h] BYREF
12   int v15; // [rsp+54h] [rbp-1Ch] BYREF
13   int v16; // [rsp+58h] [rbp-18h] BYREF
14   RPC_BINDING_HANDLE Binding; // [rsp+60h] [rbp-10h] BYREF
15   __int64 v18; // [rsp+68h] [rbp-8h] BYREF
16   unsigned int v20; // [rsp+C8h] [rbp+58h] BYREF
17 
18   v5 = 0;
19   if ( !RpcServerInqBindingHandle(&Binding) && TlsSetValue(gdwTlsBindingHandle, Binding) )
20   {
21     v6 = YIsElevationRequired();
22     v7 = YRestrictDriverInstallationToAdministrators();
23     if ( !(unsigned int)YImpersonateClient(1i64) )
24       return GetLastError();
25     v9 = YIsElevated();
26     v10 = YIsInAdministratorGroup();
27     if ( hProvider > 5u && TlgKeywordOn((TraceLoggingHProvider)&hProvider, 0x400000000000ui64) )
28     {
29       v20 = a3;
30       v14 = v10;
31       v15 = v6;
32       v16 = v9;
33       v18 = 0x1000000i64;
34       _tlgWriteTemplate<long (_tlgProvider_t const *,void const *,_GUID const *,_GUID const *,unsigned int,_EVENT_DATA_DESCRIPTOR *)
35         v11,
36         (unsigned int)&unk_14009D743,
37         v12,
38         v13,
39         (__int64)&v18,
40         (__int64)&v16,
41         (__int64)&v15,
42         (__int64)&v14,
43         (__int64)&v20);
44     }
45     if ( v6 && !v10 )
46       a3 &= ~0x8000u;
47     YRevertToSelf(1i64);
48     if ( !v7 || v10 )
49     {
50       v5 = YAddPrinterDriverEx(a1, a2, a3, 1i64);
51     }
```

Line 4 shows us that **v6** represents **r14**, which will hold the return from **YIsElevationRequired** on line 21. Line 5 shows us that **v7** represents **r15**, which will hold the return from **YRestrictDriverInstallationToAdministrators** on line 22. Line 26 sets **v10** (esi) if the user is an administrator. The condition in line 45 says that if **v6** is set (elevation required) and **not v10** (not an administrator), then we **and** variable **a3** with **0x8000**, which is **1000000000000000** in binary. This unsets a flag in the 15th bit position of **a3** (edi) to a **0**. The condition in line 48 then says if **v7** is not set (installation not restricted to administrators) **or v10** is set (is an administrator), call the function **YAddPrinterDriverEx**, passing **a3** (user-controllable flags) as one of the arguments.

If you recall, the image from Microsoft for the high-level printer provider components shows an RPC call to the remote spoolsv.exe process. In turn, execution then goes through localspl.dll prior to going into Kernel mode for communication with the actual printer. When looking at the Export Address Table (EAT) of localspl.dll, we can see the function **SplAddPrinterDriverEx**. It has been decompiled, as shown here:



```
 1 __int64 __fastcall SplAddPrinterDriverEx(
 2         LPCWSTR lpString1,
 3         unsigned int a2,
 4         __int64 a3,
 5         unsigned int a4,
 6         __int64 a5,
 7         int a6,
 8         int a7)
 9 {
10   char LastError; // al
11   int v12; // ebx
12 
13   CacheAddName();
14   if ( !(unsigned int)MyName(lpString1) )
15   {
16     if ( (_UNKNOWN *)WPP_GLOBAL_Control != &WPP_GLOBAL_Control && (*(_BYTE *)(WPP_GLOBAL_Control + 68i64) & 0x10) != 0 )
17     {
18       LastError = GetLastError();
19       WPP_SF_SD(
20         *(_QWORD *)(WPP_GLOBAL_Control + 56i64),
21         14,
22         (unsigned int)&WPP_a5361b413b033f1ece3fc99b130d484c_Traceguids,
23         (_DWORD)lpString1,
24         LastError);
25     }
26     return 0i64;
27   }
28   v12 = 0;
29   if ( (a4 & 0x8000) == 0 )
30     v12 = a7;
31   if ( v12 && !(unsigned int)ValidateObjectAccess(0i64, 1i64, 0i64) )
32     return 0i64;
33   return InternalAddPrinterDriverEx(lpString1, a2, a3, a4, a5, a6, v12, 0i64);
34 }
```

Take a look at lines 28–33. Variable **a4** is the same as variable **a3** from the prior pseudocode dump with **RpcAddPrinterDriverEx**, containing flags. We can control this value, which in the unpatched version of spoolsv.exe lacks the checks to the associated registry keys (**NoWarningNoElevationOnInstall** and **RestrictDriverInstallationToAdministrators**). We can effectively bypass the call to **ValidateObjectAccess** and go straight to **InternalAddPrinterDriverEx**. Line 28 sets **v12** to 0. Line 29 says if the 15th bit position in **a4** is not set, then set **v12** to equal that of **a7**, which likely changes the value of **v12** from being a 0. In line 31, if **v12** is set (not zero), then call **ValidateObjectAccess** and check to see if the **sedebugprivilege** right is set. If we can make it so the 15th bit position in **a4** is on, then in line 29 we will not go into the block and instead call **InternalAddPrinterDriverEx**. This effectively allows an attacker to bypass the check and install a driver, allowing for code execution as the user NT AUTHORITY\SYSTEM. There were additional findings and fixes still occurring at the time of this writing; however, this is one of the primary exploitable bugs.

Summary

This chapter introduced binary diffing and the various tools available to help speed up your analysis. We looked at a simple application proof-of-concept example, and then we looked at a real-world patch to locate the code changes, validate our assumptions, and verify the fix. This is an acquired skill that ties in closely with your experience debugging and reading disassembled code. The more you do it, the better you will be at identifying code changes and potential patched vulnerabilities. It is sometimes easier to start with earlier versions or builds of Windows, as well as a 32-bit version instead of 64-bit version, as the disassembly is often easier to read. Many bugs span a large number of versions of Windows. It is not unheard of for Microsoft to also sneak in silent code changes with another patch. This sometimes differs between versions of Windows, where diffing one version of Windows may yield more information than diffing another version.

For Further Reading

BinDiff Manual (Zynamics) www.zynamics.com/bindiff/manual/

"DarunGrim: A Patch Analysis and Binary Diffing Tool www.darungrim.org

PatchExtract gist.github.com/wumb0/306f97dc8376c6f53b9f9865f60b4fb5

delta_patch gist.github.com/wumb0/9542469e3915953f7ae02d63998d2553

"Feedback-Driven Binary Code Diversification" (Bart Coppens, Bjorn De Sutter, and Jonas Maebe) users.elis.ugent.be/~brdsutte/research/publications/2013TACOcoppens.pdf

"Fight against 1-day exploits: Diffing Binaries vs. Anti-Diffing Binaries" (Jeong Wook Oh) www.blackhat.com/presentations/bh-usa-09/OH/BHUSA09-Oh-DiffingBinaries-PAPER.pdf

patchdiff2 (Nicolas Pouvesle) code.google.com/p/patchdiff2/

Back2TheFuture github.com/SafeBreach-Labs/Back2TheFuture

BLUEHEXAGON threat advisory bluehexagon.ai/blog/threat-advisory-cve-2021-1675-aka-printnightmare/

References

1. Zynamics, *BinDiff Manual,* 2017, https://www.zynamics.com/bindiff/manual/.
2. Jaime Ondrusek et al. "Windows Updates Using Forward and Reverse Differentials," *Microsoft 365*, Microsoft, 2020, https://docs.microsoft.com/en-us/windows/deployment/update/psfxwhitepaper.
3. Jaime Geiger. "Extracting and Diffing Windows Patches in 2020." *wumb0in Full Atom*, 2020, https://wumb0.in/extracting-and-diffing-ms-patches-in-2020.html.
4. Microsoft. "Us20070260653a1: Inter-Delta Dependent Containers for Content Delivery." *Google Patents*, Google, 2006, https://patents.google.com/patent/US20070260653.
5. Barry Golden and Amy Viviano. "Local Print Provider." *Microsoft Docs*, Microsoft, 2017, https://docs.microsoft.com/en-us/windows-hardware/drivers/print/local-print-provider.