Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 176 (2020) 88–97

Procedia Computer Science

www.elsevier.com/locate/procedia

24th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Malware Persistence Mechanisms

Zane Gittins[a], Michael Soltys[a*]

[a]*California State University Channel Islands, Dept. of Computer Science, One University Drive, Camarillo, CA 93012, USA*

Abstract

In the public imagination Cybersecurity is very much about malware, even though malware constitutes only part of all the threats faced by Cybersecurity experts. However, malware is still one of the best methods to gain persistent access and control of a target system. Malware is often combined with a well socially-engineered phishing attack that deceives a user to gain a foothold on a system. Once the attakcer gains a beachhead in the victim's network, it may be used to download additional payloads and exploit vulnerabilities, to gain more control and access within a network. Using malware as their foothold, attackers are able to to conduct reconnaissance, gather intelligence (e.g., exfiltration of intellectual property) or simply inflict damage or extortion (e.g., ransomware). All of this has to be done in a way that allows an attacker to retain access for as long as possible; the ability to do so is called *persistence*, and this paper examines the different techniques used by malware to accomplish persistence in an ever evolving landscape.

Keywords: Malware Analysis; Malware Persistence; Incident Response;

1. Introduction

Malware authors continue to seek more advanced methods to maintain persistence on target systems. *Persistence* is the method by which malware survives a reboot of the victim operating system, and is a key element of attacks that require attackers to pivot through a network to accomplish their objective. Traditional methods for persistence are increasingly detected by defenders and anti-virus software. This paper seeks to give an overview of a subset of persistence mechanisms used by malware. We start with traditional persistence mechanisms used by criminal elements, and then analyze more sophisticated persistence mechanisms believed to be utilized by nation state actors. These more advanced persistence mechanisms are harder for defenders to identify, and are less likely to be discovered by anti-virus tools. The terminology of *Advanced Persistent Threat* denotes a state-sponsored group with superior

* Corresponding author. Tel.: +0-000-000-0000 ; fax: +0-000-000-0000.
E-mail address: michael.soltys@csuci.edu, zane.gittins561@myci.csuci.edu

1877-0509
10.1016/j.procs.2020.08.010

capabilities and funding that gains unauthorized access to a system and remains undetected for an extended period of time. These groups continue to attack their targets, even after failures, and often seek to deceive, deny, degrade, destroy, and disrupt their adversaries. The advances in malware development made by sophisticated groups often make their way into the hands of criminals, and then become commodities on the Internet available at little or no cost to less sophisticated actors. In the following sections we discuss malware samples and the persistence techniques they use. At the end of each section we map the persistence technique used to the Mitre ATT&CK framework. *Mitre Att&ck* is an industry standard knowledge base for attack tactics and techniques.

2. Samples of persistent malware

In this section we are going to examine five representative samples of malware: Emotet, OceanLotus Symantec DLL Sideloading, TrickBot, OceanLotus — Explorer-COM Hijack and Agent Tesla. We start each section with a brief summary of the file type that carries the malware, together with its SHA256 signature. We concentrate the analysis on the persistence mechanism.

2.1. Emotet

```
Word Document
SHA256:94926C8520049F7EE51334D699DFC63EB3DB7DDD9C29946161689E1E33BFC0F5
```

Emotet, also known as Geodo, is an information collection malware that is used to install additional malware on victim systems. At the time of this writing, Emotet targets solely Microsoft Windows systems. Emotet was first seen in 2014, and at that time its capabilities largely focused on stealing banking credentials by using web injections. *Web Injections* are a technique where malware intercepts Windows API functions called by the browser. The process of intercepting a function is called *hooking*. A commonly hooked function to perform web injections on Windows is HttpSendRequestA. By intercepting this function, the malware can scan HTTP requests for sensitive data such as credit card numbers and login details, and send these to the operator of the malware.

In the original versions of Emotet, initial access was gained through email campaigns. These emails often contained the malicious executable, or contained a link that tempted victims to download and run a malicious executable [3]. The executable would manipulate victims' browsers to show fake content overlaid on top of webpages. This technique was used to steal credentials to sensitive accounts, such as login/password pairs for online banking websites.

Until 2015, Emotet was for sale on public forums [5], after which its sale became private. Since Emotet became private, its capabilities have shifted to support reconnaissance, and serve as a first stage in an infection chain that may lead to additional malware being installed on victim systems. Emotet has been observed deploying the banking Trojans Dridex and Qakbot, information stealer Trickbot, as well as the Ryuk ransomware. Emotet still uses malicious emails to gain initial access, however these emails now contain Microsoft Office documents with embedded Visual Basic macros. *Visual basic macros* allow users to extend Microsoft Office applications by using the Visual Basic programming language to automate tasks and provide a rich feature set. However, these macros are commonly used by attackers to execute malicious code when an office document is opened by a user. *PowerShell* is a Windows command-line shell built on top of the .NET Framework, which accepts and returns .NET objects. In the case of recent versions of Emotet, these macros execute PowerShell code on the victim system, and this PowerShell code downloads the next stage of the malware. Figure 1 displays the process.

Fig. 1. Steps that lead to an Emotet infection.

PowerShell can be called with the parameter `-EncodedCommand` with Base64 encoded PowerShell code as the value. This is a powerful feature, but it is often abused by malware to hide the content of a payload within a Base64-encoded string.

Emotet invokes PowerShell with a Base64-encoded payload. This PowerShell payload attempts to download the Emotet binary from five unique command and control servers. *Command and Control* servers are systems controlled by attackers which are used to communicate with malware on compromised systems. The use of five command and control servers provides this stage of the attack some resiliency; if any of these servers are successfully contacted, then the other servers under the attacker's control are not needed, and hence not contacted. Due to this, defenders only looking at network traffic may miss some of the command and control servers and fail to have a complete understanding of the attacker's infrastructure.

If PowerShell successfully downloads the next stage from a command and control server then a Windows executable named `377.exe` is saved with the contents returned by the attackers server. The name of this executable differs between samples, however in all cases we have observed executables containing only numeric names. This executable is then started by PowerShell by making a call to the .NET System.Diagnostics.Process class. When run with the name `377.exe`, and administrative permissions, the executable saves an exact copy of itself to the following location:

```
C:\Users\TargetUserName\AppData\Local\monthlymaker
```

Where `TargetUserName` is the name of the infected user. The executable name varies between samples, however the directory is consistent in the samples that we analyzed. `377.exe` then starts the copy of itself called `monthlymaker.exe` and deletes itself from disk. `monthlymaker.exe` then creates a service for persistence. Windows Services allow for the creation of executables that run for an extended duration, and that can be set to run when a computer boots. The service created by `monthlymaker.exe` has the path where `monthlymaker.exe` was saved, and has a description that is an exact copy of the legitimate Microsoft Windows service description for Bitlocker. *Bitlocker* is a encryption feature in Microsoft Windows. The service start type is set to automatic, which will cause `monthlymaker.exe` to start each time the operating system does. In the Mitre Att&ck Framework this is assigned the identifier T1050.

Defenders can detect variants of Emotet that use services for persistence by monitoring for Windows Event ID 7045 on Windows 2008R2 and later systems. *Event logs* are files on Windows systems that store events that occur on a system. Events include security data, error logs, access logs, and many more. Event ID 7045 is generated on a Windows system whenever a new service is installed on a system.

2.2. OceanLotus Symantec DLL Hijacking

```
Vulnerable Symantec RasTLS Application
SHA256:F9EBF6AEB3F0FB0C29BD8F3D652476CD1FE8BD9A0C11CB15C43DE33BBCE0BF68

OceanLotus Dynamic Link Library
SHA256:06DEC0082EAC094DC0B4B3DE8854F190F1D3112DADA0D414D9A085A0EE309199
```

OceanLotus is an advanced persistent threat, also known as SeaLotus, APT-C-00 and APT32, that has been followed closely by security firms such as ESET. OceanLotus targets have included companies, governments, and dissidents. Most known targets of OceanLotus have been located in and around Southeast Asia [1].

In 2018, the security firm ESET released a whitepaper [2] which described how OceanLotus deployed a backdoor which made use of a flaw in the Symantec Network Access Control application to maintain persistence and bypass security products. This backdoor has only been observed effecting targets running Microsoft Windows. The flaw is a common technique, which makes use of the lack of validation when the Symantec Network Access Control application loads a dynamic link library from the Windows side by side folder. This technique is known as DLL-sideloading, and has identifier T1073 in Mitre Att&ck. *Dynamic link libraries*, abbreviated as DLLs, are libraries that contain code that can be used by multiple programs simultaneously. *DLL-Sideloading* is a technique where an attacker causes an unintended library to be loaded when a Windows Side-By-Side manifest is not properly configured. *Windows Side-By-Side manifest* is a file that specifies a Side-By-Side assembly, *Side-By-Side assemblies* are a group of libraries, and classes provided to applications. By placing a malicious DLL with the same name that the Symantec Product expects, and by exporting the same functions as the legitimate DLL, OceanLotus is able to force the Symantec Network

Access Control application to load a malicious dynamic link library. It appears that OceanLotus deletes the legitimate symantec library, and replaces it with their own malicious library. Because this version of the Symantec application does not check the digital signature of the `Rastls.dll` library it loads, attackers are able to force the application to execute their malicious code.

When the Symantec Network Access Control application attempts to call one of the functions imported from the malicious library, the OceanLotus code executes. It does not appear that the malicious library maintains any of the original functionality of the original library, and therefore may cause instability in the Symantec application.

Applocker is a tool created by Microsoft to allow for application whitelisting. If code signed by Symantec is whitelisted in Applocker then this technique could be used to bypass Applocker as the malicious code will be running under the context of the legitimately signed Symantec executable. Furthermore, using side-loading as a method of persistence makes it more difficult for defenders to detect the malware, because it is executing in the context of a trusted security vendor.

To ensure that the Symantec Network Access Control application starts each time the operating system reboots, the malware modifies a registry key in the current user registry hive:

```
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\run
```

The Windows Registry is a hierarchical database that stores settings used by the Windows operating system and Windows applications. The registry key modified by OceanLotus, is used to start the Symantec application each time that a user logs in. This technique has identifier T1060 in Mitre Att&ck. This registry hive is writable without administrative permissions and will launch Symantec Endpoint Protection when the user who was infected logs on to the system. The value stored in this registry key is the path to where the Symantec application, `rastlsc.exe`, is stored on disk by the actor:

```
C:\Users\Username\AppData\Roaming\Symantec Endpoint Protection\
12.1.671.4971.104a\DeviceAssociationService\rastlsc.exe
```

Each time a user logs in, `rasltsc.exe` is started, and once started, it imports the malicious OceanLotus DLL. Then, when `rasltsc.exe` attempts to use one of the imported functions, malicious shellcode stored in a file `SysLog.bin` — located in the same directory — is executed. This technique functions because the malicious `rastls.dll` exports the same five functions that the legitimate library exports, and because the application does not properly validate the imported library.

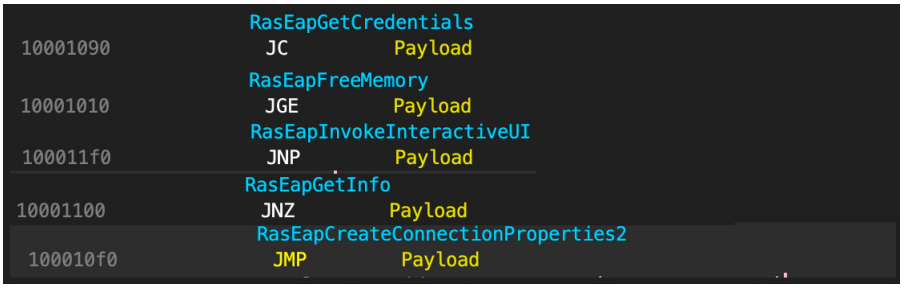

Fig. 2. Library Exports

All exported functions from the malicious DLL lead to the same malicious function. *Shellcode* is position independent code, written in assembly language, which traditionally launches a shell, but may take any action. The shellcode used by OceanLotus is a backdoor which communicates over TCP port 25123 [2]. The shellcode makes use of RtlZeroMemory to clear the MZ file signature of the Symantec application in memory. *File signatures* are the bytes at the beginning of a file, used to identify a file. Windows executables begin with the bytes 4D 5A, which in ASCII is MZ. Some security solutions scan memory for this file signature to identify Windows executables. Clearing the MZ file signature may prevent some security solutions from scanning the application in memory, and thus failing to identify the shellcode. Automatic memory dumping may also fail because of this defense mechanism [2]. *Memory dumping* is the process of writing a region of memory to disk.

To communicate, the malware chooses between one of three subdomains in its configuration and prepends a subdomain that is generated from the computer name of the victim system [2].

```
1008 43.942335    192.168.59.134    8.8.8.8           DNS       120 Standard query 0x52ea A mhggniggmlggnjggidggnnggmbggjgggjhgg.iknlbkgp.traveroyce.com
```

Fig. 3. DNS Request

If this DNS request is successful than the malware attempts to communicate over TCP with a command and control server. The communication is encrypted using RC4, however the key is prepended to the data, and thus the communication is accessible with some basic scrutiny of the packets.

2.3. TrickBot

`HTML Email Attachment`
`SHA256:7AEE90A79191DBB914C77D886C14D5BCAF217CE9C046B85407F69A6F9AB7BB73`

Trickbot is a malware aimed at stealing valuable information from the targets that it infects. Trickbot is a *modular malware*, that is, the initial executable does not contain all of the malware's functionality, instead modules are downloaded from the Trickbot command and control server based on the data sent by the client. This model has several advantages. First, it allows the malware authors to distribute new modules to clients when they see fit. Secondly, if a Trickbot module is detected by an anti-malware solution, only the module may be removed, leaving the primary Trickbot infection behind. Trickbot is well known for its ability to steal passwords saved in browsers, including Google Chrome (note that Google Chrome, when the user is not logged into their Google account, keeps all passwords in a sqlite database encrypted using a key that can be retrieved by making a Windows API call), Firefox, as well as stealing saved information from Microsoft Outlook.

Trickbot samples have been observed using the following method to gain initial access. A malicious email is sent to targets, this email contains an `.html` attachment. If the user downloads and opens the `.html` attachment then they are presented with a web page with a single line of text, that prompts them to download a Microsoft Word document. The Word document is embedded within the html page in Base64. Opening this word document results in embedded visual basic code being executed, the purpose of this code is to construct a JSCRIPT script, which is placed in the same directory as the word document. *JScript* is Microsoft's implementation of JavaScript. The VBA code ends by executing the JScript by calling Microsoft's Wscript tool. *WScript* also known as Windows Script Host, is a tool by Microsoft that allows users to execute scripts in a variety of languages.

The JScript is heavily *obfuscated* — a common technique used by writers of malware —, however using Python code we were able to extract its contents; see the code snippet in Listing 1. Note that the obfuscation, although difficult for a human to parse, can be unwound with simple regular expressions, as show in the Python code.

```
1  import re
2  def matches(regex, text):
3   pattern = re.compile(regex)
4   return pattern.search(text)
5  payload = "payload.js"
6  data = open(payload, "r")
7  payload_data = data.read()
8  data.close()
9  new_dict = {}
10 payload_data = payload_data.split("\n")
11 for line in payload_data:
12  if matches("[0-9]+:\\ \\\"", line):
13   print(line)
14   data = int(re.search(r'\d+', line).group())
15   val = re.findall(r'"([^"]*)"', line)
16   new_dict[data] = val
17  elif matches("[0-9]+\\]\\([0-9]+",line):
18   data = map(int, re.findall(r'\d+', line))
19   data = list(data)
20   key = data[len(data)-1]
```

```
    print(new_dict[key])
    new_dict = {}
```

Listing 1. Python script for de-obfuscating the Trickbot JScript

The Trickbot JScript attempts to download the next stage of the payload by making an http get request to a Trickbot command and control server. If the request is made with the proper parameters then Trickbot is downloaded to the target system. Trickbot copies itself to `%APPDATA%\Roaming\mslibs\` and chooses a random lowercase alpha string for its name.

Windows Scheduled Tasks are a feature that allows for running a executable or script whenever a given trigger is met. Trickbot creates a scheduled task named Ms Libraries with two triggers. The first trigger runs the trickbot executable every time the user logs on. The second trigger will run the executable every 9 minutes for the next 415 minutes on the day that the scheduled task was created. This is likely to make multiple attempts in case initial connection to the Trickbot command and control servers fail. Scheduled task creation has identifier T1053 in the Mitre Att&ck framework.

2.4. OceanLotus — Explorer-COM Hijack

```
COM Hijack Library
SHA256:860F165C2240F2A83EB30C412755E5A025E25961CE4633683F5BC22F6A24DDB6
```

By searching for the IP address 198.50.234.111 from the Ocean Lotus report by ESET in Hybrid Analysis, we were able to discover another sample that is possibly tied to the Ocean Lotus group. *Hybrid Analysis* is a free to use malware sandbox, that runs a given executable inside of a virtual machine, and records events that occur inside of the virtual machine. This malware sample makes use of a technique known as Component Object Model (COM) hijacking. *Component Object Model* is a standard for creating platform independent software components, and is the basis of Microsoft's OLE and ActiveX technologies. COM servers provide access to COM objects through pointers to interfaces. COM servers are implemented in the form of Windows dynamic link libraries. For a dynamic link library to be a valid COM server it must export two functions at a minimum, DllGetClassObject, and DllCanUnloadNow. DllGetClassObject is called when an application needs to access a COM object, the server which exports DllGetClassObject returns a pointer to the interface for the COM object. DLLCanUnloadNow is called when the application is no longer using any COM objects, and unloads the COM server library. COM objects are loaded by using the Windows registry. Specifically, a COM object is searched for by looking at the registry keys present in:

```
System Hive   HKLM:\\SOFTWARE\Classes\CLSID
User Hive     HKCU:\\SOFTWARE\Classes\CLSID
```

The DLL path for the COM server is stored within the registry key InProcServer32, which is a subkey of either the system or user hive. The user registry hive is queried before the system hive. This can pose a serious issue, if an attacker creates or modifies a user hive key then their COM server will be loaded instead of the legitimate COM server. In the Hybrid Analysis report for this malware sample we noticed the registry value for a COM object was modified, the CLSID for this COM object is:

```
CLSID   0E5AAE11-A475-4C5B-AB00-C66DE400274E
```

The Location of the legitimate registry key is HKLM:

```
HKLM:\\SOFTWARE\Classes\CLSID\{0E5AAE11-A475-4C5B-AB00-C66DE400274E}
```

The Location of the malicious registry key is HKCU:

```
HKCU:\\SOFTWARE\Classes\CLSID\{0E5AAE11-A475-4C5B-AB00-C66DE400274E}
```

The legitimate CLSID contains the registry value `%SystemRoot%\system32\Windows.Storage.dll` within the subkey InProcServer32. Which is a legitimate COM server used by Windows Explorer. By creating a similar registry key in HKCU that contains a path to a malicious DLL, the malware is able to perform COM hijacking. This will cause

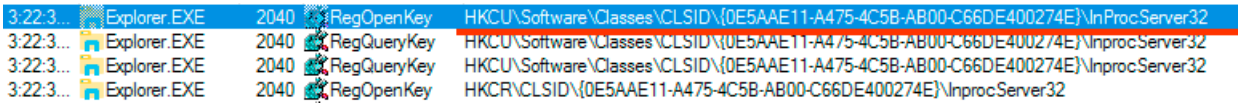

Fig. 4. Process Monitor

the malware to be loaded any time an executable is launched from the task bar. By using the Windows system internals tool, Process Monitor, we were able to verify that Explorer attempts to load the COM server:

We then used the tool Ghidra to verify that the malicious dynamic link library exports DllGetClassObject, which is necessary to perform COM hijacking of Windows Explorer. *Ghidra* is a reverse engineering tool developed by the United States National Security Agency, and released as an open source tool. Component object model hijacking is identified as T1122 in the Mitre Att&ck Framework.

```
HRESULT DllGetClassObject(IID *rclsid,IID *riid,LPVOID *ppv)

{
  FUN_180001000();
  return 0x0;
}
```

Fig. 5. COM Export

2.5. Agent Tesla

`Agent Tesla Dropper`
`878F50F5965B5C795BE1E1D7A12CE6155DC6FDE4ED127E8839EF1E4EE66BD708`

Agent Tesla is a publicly available, for purchase malware, that enables threat actors to steal passwords saved in browsers, collect keystrokes, and take screen captures of victim computers. Agent Tesla has been around since at least 2014, and has been used by groups such as Silver Terrier in attacks against businesses [**?**]. In this section we analyze a sample of Agent Tesla used in a campaign in 2020. Agent Tesla targets solely the Microsoft Windows operating system, and is focused on user workstations.

Agent Tesla employs layers of obfuscation and XOR encryption. The XOR function is a Boolean function which on input (x, y), where x, y are bits, returns 1 if and only if exactly one of x, y is 1. This function is ubiquitous in cryptography as it is reversible and easily used to encrypt a stream of data. To decode the Agent Tesla sample we opened the sample in DnSpy. *DnSpy* is an open source .NET assembler and debugger. Upon inspecting the sample we found code which executed a method named cor41, which was loaded from a resource within the Agent Tesla sample. Using DnSpy we viewed the resource in the DnSpy hexadecimal editor, and noticed the bytes MZ in the ASCII view, these two bytes are the file signature for Windows executables and dynamic link libraries. Noting that this was likely an executable or dynamic link library embedded as a resource, we extracted the resource.

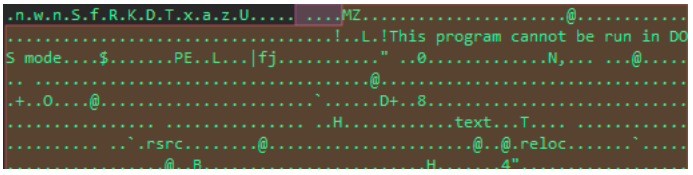

Fig. 6. MZ file signature

Once dumped to disk we discovered that this is a dynamic link library that is used to convert a PNG resource to an executable. This technique, known as *steganography*, involves hiding one file within another. In this case, Agent Tesla authors have hidden the next stage of the malware within a XOR encrypted executable embedded within a PNG resource. The PNG resource looks like the following when viewed in DnSpy:

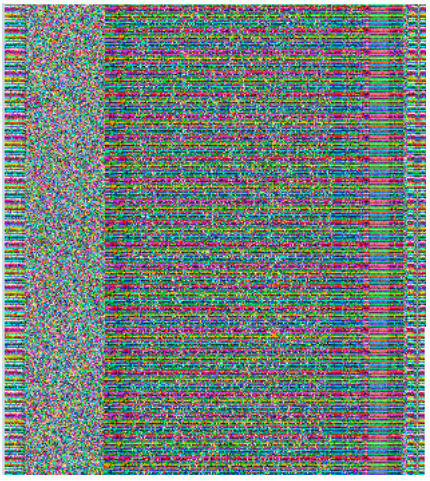

Fig. 7. Steganography

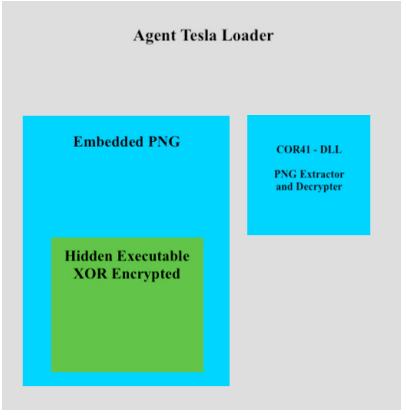

Fig. 8. Agent Tesla Structure

An overview of the structure of the executable can be found in Figure 8.

Using PowerShell and its ability to invoke C# code we were able to use the functions in the malware author's library to decode the image and produce an executable. The PowerShell code to decrypt the PNG resource is provided on Github repository for this paper (https://github.com/zaneGittins/AgentTeslaStegDecoder) but a significant snippet is provided in Listing 2.

```
public static byte[] FromBitmap(Bitmap cor23) {
  ArrayList arrayList = new ArrayList();
  checked {
    int num = cor23.Size.Width - 1;
    for (int i = 0; i <= num; i++) {
      int num2 = cor23.Size.Height - 1;
      for (int j = 0; j <= num2; j++) {
        Color pixel = cor23.GetPixel(i, j);
        Color color = Color.FromArgb(0, 0, 0, 0);
        bool flag = !pixel.Equals(color);
        if (flag) {
          arrayList.InsertRange(arrayList.Count, new byte[] {
            pixel.R,
            pixel.G,
            pixel.B }); } } }
```

```
    return (byte[])arrayList.ToArray(typeof(byte)); } }
```
Listing 2. PowerShell code to decrypt the PNG resource

In the PowerShell code we also perform XOR decryption using a copy of the C# code analyzed via DnSpy; see Listing 3. Once the image is extracted and XOR decrypted, we obtain the final payload.

```
public static byte[] XOR(byte[] cor30) {
    byte[] array = new byte[cor30.Length - 16 - 1 + 1];
    Array.Copy(cor30, 16, array, 0, array.Length);
    int num = array.Length - 1;
    for (int i = 0; i <= num; i++) {
      byte[] array2 = array;
      int num2 = i;
      array2[num2] ^= cor30[i % 16]; }
    return array; }
```
Listing 3. PowerShell code for XOR decryption

Because of the ease in reverse engineering .NET code, authors have utilized control flow flattening to make it difficult to analyze. *Control flow flattening* uses switch statements inside of a for loop to make it difficult to determine the flow of execution. When a block of code in a switch statement is executed, the program returns to the beginning of the for loop, until the final block, which exits the for loop. The end of each switch statement sets the next block of code that will be executed on the next iteration of the for loop.

When first run this payload checks for the mutex, qaxmCJedRGq, if it exists then the malware does not run. *Mutexes* are objects that allow programs to share the same resource, mutexes are commonly used by malware to check if a computer is already infected, additionally mutexes can be used by defenders to identify compromised systems. Agent Tesla uses a mutex to prevent multiple instances of Agent Tesla running on the victim system.

If run with administrative privileges the malware disables Windows Defender by setting several registry keys. Through registry manipulation, Agent Tesla disables the Windows Defender AntiSpyware, OnAccessProtection, BehaviourMonitoring, and TamperProtection keys.

To communicate Agent Tesla uses the SMTP protocol to send an email, the from and to address of the email are the same, and credentials for the email address are embedded within the executable. The sample we analyzed communicated with the target server over port 587. The subject line for an email corresponds to the type of information exfiltrated, the username, and the system name. For example,

```
PW_SYSTEMNAME/USERNAME
```

where PW denotes the exfiltrated data is a password, SYSTEMNAME is the name of the compromised system, and USERNAME is the name of the compromised user. The body of the email contains a timestamp, the username of the infected user, system name of the infected system, and operating system details. Additionally, emails exfiltrating passwords contain newline delimited data on passwords stolen from browsers, where each line is in the following format:

```
Username:StolenUsername Password:StolenPassword Application:BrowserName URL:example.com
```

Where StolenUsername, StolenPassword, BrowserName, and example.com are values assigned by the malware during exfiltration.

Analyzing the payload we found that Agent Tesla uses the following registry key to maintain persistence:

```
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run
```

This technique has identifier T1060 in Mitre Att&ck. To detect Agent Tesla defenders should alert on SMTP communication to untrusted domains, as well as monitor for the creation of startup registry keys.

3. Conclusion

For all the mythology and Hollywood glamour surrounding malware, all malware is limited to a finite number of persistence mechanisms. In the case of Emotet a Windows service is created. In the case of Trickbot scheduled tasks

are leveraged. In OceanLotus Symantec DLL Sideloading, the malware takes advantage of the lack of validation in loading of dynamic link libraries (DLL); in the case of OceanLotus — Explorer-COM Hijack, the malware rides the Component Object Model by manipulating a key in the user hive; in the case of Agent Tesla, the malware payload was embedded in a PNG file, using a modern variation of the art of steganography, and starting the malware by leveraging Windows run registry keys.

We do not want to minimize the challenges of defending systems against malware. Malware can be ingenious and persistent. In this paper we examined a subset of the different techniques malware authors use in order to gain persistence: the ability for the malware to survive a reboot of the system. The techniques described depend on the malware's ability to cloak itself as a legitimate component of the target system. In order to do so, the malware examined makes use of Windows features and takes advantage of validation vulnerabilities, especially in the loading of Dynamically Linked Libraries (DLLs).

Acknowledgments

This work was completed by the first author for a masters thesis in Computer Science at the California State University at Channel Islands, under the supervision of the second author. We are grateful to colleagues at Haas Automation, Meissner Filtration, and California State University Channel Islands for discussions about these topics. We are also grateful to Sam Decanio and Kimo Hildreth for comments on the draft of our paper.

References

[1] Dumont, R., . APT32. URL: https://attack.mitre.org/groups/G0050/.
[2] ESET, 2018. OceanLotus: Old techniques, New backdoors. Technical Report. ESET. URL: https://www.welivesecurity.com/wp-content/uploads/2018/03/ESET_OceanLotus.pdf.
[3] Hungenberg, T., . Emotet, trickbot, ryuk – ein explosiver malware-cocktail. URL: https://www.heise.de/security/artikel/Emotet-Trickbot-Ryuk-ein-explosiver-Malware-Cocktail-4573848.html.
[4] Malpedia, . Mummy spider. URL: https://malpedia.caad.fkie.fraunhofer.de/actor/mummy_spider.
[5] Meyers, A., . Meet crowdstrike's adversary of the month for february: Mummy spider. URL: https://www.crowdstrike.com/blog/meet-crowdstrikes-adversary-of-the-month-for-february-mummy-spider/.
[6] NirSoft, . Nirsoft netpass. URL: https://www.nirsoft.net/utils/network_password_recovery.html.
[7] Zhang, X., . Analysis of the new modules that emotet spreads. URL: https://www.fortinet.com/blog/threat-research/analysis-of-the-new-modules-that-emotet-spreads.html.