The Arsenal Behind the Australian Parliament Hack

blog.yoroi.company/research/the-arsenal-behind-the-australian-parliament-hack/

February 26, 2019

02/26/2019

Introduction

In the past days, an infamous cyber attack targeted an high profile target on the APAC area: the Australian Parliament House. As reported by the Australian prime minister there was no evidence of any information theft and the attack has been promptly isolated and contained by the Australian Cyber Security Centre (ACSC), however the attackers gained access the ruling Liberal and National coalition parties networks as well as the opposition Labor Party, just few months before the federal election. The first technical insight points to sophisticate state sponsored threat actors operating in the Pacific region, but no official statement has been been published and the speculation that China was behind the attack is not confirmed in any way.

Contextually to the cyber incident disclosure to the public, the ACSC declassified some of the samples involved in the parliament hack, so the Cybaze-Yoroi ZLab team decided to investigate these artifacts to have an insight of Tools and Capabilities of part of this APT cyber arsenal.

Technical analysis

All the analyzed files seem to be related to a post-exploitation phase, where the attacker leveraged them to conduct data exfiltration and lateral movements. All the modules don't belong to an open-source post-exploitation framework, like Metasploit or Empire, but they seem to be written from scratch using the high-level language C# on top of the .NET Framework.

The LazyCat DLL

The firstly analyzed sample is known in the InfoSec community. The malware is named LazyCat, mainly derived by the famous Mimikatz pentest tool.

Hash	Sha256: 1c113dce265e4d744245a7c55dadc80199ae972a9e0ecbd0c5ced57067cf755b
Threat	LazyCat
Description	LazyCat DLL to perform local privilege escalation
Ssdeep	1536:kxnT6jqsSwI1ChVKt5QtkJBDbFw+IPpUuOE7qBp69bfeL:kxCpSz1CyIGrbgKuNS69bA

Table 1: Information about LazyCat sample.

A first static analysis shows the library is written in .NET, with no heavy obfuscation, and therefore easily revertable to its source-code like representation.

Figure 1: Static info about LazyCat sample.
Figure 2: Part of malware's code.
An interesting function spotted in the code reveal its capability to inspect and gather the contests of an arbitrary process memory, through the usage of the *MiniDumpWriteDump* function belonging to *DbgHelp* library. The function's result will be stored in a file just created using "Output" string parameter as name (Figure 3).

Figure 3: *DumpMemory* function.
Moreover, the malware is able to start a "TcpRelay" service, probably with the intent of create a route between the attacker's network and the victim's one and then to make the lateral movements easier.

Figure 4: *StartTcpRelay* function.
Exploring source code, a particular module named "RottenPotato" emerges. It contains some interesting functions, such as "findNTLMBytes" and "HandleMessageAuth", related to the post-exploitation phase in MS Windows environments. After a quick search, it is possible to discover it is an open source tool publicly available on GitHub at https://github.com/foxglovesec/RottenPotato.

Figure 5: *findNTLMBytes* function.
Making a diff analysis between the Github source code and the malware's one, emerges that some functions included into malware's *RottenPoteto* are not present into public source code. This indicates that the cyber attacker has further weaponized the code to make it more effective for the malicious goal. At the same time, the usage of code publicly available and open source tools makes more difficult a punctual attribution of the weapons to a particular cyber group.

The LazyCat sample owns a specific module clerks to cover tracks, named *"LogEraser"*.

Figure 6: *LazyCat.LogEraser* module.
The main function of the module is "*RemoveETWLog*" which has the purpose of delete the ETW (Event Tracing for Windows) files related to the malicious actions the attacker has done.

Figure 7: Code to delete Windows log events.
As shown in the above figure, the malware scans all the records belonging to the Windows Log and, if the record ID is equal to the given ID, it will be deleted.

At time of analysis, the sample had a middle-low detection rate, probably due to the customization of open source code-snippets; the result of VirusTotal analysis is visible in the following figure:

Figure 8: LazyCat detection.

The Powerkatz DLL

Hash	Sha256: 08a85f5fe8714b4842180c12c4d192bd186500af01ee39825f6d5100a2019ebc
Threat	Generic
Description	powerkatz DLL
Ssdeep	192:RPmh9ncu5qqTz3XQUOsnoGWX4L4Lzn066HVV1GfzacScaz/69ek4VUAVc:ucuqqTz3gUOsnoGWoL4Lz0661V1PcS5V

Table 2: key information about powerkatz (sample 2)

Hash	Sha256: a95c9fe29a8ae0f618536fdf4874ede5412281e8dfb380bf1370a8d8794f787a
Threat	Generic
Description	powerkatz DLL
Ssdeep	192:BPmh9ncu5qqTz3XQUOsnoGWX4L4Lan066HVV1GfzacScazu69ek4VUf:ecuqqTz3gUOsnoGWoL4L00661V1PcS57

Table 3: key information about powerkatz (sample 3)

Despite the different hashes, the malicious functionalities within the DLL are the substantially the same, the attacker simply modified some strings and variables names, probably to evade av detection. The similarity between the samples is shown in Figure 9, where is possible to see the differences are minimal and they don't impact the overall behavior.

Figure 9: Diff analysis between the samples.
The decompiled source code of the main class also confirms this similarity, i.e. inside the AsyncTask class in Figure 10. For this reason we will reference a single sample in the following paragraphs.

Figure 10: Comparison between *AsyncTask* class of both samples.
The sample is composed by few classes and functions, one of them seems a good starting point for our analysis: the *"StartNew"*. As intended by its name, it is able to start a new asynchronous task on the victim's machine, executing the task object passed as *_app* parameter. Once the task is started, the function waits its completion using repeated 1-sec sleeps cycle, and then it returns a valid code status to the function caller. Probably this module can be used in conjunction with some other functions, belonging to other pieces of the implant, to perform malicious actions in background, making all more stealth.

Figure 11: Source code of the *StartNew* function.
The name of this sample, *Powerkatz,* reminds to a tool available on GitHub (https://github.com/digipenguin/powerkatz) but even if the name is the same, the code is different. As the previous sample, also the detection rate of this sample, 28 of 70, is not high, as shown in Figure 11.

Figure 12: Samples detection rate.

The Recon Module

Hash	Sha256: b63ae455f3deaca297b616dd3356063112cfda6e6c5434c407781461ae69361f
Threat	generic
Description	port scanner DLL
Ssdeep	192:P4NjWnNsFM+5Ic8l5OG/i1/5gK0kbhdeODo3:P4NWnuf5Ic8l21iK0IhDS

Table 4: key information about port scanner sample

Like other samples, it is written C# programming language too. It has two main classes named “PortScanner” and “ReconCommonFuncs”, providing a direct clue of the actions enabled by this part of the implant.

Figure 13: Sample’s classes.
Reading the first one’s code, in fact, the “portScan” contains an Integer array listing few of the well-known network ports, covering major local network services such as HTTP, TELNET, RDP, POP, IMAP, SSH, SQL .. .

Figure 14: Array containing the port numbers to scan.
For each declared port, the function is able to perform a TCP scan, trying to connect to it. If there is an available service behind the port, it responds with its own service banner, which will be stored into a “*StringBuilder*” object. The malware concatenates the responses from all the scanned ports and finally it writes the results in a file using the “*ReconCommonFuncs*” class.

Figure 15: Code to perform port scan.
Figure 16: Usage of *TcpClient* C# class to perform scan.
The “ReconCommonFuncs” class, instead, provides some utility functions, such as “Append” or “GZipAndBase64”, which are self-explanatory.

Figure 17: Functions belonging to *ReconCommonFuncs* class.

The Powershell Agent

Hash	Sha256: 1087a214ebe61ded9f61de81999868f399a1105188467e4e44182c02ee264a19
Threat	generic
Description	OfficeCommu DLL
Ssdeep	3072:JbMNa4pc+32UhnsZFM7iCHF6aZ4oFlSAsBycrxAqSPWy3it5r2py2jYN/IroVbpm:JbWa4xmZcl9fFlSBtuZWQ6qp8DrhFJ

Table 5: key information about the sample

The last sample analyzed by Yoroi ZLab - Cybaze is called “OfficeCommu.dll”, probably with the intent of being confused with the legit Office Communication module available on most Windows machines.

Also this sample is a sort of utility, probably used in the post-exploitation phase, with the purpose of creating a “PowershellAgent”, a stager component of the implant able to parse and execute Powershell commands.

Figure 18: PowershellAgent’s main function.

Conclusion

The analyzed samples show the attackers choose a multi-modular approach for the development of their cyber-arsenal, realizing a complex implant leveraging an ecosystem of libraries providing proper functionalities to conduct advanced, and offensive, cyber operations.

Despite these functions and libraries does not appear to contain any zero-day exploit or techniques, the detection of these modules within a high value perimeter such as the Australian Parliament provides important indication on cyber arsenal development strategies of this threat actor, revealing the abuse and the customization of open-source PenTest tools and proof of concept is one of the preferred way the

attackers used to build their arsenal, possibly due to the lower the “time-to-market” and resources required to write it, without impacting its effectiveness and dangerousness.

Showing also, how these supposedly “known” techniques and tools can be easily repackaged in evasive and silent implants, capable to bypass the traditional kinds of security boundaries.

Indicator of Compromise

Hashes

```
1c113dce265e4d744245a7c55dadc80199ae972a9e0ecbd0c5ced57067cf755b
08a85f5fe8714b4842180c12c4d192bd186500af01ee39825f6d5100a2019ebc
a95c9fe29a8ae0f618536fdf4874ede5412281e8dfb380bf1370a8d8794f787a
b63ae455f3deaca297b616dd3356063112cfda6e6c5434c407781461ae69361f
1087a214ebe61ded9f61de81999868f399a1105188467e4e44182c02ee264a19
```

Yara Rules

```
import "pe"
rule LazyCat_22_02_2019{

        meta:
        description = "Yara Rule for LazyCat"
        author = "Cybaze Zlab_Yoroi"
        last_updated = "2019_02_22"
        tlp = "white"
        category = "informational"

        strings:
                $a = "LazyCat"
                $b = {48 74 74 70 53 65 72 76 65 72 4C 6F}
                $c = {0A 58 73 9E 00 00 0A 2A 0F 00 28 B0}
                $d = {80 A1 4E CD 13 56 80 9F}

        condition:
                pe.number_of_sections == 3 and pe.machine == pe.MACHINE_I386 and (($b and $c and $d) or ($a))
}

import "pe"
rule Powerkatz_22_02_2019{

        meta:
        description = "Yara Rule for Powerkatz"
        author = "Cybaze Zlab_Yoroi"
        last_updated = "2019_02_22"
        tlp = "white"
        category = "informational"

        strings:
                $a1 = {C7 E8 3F}
                $b1 = {7C 43 3D}
                $a2 = {A4 58 24 8A 3A 36 8D 4B 89 15 15 33 CE 1D 1D F2}
                $b2 = {A9 B5 2D 2A 00 47 AC 44 97 7A F5 D0 04 09 75 13}

        condition:
                pe.number_of_sections == 3 and pe.machine == pe.MACHINE_I386 and (($a1 or $b1) and ($a2 or $b2))
}

import "pe"
rule Office_Commu_22_02_2019{

        meta:
        description = "Yara Rule for Office_Commu"
        author = "Cybaze Zlab_Yoroi"
        last_updated = "2019_02_22"
        tlp = "white"
        category = "informational"

        strings:
                $a = {61 E0 4B A1 1D C6 2F A7}
                $b = {8F D2 A9 E3 70 5A B4 D9 92 1D BA}
                $c = "Kill"
                $d = {DB 71 F5 4C B0 29 27 20 B8}
                $e = "get_IsAlive"

        condition:
                pe.number_of_sections == 3 and all of them
}

import "pe"
rule eba_sample_22_02_2019{

        meta:
        description = "Yara Rule for 1eba_sample"
        author = "Cybaze Zlab_Yoroi"
        last_updated = "2019_02_22"
        tlp = "white"
        category = "informational"

        strings:
                $a = {4A 02 73 29 00 00 0A 7D}
                $b = {F8 01 7A 00 1B 00 54 28}
                $c = "portScan"
                $d = {C9 45 99 B9 AA AD C7 46}
                $e = "parseHost"

        condition:
```

```
                pe.number_of_sections == 3 and all of them
}
```

This blog post was authored by Davide Testa, Antonio Farina and Luca Mella of Cybaze-Yoroi Z-LAB