


```
(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, 0x30u);
    region_addr = *region;
    if ( vquery_bytes )
    {
        if ( mm_VirtualProtectRegion(v7, *region, region[1], 0x40u) )
        {
            if ( want_decrypted )
            {
                log_result_msg = log_decrypt_ok;
                mm_PolyDiagLog((__int64)log_result_msg, *region, *((_DWORD *)region + 4), 0);
                buf = (_BYTE *)region;
                seed = *((_DWORD *)region + 4);
                if ( !want_decrypted || *((_BYTE *)region + 21) )
                {
                    seed ^= mm_J19937_Random32(); // On encrypt: this becomes the new session key
                }
                if ( region[1] )
                {
                    i = 0LL;
                    do
                    {
                        buf[i] ^= (unsigned __int8)i ^ (unsigned __int8)(seed >> 1);
                        ++i;
                    } while ( i < size );
                    size = region[1];
                    if ( want_decrypted )
                        goto LABEL_15;
                }
                else
                {
                    size = 0LL;
                    if ( want_decrypted )
                    {
                        LABEL_15:
                        if ( !*((_BYTE *)region + 21) )
                        {
                            mm_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:
                        mm_VirtualProtectRegion(1, *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 = strlenA(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 & 0x200) == 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:
*(DWORD *)temp_buf = 0LL;
QueryPerformanceCounter((LARGE_INTEGER *)temp_buf);
noise_acc += *(DWORD *)temp_buf;
if ( (api_bitmask & 0x2000) == 0 )
goto LABEL_31;
LABEL_29:
*(DWORD *)temp_buf = 0LL;
if ( CryptAcquireContextA((HCRYPTPROV *)temp_buf, 0LL, 0LL, 1u, 0x00000040) )
{
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 )
{
*(DWORD *)temp_buf = 0LL;
GetSystemTimeAsFileTime((LARGE_INTEGER *)temp_buf);
noise_acc += *(DWORD *)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

Hash	Function name
0x2C7B3D30	NtResumeThread
0x8B8E133D	NtClose
0x4C6DC63C	NtWaitForSingleObject
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	Unknown

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, 0`), 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.

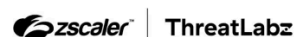
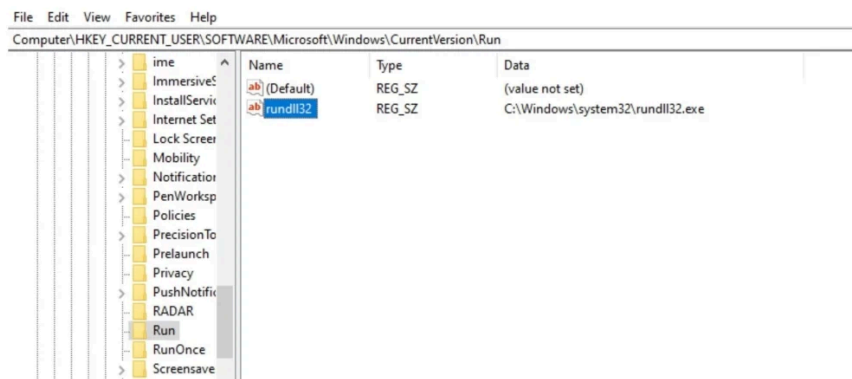


Figure 3: SloppyRAT's failed attempt at establishing persistence via the Run registry key.

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.



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](#).

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 WinHttpQueryOption. SloppyRAT computes the SHA256 hash of the entire DER-encoded certificate, rather than just the public key.

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.

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:

Page 6 of 11

HTTP method	Path	Request body	Response	Description
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 / Mechanism
whoami	—	Retrieves the current user and computer name.	GetUserNameW + GetComputerNameE
hostname	—	Retrieves the computer's DNS hostname.	GetComputerNameExW(ComputerNam
\$env:USERNAME	—	Retrieves the USERNAME environment variable.	GetEnvironmentVariableW("USERNA
\$env:COMPUTERNAME	—	Retrieves the COMPUTERNAME environment variable.	GetEnvironmentVariableW("COMPUT
[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 → dwNumberOf
currentdirectory	—	Retrieves the process's current working directory.	GetCurrentDirectoryW
uptime	—	Retrieve the number of milliseconds since boot.	GetTickCount64

Cmdlet / Expression	Parameters	Description	Windows APIs / Mechanism
[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 + Fin
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 + SHCreateDirectory
Remove-Item / del / erase / ri / rm	[path] [-Recurse]	Deletes a file or directory tree.	DeleteFileW / RemoveDirectoryW (rec
ls env:	—	Retrieves all environment variables.	GetEnvironmentStringsW + FreeEnviro
\$env:LOCALAPPDATA / APPDATA / TEMP / USERPROFILE / WINDIR / SystemRoot / SystemDrive	—	Reads user or system folder paths.	GetEnvironmentVariableW / GetTempP
[Environment]::GetEnvironmentVariable(name)	[name]	Returns an environment variable by name.	GetEnvironmentVariableW
Get-Process / ps / tasklist / gps	[name]	Enumerates running processes.	K32EnumProcesses + OpenProcess + G GetProcessTimes
Get-Service	—	Enumerates Windows services.	Dynamic advapi32: OpenSCManagerW CloseServiceHandle
Start-Process / saps / start / call / &	[exe] [args]	Spawns a new process.	CreateProcessW
Get-ComputerInfo	—	Retrieves system information.	RtlGetVersion (dyn) + GetNativeSystem GetComputerNameExW
Get-Date / date	—	Retrieves the current local date and time.	GetLocalTime + SystemTimeToFileTim
[Environment]::TickCount	—	Retrieves the tick count since boot.	GetTickCount64

Cmdlet / Expression	Parameters	Description	Windows APIs / Mechanism
\$PSVersionTable	—	Retrieves PowerShell version information.	RtlGetVersion checked against hardcoded 7/8/8.1/10/11)
Get-LocalUser	—	Enumerates local user accounts.	Dynamic netapi32: NetUserEnum + Net
Get-LocalGroupMember	[group]	Enumerates members of a local group (e.g., Administrators).	Dynamic netapi32: NetLocalGroupGetM
Get-ItemProperty	[registry path] (HKLM:\... / HKCU:\... / HKCR:\...)	Reads a registry key's values.	RegOpenKeyExW + RegQueryValueEx
Test-NetConnection / tnc	-ComputerName [host] - Port [port]	Performs a TCP connectivity probe.	WSAStartup + GetAddrInfoW + socket 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 + FreeA
Get-MpComputerStatus	—	Queries Microsoft Defender status.	CoCreateInstance(WbemLocator) → RC ExecQuery MSFT_MpComputerStatus
Set-MpPreference / Add-MpPreference	-[Setting] [Value] (e.g. - DisableRealtimeMonitoring \$true)	Modifies Microsoft Defender configuration.	CoCreateInstance(WbemLocator) → Ex
gwmi / Get-WmiObject / gcim / Get-CimInstance	[class]	Queries an arbitrary Windows Management Instrumentation (WMI) class (e.g., Win32_Process).	CoInitializeEx + CoCreateInstance(Wbe ExecQuery → IEnumWbemClassObjec
findstr / dir / search (WMI-translated)	[pattern]	Searches the filesystem by name or pattern using WMI.	same WMI path → SELECT Name,File
(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/Se + CreateFileW/WriteFile
WScript.Shell.CreateShortcut(...)	[lnk path] + target properties	Creates an .lnk shortcut (persistence helper).	CoCreateInstance(CLSID_ShellLink, II IShellLinkW::SetPath/... + IPersistFile::

Cmdlet / Expression	Parameters	Description	Windows APIs / Mechanism
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
\$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`.

Source: <https://www.zscaler.com/blogs/security-research/sloppyrat-new-tool-ransomware-attacks>