Beware the SparroWock: The backdoor that bites, the commands that catch

By Alexandre Côté CyrRomain Dumont

Archived: 2026-09-23 02:01:40 UTC

ESET Research's ongoing monitoring of FamousSparrow has borne fruit once again. Our previous public report on FamousSparrow revealed that this China-aligned APT group had developed two new versions of its custom backdoor named SparrowDoor. This time, we discovered that FamousSparrow has switched to a new backdoor, SparroWocky, and has been deploying it to several countries in Latin America since at least August 2025.

In what was probably China's reaction to the US showing increased interest in Latin America, FamousSparrow increased its targeting of the region to almost exclusively targeting it in July 2025. A month later, we noticed that the group had started using the new SparroWocky backdoor, which then quickly replaced SparrowDoor as FamousSparrow's main implant.

SparroWocky is a modular, C++ backdoor. Its architecture and the techniques used by its authors indicate strong knowledge of anti-analysis tricks and Windows internals. We chose to name the backdoor SparroWocky because the first samples we collected all contain the first stanza of Jabberwocky, a nonsense poem by Lewis Carroll. Fortunately, while advanced, SparroWocky's inner workings are much less arcane than a *gyre and gimble in the wabe*, so a *through and through [of] the vorpal blade* allowed us to bring you a detailed analysis of the backdoor.

> **Key points of the blogpost:**
>
> - FamousSparrow is extensively targeting governmental organizations in Latin America.
> - Since August 2025, the group appears to be abandoning SparrowDoor in favor of SparroWocky, a new custom C++ backdoor.
> - With the switch to SparroWocky, FamousSparrow started to incorporate code from open-source projects directly into its malware.
> - SparroWocky is a full-featured backdoor that manipulates low-level structures in memory, and patches code at runtime in order to avoid detection.
> - SparroWocky has the capability to load and execute Beacon Object Files, a special type of executable file supported by many red-teaming and penetration-testing tools.

FamousSparrow is a China-aligned cyberespionage group believed to have been active since at least 2019. We first publicly documented the group in a blogpost from September 2021 when we observed it exploiting the ProxyLogon vulnerability. The group was initially known for targeting hotels around the world but has also targeted governments, international organizations, trade groups, engineering companies, and law firms. FamousSparrow is the only known user of the SparrowDoor backdoor.

We analyzed two versions of SparrowDoor in a 2025 blogpost, in which we also discussed the attribution claims around the group. As mentioned by Trend Micro, FamousSparrow is linked to Earth Estries; however, the exact

nature of the link is not fully known. FamousSparrow has also been publicly linked to Salt Typhoon, but, due to the absence of any technical indicators, we track them as separate.

Based on our investigation, we attribute the latest campaign and the SparroWocky backdoor to FamousSparrow with high confidence, since in some of the first attacks involving this backdoor, SparroWocky was deployed by the FamousSparrow-exclusive SparrowDoor. Moreover, not only does the victimology match FamousSparrow's previous targeting, we have also recorded attempts to deploy SparroWocky at many of the same organizations that had previously been targeted with SparrowDoor.

Latin America in the crosshairs

As previously mentioned, FamousSparrow currently appears to be focused on high-profile targets in Latin America. This trend started at the latest in July 2025 and has continued with the introduction of SparroWocky. In fact, from mid-2025 and into 2026, 90% of the group's targets registered in our telemetry have been located in the region. As depicted in Figure 1, we've seen the new backdoor deployed against governmental entities in Argentina, Ecuador, Guatemala, Honduras, Panama, Peru, Puerto Rico, and Venezuela. This represents a rare occurrence among the China-aligned APT groups that we currently track, which are generally observed throughout various world regions within such an extended time frame.

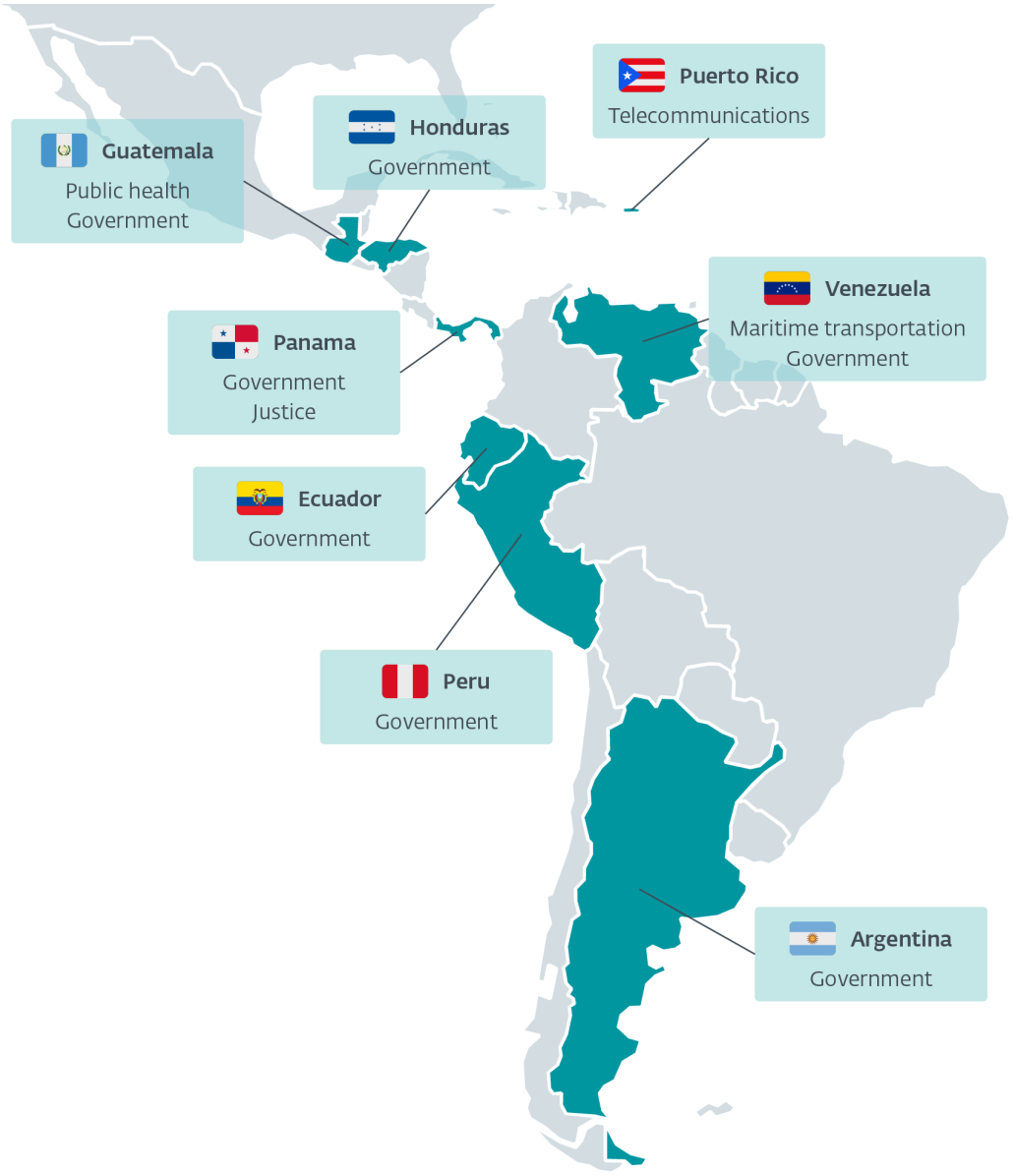

Figure 1. Victimology of SparroWocky

We believe that this focus is not coincidental and likely reflects China's reaction to various recent US initiatives in the region. Indeed, Donald Trump's second presidential term has brought about an aggressive reaffirmation of US interests in Latin America, which threatens various long-term investments that China has cultivated throughout the continent in the last decade, in domains such as energy, mining, and telecommunications. We suspect that FamousSparrow's activities are intended to help China better monitor and anticipate the reaction of local governments to current US pressures.

In some cases, we have observed elements that clearly seem to confirm this hypothesis. For instance, one of the Panamanian entities we've seen being targeted is directly involved in the ongoing commercial dispute regarding two major ports located in the canal area, which were, until recently, operated by a China-based company. As the concession granted to this company was legally challenged by the Panamanian government in early 2025, it seems highly likely that FamousSparrow's operation was intended to gain early, privileged knowledge of local authorities' intentions on this issue.

It is not clear whether the group's apparent focus on Latin America may reflect a formal, geographical mandate, or whether this focus is only temporary and dictated by the current geopolitical circumstances.

Examining SparroWocky

SparroWocky is a full-featured, modular C++ backdoor built with modularity and stealthiness in mind. It appeared shortly after FamousSparrow started focusing on Latin America and quickly became the group's new flagship implant, replacing SparrowDoor. It should be noted that SparroWocky is not a variant of SparrowDoor, but is rather a distinct malware family. The transition to this new backdoor also came with a greater level of integration of open-source tooling into FamousSparrow's workflow: while previously, standalone versions of these tools were deployed side by side with SparrowDoor, with SparroWocky, some have been incorporated directly into the malware.

Some of SparroWocky's notable features include the ability to execute arbitrary files, to act as a TCP proxy, and to execute commands. The backdoor also collects general information about the compromised machine, such as the computer name, the username, domain name, Windows version, and the IP addresses of its network interfaces. SparroWocky is also capable of exfiltrating files and taking screenshots periodically. Exfiltrated information is encrypted using RC4 and sent over the TLS protocol.

Depending on its configuration, SparroWocky can establish persistence either by creating a dedicated service or an entry in a registry Run key.

Loader

SparroWocky is deployed using the common trident loader scheme, which consists of a legitimate executable, a malicious DLL standing in for one required by that executable, and a file containing an encrypted payload (see Figure 2). The loader resides in the aforementioned DLL and is executed via DLL side-loading. We have seen FamousSparrow use a wide range of side-loading targets; in most cases, a patched version of the legitimate DLL that the executable is supposed to load. While most of the file is left untouched, an arbitrary portion of the .text section is replaced with the malicious code, and the entry point header is changed to point inside this patched region.

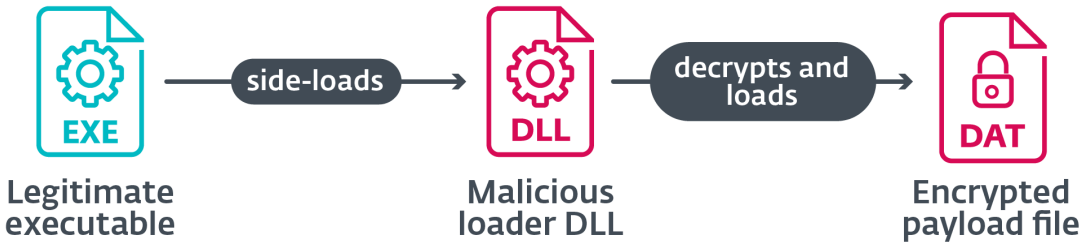

Figure 2. Trident loader scheme

This has some defense-evasion properties: having the metadata and exported function list of the malicious DLL be the same as that of the legitimate version allows it to more easily blend in. Since the code in the patched region does not align with the exported functions and calls in the untouched portion, automated analysis tools may have trouble recognizing function boundaries.

The loader's main role is to extract and decrypt its payload from a file. These files, which typically have the same name as the executable but with a .dat extension, have a specific structure detailed in Figure 3. The file has a custom header that begins with a four-byte magic value of 0x11328712, followed by the size of the configuration data, the size of the payload, and a 16-byte RC4 key. This RC4 key is used to decrypt the remainder of the file, which contains the configuration for SparroWocky (detailed in the *Configuration* section) and the backdoor itself. We provide a script to decrypt SparroWocky payload files in our GitHub repository.

```
struct PayloadFile {
  uint32_t magic_bytes;
  int config_size;
  int backdoor_size;
  char RC4_key[16];
  char config[config_size];
  char backdoor[backdoor_size];
}
```

Figure 3. Definition of the structure of SparroWocky's payload file

The plaintext backdoor payload is formatted as a portable executable (PE) file with the MZ and PE magic values deleted. This executable payload is reflectively mapped directly into memory without being written to disk. Thus, we believe that stripping these magic values is possibly an attempt to evade in-memory defense mechanisms that use simple pattern recognition to identify or dump suspicious sections of memory.

SparroWocky

Our analysis of SparroWocky is mainly based on a sample compiled on November 17th, 2025 according to its PE timestamp (SHA-1: 44F0A22B143B79FA760BF31E14C8FFF714C8A2A1). The version of this backdoor appears to be 1.8, based on the information collected by its fingerprint command explained in Table 3.

As we already mentioned, we chose the name SparroWocky because we found the first stanza of Lewis Caroll's Jabberwocky in several samples we collected. We believe that this stanza comes from the test vectors in RFC 7539, which defines the ChaCha20-Poly1305 encryption algorithm. The samples of SparroWocky also contain other strings that are used as test vectors in that RFC. However, SparroWocky does not use ChaCha20-Poly1305. While we don't know the exact version of Mbed TLS used in the backdoor, the test vectors were present in that library prior to version 4.0.0.

Notably, SparroWocky relies at least on the following public projects:

- Mbed TLS, a C library it uses to establish a secure communication channel with its C&C server,
- MinHook, a Windows API hooking library it uses to hide the start address of newly created threads from security products, and
- COFF Loader (or a similar project) that it uses to enable dynamic loading and execution of in-memory plugins in the form of COFF objects.

Additionally, our analysis revealed that the developers implemented various techniques to evade monitoring tools. This includes a variant of a technique called SilentMoonwalk (or StackMoonwalk), which allows SparroWocky to spoof the call stacks originating from MinHook routines. The backdoor also uses a custom API-hashing algorithm to dynamically resolve Windows API functions. These are explained in greater detail in the *Anti-analysis techniques* section.

Configuration

The SparroWocky loader extracts and decrypts its configuration from the payload .dat file located in the same directory, as explained in the *Loader* section. The RC4 key stored in the payload header is used to decrypt the configuration, which is provided in the form of a tab-separated string that is then parsed and stored in a structure. The fields and their values are described in Table 1 in order of appearance.

Table 1. SparroWocky configuration

Field	Value	Additional details
C&C IP address	216.238.110[.]120	
C&C port number	443	
Connection retry delay (in seconds)	10	After the first retry, the value is randomized.
Proxy connection type	0	0: If enabled, use the proxy configured on the system; otherwise, connect directly. 1: HTTP proxy via Negotiate or Basic authentication. 2: SOCKS5 proxy via Basic authentication or without authentication.
Proxy IP address	N/A	
Proxy port number	N/A	
Proxy username	N/A	
Proxy password	N/A	
Persistence method	1	1: Service persistence. 2: Registry persistence.
Service persistence:	ProcAuditManager	In the configurations we have extracted, the display name is always the same as the

Field	Value	Additional details
service name		service name. These usually match the filename of the payload file.
Service persistence: display name	ProcAuditManager	
Service persistence: service description	Tracks process creation, termination, and related system audit events.	
Registry persistence: registry value	SnapCart	
Registry persistence: registry key	SOFTWARE\Microsoft\Windows\CurrentVersion\Run	Uses HKLM or HKCU depending on privileges.

Capabilities

Argument-controlled behavior

After parsing its configuration, the backdoor checks the command line of the process in which it is running and behaves differently based on the number and value of the arguments passed. If no arguments are present, SparroWocky simply sets up persistence and executes the core logic of the backdoor. Otherwise, the value of the first argument directs the malware to follow specific instructions, as described in Table 2.

Table 2. SparroWocky command line arguments and their meaning

Argument	Behavior	Description
c	Load and execute a PE file in memory for a specified amount of time before termination.	Used in tandem with command 0x16*, SparroWocky reads a command string, an execution timeout delay, and the body of a PE file from standard input (stdin). It then loads the specified executable into memory and executes it with the given command.
p	Sleep for five seconds, set up persistence, and run the core logic of the backdoor.	
s	Run the core logic of the backdoor without	Used in tandem with command 0x2F*, this argument also means the backdoor was run as a specific user (via

Argument	Behavior	Description
	establishing persistence.	CreateProcessAsUser), identified by a session ID that was retrieved by command 0x2E*.
s2	Start a new instance of the backdoor with argument p and terminate.	This argument indicates that the backdoor was started via the service persistence.
t	Set the process working directory to the backdoor location and run the core logic of the backdoor.	

* Explained in the *Backdoor commands* section.

When SparroWocky is executed with the c option, it reads an additional comma-separated list of parameters from standard input (stdin):

- a command string,
- a timeout delay (in seconds), and
- optionally, the body of a PE file.

If this last parameter is absent, the backdoor reads the executable specified in the command string from C:\Windows\System32\ and loads the associated English MUI (Multilingual User Interface) file (from C:\Windows\System32\en-US\). This process is described in the *Host process camouflage for dynamically loaded PEs* section. Otherwise, the PE file is executed by SparroWocky's reflective loader, and the command string is passed as a command line. This functionality is likely meant to allow the backdoor to easily execute system utilities.

SparroWocky loads the specified executable into memory and executes it with the provided command. At the same time, the backdoor creates a new thread that calls ExitProcess to kill the process when the timeout delay expires. The loading process involves setting up hooks and forging structures in memory to camouflage the host process before running the target executable. These anti-analysis tricks are explained in greater detail in the dedicated *Host process camouflage for dynamically loaded PEs* section.

Additionally, when the malware is executed without arguments or with the p option, an instance synchronization mechanism is started. This feature prevents multiple instances of the backdoor from running concurrently by leveraging a custom interprocess communication (IPC) mechanism. When a new instance is launched, the currently running instance stops and, if the new instance is launched from a different location than the current one, the files and persistence configurations set by the currently running instance are deleted. This is achieved by using three types of global objects: a mutex, an event, and a shared memory block named MyMutexName, MyEventName, and MySharedMemName, respectively.

Backdoor commands

The backdoor first establishes communication with its C&C server, then executes its core logic in an infinite loop, within which it processes received commands. These are handled by a custom class named WinHandler (derived from a ServerHandler custom class), according to the runtime type information (RTTI) present in the malware. Handlers for a minimal set of commands are hardcoded in the command loop itself. ServerHandler has a dedicated virtual method to handle more commands. This method is implemented in WinHandler. While we have not observed other implementations of this method, this architecture would make it easy for its developers to change the set of commands that the backdoor can handle. The list of supported commands is shown in Table 3.

Table 3. SparroWocky commands

ID	Arguments	Description
0x10	N/A	Collects and sends the following system information: · MD5 hash of the machine GUID, · SparroWocky PID, · hostname, · IP addresses of all network interfaces, · username, · Windows product name, · backdoor version (1.8), · x64 (likely backdoor architecture), · domain name, · SparroWocky's host file path, · connection retry delay, and · self-deletion enable state (0 or 1).
0x11*	N/A	Starts a new interactive session. Establishes a new connection to the C&C server, sends an initial packet containing the byte sequence 44 33 22 11 (hex), and then starts processing received commands in a separate thread.
0x12	N/A	Terminates by calling ExitProcess.
0x13	N/A	Removes persistence then terminates by calling ExitProcess.
0x14	<function_name> <BOF_object> <function_arguments>	Loads a Beacon Object File in memory and calls <function_name> with <function_arguments> as parameters, then sends the completion status. See below for additional details.
0x16	<command_line> <execution_timeout> <PE_file>	Executes the provided PE file by spawning a new SparroWocky process with the c parameter and standard I/O and error streams redirected to the pipe \\.\pipe\ccpipe. The

ID	Arguments	Description
		arguments are written to the new process's stdin, then the output of the new process is read and sent to the C&C server.
0x17	<command>	Executes <command> by spawning cmd.exe with standard I/O and error streams redirected to two dedicated anonymous pipes.
0x1A*	<IP_address> <port>	Connects to the provided IP address (via TCP/IP) and creates a thread to forward the traffic between the remote machine and the C&C server. The completion status is sent to the C&C server.
0x1B	String of semicolon-separated values starting with two unknown values followed by the IP address and port number on which to listen	Internally named PortmapReverseServer, it accepts TCP connections and forwards traffic to the C&C server. For each accepted connection, a new connection to the C&C server is established and a first packet is sent containing the byte sequence 13 12 11 09 (hex). The listener code then sends the machine GUID followed by the received arguments and the list of connections opened so far. The code proceeds to handle the forwarding of the traffic between the distant machine and the C&C server.
0x1C	Same as 0x1B	Closes the PortmapReverseServer connection specified by the provided IP address and port. The list of remaining open connections is sent to the C&C server.
0x1D	N/A	Returns a list of all PortmapReverseServer connections to the C&C server.
0x1E	Path to the new working directory	Sets the specified current working directory and returns the CWD to the C&C server.
0x1F	N/A	Returns the current working directory to the C&C server.
0x20	Path to the target directory	Creates the specified directory, sending the completion status to the C&C server.
0x21	N/A	Returns the list of logical drives and their type to the C&C server.
0x22	Path to the target directory	Returns a list of the contents of the specified directory, their sizes and last-write times, collected via FindFirstFileW.
0x23	Path of the file to delete	Deletes the specified file and returns the completion status.

ID	Arguments	Description
0x24	Source and destination paths	Copies the specified file to the specified location and returns the completion status.
0x25	Source and destination paths	Moves the specified file to the specified location and returns the completion status.
0x26	Path of the file to rename and the desired new name	Renames the specified file to the specified new name and returns the completion status.
0x27*	File offset and target file path	Sends the file size, creation, last access, and last write timestamps, and the contents of the specified file, read from the specified offset in chunks of 4,096 bytes.
0x28*	Target file path to write to	Sends the current size of the specified file then receives the additional file contents in 4,096-byte chunks, appending them to the target file in a loop.
0x29	N/A	Enumerates display devices and associated settings, returning for each active display device: · device name, · whether it is the main display, · width (pixels), and · height (pixels).
0x2A	Display device name	Takes a screenshot periodically by sending an initial JPG screenshot with its dimensions (width and height) via command ID 0x2C. Every 500 ms, if no new commands are received, a new screenshot is taken, and the difference from the previous screenshot is sent to the C&C server. Changed blocks of pixels in these subsequent screenshots are sent along with coordinates (x, y) and dimensions via command ID 0x2D.
0x2E	N/A	Returns session IDs and usernames of enumerated remote sessions on the system, collected via WTSEnumerateSessionsW.
0x2F	Session ID of the target user session (retrieved via command 0x2E)	Spawns a new instance of SparroWocky (with option s) by duplicating the token associated with the specified session ID and calling CreateProcessAsUserW.
0x30 0x31	N/A	Echoes the command ID back to the C&C server.

ID	Arguments	Description
0x33	<command>	Executes <command> in the current directory by calling CreateProcess with lpCommandLine set to <command> and lpCurrentDirectory set to the CWD. The PID of the newly created process is returned to the C&C server.

* Hardcoded command.

Command 0x14 uses a slightly modified version of RunCOFF from the open-source COFF Loader project to load and execute a Beacon Object File (BOF). A BOF is a position-independent Common Object File Format (COFF) executable that is meant to be run within the memory of an implant. BOFs were first introduced in Cobalt Strike and have since been adopted by other popular red-teaming frameworks such as Brute Ratel, Metasploit, and Sliver. The change to RunCOFF resides in the resolution of imported symbols. SparroWocky redirects calls to external libraries in the BOF to a stack-spoofing subroutine. This effectively hides and proxies calls made by the BOF object. Once the object is loaded, the BOF loader finds and executes function_name, passing the arguments provided in function_arguments. The ability to load BOFs allows FamousSparrow to use existing modules and tools designed to work with this file type.

Self-deletion

As described in the *Argument-controlled behavior* section, SparroWocky can delete itself entirely from the system. This can be done from the C&C server via command 0x13. First, the persistence mechanism previously set is removed and then the batch file shown in Figure 4 is created and executed.

```
 @echo off
timeout /t 2
del "<legitimate_executable>" /f /q
del "<loader_library>" /f /q
del "<payload_filepath>" /f /q
del "%%0" /f /q\n
```

Figure 4. Batch file for self-deletion

This deletes the files used by the backdoor: the legitimate executable, the side-loading library, and the payload file. The batch file deletes itself at the end of the script.

Anti-analysis techniques

SparroWocky employs a few techniques to complicate its analysis and to evade security software that may be in place. A common technique that the backdoor uses is dynamic API resolution via API hashing, but the backdoor also uses more interesting ones, described below.

SilentMoonwalk

The first noteworthy technique is called SilentMoonwalk, which essentially provides a way to forge fake call stacks. Its purpose is to prevent analysis tools and products from inspecting the true caller of specific functions that are frequently monitored, such as Windows API functions. This method requires a few initialization steps:

- Finding the offset of RtlUserThreadStart and BaseThreadInitThunk, two functions that are usually found at the start (or bottom) of any call stack.
- Finding a JOP (jump-oriented programming) and a ROP (return-oriented programming) gadget in the legitimate kernel32.dll library to restore the original call stack.

Once these requirements are met, when SparroWocky makes an obfuscated call to a Windows API function, it first saves the current context (registers); next, it forges a fake stack using the gadgets found previously, and then inserts the address of a stack and context restoration routine. This makes it appear as if the calls to Windows API functions are originating from RtlUserThreadStart and BaseThreadInitThunk. Figure 5 shows the call stack view from a debugging session using WinDbg.

Stack

Frame Index	Call Site	Child-SP	Return Address
[0x0]	**ntdll!NtDelayExecution+0x14**	**0x7b3cb5f118**	**0x7ffd3a1f62ce**
[0x1]	KERNELBASE!SleepEx+0x9e	0x7b3cb5f120	0x7ffd3a682204
[0x2]	KERNEL32!BaseFlushAppcompatCacheWorker+0x44	0x7b3cb5f1c0	0x7ffd3a650be9
[0x3]	KERNEL32!BaseCheckVDMp+0xa95	0x7b3cb5f370	0x7ffd3a627374
[0x4]	KERNEL32!BaseThreadInitThunk+0x14	0x7b3cb5f720	0x7ffd3c63cc91
[0x5]	ntdll!RtlUserThreadStart+0x21	0x7b3cb5f750	0x0

Figure 5. WinDbg call stack view of an obfuscated call to Sleep

In the case of SparroWocky, this technique is used to obfuscate calls made by BOF-formatted plugins (command 0x14) or by the statically linked MinHook hooking library.

Concealing the thread start address

SparroWocky uses the MinHook library to hook the CreateThread function in order to conceal the original lpStartAddress parameter from security products. Essentially, any thread created by SparroWocky would have AnimateWindow as the starting address, which would likely be considered legitimate by a security product. The patch applied to AnimateWindow turns it into a trampoline that simply executes the original start address, as illustrated in Figure 6.

```
0x0:  48 8b 05 0d 00 00 00          mov rax,QWORD PTR [rip+0xd]  # 0x14 => ORIGINAL_API_ADDRESS
0x7:  49 bb 11 22 33 44 55 66 77 88 movabs r11, HOOK_HANDLER_ADDRESS
0x11: 41 ff e3                      jmp    r11
0x14: ORIGINAL_API_ADDRESS
```

Figure 6. AnimateWindow *API is patched to execute the original start address*

Host process camouflage for dynamically loaded PEs

The last notable piece of code from SparroWocky is its custom PE loader, used when executed with option c. While implementing PE loaders is pretty much routine for malware authors, SparroWocky authors took it a step further and integrated host process camouflage.

As described in Table 2, when SparroWocky is executed with the c option, it loads a PE file in memory and executes it. If the file is not passed as an argument, the PE loader parses the specified command line to extract the file's name. It searches for that filename in the C:\Windows\System32\ directory, but most importantly it retrieves the English localization MUI file associated with the target PE (stored as C:\Windows\System32\en-US\ <exe_name>.mui). In that case, the PE file is loaded in memory, and a few hooks are set to make sure any calls made by the loaded PE file to retrieve resource data, such as RtlLoadString or RtlFindMessage, are redirected to the .mui data. This process mirrors normal behavior of Windows when loading PEs, and reduces the risk of unexpected errors.

The command line retrieved from Stdin is parsed and SparroWocky hooks the following functions, which are used to retrieve information about command line arguments, to make them point to this command line:

- GetCommandline[AW]
- __(w}getmainargs
- __p___argc
- __p___{w}argv

The PE loader is also able to register the exception handlers of the newly loaded executable – an unusual, yet critical, addition – since it allows exceptions to be handled correctly.

Finally, before calling the entry point of the loaded PE file, SparroWocky forges and inserts a fake LDR_DATA_TABLE_ENTRY structure in the doubly linked list of the PEB_LDR_DATA structure. This doubly linked list is used by Windows to keep track of loaded modules and is usually monitored by security products. Figure 7 shows a snippet of the code used to set some of its fields.

```
new_ldr_entry->DllBase = PE_base;
new_ldr_entry->SizeOfImage = nt_header->OptionalHeader.SizeOfImage;
new_ldr_entry->TimeDateStamp = nt_header->FileHeader.TimeDateStamp;
new_ldr_entry->BaseDllName = *dll_name;
new_ldr_entry->FullDllName = *dll_path;
new_ldr_entry->EntryPoint = PE_base + nt_header->OptionalHeader.AddressOfEntryPoint;
result = 1;
new_ldr_entry->ObsoleteLoadCount = 1;
if ( !v10 )
{
  new_ldr_entry->Flags = 0xC004;
  if ( v33 )
    new_ldr_entry->Flags = 0x40C004;
}
new_ldr_entry->HashLinks.Blink = &new_ldr_entry->HashLinks;
new_ldr_entry->HashLinks.Flink = &new_ldr_entry->HashLinks;
```

Figure 7. SparroWocky forges an LDR_DATA_TABLE_ENTRY *structure*

This last technique shows that SparroWocky authors possess a deep understanding of the Windows PE loading mechanism and are willing to go the extra mile to camouflage the host process and confuse monitoring software.

Network protocol

To communicate with its C&C server, SparroWocky uses the TLS encryption protocol. Under the hood, the backdoor uses the Mbed TLS library and the only element worth mentioning is that it uses the personalization

string acdbenus when initializing the deterministic random bit generator, as seen in Figure 8.

```
24  strcpy(custom_entropy_seed, "acdbenus");
25  a1->fd = operator new(4u);
26  a1->ssl_config = operator new(0x180u);
27  a1->p_entropy_ctx = operator new(0x410u);
28  a1->p_ctr_drbg_ctx = operator new(0x160u);
29  a1->ssl_ctx = operator new(0x1D8u);
30  *a1->fd = -1;
31  memset(a1->ssl_ctx, 0, sizeof(mbedtls_ssl_context));
32  memset(a1->ssl_config, 0, sizeof(mbedtls_ssl_config));
33  p_ctr_drbg_ctx = a1->p_ctr_drbg_ctx;
34  memset(p_ctr_drbg_ctx, 0, sizeof(mbedtls_ctr_drbg_context));
35  p_ctr_drbg_ctx->reseed_counter = -1;
36  p_ctr_drbg_ctx->reseed_interval = 10000;
37  f_init_rng_provider_desc(a1->p_entropy_ctx);
38  do
39    ++custom_entropy_seed_len;
40  while ( custom_entropy_seed[custom_entropy_seed_len] );
41  mbedtls_ctr_drbg_seed_entropy_len(
42    a1->p_ctr_drbg_ctx,
43    v4,
44    a1->p_entropy_ctx,
45    custom_entropy_seed,
46    custom_entropy_seed_len);
47 }
```

Figure 8. Custom initialization of Mbed TLS random bit generator

Before the initial TLS handshake, a TCP connection is established with the C&C server using one of three connection modes:

A connection mode of 0 means that SparroWocky uses the proxy currently configured on the machine or a direct TCP connection if no system proxy is configured. This configuration is retrieved by querying the ProxyServer registry value located under the registry key HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings. If the connection to the proxy server is not successful, SparroWocky tries to connect via mode 1, then mode 2.

Connection mode 1 represents a connection via an HTTP proxy. This connection uses either the Negotiate (Kerberos or NTLM) or Basic authentication scheme with the username and password provided in the configuration. Both authentication methods use generic HTTP headers with the User-Agent string set to Mozilla/5.0.

Connection mode 2 uses a SOCKS5 proxy without authentication (AUTH field set to 0x00) or with a username and password (AUTH field set to 0x02). The values used by the latter are provided in the configuration.

Command messages

Once the TLS handshake is complete, SparroWocky sends the bytes 0x11223344 (big-endian) to indicate that it is ready to receive commands in the main session. The backdoor uses a simple format to receive commands and send results, as illustrated in Figure 9.

```
Type definition
Offset | Size | struct __unaligned __declspec(align(4)) command_t
       |      | {
  0000 | 0008 |   _QWORD packet_id;
  0008 | 0004 |   _DWORD command_id;
  000C | 0004 |   unsigned int command_arg_size;
  0010 | 0004 |   int cmd_error;
  0014 | 0008 |   unsigned __int8 RC4_key[8];
       | 001C | };
```

Figure 9. Command message format

If the command_arg_size field does not equal 0, then additional data is to be received or sent after the header. In that case, the data (command arguments or results) is encrypted via RC4, and each command message uses a newly generated eight-byte key, which is sent in the header.

Network infrastructure

SparroWocky uses the IP address of its C&C servers, which is generally running on port 443, to connect directly. We have also seen it running on port 8080 in some cases. While we have observed some self-signed certificates being reused across multiple servers, we do not have a reliable generic fingerprint.

Conclusion

Over the latter half of 2025 and the first half of 2026, FamousSparrow had been focusing on targets in Latin America. This represents a shift from its previous global targeting. To go along with this change, the group has developed SparroWocky, which replaced SparrowDoor as its main implant. While it doesn't appear to be based on the same codebase, we can see that SparroWocky still shares some of the functionality and concepts that were present in the group's previous backdoor, which we analyzed in our previous blogpost. SparroWocky uses more complex defense evasion techniques to stay under the radar.

FamousSparrow still uses open-source offensive tooling for its own malicious ends. Previously, these tools were mainly used side by side with the group's backdoor. With SparroWocky, we can observe that it also has the development capabilities to integrate open-source code directly into its own custom backdoor.

> *For any inquiries about our research published on WeLiveSecurity, please contact us at threatintel@eset.com.*
>
> *ESET Research offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence page.*

IoCs

A comprehensive list of indicators of compromise (IoCs) and samples can be found in our GitHub repository.

Files

SHA-1	Filename	Detection	Description
3209689E509205CCDB7E 49062B7B407DDC23CAC1	winfsp-x64.dll	Win64/Agent.HUP	SparroWocky loader.
52C6646759CF6037BB17 466203631C4BD794532F	winfsp-x64.dll	Win64/Agent.HUP	SparroWocky loader.
99E7070B5AF24A0FE1E6 FEBE5954B03CB385E91F	DukeQt.dll	Win64/Agent.ISF	SparroWocky loader.
44F0A22B143B79FA760B F31E14C8FFF714C8A2A1	N/A (in-memory)	Win64/Agent.ASW	SparroWocky backdoor.
9AA9FF61BC63CCAB907 4FE837F39C980CA9DDC8C	N/A (in-memory)	Win64/Agent.ASW	SparroWocky backdoor.

Network

IP	Domain	Hosting provider	First seen	Details
38.54.57[.]17	N/A	LightNode-BR	2026-02-25	SparroWocky C&C server.
38.60.197[.]55	N/A	Kaopu Cloud HK Limited	2026-03-16	SparroWocky C&C server.
38.60.209[.]106	N/A	Kaopu Cloud HK Limited	2026-02-26	SparroWocky C&C server.
38.60.224[.]51	N/A	Kaopu Cloud HK Limited	2026-02-25	SparroWocky C&C server.
38.60.224[.]235	N/A	Kaopu Cloud HK Limited	2026-02-24	SparroWocky C&C server.
38.60.241[.]65	N/A	Cogent Communications	2026-03-10	SparroWocky C&C server.
38.60.241[.]127	N/A	Cogent Communications	2026-03-04	SparroWocky C&C server.
38.60.241[.]193	N/A	KaopuCloud-BR	2026-01-22	SparroWocky C&C server.
77.111.101[.]40	N/A	Latitude.sh	2026-05-20	SparroWocky C&C server.
91.148.134[.]115	N/A	Charles-R Paquet	2026-06-17	SparroWocky C&C server.
130.94.101[.]82	N/A	NTT America, Inc.	2026-02-26	SparroWocky C&C server.
140.99.164[.]199	N/A	Private Customer	2026-02-26	SparroWocky C&C server.
149.104.87[.]228	N/A	Lightnode-MX	2026-02-24	SparroWocky C&C server.
149.104.90[.]203	N/A	BEDGE CO LIMITED	2026-01-22	SparroWocky C&C server.
216.238.92[.]2	N/A	The Constant Company, LLC	2026-02-25	SparroWocky C&C server.

IP	Domain	Hosting provider	First seen	Details
216.238.105[.]53	N/A	The Constant Company, LLC	2026-01-22	SparroWocky C&C server.
216.238.110[.]120	N/A	The Constant Company, LLC	2025-12-11	SparroWocky C&C server.
216.238.121[.]164	N/A	The Constant Company, LLC	2026-03-16	SparroWocky C&C server.

MITRE ATT&CK techniques

This table was built using *version 19* of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Resource Development	T1583.003	Acquire Infrastructure: Virtual Private Server	FamousSparrow has acquired servers to use for C&C and delivery servers for SparroWocky.
	T1587.001	Develop Capabilities: Malware	FamousSparrow has developed SparroWocky and its loader.
	T1608.001	Stage Capabilities: Upload Malware	FamousSparrow has uploaded the SparroWocky trident loader to attacker-controlled delivery servers.
Initial Access	T1190	Exploit Public-Facing Application	FamousSparrow gained access to targets' networks by exploiting publicly reachable Exchange servers.
Execution	T1059.003	Command and Scripting Interpreter: Windows Command Shell	SparroWocky has functionality to run commands via the Windows command shell.
	T1569.002	System Services: Service Execution	When establishing persistence via a service, SparroWocky starts the service directly.
	T1106	Native API	SparroWocky uses the native Windows API.
	T1559	Inter-Process Communication	SparroWocky uses an interprocess communication mechanism to synchronize instances when a new one is launched.
	T1574.001	Hijack Execution Flow: DLL	The SparroWocky loader is executed via DLL side-loading.

Tactic	ID	Name	Description
Persistence	T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	SparroWocky can persist via a registry Run key.
	T1543.003	Create or Modify System Process: Windows Service	SparroWocky can persist via a Windows service.
Stealth	T1134.002	Access Token Manipulation: Create Process with Token	SparroWocky can create processes using a token obtained from any existing user session.
	T1140	Deobfuscate/Decode Files or Information	SparroWocky's loader retrieves the configuration and payload via RC4 decryption of the content of a file with a custom format.
	T1480.002	Execution Guardrails: Mutual Exclusion	SparroWocky uses a mutex to prevent multiple instances from running concurrently.
	T1564.010	Hide Artifacts: Process Argument Spoofing	When loading an external PE file, SparroWocky hooks functions to retrieve its command line arguments from stdin.
	T1027.007	Obfuscated Files or Information: Dynamic API Resolution	SparroWocky uses a custom API hashing algorithm to dynamically resolve API functions at runtime.
	T1620	Reflective Code Loading	The SparroWocky reflectively loads its payload into memory. SparroWocky can reflectively load and execute PE and BOF objects.
	T1070.004	Indicator Removal: File Deletion	SparroWocky can delete itself from the compromised machine.
	T1070.009	Indicator Removal: Clear Persistence	SparroWocky can remove its persistence mechanism from the compromised machine.
	T1036.001	Masquerading: Invalid Code Signature	The SparroWocky loader keeps the now invalid signature of the legitimate module it is impersonating.

Tactic	ID	Name	Description
	T1036.004	Masquerading: Masquerade Task or Service	SparroWocky uses legitimate or generic names and descriptions for its persistence service.
Discovery	T1083	File and Directory Discovery	SparroWocky can list files and directories on mapped drives.
	T1680	Local Storage Discovery	SparroWocky can retrieve information about mapped storage devices.
	T1082	System Information Discovery	SparroWocky can collect information about the system it is running on, such as the Windows version, hostname, and the IP addresses of network interfaces.
	T1033	System Owner/User Discovery	SparroWocky can retrieve the username of the current user and of any user with an active session.
	T1120	Peripheral Device Discovery	SparroWocky can retrieve information about connected display devices.
Collection	T1005	Data from Local System	SparroWocky can exfiltrate files from mapped storage.
	T1113	Screen Capture	SparroWocky can periodically capture screenshots.
Command and Control	T1573.002	Encrypted Channel: Asymmetric Cryptography	SparroWocky uses TLS, which uses asymmetric cryptography in its handshake.
	T1573.001	Encrypted Channel: Symmetric Cryptography	SparroWocky uses RC4 to encrypt the information it exfiltrates.
	T1090.001	Proxy: Internal Proxy	SparroWocky can proxy connections between the C&C server and another remote machine.
	T1090.002	Proxy: External Proxy	SparroWocky can use an HTTP or SOCKS5 proxy to connect to its C&C server.
	T1095	Non-Application Layer Protocol	SparroWocky uses TLS over TCP to communicate with its C&C server.

Tactic	ID	Name	Description
Exfiltration	T1041	Exfiltration Over C2 Channel	SparroWocky exfiltrates data through the same connection used to receive commands from the C&C server.

Source: https://www.welivesecurity.com/en/eset-research/beware-sparrowock-backdoor-bites-commands-catch/