Chinese actor takes aim, armed with Nim Language and Bizarro AES

6/22/2022

June 22, 2022

Executive Summary

In this article, Check Point Research shares findings on a group / activity cluster with ties to **Tropic Trooper**:

- The infection chain includes a previously undescribed loader (dubbed “Nimbda”) written in Nim language.
- This loader was observed bundled with a Chinese language greyware “SMS Bomber” tool that is most likely illegally distributed in the Chinese-speaking web.
- A new variant of the ‘Yahoyah’ Trojan focused on collecting information about local wireless networks.
- Carefully modified AES cipher shows cryptographic know-how on part of attackers.
- Insights on the campaign infrastructure.

Introduction

Check Point Research has recently been tracking a cluster of malicious activity that has been going on for the past several years. The observed activity has strong connections to the Tropic Trooper cluster of activity, as documented by Trend Micro, based on shared infrastructure, tools, and coding practices. Tropic Trooper was previously observed targeting Philippines, Hong Kong and Taiwan; while the two latest are Chinese-speaking countries.
This activity in particular caught our interest due to its unusual technical quirks and targeting, as well as the use of a new net-capable strain of a previously documented piece of malware. In this article, we share our findings, with an

emphasis on the technical highlights (this is a polite way of saying there won't be an IDA screenshot of a single-byte-XOR decryption).

In Socialist China, Tool Hacks You

Malware authors, a superstitious and risk-averse lot, generally keep their distance from the primordial soup of programming languages trying to 'make it' in industry. We don't encounter many Haskell Banking Trojans or Prolog Ransomware binaries, and the same goes for Nim backdoors. Still, even that rule has its strange exceptions, such as this downloader used by Zebrocy or Nimar loader used by the Trickbot Group. We can now add to this list a new loader, which we dub Nimbda.

`Nimbda` contains an embedded executable, an SMS Bomber, which it drops into the victim's temp folder and then executes. Following that, Nimbda *separately* injects a *different* piece of code into a launched `notepad.exe` process.

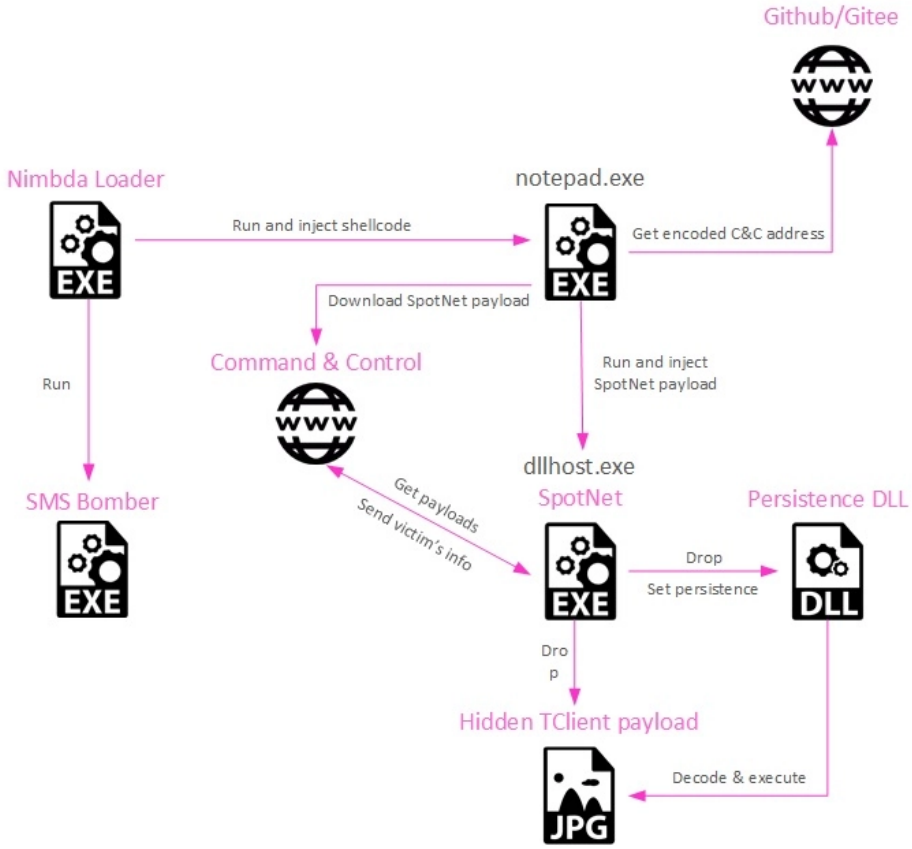

Figure 1: The infection chain

Greyware as a lure

The **dropped executable** is a Chinese GUI tool called "SMS Bomber". It should take a phone number as input and then slaps it on to a very long list of pre-baked HTTP requests asking for one-time codes, verification messages, password recoveries and the like. The result is that the victim's phone becomes flooded with messages and presumably unusable. SMS Bomber warns users not to use their own phone number for testing, is registered to "A Pot of Sake All rights reserved", and is written in EPL, a language specifically popular in China for its native support of Chinese strings.

This tool is, for lack of a better word, shady. Many security solutions will outright categorize it as malware, even if they can't agree on a specific reason why. EPL is so disproportionately popular among malware developers compared to the general population that it sometimes gets slapped with the 'malware' label on sight, regardless of what the EPL script actually does. The EPL interpreter DLL itself is stored in memory, UPX-packed and loaded in real time, which also isn't a good look. For what it's worth, the tool does what it says it does *and* the author felt

comfortable enough to leave their QQ (Chinese instant messenger) number under a ‘contact the author’ menu item, so maybe we shouldn’t be so quick to judge by a first impression.

Figure 2: SMS Bomber main interface

```
aHttpMsqYnlibCn db 'http://msq.ynlib.cn/portal/user/telVerification/',0
                                        ; DATA XREF: sub_406AB9+37FCE↑o
aStatusSendmess db 'status=sendmess&phone=',0
                                        ; DATA XREF: sub_406AB9+38397↑o
aHostQrBatchCom db 'Host: qr-batch.com',0Dh,0Ah
                                        ; DATA XREF: sub_406AB9+38432↑o
                db 'Connection: keep-alive',0Dh,0Ah
                db 'Content-Length: 33',0Dh,0Ah
                db 'Accept: */*',0Dh,0Ah
                db 'Origin: https://qr-batch.com',0Dh,0Ah
                db 'X-Requested-With: XMLHttpRequest',0Dh,0Ah
                db 'User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 11_0 like Mac OS X'
                db ') AppleWebKit/604.1.38 (KHTML, like Gecko) Version/11.0 Mobile/15'
                db 'A372 Safari/604.1',0Dh,0Ah
                db 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8',0Dh
                db 0Ah
                db 'Referer: https://qr-batch.com/register.php',0Dh,0Ah
                db 'Accept-Language: zh-CN,zh;q=0.9',0
aHttpsQrBatchCo db 'https://qr-batch.com/data/sendSms.php',0
                                        ; DATA XREF: sub_406AB9+3846E↑o
```

Figure 3: A pre-made HTTP request asking for a verification code to be sent to a given phone number, part of the SMS Bomber tool.

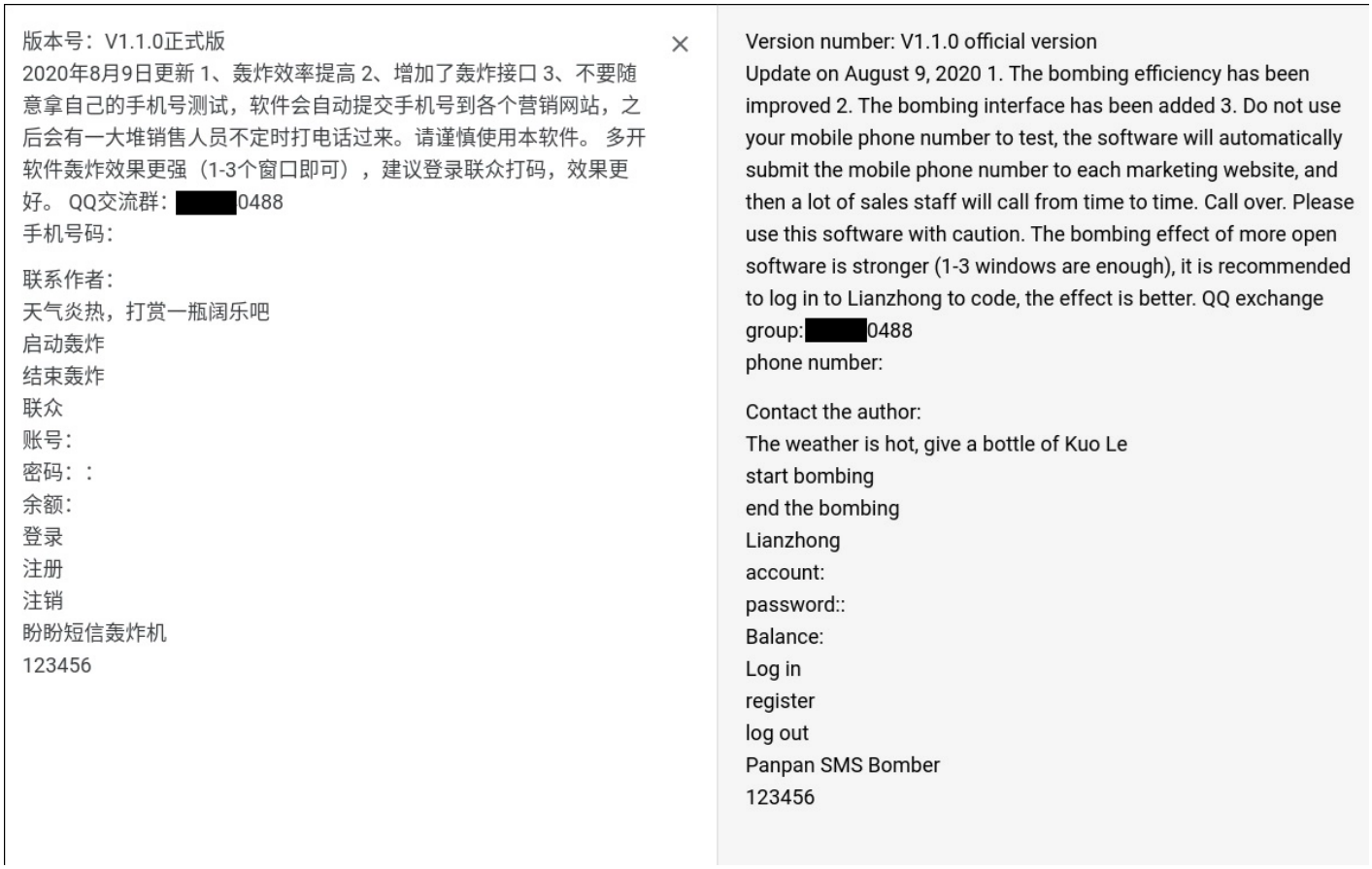

Figure 4: Google-translated highlighted strings from a sandbox run of the SMS bomber.

The payloads

As mentioned before, in addition to running an embedded executable (SMS Bomber in this specific chain), Nimbda injects a piece of code into a launched notepad.exe process, in this particular infection chain kicking off a three-tier execution flow.

Injected Shellcode

First, **initial shellcode** glue is injected into `notepad.exe` and run. This shellcode contacts an attacker-controlled GitHub or Gitee repository and downloads a `EULA.md` file that contains an obfuscated IP address. A Python script to perform deobfuscation for the IP address, as done in the initial shellcode, is included below (the resulting IP address in the below example is 159[.]75[.]144[.]13)

```
sample_obfuscated_ip =
"2b12f535097a994afe2c329ddc436792048a64b033c02d7938076350507859ff"
import re

def deobfuscate_ip(buf: str) -> str:
    buf = buf[32:]
    for (ptrn, repl) in [("[abc]",'.'),('9','0')]:
        buf = re.sub(ptrn,repl,buf)
        buf = buf.split('d')[0]
        buf = ''.join([chr(ord(c)+1) if c!='.' else '.' for c in buf])
    return buf
```

```
ip = deobfuscate_ip(sample_obfuscated_ip)
print(ip) #159.75.144.13
```

The initial shellcode then requests `https://<IP Address>/index.htm` which is a next-stage obfuscated executable, which is run via process hollowing of a launched `dllhost.exe` process.

Yahoyah

This deobfuscated payload is an improved version of `TROJ_YAHOYAH` backdoor/downloader used by Tropic Trooper previously, as detailed in the original report. This version collects information about local wireless network SSIDs in the victim machine's vicinity, in addition to information collected by the original Yahoyah (such as computer name, MAC address, OS version, installed AV products, and presence of WeChat and Tencent files). The collected information is formatted and sent to the C&C server.

```
p_uSSIDLength = &ptr_available_network_list->Network[available_networks_index_multiplied_by_size / 0x274].dot11Ssid.uSSIDLength;
memcpy_0(
  ssid_name,
  ptr_available_network_list->Network[available_networks_index_multiplied_by_size / 0x274].dot11Ssid.ucSSID,
  *p_uSSIDLength);
v5 = *p_uSSIDLength;
++ssid_number;
ssid_name[v5] = 0;
sprintf_s(
  Source,
  0x800u,
  "SSID_%d:%s|%d%%",
  ssid_number,
  ssid_name,
  curr_wifi_network_ptr->dot11DefaultAuthAlgorithm);
strcat_s(Destination, 0x800u, Source);
wlan_bss_list_index_multiplied_by_size = 0;
if ( !WlanGetNetworkBssList(
        phClientHandle,
        &ppInterfaceList->InterfaceInfo[interface_info_index_multiplied_by_size / 0x214].InterfaceGuid,
        &ppAvailableNetworkList->Network[available_networks_index_multiplied_by_size / 0x274].dot11Ssid,
        ppAvailableNetworkList->Network[available_networks_index_multiplied_by_size / 0x274].dot11BssType,
        ppAvailableNetworkList->Network[available_networks_index_multiplied_by_size / 0x274].bSecurityEnabled,
        0,
        &ppWlanBssList) )
{
  ptr_wlan_bss_list = ppWlanBssList;
  ssid_index = 0;
  if ( ppWlanBssList->dwNumberOfItems )
  {
    do
    {
      network_mac_address = (PWLAN_BSS_LIST)((char *)ptr_wlan_bss_list
                                           + wlan_bss_list_index_multiplied_by_size);
      v10 = ssid_index + 1;
      sprintf_s(
        Buffer,
        0x400u,
        "|MAC%d:%02X:%02X:%02X:%02X:%02X:%02X|",
        ssid_index + 1,
        network_mac_address->wlanBssEntries[0].dot11Bssid[0],
        network_mac_address->wlanBssEntries[0].dot11Bssid[1],
        network_mac_address->wlanBssEntries[0].dot11Bssid[2],
        network_mac_address->wlanBssEntries[0].dot11Bssid[3],
        network_mac_address->wlanBssEntries[0].dot11Bssid[4],
```

Figure 5: SN-Yahoyah iterates over available WiFi SSIDs.

```
memset(computer_name, 0, sizeof(computer_name));
nSize = 2048;
GetComputerNameA(computer_name, &nSize);
os_version = get_os_version();
memset(mac_addr, 0, sizeof(mac_addr));
get_mac_addr();
memset(is_tencent, 0, sizeof(is_tencent));
check_for_tencent(is_tencent);
memset(is_wechat, 0, sizeof(is_wechat));
check_for_wechat(is_wechat);
memset(wifi_networks, 0, sizeof(wifi_networks));
scan_wifi_networks(wifi_networks);
memset(c2_info, 0, sizeof(c2_info));
c2_data_format = (char *)aees_decrypt_string("NdogccQ0nfv7RPvJvO/5noZkKaTrvVZYISmDVz9dwemXZTtg+sUdulzO/6+qiMNdyDwtqMgENUKNDBYdyOxP6w==");//
                                             // AV(%d)||QN(%s)||WN(%s)||HN(%s)||MA(%s)||WF(%s)||VE(%s)
av_product = search_for_av_product();
sprintf(
  c2_info,
  c2_data_format,
  av_product,
  is_tencent,
  is_wechat,
  computer_name,
  mac_addr,
  wifi_networks,
  "SN202012_x86");
```

Figure 6: SN-Yahoyah reports collected data to its C&C server

Note the “SN” prefix in the string `SN202012_x86` — these strings are believed to be internal version identifiers used by the attackers (they also appear, in identical format, in the “USBFerry” malware used by Tropic Trooper). This version of Yahoyah with SSID-scanning functionality is tagged with a unique string prefix of SN (we speculate that it stands for “Spot Net” or “Scan Networks”, or something to that extent).

The malware then proceeds to act similarly to known versions of Yahoyah, setting up a DLL for purposes of persistence – it’s either registered as a service or referenced by `Run` or `Winlogon` registry key. This DLL downloads an image from the C&C server which actually contains a steganographically-encoded final payload. The DLL will extract the PE and run an exported function `StartWork`.

Other than this Yahoyah variant, the infection chains we have observed make heavy use of other custom variants of Yahoyah (some of which are not new, and predate the above-mentioned activity by a while). While these custom variants all share many features — such as steganography methods, a homebrew hash function for performing obfuscated lookup of process names, and tweaked encryption (see below) — there is also a grab-bag of features that each appear in some variants but not others. These features are all generic backdoor fare, such as anti-analysis checks or a mutex to prevent simultaneous executions (implemented in some samples via writing to an INI file, in others via direct use of the windows API). In one specific sample, we saw use of the Windows Transactional NTFS API for loading and executing an embedded PE inside a seemingly legitimate process (infamously used in the Process Doppelganging technique).

Final payload

The final payload encoded in the image is **TClient**, a backdoor known to have been used by Tropic Trooper campaigns.

A full analysis of TClient, as well as the original Yahoyah trojan and the Persistence DLL (TROJ_YAHAMAM) is available in the original “Tropic Trooper” report.

```
DWORD __stdcall StartAddress(LPVOID lpThreadParameter)
{
  DWORD CurrentProcessId; // eax
  const char *v2; // eax
  const char *v3; // eax
  CHAR pszPath[260]; // [esp+Ch] [ebp-108h] BYREF
  HANDLE hToken; // [esp+110h] [ebp-4h] BYREF

  CurrentProcessId = GetCurrentProcessId();
  hToken = 0;
  get_token(&hToken, CurrentProcessId);
  memset(pszPath, 0, sizeof(pszPath));
  SHGetFolderPathA(0, CSIDL_APPDATA, hToken, 0, pszPath);// APPDATA
  strcat_s(pszPath, 0x104u, "\\");
  v2 = (const char *)decrypt_string(enc_str_multidesktop);// MultiDesktop
  strcat_s(pszPath, 0x104u, v2);
  if ( !SHCreateDirectoryExA(0, pszPath, 0) )
    SetFileAttributesA(pszPath, 6u);
  strcat_s(pszPath, 0x104u, "\\");
  v3 = (const char *)decrypt_string(enc_str_multidesktop_jpg);// MultiDesktop.jpg
  strcat_s(pszPath, 0x104u, v3);
  exec_payload_from_image(pszPath);             // %APPDATA%\\MultiDesktop\\MultiDesktop.jpg
  return 1;
```

Figure 7: Persistence DLL code for loading and executing a payload hidden inside an image

Targeting modus operandi

At this point, we ought to ask what this unorthodox execution chain was supposed to achieve, exactly. While there are no definitive answers, we can provide an educated guess. Whoever crafted the Nim loader took special care to give it the same executable icon as the SMS Bomber that it drops and executes. Therefore the entire bundle works as a trojanized binary. That is: the victim launches what they think is just an SMS Bomber, but is actually an SMS Bomber plus a backdoor. An attack making use of such a trojanized binary is necessarily aimed at a rather unorthodox target — people who'd use such an "SMS Bomber" tool in the first place.

AEES: Analysts Hate this One Weird Trick

The routines that deobfuscate some of the strings in the SN Yahoyah sample we obtained appeared at first glance to be simple AES, except decryption failed when using AES and the given key. Further digging into the malware's implementation of AES led to the following curious piece of code:

```
aes_createroundkey();
aes_addroundkey_(10);
for ( i = 9; i > 0; --i )
{
    aes_invShiftrows_();
    aes_invBytesub_();
    aes_addroundkey_(i);
    aes_invMixcolumn_();
    aes_invShiftrows_();
    aes_invBytesub_();
    aes_addroundkey_(i);
    aes_invMixcolumn_();
}
aes_invShiftrows_();
aes_invBytesub_();
aes_addroundkey_(0);
```

The inverted sequence of round operations (`ShiftRows`, `SubBytes`, `AddRoundKey`, `MixColumns`) is performed twice, making this not quite AES — one could call it AEES, maybe. A hand-written implementation of AEES was able to properly decrypt the obfuscated strings.

When we were young and impressionable we'd often ask ourselves, "why do malware developers roll their own crypto all the time? Doesn't it just lead to mistakes and reduced security?". While sometimes it inevitably does, it also serves the important purpose of wasting copious amounts of analyst time.

Let's be candid and share our average and mediocre workflow when encountering an encryption algorithm. First we look for tell-tale strings or constants, using FindCrypt or some such, to identify the algorithm. Then, if that approach fails, we dynamically probe the code and attempt to tease out which algorithm is in use by seeing how it reacts to some choice test inputs. Finally, when all else fails, we become resigned to our cruel fate — we are looking at 'Unicorn Crypto' and as far as we are concerned there is exactly one working implementation of it, and it's inside the malware we are researching right now. Then begins the eye-straining session of looking at disassembly, decompilation, re-implementation, debugging, …

```
aes_createroundkey();
aes_addroundkey_(10);
for ( i = 9; i > 0; --i )
{
    aes_invShiftrows_();
    aes_invBytesub_();
    aes_addroundkey_(i);
    aes_invMixcolumn_();
}
aes_invShiftrows_();
aes_invBytesub_();
aes_addroundkey_(0);
```

Getting an analyst to go through that entire rigmarole is a cruel and effective feat, especially for the meager cost on the malware author's side. They just need the knowledge and self-confidence to mess with the crypto in a way that will not render it nonoperational. Executing the round operation sequence of AES twice reads like a textbook example of that. The average malware developer will wonder "Is it weaker now?", but anyone who knows anything about AES immediately knows that the answer is 'no'. (Let's not even get to the slightly-savvier folk who'll dare ask, 'does it matter? When attacking the data at rest, you have the key; and trying to attack the data in transit is, broadly speaking, insane').

The only criticism we can think of regarding this master-stroke is the authors' use of AES in the ECB mode of operation, which famously allows attackers to See the Penguin. But otherwise, good show, malware authors. That's some nice chunk of time lost that we will never get back again.

Campaign Infrastructure

We identified a few dozens of hosts served for this campaign. Most of the infrastructure used in operation the current operation has been hosted in Chinese hosting providers:

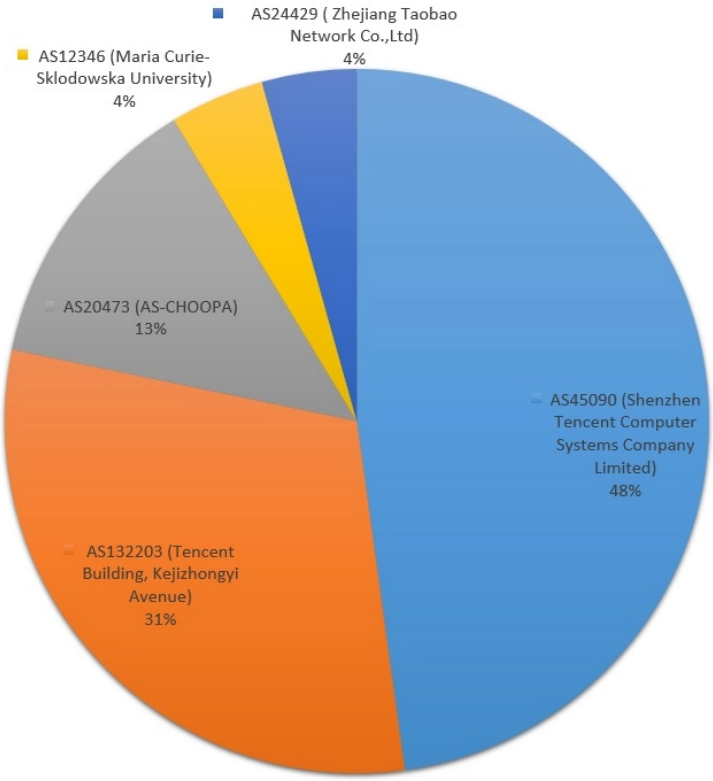

Figure 8: Distribution of the servers within different ASN

The relevant host names were extracted directly from samples, as well as from deobfuscated git commits made to the repository used by the above-mentioned Nim loader. Note that the first host is actually a local IP address, and it appears in a commit from August 12, 2020. This host might have been used to test the Nim loader locally. There was a 3-week gap in activity before the next committed host.

Date	Obfuscated host	Deobfuscated host
12/08/2020	78e1f384a5ea1092e8ae16b50662ca2b081a057b9c045d486cc94a20c00bae87	192[.]168[.]1[.]156
02/09/2020	78e1f384a5ea1092e8ae16b50662ca2b191a071b019c86d486cc94a20c00bae87	212[.]182[.]121[.]97
02/09/2020	78e1f384a5ea1092e8ae16b50662ca2b34a66b067c36d486cc94a20c00bae87	45[.]77[.]178[.]47
03/09/2020	78e1f384a5ea1092e8ae16b50662ca2b044a027b044c070d486cc94a20c00bae87	155[.]138[.]155[.]181
09/09/2020	78e1f384a5ea1092e8ae16b50662ca2b34a65b107c136d486cc94a20c00bae87	45[.]76[.]218[.]247
16/09/2021	2b12f535097a994afe2c329ddc436792048a64b044c02d7938076350507859ff	159[.]75[.]155[.]13
16/09/2021	2b12f535097a994afe2c329ddc436792048a64b033c02d7938076350507859ff	159[.]75[.]144[.]13

Table 1: Hosts extracted from obfuscated git commits.

Connection with Tropic Trooper

We have concluded that this activity is probably connected to Tropic Trooper, and TA428 by proxy, based on the following TTPs:

- The loading process of the final payload, from the use of **steganography** to hide a payload DLL inside of a downloaded image, to the **exported function name** of the payload DLL, invoked by SN Yahoyah in order to run the logic inside — StartWork.
- The use of **TClient** as a final payload.
- The **C&C check-in format** shared almost verbatim by SN Yahoyah and USBFerry, a malware deployed by Tropic Trooper that targets air-gapped networks. The version naming and the victim's data formatting in both samples are very similar, considering 4 years of development and tool differences (see above, figure 6).

Summary

The observed activity cluster paints a picture of a focused, determined actor with a clear goal in mind. Usually, when 3rd-party benign (or benign-appearing) tools are hand-picked to be inserted into an infection chain, they are chosen to be the least conspicuous possible; the choice of an “SMS Bomber” tool for this purpose is unsettling, and tells a whole story the moment one dares to extrapolate a motive and an intended victim.

The surgical modifications to AES are evidence that this actor probably has a decent grasp of block cipher internals. And, finally, the addition of network scanning functionality to Yahoyah shows us that the tools at the disposal of threat actors will be honed and improved as time passes, one way or the other. It falls to the security industry to try to keep pace with these changes as they happen, and react accordingly. For one thing, the next time we run ‘FindCrypt’, we will surely be at least a *bit* more eager to manually verify the output.

Check Point’s evasion-resistant technology maximizes zero-day protection, providing comprehensive coverage of attack tactics, file-types, and operating systems

Addendum 1: Indicators of Compromise

Hashes

MD5	Malware Family	Version
8ee94c4d4e13bf59524e1d3eb9c8c846	Nimbda Loader	
916e8b1a9d91f3649b33dd8bc0f09a8d	Yahoyah	
8b7b602e05604f61685a2cbfc16313af	Yahoyah	
68a3198fd77a063f46a1d3cddb266f02	Yahoyah	
fa9b9fbaf58ad3a1b83c6f98e67446c7	Yahoyah	
ca88c5d5f4409179b3820db7ce6b68fb	Yahoyah	
fc7b582befae6a03830c47fa2a5a8a27	Yahoyah	
a517859ee63713fa364e960def411433	Yahoyah	
64b9cf63f03eaab3959e20a2cd23b704	SN-Yahoyah	SN202012_x86
7f8b4a962edea1dd9d552fbc907f76bb	SN-Yahoyah	SN202012_x86
4207445f6cddc36d4a22151db7432158	SN-Yahoyah	202006
560545cddf3cba248702d0edb7fabff3	SN-Yahoyah	202006
87b97be92584f86dc58bf444dfe85f9d	SN-Yahoyah	
87f62453c5b8d5bd8cc6d599f1326c43	Persistence DLL	
17c87aba7eccc2148672d9e61e509906	Persistence DLL	
1e7df4685b1d4a6886215d2b0a8d9370	Persistence DLL	
8d4e128b6701f70f7337e2a479a7ec5e	Persistence DLL	
4949147393b623d3f99b858bc6467c06	Persistence DLL	
7c4cd57219d560084075beecc81532d7	Persistence DLL	
0cbca21300763c6059ccdf8f0fd46319	TClient	
f58f23c9478ca8d1bbdc7be78e7e42e0	TClient	20201212
2a68a55b226abc4e7aa940471088ceab	TClient	
223e675ddbdf74c560886f90fc114297	TClient	20220329_DOS
93f06130c7c17502bcc1a7900057898c	TClient	1998_DX_2
88c92306a190bcf1fd1f54fb327ce124	TClient	20201212
3de8c5952bf1147839399dde1eb05ce4	TClient	
2e8a8e03e82649d46c5deee2f54ce470	TClient	
760a41f6d3cac656d72d8f3d198ab9dd	TClient	202009

Infrastructure

ak[.]buycheap[.]cn
api[.]cnicchina[.]com
45[.]76[.]218[.]247

159[.]75[.]83[.]212
134[.]175[.]197[.]144
106[.]53[.]120[.]204
43[.]129[.]177[.]152
49[.]232[.]142[.]8
82[.]156[.]178[.]135
159[.]75[.]81[.]151
118[.]195[.]161[.]141
159[.]75[.]144[.]13
212[.]182[.]121[.]97
45[.]77[.]178[.]47
155[.]138[.]155[.]181
132[.]232[.]92[.]218
101[.]32[.]36[.]76
43[.]154[.]74[.]7
43[.]154[.]88[.]192
43[.]154[.]85[.]5
43[.]134[.]194[.]237
82[.]157[.]51[.]214
150[.]109[.]114[.]190
82[.]157[.]62[.]199

Addendum 2: Attacker Working Hours

Based on the Git commits and the compilation timestamps, we can guess the working hours of the attackers.

While compilation timestamps can be tampered with, the times of the Git commits are likely more reliable.

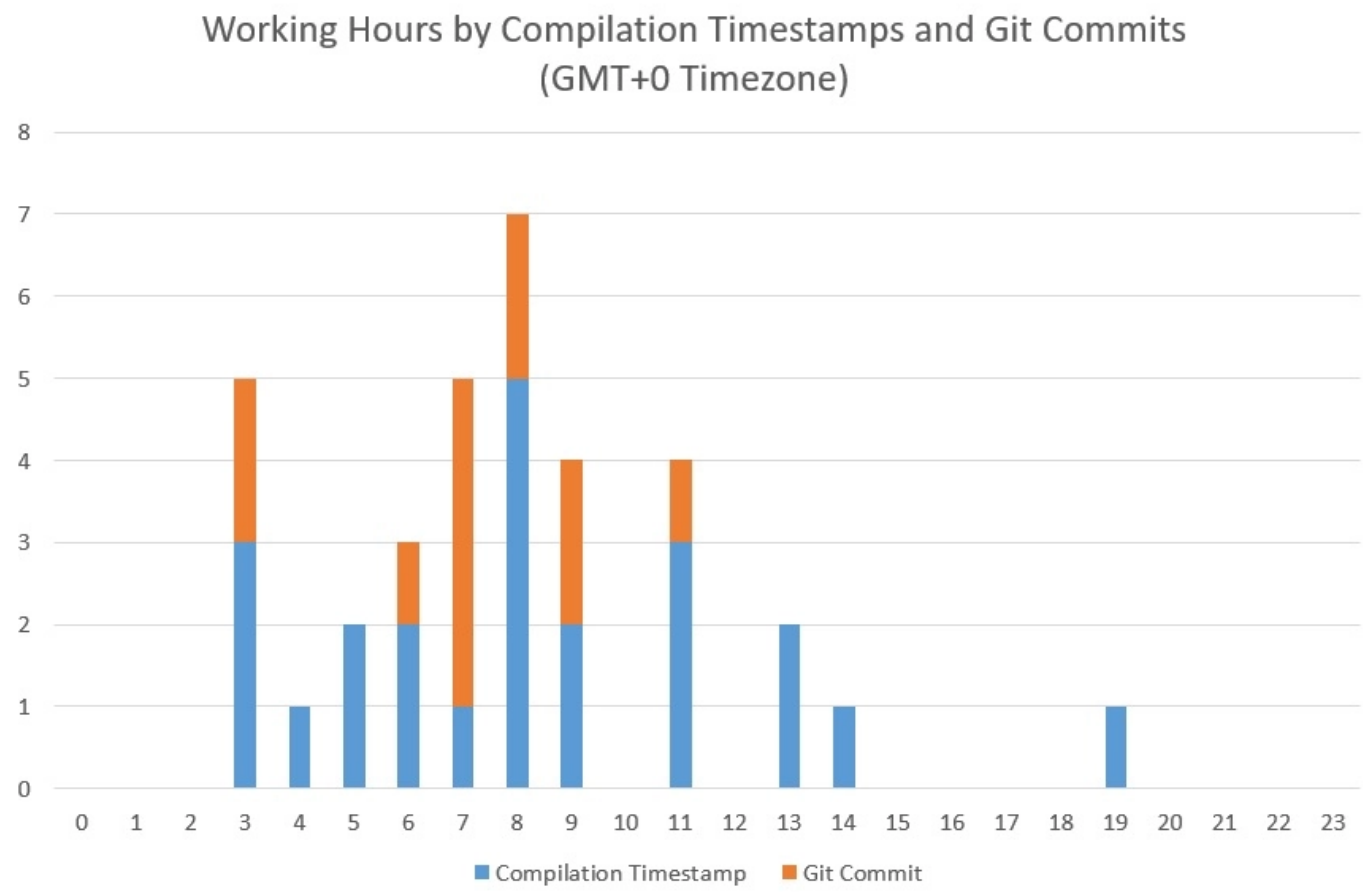

Figure 9: Distribution of actor's working hours