Greenbug's DNS-isms | NETSCOUT

Archived: 2026-04-05 16:30:18 UTC

Over the past few months there has been a lot of research and press coverage on the Shamoon campaigns. These have been the attacks on Saudi Arabian companies where a destructive malware known as Disttrack was deployed. The malware, using stolen credentials, spreads throughout the targeted networks and then at a set date and time wipes the disks attached to the victim computers.

A big question kept coming up about Disttrack: how were the hardcoded credentials stolen in the first place? In January 2017, Symantec proposed one possible answer in their "Greenbug cyberespionage group targeting Middle East, possible links to Shamoon" blog post. In it they discuss a threat group they call Greenbug and a custom info-stealing remote access trojan (RAT) the group uses called Ismdoor. Symantec's possible link between Greenbug and Shamoon is summed up with: "… the group compromised at least one administrator computer within a Shamoon-targeted organization's network prior to W32.Disttrack.B being deployed on November 17, 2016. " While there are few details about the threat group, there are even fewer details about the Ismdoor malware. Besides the brief description in the above blog, RSA released a post in February 2017 giving an overview of its HTTP communications. *Update: Shortly before publication we found an additional February 2017 analysis of "ISM RAT" by NCC Group.*

Although analysis has been slow, Ismdoor development has been quick! One major change in recent versions has been the replacement of the old HTTP based command and control (C2) functionality with a custom covert channel using AAAA DNS queries for IPv6 addresses. There are a lot of moving packets to this new mechanism so this post takes a look at our analysis of it so far.

Samples

The first sample used in this post is available on VirusTotal. It caught our attention due to the descriptive debug string:

```
C:\Projects\DNS BOT\Bot\x64\Release\Ism.pdb
```

This is version 1.0.3 and was compiled on 2016-11-11. It was first seen on VirusTotal on 2017-03-27 and was submitted from Saudi Arabia. The C2 domains for this sample are:

- winrepp[.]com
- winsecupdater[.]com

They were registered on 2016-06-09 and 2016-11-06. Since the C2s were down at the time of this research, we searched for and found a newer sample via this tweet. This sample also has a descriptive debug string:

```
C:\Users\me\Documents\Visual Studio 2013\Projects\DNS_Bot\v 10.0.19\x64\Release\Nrtscan.pdb
```

It is version 1.0.19 and was compiled (and first submitted to VirusTotal) on 2017-04-05. Again, it was submitted from Saudi Arabia. The C2 domains for this sample were registered on 2016-12-04 and will be discussed next.

Malware Configuration

Ismdoor has an encrypted configuration that contains a primary and secondary C2 domain, various identifiers, timeouts, and flags. These values can be updated by later C2 commands. A substitution cipher is used to decrypt the configuration when it is needed. The character mapping has been consistent across samples and we have made available a Python snippet of it on Github. The initial configuration for the analyzed sample looks like this:

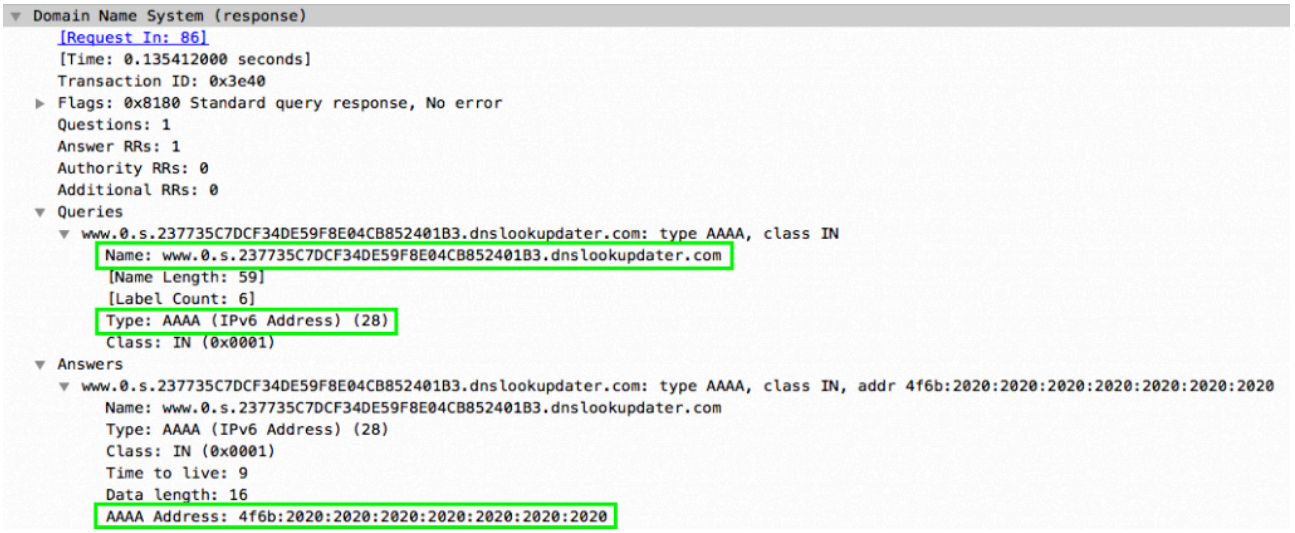
```
.dnslookupdater[.]com (primary C2 domain)
 30 (alive timeout)
 -1 (appId)
 0
 -1
 0
 0
 .dnssecupdater[.]com (secondary C2 domain)
 0
 0
 20000
 00000000-0000-0000-0000-000000000000 (uniqueId)
```

Command and Control

The C2 mechanism is built on top of DNS and can be separated into three layers: DNS, transport/session, and C2 messages.

DNS Layer

All data sent between the bot and the C2 is done using AAAA DNS UDP queries. Data to the C2 is via specially crafted query names and data from the C2 is returned via IPv6 addresses. The bot side of the connection drives all communications. An example query and response looks like this:

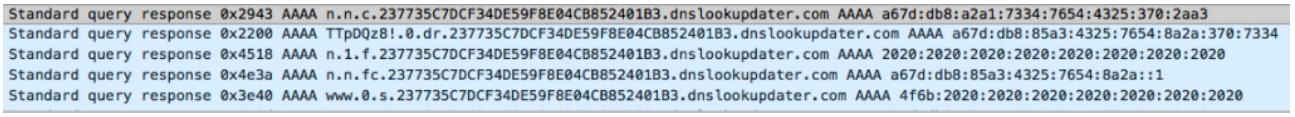


Transport/Session Layer

The next layer up can be thought of as a transport/session layer (think similar to UDP and TCP). Here, communications are divided up into sessions and most sessions consist of at least five DNS requests and replies:

1. Establish session ID
2. Send messages to C2
3. Send message count to the C2
4. Ask for count of response messages from the C2
5. Receive messages from the C2

An example of this typical session looks like this:



To establish a session, the bot generates a session ID (32 uppercase hexadecimal characters) and sends it to the C2 using a query name of the following format:

```
n.n.c.<session id>.c2.com
```

Here is the example from the above screenshot:

```
n.n.c.237735C7DCF34DE59F8E04CB852401B3.dnslookupdater[.]com
```

The C2 will respond with a static IPv6 address of:

```
a67d:db8:a2a1:7334:7654:4325:370:2aa3
```

Next, the bot sends upper layer C2 messages (described in the next section) to the C2 using query names formatted like this:

```
<encoded message>.<message number>.dr.<session id>.c2.com
```

Here is the example from above:

```
TTpDQz8!.0.dr.237735C7DCF34DE59F8E04CB852401B3.dnslookupdater[.]com
```

Message numbers start at zero and messages are base64 encoded with the following characters changed:

- = -> !
- / -> &
- + -> @

The above decodes to the following:

```
M:CC?
```

Again, the C2 responds with a static but different, IPv6 address:

```
a67d:db8:85a3:4325:7654:8a2a:370:7334
```

In the third step the bot sends a sent message count to the C2 using the following format:

```
n.<message count>.f.<session id>.c2.com
```

For example:

```
n.1.f.237735C7DCF34DE59F8E04CB852401B3.dnslookupdater[.]com
```

Here, the C2 responds with an address of hex encoded characters, namely all space characters:

```
2020:2020:2020:2020:2020:2020:2020:2020
```

The fourth step has the bot asking the C2 how many messages it should expect from it. These requests are formatted liked this:

```
n.n.fc.<session id>.c2.com
```

It looks like this for our example session:

```
n.n.fc.237735C7DCF34DE59F8E04CB852401B3.dnslookupdater[.]com
```

For this exchange the C2 responds with an IPv6 address that is static except for the last eight bytes of the address, which are used for the message count. For example if the C2 is going to send one message to the bot it will respond like this:

```
a67d:db8:85a3:4325:7654:8a2a::1
```

The final step involves the bot receiving messages from the C2. It indicates which message it wants to receive with query names like:

```
www.<message number>.s.<session id>.c2.com
```

Message numbers are again zero based, so to receive the first message of this example session the bot will send:

```
www.0.s.237735C7DCF34DE59F8E04CB852401B3.dnslookupdater[.]com
```

The C2 responds with addresses containing the hex encoded messages like this:

```
4f6b:2020:2020:2020:2020:2020:2020:2020
```

This decodes to “Ok “. To help analyze these sessions we wrote a parser in Python available on Github. Using the parser, the above session can be visualized like this:

```
session #0
session id: 237735C7DCF34DE59F8E04CB852401B3
send msg 0 to c2: M:CC?
sent 1 msgs to c2
expecting 1 msgs from c2
recv msg 0 from c2: 4F6B20202020202020202020202020 (Ok            )
```

When larger amounts of data need to be transferred (for file upload and download) a different type of session is used:

1. Send file message count to the C2
2. Send file messages to C2
3. Periodically ask the C2 if it is missing any sent file messages
4. Resend any missed file messages

For file transfers, the file message count is sent to the C2 using query names formatted like this:

```
n.<message count>.<session id>.c2.com
```

Session IDs are derived from the C2 commands that requested the file transfer. The C2 responds with a static IPv6 address:

```
a67d:db8:a2a1:7334:7654:4325:370:2aa3
```

File messages are then sent using the following types of query names:

```
<encoded data>.<message number>.d.<session id>.c2.com
```

Message numbers start at zero and data is encoded as above. The C2 will either not respond to these data message or send a DNS “Server failure” error response. Periodically the bot asks the C2 if it has missed any file messages using the following query:

```
uff<message count>.<message count>.<session.id>.c2.com
```

Here, the C2 responds with an address of hex encoded space characters. The bot then sends the following queries, as described above, to receive an answer from the C2:

```
n.n.fc.<session id>.c2.com
www.<message number>.s.<session id>.c2.com
```

The C2 then responds with a hex encoded list of missed messages. For example if the C2 were missing message numbers 5, 9, 15, and 16 it would send this IPv6 address:

```
3136:2c31:352c:392c:3520:2020:2020:2020
```

which decodes to: “(16,15,9,5)”. Missing file messages are resent with query names like:

```
<encoded data>.<message number>.dl.<session id>.c2.com
```

As with the send file message above, the C2 will either not respond to these replays or send an error message. A parsed example of this type of session looks like this:

```
session #13
session id: a1a13274c08f4730b88f1715de38068c
send file msg count (25) to c2
send file msg 0 to c2
send file msg 1 to c2
...
```

```
send file msg 24 to c2
ask c2 for missing file msgs
expecting 1 msgs from c2
recv msg 0 from c2: 31362C31352C392C3520202020202020 (16,15,9,5       )
resending file msg 16 to c2
...
```

C2 Messages

The third layer contains C2 messages and there are at least eight of them. Some messages have additional parameters. We'll go through each message and show some examples using the above parser.

M:CC?

"CC" likely means "Connection Check" or "Check Connection". This is a periodic message that checks whether the C2 server is alive and well. An example message and response looks like this:

```
send msg 0 to c2: M:CC?
recv msg 0 from c2: 4F6B20202020202020202020202020202020 (Ok             )
```

M:ME? "ME" likely means "Message" and these are used to send status messages back to the C2. Here is an example:

```
send msg 0 to c2: M:ME?appId=-1&message=Executed Successfully
recv msg 0 from c2: 20202020202020202020202020202020 (               )
```

M:AV? "AV" may mean "AliVe" or "AVailable". This is a periodic message that has two purposes. The first is to initialize two identifiers (stored in the previously mentioned config) "appId" and "uniqueId". The latter is akin to a bot ID, but it is unclear what the former represents. It changes on each malware run and tends to be a low value integer (some examples seen: 5, 8, 11, 12). An example of this message looks like:

```
send msg 0 to c2: M:AV?appId=-1&uniqueId=00000000-0000-0000-000
send msg 1 to c2: 0-000000000000
recv msg 0 from c2: 41707049647C7C7C38267569643D3432 (AppId|||8&uid=42)
recv msg 1 from c2: 6435623934352D383062362D34336430 (d5b945-80b6-43d0)
recv msg 2 from c2: 2D396330332D33323736316233323866 (-9c03-32761b328f)
recv msg 3 from c2: 34372020202020202020202020202020 (47             )
```

After the IDs have been initialized this message asks how many commands are available for the bot on the C2. These messages look like:

```
send msg 0 to c2: M:AV?appId=8&uniqueId=42d5b945-80b6-43d0-9c03
send msg 1 to c2: -32761b328f47
```

```
recv msg 0 from c2: 3120202020202020202020202020202020 (1              )
```

M:ReId?

"ReId" may mean "ReplyID" and verifies the appId with the C2 after it is initialized above:

```
send msg 0 to c2: M:ReId?Id=8
recv msg 0 from c2: 4F6B2020202020202020202020202020 (Ok             )
```

M:GAC?

"GAC" likely means "Get A Command" and that's what it does:

```
send msg 0 to c2: M:GAC?appId=8
recv msg 0 from c2: 3032306437346135 2D3230613332D3433 (020d74a5-20a3-43)
recv msg 1 from c2: 65372D6164636 42D6634383631356466 (e7-adcd-f48615df)
recv msg 2 from c2: 356137347C7C7C476574436F6E666967 (5a74|||GetConfig)
recv msg 3 from c2: 3A3A3A313020202020202020202020 (:::10           )
```

Bot commands will be discussed in a later section, but one thing to notice is that each command contains a unique GUID (020d74a5-20a3-43e7-adcd-f48615df5a74 in this example).

M:CR?

"CR" likely means "Command Received" and it acknowledges receipt of a command:

```
send msg 0 to c2: M:CR?cd=020d74a5-20a3-43e7-adcd-f48615df5a74
recv msg 0 from c2: 4F6B2020202020202020202020202020 (Ok             )
```

Notice the command GUID is returned to the C2.

M:SF?

"SF" likely means "Send File" and is used to send files and command results. Here is the result of the GetConfig command from above:

```
send msg 0 to c2: M:SF?commandId=CmdResult=020d74a5-20a3-43e7-a
send msg 1 to c2: dcd-f48615df5a74|||GetConfig:::.dnslookupdate
send msg 2 to c2: r[.]com^M
30^M
8^M
1^M
-1^M
0^M
0^M
```

```
.dnssecupdater[.]com
send msg 3 to c2: ^M
0^M
0^M
20000^M
42d5b945-80b6-43d0-9c03-32761b
send msg 4 to c2: 328f47
recv msg 0 from c2: 20202020202020202020202020202020 (               )
```

M:GF?

"GF" likely means "Get File" and is similar to "Send File" To see more of what Ismdoor's DNS C2 protocol and messages look like, we've put up a parsed PCAP from an infected VM up on Github.

Bot Commands

Ismdoor has a bunch of functionality and commands. While this post won't go into detail on them, most are self-describing:

- ChangeAliveSeconds – sets alive timeout in config
- ChangeAddress – sets C2 domains in config
- SI – likely means "System Information" – runs a bunch of system commands and sends their output to the C2
- GetConfig - sends config to C2
- RunNewVersion - updates self
- restart - restarts self
- remove - removes self
- CreateMimi1Bat - likely executes Mimikatz (executes PowerShell scripts: ccd61.ps1 and Invoke-bypassuac)
- RAAD – executes PowerShell scripts ivb.ps1 and Invoke-EventVwrBypass
- CLRAD - deletes files associated with RAAD command
- ExecutePC - likely executes Powercat—PowerShell version of Netcat (executes PowerShell scripts dp.ps1 and powercat)
- FastAlive - sets "FastAlive" in config. Unknown ramifications
- ExecuteKL - likely executes a keylogger (executes Winit.exe and sends "Start Keylog Done" message back to the C2)
- RemoveKL - kills and removes keylogger associated files
- GetVersion - sends malware version to C2
- PauseUpload
- ResumeUpload
- PauseDownload
- ResumeDownload
- PWS - takes a screenshot
- ImmediateResetRam - sets "ResetRam" flag in config. Unsure of the ramifications

- DownloadFile
- UploadFile

If a command doesn't match a predefined one, it passes it to a command shell and returns the result as a file.

Conclusion

While a definitive connection between Greenbug's Ismdoor and the Shamoon campaigns is still up in the air, we wanted to highlight some notes in its favor:

- Symantec saw an earlier version of Ismdoor on a Shamoon targeted host shortly before a Disttrack attack (unverified by Arbor)
- All the known samples of the DNS variant of Ismdoor have been originally submitted to VirusTotal from Saudi Arabia
- The DNS variant of Ismdoor has explicit capabilities for stealing credentials via its CreateMimi1Bat and ExecuteKL commands
- Pivoting on the CreateMimi1Bat command's PowerShell script name (ccd61.ps1) leads to a Shamoon related blog post by McAfee [9] where they trace a malicious, macro laden spearfishing email to the download and execution of an earlier version of Ismdoor (though they don't name the malware they're describing)

What is definite though is that Ismdoor is active and under development. This post takes a close look at one of its major new features: a full featured C2 mechanism that uses DNS.
