New Robust Technique for Reliably Identifying AsyncRAT/DcRAT/VenomRAT Servers

By Axel Mahr

Published: 2024-04-20 · Archived: 2026-04-05 21:19:21 UTC

Looking for indicators in order to detect C2 servers of the QuasarRAT family is nothing particularly new. Up to now, two main approaches are already known:

- It is trivial to identify QuasarRAT/AsyncRAT/DcRAT/VenomRAT when default TLS certificates are used. Then, we can look at the certificate's Subject/Issuer CN which directly gives us the information we look for. E.g., for AsyncRAT, Issuer CN and Subject CN are `AsyncRAT Server`. This approach has been explained in various reports and blog posts.
- Fingerprints like JA3S, JA4S/JA4X and JARM also allow for detecting these RATs.

However, both approaches have their drawbacks. Certificate-based indicators don't work anymore once the server doesn't use the default certificate. And using a non-default certificates is not a high burden at all. E.g., for AsyncRAT, a Subject and Issuer CN different from `AsyncRAT Server` is even definable via the GUI. Fingerprinting approaches may also be not sufficient since it has been shown that especially JA3S and JARM are heavily prone to false positives.

Hence, I looked for more robust and accurate ways for detecting these RATs through a scan from outside. Since AsyncRAT and DcRAT are open-source on Github (see here and here), I could easily look at the source code, deploy a C2 server locally and play around a bit. While exploring, I found some weirdly interesting behavior for AsyncRAT, DcRAT and VenomRAT:

It is possible to send valid C2 packets to the server as an unauthenticated client, which are then blindly processed by the server. By that, we can send custom packets which provoke a certain server response. The response reliably indicates us, which RAT we are facing.

For C2 communication, AsyncRAT, DcRAT and VenomRAT use a custom packet format inside TLSv1. Each packet begins with four bytes indicating the size of the following payload. The payload itself begins with again four bytes followed by gzip-compressed data which is in turn serialized using MessagePack. The serialized data are key-value pairs where the value for the key `Packet` / `Pac_ket` indicates the message type. That's it. No further client authentication or encryption (besides the TLS stuff) is done. Thus, exemplarily with the following code, we can get easily detect AsyncRAT (replace `IP` and `PORT` as you like):

```
 import socket
import ssl
import gzip
import msgpack # pip install msgpack-python
from pwn import p32
```

```
# init socket for connecting to AsyncRAT server
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
context = ssl.SSLContext(ssl.PROTOCOL_TLSv1)
context.set_ciphers('DEFAULT:@SECLEVEL=0')
context.verify_mode = ssl.CERT_NONE

# connect
wrapped_socket = context.wrap_socket(sock)
wrapped_socket.connect((IP, PORT))

# craft valid C2 packet of message type "Ping"
# (for detecting DcRAT/VenomRAT, just change b"Packet" to b"Pac_ket")
payload = gzip.compress(msgpack.packb({b"Packet": b"Ping"}))
payload_header = p32(len(payload))
payload = payload_header + payload
ping_packet = p32(len(payload)) + payload

# send our crafted packet
wrapped_socket.send(ping_packet)

# get and decode response
response = b""
response += wrapped_socket.recv(1280)
response += wrapped_socket.recv(1280)
response += wrapped_socket.recv(1280)
response += wrapped_socket.recv(1280)
print("raw response:", response)
payload_size = int.from_bytes(response[:4], "little")
print("payload size:", payload_size)
cropped_payload = response[8:]
print("cropped payload (compressed):", cropped_payload)
payload_uncompressed = gzip.decompress(cropped_payload)
print("cropped payload (decompressed):", payload_uncompressed)
print("\n... and unpacked with msgpack:", msgpack.unpackb(payload_uncompressed))
```

This is the response we then get from an AsyncRAT server:

```
 raw response: b'%\x00\x00\x00\r\x00\x00\x00\x1f\x8b\x08\x00\x00\x00\x00\x00\x04\x00k\\\x16\x90\x98\x9c\x9dZ\xb2
payload size: 37
cropped payload (compressed): b'\x1f\x8b\x08\x00\x00\x00\x00\x00\x04\x00k\\\x16\x90\x98\x9c\x9dZ\xb2\xa4 ?/\x1d\
cropped payload (decompressed): b'\x81\xa6Packet\xa4pong'

... and unpacked with msgpack: {b'Packet': b'pong'}
```

We see, the server responds with a “Pong”-Packet and doesn’t care whether the initial “Ping” message came from an actual client beacon of an infected system or just some other random client. The same works for DcRAT/VenomRAT when we change `{b"Packet": b"Ping"}` to `{b"Pac_ket": b"Ping"}` in the code above. Also, by sending such a ping message, we remain completely unnoticed on the server-side since nothing w.r.t the ping message is logged there.

Note that through this approach, it is possible to identify these RATs also when e.g. non-default TLS certificates are used. This is the point where the other detectors, which only look at the ceritificates, fail.

By the way, the ability of sending valid C2 packets which are then regardlessly processed by the server, is not limited to just ping messages. Indeed, we can send arbitrary messages, as long as they are message types which are actually handled by the server. That means, depending on the available message types, we can cause some wild things happening on the server-side. And all that as a random client from outside.

Sending valid C2 packets that are blindly processed by the server is only possible for AsyncRAT/DcRAT/VenomRAT. QuasarRAT on the other side requires that the client is actually a valid client beacon. Nevertheless there is also another possibility to identify QuasarRAT: A QuasarRAT server doesn’t begin processing a packet sent by a client until 4 or more bytes are received since a QuasarRAT packet (inside TLS) always begins with 4 bytes indicating the size of the following payload. And when these 4 bytes don’t match the actual payload size, the server disconnects. Thus, a possible indicator for identifying a QuasarRAT server is to send 3 bytes of data to the server. When the connection keeps established and is closed by the server once a fourth byte is sent, the server might be QuasarRAT.

I combined this approach with other fingerprint- and ceritificate-based indicators to an overall detection tool which identifies QuasarRAT, AsyncRAT, DcRAT and VenomRAT servers. You can find it here: https://github.com/axmahr/QuasarRAT-Family-Detection

References

- C2 Intel Feeds
- A Beginner’s Guide to Tracking Malware Infrastructure
- Analysis - Identification of 64 Quasar Servers Using Shodan and Censys
- The State of SSL/TLS Certificate Usage in Malware C&C Communications
- QuasarRAT on Github
- AsyncRAT on Github
- DcRAT on Github
