PENTERA | Pentera Labs™ Research Series

The Gateway to Control: Initiating Attacks on MSSQL Databases

MSSQL is a common database developed by Microsoft. This research shows how to use its communication protocol - TDS - to detect the database version to increase the chance of a successful attack.

Amit German

Table of contents

03 Purpose

03 Executive summary

03 Does it apply to my organization?

03 Who should read this?

03 Intro

04 The TDS Protocol: Functionality and Structure

05 The structure of the Pre-Login Packet (with example data)

06 Sending a pre-login packet to the DB

06 Retrieving the Version

07 Mitigation

08 About the author

08 About Pentera

Purpose

To identify the version of an MSSQL database to allow targeting and implementation of known attacks or exploitation of pre-existing vulnerabilities.

Executive summary

MSSQL is a common database developed by Microsoft. This research shows how to use its communication protocol, TDS, to detect the database version without the need for authentication and increase the chance of a successful attack.

Does it apply to my organization?

This research is relevant to organizations with MSSQL databases in their networks, especially if they are exposed to the public internet.

Who should read this?

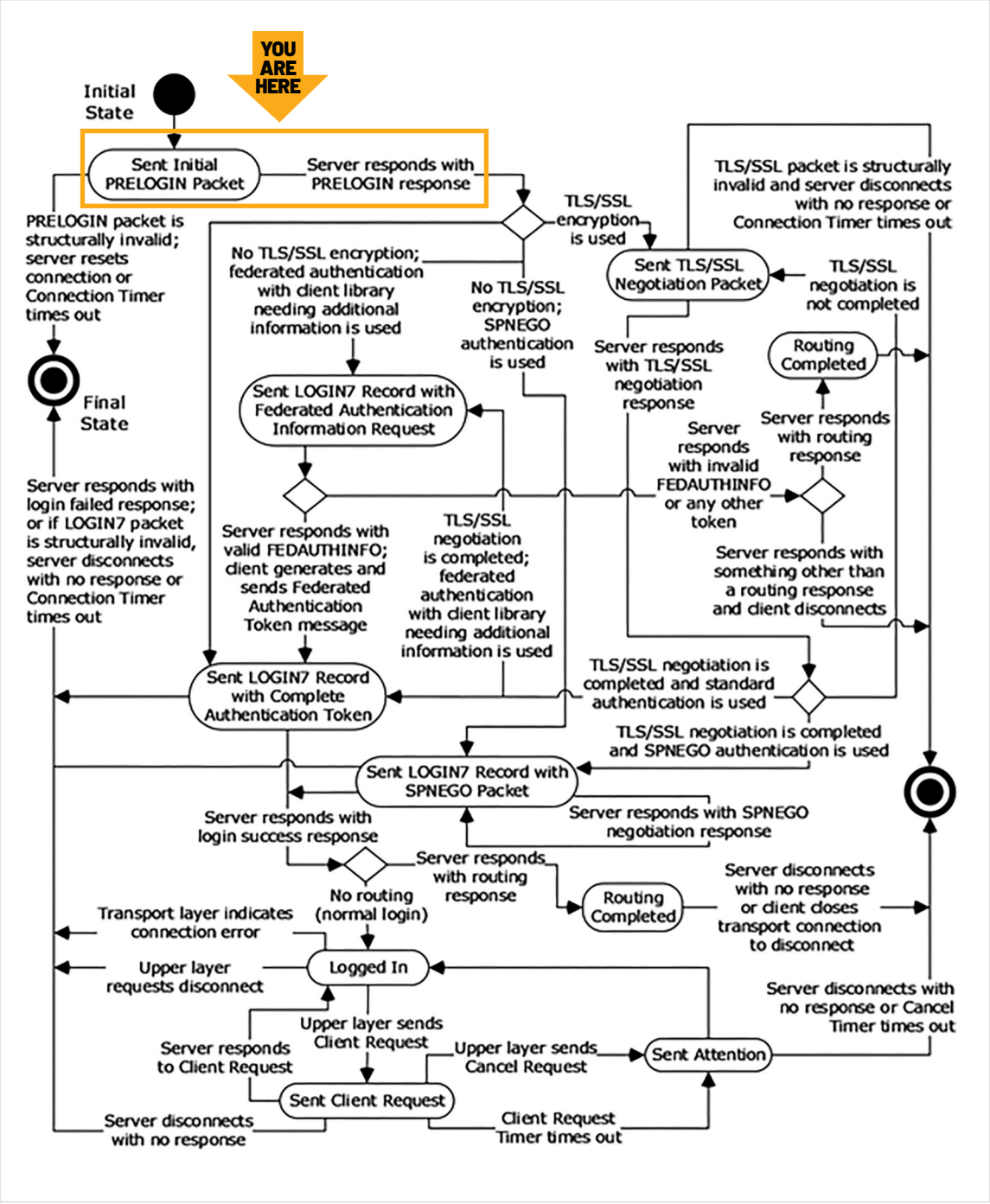
Cybersecurity researchers should read this research to become more familiar with the protocol. Pen testers can potentially use it in new attacks and defenders can make sure controls are in place to protect from attacks that use it.

Intro

MSSQL (Microsoft SQL Server) is a commonly used database developed by Microsoft.
The goal of our research was simple - to find a way to obtain an MSSQL database's version without the need to authenticate to the database. While conducting research, we discovered that Nmap, an open source security scanner, has a script named ms-sql-info that does the job effectively. There are two methods in Nmap that can be used to detect the version of MSSQL without requiring authentication: the TDS protocol and the SQL Browser Server. Of the two, the TDS protocol method is more accessible since it is available on the same port as the database itself. The SQL Browser Server, on the other hand, is accessible on a different port, decreasing the likelihood of this method being available. Therefore, we decided to break down the TDS method.

The TDS Protocol: Functionality and Structure

The Tabular Data Stream (TDS) protocol is an application layer protocol for communicating with databases - from authentication all the way to querying. In TDS, there is a sequence called "Pre-Login", which involves sending data to the server and receiving a response with information about the database. This information contains multiple tokens.



The TDS protocol pre-login sequence. A more detailed analysis of the pre-login packet can be found in Microsoft's documentation of ms-tds. Image is in use in accordance with Microsoft's copyright.

The first byte of every token is its type and the third byte is the offset to its data (from the beginning of the packet, including the header).

The structure of the Pre-Login Packet (with example data):

```
pre_login_header = [
    0x12, # Packet type - Pre-Login (18)
    0x01, # No more messages after this one
    0x00, 0x2F, # Packet length - 47
    0x00, 0x00, # SPID
    0x01, # PacketID - ignored
    0x00, # Window - ignored
    ]
```

The first 8 bytes represent the packet header

```
pre_login_data = [
    0x00, 0x00, 0x1A, 0x00, 0x06, # PL_OPTION_TOKEN - VERSION
    0x01, 0x00, 0x20, 0x00, 0x01, # PL_OPTION_TOKEN - ENCRYPTION
    0x02, 0x00, 0x21, 0x00, 0x01, # PL_OPTION_TOKEN - INSTOPT
    0x03, 0x00, 0x22, 0x00, 0x04, # PL_OPTION_TOKEN - THREADID
    0x04, 0x00, 0x26, 0x00, 0x01, # PL_OPTION_TOKEN - MARS
    0xFF, # PL_OPTION_TOKEN - TERMINATOR
    0x09, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0xB8, 0x0D, 0x00, 0x00, 0x01, # DATA
    ]
```

The rest of the bytes, as can be seen in the image above, contain tokens that provide information about various aspects of the database.

The token's data will always appear after the Terminator Token, marking the end of the token section. The Terminator's value is always 0xFF. Unlike the rest of the tokens, it contains no other data in it. You can see it in the sixth line of the pre_login_data section.

The relevant token in our context is the Version Token - whose type is 0x00. As mentioned above, it contains an offset to its data. But since the token's data is always present after a Terminator Token, it's very easy to search for a terminator byte and obtain the Version Token's data from there, since it appears first in the data section.

The Version Token's data received from the server consists of six bytes: major version (first byte, unsigned long), minor version (second byte, unsigned long), build (third and fourth bytes, unsigned short), and minor build (fifth and sixth bytes, always zero therefore redundant, unsigned short). Therefore, by retrieving the first four bytes of the Version Token (which are the first four bytes right after the Terminator Token), it is possible to obtain the database version.

Sending a pre-login packet to the DB

```
packet = []
packet.extend(pre_login_header)
packet.extend(pre_login_data)

packet_to_send = b''.join([struct.pack('B', val) for val in packet])
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    s.connect(("123.123.123.123", 1433))
except ConnectionError as e:
    print("Failed to establish a TCP connection")
else:
    s.sendall(packet_to_send)
    data = s.recv(500)
```

Once the data is sent to the server as part of the Pre-Login sequence, the response from the server will be received according to the same structure described above. After we obtain the first four bytes of the version (in one of the ways described above: using offset, or accessing it directly after the terminator token), we can easily obtain the database's version.

Retrieving the Version

Here's an example of the data sent back from the server with the six bytes of the version marked. Remember, only the first four are relevant.

```
[
    0x04, 0x01, 0x00, 0x2b, 0x00, 0x00, 0x01, 0x00,
    0x00, 0x00, 0x1a, 0x00, 0x06,
    0x01, 0x00, 0x20, 0x00, 0x01,
    0x02, 0x00, 0x21, 0x00, 0x01,
    0x03, 0x00, 0x22, 0x00, 0x00,
    0x04, 0x00, 0x22, 0x00, 0x01,
    0xff,
    0x0f, 0x00, 0x07, 0xd0, 0x00, 0x00, 0x01, 0x00, 0x01
]
```

Here's a simple code that can be used to extract the version from the four bytes we obtained:

```
data_no_header = data[8:] # Packet without the 8-byte header
terminator_index = data_no_header.find(0xff) # Find the terminator token's index
major = data_no_header[terminator_index + 1] # major version
minor = data_no_header[terminator_index + 2] # minor version
build = (data_no_header[terminator_index + 3] << 8) + data_no_header[terminator_index + 4] # build
version = f"{major}.{minor}.{build}.0"
```

The result we received is the database's version. In this case: 15.0.2000.0
It goes without saying that an attacker finding out a version of a product is never a good thing. The version information can be used to search for relevant one-days and generally gives the attacker more information about the organization.

Mitigation

Version detection is a common reconnaissance technique used by attackers to gather information about a target system. As shown in this article, MSSQL database version detection can be easily accomplished using the TDS protocol.

The version detection method using TDS is solely dependent on having the database's port open and accessible (1433 by default). To protect against this method of version detection, organizations must restrict access to the TDS port. This can be achieved by configuring rules to firewalls, iptables, and host-based firewalls. They should only allow access to servers that require it to function and block any external access.

Firewalls can be configured to limit traffic flow to the MSSQL server, ensuring that only authorized traffic is allowed through ports, reducing the attack surface of the organization. Host-based firewalls and iptables can also be used to restrict access to ports from within the server, providing an additional layer of protection.

By implementing the measures described above, organizations can significantly reduce the risk of unauthorized access to their MSSQL servers. To learn more about attack methods and how to mitigate them, you can visit Pentera Labs.

About the author

Amit German is a Security Researcher at Pentera, focusing on Windows Active Directory and web-based attacks

Reach out to us with any questions about the research at labs@pentera.io.

About Pentera

PENTERA

Pentera is the category leader for Automated Security Validation, allowing every organization to test with ease the integrity of all cybersecurity layers, unfolding true, current security exposures at any moment, at any scale. Thousands of security professionals and service providers around the world use Pentera to guide remediation and close security gaps before they are exploited.

For more info, visit: pentera.io

PENTERA | Pentera Labs™ Research Series

The Gateway to Control: Initiating Attacks on MSSQL Databases