DanaBleed: DanaBot C2 Server Memory Leak Bug

zscaler.com/blogs/security-research/danableed-danabot-c2-server-memory-leak-bug

Security Research

Introduction

DanaBot is a Malware-as-a-Service (MaaS) platform that has been active since 2018. DanaBot operates on an affiliate model, where the malware developer sells access to customers who then distribute and use the malware for activities like credential theft and banking fraud. The developer is responsible for creating the malware, maintaining the command-and-control (C2) infrastructure, and providing operational support. DanaBot has been involved in several high-profile campaigns, such as a supply chain attack on popular NPM packages and a Distributed-denial-of-Service (DDoS) attack against the Ukrainian Ministry of Defense during the 2022 Russian invasion. In May 2025, as part of a continued effort in Operation Endgame, law enforcement dismantled DanaBot's infrastructure and indicted 16 individuals affiliated with the group.

One aspect of DanaBot that has not been publicly known until recently is a bug in DanaBot's C2 server that introduced a memory leak. In June 2022, a then new version of the DanaBot malware (version 2380) was identified in-the-wild by ThreatLabz. This update introduced changes to the C2 protocol, one of which inadvertently caused the C2 server to leak snippets of its process memory in responses to infected victims. The memory leak, comparable to the Heartbleed vulnerability of 2014, offered those who discovered the bug (including ThreatLabz) to gain unique visibility into the internal operations of DanaBot. We have named this vulnerability *DanaBleed* and in this blog, we will present some of the sensitive information that we were able to recover from the memory leaks over nearly three years.

Key Takeaways

- DanaBot is a Malware-as-a-Service platform that emerged in 2018 with numerous capabilities to facilitate banking fraud, information theft, and provide remote access.
- The platform has been used for a variety of purposes from banking fraud to espionage. From June 2022 to early 2025, a programming error in the DanaBot command and control (C2) server caused a memory leak.
- Leaked information included: threat actor usernames, threat actor IP addresses, backend C2 server IP addresses and domains, infection and exfiltration statistics, malware version updates, private cryptographic keys, victim IP addresses, victim credentials, and other exfiltrated victim data.
- In May 2025, Operation Endgame dismantled DanaBot infrastructure and indicted 16 members affiliated with the group.

Technical Analysis

The DanaBleed memory leak began with the release of DanaBot version 2380 in June 2022 and continued until early 2025.

Analysis of the DanaBleed vulnerability

DanaBot is written in the Delphi programming language and uses a custom binary C2 protocol. A general overview of C2 requests prior to the June 2022 version update was the following:

1. Generate command data (e.g. key exchange, system information beacon, configuration file download, additional payload download, new C2 information, etc.)
2. Encrypt data with a session key
3. Encrypt session key
4. Generate a basic header
5. Send header and encrypted data

In June 2022, the malware developer introduced a new C2 protocol that modified the requests to perform the steps below:

1. Generate command data (e.g. key exchange, system information beacon, configuration file download, additional payload download, new C2 information, etc.)
2. Ostensibly append randomly generated bytes (although they were not random)
3. Encrypt data with a session key
4. Encrypt session key
5. Send encrypted data length and data

Responses from the C2 server to the victim were generated using the same logic and likely the same underlying code as the malware itself. This overlap allowed us to reverse engineer the vulnerability and make inferences about how the C2 server memory leak worked.

The figure below illustrates the changes to the C2 protocol introduced in the June 2022 update:

```
}
else if ( mode == 2 )                   // Mode 2 is a new addition to the C&C protocol.
{                                       // Command data is stored in a standard Delphi data structure known as a TMemoryStream.
  stream_size = (stream_command_data->vtable->GetSize)(stream_command_data);
  (stream_command_data->vtable->TCustomMemoryStream_Seek)(stream_command_data, stream_size, soBeginning);
  v27 = v9;
  s8.command_data_len = stream_command_data->vtable->GetSize(stream_command_data);// Initialize an 8-byte structure containing command data length and encoding type.
  s8.encoding = encoding;
  if ( (stream_command_data->vtable->GetSize)(stream_command_data) + 256 < 2048 )
  {
    System::Randomize();
    *range = 2048 - stream_command_data->vtable->GetSize(stream_command_data) - 256;// The maximum range can be 1792.
    *rand_num = System::Random(*range);   // Generate a random number within the range.
    stream_size_copy = stream_command_data->vtable->GetSize(stream_command_data);
    (stream_command_data->vtable->TMemoryStream_SetSize_0)(stream_command_data, (stream_size_copy + *rand_num));// Expand the size of the command data's TMemoryStream by the random number generated above.
                                        //
                                        // This is where the memory leak happens:
                                        //
                                        // When using Delphi's TMemoryStream SetSize() method, the size
                                        // of the TMemoryStream buffer is expanded, but the actual new
                                        // memory in the buffer is left uninitialized. The uninitialized
                                        // memory will contain arbitrary pieces of past heap memory.
    random_expanded_size = (stream_command_data->vtable->GetSize)(stream_command_data);
    (stream_command_data->vtable->TCustomMemoryStream_Seek)(stream_command_data, random_expanded_size, soBeginning);
    v25 = v11;
  }
  System::Classes::TStream::WriteBuffer(stream_command_data, &s8, 8);// Append the 8-byte structure to the command data TMemoryStream.
  if ( encrypt_data(command_data_len, stream_command_data) )// Encrypt the command data TMemoryStream.
  {
    *encrypted_command_data_len = stream_command_data->vtable->GetSize(stream_command_data);
    if ( send(sock, encrypted_command_data_len, 4, 0) == 4 )// Send the length of the encrypted command data.
    {
      if ( send_data(sock, stream_command_data) )// Send the encrypted command data.
        v34 = 1;
    }
  }
```

zscaler | ThreatLabz

Figure 1: Overview of C2 protocol changes introduced in DanaBot in the June 2022 update.

DanaBot's command data was stored in a Delphi TMemoryStream. A random number, capped at a maximum value of 1,792, was generated to determine the number of padding bytes to add to the command data buffer. While the size of the buffer was increased, the newly allocated memory within the buffer was *not* initialized. At first glance, this uninitialized memory appeared to be random, but closer inspection revealed that it contained arbitrary fragments of the C2 server's process memory. This oversight in memory handling created the DanaBot vulnerability that exposed the group's sensitive internal data.

Data exposed by the memory leak

The memory leak allowed up to 1,792 bytes per C2 server response to be exposed. The content of the leaked data was arbitrary and depended on the code being executed and the data being manipulated in the C2 server process at a given time. Despite this, our examination of the leaked data allowed us to extract meaningful insight into DanaBot for nearly three years.

Some of the most intriguing leaks revealed HTML snippets associated with the C2 server's web interface. The figure below, with highlights added, provides a sample of these leaked elements.

<h4>You_IP: </td><td><h4>81.8.233.252</td></tr><tr><td align=right><h4>Acc_ID:
</td><td><h4>U000005</td></tr><tr><td align=right><h4>Build_ID:
</td><td><h4>1132ED40DAFD67B2D9D73754DA097967</td></tr><tr><td align=right><h4>Drop_ID:
</td><td><h4>C99D71EBF95F752D1FD6FA9711196BF3</td></tr><tr><td align=right><h4>Key_ID:
</td><td><h4>FC62F86193916F7FE477DCEE59B59318</td></tr><tr><td align=right><h4>Write_Mode:
</td><td><h4>1</td></tr><tr><td align=right><h4>Inject_Mode: </td><td><h4>1</td></tr><tr><td
align=rig

---

ign=left size=2></td></tr><tr><td align=right><h4>User:
</td><td><h4>oracle@localhost</td></tr><tr><td align=right><h4>Domain:
</td><td><h4>ockiwumgv77jgrppj4na362q4z6flsm3uno5td423jj4lj2f2meqt6ad.onion:443</td></tr><tr><td
align=right><h4>Administration IP: </td><td><h4>188.92.79.117:35477</td></tr><tr><td
align=right><h4>You_IP: </td><td><h4>145.239.5.30</td></tr><tr><td align=right><h4>Acc_ID:
</td><td><h4>U000005</td></tr><tr><td align=ri

---

at><h5>Records: </td><td><h5>497 117</td></tr><tr><td align=right><h5>First Date:
</td><td><h5>2022-2-18</td></tr><tr><td align=right><h5>Last Date:
</td><td><h5>2022-6-27</td></tr><tr><td colspan=15><hr align=left size=2></td></tr></table><table
border=0><tr><td valign=top align=right><table

/tr><tr><td align=right><h4>Full_Threading_Created: </td><td><h4>489</td></tr><tr><td
align=right><h4>Posit_Upload: </td><td><h4>0</td></tr><tr><td colspan=2><hr align=left
size=2></td></tr><tr><td align=right><h4>User: </td><td><h4>pin@localhost</td></tr><tr><td
align=right><

---

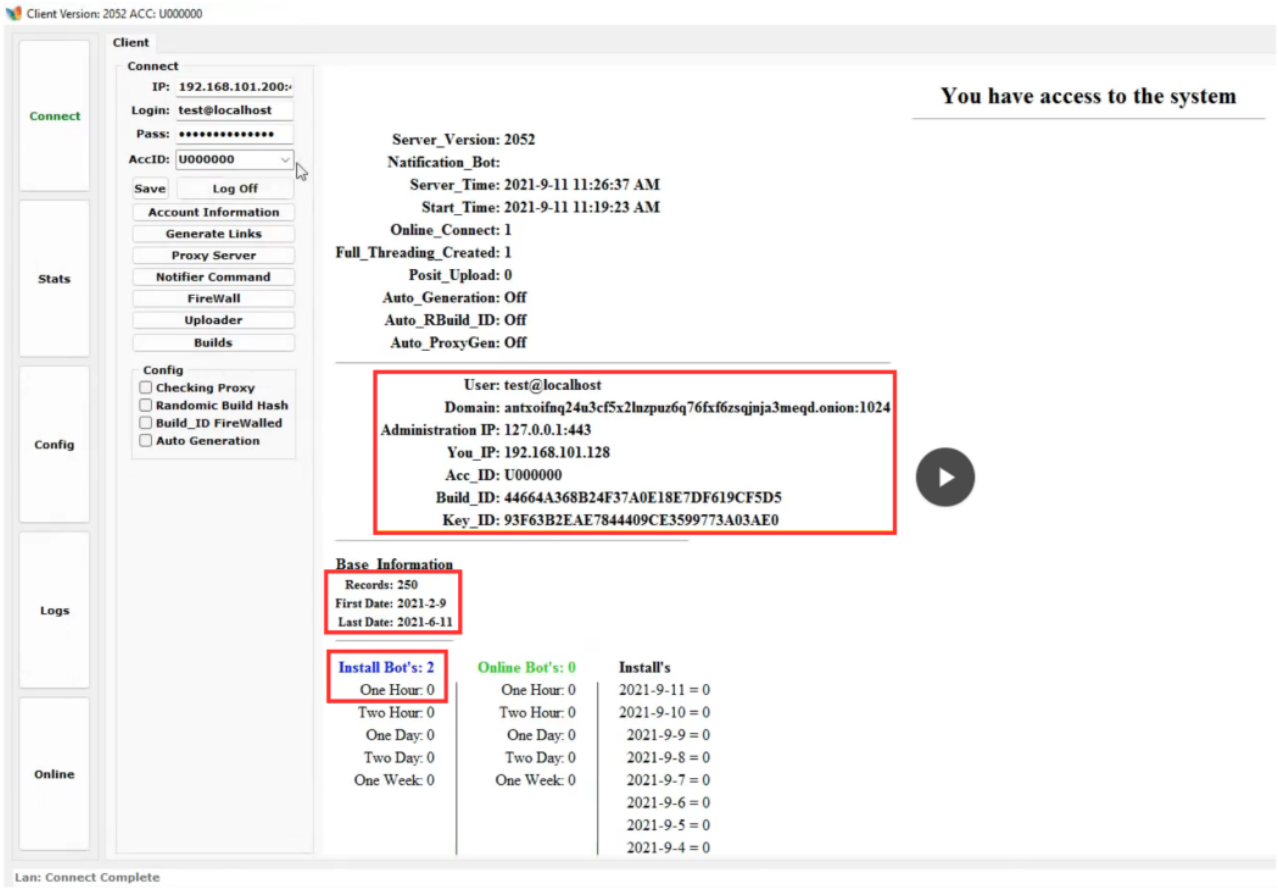
lign=top align=right><table border=0><tr><td><h4><font color=#0500fe>Install Bot's:
</td><td><h4><font color=#0500fe>110 422</td></tr><tr><td align=right>One Hour:
</td><td>0</td></tr><tr><td align=rig

=2&botid=">https://23.254.253.43:443/accid=5&key=FC62F86193916F7FE477DCEE59B59318&command=2&b
otid=</td><td></td></tr><tr><td><h5>Socks </td><td

zscaler | ThreatLabz

Figure 2: Example of leaked HTML code from DanaBot's C2 server.

These HTML snippets can be compared to the figure below (highlights added) which includes a screenshot from a video advertising DanaBot.



zscaler | ThreatLabz

Figure 3: Screenshot from a DanaBot advertisement video with content similar to the data observed in C2 server memory leaks.

The memory leak exposed sensitive data including:

- Threat actor usernames and IP addresses
- Backend C2 server IP addresses and domains
- Infection and exfiltration statistics
- Malware version update information
- Private cryptographic keys
- Victim-related data, such as IP addresses, credentials, and exfiltrated information

The DanaBot developer maintained a changelog of updates and some of those changes were also leaked, as shown in the figure below (highlights added).

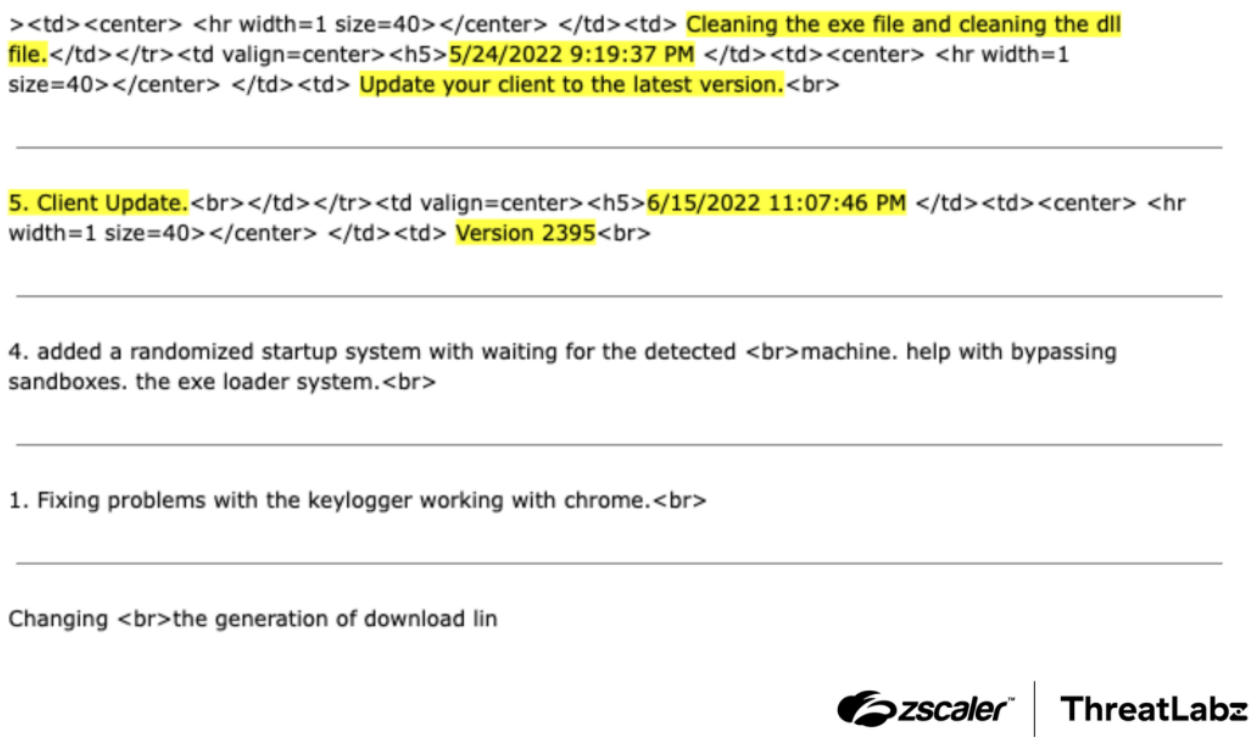


Figure 4: Sample change log discovered in DanaBot C2 server memory leaks.

In addition to HTML snippets, the memory leak also exposed debug information, including pathnames and logging messages. These are demonstrated in the figure below.

Figure 5: Sample debug information identified in DanaBot C2 server memory leaks.

Another frequent type of leak involved SQL statements. These leaks offered valuable insights into the C2 server's database structure, including information such as malware MD5 hashes, version updates, and victim IP addresses. The figure below (with highlights added) provides an example of these leaks.

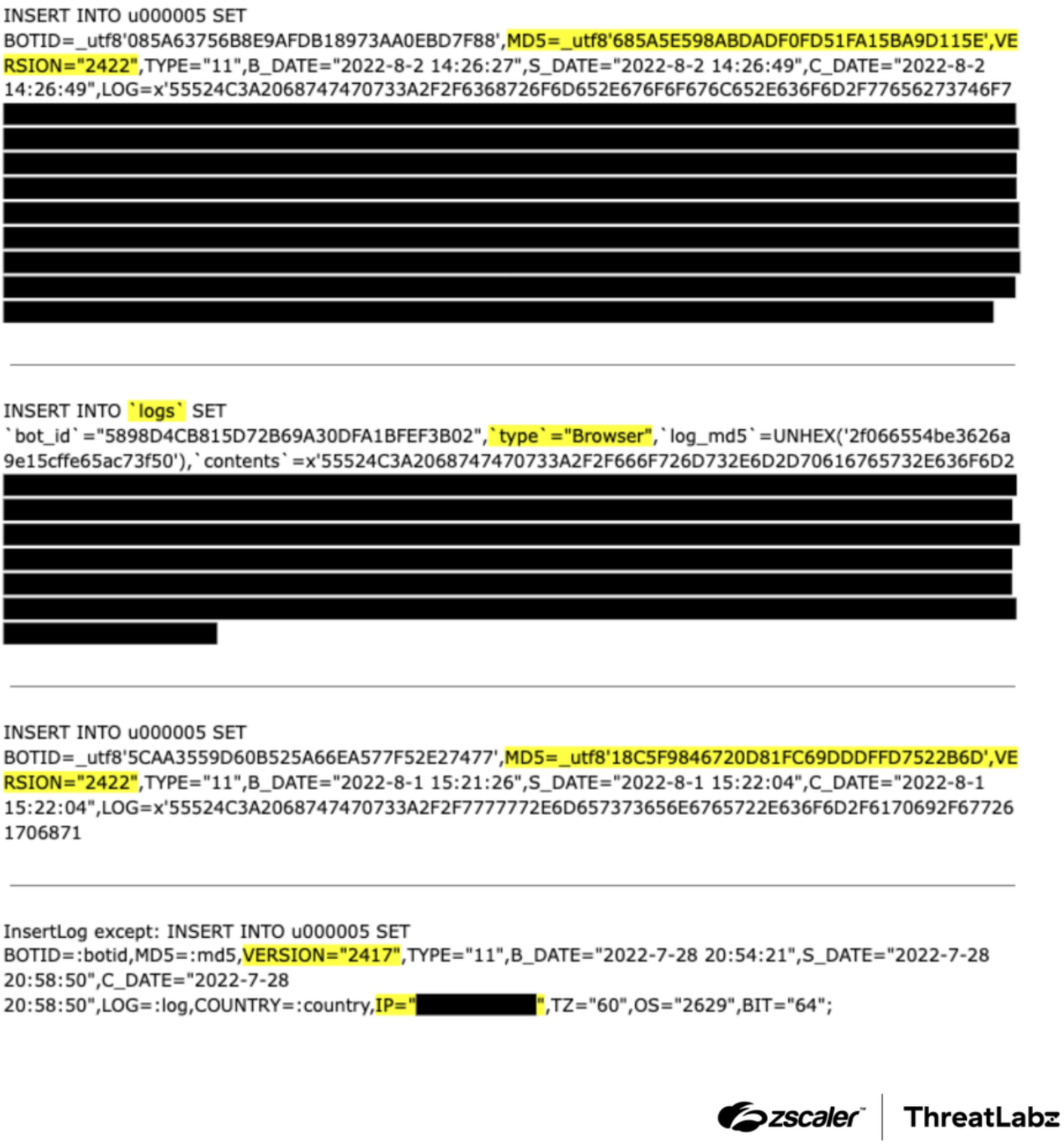


Figure 6: Sample SQL statement leak by DanaBot's C2 server.

The memory leaks also exposed private cryptographic key material, as shown in the figure below (highlight added):

Figure 7: Sample private key material leaks.

Finally, as DanaBot primarily functioned as an information stealer, the memory leak also exposed a significant amount of victim credentials and other exfiltrated data.

Conclusion

The memory leak discovered in the June 2022 update of DanaBot's C2 server protocol gave ThreatLabz and other researchers a glimpse into the inner workings of DanaBot. By analyzing the leaks from uninitialized C2 server memory over time, we gained valuable insights into the infrastructure, processes, and threat actors behind DanaBot. The leaked information revealed everything from backend server data, debugging logs, SQL statements, and cryptographic key material to sensitive victim data and elements of the C2 server's web interface. It is too soon to determine the impacts that Operation Endgame will have on DanaBot in the long term, but ThreatLabz will continue to track the activities of the group and their affiliates if they reemerge.

Zscaler Coverage

Zscaler's multilayered cloud security platform detects indicators related to DanaBot at various levels. The figure below depicts the Zscaler Cloud Sandbox, showing detection details for DanaBot.

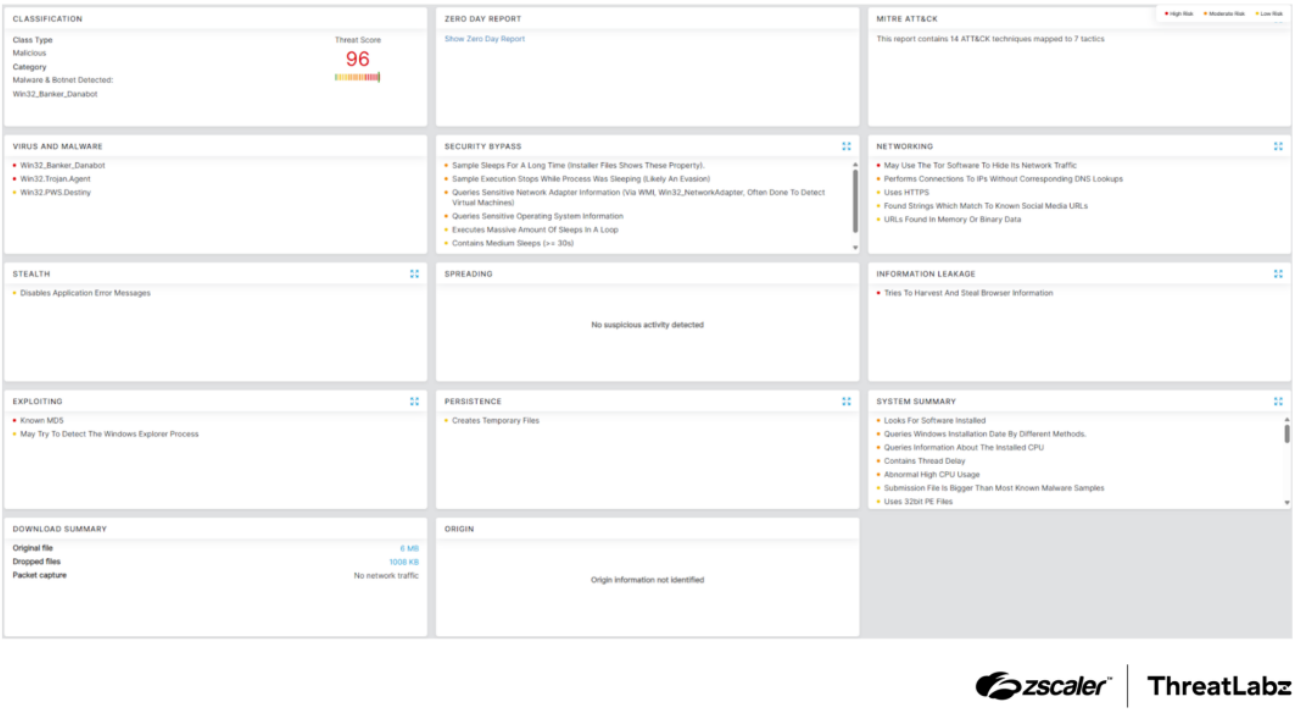


Figure 8: Zscaler Cloud Sandbox report for DanaBot.

In addition to sandbox detections, Zscaler's multilayered cloud security platform detects indicators related to DanaBot at various levels with the following threat names:

- Win32.Downloader.Danabot
- Win32.Banker.Danabot

Indicators Of Compromise (IOCs)

IOC	Notes
3ce09a0cc03dcf3016c21979b10bc3bfc61a7ba3f582e2838a78f0ccd3556555	SHA256 hash of DanaBot version 2380 main component

IOC	Notes
ae5eaeb93764bf4ac7abafeb7082a14682c10a15d825d3b76128f63e0aa6ceb9	SHA256 hash of DanaBot version 4006

Thank you for reading

Was this post useful?

Yes, very!Not really

Get the latest Zscaler blog updates in your inbox