Let’s set ice on fire: Hunting and detecting IcedID infections

By Deutsche Telekom AG

Published: 2021-05-17 · Archived: 2026-04-05 13:16:25 UTC

With the fall of the infamous Emotet botnet earlier this year, there are several cybercrime gangs competing to fill out the void that it left behind. One of these malware families that positions itself as a replacement is IcedID.

> Learn how to hunt for IcedID samples and detect local infections.

IcedID - also known as BokBot - emerged in late 2017. First, it served as a banking Trojan, which was common for this time. However, banking Trojans already faced several difficulties like two-factor authentication (2FA) introduced by many banks. Consequently, the IcedID developers transformed it into a downloader and it became part of the Malware-as-a-Service ecosystem (MaaS).

As of 2021, more and more ransomware operators use the service provided by IcedID’s operators. Telekom Security as well as others observed ransomware deployment after IcedID infections. This is in line with the general trend towards human-operated ransomware operations.

In this blog post, I will present ways how to hunt for IcedID samples and detect local IcedID infections. This allows on one side blue teamers to proactively find latent infections that could turn into ransomware deployment and on the other side incident responders to quickly find a patient zero during an investigation. In addition to this blog post, I recommend to read this technical analysis of IcedID core and a recent technical analysis of the initial IcedID GZIP loader. All scripts, YARA signatures and, additional IoCs mentioned in this blog post can be found in our Github repository.

Our Incident Response Service at Deutsche Telekom Security GmbH can quickly investigate and remediate ongoing IcedID-related intrusions. Please contact security-info@t-systems.com for more information.

IcedID infection chain overview

IcedID infection chain is quite complex when compared to other common malware families. The initial infection vector is email. Over the last months, several threat actors delivered spam that led to IcedID infections. This includes the Qakbot affiliate “TR” and TA551 / Shatak. Attachment types are as of now malicious Word / Excel documents or zipped JavaScript files.

> Figure 1 Example maldoc delivering IcedID requesting the user to activate macros

In the case of a malicious document, the infection chain goes on as follows. After the user has enabled the macro in the maldoc, the initial IcedID GZIP loader is fetched. This is a binary code component deployed as DLL. It is run using a command line similar to “rundll32.exe DLL_FILENAME,DllRegisterServer”. This DLL with internal name “loader_dll_64.dll” conducts an initial reconnaissance and fingerprints the target system. It sends this data to its command and control server (CC) and the CC may respond with a fake GZIP payload.

The GZIP loader decrypts this payload, which contains information on how to run the payload (e.g. command line to execute), another loader with internal name “sadl_64.dll”, and another encrypted payload currently with filename “license.dat”, which is the IcedID core module with internal name “fixed_loader64.dll”. The DLL with internal name “sadl_64.dll” is run with, for instance, with rundll32 using a similar command line to “rundll32.exe DLL_FILENAME,update /i:"FOLDER_NAME\license.dat"”, where in our case DLL_FILENAME was “Haimaw2.dll” and FOLDER_NAME was “GlacePlay”.

It decrypts the encrypted IcedID core module license.dat, loads it, and executes it. Noteworthy is that the core “fixed_loader64.dll” does not comprise configuration parameters like domain names of CC servers. This information is stored in the core loader “sadl_64.dll” and loaded by the core “fixed_loader64.dll” upon execution.

Figure 2 Decrypted configuration from "sadl_64.dll" decrypted by IcedID core

Hunting for IcedID samples

Proactively tracking adversaries helps us to follow their recent changes of their tooling and to improve their detection. As I’ve described in the last section, there are several stages involved in IcedID’s complex infection chain. Hence, there are several (binary) tools that are worth to hunt for.

Even though the developers of IcedID chose to encrypt strings (as described here), there are still many plain text strings allowing high confidence detection. This includes the internal project names.

Figure 3 Plain text string of the internal project name "sadl_64.dll"

In our Github repository, you can find several YARA rules to detect all binary code stages of IcedID:

- IcedID GZIP loader (“loader_dll_64.dll”)
- IcedID core module loader (“sadl_64.dll”)
- IcedID core module (“fixed_loader64.dll”)

Note that the string encryption was slightly changed in recent versions. The following screenshot shows the decompiled string decryption function. An encrypted string is prepended with its size, conceptually similar to Pascal-type strings. However, the first two WORDs have to be XORed in order to get the string size. Subsequently, the string is decrypted character after character in a for loop. In each round of the loop, first, a new key is generated based on a custom random function using x64 assembly ROR and ROL operations. Next, the character is XORed with the round’s new key.

Figure 4 Decompiled string decryption function

Detecting IcedID infections

Apart from using YARA rules to perform, for instance, memory scans, another way to detect IcedID infections reliably is searching for the registry keys it creates. The names of these registry keys are account-specific since they are derived from the bot ID, which is derived from the current user’s SID.

I implemented the bot ID computation and the derivation of the registry keys names in Python. They are also provided in our Github repository.

Computing the Bot ID

One of the first things that IcedID's core module does is computing a bot ID. This bot ID is used heavily across the code base, e.g. it is also send to the CC server as a way to identify bots and as we later will see to derive account-specific registry key names.

In a nutshell, the function "compute_bot_id" (see next screenshot) derives the bot ID from the currently logged-in account's SID (security identifier, e.g. "S-1-5-21-1984500107-304187221-49949575"). If the bot is not able to acquire this SID, it defaults to the MachineGuid. In the following, I assume that the bot can acquire the account's SID. It hashes the SID string using the FNV hash (as described here). The resulting hash value is XORed with a constant. The function returns a DWORD value.

Later, this DWORD value is negated using the x64 assembly instruction "not". For instance, the SID "S-1-5-21-1984500107-304187221-49949575" yields the bot ID 0x9c80033f and the negated bot ID 0x637ffcc0. Curiously, both bot IDs are used across the code base. However, the negated bot ID has the lion's share of the usages.

Figure 5 Decompiled function for computing the bot ID

Deriving Machine-Specific Registry Key Names

IcedID's core stores configuration information (e.g. domain names) in account-specific registry keys. All of them are subkeys of "Software\\Classes\\CLSID\\". The registry key names are derived from hard-coded GUIDs and the account-specific bot ID. Hence, they are also account-specific.

Once we know how to derive the registry keys, we can reliably detect IcedID infections for an account. Blue teamers can use this knowledge to proactively scan for infections and incident responders to find a patient zero during an IcedID-related investigation.

Recent samples derived their registry key names from the following hard-coded GUIDs:

- {0ccac395-7d1d-4641-913a-7558812ddea2}
- {d65f4087-1de4-4175-bbc8-f27a1d070723}
- {e3f38493-f850-4c6e-a48e-1b5c1f4dd35f}

To derive a registry key name from one of the hard-coded GUIDs, IcedID computes a simple hash of the GUID. Each character is rotated to the right by 13/0xD. This results in a DWORD value. This is then XORed with the negated bot ID. Next, it hashes the hardcoded GUID as well as the above DWORD value using the MD5 hash sum. This results in a value of 16 bytes. Finally, it formats these 16 bytes as a GUID.

For instance, for the above SID and bot IDs, we can derive the registry key name "{404FE54D-A5F1-3480-D7E1-2463C1FC62FD}" from the input "{0ccac395-7d1d-4641-913a-7558812ddea2}".

Figure 6 Decompiled function of registry key derivation

Conclusion

IcedID is one of the uptrending malware families that tries to fill the void that Emotet left behind. Initially a banking Trojan, it soon became a downloader. Recently, more and more ransomware deployments are associated with initial IcedID infections. It's rather complex infection chain increases analysis difficulty and ensures that an easy IoC extraction with off-the-shelf sandbox systems is not possible.

In this blog post, I've given a quick overview of IcedID current infection chain, focusing on the binary payloads and showed how to detect IcedID infections based on its use of the registry. In a nutshell, each bot computes a bot ID based on the current user's SID. This is utilized to as a seed to derive a handful of account-specific registry key names, where it stores configuration parameters like CC domains. The blog post's companion YARA rules, Python scripts, and additional IoCs can be found in our repository on Github.

Appendix: IoCs

IoC	Description
fc148647bf11e143f44e89ba3b229aca	GZIP loader dump
acc57b939ac8bb9bd4bf18f76e779977	GZIP loader
92fdfe61cf977a7e5bed5ceeabdf895f	Core loader dump
11965662e146d97d3fa3288e119aefb2	IcedID core unpacked
ameripermanentno[.]website	IcedID core CC server
odichaly[.]space	IcedID core CC server

Further information:

IcedID (Malware Family) – Malpedia

Technical Analysis of IcedID