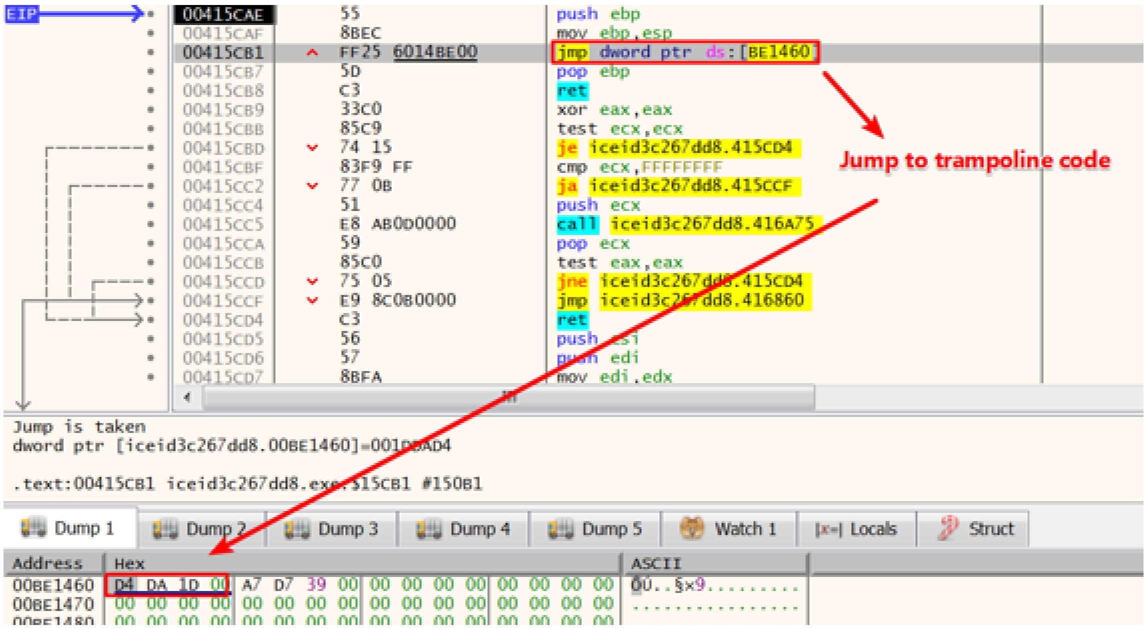
# A Deep Dive Into IcedID Malware: Part I - Unpacking, Hooking and Process Injection

fortinet.com/blog/threat-research/icedid-malware-analysis-part-one.html

July 9, 2019




*FortiGuard Labs Threat Analysis Report Series*

IcedID is a banking trojan which performs web injection on browsers and acts as proxy to inspect and manipulate traffic. It steals information, such as credentials, from victims. It then sends that stolen information to a remote server.

Recently, the FortiGuard Labs team started to investigate some IcedID samples. In this series of blogs, I will provide a detailed analysis of a new IcedID malware sample. The entire detailed analysis is divided into three parts.

This blog is Part I below. Let's dive in.

## 0x01 Malicious PE Executable

The sample being analyzed is a PE executable, and is most commonly distributed by a compromised Office file. The following image is the process tree after executing the PE file. We can see that this sample of IcedID eventually creates a svchost.exe parent process and three svchost.exe child processes. In addition, it can deliver a Trickbot payload, highlighted in red. In this series of blogs, the analysis of the Trickbot payload won't be covered. We will only focus on how IcedID works internally.

Figure 1. The process tree after executing the IcedID sample

As shown in Figure 1, the PE executable first launches itself with a command line parameter “-q=xxxxxxxxxx”. This new process then continues by launching a svchost.exe process. Once the first svchost.exe process is launched, the previous two processes exit. Finally, this svchost.exe parent process then launches three svchost.exe processes.

## 0x02 Unpacking PE Executable

We can now start to dynamically analyze the PE execution. After tracing a few steps from the entry point, the program goes into the function sub_00415CAE() as follows.

Figure 2. Jump to the trampoline code

In the trampoline code, it is used for decrypting the code segment. Eventually, it can jump to the real entry point of the program. At that point, the unpacking of the PE executable is complete.

Figure 3. Jump back to the real entry point 0x401000

The following is the pseudo code of the real entry point of the program.

Figure 4. The pseudo code of the real entry point

Here is a list of the key functions:

1. Check if the command line parameter starts with “-q=”. If yes, it jumps to step 2. Otherwise, it jumps to step 3.
2. Create the svchost.exe process and perform process injection.
3. Create a new process with a TSC parameter (“-q=xxxxxxxxx”).

We ran this sample without any parameters so it could go into the third step (sub_4012E9).

Figure 5. The function sub_4012E9()

After performing the ***rdtsc*** instruction, the return value is converted into a string as a parameter of the new process execution. Next, the program sets an environment variable in the process context. The name of the variable is the command line parameter without the prefix “-q=”.

Figure 6. Set an environment variable in process context

Finally, it invokes the CreateProcessA function to create itself with a parameter.

Next, we will continue the analysis with the new running process.

## 0x03 Hooking Technique and Process Injection

After launching the new process, the program goes to the real entry point of the program, as shown in Figure 4. At this point, the check_parameter() function returns TRUE because the command line parameter starts with “-q=”. It then goes to the sub_40124A() function.

Figure 7. The pseudo code of sub_40124A()

In the function hook_NtCreateUserProcess(), it first invokes the function NtProtectVirtualMemory to change the protection of the first five bytes of the function NtCreateUserProcess to PAGE_EXECUTE_READWRITE. It then modifies those five bytes with a JMP instruction. Finally, it again invokes the function NtProtectVirtualMemory to restore protection to the first five bytes.

Figure 8. Hooking the function NtCreateUserProcess

The following is the assembly code of the function NtCreateUserProcess hooked.

Figure 9. The assembly code of the function NtCreateUserProcess hooked

Inside the function CreateProcessA, the code invokes the low-level API NtCreateUserProcess. After the function CreateProcessA is invoked in Figure 7, the program goes to the trampoline code sub_4010B7(). The following is the pseudo code of the trampoline code.

Figure 10. The trampoline pseudo code of NtCreateUserProcess hooked

The following list is what the trampoline code actually does.

1. Unhooks the function NtCreateUserProcess.
2. Calls the function NtCreateUserProcess, which performs the main work of creating a new process.
3. Decompresses the buffer using RtlDecompressBuffer.
4. Performs process injection into the svchost.exe process and hooks RtlExitUserProcess in the process space of svchost.exe.

Let’s take a closer look at step four. The following is the pseudo code of the function sub_401745() in that step.

Figure 11. Perform process injection into the svchost.exe process and hook its RtlExitUserProcess

It first uses NtAllocateVirtualMemory to allocate the memory region in the remote process space(svchost.exe). Next, it uses ZwWriteVirutalMemory to perform the code injection into the memory region in the svchost.exe process.

Figure 12. Process injection into in svchost.exe process

It then sets up a hook for the RtlExitUserProcess API in the process space of svchost.exe. It should be noted that there is a little difference between hooking RtlExitUserProcess and hooking NtCreateUserProcess in Figure 8. The former is to hook the API of remote process space, while the latter is to hook the API in its current process space.

Figure 13. Hook RtlExitUserProcess

The assembly code of the hooked RtlExitUserProcess is shown in Figure 14.

Figure 14. The hooked RtlExitUserProcess in svchost.exe process

As shown in Figure 7, the process svchost.exe was created without a parameter. It could immediately exit if running svchost.exe without parameter, and after it exits, it could invoke the low-level API RtlExitUserProcess. Because IcedID hooks the RtlExitUserProcess, it could jump to the trampoline code to execute the IcedID payload.

The injected memory regions in the remote process svchost.ext are shown in Figure 15. We can see that two memory regions have been injected. The code segment is stored in the memory region(0xa1000 ~ 0xa7000).

Figure 15. The injected memory regions of svchost.exe process

As shown in Figure 14, it jumps to 0xA2B2D, which is in memory region(0xA0000 ~ 0xAC000). The offset of the trampoline code from this memory region is **0x2B2D**.

## 0x03 Conclusion

We have walked through how to unpack the IcedID malware, hooking, and process injection techniques used by IcedID, as well as how to execute the IcedID payload. In the next blog, I will provide a deep analysis of the IcedID payload (0xA2B2D).

## 0x04 Solution

This malicious PE file has been detected as “W32/Kryptik.GTSU!tr” by the FortiGuard AntiVirus service.

## 0x05 Reference

SHA256 Hash:

PE executable (b8113a604e6c190bbd8b687fd2ba7386d4d98234f5138a71bcf15f0a3c812e91)

*Learn more about FortiGuard Labs and the FortiGuard Security Services portfolio. Sign up for our weekly FortiGuard Threat Brief.*

*Read about the FortiGuard Security Rating Service, which provides security audits and best practices.*