Dridex's Cold War: Enter AtomBombing

securityintelligence.com/dridexs-cold-war-enter-atombombing/

February 28, 2017

Advanced Threats February 28, 2017

By Magal Baz co-authored by Or Safran 9 min read
IBM X-Force discovered that Dridex, one of the most nefarious banking Trojans active in the financial cybercrime arena, recently underwent a major version upgrade that is already active in online banking attacks in Europe.

A few weeks ago, our cybercrime labs detected a new major version of the Dridex banking Trojan, Dridex v4. The updated code features a new and innovative injection method based on a technique dubbed AtomBombing, which was first disclosed in October 2016 by security firm enSilo.

Dridex is the only banking Trojan we have encountered to use AtomBombing. This change is especially significant when it involves Trojans believed to be operated by an organized cybercrime gang because it's likely to result in other codes adopting the same method in the future.

Dridex's developers also worked on a major upgrade to the malware's configuration encryption. This upgrade includes implementing a modified naming algorithm, a robust but easy-to-spot persistence mechanism and a few additional enhancements.

According to IBM Security detection, Dridex v4 is already out and active in campaigns that mostly target UK banks. Dridex attacks on online banking users in the UK are based on its hVNC RAT capabilities and redirection attack scheme, which appears to have replaced the webinjects method as Dridex's top M.O.

A Major Version Is a Major Deal

Dridex's code is based on that of the Bugat Trojan, which was first discovered in early 2010. Bugat has since evolved into a number of different variations, including Cridex and Feodo. The Dridex form first appeared in 2014.

When it comes to their development cycles, Dridex's authors release minor versions quite often, and a major version much less frequently. The releases of the minor and major versions are easy to spot since Dridex's developers clearly track their releases.

Dridex's build numbers are found inside its configuration and in the binary's code.

Figure 1: Dridex's code version hard-coded into the binary.

When it comes to major versions for Dridex, the most stable and resilient version to date has been v3, which was released in April 2015 and has been used in all known attack campaigns up until the v4 release.

Dridex v2 did not enjoy the same kind of longevity and was only active in early 2015. The first version, released in late 2014, lasted only until the beginning of 2015.

The release of a major version upgrade is a big deal for any software, and the same goes for malware. The significance of this upgrade is that Dridex continues to evolve in sophistication, investing in further efforts to evade security and enhance its capabilities to enable financial fraud.

Read the white paper: How to outsmart Fraudsters with Cognitive Fraud Detection

How 'Bout That Code Injection?

When it comes to malware detection, the injection of malicious code from one process into another is one of the most popular events antivirus and other security solutions aim to monitor and use as an indication of compromise.

Most financial malware uses three steps to inject code: remote memory allocation, remote writing of a payload into the allocated memory and, finally, remote execution of the payload. Malware commonly uses the CreateRemoteThread method to execute a payload or to call LoadLibrary, typically using the following application program interface (API) calls:

- VirtualAllocEx to allocate a buffer in the remote process with RWX permissions;
- WriteProcessMemory to copy the payload to the allocated buffer; and
- CreateRemoteThread to execute the payload.

The problem is that this is a very suspicious and obvious way to inject code. And since it's likely to raise red flags, malware authors would much rather use alternative methods.

Enter AtomBombing, but Only Halfway Through

In October 2016, enSilo researchers exposed a new code injection technique called AtomBombing, which allows the malware to inject code without making any of the aforementioned API calls.

Rather, AtomBombing makes use of Windows' atom tables and the native API NtQueueApcThread to copy a payload into a read-write (RW) memory space in the target process. It then uses NtSetContextThread to invoke a simple return-oriented programming (ROP) chain that allocates read/write/execute (RWX) memory, copies the payload into it and executes it. Finally, it restores the original context of the hijacked thread.

In our analysis of the new Dridex v4 release, we discovered that the malware's authors have devised their own injection method, using the first step of the AtomBombing technique. They use the atom tables and NtQueueAPCThread to copy a payload and an import table into a RW memory space in the target process. But they only went halfway — they used the AtomBombing technique for the writing of the payload, then used a different method to achieve execution permissions, and for the execution itself.

Nonetheless, this is the first implementation of AtomBombing within the context of banking Trojans, probably designed to help Dridex avoid detection.

Getting the Payload to the Target Process

This stage is almost identical to what is described in the EnSilo blog, so we'll explain it briefly here and focus on what's unique in Dridex's implementation of this method.

Throughout the injection flow, Dridex needs arbitrary code to be executed by a thread in a targeted process. To do that, it uses Windows asynchronous procedure calls (APC) by calling the NtQueueAPCThread API.

There's a reason here for using the native API instead of opting for kernel32!QueueUserAPC. It allows three parameters to be passed to the APC routine being called instead of just one parameter allowed by kernel32!QueueUserAPC.

During the flow of the malicious code injection, Dridex uses APC requests, for which it needs to find a thread in the target process that is in alertable state, meaning a thread that will actually execute APC calls in its queue.

To test whether a thread is alertable, Dridex creates an event and uses an APC request to get the thread to run NtSetEvent to it. It uses a loop to do that with several of the target process's threads, then calls WaitForMultipleObjects to select the first thread to signal its event.

Figure 2: The call to kernel32!CreateEventA.

Writing the Import Table

Next, Dridex gets the injected process to run memset, again via the NtQueueAPCThread API, to zero-out a RW memory space inside Ntdll's address space. It then starts writing the import table to the allocated memory to be used later by the payload.

The writing is done by first putting the data of the import table into an atom table using a call to GlobalAddAtomW. Next, the malware uses the NtQueueAPCThread API to get the target thread to call GlobalGetAtomW, which retrieves the data and places it in the RW memory space in Ntdll.

Figure 3: Copying the import table data (from injecting process).

Figure 4: The import table being built at injected process.

Writing the Payload and Then AtomBombing Out

After handling the import table, the payload is written to another RW address space using the same technique, only this time writing large chunks of payload code at each call to GlobalGetAtomW.

At this point, the flow differs from the one described in the AtomBombing technique. To get the payload into an executable memory space, Dridex simply calls NtProtectVirtualMemory from the injecting process to change the memory where the payload is already written into RWX. It's a simple fix and a small compromise for the sake of the overall technique, designed to avoid making suspicious API calls, which are usually monitored by security software.

Executing the Payload

After the preparation, the last stage is the execution of the payload. To avoid calling CreateRemoteThread, Dridex again uses APC. Using an APC call to the payload itself would be very suspicious, however, and could be detected and stopped. Instead, it uses the same GlobalGetAtomW method to patch GlobalGetAtomA, hooking it to execute the payload.

This action, of course, requires another call to NtProtectVirtualMemory to make GlobalGetAtomA writable. Dridex then proceeds to use the Windows APC to call GlobalGetAtomA, which executes the payload.

Figure 5: Changing protection of first 7 bytes of GlobalGetAtomNameA to RWX (0X40).

Figure 6: GlobalGetAtomNameA at injected process (permission changed and code patched to jump to payload).

Additional Enhancements in Dridex v4

Besides building out a new code injection method, Dridex v4 differs from the older versions in several other ways. Below are some of the enhancements made to the code in the new release.

A Modified Naming Algorithm

Dridex uses its own method to generate MD5 hashes using a string concatenation of the hostname, active user name, OS installation date and an added seed. These hashes are used for many purposes. They can be mutex names, event names, key names for configurations in the registry, RC4 keys and more.

In the new version, while the same variables are still being used to generate these hashes, the sequence has changed to shuffle things around and prevent detection by automated checks.

Enhanced Encryption for the Configuration

Dridex's configurations contain many details about its targets and the attack types used in each case. Dridex also serves up its redirection scheme from the configuration, which makes it a file it aims to protect.

In v4, Dridex's developers significantly upgraded the cryptographic protection for the configuration. Overall, Dridex continues to use the same multilayered approach it used in v3 variants, but it has changed and enhanced the encryption while still relying heavily on the RC4 cipher.

The unique binary format in which Dridex keeps its target list configurations, such as the URLs of targeted banks, has also received a cryptographic upgrade to keep targeted entities in the dark as much as possible.

Updated Persistence Mechanism

Up until the recent v3 build, Dridex used a very specific, invisible persistence mechanism. This method was called invisible because it was not present at all as long as the infected endpoint was up and running. Only before a shutdown of the operating system, Dridex's dynamic link library (DLL) would get written to disk, and a registry value (HKCU\Software\Microsoft\Windows\CurrentVersion\Run) was created to execute the malicious DLL upon reboot.

The method was completely abandoned in v4, and Dridex now uses a DLL-hijacking technique instead. An executable is copied from system32 into a different directory and Dridex's DLL is placed in that same directory. The malicious DLL mimics a legitimate DLL that's loaded by the executable.

According to Windows file path priority, the malicious DLL gets loaded instead of the original one, which is located in system32. These setups are placed in system32 and %AppData% folders and executed by registry run keys and scheduled tasks.

X-Force research noted that this method was already observed in some of the last v3 builds detected in the past few months, but v4 has fully adopted this robustness-over-stealth approach for its persistence mechanism.

Conclusion

The Dridex malware project continues to evolve, and 2017 is likely to be another year of change for this Trojan.

Over the long reign of Dridex v3, we have seen some significant changes implemented into the malware's operations, such as modified anti-research techniques, redirection attacks and fraudulent M.O. changes. It is not surprising to see a new major version released from this gang's developers.

In this release, we noted that special attention was given to dodging antivirus (AV) products and hindering research by adopting a series of enhanced anti-research and anti-AV capabilities.

The changes to Dridex's code injection method are among the most significant enhancements in v4. They allow Dridex to propagate in the infected endpoint with minimal calls to marked API functions.

The adoption of a new injection technique shortly after its discovery demonstrates Dridex's efforts to keep up with the times and the evolution of security controls. Although they relied on a publicized method, Dridex's developers created their own version of it, a choice that is consistent with their usual preference to write proprietary code schemes for Dridex, as they did for its binary configuration format, for example.

IOCs

In this research we analyzed Dridex sample MD5s 4599fca4b67c9c216c6dea42214fd1ce and 1e6c6123af04d972b61cd3cde5e0658e.

Dridex is featured in an X-Force Exchange collection, which is updated regularly. You can view the collection and contribute to it here.

IBM Security has a great deal of information on Dridex v4 and its attack schemes and can help banks and other targeted organizations learn more about this high-risk threat. To help stop threats like Dridex, banks and service providers can use adaptive solutions to detect infections and protect customer endpoints when malware evolves or enhances its focus on the organization's locale.

Fighting evolving threats such as Dridex attacks can be made easier with the right malware detection solutions. With protection layers designed to address the ever-changing threat landscape, financial organizations can benefit from malware intelligence that provides real-time insight into fraudster techniques and capabilities.

Consumers wishing to protect themselves from malware infections on endpoints and mobile devices are invited to read our best practices page.

Read the white paper: How to outsmart Fraudsters with Cognitive Fraud Detection

Magal Baz
Malware Researcher, IBM Trusteer

Magal Baz is a malware researcher for IBM Security's Trusteer's group. He has been a member of the Trusteer cybercrime labs for the past two years. Magal has...