FriedEx: BitPaymer ransomware the work of Dridex authors

welivesecurity.com/2018/01/26/friedex-bitpaymer-ransomware-work-dridex-authors/

January 26, 2018

ESET research has found that the ransomware FriedEx, also known as BitPaymer, is actually the work of the notorious gang responsible for the Dridex banking trojan.

Michal Poslušný
26 Jan 2018 - 02:57PM

ESET research has found that the ransomware FriedEx, also known as BitPaymer, is actually the work of the notorious gang responsible for the Dridex banking trojan.

Dridex has been a nightmare for computer users, companies and financial institutions for several years now, so much so that for many, it has become the first thing that comes to mind when talking about banking trojans.

Recent ESET research shows that the authors of the infamous Dridex banking trojan are also behind another high-profile malware family – a sophisticated ransomware detected by ESET products as Win32/Filecoder.FriedEx and Win64/Filecoder.FriedEx, and also known as BitPaymer.

Dridex

The Dridex banking trojan first appeared in 2014 as a relatively simple bot inspired by older projects, but the authors quickly turned this bot into one of the most sophisticated banking trojans on the market. The development seems to be steady, with new versions of the bot including minor fixes and updates being released on a weekly basis, with occasional breaks. From time to time, the authors introduce a major update that adds some crucial functionality or larger changes. The last major update from version 3 to version 4, released at the beginning of 2017, gained attention for adopting the Atom Bombing injection technique, and later in the year also introducing a new MS Word zero-day exploit, which helped spread the trojan to millions of victims.

As of this writing, the most recent version of Dridex is 4.80 and includes support for webinjects into Chrome version 63. Dridex 4.80 was released on December 14th 2017.

Note: Last year we released a tool that helps identify malicious hooks in popular web browsers. The tool is designed to help incident responders discover potential banking trojan infections, including Dridex.

FriedEx

Initially dubbed BitPaymer, based on text in its ransom demand web site, this ransomware was discovered in early July 2017 by Michael Gillespie. In August, it returned to the spotlight and made headlines by infecting NHS hospitals in Scotland.

FriedEx focuses on higher profile targets and companies rather than regular end users and is usually delivered via an RDP brute force attack. The ransomware encrypts each file with a randomly generated RC4 key, which is then encrypted using a hardcoded 1024-bit RSA public key and saved in the corresponding .readme_txt file.

In December 2017, we took a closer look at one of the FriedEx samples and almost instantly noticed the resemblance of the code to Dridex. Intrigued by the initial findings, we dug deep into the FriedEx samples, and found out that FriedEx uses the same techniques as Dridex to hide as much information as possible about its behavior.

It resolves all system API calls on the fly by searching for them by hash, stores all strings in encrypted form, looks up registry keys and values by hash, etc. The resulting binary is very low profile in terms of static features and it’s very hard to tell what the malware is doing without a deeper analysis.

This prompted yet further analysis, which revealed a number of additional attributes that confirmed our initial suspicions – the two malware families were created by the same developers.

Code Similarities

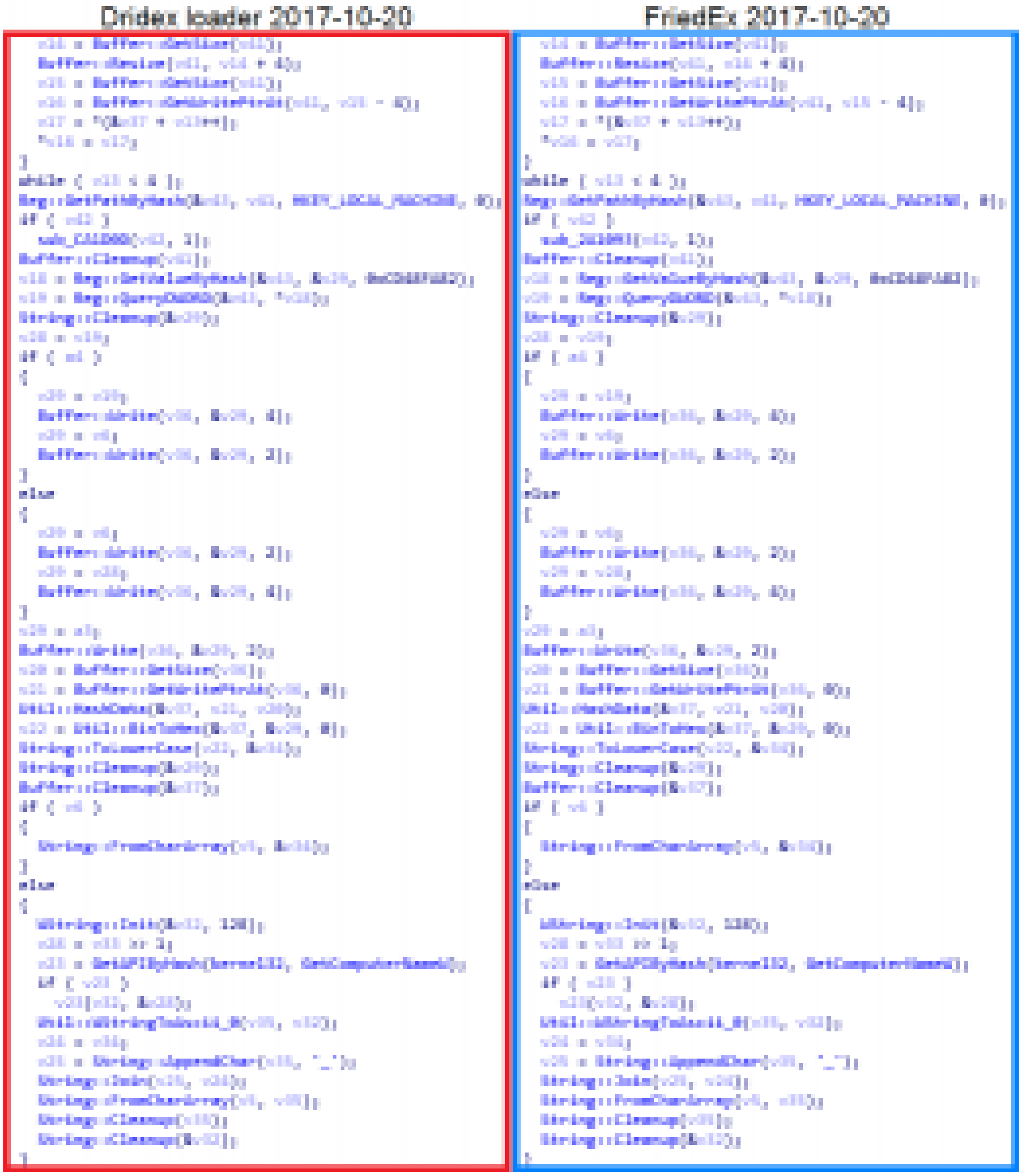
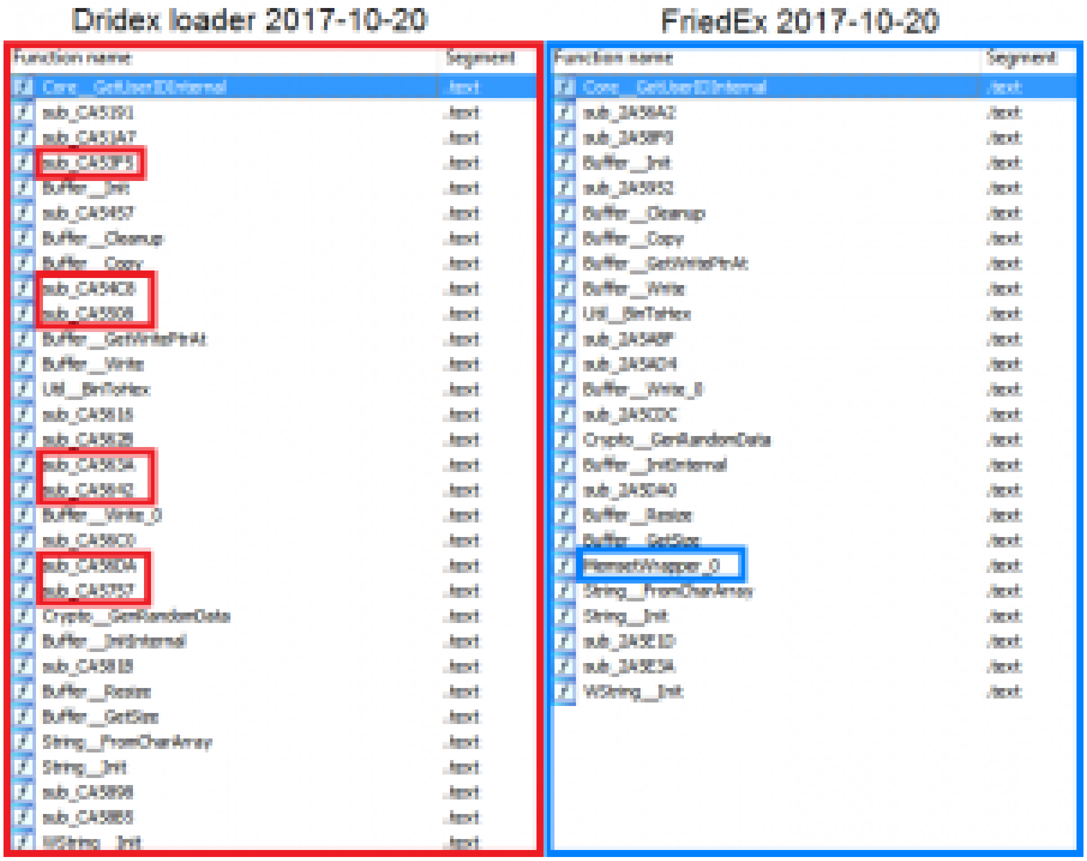


Figure 1. Comparison of GetUserID function present in both Dridex and FriedEx samples

In *Figure 1*, we can see a part of a function used for generating UserID that can be found across all Dridex binaries (both loaders and bot modules). As we can see, the very same Dridex-specific function is also used in the FriedEx binaries. The function produces the same results – it generates a string from several attributes of the victim's machine that serves as a unique identifier of the given victim, either in the botnet in the case of Dridex, or of the ransomware with FriedEx. Indeed, the screenshots would make for a good "Spot the difference" game!

This kind of similarity to Dridex is present throughout the FriedEx binaries and only very few functions that mostly correspond to the specific ransomware functionality are not found in the Dridex sample (i.e. the file encryption loop and creation of ransom message files).



Dridex loader 2017-10-20

Function name	Segment
Core__GetUserIDInternal	.text
sub_CA5191	.text
sub_CA51A7	.text
sub_CA53F5	.text
Buffer__Init	.text
sub_CA5457	.text
Buffer__Cleanup	.text
Buffer__Copy	.text
sub_CA54C8	.text
sub_CA5508	.text
Buffer__GetWritePtrAt	.text
Buffer__Write	.text
Util__BinToHex	.text
sub_CA5615	.text
sub_CA5628	.text
sub_CA5634	.text
sub_CA5642	.text
Buffer__Write_0	.text
sub_CA56C0	.text
sub_CA56D4	.text
sub_CA5757	.text
Crypto__GenRandomData	.text
Buffer__InitInternal	.text
sub_CA5815	.text
Buffer__Resize	.text
Buffer__GetSize	.text
String__FromCharArray	.text
String__Init	.text
sub_CA5898	.text
sub_CA58B5	.text
WString__Init	.text

FriedEx 2017-10-20

Function name	Segment
Core__GetUserIDInternal	.text
sub_2A56A2	.text
sub_2A58F0	.text
Buffer__Init	.text
sub_2A5952	.text
Buffer__Cleanup	.text
Buffer__Copy	.text
Buffer__GetWritePtrAt	.text
Buffer__Write	.text
Util__BinToHex	.text
sub_2A5A8F	.text
sub_2A5AD4	.text
Buffer__Write_0	.text
sub_2A5CDC	.text
Crypto__GenRandomData	.text
Buffer__InitInternal	.text
sub_2A5D40	.text
Buffer__Resize	.text
Buffer__GetSize	.text
MemsetWrapper_0	.text
String__FromCharArray	.text
String__Init	.text
sub_2A5E1D	.text
sub_2A5E3A	.text
WString__Init	.text

Figure 2. Comparison of function order in Dridex and FriedEx samples. Functions that are missing in the other sample are highlighted in the corresponding color

Another shared feature is the order of the functions in the binaries, which occurs when the same codebase or static library is used in multiple projects. As we can see in *Figure 2*, while the FriedEx sample seems to be missing some of the functions present in the Dridex sample and vice versa (which is caused by the compiler omitting unreferenced/unused functions), the order remains the same.

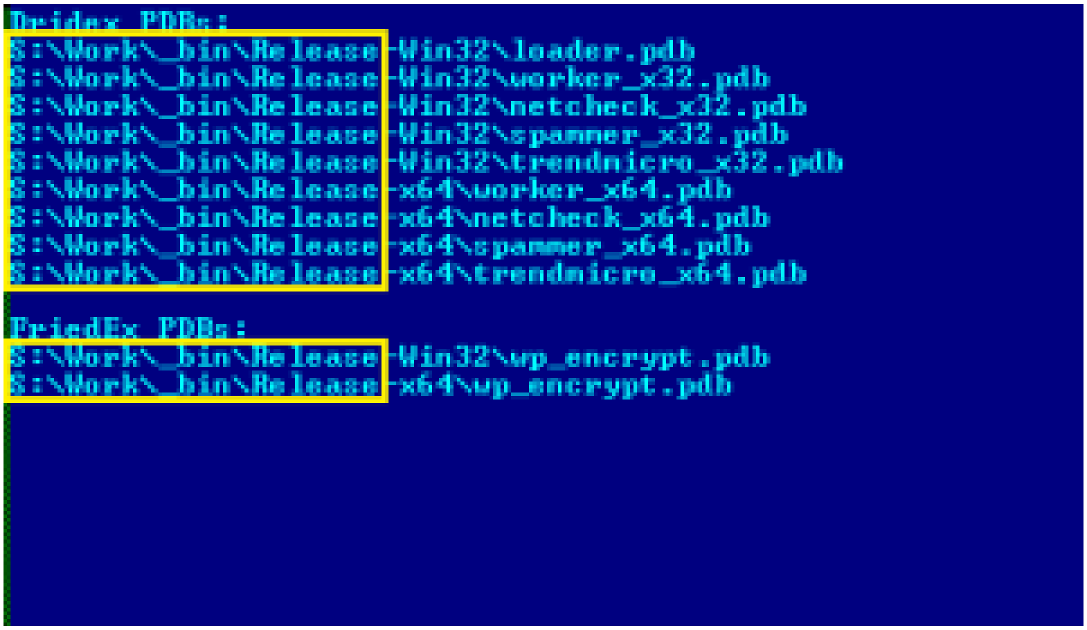
Note: Auto-generated function name pairs, based on code addresses (sub_CA5191 and sub_2A56A2, etc), obviously do not match, but the code they refer to does.

It’s also worth mentioning that both Dridex and FriedEx use the same malware packer. However, since the packer is very popular nowadays (probably due to its effectiveness in avoiding detection and hampering analysis) and used by other well-known families like QBot, Emotet or Ursnif, we don’t really consider that alone strong evidence.

PDB paths

When building a Windows executable, the linker may include a PDB (Program Database) path pointing to a file that contains debug symbols that help the developer with debugging and identifying crashes. The actual PDB file is almost never present in malware, because it’s a separate file that doesn’t get into distribution. However, sometimes even just the path, if included, can provide valuable information, because PDB files are located in the same directory as the compiled executable by default and usually also have the same base name followed by the .pdb extension.

As one might guess, PDB paths are not included in malware binaries very often, as the attackers don't want to give away any information. Fortunately, some samples of both families do include PDB paths.



```
Dridex PDBs:
S:\Work\_bin\Release-Win32\loader.pdb
S:\Work\_bin\Release-Win32\worker_x32.pdb
S:\Work\_bin\Release-Win32\netcheck_x32.pdb
S:\Work\_bin\Release-Win32\spammer_x32.pdb
S:\Work\_bin\Release-Win32\trendmicro_x32.pdb
S:\Work\_bin\Release-x64\worker_x64.pdb
S:\Work\_bin\Release-x64\netcheck_x64.pdb
S:\Work\_bin\Release-x64\spammer_x64.pdb
S:\Work\_bin\Release-x64\trendmicro_x64.pdb

FriedEx PDBs:
S:\Work\_bin\Release-Win32\wp_encrypt.pdb
S:\Work\_bin\Release-x64\wp_encrypt.pdb
```

Figure 3. List of all PDB paths found in the Dridex and FriedEx projects

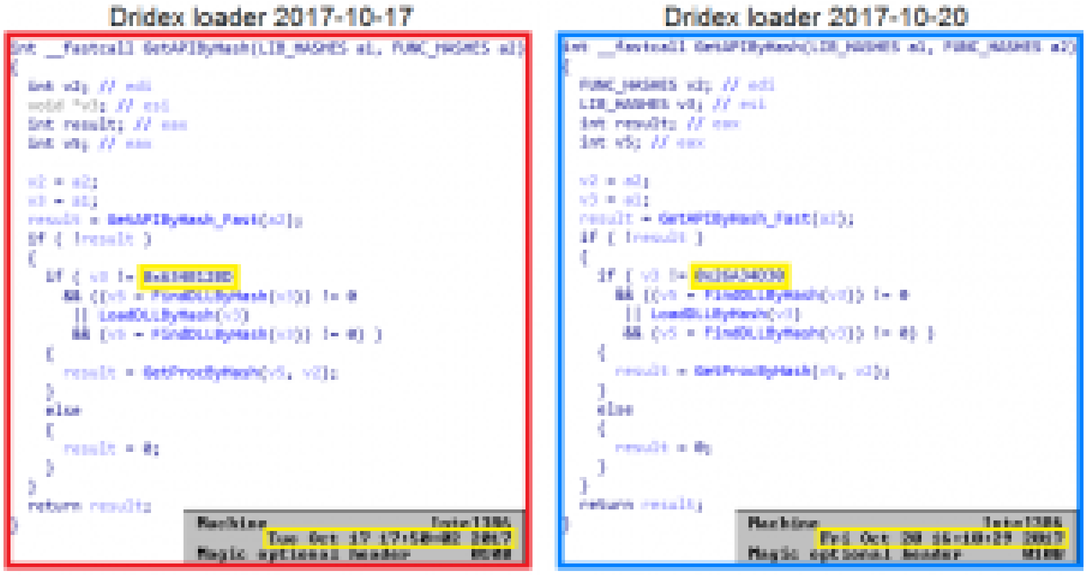
As you can see in *Figure 3*, the binaries of both projects are being built in the same, distinctively named directory. Based on a search across all of our malware sample metadata, we have concluded that the path S:\Work_bin\ is unique to the Dridex and FriedEx projects.

Timestamps

We found several cases of Dridex and FriedEx with the same date of compilation. This could, of course, be coincidence, but after a closer look, we quickly ruled out the "just a coincidence" theory.

Not only do the compilations with the same date have time differences of several minutes at most (which implies Dridex guys probably compile both projects concurrently), but the randomly generated constants are also identical in these samples. These constants change with each compilation as a form of polymorphism, to make the analysis harder and to help avoid detection.

This might be completely randomized in each compilation or based on some variable like the current date.



Dridex loader 2017-10-17

```
int __fastcall GetAPIByHash(LIB_HASHES a1, FUNC_HASHES a2)
{
  int v2; // edi
  void *v3; // esi
  int result; // eax
  int v5; // eax

  v2 = a2;
  v3 = a1;
  result = GetAPIByHash_Fast(a2);
  if ( !result )
  {
    if ( v3 != [illegible]
      && ((v5 = FindDLLByHash(v3)) != 0
       || LoadDLLByHash(v3)
        && (v5 = FindDLLByHash(v3)) != 0) )
    {
      result = GetProcByHash(v5, v2);
    }
    else
    {
      result = 0;
    }
  }
  return result;
}
```

```
Machine                          Intel386
        Tue Oct 17 17:58:02 2017
Magic optional header                0108
```

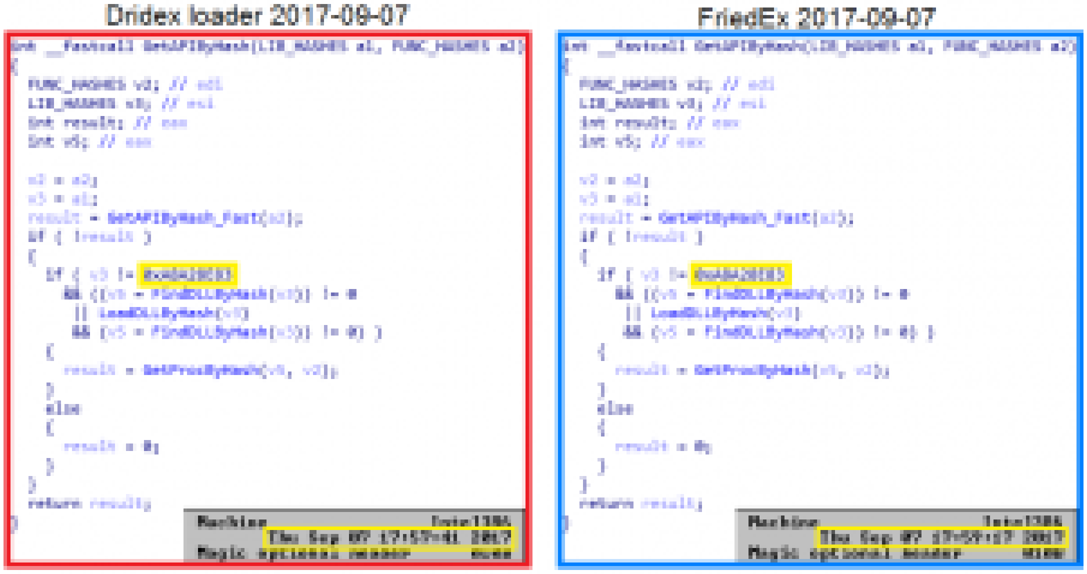
Dridex loader 2017-10-20

```
int __fastcall GetAPIByHash(LIB_HASHES a1, FUNC_HASHES a2)
{
  FUNC_HASHES v2; // edi
  LIB_HASHES v3; // esi
  int result; // eax
  int v5; // eax

  v2 = a2;
  v3 = a1;
  result = GetAPIByHash_Fast(a2);
  if ( !result )
  {
    if ( v3 != [illegible]
      && ((v5 = FindDLLByHash(v3)) != 0
       || LoadDLLByHash(v3)
        && (v5 = FindDLLByHash(v3)) != 0) )
    {
      result = GetProcByHash(v5, v2);
    }
    else
    {
      result = 0;
    }
  }
  return result;
}
```

```
Machine                          Intel386
        Fri Oct 20 16:18:29 2017
Magic optional header                0108
```

Figure 4. GetAPIByHash function in Dridex samples with compilation time difference of 3 days. The highlighted constant is different

In *Figure 4*, we have the comparison of two Dridex loader samples with a three-day difference between compilation timestamps. While the loaders are almost identical with the only difference being their hardcoded data, such as encryption keys and C&C IPs, the constants are different, and so are all the hashes that are based on them. On the other hand, in *Figure 5*, we can see the comparison of the FriedEx and Dridex loader from the same day (in fact, with timestamps just two minutes apart). Here, the constants are the same, meaning both were probably built during the same compilation session.



Dridex loader 2017-09-07

```
int __fastcall GetAPIByHash(LIB_HASHES a1, FUNC_HASHES a2)
{
  FUNC_HASHES v2; // edi
  LIB_HASHES v3; // esi
  int result; // eax
  int v5; // eax

  v2 = a2;
  v3 = a1;
  result = GetAPIByHash_Fast(a2);
  if ( !result )
  {
    if ( v3 != [illegible]
      && ((v5 = FindDLLByHash(v3)) != 0
       || LoadDLLByHash(v3)
        && (v5 = FindDLLByHash(v3)) != 0) )
    {
      result = GetProcByHash(v5, v2);
    }
    else
    {
      result = 0;
    }
  }
  return result;
}
```

```
Machine                          Intel386
        Thu Sep 07 17:57:41 2017
Magic optional header                0108
```

FriedEx 2017-09-07

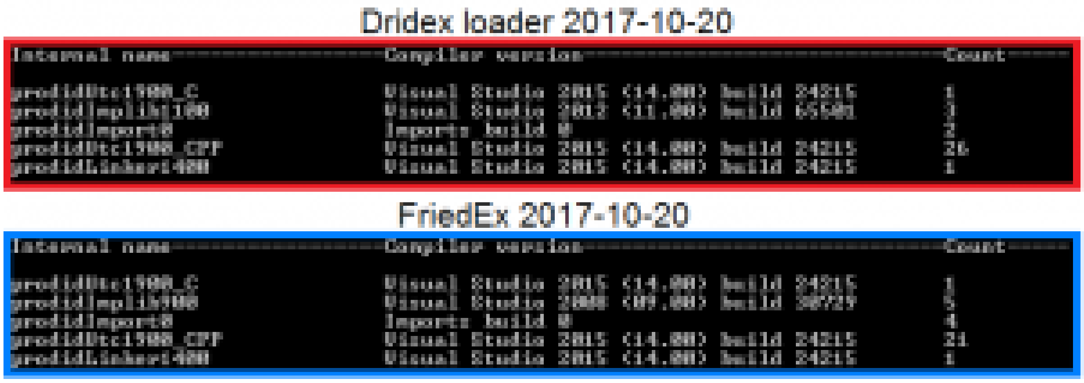
```
int __fastcall GetAPIByHash(LIB_HASHES a1, FUNC_HASHES a2)
{
  FUNC_HASHES v2; // edi
  LIB_HASHES v3; // esi
  int result; // eax
  int v5; // eax

  v2 = a2;
  v3 = a1;
  result = GetAPIByHash_Fast(a2);
  if ( !result )
  {
    if ( v3 != [illegible]
      && ((v5 = FindDLLByHash(v3)) != 0
       || LoadDLLByHash(v3)
        && (v5 = FindDLLByHash(v3)) != 0) )
    {
      result = GetProcByHash(v5, v2);
    }
    else
    {
      result = 0;
    }
  }
  return result;
}
```

```
Machine                          Intel386
        Thu Sep 07 17:59:17 2017
Magic optional header                0108
```

Figure 5. GetAPIByHash function in Dridex and FriedEx binaries compiled the same day. The highlighted constant is the same in both samples

Compiler information

The compiler information only further supports all the evidence we found so far – the binaries of both Dridex and FriedEx are compiled in Visual Studio 2015. This is confirmed by both the linker version found in the PE Header and Rich Header data.



Dridex loader 2017-10-20

```
Internal name------------------Compiler version--------------------------------Count-----

prodidUtc1900_C              Visual Studio 2015 (14.00) build 24215          1
prodidImplib1100             Visual Studio 2012 (11.00) build 65501          3
prodidImport0                Imports build 0                                 2
prodidUtc1900_CPP            Visual Studio 2015 (14.00) build 24215          26
prodidLinker1400             Visual Studio 2015 (14.00) build 24215          1
```

FriedEx 2017-10-20

```
Internal name------------------Compiler version--------------------------------Count-----

prodidUtc1900_C              Visual Studio 2015 (14.00) build 24215          1
prodidImplib900              Visual Studio 2008 (09.00) build 30729          5
prodidImport0                Imports build 0                                 4
prodidUtc1900_CPP            Visual Studio 2015 (14.00) build 24215          21
prodidLinker1400             Visual Studio 2015 (14.00) build 24215          1
```

Figure 6. Rich header data found in Dridex and FriedEx samples

Apart from the obvious similarities with Dridex, we came across a previously unreported 64-bit variant of the ransomware. As the usual 32-bit version of the ransomware can target both x86 and x64 systems, we consider this variant to be a bit of a curiosity.

Conclusion

With all this evidence, we confidently claim that FriedEx is indeed the work of the Dridex developers. This discovery gives us a better picture of the group's activities – we can see that the group continues to be active and not only consistently updates their banking trojan to maintain its webinject support for the latest versions of Chrome and to introduce new features like Atom Bombing, but that it also follows the latest malware "trends", creating their own ransomware.

We can only guess what the future will bring, but we can be sure that the Dridex gang isn't going anywhere anytime soon and that they will keep innovating their old project and possibly extend their portfolio with a new piece here and there.

For a long time, it was believed that the Dridex gang was a one-trick pony that kept their focus on their banking trojan. We have now found that this is not the case and that they can easily adapt to the newest trends and create a different kind of malware that can compete with the most advanced in its category.

IoCs

Win32/Dridex.BE	**C70BD77A5415B5DCF66B7095B22A0DEE2DDA95A0**
Win64/FriedEx.A	**CF1038C9AED9239B6A54EFF17EB61CAB2EE12141**
Win32/FriedEx.A	**8AE1C1869C42DAA035032341804AEFC3E7F3CAF1**

26 Jan 2018 - 02:57PM

Sign up to receive an email update whenever a new article is published in our Ukraine Crisis – Digital Security Resource Center

Newsletter

Discussion