Ice IX: not cool at all

By Dmitry Tarakanov

Published: 2011-09-14 · Archived: 2026-04-05 17:47:21 UTC

My colleague Jorge Mieres recently found a C&C server of a botnet based on a malicious program called Ice IX. As announced on several user forums, Ice IX is a bot created using the source code of ZeuS 2.0.8.9, which became publicly available in May. The author of the new bot says the program includes substantial enhancements, which should be interesting to those cybercriminals who steal money from users with the help of banking Trojans.

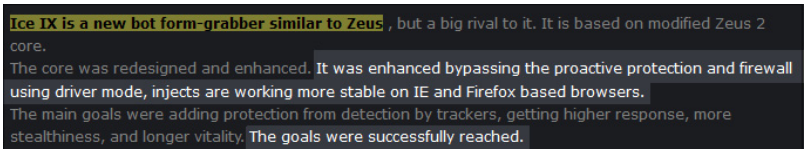

Figure 1. Description of the bot

As you can see in the screenshot, the description of the new program focuses on the enhancements allegedly introduced into the ZeuS original code. These included bypassing firewalls, bypassing proactive protection provided by security products, and protection from detection by trackers. The latter obviously refers to the ZeuS Tracker https://zeustracker.abuse.ch, which has been making cybercriminals' life difficult. The program's author charged $600 for a version of the bot with a hardwired URL that the bot must connect to after infection (i.e., the C&C address), and $1800 for a version without a hard-coded C&C address.

Unfortunately, we were unable to obtain a sample of the enhanced Ice IX version – possibly, because nobody had purchased it. Most likely, this version included a mechanism that was similar to that implemented in ZeuS beginning with version 2.1. Here is how it worked in ZeuS: the bot included a key that was used in combination with the current date to generate 1020 domain names each day. The bot searched through this entire list, trying to find its C&C server.

At the same time, someone has apparently tested the base version of the bot kit. These samples were analyzed for differences from the original ZeuS samples used as the basis for the Ice IX bot.

I must confess, I had expected more. The author advertized the programs as something special, and in addition, there was a comment in the thread that the author deserved credit for bypassing proactive detection, since it was an important improvement, concluding that this was no doubt a completely new bot, much better than ZeuS… In fact, however, it was all a bunch of lies. There were no major improvements compared to ZeuS 2.0.8.9 – the version which became publicly available.

Here are the differences I was able to identify:

1) ZeuS can find the user's email credentials saved on the infected system. The bot sends any data found to the botnet operator, giving the cybercriminal access to the victim's mailboxes. However, the code section responsible for finding and processing email credentials was commented out in the original ZeuS source code. The author of

Ice IX simply removed the comment marks from this code, so the modules that were not included in ZeuS 2.0.8.9 samples were present in his bot.

2). The ZeuS 2.0.8.9 bot can be launched with the following arguments: -f, -n, -v, -i. I won't go into the discussion of what each of them means. Let me just mention the key –i: if ZeuS 2.0.8.9 sample is launched with this key, a window with some information about the bot will be displayed:

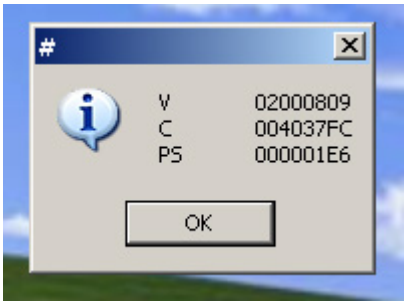

Figure 2. ZeuS 2.0.8.9 information window

The author of Ice IX simply removed the fragment processing this key from the code. Consequently, Ice IX sample does not support this argument.

3) A modified function that is associated with reading data from the registry has been identified. There is a small chance that this could prove that "enhancement" introduced in order to bypass proactive protection provided by security products. However, it could also be merely the consequence of compiler optimization – the result of compiling the bot's code might have been slightly different from that for the original ZeuS code due to the changes, albeit small, introduced into Ice IX. In the ZeuS 2.0.8.9 code, the function that reads data from the registry includes all the API functions required for this task, i.e., RegOpenKeyEx, RegQueryValueEx and RegCloseKey:

```
signed int __stdcall an_Read_Registry_40E003(HKEY hKey, signed int cbData, LPCWSTR lpValueName, LPDWORD lpType, int a5)
{
  void *v5; // ebx@5
  signed int v7; // [sp+4h] [bp-4h]@1

  v7 = -1;
  *(_DWORD *)a5 = 0;
  if ( !RegOpenKeyExW(hKey, (LPCWSTR)cbData, 0, 1u, &hKey) )
  {
    cbData = 0;
    if ( !RegQueryValueExW(hKey, lpValueName, 0, lpType, 0, (LPDWORD)&cbData) )
    {
      if ( cbData )
      {
        v5 = an_heap_alloc((LPVOID)(cbData + 4));
        if ( v5 )
        {
          if ( RegQueryValueExW(hKey, lpValueName, 0, lpType, (LPBYTE)v5, (LPDWORD)&cbData) )
          {
            an_heap_free(v5);
          }
          else
          {
            *(_DWORD *)a5 = v5;
            v7 = cbData;
          }
        }
      }
      else
      {
        v7 = 0;
      }
    }
    RegCloseKey(hKey);
  }
  return v7;
}
```

Figure 3. The function in ZeuS which reads data from the registry

When a value needed to be read from the registry, it was done as follows:

```
v2 = 0;
v3 = an_Read_Registry_40E003(HKEY_LOCAL_MACHINE, (signed int)&cbData, &ValueName, 0, (int)&lpMem);
if ( v3 != -1 && (unsigned int)v3 > 0 )
```

Figure 3a. Calling the function which reads from the registry in ZeuS

In the Ice IX sample, there are some changes in the places where the function is called. The API function RegOpenKeyEx was removed from the function that reads registry data:

```
DWORD __userpurge an_Query_Registry_4170D7<eax>(HKEY *a1<edi>, LPCWSTR lpValueName, LPDWORD lpType, int a4)
{
  BYTE *v4; // eax@4
  void *lpMem; // [sp+8h] [bp-Ch]@4
  DWORD v7; // [sp+Ch] [bp-8h]@1
  DWORD cbData; // [sp+10h] [bp-4h]@1

  v7 = -1;
  cbData = 0;
  if ( !RegQueryValueExW(*a1, lpValueName, 0, lpType, 0, &cbData) )
  {
    if ( cbData )
    {
      v4 = (BYTE *)an_heap_alloc((LPVOID)(cbData + 4));
      lpMem = v4;
      if ( v4 )
      {
        if ( RegQueryValueExW(*a1, lpValueName, 0, lpType, v4, &cbData) )
        {
          an_heap_free(lpMem);
        }
        else
        {
          *(_DWORD *)a4 = lpMem;
          v7 = cbData;
        }
      }
    }
    else
    {
      v7 = 0;
    }
  }
  RegCloseKey(*a1);
  return v7;
}
```

Figure 4. The function in Ice IX that reads data from registry

As a result, whenever a value needed to be read from the registry, the API function RegOpenKeyEx was called first to open the registry key (e.g., HKEY_CURRENT_USER or HKEY_LOCAL_MACHINE) before calling the actual registry read function:

```
if ( RegOpenKeyExW(HKEY_CURRENT_USER, &SubKey, 0, 1u, &phkResult) )
{
  v6 = -1;
}
else
{
  v6 = an_Query_Registry_4170D7(&phkResult, &ValueName, 0, (int)&v13);
  v4 = 0;
}
```

Figure 4a. Calling the function to read from registry in Ice IX

I admit that some antivirus products may possibly detect ZeuS based on the presence of the above registry read function. It is quite probable that this essential function is present in all ZeuS samples regardless of version; it is also possible that its code uniquely identifies this entire malware family. Why not, after all? In this case, modifying this function (e.g., removing RegOpenKeyEx) would help to prevent the detection which depends on it.

I didn't test all antivirus products on Ice IX samples, so I cannot say whether any product would fail to detect them because of this change. I only scanned the sample with KIS 2012 using old antivirus databases dating back to June, when nothing was known as yet about Ice IX. As the bot was runnung, KIS 2012 detected dangerous activity and blocked the program's execution. No wonder: given the long history and extensive functionality of ZeuS, there are quite a few criteria based on which KIS/KAV can detect this malware family's malicious code.

Now, let's move on to the significant changes that distinguish Ice IX from ZeuS 2.0.8.9 at least to some extent.

3) In the ZeuS configuration file, there is a section called "Web Filters" in which the botnet operator defines how the bot should respond when the user visits certain websites. This is done using special characters "!", "@", "-", "^" are used.

Let’s look at the way in which the “@” character is used. It is placed before the URL (e.g., @*/login.osmp.ru/*) to tell the bot to make screenshots when the user visits any addresses matching the mask specified every time that the user left -clicks the mouse, and then to send the screenshots to the cybercriminal. This is a mechanism that allows the cybercriminal to reconstruct the data entered by the user on the website using the virtual keyboard. Other symbols also define specific actions to be performed by the bot. All that the author of Ice IX did was to assign different characters to the same functions: the letters “N”, “S”, “C” and “B” have replaced “!”, “@”, “-” and “^”, respectively.

4) The last distinguishing feature is a slightly modified method used by the bot to download the configuration file. In its code, ZeuS includes a hard-coded URL of a configuration file that anyone can download, e.g., http://www.example.com/files/config.bin. The author of Ice IX makes the point that it is this availability of configuration files that is at the root of all problems with trackers. So how does he address this issue? Here is his solution. You can no longer simply download the configuration file from a URL. Instead, you must send a specially formed POST request to a certain address (which is actually a URL hardcoded into the bot in the same location as in ZeuS 2.0.8.9). The request must include a pair of parameters: “id=&hash=”, for example:

id=TEST_WIN_XP_B5DF77116522DF69&hash=DC0D2CAB39D49FC3D5E467501A2682C5

id is the bot’s identifier calculated using the same algorithm as in ZeuS 2.0.8.9. It is used for the bot’s direct communication with the C&C.

The identifier is the computer’s name with a unique 16-digit hexadecimal number added to it. From the C&C viewpoint, both the computer name and the 16-digit number can be arbitrary. This identifier is encrypted using the RC4 algorithm (which ZeuS uses all the time to encrypt data) in combination with an S-box that is also hardcoded into the bot. The MD5 checksum is calculated for the encrypted data and sent as a hash variable. Since the bot identifier can be arbitrary (at the most it needs to meet the COMPUTERNAME_16CHARSHEXNUMBER format), the only data needed to obtain the configuration file is the S-box – it is needed to encrypt the bot identifier. But wait a minute. The configuration file is also encrypted using the same RC4 algorithm with the same S-box. Without the S-box, the configuration file is useless, a meaningless sequence of bytes. The really valuable stuff is inside the file.

So in the end it all comes down to this:

Bot	What is needed to obtain useful data from the configuration file
Ice IX	S-box
ZeuS	S-box

The question is: what is it that is supposed to make trackers’ life more difficult? And the obvious answer is “virtually nothing”. It might perhaps take an extra half hour to run a sample, make a dump, and identify the changes made to known code. And that only is one is going to do the analysis and mass-download different bots’ configuration files. However, there is an easier way: getting the parameters of the POST request from an infected computer’s traffic, which is a matter of a few minutes. With all the hype, you wouldn’t believe it!

There is a saying about sports that can be applied to this situation: It takes one athlete with a 9-meter jump to win the Olympics not 9 athletes with 1-meter jumps. Same here: it doesn't matter how many times and on how many values are encrypted using the same algorithm and the same key – with the same source data, more iterations of encryption will not result in a significantly stronger encryption algorithm. But apparently, this is not what the cybercriminal was after, and this entire business is fraud, plain and simple. Somebody decided to make some easy money by selling supposedly enhanced malware with functionality that is already publicly available.

Source: https://securelist.com/ice-ix-not-cool-at-all/29111/