A phishing document signed by Microsoft – part 1

outflank.nl/blog/2021/12/09/a-phishing-document-signed-by-microsoft

Dima Van de Wouw December 9, 2021

This blog post is part of series of two posts that describe weaknesses in Microsoft Excel that could be leveraged to create malicious phishing documents signed by Microsoft that load arbitrary code.

These weaknesses have been addressed by Microsoft in the following patch: CVE-2021-28449. This patch means that the methods described in this post are no longer applicable to an up-to-date *and securely configured* MS Office install. However, we will uncover a largely unexplored attack surface of MS Office for further offensive research and will demonstrate practical tradecraft for exploitation.

In this blog post (part 1), we will discuss the following:

- The Microsoft Analysis ToolPak Excel and vulnerabilities in XLAM add-ins which are distributed as part of this.
- Practical offensive MS Office tradecraft which is useful for weaponizing signed add-ins which contain vulnerabilities, such as transposing third party signed macros to other documents.
- Our analysis of Microsoft's mitigations applied by CVE-2021-28449.

We will update this post with a reference to part 2 once it is ready.

An MS Office installation comes with signed Microsoft Analysis ToolPak Excel add-ins (.XLAM file type) which are vulnerable to multiple code injections. An attacker can embed malicious code without invalidating the signature for use in phishing scenarios. These specific XLAM documents are signed by Microsoft.

The resulting exploit/maldoc supports roughly all versions of Office (x86+x64) for any Windows version against (un)privileged users, without any prior knowledge of the target environment. We have seen various situations at our clients where the specific Microsoft certificate is added as a Trusted Publisher (meaning code execution without a popup after opening the maldoc). In other situations a user will get a popup showing a legit Microsoft signature. Ideal for phishing!

Research background

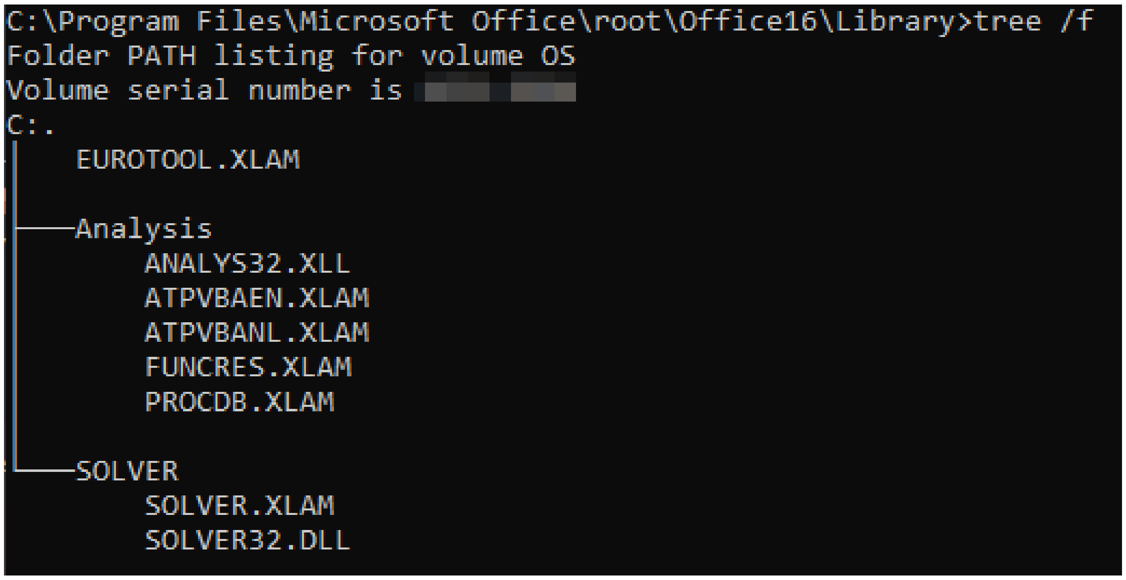
At Outflank, we recognise that initial access using maldocs is getting harder due to increased effectiveness of EDR/antimalware products and security hardening options for MS Office. Hence, we continuously explore new vectors for attacking this surface.

During one of my research nights, I started to look in the MS Office installation directory in search of example documents to further understand the Office Open XML (OpenXML) format and its usage. After strolling through the directory `C:\program files\Microsoft Office\` for hours and hours, I found an interesting file that was doing something weird.

Introduction to Microsoft's Analysis Toolpak add-in XLAMs

The Microsoft Office installation includes a component named "Microsoft's Analysis ToolPak add-in". This component is implemented via Excel Add-ins (.XLAM), typically named ATPVBAEN.XLAM and located in the office installation directory. In the same directory, there is an XLL called analys32.xll which is loaded by this XLAM. An XLL is a DLL based Excel add-in.

The folders and files structure are the same for all versions and look like this:



```
C:\Program Files\Microsoft Office\root\Office16\Library>tree /f
Folder PATH listing for volume OS
Volume serial number is [redacted]
C:.
│   EUROTOOL.XLAM
│
├───Analysis
│       ANALYS32.XLL
│       ATPVBAEN.XLAM
│       ATPVBANL.XLAM
│       FUNCRES.XLAM
│       PROCDB.XLAM
│
└───SOLVER
        SOLVER.XLAM
        SOLVER32.DLL
```

The Excel macro enabled add-in file (XLAM) file format is relatively similar to a regular macro enabled Excel file (XLSM). An XLAM file usually contains specific extensions to Excel so new functionality and functions can be used in a workbook. Our ATPVBAEN.XLAM target implements this via VBA code which is signed by Microsoft. However, signing the VBA code does not imply integrity control over the document contents or the resources it loads…

Malicious code execution through RegisterXLL

So, as a first attempt I copied ATPVBAEN.XLAM to my desktop together with a malicious XLL which was renamed to analys32.xll. The signed XLAM indeed loaded the unsigned malicious XLL and I had the feeling that this could get interesting.

Normally, the signed VBA code in ATPVBAEN.XLAM is used to load an XLL in the same directory via a call to RegisterXLL. The exact path of this XLL is provided inside an Excel cell in the XLAM file. Cells in a worksheet are not signed or validated and can be manipulated by an attacker. In addition, there is no integrity check upon loading the XLL. Also, no warning is given, even if the XLL is unsigned or loaded from a remote location.

We managed to weaponize this into a working phishing document loading an XLL over WebDAV. Let's explore why this happened.

No integrity checks on loading unsigned code from a signed context using RegisterXLL

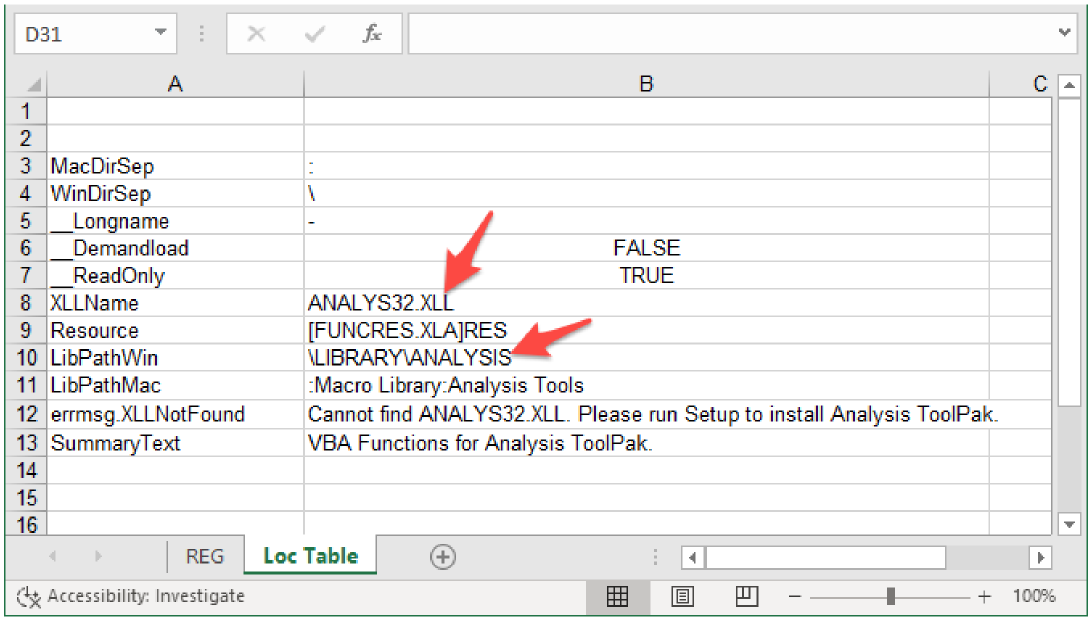
ATPVBAEN.XLAM loads ANALYS32.XLL and uses its exported functions to provide functionality to the user. The XLAM loads the XLL using the following series of functions which are analysable using the `olevba` tool. Note that an XLL is essentially just a DLL with the function `xlAutoOpen` exported. The highlighted variables and functions are part of the vulnerable code:

```
> olevba ATPVBAEN.XLAM
olevba 0.55.1 on Python 3.7.3 - http://decalage.info/python/oletools
=================================================================
VBA MACRO VBA Functions and Subs.bas
in file: xl/vbaProject.bin - OLE stream: 'VBA/VBA Functions and Subs'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
' ANALYSIS TOOLPAK  -  Excel AddIn
' The following function declarations provide interface between VBA and ATP XLL.

' These variables point to the corresponding cell in the Loc Table sheet.
Const XLLNameCell = "B8"
Const MacDirSepCell = "B3"
Const WinDirSepCell = "B4"
Const LibPathWinCell = "B10"
Const LibPathMacCell = "B11"

Dim DirSep As String
Dim LibPath As String
Dim AnalysisPath As String
```

The name of the XLL is saved in cell B4 and the path in cell B10. Which looks as follows, if you unhide the worksheet:



D31

	A	B
1		
2		
3	MacDirSep	:
4	WinDirSep	\
5	__Longname	-
6	__Demandload	FALSE
7	__ReadOnly	TRUE
8	XLLName	ANALYS32.XLL
9	Resource	[FUNCRES.XLA]RES
10	LibPathWin	\LIBRARY\ANALYSIS
11	LibPathMac	:Macro Library:Analysis Tools
12	errmsg.XLLNotFound	Cannot find ANALYS32.XLL. Please run Setup to install Analysis ToolPak.
13	SummaryText	VBA Functions for Analysis ToolPak.
14		
15		
16		

The `auto_open()` function is called when the file is opened and the macro's are enabled/trusted.

```
' Setup & Registering functions

Sub auto_open()
    Application.EnableCancelKey = xlDisabled
    SetupFunctionIDs
    PickPlatform
    VerifyOpen
    RegisterFunctionIDs
End Sub
```

First, the `PickPlatform` function is called to set the variables. `LibPath`, is set here to `LibPathWinCell`'s value (which is under the attacker's control) in case the workbook is opened on Windows.

```
Private Sub PickPlatform()
    Dim Platform

    ThisWorkbook.Sheets("REG").Activate
    Range("C3").Select
    Platform = Application.ExecuteExcel4Macro("LEFT(GET.WORKSPACE(1),3)")
    If (Platform = "Mac") Then
        DirSep = ThisWorkbook.Sheets("Loc Table").Range(MacDirSepCell).Value
        LibPath = ThisWorkbook.Sheets("Loc Table").Range(LibPathMacCell).Value
    Else
        DirSep = ThisWorkbook.Sheets("Loc Table").Range(WinDirSepCell).Value
        LibPath = ThisWorkbook.Sheets("Loc Table").Range(LibPathWinCell).Value
    End If
End Sub
```

The function `VerifyOpen` will try looking for the XLL, as named in `XLLNameCell = "B8"`, then start looking in the entire `PATH` of the system and finally look in the path as defined by `LibPath`. Note, all (red / orange) highlighted variables are under the attacker's control and vulnerable. We are going to focus on attacking the red highlighted code.

```
Private Sub VerifyOpen()
    XLLName = ThisWorkbook.Sheets("Loc Table").Range(XLLNameCell).Value

    theArray = Application.RegisteredFunctions
    If Not (IsNull(theArray)) Then
        For i = LBound(theArray) To UBound(theArray)
            If (InStr(theArray(i, 1), XLLName)) Then
                Exit Sub
            End If
        Next i
    End If

    Quote = String(1, 34)
    ThisWorkbook.Sheets("REG").Activate
    WorkbookName = "[" & ThisWorkbook.Name & "]" & Sheet1.Name
    AnalysisPath = ThisWorkbook.Path

    AnalysisPath = AnalysisPath & DirSep
    XLLFound = Application.RegisterXLL(AnalysisPath & XLLName)
    If (XLLFound) Then
        Exit Sub
    End If

    AnalysisPath = ""
    XLLFound = Application.RegisterXLL(AnalysisPath & XLLName)
    If (XLLFound) Then
        Exit Sub
    End If

    AnalysisPath = LibPath
    XLLFound = Application.RegisterXLL(AnalysisPath & XLLName)
    If (XLLFound) Then
        Exit Sub
    End If

    XLLNotFoundErr = ThisWorkbook.Sheets("Loc Table").Range("B12").Value
    MsgBox (XLLNotFoundErr)
    ThisWorkbook.Close (False)
End Sub
```

`RegisterXLL` will load any XLL without warning / user validation. Copying the XLAM to another folder and adding a (malicious) XLL named ANALYS32.XLL in the same folder allows for unsigned code execution from a signed context.

There are no integrity checks on loading additional (unsigned) resources from a user-accepted (signed) context.

Practical weaponization and handling different MS Office installs

For full weaponization, an attacker needs to supply the correct XLL (32 vs 64 bit) as well as a method to deliver multiple files, both the Excel file and the XLLs. How can we solve this?

Simple weaponization

The simplest version of this attack can be weaponized by an attacker once he can deliver multiple files, an Excel file and the XLL payload. This can be achieved by multiple vectors, e.g. offering multiple files for download and container formats such as .zip, .cab or .iso.

In the easiest form, the attacker would copy the ATPVBAEN.XLAM from the Office directory and serve a malicious XLL, named ANALYS32.XLL next to it. The XLAM can be renamed according to the phishing scenario. By changing the `XLLName` Cell in the XLAM, it is possible to change the XLL name to an arbitrary value as well.

MS Office x86 vs x64 bitness – Referencing the correct x86 and x64 XLLs (PoC 1)

For a full weaponization, an attacker would require knowledge on whether 64-bit or 32-bit versions of MS Office are used at a victim. This is required because an XLL payload (DLL) works for either x64 or x86.

It is possible to obtain the Office bitness using `=INFO("OSVERSION")` since the function is executed when the worksheet is opened, before the VBA Macro code is executed. For clarification, the resulting version string includes the version of Windows and the bitness of Office, ex; “Windows (32-bit) NT 10.00”. An attacker can provide both 32- and 64-bit XLLs and use Excel formulas to load the correct XLL version.

The final bundle to be delivered to the target would contain:

PoC-1-local.zip
├ Loader.xlam
├ demo64.dat
├ demo32.dat

A 64-bit XLL is renamed to demo64.dat and is loaded from the same folder. It can be served as zip, iso, cab, double download, etc.

Payload: Changed `XLLName` cell B8 to
```
= "demo" & IF(ISERROR(SEARCH("64";INFO("OSVERSION"))); "32"; "64") &
".dat"
```

Loading the XLL via webdav

With various Office trickery, we also created a version where the XLAM/XLSM could be sent directly via email and would load the XLL via WebDAV. Details of this are beyond the scope of this blog, but there are quite a few tricks to enable the WebDAV client on a target's machine via MS Office (but that is for part 2 of this series).

Signed macro transposing to different file formats

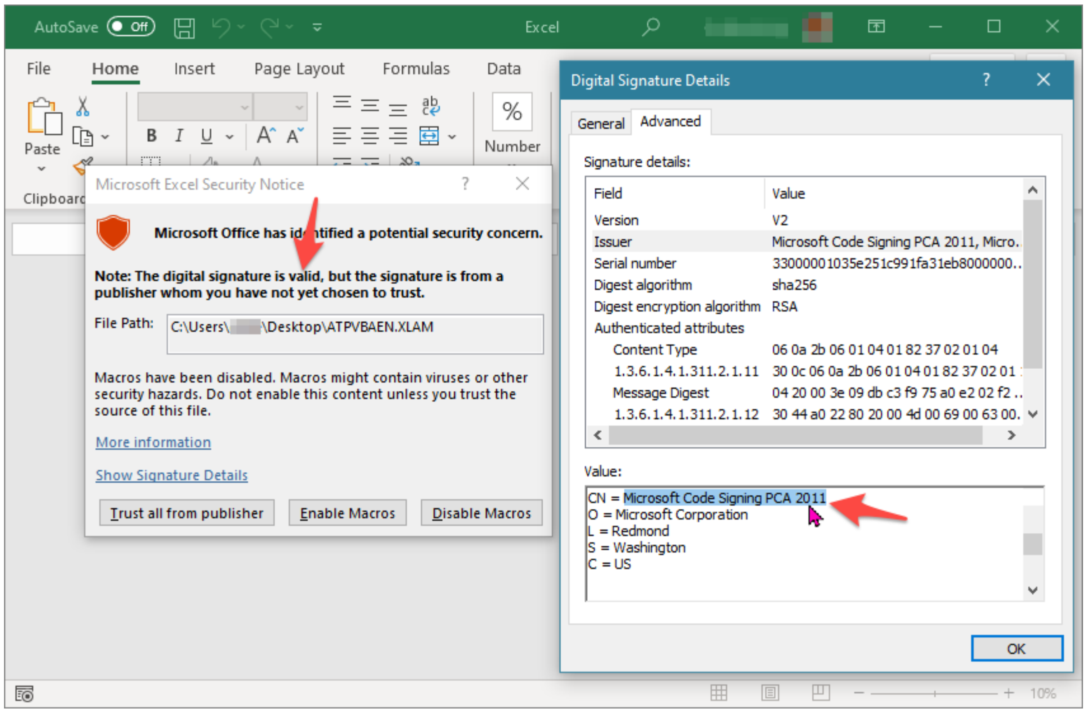
By copying the `vbaproject.bin`, signed VBA code can be copied/transposed into other file formats and extensions (e.g. from XLAM to XLSM to XLS).

Similarly, changing the file extension from XLAM to XLSM can be performed by changing one word inside the document in `[Content_Types].xml` from '`addin`' to '`sheet`'. The Save As menu option can be used to convert the XLSM (Open XML) to XLS (compound file).

Signature details

Some noteworthy aspects of the signature that is applied on the VBA code:

- The VBA code in the XLAM files is signed by Microsoft using timestamp signing which causes the certificate and signature to remain valid, even after certificate expiration. As far as we know, timestamp signed documents for MS Office cannot be revoked.
- The XLAMs located in the office installer are signed by `CN = Microsoft Code Signing PCA 2011` with varying validity start and end dates. It appears that Microsoft uses a new certificate every half year, so there are multiple versions of this certificate in use.
- In various real-world implementations at our clients, we have seen the Microsoft Code Signing PCA 2011 installed as a Trusted Publisher. Some online resources hint towards adding this Microsoft root as a trusted publisher. This means code execution without a popup after opening the maldoc.
- In case an environment does not have the Microsoft certificate as trusted publisher, then the user will be presented with a macro warning. The user can inspect the signature details and will observe that this is a genuine Microsoft signed file.



Microsoft Excel Security Notice

Microsoft Office has identified a potential security concern.

Note: The digital signature is valid, but the signature is from a publisher whom you have not yet chosen to trust.

File Path: C:\Users\[redacted]\Desktop\ATPVBAEN.XLAM

Macros have been disabled. Macros might contain viruses or other security hazards. Do not enable this content unless you trust the source of this file.

More information

Show Signature Details

Trust all from publisher | Enable Macros | Disable Macros

Digital Signature Details

General | Advanced

Signature details:

Field	Value
Version	V2
Issuer	Microsoft Code Signing PCA 2011, Micro..
Serial number	33000001035e251c991fa31eb8000000..
Digest algorithm	sha256
Digest encryption algorithm	RSA
Authenticated attributes	
Content Type	06 0a 2b 06 01 04 01 82 37 02 01 04
1.3.6.1.4.1.311.2.1.11	30 0c 06 0a 2b 06 01 04 01 82 37 02 01
Message Digest	04 20 00 3e 09 db c3 f9 75 a0 e2 02 f2 ..
1.3.6.1.4.1.311.2.1.12	30 44 a0 22 80 20 00 4d 00 69 00 63 00.

Value:

```
CN = Microsoft Code Signing PCA 2011
O = Microsoft Corporation
L = Redmond
S = Washington
C = US
```

OK

Impact summary

An attacker can transpose the signed code into various other formats (e.g. XLS, XLSM) and use it as a phishing vector.

In case a victim system has marked the Microsoft certificate as trusted publisher and the attacker manages to target the correct certificate version a victim will get no notification and attacker code is executed.

In case the certificate is not trusted, the user will get a notification and might enable the macro as it is legitimately signed by Microsoft.

Scope: Windows & Mac?

Affected products: confirmed on all recent versions of Microsoft Excel (2013, 2016, 2019), for both x86 and x64 architectures. We have found signed and vulnerable ATPVBAEN.XLAM files dating back from 2009 while the file contains references to “*Copyright 1991,1993 Microsoft Corporation*”, hinting this vulnerability could be present for a very long time.

It is noteworthy to mention that the XLAM add-in that we found supports both paths for Windows and for MacOS and launches the correct XLL payload accordingly. Although MacOS is likely affected, it has not been explicitly tested by us. Theoretically, the sandbox should mitigate (part of) the impact. Have fun exploring this yourself, previous applications by other researchers of our MS Office research to the Mac world have had quite some impact.

Microsoft’s mitigation

Microsoft acknowledged the vulnerability, assigned it https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-28449 and patched it 5 months later.

Mitigation of this vulnerability was not trivial, due to other weaknesses in these files (see future blog post 2 of this series). Mitigation of the weakness described in this post has been implemented by signing the XLL and a new check that prevents 'downgrading' and loading of an unsigned XLL. By default, downgrading is not allowed but this behavior can be manipulated/influenced via the registry value `SkipSignatureCheckForUnsafeXLL` as described in https://support.microsoft.com/en-gb/office/add-ins-and-vba-macros-disabled-20e11f79-6a41-4252-b54e-09a76cdf5101.

Disclosure timeline

Submitted to MSRC: 30 November 2020

Patch release: April 2021

Public disclosure: December 2021

Acknowledgement: Pieter Ceelen & Dima van de Wouw (Outflank)

Next blog: Other vulnerabilities and why the patch was more complex

The next blog post of this series will explain other vulnerabilities in the same code, show alternative weaponization methods and explain why the patch for CVE-2021-28449 was a complex one.