Having fun with an Ursnif VBS dropper

By Robert Giczewski

Published: 2020-11-27 · Archived: 2026-04-05 22:37:03 UTC

I recently stumbled across an interesting sample that was delivered as part of an encrypted zip archive via a Google-Drive link. The password for the archive was sent by email together with the Google-Drive link. Since the sample runs only partially in some sandboxes and it's not even starting in others, I took a closer look at it.

The sample can be found on VirusTotal and there are still only ten detections so far (even though it's on VT for two months now).
fd490c7b728af08052cf4876c1fc8c6e290bde368b6343492d60fc9d8364a7e5 - aPsYyn8Rw2Xf.vbs

Looking at the file extension, you could already guess it's a Visual Basic Script file, which however appears unusually large. Due to the size, the actual payload is most probably somehow hidden in the VBS file so lets have a look into the file.

Deobfuscation

Scrolling through the file we see lots of useless comments, some array definitions, some constant definitions and a for loop.

To get rid of all the useless code, I wrote a quick'n'dirty python tool to remove all the junk code and convert the remaining code to python for easier analysis. Since the constant and array definitions are mixed up in the code, we have to restructure them. I moved all const definitions to the beginning followed by the array definitions, the function calls and everything else at the end.

```
 f = open("aPsYyn8Rw2Xf.vbs", "r")

const_lines = []
array_lines = []
execute_lines = []
loop_lines = []
everything_else = []


for line in f:
    if not (line.startswith("'") or line.startswith("REM")):
        if "const" in line:
            const_lines.append(line.replace("const", "").replace("\n", "").replace(" ", ""))
        elif "Array(" in line:
```

```
            array_lines.append(line.replace("Array(", "[").
                               replace(")", "]").replace("\n", "").strip())
        elif "Execute" in line:
            execute_lines.append(line.replace("Execute", "print").replace("\n", "").strip())
        elif line.startswith("for"):
            loop_lines.append(line.replace("\n", "").strip())
        else:
            everything_else.append(line.replace("\n", ""))

for item in const_lines:
    print(item)
for item in array_lines:
    print(item)
for item in execute_lines:
    print(item)
for item in loop_lines:
    print(item)
for item in everything_else:
    if len(item) > 0:
        print(item)
```

After running the python script, we will get a new cleaned up code which is almost runnable in python.

At the end we can spot a function `kuHKE()` which is called several times and is taking an array as an argument. This is most probably the function which is used for decoding all the arrays. Another thing here to mention are the function calls at the end of the cleaned code. Those will be relevant later when we have the final deobfuscated code.

So let's rewrite the `kuHKE()` function into python and remove the function calls at the end.

```
def kuHKE(EUnWxs):
    result = ""
    for Mali842 in EUnWxs:
        result += chr(Mali842 - ((26 + 30) - ((17 - 1) + 35)))

    return result
```

After executing the cleaned code, we still get a little bit of obfuscated code but since it's not very much, we can easily do it manually.

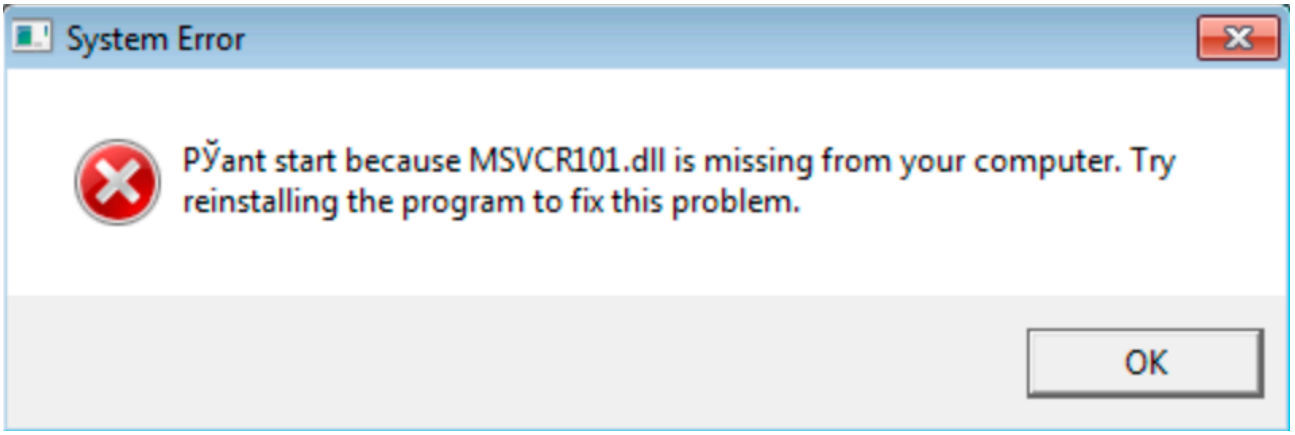
So the final deobfuscated but still not annotated code can be found here. I will break it down into the most interesting things since it will be too much otherwise.

Analysis

The sample contains several anti-sandbox tricks and uses WMI and WSH objects to perform them. If one of those anti sandbox tricks succeed, the script will call a clean up routine which looks as follows (I have annotated the function accordingly for better readability):

```
 Function clean_up_routine()
    send_http_get_request("none")
    delete_itself
    print_fake_message
    WScript.Quit
End Function
```

It's sending a HTTP GET request to `none` (for whatever reason), deleting itself and showing a fake error message in a message box:



In the following, I explain the functions in the order in which they are called.

1. Anti Sandbox - Check physical space

The first function `NoSkh()` is calling the clean up routine when the file `"%USERPROFILE%\Downloads\614500741.txt"` is already there or when your TotalPhysicalMemory is smaller than 1GB.

2. Anti Sandbox - Check Disk space

If your TotalPhysicalMemory is bigger than 1GB, the next function `vgdKyGt()` is called which is terminating the script if your total disk space is smaller than 60GB.

3. Anti Sandbox - Check country code

When the first two anti sandbox checks were not successful, the next function `ULLhsI()` is called. It checks your configured country code at `"HKEY_CURRENT_USER\Control Panel\International\Geo\Nation"`. If your nation key

is configured to `203`, which is Russia, the script is terminating with its clean up routine. Otherwise it will proceed.

4. Anti Sandbox - Check LastBootUpTime

The next function `OUbPa()` checks how long your machine is already running. Therefor, it's checking the LastBootUpTime via WMI and if it's less than 10 minutes, it will terminate calling its clean up routine.

5. Anti Sandbox - Check Processes

Since the malware does not want to run on an analyst system the function `confidante615()` is checking for specific processes from analysis tools.

```
rZRjk = Array("frida-winjector-helper-64.exe","frida-winjector-helper-32.exe","pythonw.exe","pyw
```

If there is such a process, it's terminating with its clean up routine. Additionally, it will terminate if there are less than 28 processes running on the system.

Finally..

The next function `qlqDsdN()` is terminating if the file `%TEMP%\microsoft.url` exists. If not, it creates a shortcut file `%TEMP%\adobe.url` which points to `https://adobe.com` (No idea why. If someone knows, please tell me. Maybe a red herring but nobody is looking into the %TEMP% folder, so why!?).

The function `WjwMtT()` is making use of the before mentioned `kuHKE()` function to write a large byte array to a zip file `%TEMP%\Monica.zip`. Inside `Monica.zip`, there are three files:

- accouter.dxf (the final payload)
- inhibitory.tif (contains part of a string which may be used from `accouter.dfx`)
- isolate.woff (the other part of a string which may be used from `accouter.dfx`)

`bluish578()` copies the three items of `Monica.zip` into `%TEMP%`, deletes `Monica.zip` and `gMcKFIz()` ` finally executes the file `accouter.dxf` which was before copied from Monica.zip into `%TEMP%`.

Execution is performed via `rundll32`:

```
sXmEKs.Create "rundll32" + " " + Get_Temp_Folder + "accouter.dxf" + ",DllRegisterServer"
```

The dropped file `accouter.dfx` can be found on VT and it seems like its Ursnif.

IOCs:

```
 fd490c7b728af08052cf4876c1fc8c6e290bde368b6343492d60fc9d8364a7e5
%TEMP%\adobe.url
%TEMP%\Monica.zip
%TEMP%\accouter.dfx
%TEMP%\inhibitory.tif
%TEMP%\isolate.woff

ed7d22c2f922df466fda6914eb8b93cc27c81f16a60b7aa7eac9ca033014c22c
```

Source: https://malware.love/malware_analysis/reverse_engineering/2020/11/27/analyzing-a-vbs-dropper.html