COMPASS SECURITY BLOG

Offensive Defense

Reversing obfuscated passwords

MAY 23, 2019 / SYLVAIN HEINIGER / 2 COMMENTS

During security assessments (internal penetration tests or Windows client hardening) at our customers, we often find configuration files with a content resembling:

```
UserName = COMPANY\service_user
Password = S0mE+b4sE/64==
```

Against security best practices, service accounts often have more privileges than what they really need or have different policies applied to them than a user account. This makes them interesting targets for attackers.

Several antivirus rely on obfuscation to store passwords:

- McAfee (SiteList.xml)
- Sophos Enterprise Console

Case study: Sophos AutoUpdate

Sophos AutoUpdate is part of the Sophos Endpoint Security and Control solution. It is responsible for connecting to a network share or web server and downloading the latest virus signatures.

According to Sophos, the so-called Sophos Update Manager account should not be and administrative account and should only have read and execute rights on the target resource.

However, in real-life situations, Compass analysts find that such accounts have often too many rights. Hence, it might be interesting to reverse the obfuscated string and recover the password in order impersonate this user.

Is the password secure?

The configuration of the Sophos AutoUpdate tool is located under C:\ProgramData\Sophos\AutoUpdate\Config\iconn.cfg:

```
[PPI.WebConfig_Primary]
AllowLocalConfig = 0
AutoDialTimeout =
LocalPath =
DownloadGranularity =
ConnectionAddress =\\someserver\somepath\
UserName = COMPANY\sophos_admin
UserPassword =  GKGYqkTO3VAntZ9L6kR/FWN51GfpvImvPA==
ConnectionType = UNC
UseSophos = 0
AutoDial = 0
BandwidthLimit = 0
PortNumber =
```

A clever hacker could use search engines in order to find such files online…

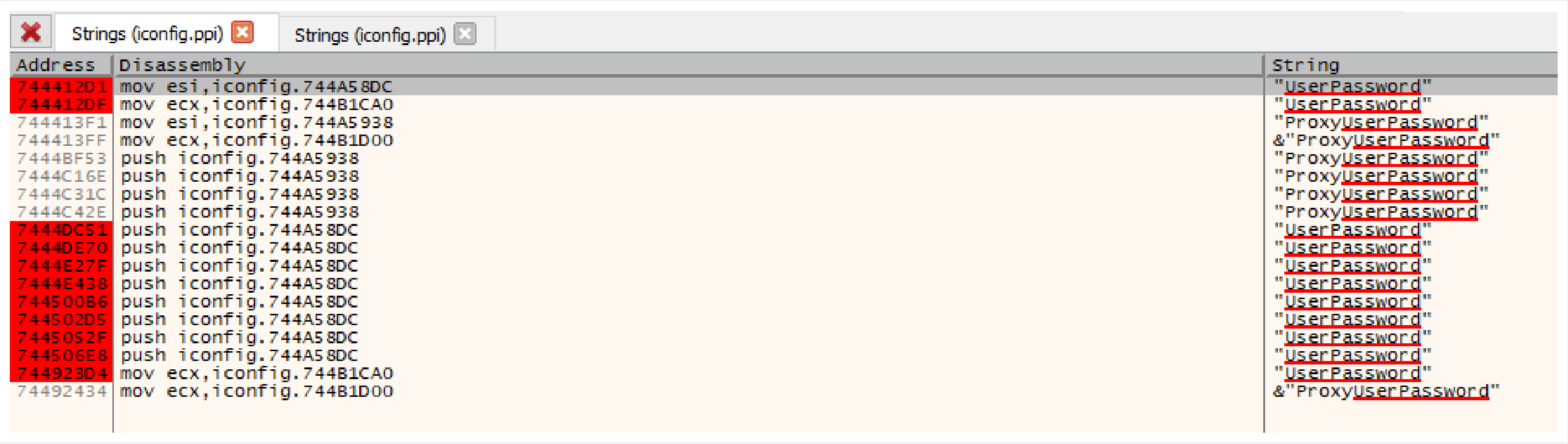
A Sophos Moderator mention on the Sophos Community forums:

> iconn.cfg contains the credentials for accessing the update share. By default this is the so-called Sophos Update Manager account that has limited rights.
>
> […]
> the term for the method used is *obfuscation*, it is of course reversible and just for hiding the credentials from prying eyes.
>
> […]
>
> there is no *secure method of storing* data (password, key, whatever) needed to authenticate against a service/server.
>
> **— QC**

Deobfuscating the password

First solution: debug the program

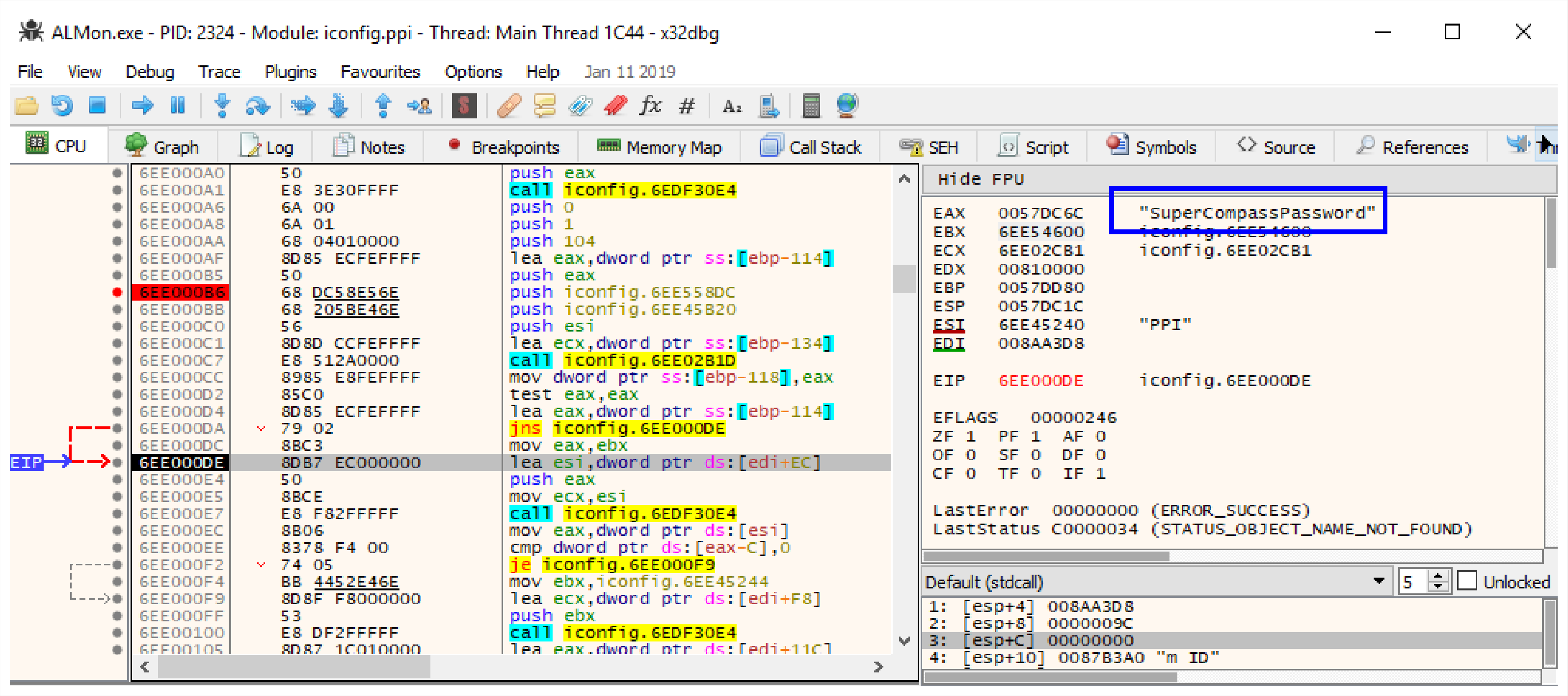
By looking for the string “UserPassword” in our favorite debugger, we find many occurences and add debug points:



Address	Disassembly	String
744412D1	mov esi,iconfig.744A58DC	"UserPassword"
744412DF	mov ecx,iconfig.744B1CA0	"UserPassword"
744413F1	mov esi,iconfig.744A5938	"ProxyUserPassword"
744413FF	mov ecx,iconfig.744B1D00	&"ProxyUserPassword"
7444BF53	push iconfig.744A5938	"ProxyUserPassword"
7444C16E	push iconfig.744A5938	"ProxyUserPassword"
7444C31C	push iconfig.744A5938	"ProxyUserPassword"
7444C42E	push iconfig.744A5938	"ProxyUserPassword"
7444DC51	push iconfig.744A58DC	"UserPassword"
7444DE70	push iconfig.744A58DC	"UserPassword"
7444E27F	push iconfig.744A58DC	"UserPassword"
7444E438	push iconfig.744A58DC	"UserPassword"
744500B6	push iconfig.744A58DC	"UserPassword"
744502D5	push iconfig.744A58DC	"UserPassword"
7445052F	push iconfig.744A58DC	"UserPassword"
744506E8	push iconfig.744A58DC	"UserPassword"
744923D4	mov ecx,iconfig.744B1CA0	"UserPassword"
74492434	mov ecx,iconfig.744B1D00	&"ProxyUserPassword"

x64dbg: find references

After a few steps in the debugger, the deobfuscated password appears:


ALMon.exe - PID: 2324 - Module: iconfig.ppi - Thread: Main Thread 1C44 - x32dbg
File View Debug Trace Plugins Favourites Options Help Jan 11 2019
CPU Graph Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References
Hide FPU
EIP
Default (stdcall)
5
□ Unlocked

```
6EE000A0   50            push eax
6EE000A1   E8 3E30FFFF   call iconfig.6EDF30E4
6EE000A6   6A 00         push 0
6EE000A8   6A 01         push 1
6EE000AA   68 04010000   push 104
6EE000AF   8D85 ECFEFFFF lea eax,dword ptr ss:[ebp-114]
6EE000B5   50            push eax
6EE000B6   68 DC58E56E   push iconfig.6EE558DC
6EE000BB   68 205BE46E   push iconfig.6EE45B20
6EE000C0   56            push esi
6EE000C1   8D8D CCFEFFFF lea ecx,dword ptr ss:[ebp-134]
6EE000C7   E8 512A0000   call iconfig.6EE02B1D
6EE000CC   8985 E8FEFFFF mov dword ptr ss:[ebp-118],eax
6EE000D2   85C0          test eax,eax
6EE000D4   8D85 ECFEFFFF lea eax,dword ptr ss:[ebp-114]
6EE000DA   79 02         jns iconfig.6EE000DE
6EE000DC   8BC3          mov eax,ebx
6EE000DE   8DB7 EC000000 lea esi,dword ptr ds:[edi+EC]
6EE000E4   50            push eax
6EE000E5   8BCE          mov ecx,esi
6EE000E7   E8 F82FFFFF   call iconfig.6EDF30E4
6EE000EC   8B06          mov eax,dword ptr ds:[esi]
6EE000EE   8378 F4 00    cmp dword ptr ds:[eax-C],0
6EE000F2   74 05         je iconfig.6EE000F9
6EE000F4   BB 4452E46E   mov ebx,iconfig.6EE45244
6EE000F9   8D8F F8000000 lea ecx,dword ptr ds:[edi+F8]
6EE000FF   53            push ebx
6EE00100   E8 DF2FFFFF   call iconfig.6EDF30E4
6EE00105   8D87 1C010000 lea eax,dword ptr ds:[edi+11C]
```

```
EAX   0057DC6C     "SuperCompassPassword"
EBX   6EE54600     iconfig.6EE54600
ECX   6EE02CB1     iconfig.6EE02CB1
EDX   00810000
EBP   0057DD80
ESP   0057DC1C
ESI   6EE45240     "PPI"
EDI   008AA3D8

EIP   6EE000DE     iconfig.6EE000DE

EFLAGS   00000246
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)
```

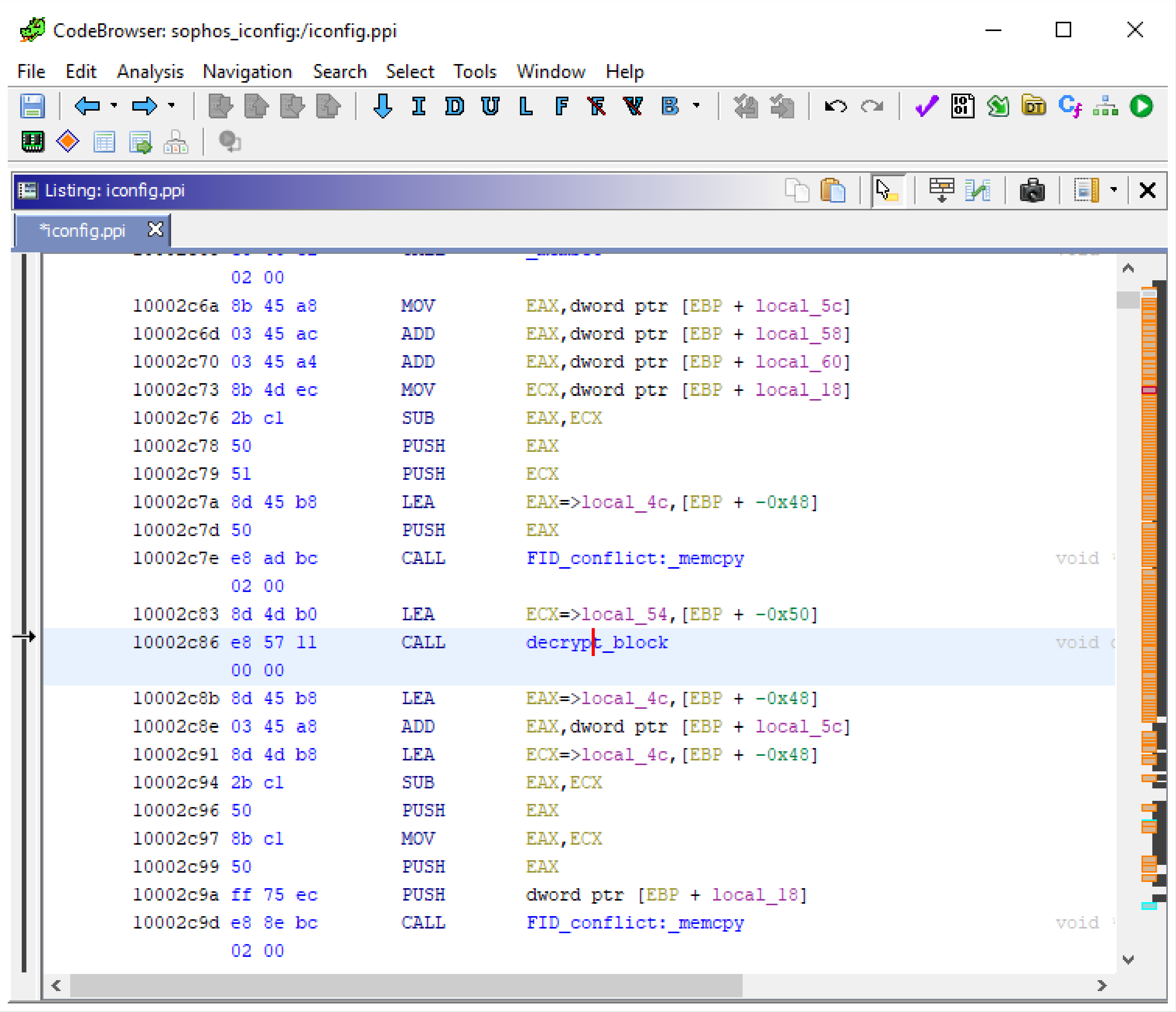
```
1: [esp+4]  008AA3D8
2: [esp+8]  0000009C
3: [esp+C]  00000000
4: [esp+10] 0087B3A0 "m ID"
```

x64dbg: debugging

Time needed: minutes to hours

Second solution: decompile and reverse the algorithm

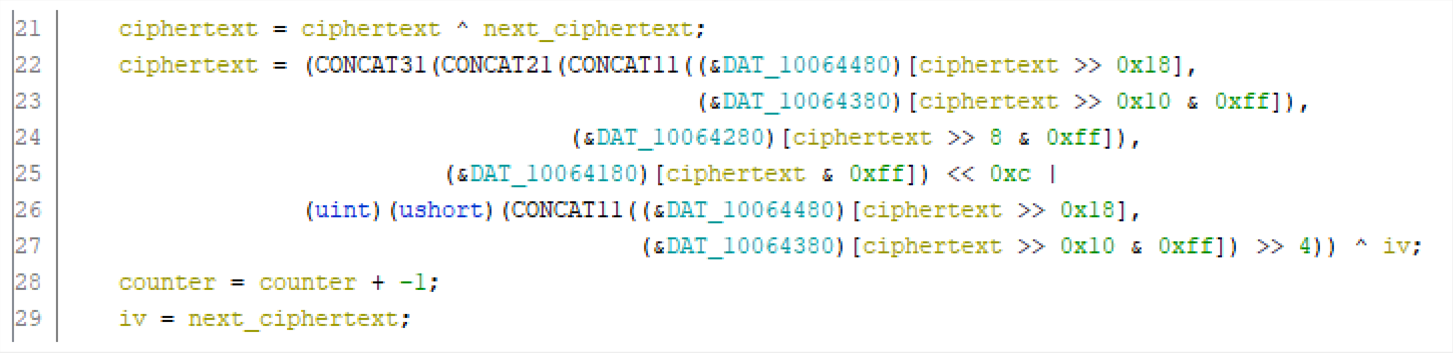
Using the knowledge acquired through our debugging, we know where to look for the decryption code. We use NSA's decompiler Ghidra and identify the portion of the machine code responsible for deobfuscation:



```
                02 00
     10002c6a 8b 45 a8       MOV       EAX,dword ptr [EBP + local_5c]
     10002c6d 03 45 ac       ADD       EAX,dword ptr [EBP + local_58]
     10002c70 03 45 a4       ADD       EAX,dword ptr [EBP + local_60]
     10002c73 8b 4d ec       MOV       ECX,dword ptr [EBP + local_18]
     10002c76 2b c1          SUB       EAX,ECX
     10002c78 50             PUSH      EAX
     10002c79 51             PUSH      ECX
     10002c7a 8d 45 b8       LEA       EAX=>local_4c,[EBP + -0x48]
     10002c7d 50             PUSH      EAX
     10002c7e e8 ad bc       CALL      FID_conflict:_memcpy                      void
              02 00
     10002c83 8d 4d b0       LEA       ECX=>local_54,[EBP + -0x50]
     10002c86 e8 57 11       CALL      decrypt_block                             void
              00 00
     10002c8b 8d 45 b8       LEA       EAX=>local_4c,[EBP + -0x48]
     10002c8e 03 45 a8       ADD       EAX,dword ptr [EBP + local_5c]
     10002c91 8d 4d b8       LEA       ECX=>local_4c,[EBP + -0x48]
     10002c94 2b c1          SUB       EAX,ECX
     10002c96 50             PUSH      EAX
     10002c97 8b c1          MOV       EAX,ECX
     10002c99 50             PUSH      EAX
     10002c9a ff 75 ec       PUSH      dword ptr [EBP + local_18]
     10002c9d e8 8e bc       CALL      FID_conflict:_memcpy                      void
              02 00
```

Ghidra: disassembler window

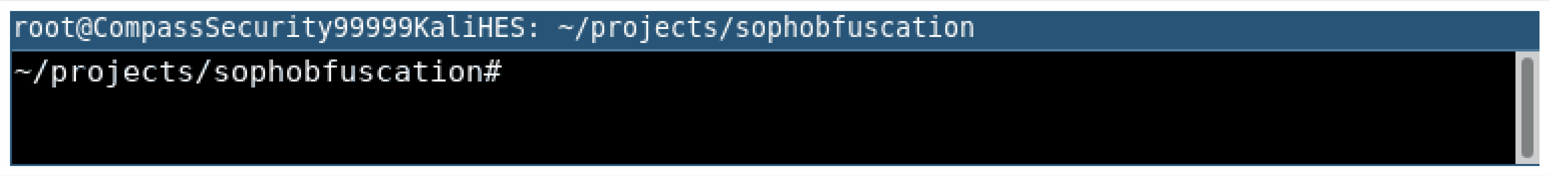
The decompiler’s view of Ghidra provides us with a nice code:



```
    ciphertext = ciphertext ^ next_ciphertext;
    ciphertext = (CONCAT31(CONCAT21(CONCAT11((&DAT_10064480)[ciphertext >> 0x18],
                                             (&DAT_10064380)[ciphertext >> 0x10 & 0xff]),
                                    (&DAT_10064280)[ciphertext >> 8 & 0xff]),
                           (&DAT_10064180)[ciphertext & 0xff]) << 0xc |
                 (uint)(ushort)(CONCAT11((&DAT_10064480)[ciphertext >> 0x18],
                                         (&DAT_10064380)[ciphertext >> 0x10 & 0xff]) >> 4)) ^ iv;
    counter = counter + -1;
    iv = next_ciphertext;
```

Ghidra: decompiler window

We just need to extract some constants from the program and in some 80 lines of python code, we can reproduce the behaviour of the deobfuscation successfully:



```
root@CompassSecurity99999KaliHES: ~/projects/sophobfuscation
~/projects/sophobfuscation#
```

Our deobfuscation script in action

Time needed: hours to days

Bottomline

- Apply the least privilege password to service accounts
- Security through obscurity is not a good solution, obfuscation is no exception
-

Uncategorized

DEOBFUSCATION ELEVATION GHIDRA PASSWORD PRIVILEGE PYTHON X64DBG

PREVIOUS POST
Swiss QR Code Invoices for Phun and Profit

NEXT POST
Practical OpenID Connect Pentesting

2 Comments

Kate
JULY 23, 2019 AT 17:04

Hi,

Do you have any plans to publish the python script you developed for this? I'm sure there are plenty of people on both the sysadmin and pentest side who'd find it very useful.

Cheers,

–Kate

REPLY

Sylvain Heiniger (Post author)
JULY 25, 2019 AT 11:55

Hi Kate,
Unfortunately we wont be releasing the code.
This reversing was part of an assessment with a client and we don't have rights on the software 🙁
However, it's fairly easy to reproduce the steps mentioned in the article! Give it a try if you encounter this during a pentest and don't hesitate to come back to me if anything is unclear.

Cheers

–Sylvain

REPLY

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

COMPASS LINKS

- Legal
- Impressum
- Compass Website
- RSS Feed
- Hacking-Lab
- Swiss Cyber Storm
- FileBox

CATEGORIES

Select Category ▾

UP ↑