Detecting macOS High Sierra root account without authentication

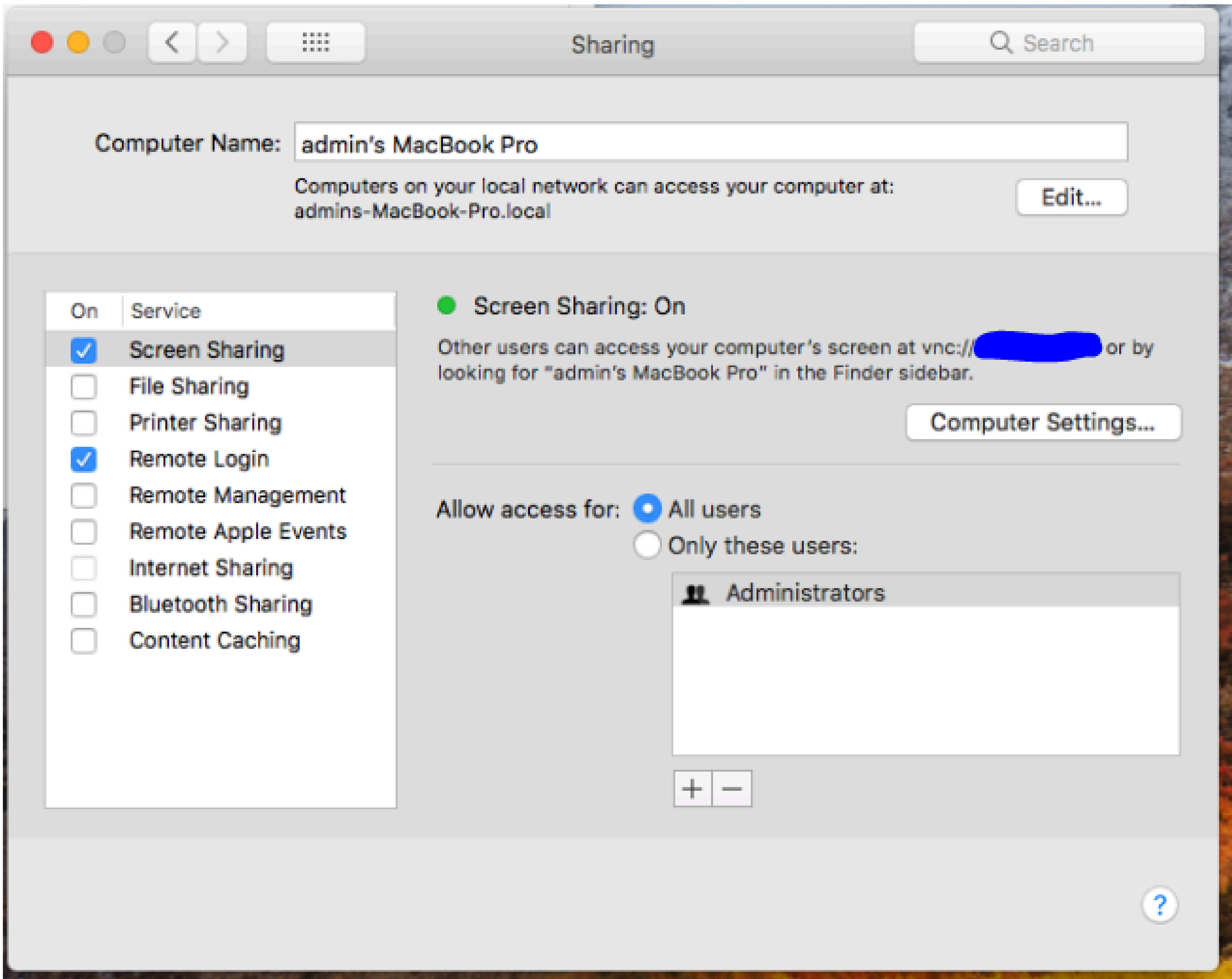
By Nick Miles

Published: 2017-11-30 · Archived: 2026-04-05 15:02:29 UTC

Yesterday, Tenable™ released two plugins to detect macOS High Sierra installs which allow a local user to login as root without a password after several login attempts. Both plugins require authentication, however, there was one scenario where a user could log in over VNC protocol with the root account and no password if screen sharing was enabled. Today, we are releasing a plugin to remotely detect the vulnerability without authentication.

Confirming the Vulnerability

One of my colleagues initially reported that exploitation was possible remotely over VNC after trying against his personal laptop. To confirm the report, I fired up tightVNC (an open source VPN server/client) and tried to exploit the issue on an lab box with "Screen Sharing" enabled (see screenshot below). I ran into a problem were tightVNC couldn't connect to OSX (more on this later). I then tried another VNC client, realVNC, and was able to successfully exploit the issue. After two attempted logins with root and a blank password, the VNC client drops you to a desktop on the remote host, as *root*. Now it's time to look into the VNC protocol, and figure out how we can write a remote check for this!



Delving into the VNC Protocol

Anytime you want to learn a new protocol, a good place to start is the RFC. The RFC for VNC can be found here: https://tools.ietf.org/html/rfc6143

The RFC refers to the protocol as RFB (remote frame buffer). In order to exploit the vulnerability, we need to figure out how to perform authentication over the VNC protocol. The first step is to connect to the host and receive a banner, which looks something like this:

```
RFB protocol version = 3.889
```

Next, we send a similar banner string to the server (ending in a new line), and receive a response for the server that contains the supported authentication types (response decoded below):

```
 Server Auth Types:
30,33,36,35
```

The RFC doesn't mention anything about these authentication types. I needed to figure out what these were, so I loaded up the debug log for realVNC and saw the following:

```
 15> 2017-11-30T00:45:25.175Z TNS5872L vncviewer[13452]: Child: 12148: CProtoPreV5: processing security types me
15> 2017-11-30T00:45:25.175Z TNS5872L vncviewer[13452]: Child: 12148: CProtoPreV5: Server offers security type A
15> 2017-11-30T00:45:25.175Z TNS5872L vncviewer[13452]: Child: 12148: CProtoPreV5: Server offers security type |
15> 2017-11-30T00:45:25.175Z TNS5872L vncviewer[13452]: Child: 12148: CProtoPreV5: Server offers security type |
15> 2017-11-30T00:45:25.175Z TNS5872L vncviewer[13452]: Child: 12148: CProtoPreV5: Server offers security type |
```

So realVNC didn't know what any of these types were, other than type 30, which they have labeled as Ard. A few Google searches later and we found that Ard stands for "Apple Remote Desktop". No wonder tightVNC didn't work, as the application only supports the RFC standard authentication types.

To use that type, we send security type 30 (0x1E) to the server, and extract the response which contains parameters for the authentication. Here is this response in wireshark:

```
Virtual Network Computing
    Generator: 2
    Key length: 128
    Prime modulus: ffffffffffffffffc90fdaa22168c234c4c6628b80dc1cd1...
    Server public key: d3f72ec062839fd1c823a15b6baaf623f2fbb1b989b1e727...

0000  00 05 9a 3c 7a 00 00 11  22 33 44 55 08 00 45 00   ...<z... "3DU..E.
0010  01 2c 00 00 40 00 3c 06  aa c8 ac 1a 00 93 c0 a8   .,..@.<. ........
0020  25 ae 17 0c c3 53 68 ad  40 04 ad ee dd b6 50 18   %....Sh. @.....P.
0030  20 00 f2 5e 00 00 00 02  00 80 ff ff ff ff ff ff    ..^.... ........
0040  ff ff c9 0f da a2 21 68  c2 34 c4 c6 62 8b 80 dc   ......!h .4..b...
0050  1c d1 29 02 4e 08 8a 67  cc 74 02 0b be a6 3b 13   ..).N..g .t....;.
0060  9b 22 51 4a 08 79 8e 34  04 dd ef 95 19 b3 cd 3a   ."QJ.y.4 .......:
0070  43 1b 30 2b 0a 6d f2 5f  14 37 4f e1 35 6d 6d 51   C.0+.m._ .7O.5mmQ
0080  c2 45 e4 85 b5 76 62 5e  7e c6 f4 4c 42 e9 a6 37   .E...vb^ ~..LB..7
0090  ed 6b 0b ff 5c b6 f4 06  b7 ed ee 38 6b fb 5a 89   .k..\... ...8k.Z.
00a0  9f a5 ae 9f 24 11 7c 4b  1f e6 49 28 66 51 ec e6   ....$.|K ..I(fQ..
00b0  53 81 ff ff ff ff ff ff  ff ff d3 f7 2e c0 62 83   S....... ......b.
00c0  9f d1 c8 23 a1 5b 6b aa  f6 23 f2 fb b1 b9 89 b1   ...#.[k. .#......
00d0  e7 27 f2 b6 a1 15 f4 d0  e1 01 5d 69 08 ba e1 73   .'...... ..]i...s
00e0  24 90 de 28 a9 24 20 4f  a9 e7 82 38 38 29 7c 58   $..(.$ O ...88)|X
00f0  1b 4f 3f f6 44 cd 8b ec  f7 31 ca 69 55 d5 ae e1   .O?.D... .1.iU...
0100  c3 ac ba e5 a2 40 ad b5  c4 a8 ce ab 2a 39 b6 98   .....@.. ....*9..
0110  3c fb 29 d3 81 78 3a 36  bd a9 75 de 06 fa 1c 63   <.)..x:6 ..u....c
0120  df 5e ab a6 c1 84 e4 c5  b9 36 f2 cc c6 11 c2 3e   .^...... .6.....>
0130  2c cc 61 2a d8 11 42 00  3e b9                     ,.a*..B. >.
```

Looks like Diffie Hellman! The generator value is always two bytes and is first in the packet. The key length is next and is a two byte integer. The prime modulus and public key follow and are the same size as the key length. So far, the debugging output of the plugin outputs this:

```
 RFB protocol version = 3.889
Server Auth Types:
```

```
30,33,36,35
Doing apple auth!
ARD Material:
  Generator     : 0002
  Key Length    : 128
  Prime Modulus : ffffffffffffffffc90fdaa22168c234c4c6628b80dc1cd129024e088a67cc74020bbea63b139b22514a08798e3404
  Public Key    : a1ef2769ecfa51e2913751a3c51e3fabde0732466915fe0f65cf0aa61f468a929850717f4258a9449da3ba92e3a7ab
```

So, the obvious thing we need to do is generate our own DH key-pair and calculate the shared secret. What's next?

After a bit more Googling and wireshark analysis, we learn that the username and password are sent using AES encryption (using ECB mode), with the key being the MD5 value of the shared secret. The username/password are sent in a 128 byte blob. The first 64 bytes is for the username, the last 64 is for the password. The username and password are null terminated and the remaining space is padded with random bytes. Here is an example of the blob with username admin, password FooBar12 (before we encrypt the data):

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  61 64 6D 69 6E 00 33 37 4F 77 70 45 79 62 35 72  admin.37OwpEyb5r
00000010  63 4A 75 75 45 37 55 32 66 50 7A 31 35 7A 52 58  cJuuE7U2fPz15zRX
00000020  56 6C 56 46 79 77 4A 33 5A 38 63 46 6C 48 65 6F  VlVFywJ3Z8cFlHeo
00000030  46 30 54 67 4D 4A 31 59 77 58 79 4C 69 32 63 55  F0TgMJ1YwXyLi2cU
00000040  46 6F 6F 42 61 72 31 32 00 6F 51 79 39 65 51 41  FooBar12.oQy9eQA
00000050  62 46 65 47 54 62 39 56 39 55 78 67 53 52 32 73  bFeGTb9V9UxgSR2s
00000060  71 49 44 52 61 6A 58 37 50 7A 6A 48 34 31 42 36  qIDRajX7PzjH41B6
00000070  37 35 49 58 43 55 56 52 47 42 61 70 73 79 56 4C  75IXCUVRGBapsyVL
```

To complete the authentication, we need to send the AES encrypted password blob, plus our public DH key and the server will send a four byte integer to indicate success/failure. 0 is success, 1 is failure. Now we can write our remote check!

Exploiting the Vulnerability

Exploiting the vulnerability is easy. We try to log in using root and a blank password multiple times (four times max) using the process described above. If we are successful logging in, then the remote host is vulnerable.

The Nessus® Plugin

Plugin [104885]() was created to exploit the issue remotely over VNC. You must have "safe checks" disabled in order for this run to plugin. This is because successful exploitation will enable the root account, so there is a bit of cleanup involved afterward if you find any affected boxes with the plugin. You need to both disable the root account, and patch the underlying vulnerability.

CRITICAL MacOS root Authentication Bypass Direct check over VNC Server (unauthenticated)

Description

The remote host is running a version of macOS that has a root authentication bypass vulnerability. This plugin tries to exploit this vulnerability remotely over VNC protocol. If it is successful, a root user with blank password will be enabled. This check is only enabled if unsafe checks are enabled. If this plugin is successful, you will need to log in to the target box and disable the root account as well as patch the underlying vulnerability.

Solution

Apply the patch from Apple, or as a workaround, enable the root account and set a strong root account password.

See Also

http://www.nessus.org/u?2cf4b55a
http://www.nessus.org/u?9ff9ff45
http://www.nessus.org/u?1e5890f3
http://www.nessus.org/u?f367aab4
http://www.nessus.org/u?f9f9bbc3

Output

```
Nessus was able to log in to the remote VNC server using the "root" account
and a blank password after exploiting the vulnerability.  You should log in
and disable the root account on the host that this exploit enabled, as well as
patch the vulnerability.
```

Port	Hosts
5900 / tcp / vnc	

Wrap-up

Apple has already released a patch for this vulnerability. This is a critical vulnerability by any standard, so please take all necessary steps to patch your systems as soon as possible. If the patch can't be applied for some reason, please disable screen sharing if it's not needed. If it is needed, then enable the root account and set a strong password.

Nick Miles

Nicholas Miles, Staff Research Engineer, Tenable

Nick Miles has worked for Tenable Research since 2011. He has written hundreds of Nessus plugins, and has published research on multiple vulnerabilities, with a focus on IoT / OT devices. He now works on Tenable's Zero Day research team, where he researches industrial control systems. Nick has a Master's Degree in Computer Engineering, and is an inventor with several patents to his credit.

Source: https://www.tenable.com/blog/detecting-macos-high-sierra-root-account-without-authentication