(2)

Intro to pipes

A (theoretical safe way) to pass data between/among processes stacks.

From session with pts/1

To create a named pipe: BCRL\dennis.guster@eros:~$ mkfifo gpipe

BCRL\dennis.guster@eros:~$ ls -l gp*

prw-r--r-- 1 BCRL\dennis.guster BCRL\domain^users 0 May 22 2017 gpipe

To load a pipe: BCRL\dennis.guster@eros:~$ ps -al > gpipe

From another session:

```
BCRL\de+ 31589  0.0  0.1  92956  4496 ?        S    10:27   0:00 sshd:
BCRL\\dennis.guster@pts/1

BCRL\de+ 31590  0.0  0.1  14896  4588 pts/1    Ss   10:27   0:00 -bash

BCRL\de+ 31781  0.0  0.0  14896  1700 pts/1    S+   10:51   0:00 -bash

BCRL\de+ 31791  0.0  0.0  12728  2136 pts/2    S+   10:53   0:00 grep pts/1
```

To look for the pipe as an open file:

```
BCRL\dennis.guster@eros:~$ lsof -p 31781

COMMAND   PID                USER   FD   TYPE DEVICE SIZE/OFF    NODE NAME

bash    31781 BCRL\dennis.guster  cwd    DIR   0,34    28672 2229102
/rhome/dennis.guster (10.10.3.18:/nfs)

bash    31781 BCRL\dennis.guster  rtd    DIR    8,1     4096       2 /

bash    31781 BCRL\dennis.guster  txt    REG    8,1  1029624  652839 /bin/bash

bash    31781 BCRL\dennis.guster  mem    REG    8,1  1738176  914398
/lib/x86_64-linux-gnu/libc-2.19.so

bash    31781 BCRL\dennis.guster  mem    REG    8,1    14664  914402
/lib/x86_64-linux-gnu/libdl-2.19.so

bash    31781 BCRL\dennis.guster  mem    REG    8,1   171800  914023
/lib/x86_64-linux-gnu/libtinfo.so.5.9

bash    31781 BCRL\dennis.guster  mem    REG    8,1   151120  913993
/lib/x86_64-linux-gnu/libncurses.so.5.9
```

True end to end encryption requires more than just the network. SSl does a credible job of protecting data as it travel across a network. However, with the advent of cloud computing more and more data is being stored in some type of memory and hackers have devised methods to exploit this feature. Too often the primary allocation of resources is placed on protecting the network and intrusion detection systems related to network traffic are mature so hackers often seek alternate means to obtain sensitive data. To illustrate this, a level 3 client server connection (server <----> client) using SSL will be used to illustrate an interesting vulnerability involving piping of data. Let's begin by looking at the server side

1. The server is instantiated below. Note that it creates 3 processes, one to encrypt via a bash, the server itself (on port 8877) and one to de-encrypt on a spate bash.

```
BCRL\dennis.guster@eros:~$ cat encServ
while true; do read -n30 ui; echo $ui |openssl enc -aes-256-ctr -a -k PaSSw;
done | nc -l -p 8877 | while read so; do decoded_so=`echo "$so"| openssl enc
-d -a -aes-256-ctr -k PaSSw`; echo -e "Incoming: $decoded_so" |tee ~/encBuf;
done

BCRL\dennis.guster@eros:~$ ps -al
F S   UID       PID  PPID    C PRI  NI ADDR SZ WCHAN  TTY   TIME CMD
1 S 1018168411 18521 18121  0 80 0 - 3727 -      pts/6    00:00:00 bash
0 S 1018168411 18522 18121  0 80 0 - 1111 -      pts/6    00:00:00 nc
1 S 1018168411 18523 18121  0 80 0 - 3727 -      pts/6    00:00:00 bash
```

2. The client is started below one again three processes are required so that encryption, client access and de-encryption can take place.

```
BCRL\dennis.guster@eros:~$ cat encCli

while true; do read -n30 ui; echo $ui |openssl enc -aes-256-ctr -a -k PaSSw
; done | nc localhost 8877 | while read so; do decoded_so=`echo "$so"|
openssl enc -d -a -aes-256-ctr -k PaSSw`; echo -e "Incoming: $decoded_so";
done

BCRL\dennis.guster@eros:~$ ps -al
F S   UID       PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
1 S 1018168411 18534 18533  0 80 0 - 3728 -      pts/4    00:00:00 bash
0 S 1018168411 18535 18533  0 80 0 - 1164 -      pts/4    00:00:00 nc
1 S 1018168411 18536 18533  0 80 0 - 3728 -      pts/4    00:00:00 bash
```

3. The next step is to verify that the data traveling across the network is in fact encrypted. Note that the un-encrypted payload is set at 30 bytes and the encrypted payload is 65 bytes. The unencrypted payload was a string of the letter "d". The encrypted string which is encrypted using the AES-256 algorithm appears to be very robust. Also, each block of 30 bytes appears to be encrypted differently even though they contain the same payload.

```
BCRL\dennis.guster@eros:~$ sudo tcpdump port 8877 -n -vv -X -c18 -i lo

10:31:26.690882 IP (tos 0x0, ttl 64, id 46862, offset 0, flags [DF], proto
TCP (6), length 117)
    127.0.0.1.53004 > 127.0.0.1.8877: Flags [P.], cksum 0xfe69 (incorrect ->
0x8d80), seq 65:130, ack 1, win 342, options [nop,nop,TS val 1197883276 ecr
1197882694], length 65
        0x0000:  4500 0075 b70e 4000 4006 8572 7f00 0001  E..u..@.@..r....
        0x0010:  7f00 0001 cf0c 22ad 740f 0be7 15b9 1d7b  ......".t......{
        0x0020:  8018 0156 fe69 0000 0101 080a 4766 3f8c  ...V.i......Gf?.
        0x0030:  4766 3d46 5532 4673 6447 566b 5831 3941  Gf=FU2FsdGVkX19A
        0x0040:  7647 7a4e 5a42 5979 5143 4d65 5366 3973  vGzNZBYyQCMeSf9s
        0x0050:  5238 5738 656c 7975 7457 484d 342b 3361  R8W8elyutWHM4+3a
        0x0060:  6a58 384b 3630 3368 6f45 544f 4155 315a  jX8K603hoETOAU1Z
        0x0070:  4738 413d 0a                             G8A=.

10:43:09.946079 IP (tos 0x0, ttl 64, id 46865, offset 0, flags [DF], proto
TCP (6), length 117)
    127.0.0.1.53004 > 127.0.0.1.8877: Flags [P.], cksum 0xfe69 (incorrect ->
0xb92a), seq 65:130, ack 1, win 342, options [nop,nop,TS val 1198059089 ecr
1198058626], length 65
        0x0000:  4500 0075 b711 4000 4006 856f 7f00 0001  E..u..@.@..o....
        0x0010:  7f00 0001 cf0c 22ad 740f 0c96 15b9 1d7b  ......".t......{
        0x0020:  8018 0156 fe69 0000 0101 080a 4768 ee51  ...V.i......Gh.Q
        0x0030:  4768 ec82 5532 4673 6447 566b 5831 2b58  Gh..U2FsdGVkX1+X
        0x0040:  7442 7a75 422f 4339 5832 674b 6633 2b75  tBzuB/C9X2gKf3+u
        0x0050:  366f 614d 3777 4558 7366 7473 6764 5849  6oaM7wEXsftsgdXI
        0x0060:  4942 6d71 6767 4e75 4950 4e74 4865 6b43  IBmqggNuIPNtHekC
        0x0070:  5041 413d 0a                             PAA=..
```

4. Next we look at the open files related to the 2nd bash process which is responsible for the de-encryption process and hence the best place for a hacker to search for the data being xmitted. In an effort to help in tracing the data a tee command was placed in the server code. This allow us to still see the data after it is un encrypted to appear on the server console as well as attempting to create a log of the transaction. However, the attempted log attempt does not show up as an open file either as a container or after data is suppose to be placed in it. A look at the open files reveals no reference to any

active data containers. However, a typical safe place to hold data is in a pipe. Especially an unnamed pipe as appears below. This makes it difficult to find the data and complicated searches using traditional memory maps and registers.

```
BCRL\dennis.guster@eros:/proc/18523$ lsof -p 18523
COMMAND   PID                USER   FD   TYPE DEVICE SIZE/OFF    NODE NAME
bash    18523 BCRL\dennis.guster  cwd    DIR   0,33    28672 2229102
/rhome/dennis.guster (10.10.3.18:/nfs)
bash    18523 BCRL\dennis.guster  rtd    DIR    8,1     4096       2 /
bash    18523 BCRL\dennis.guster  txt    REG    8,1  1029624  652839
/bin/bash
bash    18523 BCRL\dennis.guster  mem    REG    8,1  1738176  915706
/lib/x86_64-linux-gnu/libc-2.19.so
bash    18523 BCRL\dennis.guster  mem    REG    8,1    14664  916906
/lib/x86_64-linux-gnu/libdl-2.19.so
bash    18523 BCRL\dennis.guster  mem    REG    8,1   171800  914023
/lib/x86_64-linux-gnu/libtinfo.so.5.9
bash    18523 BCRL\dennis.guster  mem    REG    8,1   151120  913993
/lib/x86_64-linux-gnu/libncurses.so.5.9
bash    18523 BCRL\dennis.guster  mem    REG    8,1   140928  915700
/lib/x86_64-linux-gnu/ld-2.19.so
bash    18523 BCRL\dennis.guster  mem    REG    8,1   217032  393509
/var/cache/nscd/passwd
bash    18523 BCRL\dennis.guster  mem    REG    8,1  1607712  792355
/usr/lib/locale/locale-archive
bash    18523 BCRL\dennis.guster  mem    REG    8,1    26258  794070
/usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache
bash    18523 BCRL\dennis.guster    0r   FIFO    0,8      0t0 2038584 pipe
bash    18523 BCRL\dennis.guster    1u   CHR  136,6      0t0       9
/dev/pts/6
bash    18523 BCRL\dennis.guster    2u   CHR  136,6      0t0       9
/dev/pts/6
bash    18523 BCRL\dennis.guster  255u   CHR  136,6      0t0       9
/dev/pts/6
```

5. However, the LINUX operating system is full of useful tools originally devised to provide low level diagnostics. Strace is on such tool. In the example below where strace is attached to the bash processed one can see the encrypted date come in from standard IO one character at a time until 65 is reached the last character being the control character for newline. After that the actual data a string of d's appears. Because it is sent in 30 byte blocks it shows up as being 31 bytes in length (+1 for a new line). Note that it is being pull from a pipe (device 3). Also, the file descripter (FD) is set for read (r) which indicates only the ability to pull from the pipe not insert into the pipe. Further, while the pipe is actually a memory buffer, there is no user level address reference to

that buffer because it is stored in kernel level memory space which should be the safest memory ring.

```
BCRL\dennis.guster@eros:~$ strace -p 18523

read(0, "U", 1)                         = 1
read(0, "2", 1)                         = 1
read(0, "F", 1)                         = 1
read(0, "s", 1)                         = 1
read(0, "d", 1)                         = 1
read(0, "G", 1)                         = 1
read(0, "V", 1)                         = 1
read(0, "k", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "1", 1)                         = 1
read(0, "/", 1)                         = 1
read(0, "A", 1)                         = 1
read(0, "R", 1)                         = 1
read(0, "P", 1)                         = 1
read(0, "o", 1)                         = 1
read(0, "Z", 1)                         = 1
read(0, "L", 1)                         = 1
read(0, "I", 1)                         = 1
read(0, "q", 1)                         = 1
read(0, "d", 1)                         = 1
read(0, "v", 1)                         = 1
read(0, "K", 1)                         = 1
read(0, "I", 1)                         = 1
read(0, "G", 1)                         = 1
read(0, "y", 1)                         = 1
read(0, "L", 1)                         = 1
read(0, "6", 1)                         = 1
read(0, "E", 1)                         = 1
read(0, "F", 1)                         = 1
read(0, "O", 1)                         = 1
read(0, "0", 1)                         = 1
read(0, "1", 1)                         = 1
read(0, "1", 1)                         = 1
read(0, "q", 1)                         = 1
read(0, "V", 1)                         = 1
read(0, "g", 1)                         = 1
read(0, "p", 1)                         = 1
read(0, "Y", 1)                         = 1
read(0, "L", 1)                         = 1
read(0, "i", 1)                         = 1
read(0, "+", 1)                         = 1
```

```
read(0, "3", 1)                                 = 1
read(0, "m", 1)                                 = 1
read(0, "d", 1)                                 = 1
read(0, "H", 1)                                 = 1
read(0, "H", 1)                                 = 1
read(0, "i", 1)                                 = 1
read(0, "G", 1)                                 = 1
read(0, "o", 1)                                 = 1
read(0, "W", 1)                                 = 1
read(0, "5", 1)                                 = 1
read(0, "F", 1)                                 = 1
read(0, "r", 1)                                 = 1
read(0, "6", 1)                                 = 1
read(0, "I", 1)                                 = 1
read(0, "N", 1)                                 = 1
read(0, "I", 1)                                 = 1
read(0, "3", 1)                                 = 1
read(0, "R", 1)                                 = 1
read(0, "z", 1)                                 = 1
read(0, "j", 1)                                 = 1
read(0, "5", 1)                                 = 1
read(0, "0", 1)                                 = 1
read(0, "=", 1)                                 = 1
read(0, "\n", 1)                                = 1

pipe([3, 4])                                    = 0
read(3, "ddddddddddddddddddddddddddddddd\n", 128) = 31
```

6. However, the date could also be compromised from the client side as well. The first step is to start the server as before (on pts/0):

```
BCRL\dennis.guster@eros:~$ ./encServ

Incoming: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Incoming: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

7. The next steps is to determine the appropriate process ID to trace so that the encrypted data can be read from the unnamed pipe before it is forwarded to the encryption algorithm which will be a subsequent process. In the example below the client is running on pts/1 and the process ID to be evaluated is 27180. From the screen with the running client code on can see that a string of "X"s was entered and the strace command is able to pick them up unencrypted.

```
BCRL\dennis.guster@eros:~$ ps -al
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
1 S 1018168411 27169 27131  0 80  0 - 3737 -      pts/0    00:00:00 bash
1 S 1018168411 27170 27169  0 80  0 - 3737 -      pts/0    00:00:00 bash
```

```
0 S 1018168411 27171 27169  0 80 0 - 1111 -     pts/0    00:00:00 nc
1 S 1018168411 27172 27169  0 80 0 - 3737 -     pts/0    00:00:00 bash
1 S 1018168411 27179 27141  0 80 0 - 3737 -     pts/1    00:00:00 bash
1 S 1018168411 27180 27179  0 80 0 - 3737 -     pts/1    00:00:00 bash
0 S 1018168411 27181 27179  0 80 0 - 1164 -     pts/1    00:00:00 nc
1 S 1018168411 27182 27179  0 80 0 - 3737 -     pts/1    00:00:00 bash
0 S 1018168411 27189 27154  0 80 0 - 1255 -     pts/2    00:00:00 strace
0 R 1018168411 27211 27163  0 80 0 - 2672 -     pts/3    00:00:00 ps


BCRL\dennis.guster@eros:~$ ./encCli

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

BCRL\dennis.guster@eros:~$ strace -p 27180
Process 27180 attached
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
read(0, "X", 1)                         = 1
```

8. Note that there is an unnamed pipe associated with the data storage as in the server side, but its file descripter is set to write (w) so that data can be placed in the pipe and passed to a subsequent process that will actually encrypt the data before passing it to the network layer socket.

```
BCRL\dennis.guster@eros:~$ lsof -p 27180
COMMAND   PID                USER   FD   TYPE DEVICE SIZE/OFF    NODE NAME
bash    27180 BCRL\dennis.guster    1w  FIFO    0,8      0t0  590652 pipe
```

9. By looking at process ID 27181 we can see that the data being passed from the unnamed pipe (device 0) as well as the data being written to an unnamed pipe and being passed to the next process in the process stack are both encrypted. To accomplish this 2 unnamed pipes are required and appear in the lsof output below.

```
BCRL\dennis.guster@eros:~$ strace -p 27181
Process 27181 attached
select(4, [0 3], NULL, NULL, NULL)      = 1 (in [0])
read(0, "U2FsdGVkX1/JvQ/9VR3R2upy8pXnlxeP"..., 8192) = 65
write(3, "U2FsdGVkX1/JvQ/9VR3R2upy8pXnlxeP"..., 65) = 65
```

```
BCRL\dennis.guster@eros:~$ lsof -p 27181
COMMAND   PID                USER   FD   TYPE DEVICE SIZE/OFF    NODE NAME
nc      27181 BCRL\dennis.guster    0r  FIFO    0,8      0t0  590652 pipe
nc      27181 BCRL\dennis.guster    1w  FIFO    0,8      0t0  590653 pipe
```