Installation

- Installing OpenBSD 6.9 on a laptop with encryption

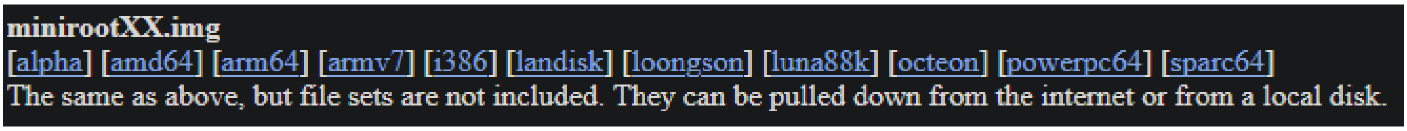
Installing OpenBSD 6.9 on a laptop with encryption

Lately, I've been looking into some old laptops which I could spend my upcoming college years with. I wanted something cheap, but also good enough that I could use it for basic university related work. Of couse I could just buy a new low-end Windows laptop and be happy, but that would be a waste of money. Instead, I searched through used market for an old ThinkPad, T500 in particular. If you are a bit confused or haven't heard of Lenovo's ThinkPads and why they have a special place among many, I recommend you check out this guide. There are many newer models than the 2009's T500, however T500 seems to be the last "*Librebootable*" 15" ThinkPad. Flashing Libreboot is something I would love to do as well, but currently don't have the right tools to complete the process documented on the Libreboot wiki here. After T500 there are still a couple of pretty decent ThinkPad models, altough it is not possible to install completely Libre firmware on them.

Downloading OpenBSD

Downloading OpenBSD is pretty straightforward, just prepare two USB drives – one which we will boot the installer from and the other for our encryption key.

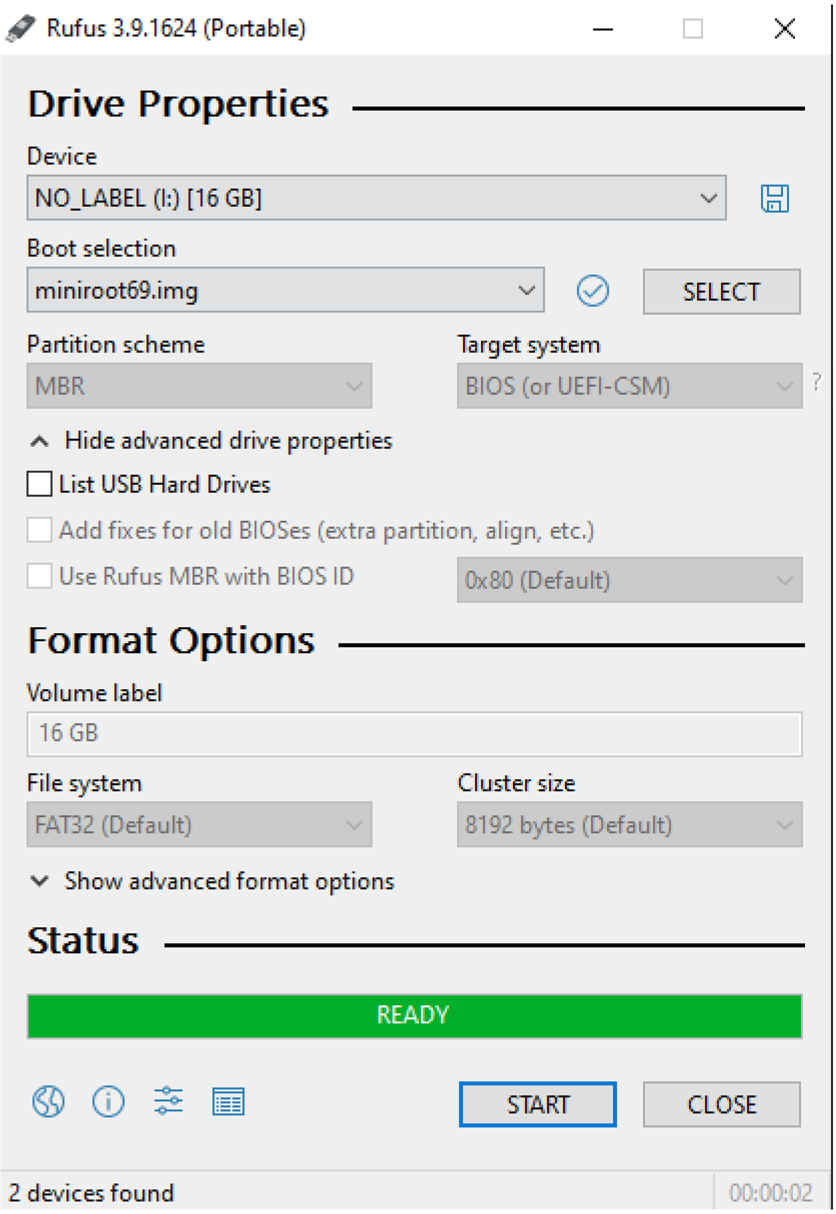
Head over to the download page and click on minirootXX.img



minirootXX.img
[alpha] [amd64] [arm64] [armv7] [i386] [landisk] [loongson] [luna88k] [octeon] [powerpc64] [sparc64]
The same as above, but file sets are not included. They can be pulled down from the internet or from a local disk.

If you have a 64-bit machine, which the T500 is, select *amd64*. This will be a minimal intallation and we will only download file sets that we want during the install. If you do not care about this, you can pick one with the file sets already included.

For those unfortunate people who still use Windows on their main machine like me, you may use utility like Rufus to burn the image to a USB. Just pick the file and drive, make sure to choose the right one, this will **overwrite all data** on the USB drive.



Device: NO_LABEL (I:) [16 GB]

Boot selection: miniroot69.img

Partition scheme: MBR

Target system: BIOS (or UEFI-CSM)

▲ Hide advanced drive properties

□ List USB Hard Drives

□ Add fixes for old BIOSes (extra partition, align, etc.)

□ Use Rufus MBR with BIOS ID 0x80 (Default)

Volume label: 16 GB

File system: FAT32 (Default)

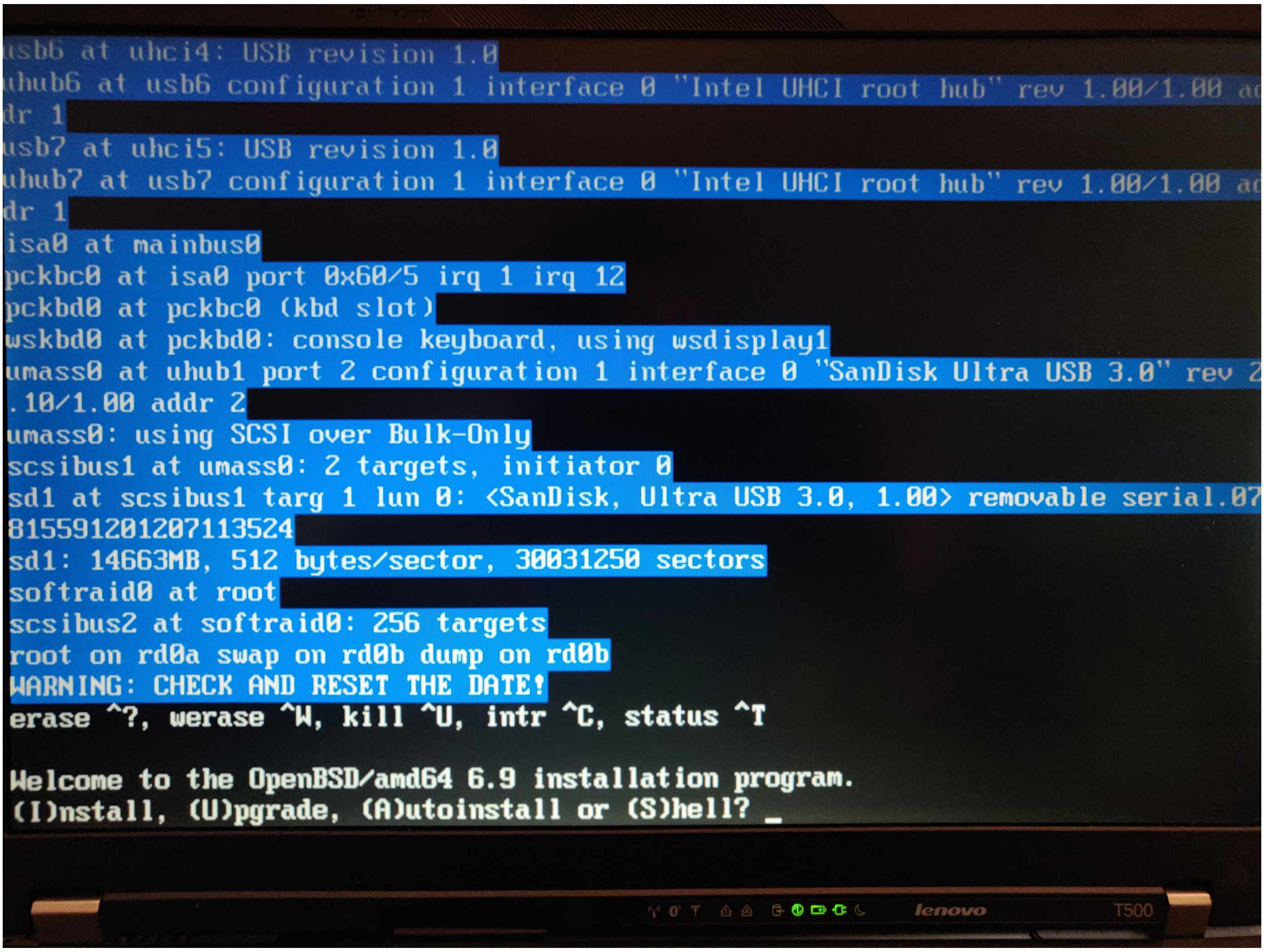
Cluster size: 8192 bytes (Default)

▼ Show advanced format options

READY

2 devices found 00:00:02

Once we have the USB ready, we can plug it into the T500, press F12 and choose to boot from it. We will be greeted by this screen:



```
usb6 at uhci4: USB revision 1.0
uhub6 at usb6 configuration 1 interface 0 "Intel UHCI root hub" rev 1.00/1.00 ad
dr 1
usb7 at uhci5: USB revision 1.0
uhub7 at usb7 configuration 1 interface 0 "Intel UHCI root hub" rev 1.00/1.00 ad
dr 1
isa0 at mainbus0
pckbc0 at isa0 port 0x60/5 irq 1 irq 12
pckbd0 at pckbc0 (kbd slot)
wskbd0 at pckbd0: console keyboard, using wsdisplay1
umass0 at uhub1 port 2 configuration 1 interface 0 "SanDisk Ultra USB 3.0" rev 2
.10/1.00 addr 2
umass0: using SCSI over Bulk-Only
scsibus1 at umass0: 2 targets, initiator 0
sd1 at scsibus1 targ 1 lun 0: <SanDisk, Ultra USB 3.0, 1.00> removable serial.07
815591201207113524
sd1: 14663MB, 512 bytes/sector, 30031250 sectors
softraid0 at root
scsibus2 at softraid0: 256 targets
root on rd0a swap on rd0b dump on rd0b
WARNING: CHECK AND RESET THE DATE!
erase ^?, werase ^W, kill ^U, intr ^C, status ^T

Welcome to the OpenBSD/amd64 6.9 installation program.
(I)nstall, (U)pgrade, (A)utoinstall or (S)hell? _
```

Installing OpenBSD

Full disk encryption with key (FDE)

I'm using a laptop, so it would be advisable to implement some sort of encryption in case it gets lost or someone steals it. We will follow the official guide here (with a little modification) and setup FDE with a USB drive for convenience, so that we don't have to type the encryption password every time and instead rely on the possesion of the USB drive.

Jump out of the installer into the shell by typing S

Begin the proces by creating a device node. You may be familiar with the naming convetion from Linux. sd is essentially SCSI disk driver. Here's a little comparasion of naming drives Linux vs OpenBSD:

	Linux	OpenBSD
First hard drive	/dev/sda	/dev/sd0

First partition of the second hard drive	/dev/sdb1	/dev/sd1a
Partition naming	/dev/sda1, /dev/sda2, /dev/sda3, etc.	/dev/sd0a, /dev/sd0b, /dev/sd0c

As you can see, it's similar but not the same – Linux names devices using letters and partitions are named by numbers, OpenBSD does the opposite.

```
# cd /dev && sh MAKEDEV sd0
```

We may also want to write some random data to the device, this is a fairly time consuming process, so get some coffee and do some work while it does it's thing. The bigger the drive, the longer this will take. The computer's speed also plays a big role. *Wait, what is rsd0c? Haven't we just typed in sd0? Why are we writing something to rsd0c?* I had the same questions, but the explanation is quite simple. *R* in the beginning stands for *raw*, as it is a raw (character) device. sd0 was already explained above (first SCSI driver device), but why *c* at the end? According to disklabel(5) "*The ‘c’ partition is reserved for the entire physical disk*"*.* So there we have it, rsd0c points to the *whole raw first hdd handled by the SCSI driver,* to which we are currently writing random data.

```
# dd if=/dev/urandom of=/dev/rsd0c bs=1m
```

Since we have *a crap load of time* before it finishes, let's dissect the command a little further. *dd* is a well known utility from the world of Linux and other UNIX-like operating systems. It can be used to copy *standard input* (stdin) *to standard output (stdout)*. These two aforementioned terms usually mean "what you type" and "what you get" to/from the terminal. In this case we are using the *if* and *of* options of *dd* which replace stdin and stdout with files. Basically copying the contents of `/dev/urandom` to `/dev/rsd0c`. As you can tell by the fact that *urandom* resides in /dev, it is some kind of a device, but not a physical one. Together with *random, urandom* is a data source device which provides high quality *pseudo-random* data. *Why is it called pseudo-random?* Well, computers can't actually produce truly random data (but they can use realistically random sources), instead, the kernel utilizes all sorts of different things that are happening at any given moment (system activity, network, hadware output) and through some programming and math trickery output seemingly random data. It is important to make sure that the same data cannot be generated again, because that could lead to security issues, since actions like generating SSH key pairs use these *pseudo-random* data sources. *(Imagine you generate an SSH key pair and someone finds a way to get the same pseudo-random output that you got, which will result in the same SSH key pair being generated, even though I'm sure there are reasons why this isn't realistically possible, correct me if I'm wrong about this whole thing).* The last option *bs* specifies the block size (how large the blocks of data will be)

As far as I know, the T500 doesn't support UEFI, so we'll have to use MBR partitioning table. Use fdisk to initialize MBR.

- -i uses default MBR

- -y avoids unnecessary yes/no questions.

```
# fdisk -iy sd0
```

If you have an UEFI device, use this command:

```
# fdisk -iy -g -b 960 sd0
```

To create a partition layout, we need to enter disklabel's label editor with the -E option and name of the device we want to edit.

```
# disklabel -E sd0
```

In the Label editor, enter the following commands *(// respresent comments, don't type that into the terminal)*

```
#sd0> a a // "a" to add partition and name it "a"
offset: [64] // press enter
size: [488391056] // enter
FS type: [4.2BSD] RAID // default is 4.2BSD, but we want to create RAID volume
sd0*> w // write changes to the disk
sd0> q // quit the editor
No label changes.
```

Let's stop for a moment and let me explain the naming and structure. The device that we are now working with is *sd0*. We have just created an *a* parition on *sd0*, which resulted in *sd0a* RAID type filesystem that spans **across the entire disk**. Now we will use RAID management interface called bioctl to create a CRYPTO volume on *sd0a.* This encrypted volume will appear as another device, but it's just an encryption layer on the same physical device. Afterwards we can create volumes and file systems how we want, just like with a regular OpenBSD install without encryption.

If you are using a regular passphrase, the new encrypted device will become *sd1*. However, since we want a keydrive, we have to account for the other USB drive. Connect the USB that you want to use to unlock your laptop/PC and note the assigned name. In my case, it got recognized as ***sd2.*** This is important, becuase now our encrypted volume won't be *sd1*, but ***sd3.***

Assume the key drive is *sd2* and our initial parition is still *sd0*.

- -c sets the RAID level, but we just want an encryption layer, so we will use "C" which stands for CRYPTO
- -k use key disk device sd2a
- -l create volume on sd0a with software raid

This part took me a long time, because I got stuck on the -r option which specifies number of iterations for the KDF algorithm. What got me thinking was this article, which goes in depth into the encryption of OpenBSD. What confused me the most is that according to the bioctl man page, the default number of rounds is 16, however the author of the article mentions "*The interesting part is the number of iterations which directly impact the resistance to brute force attacks. On OpenBSD, it is set to the hard-coded value of **8192**. As discussed previously, this may be changed using bioctl options.*" If we are talking about the same thing, why the man page mentions 16 and he says 8192? Also "*On my home install, my LUKS volume is configured for about 400,000 iterations, which is significantly higher than the OpenBSD default.*". His closing thoughts are "*If using OpenBSD on a recent computer, bump up the number of PBKDF2 iterations when creating the volume.*" What does *bump up* mean in this case? More than 10 000? 100 000? Anyway, take what you want from this, I will try 50 000 and see what happens.

```
# bioctl -c C -r 50000 -k sd2a -l sd0a softraid0
```

Well, this happened.

```
bioctl: could not open sd2a: No such file or directory
```

Of course, we don't have any sd2x devices in /dev, which we can confirm by ls /dev/. Correct that by issuing

```
cd /dev
sh MAKEDEV sd2
```

Aaaand, again we fail because:

```
bioctl: could not open sd2a: Device not configured
```

Sure, even though our key drive showed up as sd2, we haven't touched it with fdisk like we did with our sd0. If I could read, I would know since the guide clearly mentions it "*Initialize your keydisk with fdisk(8), then use disklabel(8) to create a 1 MB RAID partition for the key data*"

I grouped all the commands we need into the following block:

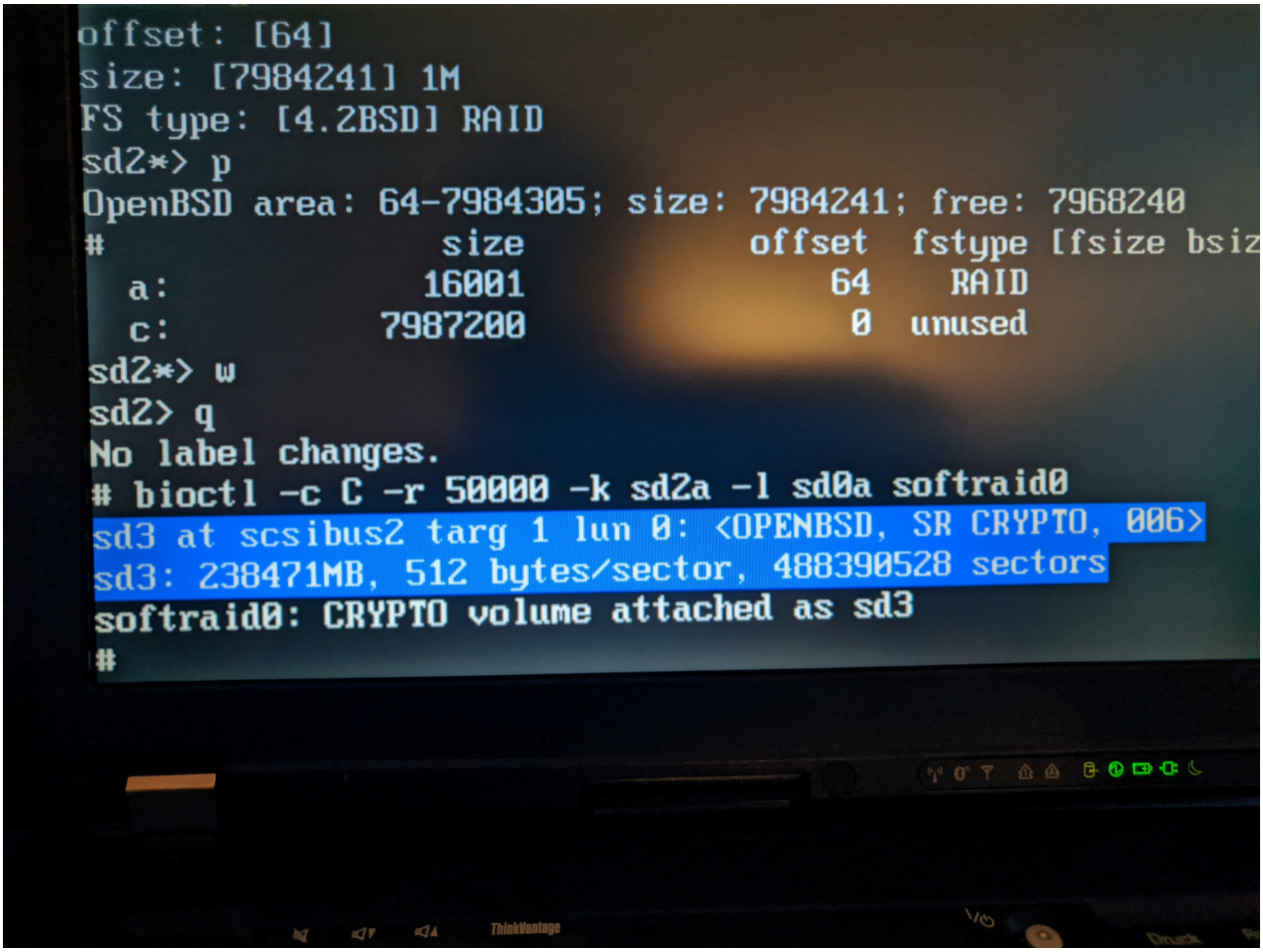
```
# dd if=/dev/urandom of=/dev/rsd2c bs=1m // rsd2c - our raw sd2 key drive
# fdisk -iy sd2
# disklabel -E sd2
sd2> a a // create "a" partition
offset: [64] // accept default with enter
size: [7984241] 1M // we only need a small 1M partition
FS type: [4.2BSD] RAID // again, create RAID type partition
```

```
// we can confirm that everything is ok
> p
#      size  offset  fstype [fsize bsize   cpg]
a:    16001      64    RAID
c:  7987200       0  unused

sd2*> w // write changes
sd2*> q // quit
```

Now we can run bioctl again

```
# bioctl -c C -r 50000 -k sd2a -l sd0a softraid0
```

You should see similar output:



```
offset: [64]
size: [7984241] 1M
FS type: [4.2BSD] RAID
sd2*> p
OpenBSD area: 64-7984305; size: 7984241; free: 7968240
#                size           offset  fstype [fsize bsiz
  a:            16001               64    RAID
  c:          7987200                0  unused
sd2*> w
sd2> q
No label changes.
# bioctl -c C -r 50000 -k sd2a -l sd0a softraid0
sd3 at scsibus2 targ 1 lun 0: <OPENBSD, SR CRYPTO, 006>
sd3: 238471MB, 512 bytes/sector, 488390528 sectors
softraid0: CRYPTO volume attached as sd3
#
```

Please note the volume which was created, in this case sd3, you will need it later!

This is everything we had to do in order to setup encryption. What follows is only the installation process described bellow.

Installation script

Type I to begin the installer. The system will ask you some basic information, stuff in [] is default when you press enter.

You can pick a different keyboard layout, but I will stick with the default one

```
Choose your keyboard layout ('?' or 'L' for list) [default]
```

Pick a hostname that you like

```
System hostname? (short form, e.g. 'foo')
```

We are using a minimal image file, therefore we will need internet connection to download all required file sets, so it makes sense to configure network interface. I will just connect cable to the on board LAN (em0) and let DHCP give it lease. Unless you want to configure IPv6 stick to default none

```
Available network interfaces are: em0 iwn0 vlan0.
Which network interface do you wish to configure? (or 'done') [em0]
IPv4 address form em0? (or 'dhcp' or 'none') [dhcp]
em0: no link...got link
em0: no lease......got lease
em0: 10.5.51.122 lease accepted from 10.1.4.1 (mac address)
IPv6 address for em0 (or 'autoconf' or 'none') [none]
Which network interface do you wish to configure? (or 'done') [done]
Using DNS domainname your.server
Using DNS nameservers at 10.6.8.77
```

Enter you root account, make sure it is a strong password and store it in a secure place. You will mostly interact with the system using a regular user and doas to elevate privileges. **It is not a good security practice to log in with root!**

```
Password for root account? (will not echo)
Password for root account? (again)
```

This isn't going to be a server, so type no to SSH

```
Start sshd(8) by default? [yes] no
```

I personally had some troubles with xenodm, but they mostly came down to me being incompetent. Using xenodm is a recommended security practice, instead of starting X server with startx. Not entirely sure why it's not default.

```
Do you want the X Window System to be started by xenodm(1)? [no] yes
```

Setup a user that you will login with.

```
Setup a user? (enter a lower-case loginname, or 'no') [no] user
```

Timezone settings. If the correct timezone wasn't selected automatically, press ? and pick a correct one and type it into the installer.

```
What timezone are you in? ('?' for list) [Europe/Prague]
```

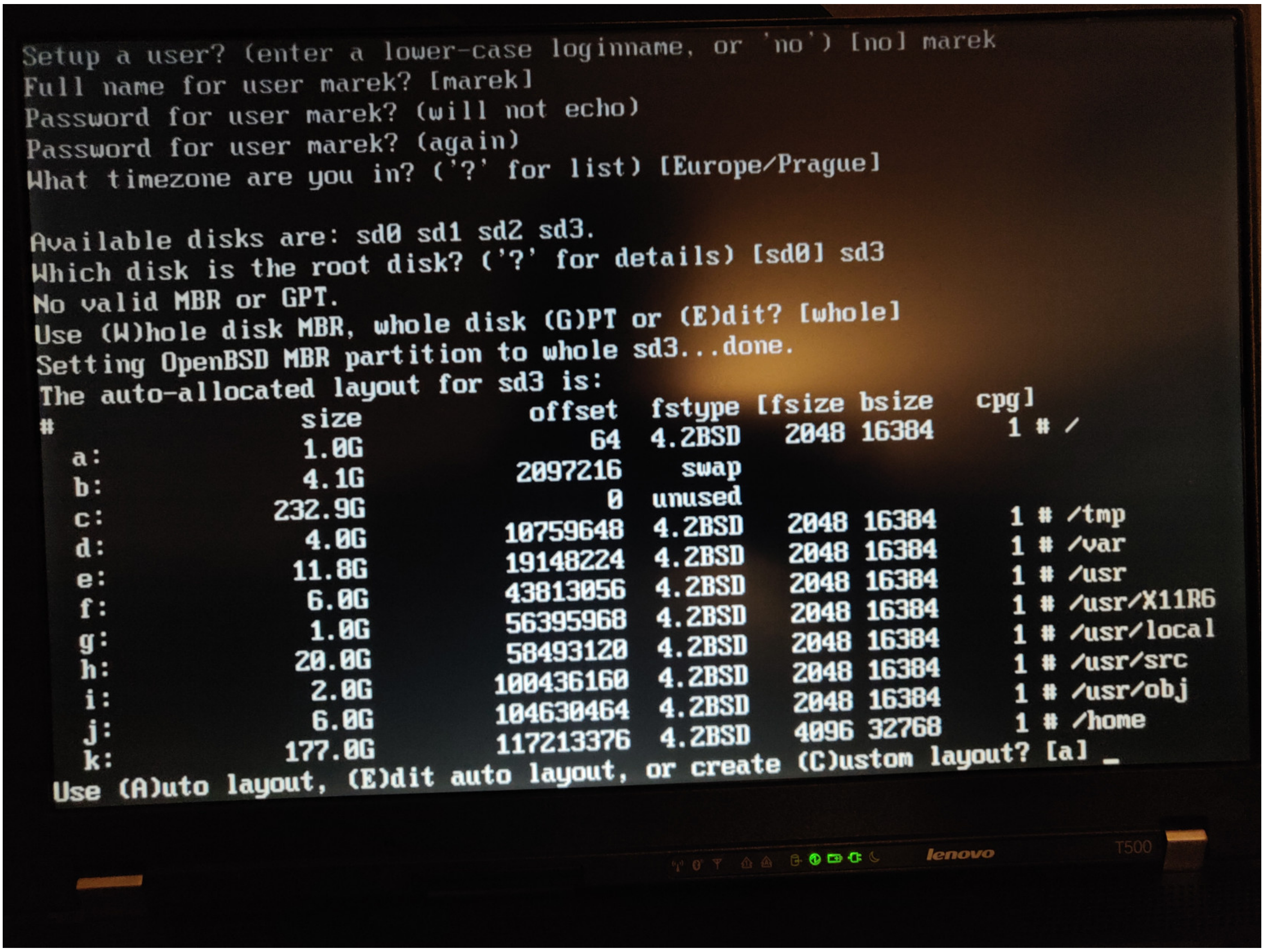
Now make sure you install into the right volume – in our case, *sd3* because it's the crypto volume.

```
Available disks are: sd0 sd1 sd2 sd3
Which disk is the root disk? ('?' for details) [sd0] sd3
```

We have a MBR machine, so pick MBR

```
No valid MBR or GPT
Use (W)whole disk MBR, whole disk (G)PT or (E)dit? [whole]
Setting OpenBSD MBR partition to whole sd3...done.
```

OpenBSD partitioning is something we will leave for another time. This time I'm going with the default which is surprisingly sane.

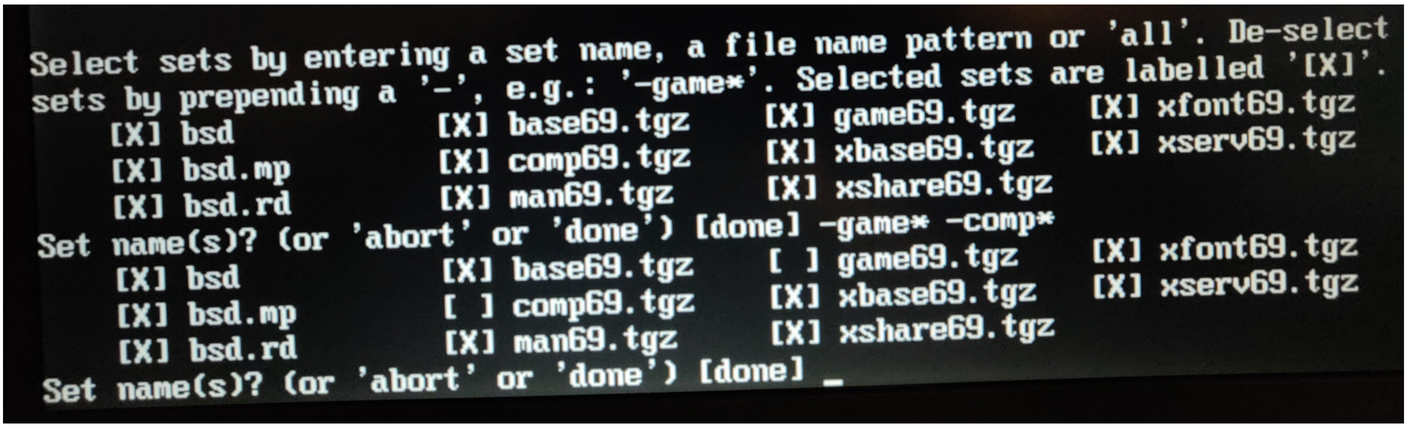


```
Setup a user? (enter a lower-case loginname, or 'no') [no] marek
Full name for user marek? [marek]
Password for user marek? (will not echo)
Password for user marek? (again)
What timezone are you in? ('?' for list) [Europe/Prague]

Available disks are: sd0 sd1 sd2 sd3.
Which disk is the root disk? ('?' for details) [sd0] sd3
No valid MBR or GPT.
Use (W)hole disk MBR, whole disk (G)PT or (E)dit? [whole]
Setting OpenBSD MBR partition to whole sd3...done.
The auto-allocated layout for sd3 is:
#                size           offset  fstype [fsize bsize   cpg]
  a:             1.0G               64  4.2BSD   2048 16384     1 # /
  b:             4.1G          2097216    swap
  c:           232.9G                0  unused
  d:             4.0G         10759648  4.2BSD   2048 16384     1 # /tmp
  e:            11.8G         19148224  4.2BSD   2048 16384     1 # /var
  f:             6.0G         43813056  4.2BSD   2048 16384     1 # /usr
  g:             1.0G         56395968  4.2BSD   2048 16384     1 # /usr/X11R6
  h:            20.0G         58493120  4.2BSD   2048 16384     1 # /usr/local
  i:             2.0G        100436160  4.2BSD   2048 16384     1 # /usr/src
  j:             6.0G        104630464  4.2BSD   2048 16384     1 # /usr/obj
  k:           177.0G        117213376  4.2BSD   4096 32768     1 # /home
Use (A)uto layout, (E)dit auto layout, or create (C)ustom layout? [a] _
```

We are using a minimal instalation file, therefore we need to download required file sets from the internet – http option. We aren't using any proxy servers, so leave that at default. You can also pick a server that is closer to you or that you prefer.

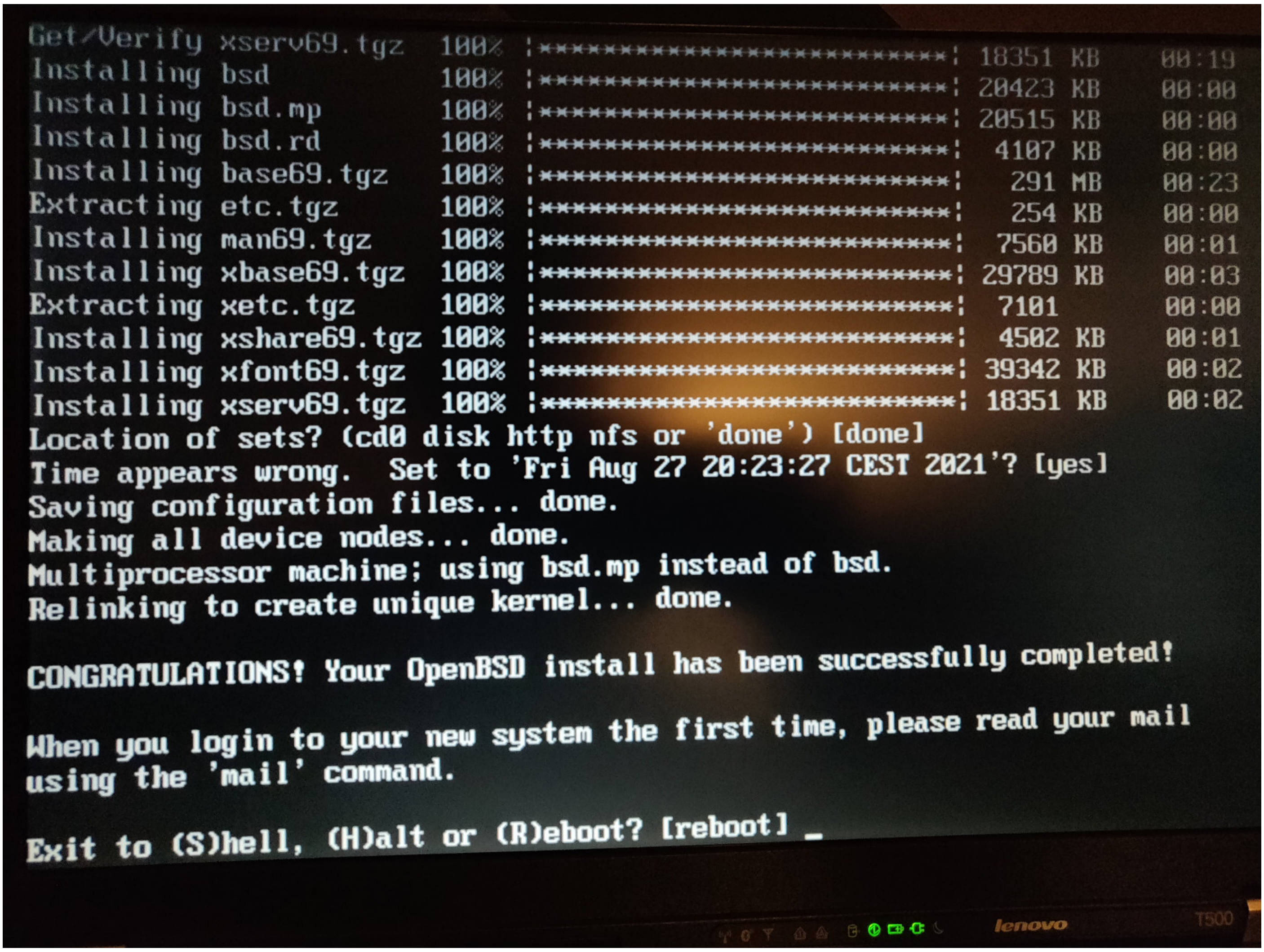
```
Let's install the sets!
Location of sets (cd0 disk http nfs or 'done') [http]
HTTP proxy URL? (e.g. 'http://proxy:8080', or 'none') [none]
HTTP Server? (hostname, list#, 'done' or '?') [ftp.eu.openbsd.org]
Server direcotry? [pub/OpenBSD/6.9/amd64]
```

When it comes to file sets, most of the time it is easiest to just install them all. On a headless server installation, you certainly won't need any sets beginning with ***x***, but we are running a laptop which will need X packages for GUI. I usually leave out game69.tgz and comp69.tgz, but if you aren't constrained by limited disk space available, just leave it at default. To read more about what these file sets are, check out OpenBSD FAQ.



```
Select sets by entering a set name, a file name pattern or 'all'. De-select
sets by prepending a '-', e.g.: '-game*'. Selected sets are labelled '[X]'.
    [X] bsd           [X] base69.tgz    [X] game69.tgz    [X] xfont69.tgz
    [X] bsd.mp        [X] comp69.tgz    [X] xbase69.tgz   [X] xserv69.tgz
    [X] bsd.rd        [X] man69.tgz     [X] xshare69.tgz
Set name(s)? (or 'abort' or 'done') [done] -game* -comp*
    [X] bsd           [X] base69.tgz    [ ] game69.tgz    [X] xfont69.tgz
    [X] bsd.mp        [ ] comp69.tgz    [X] xbase69.tgz   [X] xserv69.tgz
    [X] bsd.rd        [X] man69.tgz     [X] xshare69.tgz
Set name(s)? (or 'abort' or 'done') [done] _
```

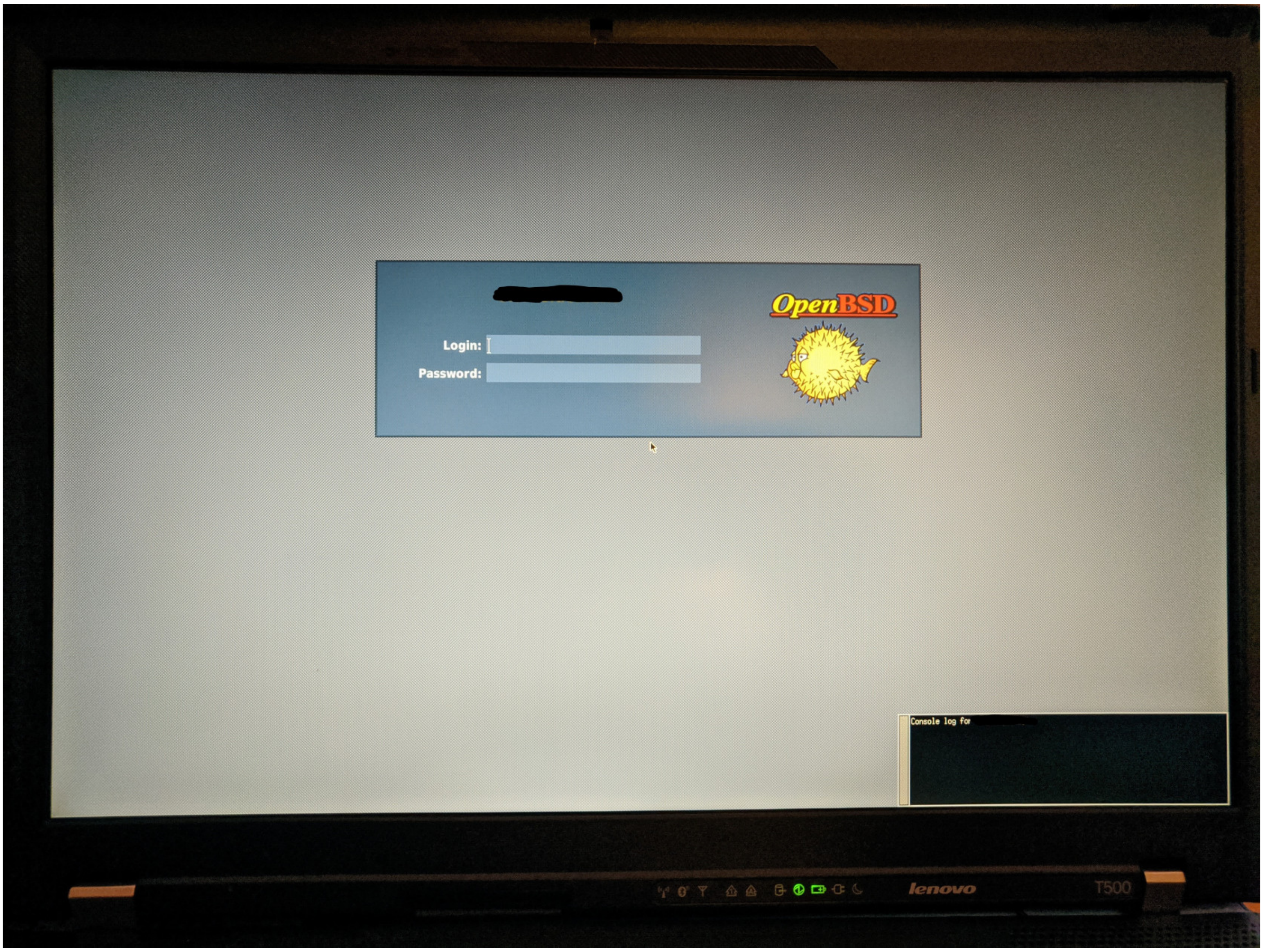
It will take some time to download all the file sets, but after that's done, you should see a similar screen:



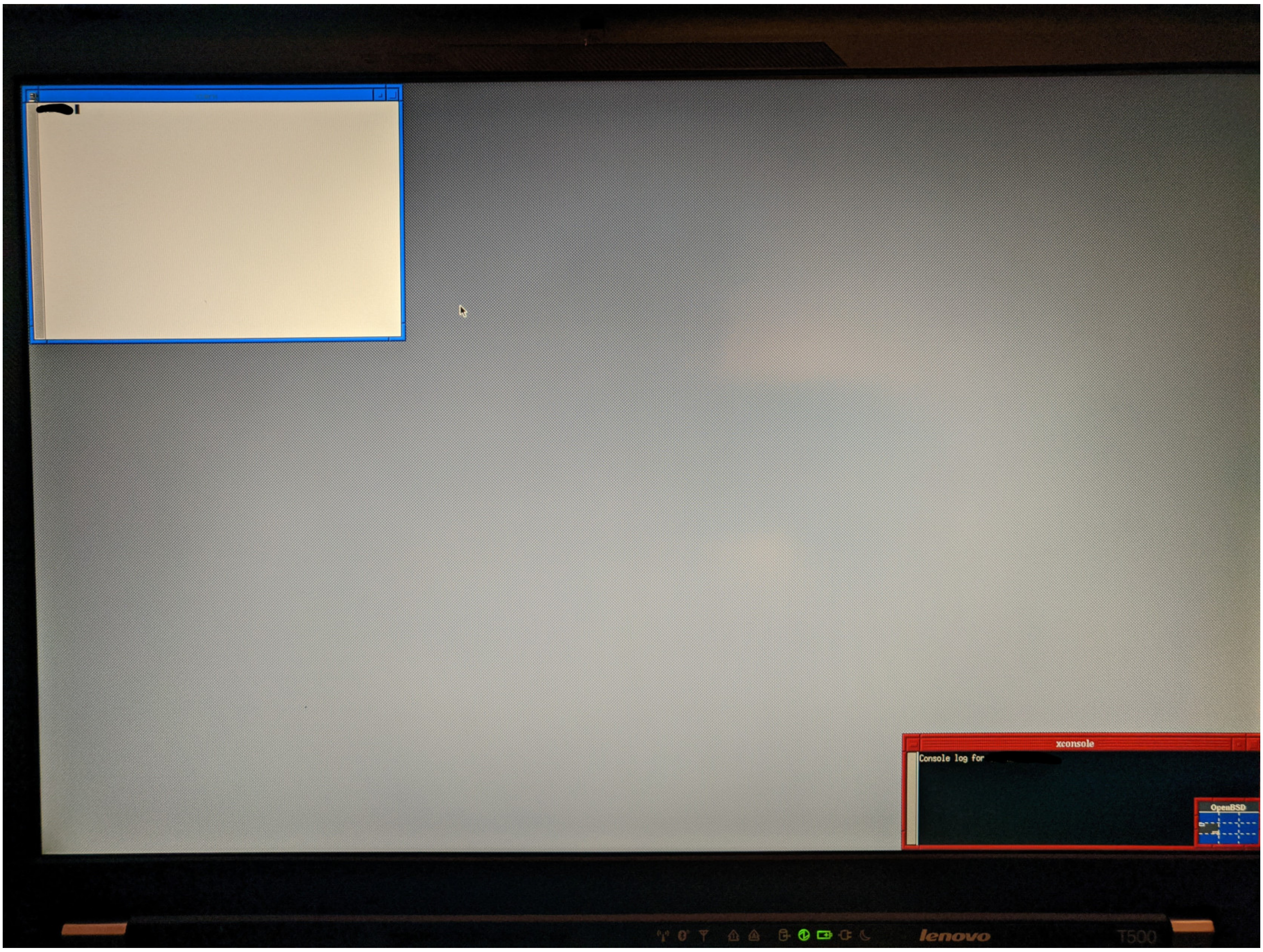
```
Get/Verify xserv69.tgz   100% |***************************| 18351 KB    00:19
Installing bsd           100% |***************************| 20423 KB    00:00
Installing bsd.mp        100% |***************************| 20515 KB    00:00
Installing bsd.rd        100% |***************************|  4107 KB    00:00
Installing base69.tgz    100% |***************************|   291 MB    00:23
Extracting etc.tgz       100% |***************************|   254 KB    00:00
Installing man69.tgz     100% |***************************|  7560 KB    00:01
Installing xbase69.tgz   100% |***************************| 29789 KB    00:03
Extracting xetc.tgz      100% |***************************|  7101       00:00
Installing xshare69.tgz  100% |***************************|  4502 KB    00:01
Installing xfont69.tgz   100% |***************************| 39342 KB    00:02
Installing xserv69.tgz   100% |***************************| 18351 KB    00:02
Location of sets? (cd0 disk http nfs or 'done') [done]
Time appears wrong.  Set to 'Fri Aug 27 20:23:27 CEST 2021'? [yes]
Saving configuration files... done.
Making all device nodes... done.
Multiprocessor machine; using bsd.mp instead of bsd.
Relinking to create unique kernel... done.

CONGRATULATIONS! Your OpenBSD install has been successfully completed!

When you login to your new system the first time, please read your mail
using the 'mail' command.

Exit to (S)hell, (H)alt or (R)eboot? [reboot] _
```

Just press *enter* to reboot. When it shuts down, remove the installation USB drive, but keep the key drive in, otherwise it won't boot.

You will be presented this *very elegant* login screen.



What comes next looks even better, doesn't it?



[redacted]

```
Console log for [redacted]
```

Don't worry, this is just the default FVWM windows manager. You can install any other windows manager you like or rice FVWM, which some people actually do, but I wouldn't recommend it. We will go over how to install DWM later in another write-up.

Backup the USB key drive

What we should do first before anything else is **backup the damned USB key drive.** There is no point of setting anything up when all our work can be lost at any moment due to the USB drive failure. You may either lose it or it might just die, not uncommon in the world of removable media devices. The official guide mentions two command for backup and restore, however we need to get the backup file off the system and store it in a secure place. For that, we will use another USB drive. This may seem a little daunting especially if you have never manipulated devices on Linux or BSD systems.

On Linux, there is a nice command to show what devices you have connected and what is on them – *lsblk.* Unfortunately, that is not a thing on OpenBSD.

```
T500# lsblk
ksh: lsblk: not found
```

First of all login with root, you will most likely need to elevate privileges. It is not a good idea to do it regulary, but totally fine until we setup doas for our normal user. Before connecting anything, issue the following commands to see how many *devices* there are and what they contain. **T500** is our hostname, only type in stuff after the #, if there is no T500 at the front, that usually means it is an output of the previous command or a special sort-of "nested" or "interactive" command.

```
T500# sysctl hw.diskcount
hw.diskcount=3


T500# sysctl hw.disknames
hw.disknames=sd0:f985648fds15dsf,cd0:,sd2:d5876a3854643fc3de
```

or you can list everything from sysctl and filter its output using grep

```
T500# sysctl -a | grep hw.disk
hw.disknames=sd0:f985648fds15dsf,cd0:,sd2:d5876a3854643fc3de
hw.diskcount=3
```

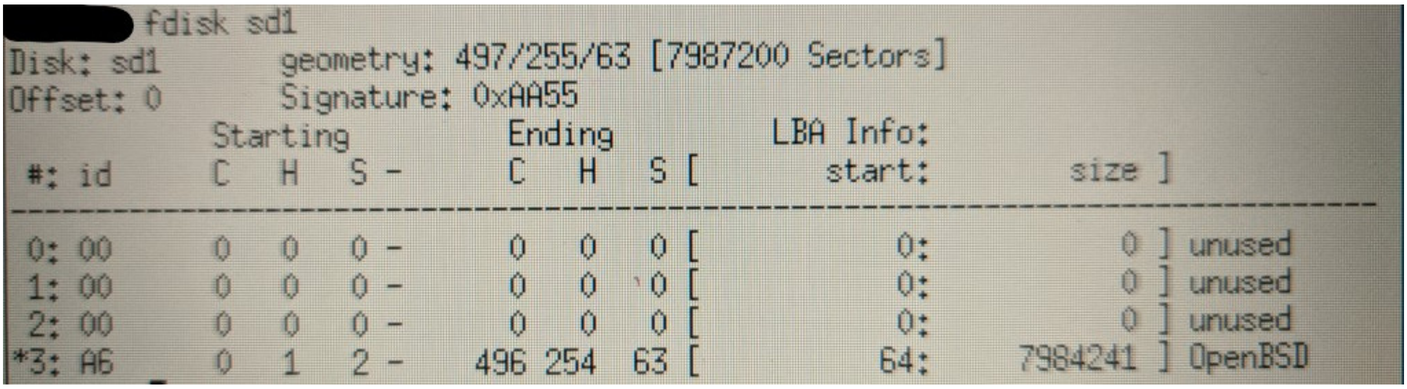
I currently don't have any USB drives connected (I also disconnected the encryption key drive). *sd0* is my SSD, but *sd2* is the encryption layer where all partitions reside. You may have noticed that the encryption layer used to be *sd3* during install. That's because these names don't always have to belong to the same device, they are assigned at boot. In our case, the laptop starts booting from the SSD – *sd0*, then finds a USB drive, recognizes it as *sd1* and unlocks the encryption layer in which is another filesystem that gets named *sd2*.

Plug in the USB drive with the encryption key on it and run the commands above again. You should now see 4 devices and *sd1* should appear.

```
T500# sysctl -a | grep
hw.diskhw.disknames=sd0:f985648fds15dsf,cd0:,sd2:d5876a3854643fc3de,sd1:0dfca798643fdjlk5
hw.diskcount=4
```

To make sure you know which device is which, use fdisk on sd1. It should be significantly smaller than sd2 and sd0.

```
T500# fdisk sd1
```



```
[redacted] fdisk sd1
Disk: sd1       geometry: 497/255/63 [7987200 Sectors]
Offset: 0       Signature: 0xAA55
            Starting          Ending          LBA Info:
 #: id      C   H   S -      C   H   S [        start:          size ]
------------------------------------------------------------------------------
 0: 00      0   0   0 -      0   0   0 [            0:             0 ] unused
 1: 00      0   0   0 -      0   0   0 [            0:             0 ] unused
 2: 00      0   0   0 -      0   0   0 [            0:             0 ] unused
*3: A6      0   1   2 -    496 254  63 [           64:       7984241 ] OpenBSD
```

Plug in the **second** USB drive that will be used as a backup and repeat the process above.

```
T500# sysctl -a | grep
hw.diskhw.disknames=sd0:f985648fds15dsf,cd0:,sd2:d5876a3854643fc3de,sd1:0dfca798643fdjlk5,sd3:
hw.diskcount=5
```

With all the devices connected, you can make sure once more you know which is which. To get more info about devices on OpenBSD, use the *disklabel* command. Running disklabel on all devices (e.g. disklabel sd0) will reveal what they are, the *label* line in the output should show this for each:

- sd0 – CT250MX500SSD1
- sd1 – Flash Disk
- sd2 – SR CRYPTO
- sd3 – Flash Disk

Example output for *sd0*:

```
T500# disklabel sd0
# /dev/rsd0c:
type: SCSI
disk: SCSI disk
label: CT250MX500SSD1
duid: f985648fds15dsf
flags:
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 240
sectors/cylinder: 15120
cylinders: 32301
total sectors: 488397168
boundstart: 64
boundend: 488391120
drivedata: 0
```

```
16 partitions:
#□□□size□□□offset□fstype [fzize bsize cpg]
 a:□□488391056□□□□64□  RAID
 c:□□488397168□□□□ 0□unused
```

Mount the backup USB

Encryption key drive is *sd1* and the backup drive is *sd3.* Note that the backup drive already has a FAT32 partition set up from Windows, if it's a unformatted drive, either format it in OpenBSD using *fdisk* and *disklabel* or on another OS. When running *disklabel* on *sd3* we can see the following partitions:

```
i:□□7862272□□□2048  MSDOS
```

Create a mount point in /mnt:

```
mkdir /mnt/backup_drive
```

Mount the MSDOS partition of *sd3* to */mnt/backup_drive*

```
mount -t msdos /dev/sd3i /mnt/backup_drive
```

You can now cd into the mount point, there might be some kind of System Volume Information leftover from Windows.

```
T500# cd /mnt/backup_drive
T500# ls
System Volume Information
```

Write data to the backup USB

Again, use the dd command with a couple of options according to the official guide. *if* takes data from the *a* partition on sd1, which is our key drive and writes it using of to our backup drive mounted at */mnt/backup_drive* to a file *openbsd-keydisk_backup.img.* The name is completely up to you of course.

```
T500# dd bs=8192 skip=1 if=/dev/rsd1a of=/mnt/backup_drive/openbsd-keydisk_backup.img
999+1 records in
999+1 records out
8184320 bytes transferred in 2.982 secs (2744288 bytes/sec)
```

Check if there is a new file in */mnt/backup_drive* directory, you should see something like this:

```
T500# ls -lah
total 16012
drwxr-xr-x 1 root wheel 4.0K Jan  1 1980  .
drwxr-xr-x 3 root wheel 512B Aug 29 17:34 ..
drwxr-xr-x 1 root wheel 4.0K Aug 29 17:23 System Volume Information
-rw-r--r-- 1 root wheel 7.8M Aug 29 17:42 openbsd-keydisk_backup.img
```

Before you shutdown you PC, don't forget to unmount the USB drive.

```
umount /mnt/backup_drive
```

In case you run into the following error,

```
umount: /mnt/backup_drive: Device busy
```

make sure you aren't in the directory with your terminal and move somewhere else with *cd*.

Test the backup

You can shutdown and try booting into the system using the backup USB, but you will quickly realize that it doesn't work. We haven't created an identical backup key drive, only an image of the original one, which we saved to the backup USB. It is even viewable in Windows since it's just a FAT32 partition, so store it in a secure place.

With that in mind, how do you now know that the restore command will work? It wouldn't be a happy moment to discover that your backup wasn't working when something happens. Well, if you want to have peace in mind, get another blank USB drive.

Plug the backup USB and the new testing USB into the PC. Find out which is which using commands you already know (disklabel). In my case, the new USB is easily recongizable because its bigger and has a different label. We want to create a complete copy of the original key drive, not just a regular flash drive with a backup file like we just did.

Initialize MBR on the new drive (sd3). **THIS WILL DELETE EVERYTHING ON THE USB DRIVE!** But I hope you know that.

```
T500# fdisk -iy sd3
Writing MBR at offset 0.
```

Open disklabel editor to create a RAID partition (same process when we were creating the first key drive during installation).

```
T500# disklabel -E sd3
Label editor (enter '?' for help at any prompt)
```

```
sd3> a a
offset: [64] // enter
size: [30025421] 1M // we only want a small partition
FS type: [4.2BSD] RAID // RAID partition
sd3*> w // write changes
sd3> q // quit
```

The drive is now prepared. Now mount the backup drive again. The *backup_drive* folder should still be in */mnt.* The backup drive is sd1 in my case.

```
mount -t msdos /dev/sd1i /mnt/backup_drive
```

Write the image file to the new USB drive (*a* partition of the raw sd3 device)

```
# dd bs=8192 seek=1 if=/mnt/backup_drive/openbsd-keydisk_backup.img of=/dev/rsd3a
999+1 records in
999+1 records out
8184320 bytes transferred in 2.822 secs (2899937 bytes/sec)
```

Unmount the USB drive with the backup image file and shutdown. Only leave in the third USB which we wrote the backup into.

```
T500# umount /mnt/backup_drive
shutdown -p now
```

If you followed everything correctly, the system should now boot with the new USB drive! Now you have two drives for decrypting and one backup. Because you now know that the backup image file is working, you can scrape the second decrypting USB and store the backup image file in a secure place. In case something happens to the original decrypting drive, you have a tested backup.

This is it for now, to make the laptop a bit more usable, check out my other guides :)