Global Information Assurance Certification Paper

Securing User Data With CGD

Owen Becker

March 21, 2006

Abstract

This paper will introduce the issues surrounding the securing of user data on untrusted machines. It will present a solution to these issues utilizing CGD, NetBSD's disk encryption technology. The paper will also explore the CGD architecture and provide instructions for implementing disk-level encryption on a new installation of NetBSD.

Contents

1 Data security in untrusted environments

System Administrators have often thought that it is impossible to secure a system when it is located in an untrusted environment. Historically, they have been right. After an attacker has direct access to a machine it is trivial to insert a bootable CD and bypass software access controls. A password protected bios can be sidestepped by opening the case and installing the drives into an alternate machine. In enterprise environments, servers are usually placed behind the locked doors of a data center. This strategy has several important shortcomings.

1.1 Three Scenarios for Consideration

Networks are, in the end, a good deal more organic than most security personnel will acknowledge. Locking away the servers is often less helpful than assumed. The following examples will show where normal physical access controls fail.

1.2 Wandering Laptops

Laptops have a way of dissapearing. During the summer of 2000, Los Alamos National Laboratory lost several, and the information they contained was vital to national security. They were later found behind a copy machine and no real damage was done. The idea of such sensitive data being disseminated launched a Congressional investigation. Last January the financial services company Ameriprise Financial Inc. was forced to notify over 158,000 customers that their personal information had been compromised after one of its laptop was stolen. The missing data contained names, Social Security numbers and account information.

1.3 Insecure Hardware Disposal

Hardware disposal is the last phase of the infrastructure life-cycle. Because computers contain large quantities of lead and other environmentally unpleasant materials, local landfills rarely accept them. An entire industry has evolved to recycle hardware, and many people have been surprised to find their financial data sitting on drives for sale on e-bay. Secure drive disposal nessisicates overwriting the data multiple times; when there is an entire office worth of machines the required effort can be immense. It is hardly surprising that this time consuming task is rarely completed.

1.4 Theft at the Data Center

Outsourcing creates another set of issues for physical data center security. When a company owns the servers, the network, and the data center, it

is straightforward to set and enforce policy. When a third party begins to assume these functions, a company now only has the ability to make agreements that may or may not be followed by the provider. The physical security they provide may break down in unexpected ways, leaving machines vulnerable to theft. This is an especially large concern for companies that employ overseas firms to manage their equipment.

2 Using CGD to Mitigate the Threat

To restate the problem, "How can our data be kept secure if our hardware cannot?" NetBSD, a freely available Unix-like operating provides a novel solution called the Cryptographic Disk Driver, or CGD.

2.1 Introducing the Technology

CGD is a cryptographic subsystem that provides secure data storage. With it employed, a system may be lost, stolen, or improperly disposed of without risking the loss of sensitive data.

2.2 Capabilities

With CGD, you are, in essence, given an encrypted filesystem. Even if an attacker reboots the computer with a live CD such as Knoppix, without the pass-phrase the data remains inaccessible. Compared to similar technologies CGD is quite transparent. It provides a virtual drive on which a filesystem is built, and as such does not alter the traditional Unix security model. When a machine is booted, before the normal startup routine, the user is presented with a password prompt that must be entered before the filesystem can be mounted. After mounting, the system appears to be a normal Unix-like system.

2.3 Limitations

Any cryptographic subsystem carries with it a performance penalty. Encrypting and decrypting data uses cpu cycles and will slow a system during periods of heavy disk I/O. There is also the issue of a lost pass-phrase; when it is gone, so is your data. Without it you are in the same position as the person who stole your laptop. Creating regular secure backups is advised.

3 Exploring the CGD Architecture

NetBSD's CGD is implemented in two parts, a kernel driver that provides an encrypted interface to raw partitions and a userland configuration utility.

3.1 Raw Partition Encryption

The kernel portion of CGD is implemented as a psuedo-device driver. It is in the same class as other special purpose drivers in the NetBSD kernel. Two examples are the ccd interface to multiple concatenenated disks and the rnd interface for random numbers. When employed, it sits below the buffer cache and exports an encrypted interface to a raw device.

Because of its position in the kernel, any filesystem can be created on top of an encrypted partition. Although primarily used for the FFS filesystem, it is possible to create an MSDOS compatable FAT filesystem or the Sprite derived LFS filesystem. It will, however, not be possible to directly access these filesystems from anything other than a NetBSD machine. Perhaps more useful is the possiblitiy of creating of an encrypted backup with a cgd based CD9660 image. A CGD can also be used for the backing of a swap partition or a raw device for database storage.

CGD is designed to be modular; it is independent of any one particular cipher or key generation method. For encryption methods, it currently supports aes-cbc, blowfish and 3des. Each have various strengths that make them more appropriate for different threat vectors. For key generation, pkcs5_pbkdf2 is used for pass-phrases. Also supported is a gssapi interface. With a gssapi compatible keyserver remote reboots become a possibility. In addition, there exists a randomkey method which uses a random string as the encryption key. It is intended for use in swap space, where having contents survive across reboots is a disadvantage.

3.2 Userland Configuration Utility

The userland tool is named cgdconfig. It can configure a new CGD device, verify that it contains a valid filesystem, configure the encryption scheme, and output a parameter file that will allow the device to be reconfigured on reboot. With it you can also scrub a disk bit-by-bit with random data. This will prevent an attacker from being able to determine which parts of the drive are blank.

4 Instructions for Installing CGD

The proceeding section will give a step by step overview for installing NetBSD-3.0 with CGD. With the exception of a tiny root filesystem, everything will be encrypted. This process involves a normal install of NetBSD followed by a dump of all but the root filesystem to a remote machine. Once the filesystems have been backed up, the disklabel will be reworked and made CGD aware. A CGD device will then be created and the filesystem will be restored.

4.1 Machine Description

The target system is a Pentium II with 256 megabytes of RAM. It contains a 10 gigabyte hard drive and a generic intel ethernet card. NetBSD is quite capable on older hardware, and the performance penalty incurred from running on such a modest system is negligible. An ssh server with sufficient disk space should exist somewhere on the network. It will hold the filesystem dump images.

4.2 Initial NetBSD Installation

The initial install should follow the NetBSD-3.0 installation guide for the i386 architecture. It is straightforward and should not cause any significant difficulties. It might be prudent to delay installing the XFree86 sets until after the CGD filesystems are created and restored. Since we are going to dump the filesystems over the network, the space savings will reduce our configuration time. We will install the compiler tools as we need to rebuild the GENERIC kernel with support for CGD.

4.3 Kernel Re-Compilation

After the initial install is completed, log on to the NetBSD ftp server and download tar the kernel sources, syssrc.tar.gz. As root extract the source distribution:

```
localhost# tar xvfz syssrc.tar.gz -C /
```

To rebuild the kernel, change into the /usr/src/sys/arch/i386/conf directory and copy the GENERIC kernel configuration file to GENERIC-CGD.

```
localhost# cd /usr/src/sys/arch/i386/conf
localhost# cp GENERIC GENERIC-CGD
```

Open the GENERIC-CGD file with your preferred text editor and uncomment the line containing "psuedo-device cgd." We will now configure and build the new kernel.

```
localhost# config GENERIC-CGD
Build directory is ../compile/GENERIC-CGD
Don't forget to run "make depend"
localhost# cd ../compile/GENERIC-CGD
```

```
localhost# make depend && make
(kernel build output follows...)
```

The kernel may take some time to build depending on the speed of your machine. Once the compilation is finished, the currently running kernel needs to be backed up and replace with the new one. Assuming you are in /usr/src/sys/arch/i386/compile/GENERIC-CGD, execute the following commands:

```
localhost# cp /netbsd /netbsd.old
localhost# cp netbsd /
localhost# reboot
```

4.4 Filesystem Backups

Once the system is rebooted we can start creating the cgd filesystems. Login as root and bring the system down into single user mode.

```
localhost# shutdown now
Shutdown NOW!
shutdown: [pid 508]
wall: You have write permission turned off; no reply possible

*** FINAL System shutdown message from root@localhost.localnet.org ***
System going down IMMEDIATELY

localhost# Mar 20 13:20:50 localhost shutdown: shutdown by root:

System shutdown time has arrived

About to run shutdown hooks...
Stopping cron.
Waiting for PIDS: 506.
Stopping inetd.
Mon Mar 20 13:21:02 EST 2006

Done running shutdown hooks.
Mar 20 13:21:17 localhost syslogd: Exiting on singlal 15
Enter pathname of shell or RETURN for /bin/sh: (Hit return)
#
```

We need to backup the default mbr as an individual file. This will be needed later to fdisk the cgd.

```
# scp /usr/mdec/mbr user@shell.somemachine.net:/home/user
```

At this point we need to unmount the all but the root filesystem and dump the images. Doing so in single user mode ensures that the backups are made correctly.

```
# mount
/dev/wd0a on / type ffs (local)
/dev/wd0f on /var type ffs (local)
/dev/wd0e on /usr type ffs (local)
/dev/wd0g on /home type ffs (local)
kernfs on /kern type kernfs (local)

# umount /var /usr /home
# mount
/dev/wd0a on / type ffs (local)
kernfs on /kern type kernfs (local)
```

Now that everything is unmounted, we begin the dump. Since we only have one disk in this system, we are pushing the dump to a remote filesystem via ssh. The majority of the tools we need for the dump sit under /usr, we need to use the staticly linked binaries under /rescue.

```
# cd /rescue
# ./dump -0uan -f - /var | ./gzip | ./ssh user@shell.somemachine.net \
> "dd of=/home/user/var.dmp.gz"
(Dump output begins. For each dump type in ssh password when prompted.)
# ./dump -0uan -f - /usr | ./gzip | ./ssh user@shell.somemachine.net \
> "dd of=/home/user/usr.dmp.gz"
# ./dump -0uan -f - /home | ./gzip | ./ssh user@shell.somemachine.net \
> "dd of=/home/user/home.dmp.gz"
```

4.5 Disk Preparation

Our next task is to modify the disklabel. We will delete the entries for /home, /var, and /usr and create a single entry to contain the cgd. Your individual drive details (tracks, sector size) will vary.

```
# disklabel /dev/wd0

type: unknown
disk: HARDDISK
label:
flags:
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 8322
total sectors: 8388608
rpm: 3600
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0 # microseconds
track-to-track seek: 0 # microseconds
drivedata: 0

16 partitions:
#        size    offset     fstype [fsize bsize cpg/sgs]
 a:    525168        63     4.2BSD   1024  8192 43768  # (Cyl.      0*-    521*)
 b:    525168    525231       swap                     # (Cyl.    521*-   1042*)
 c:   8388545        63     unused      0     0        # (Cyl.      0*-   8322*)
 d:   8388608         0     unused      0     0        # (Cyl.      0 -   8322*)
 e:   2097648   1050399     4.2BSD   2048 16384 21872  # (Cyl.   1042*-   3123*)
 f:     66528   3148047     4.2BSD   1024  8192  8320  # (Cyl.   3123*-   3189*)
 g:   5174033   3214575     4.2BSD   2048 16384 26864  # (Cyl.   3189*-   8322*)


# disklabel -i wd0
partition> P
16 partitions:
#        size    offset     fstype [fsize bsize cpg/sgs]
 a:    525168        63     4.2BSD   1024  8192 43768  # (Cyl.      0*-    521*)
 b:    525168    525231       swap                     # (Cyl.    521*-   1042*)
 c:   8388545        63     unused      0     0        # (Cyl.      0*-   8322*)
 d:   8388608         0     unused      0     0        # (Cyl.      0 -   8322*)
 e:   2097648   1050399     4.2BSD   2048 16384 21872  # (Cyl.   1042*-   3123*)
 f:     66528   3148047     4.2BSD   1024  8192  8320  # (Cyl.   3123*-   3189*)
 g:   5174033   3214575     4.2BSD   2048 16384 26864  # (Cyl.   3189*-   8322*)
partition> b
```

```
Filesystem type [?] [swap]: (Hit return)
Start offset ('x' to start after partition 'x') [[n]c, [n]s, [n]M]: 0
Partition size ('$' for all remaining) [[n]c, [n]s, [n]M]: 0
partition> e
Filesystem type [?] [4.2BSD]: (Hit return)
Start offset ('x' to start after partition 'x') [[n]c, [n]s, [n]M]: 0
Partition size ('$' for all remaining) [[n]c, [n]s, [n]M]: 0
partition> f
Filesystem type [?] [4.2BSD]: (Hit return)
Start offset ('x' to start after partition 'x') [[n]c, [n]s, [n]M]: 0
Partition size ('$' for all remaining) [[n]c, [n]s, [n]M]: 0
partition> g
Filesystem type [?] [4.2BSD]: (Hit return)
Start offset ('x' to start after partition 'x') [[n]c, [n]s, [n]M]: 0
Partition size ('$' for all remaining) [[n]c, [n]s, [n]M]: 0
```

We are actually setting each partition to zero bytes. It has the same effect as deleting it. Partitions "c" and "d" represent the entire drive, do not alter them. Now we create a partion for the cgd device. We will set the filesystem type to "ccd" as NetBSD's disklabel has no cgd type. It has no real effect other than as a reminder.

```
partition> e
Filesystem type [?] [4.2BSD]: ccd
Start offset ('x' to start after partition 'x') [0c, 0s, 0M]: a
Partition size ('$' for all remaining) [0c, 0s, 0M]: $
e:   786337    525231 ccd        # (Cyl.   521*-   8322*)
partition> W (write the label)
Label disk [n]? y
Label written
partition> Q
```

4.6 CGD Creation

Before we actually create the cgd, we need to scrub the drive with random data. The lack of empty sectors on the drive will discourage forensic analysis. We do this is two steps. First, initialize the cgd device using /dev/urandom as the key and next, write over each bit of the device with dd.

```
# cgdconfig -s cgd0 /dev/wd0e aes-cbc 128 < /dev/urandom
# dd if=/dev/zero of=/dev/rcgd0d bs=32k
# cgdconfig -u cgd0
```

Now for the actual cgd creation.

```
# echo 'cgd0     /dev/wd0e' > /etc/cgd/cgd.conf
# cgdconfig -g -V disklabel -o /etc/cgd/wd0e aes-cbc 256
# cgdconfig -V re-enter cgd0 /dev/wd0e
/dev/wd0e's passphrase:
re-enter device's passphrase:
```

A bit of explanation is in order. We first create the cgd configuration file, then we build the device, and last we configure the passphrase. Please remember whatever passphrase you use. After this point, losing it will render your data completely inaccessible. You cannot get it back.

Now to begin the fdisk and disklabel creation. This will build filesystems for swap, /var, /usr, and /home.

```
# mkdir /usr/mdec
# /rescue/ssh user@somemachine.net "cat /home/user/mdec" | dd of=/usr/mdec/mbr
(This restores the mbr for the fdisk)
# fdisk -u cgd0
fdisk: primary partition table invalid, no magic in sector 0
Disk: /dev/rcgd0d
NetBSD disklabel disk geometry:
cylinders: 3839, heads: 1, sectors/track: 2048 (2048 sectors/cylinder)
total sectors: 7863377

BIOS disk geometry:
cylinders: 489, heads: 255, sectors/track: 63 (16065 sectors/cylinder)
total sectors: 7863377

Do you want to change our idea of what BIOS thinks? [n]

Partition table:
0: <UNUSED>
1: <UNUSED>
2: <UNUSED>
3: <UNUSED>
Bootselector disabled.
Which partition do you want to change?: [none]

We haven't written the MBR back to disk yet.  This is your last chance.
Partition table:
```

```
0: <UNUSED>
1: <UNUSED>
2: <UNUSED>
3: <UNUSED>
Bootselector disabled.
Should we write new partition table? [n] y

# disklabel -I -i cgd0
partition> a
Filesystem type [?] [4.2BSD]:
Start offset ('x' to start after partition 'x') [0c, 0s, 0M]: 0
Partition size ('$' for all remaining) [3839.54c, 7873377s, 3839.54M]: 0
partition> b
Filesystem type [?] [unused]: swap
Start offset ('x' to start after partition 'x') [0c, 0s, 0M]: 0
Partition size ('$' for all remaining) [0c, 0s, 0M]: 256M
 b: 524288 0 swap # (Cyl.  0 -  255)
partition> e
Filesystem type [?] [unused]: 4.2BSD
Start offset ('x' to start after partition 'x') [0c, 0s, 0M]: b
Partition size ('$' for all remaining) [0c, 0s, 0M]: 512M
 e: 1048576 524288 4.2BSD 0   0   0  # (Cyl.  256 - 767)
partition> f
Filesystem type [?] [unused]: 4.2BSD
Start offset ('x' to start after partition 'x') [0c, 0s, 0M]: e
Partition size ('$' for all remaining) [0c, 0s, 0M]: 1024M
 f: 2097152 1572864 4.2BSD 0   0   0  # (Cyl.  768 - 1791)
partition> g
Filesystem type [?] [unused]: 4.2BSD
Start offset ('x' to start after partition 'x') [0c, 0s, 0M]: f
Partition size ('$' for all remaining) [0c, 0s, 0M]: 1024M
 f: 4193361 3670016 4.2BSD 0   0   0  # (Cyl.  1792 - 3839*)
partition> W
Label disk [n]? y
Label written
partition> Q
# newfs /dev/cgd0e
(newfs output follows)
# newfs /dev/cgd0f
(newfs output follows)
# newfs /dev/cgd0g
(newfs output follows)
# cp /etc/fstab /etc/fstab.bak
# cat > /etc/fstab
```

```
/dev/wd0a       /       ffs     rw 1 1
/dev/cgd0b      none    swap    sw 0 0
/dev/cgd0b      /tmp    mfs     rw,-s=132048 0 0
kernfs          /kern   kernfs  rw
procfs          /proc   procfs  rw,noauto
/dev/cgd0e      /var    ffs     rw,softdep 1 2
/dev/cgd0f      /usr    ffs     rw,softdep 1 2
/dev/cgd0g      /home   ffs     rw,softdep 1 2

(Hit Ctrl-D)
# mount -a
```

All the cgd aware filesystems should be mounted. Now, on to the data restoration.

4.7 Data Restoration

To restore the data, we pull it off the server via ssh and cat it through /rescue/restore.

```
# cd /var
# /rescue/ssh user@shell.somemachine.net "cat /home/user/var.dmp.gz" | \
/rescue/gunzip | /rescue/restore rf -
user@shell.somemachine.net's password: (Enter password)

# cd /usr
# /rescue/ssh user@shell.somemachine.net "cat /home/user/usr.dmp.gz" | \
/rescue/gunzip | /rescue/restore rf -
user@shell.somemachine.net's password: (Enter password)

# cd /home
# /rescue/ssh user@shell.somemachine.net "cat /home/user/home.dmp.gz" | \
/rescue/gunzip | /rescue/restore rf -
user@shell.somemachine.net's password: (Enter password)

# reboot
```

The system will prompt for a passphrase before it mounts the cgd filesystems. Beyond that, it should look like a normal installation of NetBSD.

4.8 Conclusions

NetBSD's CGD subsystem is a practical method for securing data in untrusted environments. It provides system administrators a level of assurance that has up to now been unavailable.