GELI Boot

Booting from Encrypted Disks on FreeBSD

Allan Jude -- ScaleEngine Inc.

allanjude@freebsd.org twitter: @allanjude

Introduction

Allan Jude

- 13 Years as FreeBSD Server Admin
- FreeBSD src/doc committer (focus: ZFS, bhyve, ucl, xo)
- Co-Author of “*FreeBSD Mastery: ZFS*” and “*FreeBSD Mastery: Advanced ZFS*” with Michael W. Lucas (For sale in the hallway)
- Architect of the ScaleEngine CDN (HTTP and Video)
- Host of BSDNow.tv & TechSNAP.tv Podcasts
- Use ZFS for large collections of videos, extremely large website caches, mirrors of PC-BSD pkgs and RaspBSD
- Single Handedly Manage Over 1000TB of ZFS Storage

Overview

- Do a lot of work with ZFS
- Helped build the ZFS bits of the installer
- Integrated ZFS Boot Environments
- Created ZFS Boot Env. Menu
- ZFS Boot Env. do not work with GELI
- Booting from GELI encrypted pool requires creating an unencrypted “boot pool” with the kernel and GELI module
- Boot Environments are awesome, you should use them too

I Have Written A Thing

- I am a very novice C programmer
- Implemented a minimal version of GELI in the gpt{,zfs}boot (UFS and ZFS) bootcodes
- Took a lot of time to understand the existing bootcode and how it works
- Took a lot of learning about C
- The existing boot code is terrible, and needs much love, too much copy-pasta
- Had to navigate many obstacles
- but, it works!

How Do Computers Even Work?

- BIOS reads the 512 bytes MBR
- Consists of 446 byte bootstrap program, and partition table (4 entries)
- This bootstrap is then executed (boot0.S)
- It examines the partition table and finds the active partition, reads the first 512 bytes
- This is boot1. It loads boot2, which in a UFS formatted partition is the first 15 sectors
- This can understand UFS, loads /boot/loader
- The loader presents a menu, and loads the kernel, then the system boots

ZFS + MBR = Evil

- To boot ZFS w/ MBR, much evil is required
- boot0 reads a different boot1 from the first sector of the active partition
- This boot1 is different, it seeks to an offset of 1 MB in the ZFS partition, and reads 64 KB
- This is ‘zfsboot’, analogous to UFS’s boot2
- This version of boot2 can understand ZFS
- Passes the loader the zpool GUID to boot
- Reads /boot/loader and executes it
- The loader presents a menu, and loads the kernel, then the system boots

GPT - Less Complicated

- GPT partition tables can address disks larger than 2 TB and can have 128 partitions
- The first 512 bytes is a Protective MBR
- The FreeBSD pmbr find freebsd-boot partition, and loads up to 545 KB from it
- This is usually gptboot (UFS) or gptzfsboot
- These bootcodes contain gptldr+boot2
- gptldr relocates the code to the expected memory offset, then executes boot2
- These reads and executes /boot/loader or /boot/zfsloader which starts the kernel

The Start Of a Journey

- Where to start?
- Make a copy of gptzfsboot -> gptgeliboot
- Idea: Make a single bootcode that can do UFS and ZFS (this is still a good idea)
- If system has both, how do you decide?
- Instead: implement GELI in both separately
- zfsboot is MBR only, fixed size, do not touch
- Working with bootcode is hard. There are no debugging facilities, errors either hang the system or produce undecipherable errors

Plan Of Attack

- How do you tell if a partition is encrypted?
- Read the very last sector of the partition
- Not always that least, `struct dsk` may or may not have ‘start’ set to the offset of the partition, so reads may be relative to the whole disk, or just the partition
- Parse it into the GELI metadata struct
- Is the ‘magic’ “GEOM::ELI”?
- Then it is GELI

ZFS Makes Life Easier, As Usual

- It turns out the ZFS boot code made this easier, instead of reading from the disk directly, it takes a pointer to a function that does the reading
- Conditionally replace this function with one that also decrypts the sector before returning it to the ZFS code
- Adapt the UFS code to do similar, to increase code sharing and reuse

Initial Implementation

- After figuring out that the partition is encrypted, the obvious next step is to decrypted it
- Read the GELI metadata to determine algorithm, key size, master key
- Decrypt master key with user provided pass phrase, no support for key files yet
- Need some crypto
- GELI uses kernel crypto APIs, too big and too complicated for bootcode

Tiny-AES-C

- Needed an AES-CBC implementation small enough to use in the boot code
- Found public domain Tiny-AES-C on github
- Only does AES-CBC-128, no 256, no XTS
- Borrow some functions from GELI and adapt them to use this AES implementation
- Check GELI version, set some flags
- Decrypt and validate master key
- Calculate HMAC, Sector Key, and IV

Hash Party

- GELI uses MD5 to verify metadata
- GELI uses SHA256 for generating the unpredictable sector IV
- SHA512 used for HMACs all over
- Can’t just #include them like other stuff, conflicting #defines in the algorithm
- Just add them to libstand32!
- (eventually replaced by creating libgeliboot to house all of the dependencies and helper functions)

Prompting For A Password

- This should be easy...
- Borrow `getstr()` from `sys/boot/i386/common/cons.c`
- Modified to echo * instead of original char
- Loader works differently, because serial
- uses `xgetc()` instead of `getchar()`
- So need two different versions...
- Turns out both contain the same bug too
- Instead use `ngets()` from libstand (NetBSD)
- Call it `pwgets()` and put it in libgeliboot

Broken getstr()

```
void getstr(char *cmdstr, size_t cmdstrsize) {
    char *s; int c; s = cmdstr;
    for (;;) {  switch (c = xgetc(0)) {
            case '\n':
                *s = 0; return;
            default:
                if (s - cmdstr < cmdstrsize - 1)
                        *s++ = c;
                putchar(c);
                break;
            } }
}
```

Test Drive

- Expectation: `boot2` would start, taste the partition, determine it was GELI encrypted, read the master key, prompt the user for the password, decrypt it with the passphrase, and stand ready to determine the sector key and decrypt each block as needed
- Result: Triple Fault, VirtualBox crashes
- What is a triple fault anyway?
- Try real hardware: reboot loop

First Roadblock

- When gptldr was created, for ease of implementation owing it its 16 bit nature, only the first 64 KB of the bootcode is relocated to the correct memory address
- When work started on this project gptboot (UFS) was less than 16 KB, and gptzfsboot was only 42 KB
- However, now gptzfsboot has grown an AES implementation, both SHA256 and SHA512, and the important bits of GELI, leaving it on the heavy side of 90KB

Roadblock Avoidance

- We’ll come back to solving that problem
- Just stick to UFS, which is easier, and has a smaller boot code, we can still progress
- Rework the code to use the same callback as ZFS to read from the encrypted disk
- After much fiddling, decryption worked
- gptboot decrypted the file system and read /boot/loader and launched it
- The loader immediately failed, was not yet GELI enabled, could not read the file system

Boot Code Environment Constraints

- bootcode is a very restricted environment
- No kernel
- No libc
- No malloc()
- No panic()
- libstand means #include <string.h> conflicts
- The bootcode implements a very simple malloc() that is basically 3 MB of heap space and a cursor
- This means there is no free()

Teaching the Loader to Speak GELI

- Find the place where the loader reads from the disk, and insert GELI decryption
- First need to have tasted the disk and determined it was GELI, read master key
- Loader has a filesystems array, could add a GELI_UFS file system…
- Instead, libi386 has the low level routines to access the disk, intercept data here, decrypt
- Ideally: implement more transparently, maybe layered with bcache

First Boot

After teaching the loader to how to decrypt:

- First Successful Boot of GELI encrypted disk
- Supports AES-CBC 128 only…
- No support for ZFS
- 64 KB binary size limit
- Not that useful…

In order to proceed any further, the 64 KB limit needed to be overcome

Breaking the Limits

- How to get past 64 KB limitation of gptldr?
- Compile with -Os … no real different
- -O2 is bigger than -O1
- Try increasing the number of blocks copied by gptldr, asm compiler laughs at me
- Try converting asm to 32 bit to copy more data, CPU laughs at me
- Summon the collective wisdom of the FreeBSD super friends...

Super Friends

- Eitan Adler - Looked at 32 bit conversion, or copying 2 blocks of 64 KB, ENOTIME
- John-Mark Gurney - Seemed receptive, once understood scope and read existing code, quickly suggested asking others
- John Baldwin - (Original Author of gptldr) suggests finding some other way, like partition with only GELI enabled loader
- Peter Grehan - Tried to help, once understood, taught me qemu asm debugging instead
- Dylan Cochran - Drafted new asm, ENOTIME

EuroBSDCon 2015 - Stockholm, SE

- During the developer summit in Stockholm, Colin Percival approached me, had heard of my plight with 16 bit assembly
- "It’s 16 bit assembly, I know this"
- First draft: BTX crash again
- Second draft: still crash
- Iterations each night, no luck
- We return home
- IRC: Colin has new patch, only does 64 KB but does it a new way, in 32 KB chunks… works
- Last draft 2015-10-08: Copy a variable number of 32 KB chunks, default: 4. #FutureProof

All Hail Dr. Colin Percival

Crypto You Might Actually Use

- Now we have enough space to implement crypto you might actually consider using
- Borrow AES from `sys/crypto/rijndael`
- Fake #define _STRING_H_ so it will not pull in headers that will conflict with libstand
- Borrow AES-XTS from `sys/opencrypto`
- OpenCrypto xform.c has everything, copy & paste AES-XTS, modify to avoid malloc()
- Result: working UFS and ZFS with AES-XTS 256 and AES-CBC 256

Ugly Mess

- At this point the code base is an ugly mess of copy/paste, debug printf()s, and newbie C
- Switch to #include rijndael*.c
- Works, but still kind of ugly
- Doesn’t work for AES-XTS because xform.c is EVERY algorithm, plus hashes and deflate
- SO says better to break up OpenCrypto
- “They let you do that!?”
- svn copy xform.c xform_<algo>.c
- Result: diff of all deletes, shows most code was unmodified, maybe helpful, maybe not

Reusing GELI Code

- Most GELI code has survived unmodified
- struct g_eli_softc had been replaced with the simpler metadata struct, had to undo
- Split g_eli_crypto.c into g_eli_hmac.c
- Bootcode avoids kernel crypto framework and OpenSSL, instead uses OpenCrypto
- Override one function g_eli_crypto_decrypt()
- Add some #ifdef _KERNEL to GELI
- Move some definitions and structs around
- Result: more reusable code

Password Caching

- Boot2 prompts for password per disk
- Loader prompts for password per disk
- geom_eli.ko prompts for password per disk
- Colin Percival (again, hero) had previously implemented password caching in GELI to attempt same password for each disk
- Colin had also added a way to pass passphrase from loader to kernel, with Kris Moore (replace grub) and Devin Teske (add prompt to loader, avoid mountroot)

Passing the Password

- Password now entered twice: boot2 & loader
- How to pass data from boot2 to loader?
- ZFS has the answer! Passes zfs_boot_args
- Struct that has zpool GUID etc
- Starts with ‘size’ member
- Loader reads size, before accessing a member, checks if offsetof(member) > size
- Newer loader doesn’t barf if passed smaller struct by older boot2
- Add new member to hold GELI password
- Added to UFS too

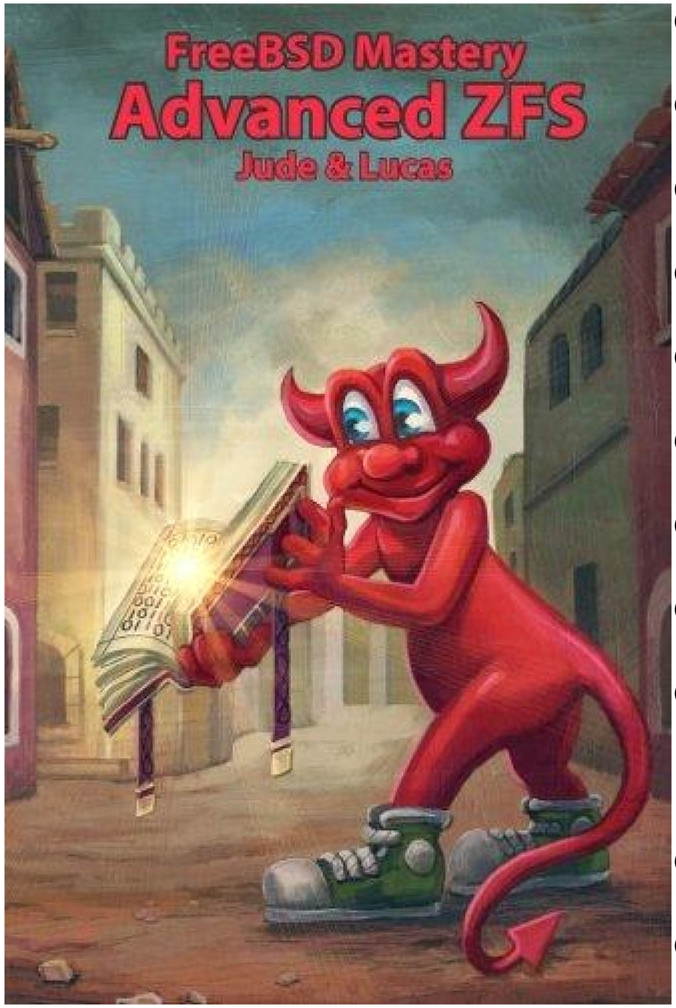
Remaining To Be Done

- Currently geliboot only supports user-entered pass phrases, no key files
- Where to store keys? Consensus is a new unformatted raw partition type: gelikey
- How well will USB “keys” work during boot?
- Add algorithms like blowfish and camellia?
- Support GELI sector auth? Rarely used now
- Support UEFI - Work by Eric McCorkel
- can CSM GELIBoot be layered nicely like it is in EFI? Can bcache update help?

Other Improvements

- Replace GELI uses of SHA256 with SHA512t/256 (50% faster on 64bit platforms)
- Cleanups and more code-reuse in various boot codes, fix shared bugs in getstr() etc
- A single unified bootloader for UFS/ZFS
- Create fuse-geli so GELI encrypted file systems can be used on other OSs
- Ideas?

Get The Book @ ZFSBook.com



- 215 pages of easy reading
- DRM-Free ebook or Print
- Using boot environments
- Delegate filesystem privileges
- Containerize ZFS datasets, jails
- Replicate between machines
- Optimize ZFS block storage
- Select caching strategies
- Manage next-generation storage hardware
- Identify and remove bottlenecks
- Optimizing database storage
- ZFS Internals

Podcasts

BSDNow.tv is a weekly video podcast featuring News, Interviews and Tutorials about the BSD family of Operating Systems. Hosted by Kris Moore (founder of PC-BSD) and Myself.

TechSNAP.tv is a weekly sysadmin video podcast covering an OS agnostic range of security and production issues for those working, studying or interested in the sysadmin / devops / infosec field.

Twitter: @allanjude Email: allanjude@freebsd.org