Practical Filesystem Security

Richard Weinberger
sigma star gmbh

Hello

Richard Weinberger

- Co-founder of sigma star gmbh
- Linux kernel developer and maintainer
- Focus on Linux kernel, low-level components, virtualization, security

Overview of this Talk

- Practical overview of filesystem security on embedded Linux systems
- Hopefully some guidance for your next project
- By no means a complete guide how to implement a whole security concept
- More a collection of pointers

Motivation for Filesystem Security **tongue-in-cheek**

- Care about customer data on the device
- Care about data integrity
- Keep your magic sauce secret
- Have creative licensing
- Pass some certification test

Know your threat model

- Has attacker hardware access?
  - To a running device?
  - Able to dump main memory?
- Access to a shell?
  - root?
  - kernel level?
- Who is the attacker?
  - Nosy neighbor?
  - Competitor?
  - Secret agency?

Filesystem encryption

- Encrypted...
  - Disk?
  - Filesystem structures?
  - Files?
  - Directories?
  - File names?
  - Out of band data (xattr, ...)?

Filesystem encryption: eCryptfs

- Kernel mode stacked filesystem (no FUSE)
- Encrypts file content and file names on top of another filesystem
- Per directory basis
- No authenticated encryption

Filesystem encryption: Possible eCryptfs issues

- Performance overhead from stacking
- File name limit

```
$ stat -f -c "maxlen: %l" /some/ecryptfs
maxlen: 143

$ stat -f -c "maxlen: %l" /other/fs
maxlen: 255
```

- Who of you checks file length limit before creating a file?

Filesystem encryption: Possible eCryptfs issues (cont'd)

- On Linux a file name must not contain a nul byte or a slash
- Encrypting a string can give you any result, including nul bytes or slashes
- eCryptfs has to encode cipher text: Increases length

Filesystem encryption: Using eCryptfs on Yocto

- Add ecryptfs-utils to your rootfs
- Enable `CONFIG_ECRYPT_FS` in kernel config
- mount ecryptfs before you need it
  - initramfs
  - PAM
  - application level

Filesystem encryption: dm-crypt

- Block level encryption, uses device mapper
- Works with any block based filesystem
- Used for FDE (Full Disk Encryption)
- Rich cipher suite
- No authenticated encryption

Filesystem encryption: Using dm-crypt in Yocto

- Add cryptsetup to your rootfs
- Enable `CONFIG_DM_CRYPT` in kernel config
- Setup dm-crypt before you mount the filesystem
  - Happens usually in initramfs

Filesystem encryption: fscrypt

- File encryption at filesystem level (no stacking)
- Currently supported by ext4, f2fs and ubifs
- File content and file names are encrypted
- No meta data nor out of band data (xattr)!
- Per directory basis (per directory encryption policy)
- Per inode AES key
- No authenticated encryption

Filesystem encryption: fscrypt (cont'd)

- Primary use case: per user and directory encryption
- Can be abused to encrypt whole filesystem
- Master key provided via `keyctl`
- Key has to reside in an accessible keyring (e.g. session keyring)
- Has no problem with long file names.
  - Quiz question: Why doesn't it suffer from the same problem as eCryptfs?

Filesystem encryption: Possible fscrypt issues

- `pam_keyinit` is your enemy
- File content in page cache, if user A has a key and reads a file, user B can read it too if access control allows it!
    - Consider mount namespaces or strict DAC/ACL
- Without the key nobody can read cipher text
    - No backup possible without key!

Filesystem encryption: Using fscrypt in Yocto

- Add fscryptctl to your rootfs
- Enable `CONFIG_FS_ENCRYPTION` in your kernel config
- After mounting fileystem make sure either all or selected users have a key

Filesystem encryption: More considerations

- Full disk encryption is the last resort option
- Think of fine grained encryption, eCryptfs or fscrypt help here
  - Do you really need an encrypted /usr and /lib?
  - If possible, combine dm-crypt and eCryptfs/fscrypt

Filesystem encryption: What about data integrity?

- Changed ciphertext usually remains unnoticed
- Just decrypts to garbage
- Attackers can still do evil things
- Think of block swapping or swapping whole (encrypted) files
- e.g. if location of `true` and `login` are known their content can get swapped
  - No plaintext needed
- Pre-generated filesystem images help attackers

Filesystem integrity: dm-verity

- Read-only device mapper target
- Useful for read-only block based filesystem such as squashfs or erofs
- Fast, uses a hash tree
- Use cryptsetup/veritysetup on target
- `CONFIG_DM_VERITY` in kernel config

Filesystem integrity: dm-integrity

- Read-write device mapper target
- Basically adds an auth tag to every block
- Can be combined with dm-crypt
- Use cryptsetup/veritysetup on target
- `CONFIG_DM_INTEGRITY` in kernel config
- Non-negligible overhead

Filesystem integrity: fs-verity

- Integrity for selected files
- Read-only!
- Supported on ext4, f2fs and btrfs
- Use fsverity-utils on target
- Enable `CONFIG_FS_VERITY` in kernel config

Filesystem integrity: authenticated ubifs

- Full authentication support and read-write
- Works because ubifs is strictly copy-on-write
- Can be combined with fscrypt
- Be aware: Featre is rather new

Wait, what about generating images?

- Most mechanisms don't have tooling to generate encrypted/authed images
- We don't recommend it
- Installer approach:
  - rootfs as tarball
  - Generate an installer (IOW a livecd)
  - The installer will setup everything, plus locking down the device

Filesystem encryption: Using fscrypt with mkfs.ubifs

- mkfs.ubifs can generate a pre-encrypted ubifs filesystem, whole filesystem same policy
- `mkfs.ubifs -r rootfs/ -m 2048 -e 126976 -c 1024 -o ubifs_crypt.img -b ddeeaaddbbeeeeff -K ubifs_masterkey.bin`
- ddeeaaddbbeeeeff is the key descriptor, see fscryptctl

Filesystem integrity: Using ubifs authentication with mkfs.ubifs

- Just like for fscrypt
- We use the signing key from the kernel build
- `mkfs.ubifs --hash-algo=sha256 --auth-cert=signing_key.x509 -r rootfs -e 126976 -o ubifs_auth.img -c 1024 -m 2048 --auth-key=signing_key.pem`

The “key” to success

- No human interaction wanted (e.g. mount must not ask for passwords)
- Key material must be stored in device itself to unlock device
- Attacker must not extract key
- Major challenge
- No way without support from hardware

The "key" to success: Naive approach

- Derive key from hardware properties
- CPU ID, MAC from network card, etc...
- Security by obscurity, IMHO
- More often used than you'd assume

The “key” to success: External Secure Element (TPM, etc.)

- Can store key material in a secure way
- Problem: Doing all crypto on the secure element is slow
- To utilize CPU, key needs get transferred into main memory
- Attacker can read the key while it is transferred
- Common attack: Bitlocker TPM sniffing

The “key” to success: Internal Secure Element (i.MX CAAM, etc.)

- Some SoC have a built in secure element
- e.g. i.MX CAAM or DCP
- In short: SoC can do AES with a fused key
- Typical use case: Store encrypted FDE key on distrusted location
- Problem: Fails if attacker can execute code, you need verified boot
    - Applies to the external secure element case too

The “key” to success: Key not in main memory

- Common requirement: KEY MUST NO RESIDE IN RAM!!!11elf
- Technically possible if you have a secure element
- Keep in mind:
  - Some mechanisms need the key in plaintext and do manual key derivation, e.g. fscrypt
  - Linux’s page cache is not your friend
  - Consider RAM encryption too
  - Know your threat model!

A few words on performance

- Crypto on SoC can be slow
- Crypto accelerators are not always faster
  - Filesystem encryption/auth is not their use-case
- Consider using AES-128 instead of AES-256
- When using dm-crypt, consider `no-read-workqueue` and `no-write-workqueue`
- Do your own benchmarks!

Summary

- Know your threat model
- There is no one-fits-all solution
- Know your threat model
- Full disk encryption is the last resort
- Know your threat model
- Storing the key material is the hard part
- Know your threat model

Further reading

- https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/dm-crypt.html
- https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/verity.html
- https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/dm-integrity.html
- https://www.kernel.org/doc/html/latest/filesystems/fscrypt.html
- https://www.kernel.org/doc/html/latest/filesystems/fsverity.html
- https://www.kernel.org/doc/html/latest/filesystems/ubifs-authentication.html
- https://www.spinics.net/lists/linux-mtd/msg08477.html
- https://www.jakoblell.com/blog/2013/12/22/practical-malleability-attack-against-cbc-encrypted-luks-partitions/
- https://pulsesecurity.co.nz/articles/TPM-sniffing
- https://blog.cloudflare.com/speeding-up-linux-disk-encryption/

Thank you!

Questions, Comments?

David Gstir
david@sigma-star.at

Richard Weinberger
richard@sigma-star.at