Key Chaining for Access Control and Instant Secure Deletion

James A Sutherland
University of Abertay Dundee

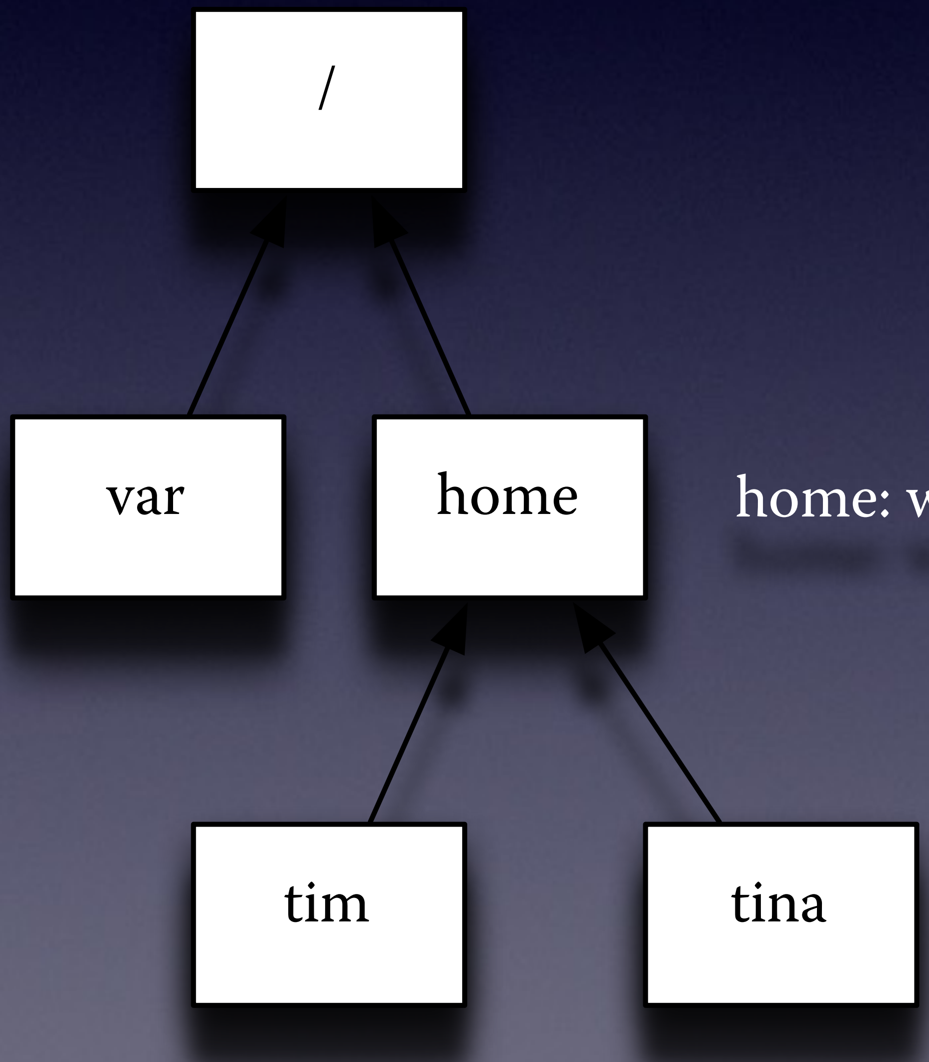
Background: Unix FS 1

- Standard Unix principals:
  - User
  - Group
  - Other
- Permission bits:
  - Read
  - Write
  - Execute

Background: Unix FS 2

- Each file is just a numbered *inode*
- Directory: list of name => node number
    - The directory is itself a file, with the same ACL

Traversal Example



home: write(root), read(users)

tina: write(tina)

Cold Boot Attack

- Replace the OS with your own
  - Easy to change or ignore any flag you like!

Read Protection

- That one’s easy: encrypt, hide the key.
- Different key per block (convergent encryption)
- Instead of “block n” store “block n with key/hash X”

Write Protection

- Slightly more complicated…
  - Instead of a straight checksum (ZFS SHA256):
    - Use a signature. Private key == write key.

Execute

- Trick question - reading IS executing!
- Except for directories:
  - ’execute’ = open by name
  - ‘read’ = enumerate names

Directories

- This is the clever bit…

- Instead of just the inode number, store key list too

- So, can’t access a file except via directory

  - (“Traverse checking”: Unix does, NT doesn’t by default, though the NSA kit changes that)

Deleting Files

- Normally, either ‘mark’ deleted (a flag)
    - Easy to recover with forensic tools
    - Still requires marking every file individually
- Or overwrite to prevent recovery
    - Takes O(n) time in file size: delete 10Gb is slow!

Faster Deletion

- Delete a file: zero the key, contents not recoverable
- Zero a directory’s key, whole directory gone in one
  - (Append its block list to the free list for later reuse)

Deduplicating Encrypted File Contents

- Convergent encryption: use a hash of each block as the key
- Good: can’t decrypt block without already knowing either the contents, or its hash
- But, privacy issue: allows *detection* of blocks of data
  - “Has this user got a copy of Windows 10?”

Deduplicating Securely With Privacy

- Instead of a straight hash, use HMAC
- Prevents searching without knowing the key
- Tradeoff between effectiveness and privacy
- Inherent information leak through dedupe count:
  - Add the target block(s), check free space counter
- Mitigation: deduplication domains
  - Write access to a DD implies ability to search for collisions

Performance

- What's the overhead of all this hashing and encrypting?
  - Smaller than you'd think
  - Hash in memory as part of async write path
    - No latency impact
    - Extra write needed to update reference checksum though
  - Decrypt and check checksum on read
    - Some latency impact, but no extra seeks
  - Extra 4-7k (Poly1305, AES) of code in FS codebase; ~30k cycles for 4k (from http://cr.yp.to/)

Authentication

- The file system’s job is normally confined to *authorisation* rather than authentication

- However, if we’re storing the user’s keys …

- Store with other account data, part-encrypt with hash of password

- Wait - password - what about SSH key auth?

  - Could encrypt the user data with each public key too

Project Overview

- Mostly, surprisingly small extensions to ext4 structures
    - (Adding hash/key to each block reference, not using extent mode)
    - Plus changing the free list more significantly, adding a directory-delete operation
- Authentication and SSH extensions much further off