EXT4 Encryption

Harder, Better, Faster, Stronger

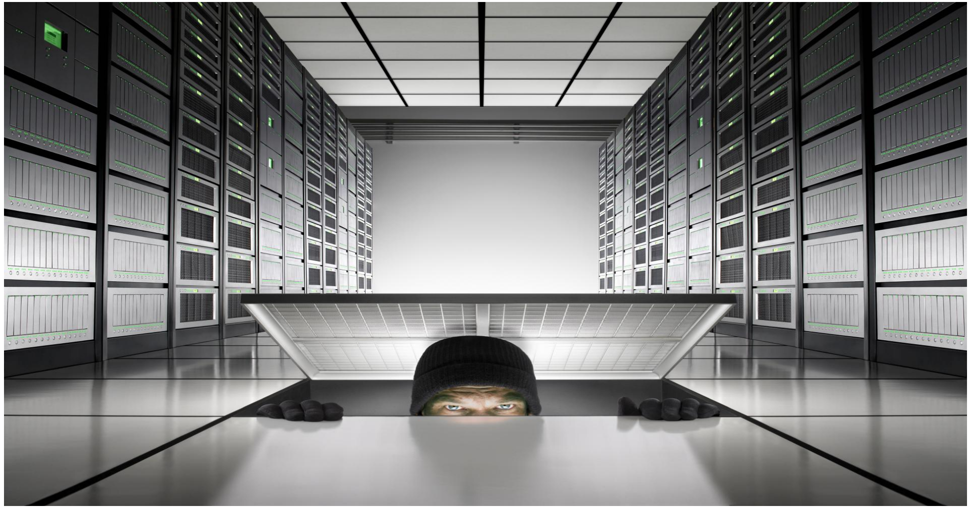
Google

Agenda

- State of Linux storage encryption

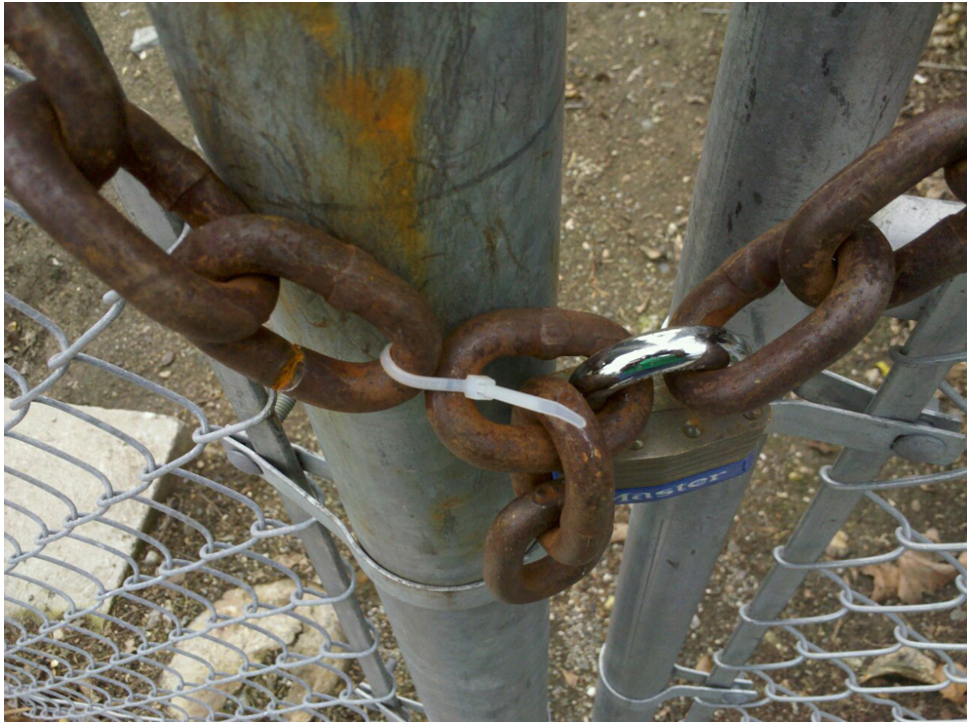
Agenda

- State of Linux storage encryption
- The Cloud, the Device, and Your Data: Adversarial Models



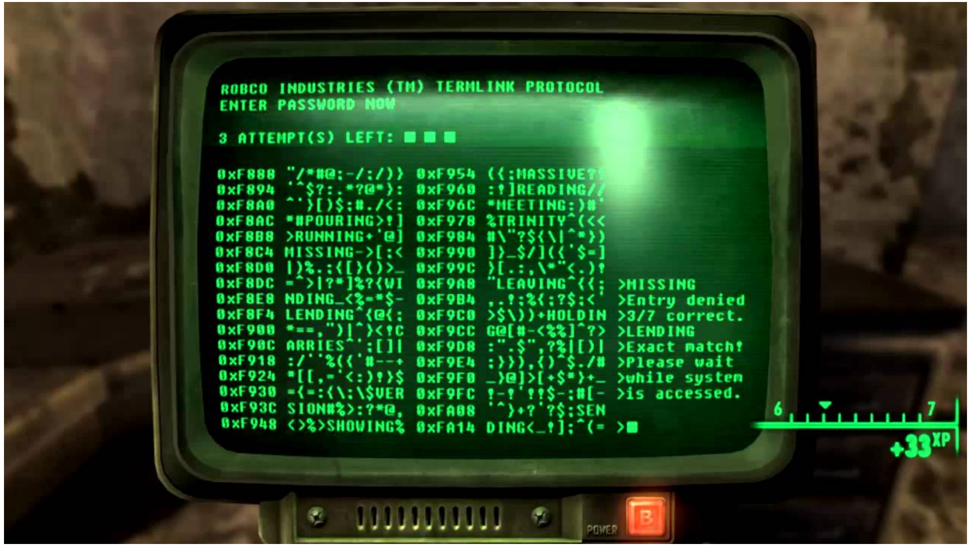
Agenda

- State of Linux storage encryption
- The Cloud, the Device, and Your Data: Adversarial Models
- Encrypting with Integrity



Agenda

- State of Linux storage encryption
- The Cloud, the Device, and Your Data: Adversarial Models
- Encrypting with Integrity
- **Key Management and Protection**



Agenda

- State of Linux storage encryption
- The Cloud, the Device, and Your Data: Adversarial Models
- Encrypting with Integrity
- Key Management and Protection
- Discussion

The State of Linux Storage Encryption

- Block Device Encryption (dm-crypt, TrueCrypt)
  - **Great for single-tenant devices**, problematic for the Cloud
- File-level encryption (eCryptfs)
  - Useful for some multi-tenant devices (e.g., Chromium OS), **many Cloud applications**
  - eCryptfs issues: Correctness, performance, **mixed benefits from stacking**
- Both lack strong encryption options (encryption with **integrity**)
  - Necessary properties: **IND-CCA2**, IND-CPA
  - Integrity data management introduces **complexity**

Adversarial Models

- **File level encryption** primarily targets **multi-tenant systems**
- **Extends run-time isolation protections** to the storage layer to protect against some (not all) online and offline attacks
    - Total Security ←→ **Risk Mitigation**
    - **Ring 0 compromise** remains a tough scenario to counter
- Increasing case for Cloud security benefit with Intel SGX (a.k.a. secure enclaves) coming in Skylake
    - If only we could keep the **keys** for an app **inside an enclave**, yet still usable by the kernel
        - TRESOR (keys in debug registers) can help against cold boot attacks, but that’s not the Cloud (multi-tenant) threat model

Adversarial Model: Phase 1

- **Single point-in-time permanent offline compromise** of the block device content, where loss of confidentiality of file metadata, including the file sizes, names, and permissions, is tolerable
- AES-256-XTS
  - Insecure against multiple point-in-time observations
  - 256 bits should be enough for everybody
    - Actually, 128 bits is, but enterprise policy has settled on 256
- No encryption metadata
- Patchset delivered to fsdevel for comment July 23rd

Adversarial Model: Phase 2

- **Occasional** temporary offline compromise of the block device content, where loss of confidentiality of file metadata, including the file sizes, names, and permissions, is tolerable
  - “Occasional”: Adversary can read and/or manipulate the offline ciphertext and/or authentication tags on the order of dozens of times
- **AES-256-GCM**
  - Requires conformance with NIST SP 800-38D recommendations
- Encryption **metadata**
- Extension to patchset underway
  - I’ve got sibling files mostly working

Adversarial Model: Phase 3

- Occasional temporary offline compromise of the block device content, where loss of confidentiality of **some** file metadata, including the file sizes, and permissions, is tolerable
    - **File names** will be **encrypted** (with integrity)
        - If we can figure out how to do it sanely

Adversarial Model: Phase 4

- Occasional temporary offline compromise of the block device content, where shared users on a mount are privy to other users' file metadata, including the file sizes and permissions
  - **Directory inodes** will be **encrypted** (with integrity) using a mount-wide key

Adversarial Model: Phase 5

- Something addressing the Integrity Measurement Architecture (IMA) adversarial model, only a faster approach
  - **Per-page validation** vs. entire-file validation
- For IMA, memory attacks are out-of-scope
  - Another approach: **reduce the measurements to encryption keys**
    - Persistent kernel compromise vs. Recoverable kernel compromise
  - **One-time measurement** compared against the trusted list of measurements at **time of provision**
  - **Sign the measurement** for each file with the per-file key; store in protector set
  - Per-page validation occurs during active I/O

Encrypting With Integrity

- If you don’t have data **integrity**, you very well may not have data confidentiality either
    - 2011 Attack against XML encryption in Apache Axis2: 1 byte of plaintext for every 14 rounds of ciphertext manipulations

How to Break XML Encryption*

Tibor Jager
Horst Görtz Institute for IT Security
Chair for Network- and Data Security
Ruhr-University Bochum
tibor.jager@rub.de

Juraj Somorovsky
Horst Görtz Institute for IT Security
Chair for Network- and Data Security
Ruhr-University Bochum
juraj.somorovsky@rub.de

RACT

cryption was standardized by W3C in 2002, and is nted in XML frameworks of major commercial and rce organizations like Apache, redhat, IBM, and t. It is employed in a large number of major web-plications, ranging from business communications,

tributed applications. The use of XML as core tax, e.g. for major business, e-commerce, financ care, governmental and military applications, has broad adoption of XML Encryption to protect c data—especially, but not exclusively, in the cont Services. On the technical level, the XML Encryp

Encrypting With Integrity

- HMAC over the ciphertext works
  - Slow for now; will get faster with Skylake SHA1/SHA256 acceleration
- **AES-GCM** incorporates an integrity measurement (GHASH) into the encryption and chaining process
  - Benefits from CLMUL acceleration in current-generation Intel hardware
    - Sandy Bridge: 2.75 cycles/byte, Haswell: 1.1 cycles/byte, Skylake: Faster...
  - Brittle; IV reuse is “sudden death”

Encrypting With Integrity

- Strong cryptographic integrity requires **additional data** per segment of verifiable data
- Once we’ve crossed that bridge, we can also generate a **unique IV** per block device segment offset
    - Hard requirement for GCM
    - Protection against injected plaintext attacks
- **One-to-one mapping** of plaintext blocks to ciphertext blocks **no longer holds**
    - Transactional semantics required for correctness
    - Where can we best **manage this complexity**?

Key Management and Protection

- eCryptfs model
    - **Per-file keys**, wrapped and stored in metadata for each file
    - Mount-wide key that wraps the per-file keys
    - Userspace tools do higher-level key management functions
    - Complete reliance on kernel integrity
        - On multi-tenant systems, this is already an accepted risk
        - Maybe we can do a little better
            - KASLR + obfuscation of key material in ring 0 memory
            - DMA attacks, etc. -- need more hardware support, or all crypto happens in ring 3 under SGX
            - FUSE redux, only with add'l context switch penalty

Key Management and Protection

- EXT4 model
  - Same as eCryptfs, only store metadata in xattr
    - And it’s correct, fast, and reliable
  - Per-mount keys no longer make sense
    - Wrapping key specifiers/policy in parent dir xattr?
    - IOCTL-based?
    - User session-based (e.g., policy in user session keyring)?

Discussion

- Basic approach
  - Hook EXT4 data path
  - Bounce pages for write, BIO callback for read
  - Sibling file for metadata
    - Per-block metadata?
- Potential features
  - In-place conversion
  - Versioning
  - Sub-file encryption contexts
- Distro integration

<EOP>

Mike Halcrow
mhalcrow@google.com

Ted Ts'o
tytso@google.com

Backup Slide: Q&A: Why aren't you doing this in XFS or BTRFS first?

A: Because Google is using EXT4 on Chrome OS and in its data centers.

I can probably find some time to review encryption patches from the XFS and/or BTRFS teams. Or maybe even talk to them.