Encrypting Virtual Memory

Niels Provos

provos@citi.umich.edu

CITI - Center for Information Technology Integration

University of Michigan

 Copyright Niels Provos 2000

Overview

1. Introduction
2. Related Work
3. Virtual Memory System
4. Swap Encryption
5. Performance Evaluation
6. Conclusion

 Copyright Niels Provos 2000

Introduction

- A Cryptographic file system protects confidential data from unauthorized access.
- The proper cryptographic key is required to read its contents.
- However, the virtual memory system's **backing store** is generally **unprotected**.
- Passwords and pass phrases reside in it long after they have been typed in, even across reboots.

 Copyright Niels Provos 2000

Introduction

- A user
  - expects that all confidential data vanishes with process termination,
  - is unaware that sensitive data may remain on backing store.
- When an attacker **compromises** the operating **system's integrity**
  - by gaining root privileges,
  - or by physical access to the machine itself

  she also **gains access** to **sensitive data** retained in backing store.

 Copyright Niels Provos 2000

Introduction

- Our solution is to encrypt pages before they are written to secondary storage
- When the pages are brought back into physical memory, they are decrypted
- Each page has an associated encryption key.
- Encryption keys are destroyed, when they are no longer needed

 Copyright Niels Provos 2000

Related Work

- Data protection with the "Cryptographic File System" by Blaze,
- Data hiding with the "Steganographic File System" by Anderson, Needham and Shamir.

$\Rightarrow$ data on secondary storage can reveal the **content** and **existence**.

 Copyright Niels Provos 2000

Related Work

- Erasing the data on secondary storage could achieve the same as encryption,
- but Gutmann has shown that it is very difficult to thoroughly delete data from magnetic-media.

 Copyright Niels Provos 2000

Virtual Memory System

- Virtual Memory increases the address space visible to application beyond the limits of physical memory.
- Data that does not fit into physical memory is saved on secondary storage.
- When a process accesses a page that has been stored on secondary storage a page fault occurs.
- The page fault causes the page to be restored from backing store.

 Copyright Niels Provos 2000

Virtual Memory System

- Secondary storage
  - is usually slower than RAM,
  - is non-volatile, data persists over system shutdowns.
- Confidential data can survive on it beyond a user's expectations.
- At CITI we found,
  - login passwords,
  - PGP pass phrases,
  - email messages, ...

 Copyright Niels Provos 2000

Virtual Memory System

Possible solutions:

- Avoid swapping completely: not a general solution, many applications require address space bigger than physical memory
- Use `mlock()` to prevent special memory areas to be paged out: applications need to be rewritten, reduces effectiveness of VM system, can result in worse performance

$\Rightarrow$ use **encryption** to protect confidential data.

 Copyright Niels Provos 2000

Virtual Memory System

Encryption comes in several different flavors:

- User program installs own encrypting pager:
  - increases complexity,
  - requires applications to be modified,
  - difficult design decision about crypto.
- VM system swaps to a file in a cryptographic file system.

 Copyright Niels Provos 2000

Virtual Memory System

In contrast to **common use of encryption**, we require

- when a page is no longer referenced, its encryption key should be destroyed after a time period (t_R) has passed,
- only the virtual memory pager should be able to decrypt pages

Best protection with $t_R = 0$, also meets user's expectation that her confidential data is deleted with process termination.

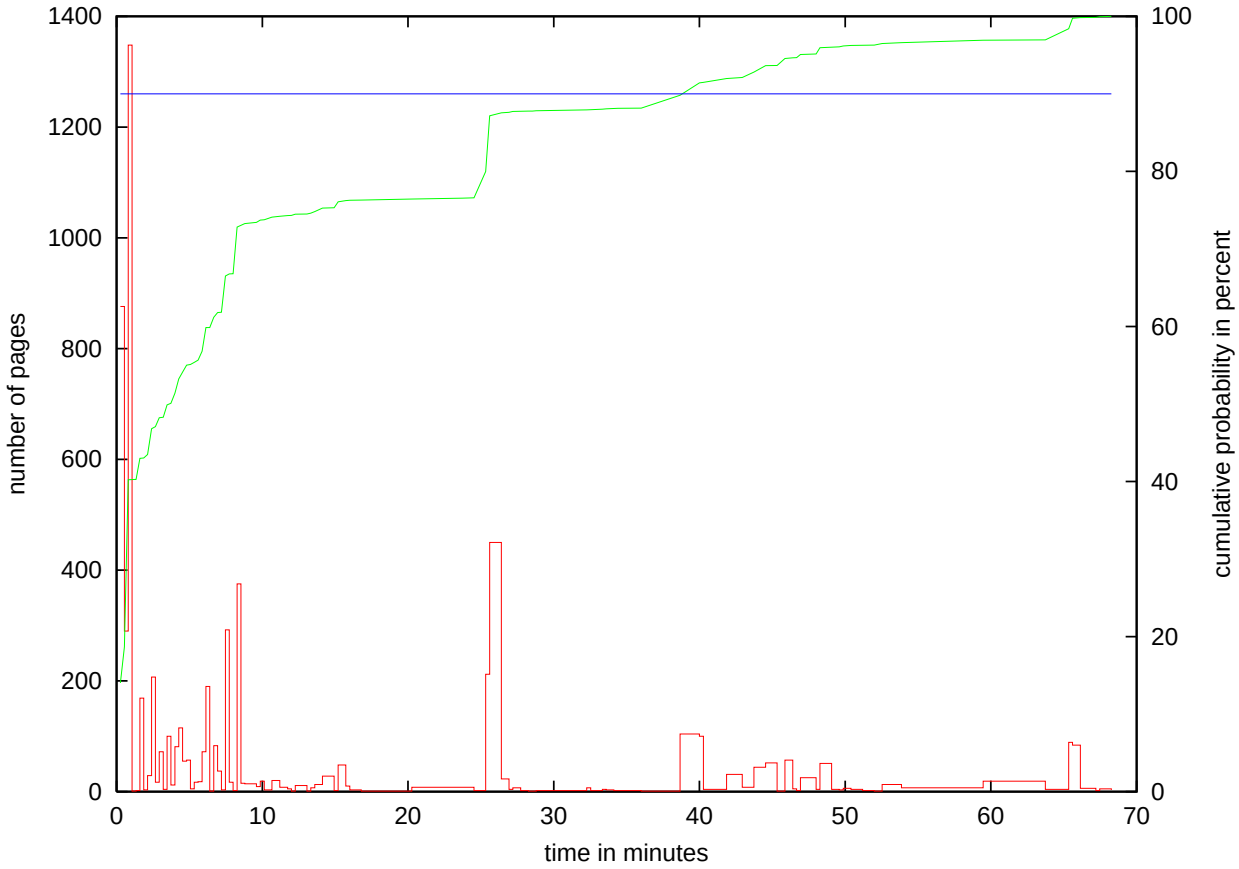
 Copyright Niels Provos 2000

Virtual Memory System

- $t_R = 0$ too impractical, we guarantee $t_R <$ system uptime, but attempt to minimize average t_R,
- use **volatile encryption keys**
  - valid maximally for the duration of the system's uptime
  - completely independent of each other $\Rightarrow$ perfect forward secrecy
  - no complicated key management.
- $\Rightarrow$ employ encryption at pager level.

 Copyright Niels Provos 2000

Virtual Memory System



- Most pages remain only for a few minutes, correlation: unnecessary zeroing, bad impact on system performance

 Copyright Niels Provos 2000

Virtual Memory System

In comparison,

- deleting data by erasing incurs extra seek time and additional I/O,
- erasing a page with encryption is fast, just destroy the encryption key,
- encryption provides better protection against physical attacks, mere possession of disk is not sufficient,
- reliably erasing data from magnetic-media is difficult, does not matter for encryption.

 Copyright Niels Provos 2000

Swap Encryption

- Encryption and decryption are separated: policy decision vs. need of decryption
  - Policy: encrypt everything, only encrypt data from cfs, etc...
  - Decryption: need to remember which pages to decrypt, keep a bitmap $\Rightarrow$ allows change of encryption policy.

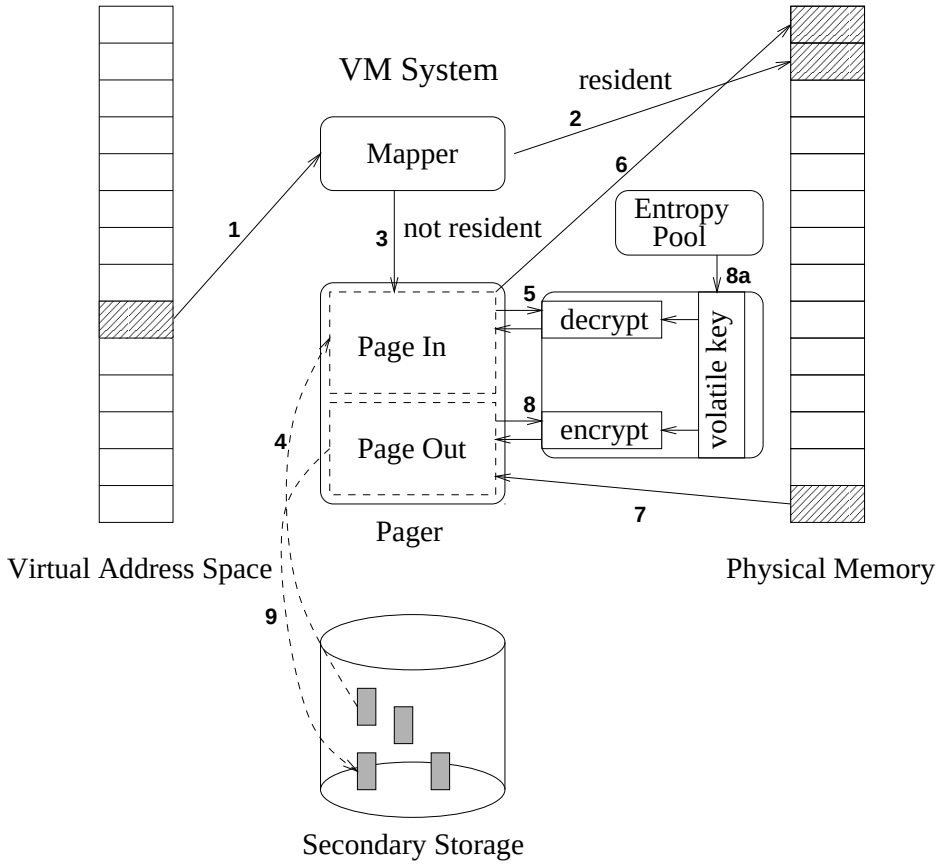
 Copyright Niels Provos 2000

Swap Encryption

- Keep upper bound on t_R small by dividing the backing store into sections of 512 KByte, each section has
  - a 128-bit cryptographic key,
  - reference counter,
  - and an expiration time.
- 256 MByte backing store requires 14KB of memory for keys.
- Section's 128-bit key is created randomly on first use.
- If a section's reference counter is 0, its key is destroyed.

 Copyright Niels Provos 2000

Swap Encryption



Overview of the swap encryption process

 Copyright Niels Provos 2000

Swap Encryption

Cipher Selection.

For swap encryption, a cipher needs to fulfill:

- Encryption and Decryption need to be fast compare to disk I/O
- Generation of the cipher's key schedule has to be inexpensive compared to encrypting a page.
- Cipher has to support encryption and decryption on page by page basis, can not use stream cipher.

 Copyright Niels Provos 2000

Swap Encryption

- Schneier's Blowfish encryption algorithm not suitable:
  - key schedule computation is very slow
  - key schedule requires a lot of memory
- Use **Rijndael**:
  - is finalist in advanced encryption standard (AES) competition,
  - 128-bit blocks and 128-bit keys,
  - round transformation does not have Feistel structure, instead different layers,
  - is faster in all aspects compared to Blowfish.

 Copyright Niels Provos 2000

Swap Encryption

- Key schedule computation cost is amortized when encrypting a single 4 KByte page. (352 cycles vs. 357 cycles)
- We use the cipher in cipher-block chaining (CBC) mode.
- Encrypted block number is used as 128-bit initialization vector (IV)
  - each page is encrypted uniquely,
  - try to avoid cipher text only attacks.

 Copyright Niels Provos 2000

Swap Encryption

- Security relies on good encryption keys.
- Require a good source of randomness.
- **Entropy pool** collects entropy from many physical events observable by the operating system:
  - inter-keypress timing from terminals,
  - arrival time of network packets,
  - finishing time of disk requests.
- Not practical for an attacker to observe all events.

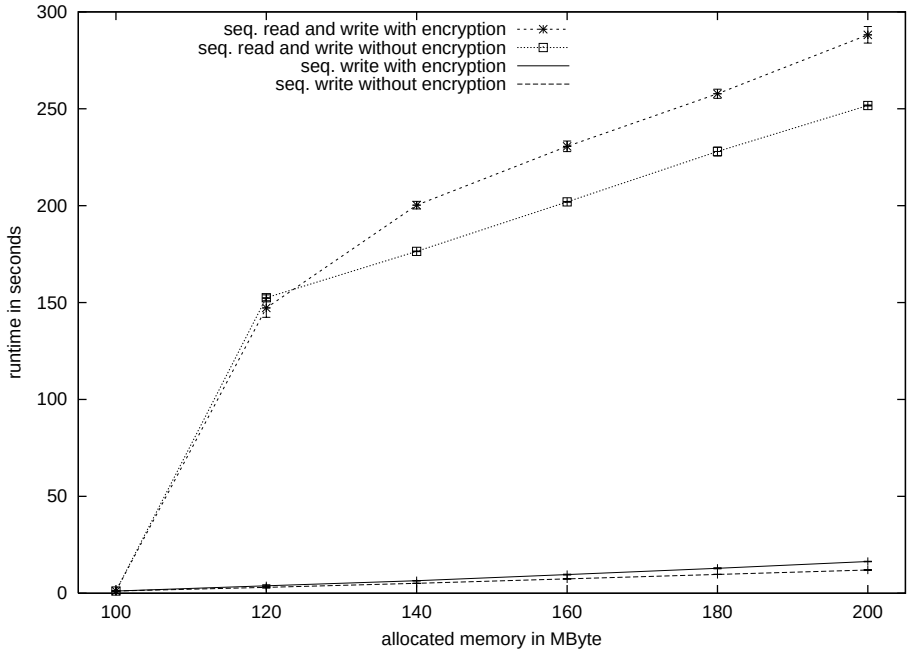
 Copyright Niels Provos 2000

Swap Encrytion

- Use ARC4 stream cipher to extract random encryption keys.
- RC4's internal state is initialized by the entropy pool.
- Frequently reseed RC4's state to prevent none-uniform output

 Copyright Niels Provos 2000

Performance Evaluation



- Running OpenBSD 2.6-current with UVM with 6 GByte Ultra-DMA disk, 7.5MByte/s write and 6.3 MByte/s read.
- Micro benchmark fills memory with zeros and reads it.
- Runtime increase for reads about 14%, for writes between 26%-36%

 Copyright Niels Provos 2000

Performance Evaluation

- Macro benchmark using ImagicMagick: magnify 960 $\times$ 1280 image and rotate by 24^o.
- For magnification by 2.5 runtime increases nearly by 70%.
- However, we believe that the overhead is still acceptable.

 Copyright Niels Provos 2000

Conclusion

- Confidential data can remain on backing store.
- Looked at several alternative solutions, encrypting data on backing store with volatile random keys has several advantages.
- Demonstrated acceptable performance and a viable solution.
- Software is freely available, contact the author.
- Acknowledgments:
  - Patrick McDaniel and Peter Honeyman for reviews and comments,
  - Chuck Lever for getting me interested in swap encryption,
  - Artur Grabowski for help in understanding UVM,
  - David Wagner for feedback on cipher selection.

 Copyright Niels Provos 2000

Physical Memory

- RIO shows that physical memory can be persistent across reboots.
- However, it is common practice to erase keys before application exit, *e.g.*, OpenSSL, OpenSSH, etc...
- Encryption protects against persistent storage of data before the application can clean up.

 Copyright Niels Provos 2000