SeMiNAS: A Secure Middleware for Wide-Area Network-Attached Storage

Ming Chen
Erez Zadok
{mchen, ezk}@cs.stonybrook.edu

Arun O. Vasudevan
aov@nutanix.com

Kelong Wang
kelong@dssd.com

Stony Brook University
Nutanix
EMC²
DSSD
Outline

- Background & Motivation
  - Design
  - Implementation
  - Evaluation
  - Conclusions
Cloud Computing

Accessibility

Scalability

Agility

Productivity

Availability

Economy

Flexibility
Security Concerns of Cloud

- Raised by cloud nature
  - Opaque & intangible
  - Multi-tenant
  - Large exploit surface
  - Complexity (buggy)

- Intensified by high-profile incidents
  - Silent data corruption
  - Leak of intimate photos of celebrities
  - Leak of user accounts and credentials
Securing Cloud Storage

New challenges:
1. Cost-efficiency despite high latency
2. Heterogeneous clients & clouds
3. Complex storage stack
Outline

☑ Background & Motivation
➤ Design
☐ Implementation
☐ Evaluation
☐ Conclusions
Benefits of a middleware:
1. Easy management (a few proxies vs. many clients)
2. Simple key distribution without trusted third parties
3. Fit well with WAN caching and firewalls
Why Use NFSv4?

- Advantages over vendor-specific key-value stores
  - Open, pervasive, and standard
    - POSIX-compliant and cross-platform interoperability
    - Suffering less from data or vendor lock-in
  - Optimized for WAN
    - Compound procedures
    - Delegations
  - Richer semantics
    - Simplify application development
    - More optimizations: server-side copying, ADB

- Advantages over older versions
  - Easier administration with a single port
  - More scalable with pNFS
  - More secure with RPCSEC_GSS, ACL, and Labeled NFS
SeMiNAS Data Path

```
<\mathbf{C}, \mathbf{M}> = \text{AuthEncrypt}(\mathbf{K}, \mathbf{P})
<\mathbf{P}, \mathbf{\nabla}> = \text{AuthDecrypt}(\mathbf{K}, \mathbf{C}, \mathbf{M})
```

- **LAN**
  - `nfs_write(\mathbf{P})`
  - Client 1
  - Caching Layer
    - `Insert(\mathbf{P})`
  - Auth-Encrypt Layer
    - `<\mathbf{C}, \mathbf{M}> = \text{AuthEncrypt}(\mathbf{K}, \mathbf{P})`
    - `<\mathbf{P}, \mathbf{\nabla}> = \text{AuthDecrypt}(\mathbf{K}, \mathbf{C}, \mathbf{M})`

- **WAN**
  - `write_plus(\mathbf{C}, \mathbf{M})`
  - Client 2
  - Persistent Cache
  - Cloud
  - `nfs_read()`: \mathbf{P}
  - Read plus
    - `read_plus()`: `<\mathbf{C}, \mathbf{M}>`

SeMiNAS

Stony Brook University
Meta-Data Management

- Each SeMiNAS proxy has <SID, PubKey, PriKey>
  - Each proxy knows public keys of all proxies
  - Distributed via a secret channel or manually

- Each file has a unique symmetric file key
  - Encrypted by master key pairs
  - Encrypt each block with GCM:

- File layout:
NFSv4-Based Optimizations (1)

- NFS Data-Integrity eXtensions

<table>
<thead>
<tr>
<th>Alternatives</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concatenate a block and its MAC as a separate file.</td>
<td>Break close-to-open consistency</td>
</tr>
<tr>
<td>Uses a separate file for all MACs of a file.</td>
<td>Add extra I/O and disk seeks</td>
</tr>
<tr>
<td>Map a block to a larger block in cloud (16→20KB).</td>
<td>Waste space for small block sizes</td>
</tr>
</tbody>
</table>
NFSv4-Based Optimizations (2)

- Compound Procedures

- SeMiNAS Compounds
  1. Write header after creating a file
  2. Read header after opening a file
  3. Update header before closing a dirty file
  4. Read header when getting attributes
  5. Get attributes after writing to a file
Outline

☑ Background & Motivation
☑ Design
➢ Implementation
☐ Evaluation
☐ Conclusions
SeMiNAS Implementation

- NFS-Ganesha: a user-land NFS server
  - File System Abstraction Layer (FSAL) back-ends
  - FSAL_VFS, FSAL_PROXY, and stackable FSALs
Extending DIX to NFS

- Data Integrity eXtensions (DIX) in NFS
  - READ_PLUS
  - WRITE_PLUS
Implementation Details

**Details**
- Added caching and security layers in NFS-Ganesha
- Added support of multiple stackable layers
- Extended DIX further to NFS
- Cryptographic C++ library: cryptopp
- Pass all applicable xfstests cases

**Development efforts**
- 25 man-months of 3 graduate students over 3 years
- Added 13,000 lines of C/C++ code to NFS-Ganesha
- Fixed 11 NFS-Ganesha and 4 kernel bugs
Outline

☑ Background & Motivation
☑ Design
☑ Implementation
☑ Evaluation
☐ Conclusions
Setup & Workloads

● Experimental setup
  ◆ Five NFS clients: 1G RAM; 6-core CPU; 10GbE NIC
  ◆ SeMiNAS proxy: 64G RAM; 6-core CPU; 10GbE NIC for LAN; 1GbE NIC for WAN; 200GB SSD for cache
  ◆ Server: 64G RAM; 6-core CPU; 1GbE NIC; 20GB virtual SCSI DIX disk backed by RAM

● Workloads

<table>
<thead>
<tr>
<th>Micro-Workloads</th>
<th>Filebench Workloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random file read/write</td>
<td>NFS Server</td>
</tr>
<tr>
<td>File creation</td>
<td>Web Proxy</td>
</tr>
<tr>
<td>File deletion</td>
<td>Mail Server</td>
</tr>
</tbody>
</table>
NAS because SeMiNAS uses compound procedures to pack network packets. In contrast, the algorithm favors SeMi-TCP Nagle trades off latency for longer network delay no matter if the cache is on or off. As shown in Figure 6, SeMiNAS has only negligible performance loss compared to the baseline. Depending on the number of threads, SeMiNAS has different performance gains ranging from 10ms to 30ms network delays.

5.2.2 File-Creation Workload

Different R/W Ratios

(a) Persistent Cache (FSAL-PCACHE) Off

(b) Persistent Cache (FSAL-PCACHE) On

Without cache, the baseline proxy's throughput decreases (i.e., by about 46% to +4%) compared to SeMiNAS. SeMiNAS consistently outperforms the baseline, especially in write intensive environments and generates many small file handles to specify the file. Those extra lookups without the persistent cache.

The number of threads influences the performance between the proxy and the server. More threads bring more coalescing opportunities; otherwise, the extra waiting of TCP Nagle adds extra delay to outbound packets in the hope of optimizing wide-area bandwidth. This is because SeMiNAS does not trade off latency.

Figure 7 shows the results of deleting files, where SeMiNAS have the same throughput as the baseline with and without the persistent cache.

Filebench's NFS-Server workload emulates the I/O activities of this file-creation workload, which contains 10,000 1KB-to-1700KB-headers) together to form larger packets. This is because SeMiNAS makes file creation 35% faster than the baseline-cache.

The workloads perform a variety of operations including open, read, write, append, and close. The number of threads influences the performance between the proxy and the server. More threads bring more coalescing opportunities; otherwise, the extra waiting of TCP Nagle adds extra delay.
SeMiNAS makes file creation faster

- TCP Nagle Algorithm
- Multiple threads sharing one TCP connection
- SeMiNAS write extra file headers
Filebench NFS-Server Workload

- SeMiNAS performance penalty
  - 8–17% without cache
  - 18–26% with cache
  - Decreases as network delay increases
SeMiNAS makes web-proxy
- 4–6% slower without cache
- 9–19% faster with cache (because of TCP Nagle)
Outline

- Background & Motivation
- Design
- Implementation
- Evaluation
- Conclusions
Conclusions

- We proposed SeMiNAS to secure cloud storage
- We designed SeMiNAS to
  - Be a middleware
  - Take advantages of NFSv4 compounds, and
  - Data Integrity eXtensions
- We implemented SeMiNAS based on
  - Add security stackable file-systems layers
  - Extend DIX to NFS
- We evaluated SeMiNAS:
  - small performance penalty less than 26%
  - performance boost by up to 19%
Limitations & Future Work

- Limitations
  - Not safe against replay attacks
  - Does not handle side-channel attacks

- Future work
  - Efficiently detect replay attacks
    - Avoid using expensive Merkle trees
    - Synchronize file versions among proxies
  - File- and directory-name encryption
  - Transactional Compounds
    https://github.com/sbu-fsl/txn-compound
SeMiNAS: A Secure Middleware for Wide-Area Network-Attached Storage

Q&A

Ming Chen
Erez Zadok
{mchen, ezk}@cs.stonybrook.edu

Arun O. Vasudevan
aov@nutanix.com

Kelong Wang
kelong@dssd.com

Stony Brook University

Nutanix

NSF
Network File System (NFS)

- An IETF standardized storage protocol
- Provides transparent remote file access
- Shares files over networks
Methodology

- **Benchmaster**
  - Automate multiple runs of experiments
  - Launch workloads concurrently on clients
  - Periodically collect system statistics

- **Workloads**
  - Data-intensive workloads
  - Metadata-intensive workloads
  - Delegation workloads
  - Filebench macro-workloads
Random Read/Write

1:1 Read-Write Ratio

-10%

-34%

Throughput (Ops/Sec)

Network Delay (ms)

5:1 Read-Write Ratio

1:5 Read-Write Ratio

baseline-nocache
baseline-cache
seminas-nocache
seminas-cache
File-Deletion Workload

- Caching makes file deletion slower
  - Introduce extra network round-trip
  - Remove cache upon unlink()

- However, SeMiNAS does not make file deletion slower
Goal: Securely and efficiently store and share files in cloud for geo-distributed organizations.

Approach: take advantages of new opportunities in NFSv4 and Data Integrity eXtensions (DIX).
Kurma Architecture

Region 1

Clients

Secure Middleware

Region 2

Untrusted Public Clouds

Region 3

S3

Azure

Drive

metadata

metadata
Kurma Components

file system
meta-data

Secure Middleware

ZooKeeper

BookKeeper

Hedwig

Kurma FS

PCache

FSAL_KURMA

FSAL_PCACHE

NFS-Ganesha

Clients

Public Clouds

Other Middlewares

S3

Azure

Drive

<BK, BV>