Scalable Security for High Performance, Petascale Storage

Andrew Leung, Ethan Miller, Stephanie Jones
SSRC Retreat, May-June ’07
Why Scalable Storage Security?

❖ Demand for large, high-performance storage increasing
  • Scientific applications
    - National Laboratories
  • Business
    - Google, Yahoo
❖ Requires scalability!

❖ Security is necessary
  • Scientific research data is classified
    - Nuclear experiment data
    - National security documents
  • Business data and documents are internal and proprietary
    - Application data
    - Source code
Petascale, HPC Challenges

❖ Upwards of tens of thousands of nodes

❖ Data is large and distributed
  • Gigabyte size files, striped over thousands of devices

❖ Additional security threats
  • No physical security
  • Cannot assume secure network

❖ Demanding I/O patterns
  • Parallel
  • Bursty
  • Flash-crowds common
Related Security Work

- Per-block/object capabilities not feasible
  - Group authorizations per-device

- Integrity via shared keys
  - Too insecure, limits authorization grouping

- Revocation through explicitly invalidating all capabilities
  - Not scalable
Maat: Scalable Security

❖ Extended capabilities
  • Can authorize I/O for any number of clients to any number of whole files
  • Reduce number of capabilities

❖ Automatic revocation
  • Capability expiration = capability revocation
  • Revocation without any contact

❖ Secure delegation
  • Delegate access rights to other clients
  • Shift security to key possession
Authentication

❖ Every node has asymmetric key pair
  • Configured off-line

❖ Ticket authenticates client public key
  • Refreshed periodically
  • Similar to Kerberos tickets

❖ Shared session key between each client-MDS, client-OSD pair
  • Key seed provided in ticket
  • Authentication via HMACs
Extended Capabilities

- Authorize I/O for many clients to many files
  - Reduces capability generation
- Capabilities verified and cached as OSDs
  - Verification paid once per OSD
- Signed by MDS
  - More secure, all OSDs can verify a capability, affordable
Fixed-size Capabilities

- Capabilities name authorized users and files
  - Alice, Bob, Carol can read /foo, /bar, /usr/tmp
  - 0x23FA5 can read 0x17AC2
- Root hash of Merkle tree names users/files
- OSDs must know ‘root hash → list’ mapping
  - Query from client
  - Cached at OSDs
- Only done on first use
  - Cost amortized by reuse of capability
Grouping Authorizations

❖ Unix permissions semantics
  • Relatively static
  • Greatly limits capability generation, proactive

❖ Dynamic
  • Buffer incoming requests
  • Based on temporal access

❖ Prediction
  • Proactively include clients who will likely access file
  • Time/space overhead

❖ Client defined
  • Created by users/application
  • T10 SETATTR function
Automatic Revocation

- Capability expiration = capability revocation

- Capabilities have short lifetimes: ~5 min
  - Expiration is equivalent to revocation
  - Limit window of vulnerability

- Renew valid capabilities
  - Clients request renewal
  - MDS batches renewal request
  - Issue signed renewal ‘token’
  - Token passed during I/O, cached at OSD

- Shift cost from revocation to renewal
  - Can be more scalable
Capability Renewal

- Client proactively requests n capabilities to renew

- MDS issues renewal for n capabilities + m most recently used capabilities
  - M is large and for all clients
  - Extends lifetime ~5 minutes

- Recent capabilities have
  - Renewal requested in previous p time intervals
  - Purges unused capabilities

- Many capabilities renewed in single renewal token
- Other renewal request served by MDS token cache
Secure Delegation

❖ Group opening
  • A client opens a file on ‘behalf’ of other clients
  • POSIX HPC I/O openg()

❖ Delegation of privilege
  • A client may give file access to any other client
  • POSIX HPC I/O openfh()

❖ Capability must be transferable to any user
  • A ‘pure’ capability is insecure
Maat’s Secure Delegation

- Delegating opaque capability is insecure
- Shift security from capability to capability + key possession
  - Temporary asymmetric computation key pair
  - Securely distribute private key
- Clients submit capability and proof token with I/O
Implementation Details

❖ Integrated in Ceph, a petascale distributed file system

❖ Algorithms: PK = ESIGN, SK = AES, one-way = SHA-1
  • Built on top of Crypto++

❖ Grouping: Unix, Prediction (recent popularity), batching

❖ 1 MDS, 10 OSDs, 7 client nodes
  • Ran up to 20 clients on a single client node
  • Each client acts as unique user

❖ Compared to baseline insecure Ceph
❖ Compared to secure I/O without authorization grouping
Micro-benchmark

- Clients write 5MB to each of 6 shared files and 4 non-shared in 128KB chunks
- Unix groups of 10 clients
- Vary clients from 10 to 140

- Unix and prediction over 3 times better than no grouping
- Grouping does not noticeably reduce throughput
IOR2 Benchmark

- Client runs 2 random trace files
  - Each file represents a unique process
  - Modified for fresh file system

- Recent Popularity made few predictions
- Without grouping incurs 22% overhead
- Unix grouping 6% overhead
- Prediction 7% overhead
Future Work

❖ Address security with a formal analysis or proof
  • Show extended capabilities equivalent to traditional capabilities
  • Pi-Calculus

❖ Improve grouping authorization strategies
  • Our list is not exhaustive
  • Optimal solution depends on system

❖ Address on-disk security
  • Data on OSD is vulnerable to attack if OSD subverted
  • Encrypt data on disk
  • Share/rotate keys effectively
Conclusions

❖ Maat - Scalable security for high performance, petascale storage
  • Prevent unauthorized data access, allow revocation, guard against common attacks

❖ Access control with extended capabilities
  • Capabilities authorize many I/Os

❖ Automatic revocation
  • Capability expiration acts as revocation

❖ Secure delegation
  • Secure group file opening and delegation

❖ Results: 6-7% overhead compared to insecure Ceph