Building an Air Gap Server

Purpose: Secure long-term backup storage isolated from network threats
Key Requirements: Full disk encryption, physical security, validated data transfer
Target Environment: FreeBSD with ZFS and GELI encryption
Status: Powered off when not actively receiving updates

Overview

An **Air Gap Server** is a server that operates without network connectivity to protect critical backup data from remote attacks. A modified approach allows temporary network access for system updates while maintaining security boundaries.

The primary purpose is to store long-term backups with significantly reduced attack surface. Physical isolation combined with encryption provides defense-in-depth against:

- Remote network attacks (ransomware, unauthorized access)
- Physical theft or unauthorized access
- Compromised source systems

Critical: If the server must be stored in an unsecured location, full disk encryption is **mandatory**, not optional.

Core Security Principles

- **Physical Isolation** — Geographic separation from primary servers with controlled access
- **Defense in Depth** — Multiple security layers protect data at rest and in transit
- **Data Validation** — Verify integrity of all data transfers and scripts
- **Automated Reporting** — Track all operations for audit and monitoring
- **Powered Off by Default** — Server only active during updates or maintenance

Implementation Guidelines

Physical Security and Access Control

Ideal Configuration:

- Store in secure facility requiring authenticated access (e.g., Network Operations Center)
- Geographic separation from primary production servers
- Documented access procedures and audit logs

Fallback for Insecure Locations:

When secure facilities are unavailable:

- **Mandatory:** Full disk encryption (GELI or equivalent)
- **Mandatory:** Documented key management procedures
- **Recommended:** Physical locks, tamper-evident seals
- **Recommended:** Motion detection or access logging

Store encryption keys in a different physical location than the server. Consider splitting keys across multiple secure locations.

Encryption Strategy

This implementation uses FreeBSD with GELI disk encryption backing a ZFS filesystem.

At Rest Protection:

- Full disk encryption using GELI (minimum requirement)
- All data pools encrypted with strong passphrases or key files
- Keys never stored on the server itself

Split-Key Architecture:

For enhanced security, consider using split-key encryption where the final encryption key is derived from combining two separate key components. This enhances security by allowing the actual GELI key to be stored securely off-site, as it cannot be reconstructed without both components:

- **Two-Operator Model:** Each key component held by different operators
  - Requires both operators present to unlock encrypted data
  - Maximum security: no single person can access data alone
  - Higher operational overhead
- **Operator + Automated Model:** One key with operator, one on server
- Operator key: Physically carried by trusted operator
- Server key: Stored on automated script (never on target server)
- Keys combined via XOR or similar operation at decrypt time
- Balances security with automation needs

In Transit Protection:

- Transport media (external drives) fully encrypted
- Delta data encrypted before writing to transport media
- Encryption keys validated at both source and destination

Example GELI Setup:

```
# Generate a random key file (4096 bits = 512 bytes)
openssl rand 512 > /secure/path/geli.key
chmod 400 /secure/path/geli.key
```

```
# Initialize GELI encryption on disk using the key file
geli init -s 4096 -K /secure/path/geli.key /dev/ada0

# Attach encrypted device
geli attach -k /secure/path/geli.key /dev/ada0

# Create ZFS pool on encrypted device
zpool create backup /dev/ada0.eli
```

Key size of 4096 bits provides strong encryption. The key file should be stored securely and backed up to a separate location. Use `-P` flag to add passphrase protection in addition to key file.

Data Transfer Validation

Transport Media Requirements:

- Large capacity drives (match expected delta sizes)
- Encrypted filesystem (GELI, LUKS, or BitLocker) or Encryption of individual files in transit
- Labeled with GPT labels for automated mounting

Delta Monitoring:

Monitor transfer sizes to detect anomalies:

- Establish baseline delta sizes for normal operations
- Alert on deltas exceeding 150-200% of baseline
- **Large deltas may indicate ransomware on source system**

Data Integrity Verification:

```
# Generate checksum on source
zfs send pool/dataset@snapshot | tee >(sha256) > /mnt/transport/delta.zfs

# Verify checksum on air gap server
sha256 /mnt/transport/delta.zfs
```

Data Validation:

- All data encrypted with symmetric key at source
- Decryption failure automatically rejects the data
- Failed decryption indicates corruption or tampering
- Process terminates on any decryption failure

Script Validation and Maintenance

Air gap servers require special consideration for maintenance since they lack network access for updates.

Validated Script Execution:

Scripts may be deployed to perform maintenance tasks:

- ZFS scrubs and pool health checks
- Snapshot cleanup and rotation
- SMART disk monitoring
- System updates (if temporarily networked)

Script Deployment Process:

1. Scripts stored on source server and version controlled
2. Scripts encrypted with symmetric key before transfer
3. Air gap server must successfully decrypt before execution
4. Decryption failure prevents script execution and terminates process
5. Scripts run automatically during replication operations

Example Script Encryption/Decryption:

```
# On source server: encrypt script
openssl enc -aes-256-cbc -salt -in cleanup_script.sh \
  -out cleanup_script.sh.enc -pass file:/secure/transport.key

# On air gap server: decrypt and execute
openssl enc -aes-256-cbc -d -in cleanup_script.sh.enc \
  -out cleanup_script.sh -pass file:/secure/transport.key && \
  sh cleanup_script.sh || { echo "Decryption failed - aborting"; exit 1; }
```

Security through decryption: Scripts that cannot be decrypted with the correct symmetric key are rejected. Any decryption failure terminates the entire process to prevent execution of potentially tampered scripts.

Reporting and Audit Trail

Reporting Challenges:

- Air gap servers cannot send email reports
- No network access for remote monitoring
- Reports must be physically retrieved

Solution — Report Drive:

- Dedicated removable media for reports (USB drive, small HDD)
- Reports written to transport drive after each operation
- Administrator retrieves and processes reports manually

Report Contents:

- Timestamp of operation
- Data volumes transferred (size, snapshot names)

- Success/failure status of each operation
- Disk health (SMART status, ZFS pool health)
- Script execution results
- Any errors or warnings

Example Report Structure:

```
=== Air Gap Backup Report ===
Date: 2026-01-18 03:00:00
Operation: Incremental Backup
Source: production.example.com
Target: airgap-backup01

Datasets Processed:
  - pool/data: 45.2 GB transferred
    Latest: pool/data@2026-01-18_02:00:00
  - pool/databases: 12.8 GB transferred
    Latest: pool/databases@2026-01-18_02:00:00

Pool Health: ONLINE
Disk Status: All disks PASSED SMART checks

Maintenance Scripts Executed:
  - snapshot_cleanup.sh: SUCCESS (removed 3 old snapshots)
  - zfs_scrub.sh: SUCCESS (no errors found)

System Shutdown: 2026-01-18 03:45:00
Next Expected Update: 2026-01-25
```

Power Management

Default State: Powered Off

The air gap server should remain powered off except during:

- Scheduled data imports
- Manual maintenance operations
- Security audits

Benefits of Power-Off Strategy:

- Encrypted drives are locked (keys in memory are cleared)
- Eliminates risk of remote exploitation during off time
- Reduces hardware wear and power consumption
- Limits window of opportunity for physical attacks

Automated Shutdown:

Final script in maintenance chain should power off the system:

```
#!/bin/sh
```

```
# Final maintenance script - shutdown system

# Verify all operations completed successfully
if [ -f /var/run/backup_complete ]; then
    # Write final report
    echo "Backup completed successfully at $(date)" >>
/mnt/report/status.log

    # Sync all filesystem buffers
    sync

    # Unmount transport media
    umount /mnt/transport
    umount /mnt/report

    # Power off system
    shutdown -p now
else
    echo "ERROR: Backup did not complete. Manual intervention required." >>
/mnt/report/error.log
    # Do NOT shutdown - leave powered on for troubleshooting
fi
```

Do not configure automatic shutdown if backups fail. A powered-on system indicates problems requiring manual investigation.

Example Workflow

A typical weekly backup cycle:

Day 1 (Monday) — Source Server:

1. Automated script takes ZFS snapshots of all datasets
2. Calculates incremental changes since last backup
3. Encrypts delta data to transport drive with symmetric key
4. Encrypts maintenance scripts with same symmetric key
5. Operator notified that transport drive is ready

Day 2 (Tuesday) — Physical Transport:

1. Operator removes transport drive from source server
2. Drive physically transported to air gap location
3. Transport logged in access control system

Day 3 (Wednesday) — Air Gap Server:

1. Operator inserts transport drive and powers on server
2. Server boots, mounts transport drive
3. Automated script begins:

    - Attempts to decrypt delta files with symmetric key
    - Validates delta sizes against baseline
    - Imports ZFS datasets (decryption happens during import)
    - Attempts to decrypt and run maintenance scripts
    - Any decryption failure terminates the entire process
    - Generates report to report drive
    - Powers off system (only if all operations succeed)
4. Operator retrieves report drive for later review

Day 4 (Thursday) — Report Processing:

1. Operator reviews reports from air gap server
2. Verifies all backups completed successfully
3. Archives reports for audit trail
4. Updates monitoring dashboard

Day 8 (Next Monday):

1. Process repeats with fresh delta data

Pre-Implementation Checklist

```
[ ] Physical Security
    [ ] Secure location identified and documented
    [ ] Access procedures established
    [ ] Key storage locations determined
[ ] Hardware
    [ ] Air gap server procured and tested
    [ ] Transport drives procured (minimum 2 for rotation)
    [ ] Report drive procured
    [ ] All drives labeled appropriately
[ ] Encryption
    [ ] GELI encryption configured and tested
    [ ] Encryption keys generated and stored securely
    [ ] Key recovery procedures documented
    [ ] Transport drives encrypted
[ ] Software
    [ ] FreeBSD installed and hardened
    [ ] ZFS pools created and tested
    [ ] Replication scripts developed and tested
    [ ] Maintenance scripts developed and tested
    [ ] Symmetric transport keys generated and deployed
[ ] Procedures
    [ ] Backup schedule documented
    [ ] Transport procedures documented
    [ ] Report review procedures documented
    [ ] Key rotation schedule established
    [ ] Disaster recovery plan created
[ ] Testing
    [ ] Full backup cycle tested end-to-end
```

```
    [ ] Recovery procedures tested
    [ ] Failure scenarios tested
    [ ] Report generation verified
    [ ] Automated shutdown verified
```

Security Considerations

Threat Model:

This design protects against:

- ✓ Remote network attacks (ransomware, unauthorized access)
- ✓ Compromised source systems
- ✓ Physical theft (with encryption)
- ✓ Unauthorized physical access (with encryption)

This design does NOT fully protect against:

- ✗ Sophisticated attackers with physical access and unlimited time
- ✗ Compromised encryption keys
- ✗ Attacks on the transport process itself
- ✗ Insider threats with authorized access

Best Practices:

- Rotate encryption keys annually
- Test recovery procedures quarterly
- Review audit logs monthly
- Update maintenance scripts as needed
- Keep offline backups of critical configuration

Troubleshooting

Common Issues:

Problem	Symptom	Solution
Transport drive not mounting	Server unable to find `/dev/gpt/label`	Verify GPT label, check dmesg for device detection
Decryption fails	OpenSSL reports bad decrypt error	Verify correct symmetric key in use, check file integrity, investigate potential tampering or corruption
Large delta size	Delta exceeds baseline by 200%+	**Do not import** — investigate source system for compromise or legitimate growth
Server won't shutdown	Remains powered on after backup	Check `/var/run/backup_complete` flag, review error logs on report drive
ZFS pool won't import	Import command fails	Verify encryption key, check pool status with `zpool import -F`

Real-World Implementation

Client Requirements

A production deployment required the following specifications:

Requirement	Implementation
Replication Schedule	Monthly updates from in-house backup server to air gap server
Transport Media	3× 1.9TB SSD drives in rotation
Drive Rotation	One at source, one at target, one in transit — minimizes site visits
Security Model	Multi-layer encryption with split-key architecture
Location	Air gap server in unsecured location (mandatory encryption)
Automation	Fully automated with maintenance script execution

Security Architecture

Encryption Layers:

1. **At Rest (Target):** GELI full disk encryption on air gap server
2. **In Transit:** Symmetric key encryption for all data on transport drives
3. **Maintenance Scripts:** Encrypted with same symmetric key
4. **Split-Key Design:** Target GELI key derived from:
    - Server-resident key component (stored locally)
    - Operator-carried key component (physical transport)
    - Combined via XOR bitwise operation at decrypt time
    - Target GELI key stored securely to facilitate key rotation and recovery

Split-key advantage: Neither component alone can decrypt the air gap server. Compromise of a single key (server or transport) does not expose data.

Implementation Scripts

Custom automation scripts handle the complete workflow. Source code is available via Subversion:

Repository URL: `http://svn.dailydata.net/svn/zfs_utils/trunk`
Sub-project: `sneakernet`

Export the project:

```
mkdir -p /usr/local/opt
svn export http://svn.dailydata.net/svn/zfs_utils/trunk
/usr/local/opt/zfs_utils
```

Source Server Workflow:

1. Auto-detect operating mode (source vs. target)
2. Mount transport drive using GPT label detection

3. Verify transport drive processed by target (check status file)
4. Securely erase previous data from transport drive
5. Calculate incremental ZFS replication stream
6. Encrypt and write replication data to transport drive
7. Record latest snapshots sent (update status file)
8. Encrypt and write maintenance scripts to transport drive
9. Unmount transport drive
10. Email completion report to administrators

Target Server Workflow:

1. Mount transport drive
2. Detect operator-provided secure key (USB/separate media)
3. Combine server key with operator key (XOR operation)
4. Unlock GELI encrypted disks using combined key
5. Import ZFS pool
6. Save current snapshot list to state file (enable rollback if needed)
7. Decrypt and import replication streams from transport
8. Collect system statistics (pool health, disk status, capacity)
9. Decrypt and execute maintenance scripts
10. Generate detailed report and write to report drive
11. Unmount all media
12. Power off system

```
# Example: Simplified detection logic
# this is actually accomplished within sneakernet automatically, so
# not necessary. This just shows the logic used.
HOSTNAME=$(hostname -s)
if [ "$HOSTNAME" = "backup-source" ]; then
    # Source mode
    /usr/local/sbin/sneakernet --mode=source
elif [ "$HOSTNAME" = "airgap-target" ]; then
    # Target mode
    /usr/local/sbin/sneakernet --mode=target
else
    echo "ERROR: Unknown host" >&2
    exit 1
fi
```

Three-Drive Rotation Strategy

The three-drive rotation minimizes operational overhead:

Normal Operation Cycle:

Month	Drive A	Drive B	Drive C	Action Required
1	At Source (ready)	At Target	In Transit to Target	Operator: Deliver Drive C to target

Month	Drive A	Drive B	Drive C	Action Required
2	At Source (ready)	In Transit to Source	At Target (ready)	Operator: Collect Drive B from target
3	In Transit to Target	At Source (ready)	At Target	Operator: Deliver Drive A to target
4	At Target	At Source (ready)	In Transit to Source	Operator: Collect Drive C from target

Benefits:

- Each site visit handles both delivery and pickup
- No waiting time for drive processing
- Reduced frequency of site access (security benefit)
- Built-in offline backup (data exists on multiple drives)

Key Management Strategy

Symmetric Transport Key:

- Unique key per deployment
- Stored on both source and target servers
- Used to encrypt data and scripts on transport drives

Split GELI Key:

- Server component: Stored on target server (never leaves facility)
- Operator component: Carried by trusted operator (never stored at target)
- Combined at runtime via XOR: `final_key = server_key ⊕ operator_key`

Key Rotation Procedures:

If transport drive compromised:

```
# Generate new symmetric key
openssl rand 32 | xxd -p | tr -d '\n' > /secure/path/new_transport.key

# Deploy as maintenance script on next run
# Old data on compromised drive remains encrypted with old key
```

If operator key compromised, retrieve the geli key from secure storage in hex format, then run the following commands:

```
# Generate new key pair
openssl rand 32 > operator.key
xxd -p -c 999 operator.key > operator.key.hex
# retrieve server.key from secure storage in binary format and run the
following
# perl on-liner on them. This is not tested. The keys are in hex, not binary
# and the result is in hex (use xxd for two way processing)
perl -e '
    # Iterate over each byte index of the keys
```

```
    print join("",
        map {
            # Extract bytes and perform XOR
            sprintf("%02x",
                hex(substr($ARGV[0], $_, 2)) ^
                hex(substr($ARGV[1], $_, 2))
            )
        }
        0 .. (length($ARGV[0]) / 2 - 1)  # Calculate the number of bytes
    ) . "\n"  # Print the result
' 'operator.key.hex' 'server.key.hex'

# Operator must use new key on next visit after updating the key on the air
gap server
# If server.key is ever lost, must
```

Key rotation can be automated through maintenance scripts. New keys deployed during normal replication cycles without requiring emergency site visits.

Operational Benefits

This implementation balances security with operational efficiency:

Security Advantages:

- No single point of key compromise
- Lost transport drive: data remains encrypted
- Lost operator key: server data still protected
- Automated key rotation capability
- Audit trail via detailed reports

Operational Advantages:

- Minimal site visits (monthly vs. weekly)
- No waiting time for processing
- Fully automated operation (no manual commands)
- Email reports from source (connected)
- Physical reports from target (air-gapped)
- Automated maintenance without network access

References

- FreeBSD GELI Encryption Documentation
- FreeBSD ZFS Administration Guide
- OpenSSL Documentation — For symmetric encryption operations
- NIST Storage Encryption Guide

Related Documentation

- ZFS Replication Scripts — Automated snapshot and transfer scripts
- Encryption Key Management — Key generation, storage, and rotation
- Air Gap Incident Response Plan — What to do if compromise suspected
- Disaster Recovery Procedures — Restoring from air gap backups

From:
https://wiki.linuxservertech.com/ - **Unix Server Tech Knowledge Base**

Permanent link:
https://wiki.linuxservertech.com/unix/freebsd/system_builds/airgap/buildairgap

Last update: **2026/01/20 07:14**