CloudSync Ultra

Comprehensive Security Audit

16 Vulnerabilities Identified - Issue #58

Generated: January 14, 2026

Version: 2.0.16

CloudSync Ultra - Comprehensive Security Audit Report

Audit Date: January 14, 2026

Auditor: Security-Auditor (AI Agent)

Application: CloudSync Ultra v2.0

Platform: macOS

Audit ID: #58

Executive Summary

CloudSync Ultra is a macOS cloud synchronization application that interfaces with 42+ cloud providers through rclone. This security audit identified **16 security findings** across credential handling, encryption practices, input validation, and macOS security configurations.

Risk Summary

Severity	Count	Status
Critical	2	Immediate action required
High	4	Action required within 7 days
Medium	6	Action required within 30 days
Low	4	Recommended improvements

Critical Findings

VULN-001: Encryption Passwords Stored in UserDefaults (Plaintext)

Severity: CRITICAL

Location: `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/EncryptionManager.swift` (Lines 130-166)

Description:

Encryption passwords and salts are stored in UserDefaults instead of the macOS Keychain. UserDefaults stores data in plaintext plist files that are accessible to any process running under the same user account.

Vulnerable Code:

```
func savePassword(_ password: String) throws { UserDefaults.standard.set(password, forKey:
"encryption.password") } func saveSalt(_ salt: String) throws { UserDefaults.standard.set(salt,
forKey: "encryption.salt") } func savePassword(_ password: String, for remoteName: String) throws
{ UserDefaults.standard.set(password, forKey: "encryption_password_\(remoteName)") }
```

Impact:

- Encryption master passwords exposed in `~/Library/Preferences/`
- Any application or malware can read these credentials
- Compromises all encrypted data across all remotes

Remediation:

- Migrate all password storage to `KeychainManager`
- Use `kSecAttrAccessibleWhenUnlocked` for protection
- Implement secure deletion of legacy UserDefaults entries
- Add data migration for existing users

VULN-002: App Sandbox Disabled

Severity: CRITICAL

Location:
`/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/CloudSyncApp.entitlements`

Description:

The application has App Sandbox completely disabled (`com.apple.security.app-sandbox = false`). This removes macOS's primary application security boundary.

Current Configuration:

```
<key>com.apple.security.app-sandbox</key> <false/>
```

Impact:

- Application has unrestricted file system access
- No protection against privilege escalation attacks
- Malicious code can access all user data
- App Store distribution impossible without sandbox

Remediation:

- Enable App Sandbox with minimum required entitlements
- Request specific file access through entitlements:

- `com.apple.security.files.downloads.read-write`
- `com.apple.security.temporary-exception.files.absolute-path.read-write`
- Use Security-Scoped Bookmarks for user-selected directories
- Test all functionality with sandbox enabled

High Severity Findings

VULN-003: Command Injection via Unsanitized File Paths

Severity: HIGH

Location: `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/RcloneManager.swift` (Multiple locations)

Description:

User-provided file paths are passed directly to rclone process arguments without sanitization. Malicious filenames containing shell metacharacters could potentially be exploited.

Vulnerable Pattern:

```
process.arguments = [ "copy", localPath, // User-controlled, unsanitized
"\(remoteName):\(remotePath)", // User-controlled "--config", configPath, ... ]
```

Impact:

- Path traversal attacks
- Potential command injection via specially crafted filenames
- Unauthorized file access outside intended directories

Remediation:

- Implement path validation function:

```
func validatePath(_ path: String) throws -> String { let resolved = (path as
NSString).standardizingPath // Check for path traversal guard !resolved.contains("..") else {
throw SecurityError.invalidPath } // Whitelist allowed characters let allowed =
CharacterSet.alphanumerics.union(CharacterSet(charactersIn: "/_-.")) guard
path.unicodeScalars.allSatisfy({ allowed.contains($0) }) else { throw
SecurityError.invalidCharacters } return resolved }
```

- Apply validation before all Process arguments
- Use URL-based APIs instead of string paths where possible

VULN-004: Password Passed as Command Line Argument

Severity: HIGH

Location: `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/RcloneManager.swift` (Lines 1500-1524)

Description:

The `obscurePassword` function passes cleartext passwords as command line arguments to rclone. Command line arguments are visible in process listings.

Vulnerable Code:

```
func obscurePassword(_ password: String) async throws -> String { let process = Process()
process.executableURL = URL(fileURLWithPath: rclonePath) process.arguments = ["obscure", password]
// Password visible in ps output ... }
```

Impact:

- Passwords visible via `ps aux` during obscuring process
- Passwords logged in shell history if debugging
- Potential credential exposure to other local processes

Remediation:

- Use stdin to pass passwords:

```
func obscurePassword(_ password: String) async throws -> String { let process = Process()
process.executableURL = URL(fileURLWithPath: rclonePath) process.arguments = ["obscure", "-"] //
Read from stdin let inputPipe = Pipe() process.standardInput = inputPipe try process.run()
inputPipe.fileHandleForWriting.write(password.data(using: .utf8)!)
inputPipe.fileHandleForWriting.closeFile() ... }
```

VULN-005: Insufficient File Permission Validation

Severity: HIGH

Location: `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/RcloneManager.swift` (Lines 29-35)

Description:

The application sets the rclone binary permissions to 0o755 without verifying the binary's integrity or source.

Vulnerable Code:

```
if let bundledPath = Bundle.main.path(forResource: "rclone", ofType: nil) { self.rclonePath =
bundledPath try? FileManager.default.setAttributes( [.posixPermissions: 0o755], ofItemAtPath:
bundledPath ) }
```

Impact:

- No code signature verification of rclone binary
- Replaced binary would gain execute permissions
- Supply chain attack vector

Remediation:

- Verify code signature before execution
- Implement checksum verification
- Use Hardened Runtime with Library Validation

VULN-006: OAuth Tokens Stored Without Encryption

Severity: HIGH

Location: `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/KeychainManager.swift` (Lines 334-346)

Description:

While Keychain is used, OAuth tokens are stored with `kSecAttrAccessibleWhenUnlocked` which allows access whenever the device is unlocked. For high-value OAuth tokens, stronger protection is needed.

Current Implementation:

```
func saveToken(provider: String, token: String) throws { try save(token, forKey:
"token_\(provider)") }
```

Remediation:

- Use `kSecAttrAccessibleWhenUnlockedThisDeviceOnly` for tokens
- Implement token encryption before Keychain storage
- Add token expiry tracking and automatic refresh

Medium Severity Findings

VULN-007: Debug Logging to World-Readable Location

Severity: MEDIUM

Location: `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/RcloneManager.swift` (Lines 1784-1800)

Description:

Debug logging writes to `/tmp/cloudsync_upload_debug.log`, a world-readable location.

Vulnerable Code:

```
let logPath = "/tmp/cloudsync_upload_debug.log" let log = { (msg: String) in let timestamp =
Date().description let line = "\(timestamp): [RcloneManager] \(msg)\n" ... }
```

Impact:

- Sensitive file paths exposed in logs
- Potential credential leakage in debug output
- Information disclosure to other users

Remediation:

- Use Application Support directory for logs
- Set restrictive file permissions (0o600)
- Implement log rotation and automatic cleanup
- Remove debug logging from production builds

VULN-008: rclone.conf Contains Cleartext Credentials

Severity: MEDIUM

Location: `~/Library/Application Support/CloudSyncApp/rclone.conf`

Description:

While rclone obscures passwords, the config file is stored with standard file permissions and contains sensitive API keys, tokens, and obscured passwords.

Impact:

- Config file readable by any process under user account
- Obscured passwords can be reversed with rclone
- OAuth tokens and API keys in cleartext

Remediation:

- Set file permissions to 0o600 after creation
- Consider encrypting config file at rest
- Use FileProtection attributes on macOS
- Implement config file integrity checking

VULN-009: Missing Input Length Validation

Severity: MEDIUM

Location: `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/Views/ProtonDriveSetupView.swift`

Description:

User input fields for credentials lack length and format validation beyond basic empty checks.

Vulnerable Pattern:

```
private var canTest: Bool { !username.isEmpty && !password.isEmpty && is2FAValid } private var
is2FAValid: Bool { switch twoFactorMode { case .none: return true case .code: return
twoFactorCode.count == 6 case .totp: return !otpSecretKey.isEmpty // No max length } }
```

Impact:

- Buffer overflow potential in rclone
- Denial of service via extremely long inputs

- Memory exhaustion attacks

Remediation:

- Add maximum length validation (e.g., 1000 chars for passwords)
- Implement input sanitization
- Add format validation for email fields
- Rate limit authentication attempts

VULN-010: Insecure Temporary File Handling

Severity: MEDIUM

Location:
`/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/CrashReportingManager.swift` (Lines 89-112)

Description:

Crash logs and exported logs use predictable paths in temporary directories without secure file creation.

Vulnerable Code:

```
let exportDir = FileManager.default.temporaryDirectory
.appendingPathComponent("CloudSyncLogs_\(Date().timeIntervalSince1970)")
```

Impact:

- Race condition vulnerabilities (TOCTOU)
- Symlink attacks on temporary files
- Sensitive data in shared temp directory

Remediation:

- Use `mkdtemp()` for secure directory creation
- Set exclusive file creation flags
- Use Application Support instead of temp
- Implement secure file deletion

VULN-011: Network Security - No Certificate Pinning

Severity: MEDIUM

Location: Application-wide

Description:

The application relies on rclone for network communication without additional certificate pinning or TLS verification.

Impact:

- Vulnerable to MITM attacks with compromised CA
- No protection against SSL stripping
- Trust in third-party (rclone) for TLS handling

Remediation:

- Implement certificate pinning for critical endpoints
- Verify rclone uses TLS 1.2+ only
- Add network security monitoring
- Consider App Transport Security (ATS) configuration

VULN-012: Crash Reports May Contain Sensitive Data

Severity: MEDIUM

Location:
`/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/CrashReportingManager.swift`

Description:

Crash reports capture call stacks which may contain sensitive data in function arguments.

Vulnerable Code:

```
let crashLog = """ === CRASH REPORT === Date: \(Date()) Exception: \(exception.name.rawValue)
Reason: \(exception.reason ?? "Unknown") Call Stack:
\(exception.callStackSymbols.joined(separator: "\n")) """
```

Impact:

- Passwords/tokens may appear in stack traces
- File paths expose user directory structure
- Sensitive usernames in error messages

Remediation:

- Scrub sensitive patterns from crash logs
- Implement log redaction for known sensitive keys
- Warn users before log export
- Add user consent for crash reporting

Low Severity Findings

VULN-013: Hardcoded rclone Path

Severity: LOW

Location: `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/RcloneManager.swift` (Line 38)

Description:

Fallback rclone path is hardcoded to Homebrew location.

Code:

```
self.rclonePath = "/opt/homebrew/bin/rclone"
```

Impact:

- Fails on Intel Macs (uses `/usr/local/bin`)
- May execute unintended binary if present at that path

Remediation:

- Use `which rclone` or search PATH
- Verify binary signature before use
- Document system requirements

VULN-014: No Rate Limiting on Auth Attempts

Severity: LOW

Location: Application-wide authentication flows

Description:

No client-side rate limiting on authentication attempts to cloud providers.

Impact:

- Enables brute force attacks
- May trigger account lockouts
- API rate limit exhaustion

Remediation:

- Implement exponential backoff
- Add attempt counters per provider
- Display warnings after failed attempts

VULN-015: Missing Secure Memory Handling

Severity: LOW

Location: Throughout credential handling code

Description:

Credentials stored in standard Swift Strings which may persist in memory and be swapped to disk.

Impact:

- Credentials recoverable from memory dumps
- Swap file may contain sensitive data
- No secure deallocation

Remediation:

- Use `Data` with `resetBytes(in:)` for sensitive data
- Implement SecureString wrapper class
- Lock memory pages containing credentials

VULN-016: Privacy - Account Names in UserDefaults

Severity: LOW

Location:
`/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/Models/CloudProvider.swift` (Line 403)

Description:

Account names (often email addresses) stored in UserDefaults as part of CloudRemote model.

Code:

```
var accountName: String? // Email/username for the connected account
```

Impact:

- User email addresses in plaintext plist
- Privacy concern for shared computers
- PII exposure

Remediation:

- Store account identifiers only, not full emails
- Hash email addresses for display
- Option to hide account names

Security Best Practices Checklist

Credential Storage

- [ ] **CRITICAL:** Move all encryption passwords from UserDefaults to Keychain
- [ ] **HIGH:** Use stdin for passing secrets to child processes
- [ ] **MEDIUM:** Implement secure memory handling for credentials
- [ ] Set Keychain accessibility to `WhenUnlockedThisDeviceOnly` for high-value secrets
- [ ] Add token expiry tracking and refresh

macOS Security

- [ ] **CRITICAL:** Enable App Sandbox with minimum required entitlements
- [ ] **HIGH:** Verify code signatures of bundled binaries
- [ ] Enable Hardened Runtime
- [ ] Implement Library Validation
- [ ] Use Security-Scoped Bookmarks for file access

Input Validation

- [ ] **HIGH:** Sanitize all file paths before use in Process arguments
- [ ] **MEDIUM:** Add length limits to all text inputs
- [ ] Implement format validation for email fields
- [ ] Sanitize remote names and paths

Network Security

- [ ] Verify TLS configuration
- [ ] Consider certificate pinning for auth endpoints
- [ ] Implement connection timeout handling
- [ ] Add network error logging (sanitized)

Logging & Debugging

- [ ] **MEDIUM:** Remove debug logging to /tmp in production
- [ ] Use Application Support for all logs
- [ ] Implement log redaction for sensitive data
- [ ] Add secure log rotation

Data Protection

- [ ] Set restrictive permissions (0o600) on config files
- [ ] Implement secure file deletion
- [ ] Add integrity checking for configuration

- [ ] Encrypt sensitive data at rest

Remediation Priority Matrix

Priority	Finding	Effort	Business Impact
P0	VULN-001 (Encryption in UserDefaults)	Medium	Critical
P0	VULN-002 (Sandbox Disabled)	High	Critical
P1	VULN-003 (Command Injection)	Medium	High
P1	VULN-004 (Password in CLI)	Low	High
P1	VULN-005 (Binary Verification)	Medium	High
P1	VULN-006 (Token Protection)	Low	High
P2	VULN-007 (Debug Logging)	Low	Medium
P2	VULN-008 (Config Permissions)	Low	Medium
P2	VULN-009 (Input Validation)	Medium	Medium
P2	VULN-010 (Temp Files)	Low	Medium
P2	VULN-011 (Cert Pinning)	High	Medium
P2	VULN-012 (Crash Data)	Medium	Medium
P3	VULN-013-016	Low	Low

Recommendations

Immediate Actions (Week 1)

- Migrate encryption passwords to Keychain (VULN-001)
- Enable App Sandbox with required entitlements (VULN-002)
- Implement path validation for Process arguments (VULN-003)

Short-term (Month 1)

- Use stdin for password obscuring (VULN-004)

- Add binary signature verification (VULN-005)
- Remove debug logging from production (VULN-007)
- Fix config file permissions (VULN-008)

Medium-term (Quarter 1)

- Implement comprehensive input validation (VULN-009)
- Secure temporary file handling (VULN-010)
- Evaluate certificate pinning (VULN-011)
- Add crash log redaction (VULN-012)

Audit Methodology

This audit included:

- Static code analysis of Swift source files
- Review of entitlements and Info.plist configurations
- Analysis of credential handling flows
- Evaluation of external process invocations
- Review of data persistence mechanisms
- Assessment against macOS security best practices

Files Reviewed

- `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/RcloneManager.swift` (2000+ lines)
- `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/KeychainManager.swift` (348 lines)
- `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/EncryptionManager.swift` (374 lines)
- `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/SyncManager.swift` (295 lines)
- `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/ProtonDriveManager.swift` (376 lines)
- `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/ScheduleManager.swift` (305 lines)
- `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/Models/*.swift` (5 files)
- `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/Views/ProtonDriveSetupView.swift`
- `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/CloudSyncApp.entitlements`
- `/sessions/wonderful-fervent-noether/mnt/Claude/CloudSyncApp/Info.plist`

Conclusion

CloudSync Ultra demonstrates solid architecture for cloud synchronization but has significant security gaps in credential storage and macOS security configurations. The two critical findings (encryption passwords in UserDefaults and disabled sandbox) require immediate remediation before any production deployment.

The application makes appropriate use of macOS Keychain for some credentials (Proton Drive), showing awareness of security best practices. However, this security-conscious approach needs to be applied consistently across all credential types, especially encryption master passwords.

Addressing the identified vulnerabilities will significantly improve the application's security posture and align it with macOS security best practices and App Store requirements.

Report generated by Security-Auditor AI Agent

Model: Claude Opus 4.5