HMAC

```
#!/usr/bin/env python3
"""
Simple HMAC Verification Script

This script verifies HMAC signatures for message files by:
1. Finding all message_#.txt and message_#.hmac file pairs
2. Checking each line to verify the HMAC integrity
3. Reporting only basic verification results without additional analysis

Usage:
    python simple_hmac_verify.py --directory <logs_directory> --key <hmac_key>
"""

import hmac
import hashlib
import os
import sys
import re
import glob
import argparse
from datetime import datetime

# The valid signing key
VALID_KEY = 'ciCloud-API-20240315-4f7b9c'

def calculate_hmac(message, key):
    """Calculate HMAC signature for a message."""
    key_bytes = key.encode('utf-8')
    message_bytes = message.encode('utf-8')
    signature = hmac.new(key_bytes, message_bytes, hashlib.sha256)
    return signature.hexdigest()

def verify_hmac(message, signature, key):
    """Verify if a message's HMAC signature is valid."""
    calculated_signature = calculate_hmac(message, key)
    return hmac.compare_digest(calculated_signature, signature)
```

```
def read_file(file_path):
    """Read a file and return its lines."""
    with open(file_path, 'r') as f:
        return [line.rstrip() for line in f.readlines()]

def find_file_pairs(directory):
    """Find matching message/HMAC file pairs in the directory."""
    file_pairs = []

    # Find all message_*.txt files
    message_files = glob.glob(os.path.join(directory, "message_*.txt"))

    for message_file in message_files:
        # Extract the number part
        match = re.search(r'message_(\d+)\.txt$', message_file)
        if match:
            number = match.group(1)
            hmac_file = os.path.join(directory, f"message_{number}.hmac")

            # Check if the corresponding HMAC file exists
            if os.path.exists(hmac_file):
                file_pairs.append((message_file, hmac_file))

    return file_pairs

def process_file_pair(message_file, hmac_file, key):
    """Process a single message/HMAC file pair."""
    # Extract file number for identification
    match = re.search(r'message_(\d+)\.txt$', message_file)
    file_id = match.group(1) if match else os.path.basename(message_file)

    try:
        # Read files
        message_lines = read_file(message_file)
        hmac_lines = read_file(hmac_file)

        total_lines = min(len(message_lines), len(hmac_lines))
        valid_lines = 0
        invalid_lines = 0
        mismatched_entries = []
```

```
        print(f"Processing file {file_id}: {os.path.basename(message_file)}")
        print(f"  - Total lines: {total_lines}")

        # Process each line
        for i in range(total_lines):
            message = message_lines[i]
            signature = hmac_lines[i]

            # Skip empty lines
            if not message or not signature:
                continue

            # Debug: Print first few characters of message and signature
            if i < 3:  # Just print a few examples for debugging
                print(f"  - Line {i+1} check:")
                print(f"    Message: {message[:30]}{'...' if len(message) > 30 else ''}")
                print(f"    Signature: {signature[:30]}{'...' if len(signature) > 30 else
''}")
                print(f"    Calculated: {calculate_hmac(message, key)[:30]}...")

            # Verify HMAC
            is_valid = verify_hmac(message, signature, key)

            if is_valid:
                valid_lines += 1
            else:
                invalid_lines += 1
                mismatched_entries.append({
                    'line': i + 1,
                    'message': message,
                    'provided_signature': signature,
                    'calculated_signature': calculate_hmac(message, key)
                })

        result = {
            'file_id': file_id,
            'message_file': message_file,
            'hmac_file': hmac_file,
            'total_lines': total_lines,
            'valid_lines': valid_lines,
```

```
            'invalid_lines': invalid_lines,
            'mismatched_entries': mismatched_entries[:10]  # Only include first 10 for brevity
        }

        print(f"  - Valid lines: {valid_lines}")
        print(f"  - Invalid lines: {invalid_lines}")
        print(f"  - Integrity: {'INTACT' if invalid_lines == 0 else 'COMPROMISED'}")
        print()

        return result

    except Exception as e:
        print(f"Error processing file pair ({message_file}, {hmac_file}): {e}")
        return {
            'file_id': file_id,
            'message_file': message_file,
            'hmac_file': hmac_file,
            'error': str(e)
        }

def main():
    """Main entry point for the script."""
    parser = argparse.ArgumentParser(description='Simple HMAC Verification')
    parser.add_argument('--directory', '-d', required=True, help='Directory containing log
files')
    parser.add_argument('--key', '-k', default=VALID_KEY, help=f'HMAC signing key (default:
{VALID_KEY})')
    parser.add_argument('--verbose', '-v', action='store_true', help='Enable verbose output')

    args = parser.parse_args()

    try:
        start_time = datetime.now()
        print(f"Starting HMAC verification in {args.directory}")
        print(f"Using key: {args.key}")
        print(f"Started at: {start_time.isoformat()}")
        print("-" * 60)

        # Find all file pairs
        file_pairs = find_file_pairs(args.directory)
```

```
if not file_pairs:
    print(f"No matching message/HMAC file pairs found in {args.directory}")
    sys.exit(1)

print(f"Found {len(file_pairs)} file pairs")
print("-" * 60)

# Process each file pair
results = []
total_files = len(file_pairs)
files_with_errors = 0
files_with_mismatches = 0
total_lines_processed = 0
total_mismatched_lines = 0

for message_file, hmac_file in file_pairs:
    result = process_file_pair(message_file, hmac_file, args.key)
    results.append(result)

    if 'error' in result:
        files_with_errors += 1
    else:
        total_lines_processed += result['total_lines']
        total_mismatched_lines += result['invalid_lines']

        if result['invalid_lines'] > 0:
            files_with_mismatches += 1

# Summary
print("-" * 60)
print("VERIFICATION SUMMARY")
print("-" * 60)
print(f"Total file pairs processed: {total_files}")
print(f"Files with errors: {files_with_errors}")
print(f"Files with mismatched HMACs: {files_with_mismatches}")
print(f"Total lines processed: {total_lines_processed}")
print(f"Total mismatched lines: {total_mismatched_lines}")

end_time = datetime.now()
duration = end_time - start_time
print(f"Duration: {duration.total_seconds():.2f} seconds")
```

```
        # List files with mismatches
        if files_with_mismatches > 0:
            print("\nFiles with mismatched HMACs:")
            for result in results:
                if 'invalid_lines' in result and result['invalid_lines'] > 0:
                    print(f"- {os.path.basename(result['message_file'])}:
{result['invalid_lines']} mismatched lines")

                    # Show example of first mismatched entry
                    if args.verbose and result['mismatched_entries']:
                        first_mismatch = result['mismatched_entries'][0]
                        print(f"  Example (line {first_mismatch['line']}):")
                        print(f"  Message: {first_mismatch['message'][:50]}...")
                        print(f"  Provided HMAC: {first_mismatch['provided_signature']}")
                        print(f"  Calculated HMAC: {first_mismatch['calculated_signature']}")
                        print()

    except Exception as e:
        print(f"Error: {e}")
        import traceback
        traceback.print_exc()
        sys.exit(1)

if __name__ == "__main__":
    main()
```

Revision #1

Created 2025-11-25 18:04:56 UTC by David Rizzo

Updated 2025-11-25 18:05:04 UTC by David Rizzo