VEALC Server
Complete Project Documentation

Generated from Source Code

Version 1.0

December 19, 2025

Contents

Introduction

VEALC (Vector Encryption Authentication and Logic Calculator) Server is a TCP server application written in C++ that provides:

- Client authentication using MD5 hashing with salt
- Vector data processing with overflow control
- Concurrent client session management
- Comprehensive logging system
- Configurable server settings

Chapter 1

Project Architecture

1.1 System Overview

The server follows a modular architecture with clear separation of concerns:

```
        +---------------+
        |    Client     |
        +-------+-------+
                |
        +-------+-------+
        |    Session    |
        +-------+-------+
                |
    +-----------+-----------+
    |                       |
+---+-------+         +-----+-----+
|  Auth     |         |  Vector   |
|  Module   |         |  Processor|
+-----------+         +-----------+
    |                       |
+---+-------+         +-----+-----+
|  Logger   |         |   Config  |
+-----------+         +-----------+
```

Figure 1.1: VEALC Server Architecture

1.2 Component Responsibilities

Component	**Responsibility**
`Server`	Main server class, manages connections and sessions
`Session`	Handles individual client communication
`Authenticator`	Client authentication using MD5
`VectorProcessor`	Mathematical vector calculations
`Logger`	Logging system for events and errors
`Config`	Server configuration management

Chapter 2

Class Documentation

2.1 Class: Server

2.1.1 Class Definition

```cpp
class Server {
private:
    ServerConfig config_;
    Logger logger_;
    std::unordered_map<std::string, std::string> clients_;
    int server_fd_;

    void load_clients();
    void setup_socket();
    void accept_connections();

public:
    Server(const ServerConfig& config);
    ~Server();
    void run();
};
```

Listing 2.1: Server Class Header

2.1.2 Public Methods

Method	Description
`Server(const ServerConfig& config)`	Constructor with configuration
`Šerver()`	Destructor, closes socket
`void run()`	Starts the main server loop

2.1.3 Private Methods

Method	Description
`void load_clients()`	Loads client database from file
`void setup_socket()`	Configures server socket
`void accept_connections()`	Accepts incoming client connections

2.2 Class: Session

2.2.1 Class Definition

```cpp
class Session {
private:
    int client_socket;
    std::unordered_map<std::string, std::string>& clients;
    Logger& logger;
    std::string receive_buffer;

    void receive_to_buffer();
    std::string extract_from_buffer_until_non_hex();
    std::string extract_from_buffer_exact(size_t length);
    bool verify_authentication(const std::string& login,
                               const std::string& salt,
                               const std::string& received_hash);
    void process_vectors();
    uint32_t receive_uint32();
    std::vector<int32_t> receive_vector(uint32_t size);
    void send_uint32(uint32_t value);
    void send_int32(int32_t value);
    std::string calculate_md5(const std::string& data);
    int32_t calculate_vector_product(const std::vector<int32_t>& vector);

public:
    Session(int client_socket,
            std::unordered_map<std::string, std::string>& clients,
            Logger& logger);
    void handle();
    bool send_text(const std::string& text);
};
```

Listing 2.2: Session Class Header

2.2.2 Key Methods

- `handle()` - Main session handler
- `verify_authentication()` - Client authentication
- `calculate_vector_product()` - Vector multiplication with overflow control

2.3 Class: Authenticator

2.3.1 Class Definition

```cpp
class Authenticator {
public:
    static std::string generate_salt_16();
    static std::string calculate_md5_hash(const std::string& salt,
                                          const std::string& password);
    static bool verify_client(const std::string& login,
                              const std::string& received_hash,
                              const std::string& salt,
                              const std::unordered_map<std::string, std::string
                                  >& clients);
};
```

Listing 2.3: Authenticator Class Header

2.3.2 Static Methods

- `generate_salt_16()` - Generates 16-byte random salt
- `calculate_md5_hash()` - Computes MD5(salt + password)
- `verify_client()` - Verifies client credentials

2.4 Class: VectorProcessor

2.4.1 Class Definition

```cpp
class VectorProcessor {
public:
    static int32_t calculate_product(const Vector& vector);
    static std::vector<int32_t> multiply_vectors(const std::vector<Vector>&
        vectors);
};
```

Listing 2.4: VectorProcessor Class Header

2.4.2 Algorithm Implementation

```cpp
int32_t VectorProcessor::calculate_product(const Vector& vector) {
    if (vector.empty()) return 0;

    int64_t product = 1;
    for (int32_t val : vector) {
        int64_t val64 = static_cast<int64_t>(val);

        // Overflow check
        if (val64 != 0 && llabs(product) > INT64_MAX / llabs(val64)) {
            return (product > 0 && val64 > 0) ? INT32_MAX : INT32_MIN;
        }
        product *= val64;
    }

    // Clamp to int32 range
    if (product > INT32_MAX) return INT32_MAX;
    if (product < INT32_MIN) return INT32_MIN;
    return static_cast<int32_t>(product);
}
```

Listing 2.5: Vector Product Calculation with Overflow Control

2.5 Class: Logger

2.5.1 Class Definition

```cpp
class Logger {
private:
    std::string log_file_;
    std::string get_current_time();

public:
    Logger(const std::string& filename);
    void log(const std::string& message, bool critical = false);
    void log_add(const std::string& message);
    void log_error(const std::string& error, bool critical = false);
};
```

Listing 2.6: Logger Class Header

2.5.2 Log Format

```
[2024-01-15 10:30:00] [CRITICAL] err: Error message
[2024-01-15 10:30:05] [NON-CRITICAL] Info message
```

2.6 Struct: ServerConfig

2.6.1 Structure Definition

```cpp
struct ServerConfig {
    std::string client_db_file = "/etc/vealc.conf";
    std::string log_file = "/var/log/vealc.log";
    int port = 33333;

    static ServerConfig parse_args(int argc, char* argv[]);
    static void print_help();
};
```

Listing 2.7: ServerConfig Structure

Chapter 3

File Documentation

3.1 Header Files (.h)

3.1.1 auth.h

- **Purpose:** Client authentication declarations
- **Contains:** Authenticator class
- **Dependencies:** `<string>`, `<unordered_map>`

3.1.2 config.h

- **Purpose:** Server configuration
- **Contains:** ServerConfig structure
- **Dependencies:** `<string>`

3.1.3 logger.h

- **Purpose:** Logging system interface
- **Contains:** Logger class
- **Dependencies:** `<string>`

3.1.4 session.h

- **Purpose:** Client session management
- **Contains:** Session class
- **Dependencies:** `<string>`, `<vector>`, `<unordered_map>`

3.1.5 server.h

- **Purpose:** Main server interface
- **Contains:** Server class
- **Dependencies:** `"config.h"`, `"logger.h"`, `<unordered_map>`

3.1.6 types.h

- **Purpose:** Common type definitions
- **Contains:** Vector, ByteArray type aliases
- **Dependencies:** `<cstdint>`, `<vector>`

3.1.7 vector_processor.h

- **Purpose:** Vector calculations interface
- **Contains:** VectorProcessor class
- **Dependencies:** `"types.h"`, `<cstdint>`, `<vector>`

3.2 Source Files (.cpp)

3.2.1 auth.cpp

- **Purpose:** Authentication implementation
- **Implements:** Authenticator methods
- **Dependencies:** `"auth.h"`, `<openssl/md5.h>`, `<random>`

3.2.2 config.cpp

- **Purpose:** Configuration parsing
- **Implements:** ServerConfig methods
- **Dependencies:** `"config.h"`, `<iostream>`, `<cstring>`

3.2.3 logger.cpp

- **Purpose:** Logging implementation
- **Implements:** Logger methods
- **Dependencies:** `"logger.h"`, `<fstream>`, `<ctime>`

3.2.4 main.cpp

- **Purpose:** Program entry point
- **Contains:** main() function
- **Dependencies:** `"server.h"`, `"config.h"`

3.2.5 session.cpp

- **Purpose:** Session handling implementation
- **Implements:** Session methods
- **Dependencies:** `"session.h"`, `<openssl/md5.h>`, `<sys/socket.h>`

3.2.6 server.cpp

- **Purpose:** Server implementation
- **Implements:** Server methods
- **Dependencies:** `"server.h"`, `"session.h"`, `<sys/socket.h>`

3.2.7 vector_processor.cpp

- **Purpose:** Vector calculations implementation
- **Implements:** VectorProcessor methods
- **Dependencies:** `"vector_processor.h"`, `<climits>`

Chapter 4

Communication Protocol

4.1 Authentication Phase

```
Client → Server: login[16_hex_salt][32_hex_hash]
Server → Client: "OK\n" or "err\n"
```

4.2 Data Processing Phase

```
Client → Server: vector_count (uint32, 4 bytes)
For each vector (vector_count times):
    Client → Server: vector_size (uint32, 4 bytes)
    Client → Server: vector_data (int32[vector_size], 4*size bytes)
    Server → Client: product (int32, 4 bytes)
```

4.3 Example Session

```
// Authentication
Client sends: "alice4F3A1B8C9D2E7F5A32B4C6D8E0F1A2B4"
Server sends: "OK\n"

// Vector processing
Client sends: 0x00000002 (2 vectors)

// First vector
Client sends: 0x00000003 (3 elements)
Client sends: 0x00000002 0x00000003 0x00000004 (2, 3, 4)
Server sends: 0x00000018 (24)

// Second vector
Client sends: 0x00000002 (2 elements)
Client sends: 0x7FFFFFFF 0x00000002 (INT32_MAX, 2)
Server sends: 0x7FFFFFFF (INT32_MAX, overflow detected)
```

Listing 4.1: Example Protocol Exchange

Chapter 5

Configuration

5.1 Client Database Format

Client credentials are stored in `/etc/vealc.conf`:

```
username:password
alice:P@ssw0rd1
bob:Secret123
charlie:Qwerty!@#
```

5.2 Command Line Arguments

Option	Default	Description
`-p PORT`	33333	Server port number
`-c FILE`	`/etc/vealc.conf`	Client database file
`-d FILE`	(same as `-c`)	Alias for `-c`
`-l FILE`	`/var/log/vealc.log`	Log file location
`-h, --help`	-	Show help message

5.3 Example Usage

```
# Default configuration
./server

# Custom port and config
./server -p 44444 -c ./myclients.conf -l ./server.log

# Show help
./server --help
```

Chapter 6

Build and Deployment

6.1 Compilation

```
g++ -std=c++11 -o server \
    main.cpp server.cpp session.cpp \
    auth.cpp config.cpp logger.cpp \
    vector_processor.cpp \
    -lssl -lcrypto
```

Listing 6.1: Compilation Command

6.2 Dependencies

- **Compiler:** g++ with C++11 support
- **Libraries:** OpenSSL (libssl-dev)
- **System:** Linux with POSIX sockets

6.3 Installation

1. Install dependencies: `sudo apt-get install g++ libssl-dev`
2. Compile the server: `make` or `g++ ...`
3. Create configuration: `sudo cp vealc.conf /etc/`
4. Set up log file: `sudo touch /var/log/vealc.log`
5. Run server: `./server`

Chapter 7

Security Considerations

7.1 Current Implementation

- **Authentication:** MD5(salt + password)
- **Password Storage:** Plain text in file
- **Network:** Unencrypted TCP
- **Salt:** 16-byte random hex string

7.2 Security Limitations

Warning: This implementation is for educational purposes only. Production use requires:

- Replace MD5 with SHA-256 or bcrypt
- Encrypt network traffic with TLS/SSL
- Hash passwords before storage
- Implement rate limiting
- Add input validation and sanitization
- Use secure random number generation

Chapter 8

Troubleshooting

8.1 Common Issues

Issue	**Cause**	**Solution**
`Bind failed`	Port already in use	Use different port or kill existing process
`Cannot open client database`	File doesn't exist or no permissions	Check file path and permissions
`Authentication failed`	Wrong password or hash calculation	Verify password in config file
`Connection refused`	Server not running or firewall	Start server and check firewall

8.2 Debugging

- Check logs: `tail -f /var/log/vealc.log`
- Verify configuration: `cat /etc/vealc.conf`
- Test connectivity: `netstat -tlnp | grep 33333`
- Monitor processes: `ps aux | grep server`

Appendix A

Type Definitions

A.1 Vector Type

```cpp
using Vector = std::vector<int32_t>;
```

A.2 ByteArray Type

```cpp
using ByteArray = std::vector<uint8_t>;
```

Appendix B

Complete File Listing

B.1 Project Structure

```
vealc-server/
 include/
    auth.h
    config.h
    logger.h
    server.h
    session.h
    types.h
    vector_processor.h
 src/
    auth.cpp
    config.cpp
    logger.cpp
    main.cpp
    server.cpp
    session.cpp
    vector_processor.cpp
 Doxyfile
 Makefile
 vealc.conf
 README.md
```

Appendix C

Index

C.1 Classes

- Server - Main server class
- Session - Client session handler
- Authenticator - Authentication module
- VectorProcessor - Vector calculations
- Logger - Logging system
- ServerConfig - Configuration structure

C.2 Files

- `auth.h/cpp` - Authentication
- `config.h/cpp` - Configuration
- `logger.h/cpp` - Logging
- `main.cpp` - Entry point
- `session.h/cpp` - Session handling
- `server.h/cpp` - Server implementation
- `types.h` - Common types
- `vector_processor.h/cpp` - Vector processing

Conclusion

VEALC Server provides a complete solution for client authentication and vector data processing. The modular architecture allows for easy maintenance and extension. While suitable for educational purposes, production deployment would require additional security measures.