🚀 Quick Reference Guide - CyberSec Multitool

⚡ Common Commands

Compilation

```bash
# Windows (MSVC) - Production
cl /EHsc /std:c++17 /O2 /GS /DYNAMICBASE /NXCOMPAT multitool_pro.cpp

# Windows (MinGW) - Production
g++ -std=c++17 -O2 -fstack-protector-strong -fPIE -pie multitool_pro.cpp ^
    -lws2_32 -lurlmon -lshlwapi -lpdh -lpsapi -o multitool.exe

# Debug Build with Sanitizers
g++ -std=c++17 -g -fsanitize=address,leak multitool_pro.cpp -o debug.exe
```

🎯 Feature Quick Access

Network Security

Feature	Path	Key Functionality
Port Scanner	Menu 3 → 1	Scans 20 common ports with banner grabbing
Ping	Menu 3 → 2	ICMP ping to check host availability
Download	Menu 3 → 3	Download files from URLs

Forensics

Feature	Path	Key Functionality
Process Inspector	Menu 6 → 1	View memory usage + loaded DLLs
File Hash	Menu 5 → 1	Calculate MD5 and SHA256 hashes
Secure Delete	Menu 4 → 3	DoD 5220.22-M compliant deletion

Cryptographic

Feature	Path	Key Functionality
Base64	Menu 5 → 2	Encode/decode Base64 strings
Password Gen	Menu 5 → 3	Generate secure random passwords
File Hash	Menu 5 → 1	MD5 + SHA256 calculation

🔧 Key Classes Reference

ThreadPool

cpp

```cpp
// Create pool with 10 workers
ThreadPool pool(10);

// Enqueue task
auto future = pool.enqueue([]() {
    return scanPort(80);
});

// Get result
auto result = future.get();
```

SecureLogger

cpp

```cpp
// Global instance
g_logger->info("Normal operation");
g_logger->warning("Suspicious activity");
g_logger->error("Operation failed");
g_logger->critical("System compromised!");
```

NetworkScanner

cpp

```cpp
auto scanner = std::make_unique<NetworkScanner>();
std::vector<int> ports = {21, 22, 80, 443};
scanner->scanPorts("192.168.1.1", ports);
```

CryptoUtils

```cpp
// File hashing
std::string md5 = CryptoUtils::calculateMD5("file.exe");
std::string sha256 = CryptoUtils::calculateSHA256("file.exe");

// Base64
std::string encoded = CryptoUtils::base64Encode("Hello");
std::string decoded = CryptoUtils::base64Decode("SGVsbG8=");
```

ProcessInspector

```cpp
auto processes = ProcessInspector::getDetailedProcessList();
for(const auto& proc : processes) {
    std::cout << "PID: " << proc.pid << "\n";
    std::cout << "Memory: " << proc.workingSetSize / 1024 / 1024 << " MB\n";
    for(const auto& dll : proc.dlls) {
        std::cout << "  - " << dll << "\n";
    }
}
```

SecureFileOps

```cpp
// Secure deletion (3 passes)
SecureFileOps::secureDelete("sensitive.txt", 3);
```

SystemMonitor

```cpp
```

```
SystemMonitor monitor;
double cpu = monitor.getCpuUsage();     // Returns 0-100
double ram = monitor.getRamUsage();     // Returns 0-100
monitor.displayLiveStats();             // Shows in UI
```

🎨 ANSI Color Codes

cpp

```
// Available color constants
Color::NEON_GREEN   // Primary highlights
Color::PURPLE       // Headers and titles
Color::CYAN         // Secondary info
Color::RED          // Errors and warnings
Color::YELLOW       // User prompts
Color::DARK_GRAY    // Borders and subtle text
Color::RESET        // Reset to default
```

Usage:

cpp

```
std::cout << Color::NEON_GREEN << "[+] Success!" << Color::RESET << "\n";
std::cout << Color::RED << "[-] Failed!" << Color::RESET << "\n";
```

🔐 Security Flags Cheatsheet

MSVC

```
/GS             - Buffer security check (stack canaries)
/DYNAMICBASE    - ASLR (randomize memory addresses)
/NXCOMPAT       - DEP (non-executable stack)
/guard:cf       - Control Flow Guard (anti-ROP)
/sdl            - Security Development Lifecycle checks
/SAFESEH        - Safe exception handlers
```

GCC/MinGW

```
-fstack-protector-strong  - Stack canaries
-D_FORTIFY_SOURCE=2       - Runtime overflow detection
-fPIE -pie                - Position independent (ASLR)
-Wl,-z,relro,-z,now       - Read-only relocations
-Wl,-z,noexecstack        - Non-executable stack
-fcf-protection=full      - Control Flow Integrity
```

📊 Performance Benchmarks

Operation	Sequential	Multi-threaded	Speedup
Scan 20 ports	20 sec	2 sec	**10x**
Hash 100MB file	3 sec	3 sec	1x (I/O bound)
List processes	0.5 sec	0.5 sec	1x (API bound)
Base64 1MB	50ms	50ms	1x (CPU bound, single task)

🐛 Debugging Tips

Enable Verbose Logging

cpp

```cpp
// In main():
g_logger = std::make_unique<SecureLogger>("verbose.log");
g_logger->info("Starting...");
```

View Encrypted Logs

cpp

```
// Decrypt script:
std::ifstream log("debug.log", std::ios::binary);
std::string data((std::istreambuf_iterator<char>(log)), {});
for(char& c : data) c ^= 0x5A;  // XOR key
std::cout << data;
```

Memory Leak Detection

```
bash

# Compile with ASan
g++ -fsanitize=address,leak -g multitool_pro.cpp

# Run
./multitool_pro.exe

# Check output for leaks
# Should show: "ERROR: LeakSanitizer: detected memory leaks" if any
```

Common Error Messages

Error	Cause	Solution
`unresolved external symbol WSAStartup`	Missing library	Add `-lws2_32`
`Failed to create process snapshot`	No admin rights	Run as Administrator
`ANSI colors not showing`	Legacy console	Use Windows Terminal
`Port scan timeout`	Firewall blocking	Disable firewall for test

📁 File Structure

```
multitool_pro/
├── multitool_pro.cpp       # Main source file (3500+ lines)
├── debug.log               # Encrypted runtime log
├── quick_notes.txt         # User notes (if using feature)
├── output.txt              # Sample file creation
└── CMakeLists.txt          # Optional build system
```

🚨 Security Warnings

NEVER

- ❌ Run untrusted code from the network
- ❌ Scan networks you don't own
- ❌ Store credentials in plaintext
- ❌ Disable security features for convenience
- ❌ Run without antivirus exclusions (adds tool to quarantine)

ALWAYS

- ✅ Test in virtual machines first
- ✅ Keep tools updated
- ✅ Use encrypted channels for sensitive data
- ✅ Verify file hashes before execution
- ✅ Run with least privilege (when possible)

🎯 Common Use Cases

1. Internal Network Audit

```
1. Menu 3 → 1 (Port Scanner)
2. Target: Your internal network range
3. Review open ports for unauthorized services
```

2. Malware Analysis Preparation

```
1. Menu 5 → 1 (File Hash)
2. Calculate hashes before analysis
3. Menu 6 → 1 (Process Inspector)
4. Monitor memory during execution
```

3. Forensic File Recovery Prevention

```
1. Menu 4 → 3 (Secure Delete)
2. Enter sensitive file path
3. Confirm 3-pass overwrite
```

4. Deobfuscate PowerShell

```
1. Menu 5 → 2 (Base64 Decode)
2. Paste encoded command
3. Analyze decoded command
```

5. System Health Check

```
1. Main Menu (view live CPU/RAM)
2. Menu 6 → 1 (Process Inspector)
3. Look for high memory usage
```

🔍 Keyboard Shortcuts

Key	Action
`1-7`	Navigate main menu
`Enter`	Confirm selection
`Ctrl+C`	Interrupt operation
Any key	Continue after pause

📖 API Reference

Windows API Used

```cpp
```

```
// Network
WSAStartup()          - Initialize Winsock
socket()              - Create socket
connect()             - Connect to host
recv()                - Receive data

// Process
CreateToolhelp32Snapshot()  - Snapshot processes
Process32First/Next()       - Enumerate processes
GetProcessMemoryInfo()      - Get memory usage
Module32First/Next()        - Enumerate DLLs

// Performance
PdhOpenQuery()              - Open PDH query
PdhAddEnglishCounter()      - Add CPU counter
PdhCollectQueryData()       - Collect data

// File
CreateFile()                - Open file
DeleteFile()                - Delete file
GlobalMemoryStatusEx()      - Get RAM info

// Network Download
URLDownloadToFile()         - Download from URL
```

🎓 Learning Path

Beginner

1. Start with `basicCalculator()` - understand UI flow
2. Examine `SecureLogger` - learn RAII pattern
3. Study `CryptoUtils` - see static methods

Intermediate

4. Analyze `ThreadPool` - understand concurrency
5. Review `NetworkScanner` - async operations
6. Study `ProcessInspector` - Windows API usage

Advanced

7. Implement new hash algorithm (SHA-512)
8. Add new scanner (UDP ports)
9. Integrate Volatility framework

💡 Pro Tips

Optimization

cpp

```cpp
// Reserve capacity for known sizes
std::vector<int> ports;
ports.reserve(100);  // Prevent reallocations

// Use move semantics
return std::move(largeObject);  // Transfer ownership

// Avoid copying
void process(const std::string& data);  // Pass by const ref
```

Error Handling

cpp

```cpp
try {
    riskyOperation();
} catch(const NetworkException& e) {
    g_logger->error(e.what());
    // Graceful fallback
} catch(const std::exception& e) {
    g_logger->critical(e.what());
    // General error handling
}
```

Thread Safety

cpp

```
std::mutex mtx;
void threadSafeFunction() {
    std::lock_guard<std::mutex> lock(mtx);
    // Critical section protected
}
```

🆘 Troubleshooting FAQ

Q: Port scanner shows no results? A: Check firewall settings, ensure target is reachable with `ping`

Q: Process inspector shows "Access Denied"? A: Run as Administrator (right-click → Run as administrator)

Q: Colors not displaying? A: Use Windows Terminal instead of cmd.exe

Q: Secure delete takes too long? A: Normal for large files (overwrites 3 times). Use fewer passes if needed.

Q: Memory usage high? A: Expected during port scans (thread pool). Monitor drops after completion.

Q: Antivirus flagged the tool? A: Common for security tools. Add exclusion or compile yourself.

📚 Further Reading

- C++ Core Guidelines
- Winsock Programming
- SANS Reading Room
- OWASP Secure Coding
- Volatility Framework

Version: 2.0 | **Last Updated:** 2025 | **License:** MIT