Trying to bypass mte using threads:

#include <jni.h>

#include <string>

#include <chrono>

#include <vector>

#include <thread>

#include <signal.h>

#include <setjmp.h>

#include <android/log.h>

#include <sys/mman.h>

#include <unistd.h>

#include <sys/auxv.h>

#include <sys/prctl.h>

#include <execinfo.h>

#include <unistd.h>

#include <unwind.h>

#include <dlfcn.h>

#include <android/log.h>

#include <cxxabi.h>

#define TAG\_SHIFT 56

#define NUM\_THREADS 15

static sigjmp\_buf jump\_buffer[NUM\_THREADS];

namespace {

struct BacktraceState {

void\*\* current;

void\*\* end;

};

static \_Unwind\_Reason\_Code UnwindCallback(struct \_Unwind\_Context\* context, void\* arg) {

BacktraceState\* state = static\_cast<BacktraceState\*>(arg);

uintptr\_t pc = \_Unwind\_GetIP(context);

if (pc) {

if (state->current == state->end) {

return *\_URC\_END\_OF\_STACK*;

} else {

\*state->current++ = reinterpret\_cast<void\*>(pc);

}

}

return *\_URC\_NO\_REASON*;

}

} // namespace

size\_t CaptureBacktrace(void\*\* buffer, size\_t max) {

BacktraceState state = {buffer, buffer + max};

\_Unwind\_Backtrace(UnwindCallback, &state);

return state.current - buffer;

}

void DumpBacktrace(void\*\* buffer, size\_t count) {

for (size\_t idx = 0; idx < count; ++idx) {

const void\* addr = buffer[idx];

Dl\_info info;

if (dladdr(addr, &info) && info.dli\_sname) {

int status = 0;

const char\* demangled = \_\_cxxabiv1::\_\_cxa\_demangle(info.dli\_sname, nullptr, nullptr, &status);

const char\* symbol\_name = (status == 0 && demangled != nullptr) ? demangled : info.dli\_sname;

\_\_android\_log\_print(*ANDROID\_LOG\_INFO*, "Backtrace", "#%zu: %p %s", idx, addr, symbol\_name);

if (demangled) free(const\_cast<char\*>(demangled));

}

}

}

void SignalHandler(int signal, siginfo\_t\* info, void\* reserved) {

void\* buffer[64];

size\_t count = CaptureBacktrace(buffer, 64);

\_\_android\_log\_print(*ANDROID\_LOG\_ERROR*, "MTEBenchmarking", "Caught signal %d", signal);

//DumpBacktrace(buffer, count);

// Optionally, use siglongjmp to return to a safe state in your application

// This assumes you've set up sigsetjmp at a safe point

siglongjmp(jump\_buffer[0], 1); // Example for a single thread; adapt as needed

}

void SetupSignalHandlers() {

struct sigaction sa;

sa.sa\_flags = SA\_SIGINFO;

sigemptyset(&sa.sa\_mask);

sa.sa\_sigaction = SignalHandler;

sigaction(SIGILL, &sa, nullptr);

sigaction(SIGSEGV, &sa, nullptr);

}

// Adjust the address with a tag, without setting it in memory

void adjust\_address\_with\_tag(void\* &address, unsigned char tag) {

uintptr\_t addr = reinterpret\_cast<uintptr\_t>(address);

uintptr\_t tagged\_addr = (addr & ~(0xFFUL << TAG\_SHIFT)) | (static\_cast<uintptr\_t>(tag) << TAG\_SHIFT);

address = reinterpret\_cast<void\*>(tagged\_addr);

}

void handle\_sigsegv(int signal) {

//for (int i = 0; i < NUM\_THREADS; ++i) {

siglongjmp(jump\_buffer[1], 1);

//}

}

void accessMemoryWithMTE(unsigned char\* buffer, size\_t pageSize, unsigned int tag, std::string &result) {

\_\_android\_log\_print(*ANDROID\_LOG\_INFO*, "MTEBenchmarking", "Thread %u start: Attempting access with tag %u", tag, tag);

void\* tagged\_ptr = buffer;

adjust\_address\_with\_tag(tagged\_ptr, tag);

if (sigsetjmp(jump\_buffer[tag], 1) == 0) {

auto startTime = std::chrono::high\_resolution\_clock::now();

// if(tag != 0)

static\_cast<unsigned char\*>(tagged\_ptr)[pageSize] = 'a'; // Possible trigger for MTE

auto endTime = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double, std::nano> accessTime = endTime - startTime;

\_\_android\_log\_print(*ANDROID\_LOG\_INFO*, "MTEBenchmarking", "Thread %u access with tag %u successful: %.0f ns", tag, tag, accessTime.count());

} else {

\_\_android\_log\_print(*ANDROID\_LOG\_ERROR*, "MTEBenchmarking", "Thread %u encountered SIGSEGV with tag %u", tag, tag);

}

\_\_android\_log\_print(*ANDROID\_LOG\_INFO*, "MTEBenchmarking", "Thread %u completed", tag);

}

jstring allocateAndMeasureAccessTime(JNIEnv \*env) {

size\_t pageSize = sysconf(\_SC\_PAGESIZE);

unsigned char\* buffer = static\_cast<unsigned char\*>(mmap(NULL, pageSize, PROT\_READ | PROT\_WRITE | PROT\_MTE, MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0));

if (buffer == MAP\_FAILED) {

\_\_android\_log\_print(*ANDROID\_LOG\_ERROR*, "MTEBenchmarking", "Memory allocation with MTE failed");

return env->NewStringUTF("Failed to allocate memory with MTE.");

}

\_\_android\_log\_print(*ANDROID\_LOG\_INFO*, "MTEBenchmarking", "Starting to measure access time with %d threads", NUM\_THREADS);

std::vector<std::thread> threads;

std::vector<std::string> results(NUM\_THREADS);

for (unsigned int tag = 0; tag < NUM\_THREADS; ++tag) {

threads.emplace\_back(accessMemoryWithMTE, buffer, pageSize, tag, std::ref(results[tag]));

\_\_android\_log\_print(*ANDROID\_LOG\_INFO*, "MTEBenchmarking", "Thread %u created for access with tag %u", tag, tag);

}

for (auto &th : threads) {

th.join();

}

\_\_android\_log\_print(*ANDROID\_LOG\_INFO*, "MTEBenchmarking", "All threads joined");

std::string combinedResult;

for (const auto &res : results) {

combinedResult += res + "\n";

}

munmap(buffer, pageSize);

\_\_android\_log\_print(*ANDROID\_LOG\_INFO*, "MTEBenchmarking", "Memory unmapped, returning results");

return env->NewStringUTF(combinedResult.c\_str());

}

extern "C" JNIEXPORT jstring JNICALL Java\_com\_example\_mtestudy\_1benchmarking\_MainActivity\_stringFromJNI(JNIEnv \*env, jobject) {

SetupSignalHandlers();

return allocateAndMeasureAccessTime(env);

}