#include <ntddk.h>

typedef struct {

PDEVICE\_OBJECT LowerKbdDevice;

LARGE\_INTEGER LastKeyTime; // Thời gian nhấn phím cuối cùng

ULONG KeyIntervals[100]; // Lưu 100 khoảng thời gian gần nhất (ms)

ULONG IntervalIndex; // Chỉ số hiện tại trong KeyIntervals

ULONG KeyCount[256]; // Đếm số lần nhấn mỗi mã quét

ULONG SuspiciousCount; // Đếm số lần phát hiện bất thường

ULONG TotalIrpsProcessed; // Tổng số IRP được xử lý (debug)

ULONG KeyPressesInIrp; // Số lần nhấn trong IRP hiện tại (debug)

// Thêm các field mới cho blocking và pattern analysis

ULONG BlockedKeys; // Số phím bị chặn

BOOLEAN SecurityMode; // Bật/tắt chế độ bảo mật

ULONG MaxKeysPerSecond; // Ngưỡng tối đa phím/giây

LARGE\_INTEGER WindowStartTime; // Thời gian bắt đầu cửa sổ đếm

ULONG KeysInWindow; // Số phím trong cửa sổ thời gian

ULONG PatternBuffer[20]; // Buffer lưu pattern 20 phím gần nhất

ULONG PatternIndex; // Chỉ số trong pattern buffer

ULONG ConsecutiveRepeats; // Số lần lặp liên tiếp cùng phím

ULONG LastPatternKey; // Phím cuối cùng trong pattern

ULONG HighFreqKeyCount; // Đếm phím tần suất cao

KSPIN\_LOCK DataLock; // Lock bảo vệ dữ liệu

ULONG AverageInterval; // Khoảng thời gian trung bình

ULONG FastKeysInRow; // Số phím nhanh liên tiếp

BOOLEAN BlockingEnabled; // Có cho phép chặn phím không

} DEVICE\_EXTENSION, \* PDEVICE\_EXTENSION;

typedef struct \_KEYBOARD\_INPUT\_DATA {

USHORT UnitId;

USHORT MakeCode;

USHORT Flags;

USHORT Reserved;

ULONG ExtraInformation;

} KEYBOARD\_INPUT\_DATA, \* PKEYBOARD\_INPUT\_DATA;

// Các hằng số cấu hình

#define MAX\_KEYS\_PER\_SECOND 60 // Ngưỡng tối đa phím/giây

#define WINDOW\_SIZE\_MS 1000 // Cửa sổ thời gian 1 giây

#define SUSPICIOUS\_THRESHOLD 15 // Ngưỡng để kích hoạt bảo mật

#define FAST\_KEY\_THRESHOLD 8 // Ngưỡng phím nhanh (ms)

#define PATTERN\_REPEAT\_THRESHOLD 8 // Ngưỡng pattern lặp lại

#define CONSECUTIVE\_REPEAT\_THRESHOLD 5 // Ngưỡng lặp liên tiếp

#define HIGH\_FREQ\_THRESHOLD 20 // Ngưỡng tần suất cao

#define FAST\_KEYS\_IN\_ROW\_THRESHOLD 10 // Ngưỡng phím nhanh liên tiếp

PDEVICE\_OBJECT myKbdDevice = NULL;

ULONG pendingkey = 0;

// Prototype functions

BOOLEAN IsRateLimitExceeded(PDEVICE\_EXTENSION devExt);

BOOLEAN IsPatternSuspicious(PDEVICE\_EXTENSION devExt, ULONG scanCode);

BOOLEAN IsTimingAbnormal(PDEVICE\_EXTENSION devExt, ULONG interval);

BOOLEAN IsKeyFrequencyHigh(PDEVICE\_EXTENSION devExt, ULONG scanCode);

VOID UpdatePatternAnalysis(PDEVICE\_EXTENSION devExt, ULONG scanCode);

VOID UpdateAverageInterval(PDEVICE\_EXTENSION devExt, ULONG interval);

VOID LogSuspiciousActivity(PDEVICE\_EXTENSION devExt, ULONG scanCode, ULONG reason, ULONG interval);

BOOLEAN ShouldBlockKey(PDEVICE\_EXTENSION devExt, ULONG scanCode, ULONG interval);

// Kiểm tra rate limit

BOOLEAN IsRateLimitExceeded(PDEVICE\_EXTENSION devExt)

{

LARGE\_INTEGER currentTime;

KeQuerySystemTime(&currentTime);

// Kiểm tra nếu cửa sổ thời gian mới

if ((currentTime.QuadPart - devExt->WindowStartTime.QuadPart) > (WINDOW\_SIZE\_MS \* 10000LL)) {

devExt->WindowStartTime = currentTime;

devExt->KeysInWindow = 0;

}

devExt->KeysInWindow++;

return (devExt->KeysInWindow > devExt->MaxKeysPerSecond);

}

// Phân tích pattern nâng cao

BOOLEAN IsPatternSuspicious(PDEVICE\_EXTENSION devExt, ULONG scanCode)

{

ULONG i;

ULONG sameKeyCount = 0;

ULONG sequentialCount = 0;

ULONG recentRepeats = 0;

// Đếm số lần xuất hiện của phím này trong buffer

for (i = 0; i < 20; i++) {

if (devExt->PatternBuffer[i] == scanCode) {

sameKeyCount++;

}

}

// Kiểm tra sequence liên tiếp

for (i = 1; i < 20; i++) {

if (devExt->PatternBuffer[i] == devExt->PatternBuffer[i - 1]) {

sequentialCount++;

}

}

// Kiểm tra 5 phím gần nhất có lặp lại không

for (i = 15; i < 20; i++) {

if (devExt->PatternBuffer[i] == scanCode) {

recentRepeats++;

}

}

// Suspicious nếu:

// 1. Cùng phím xuất hiện quá nhiều trong buffer

// 2. Quá nhiều phím liên tiếp giống nhau

// 3. Phím này lặp lại quá nhiều trong 5 phím gần nhất

return (sameKeyCount > PATTERN\_REPEAT\_THRESHOLD ||

sequentialCount > 8 ||

recentRepeats > 3);

}

// Kiểm tra timing bất thường

BOOLEAN IsTimingAbnormal(PDEVICE\_EXTENSION devExt, ULONG interval)

{

// Quá nhanh

if (interval < FAST\_KEY\_THRESHOLD) {

devExt->FastKeysInRow++;

return TRUE;

}

else {

devExt->FastKeysInRow = 0; // Reset counter

}

// Quá nhiều phím nhanh liên tiếp

if (devExt->FastKeysInRow > FAST\_KEYS\_IN\_ROW\_THRESHOLD) {

return TRUE;

}

// So sánh với average interval

if (devExt->AverageInterval > 0) {

// Nếu interval hiện tại quá khác biệt so với average (quá nhanh)

if (interval < devExt->AverageInterval / 4) {

return TRUE;

}

}

return FALSE;

}

// Kiểm tra tần suất phím cao

BOOLEAN IsKeyFrequencyHigh(PDEVICE\_EXTENSION devExt, ULONG scanCode)

{

return (devExt->KeyCount[scanCode] > HIGH\_FREQ\_THRESHOLD);

}

// Cập nhật pattern analysis

VOID UpdatePatternAnalysis(PDEVICE\_EXTENSION devExt, ULONG scanCode)

{

// Thêm vào pattern buffer

devExt->PatternBuffer[devExt->PatternIndex] = scanCode;

devExt->PatternIndex = (devExt->PatternIndex + 1) % 20;

// Kiểm tra consecutive repeats

if (devExt->LastPatternKey == scanCode) {

devExt->ConsecutiveRepeats++;

}

else {

devExt->ConsecutiveRepeats = 1;

devExt->LastPatternKey = scanCode;

}

// Cập nhật counter phím tần suất cao

if (devExt->KeyCount[scanCode] > HIGH\_FREQ\_THRESHOLD) {

devExt->HighFreqKeyCount++;

}

}

// Cập nhật average interval

VOID UpdateAverageInterval(PDEVICE\_EXTENSION devExt, ULONG interval)

{

if (devExt->AverageInterval == 0) {

devExt->AverageInterval = interval;

}

else {

// Exponential moving average

devExt->AverageInterval = (devExt->AverageInterval \* 7 + interval \* 3) / 10;

}

}

// Log hoạt động bất thường

VOID LogSuspiciousActivity(PDEVICE\_EXTENSION devExt, ULONG scanCode, ULONG reason, ULONG interval)

{

CHAR\* reasons[] = {

"Unknown",

"Fast typing",

"Rate limit exceeded",

"Suspicious pattern",

"High frequency key",

"Consecutive repeats",

"Timing abnormal"

};

if (reason < 7) {

KdPrint(("SUSPICIOUS: %s - Key: 0x%x, Interval: %u ms, Count: %u, Total: %u\n",

reasons[reason], scanCode, interval,

devExt->KeyCount[scanCode], devExt->SuspiciousCount));

}

}

// Quyết định có chặn phím không

BOOLEAN ShouldBlockKey(PDEVICE\_EXTENSION devExt, ULONG scanCode, ULONG interval)

{

if (!devExt->BlockingEnabled) {

return FALSE;

}

BOOLEAN block = FALSE;

ULONG blockReason = 0;

// Kiểm tra rate limit

if (IsRateLimitExceeded(devExt)) {

block = TRUE;

blockReason = 2;

}

// Kiểm tra pattern suspicious

if (IsPatternSuspicious(devExt, scanCode)) {

block = TRUE;

blockReason = 3;

}

// Kiểm tra timing abnormal

if (IsTimingAbnormal(devExt, interval)) {

block = TRUE;

blockReason = 6;

}

// Kiểm tra high frequency

if (IsKeyFrequencyHigh(devExt, scanCode)) {

block = TRUE;

blockReason = 4;

}

// Kiểm tra consecutive repeats

if (devExt->ConsecutiveRepeats > CONSECUTIVE\_REPEAT\_THRESHOLD) {

block = TRUE;

blockReason = 5;

}

// Chỉ chặn trong security mode hoặc khi quá nghiêm trọng

if (block && (devExt->SecurityMode || devExt->SuspiciousCount > SUSPICIOUS\_THRESHOLD)) {

LogSuspiciousActivity(devExt, scanCode, blockReason, interval);

return TRUE;

}

return FALSE;

}

VOID DriverUnload(PDRIVER\_OBJECT DriverObject)

{

LARGE\_INTEGER interval = { 0 };

PDEVICE\_OBJECT DeviceObject = DriverObject->DeviceObject;

interval.QuadPart = -10 \* 1000 \* 1000; // 1 giây

if (DeviceObject) {

PDEVICE\_EXTENSION devExt = (PDEVICE\_EXTENSION)DeviceObject->DeviceExtension;

// In thống kê cuối cùng

KdPrint(("=== DRIVER UNLOAD STATISTICS ===\n"));

KdPrint(("Total IRPs processed: %u\n", devExt->TotalIrpsProcessed));

KdPrint(("Suspicious activities: %u\n", devExt->SuspiciousCount));

KdPrint(("Blocked keys: %u\n", devExt->BlockedKeys));

KdPrint(("High frequency keys: %u\n", devExt->HighFreqKeyCount));

KdPrint(("Security mode: %s\n", devExt->SecurityMode ? "ON" : "OFF"));

KdPrint(("Average interval: %u ms\n", devExt->AverageInterval));

if (devExt->LowerKbdDevice) {

IoDetachDevice(devExt->LowerKbdDevice);

}

}

while (pendingkey) {

KeDelayExecutionThread(KernelMode, FALSE, &interval);

}

if (myKbdDevice) {

IoDeleteDevice(myKbdDevice);

}

KdPrint(("Enhanced Keyboard Filter Driver Unloaded\n"));

}

NTSTATUS DispatchPass(PDEVICE\_OBJECT DeviceObject, PIRP Irp)

{

if (!DeviceObject || !((PDEVICE\_EXTENSION)DeviceObject->DeviceExtension)->LowerKbdDevice) {

Irp->IoStatus.Status = STATUS\_DEVICE\_NOT\_READY;

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

return STATUS\_DEVICE\_NOT\_READY;

}

IoCopyCurrentIrpStackLocationToNext(Irp);

return IoCallDriver(((PDEVICE\_EXTENSION)DeviceObject->DeviceExtension)->LowerKbdDevice, Irp);

}

NTSTATUS ReadComplete(PDEVICE\_OBJECT DeviceObject, PIRP Irp, PVOID Context)

{

CHAR\* keyflag[4] = { "KeyDown", "KeyUp", "E0", "E1" };

PKEYBOARD\_INPUT\_DATA Keys = (PKEYBOARD\_INPUT\_DATA)Irp->AssociatedIrp.SystemBuffer;

PDEVICE\_EXTENSION devExt = (PDEVICE\_EXTENSION)DeviceObject->DeviceExtension;

int structnum = Irp->IoStatus.Information / sizeof(KEYBOARD\_INPUT\_DATA);

int i;

KIRQL oldIrql;

devExt->TotalIrpsProcessed++;

devExt->KeyPressesInIrp = 0;

KdPrint(("Processing IRP %u, structnum %d\n", devExt->TotalIrpsProcessed, structnum));

if (Irp->IoStatus.Status == STATUS\_SUCCESS) {

LARGE\_INTEGER currentTime;

KeQuerySystemTimePrecise(&currentTime);

KeAcquireSpinLock(&devExt->DataLock, &oldIrql);

for (i = 0; i < structnum; i++) {

// Ghi log phím

KdPrint(("Scan code: %x (%s)\n", Keys[i].MakeCode, keyflag[Keys[i].Flags]));

// Chỉ xử lý KeyDown để tránh đếm trùng

if (Keys[i].Flags == 0) {

devExt->KeyPressesInIrp++;

ULONG interval = 0;

BOOLEAN shouldBlock = FALSE;

// Tính khoảng thời gian giữa các phím

if (devExt->LastKeyTime.QuadPart != 0) {

interval = (ULONG)((currentTime.QuadPart - devExt->LastKeyTime.QuadPart) / 10000);

devExt->KeyIntervals[devExt->IntervalIndex++] = interval;

if (devExt->IntervalIndex >= 100) devExt->IntervalIndex = 0;

// Cập nhật average interval

UpdateAverageInterval(devExt, interval);

// Kiểm tra có nên chặn phím không

shouldBlock = ShouldBlockKey(devExt, Keys[i].MakeCode, interval);

}

// Cập nhật thống kê

devExt->KeyCount[Keys[i].MakeCode]++;

UpdatePatternAnalysis(devExt, Keys[i].MakeCode);

// Phát hiện bất thường cơ bản

if (interval < 10 || devExt->KeyCount[Keys[i].MakeCode] > 10) {

devExt->SuspiciousCount++;

KdPrint(("Suspicious activity: Key %x, Interval %u ms, Count %u, SuspiciousCount %u\n",

Keys[i].MakeCode, interval, devExt->KeyCount[Keys[i].MakeCode], devExt->SuspiciousCount));

}

// Kích hoạt security mode nếu cần

if (devExt->SuspiciousCount > SUSPICIOUS\_THRESHOLD && !devExt->SecurityMode) {

devExt->SecurityMode = TRUE;

devExt->BlockingEnabled = TRUE;

KdPrint(("SECURITY MODE ACTIVATED - Suspicious count: %u\n", devExt->SuspiciousCount));

}

// Chặn phím nếu cần

if (shouldBlock) {

devExt->BlockedKeys++;

KdPrint(("KEY BLOCKED: 0x%x, Total blocked: %u\n", Keys[i].MakeCode, devExt->BlockedKeys));

// Vô hiệu hóa phím bằng cách đặt MakeCode = 0

Keys[i].MakeCode = 0;

Keys[i].Flags = 0xFF; // Invalid flag

}

devExt->LastKeyTime = currentTime;

// Log chi tiết

KdPrint(("Key %x: Count=%u, Interval=%u ms, Avg=%u ms, FastInRow=%u, ConsecutiveRepeats=%u %s\n",

Keys[i].MakeCode, devExt->KeyCount[Keys[i].MakeCode], interval,

devExt->AverageInterval, devExt->FastKeysInRow, devExt->ConsecutiveRepeats,

shouldBlock ? "[BLOCKED]" : ""));

}

}

KeReleaseSpinLock(&devExt->DataLock, oldIrql);

}

if (Irp->PendingReturned) {

IoMarkIrpPending(Irp);

}

pendingkey--;

return Irp->IoStatus.Status;

}

NTSTATUS DispatchRead(PDEVICE\_OBJECT DeviceObject, PIRP Irp)

{

if (!DeviceObject || !((PDEVICE\_EXTENSION)DeviceObject->DeviceExtension)->LowerKbdDevice) {

Irp->IoStatus.Status = STATUS\_DEVICE\_NOT\_READY;

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

return STATUS\_DEVICE\_NOT\_READY;

}

IoCopyCurrentIrpStackLocationToNext(Irp);

IoSetCompletionRoutine(Irp, ReadComplete, NULL, TRUE, TRUE, TRUE);

pendingkey++;

return IoCallDriver(((PDEVICE\_EXTENSION)DeviceObject->DeviceExtension)->LowerKbdDevice, Irp);

}

NTSTATUS MyAttachDevice(PDRIVER\_OBJECT DriverObject)

{

NTSTATUS status;

UNICODE\_STRING TargetDevice = RTL\_CONSTANT\_STRING(L"\\Device\\KeyboardClass0");

status = IoCreateDevice(DriverObject, sizeof(DEVICE\_EXTENSION), NULL, FILE\_DEVICE\_KEYBOARD, 0, FALSE, &myKbdDevice);

if (!NT\_SUCCESS(status)) {

KdPrint(("Failed to create device: %x\n", status));

return status;

}

myKbdDevice->Flags |= DO\_BUFFERED\_IO;

myKbdDevice->Flags &= ~DO\_DEVICE\_INITIALIZING;

PDEVICE\_EXTENSION devExt = (PDEVICE\_EXTENSION)myKbdDevice->DeviceExtension;

RtlZeroMemory(devExt, sizeof(DEVICE\_EXTENSION));

// Khởi tạo các giá trị mặc định

devExt->TotalIrpsProcessed = 0;

devExt->KeyPressesInIrp = 0;

devExt->SecurityMode = FALSE;

devExt->BlockingEnabled = FALSE;

devExt->MaxKeysPerSecond = MAX\_KEYS\_PER\_SECOND;

devExt->ConsecutiveRepeats = 0;

devExt->LastPatternKey = 0;

devExt->HighFreqKeyCount = 0;

devExt->AverageInterval = 0;

devExt->FastKeysInRow = 0;

// Khởi tạo spinlock

KeInitializeSpinLock(&devExt->DataLock);

// Khởi tạo thời gian window

KeQuerySystemTime(&devExt->WindowStartTime);

status = IoAttachDevice(myKbdDevice, &TargetDevice, &devExt->LowerKbdDevice);

if (!NT\_SUCCESS(status)) {

KdPrint(("Failed to attach device: %x\n", status));

IoDeleteDevice(myKbdDevice);

myKbdDevice = NULL;

return status;

}

KdPrint(("Enhanced Keyboard Filter attached successfully\n"));

KdPrint(("Configuration: MaxKeys/sec=%u, FastThreshold=%u ms, BlockingEnabled=%s\n",

devExt->MaxKeysPerSecond, FAST\_KEY\_THRESHOLD, devExt->BlockingEnabled ? "YES" : "NO"));

return STATUS\_SUCCESS;

}

NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath)

{

NTSTATUS status;

int i;

DriverObject->DriverUnload = DriverUnload;

for (i = 0; i <= IRP\_MJ\_MAXIMUM\_FUNCTION; i++) {

DriverObject->MajorFunction[i] = DispatchPass;

}

DriverObject->MajorFunction[IRP\_MJ\_READ] = DispatchRead;

KdPrint(("Enhanced Keyboard Filter Driver loaded\n"));

status = MyAttachDevice(DriverObject);

if (!NT\_SUCCESS(status)) {

KdPrint(("Attaching failed: %x\n", status));

}

else {

KdPrint(("Enhanced Keyboard Filter with blocking and pattern analysis active\n"));

}

return status;

}