Anti-RE Techniques in DRM Code

Jan Newger

Seminar on Advanced Exploitation Techniques
i4, RWTH Aachen, Germany

DEF CON 16 / 2008

Outline

1. Introduction
   - About DRMs
   - Approaching the DRM
2. SEH Basics
   - Overview
   - Exception Dispatching
3. Anti-RE Techniques
   - Overview
   - Trampolines
   - Debug Registers
   - P-Code Machine
4. Decrypting the Content
   - The Algorithm
   - Demo

Outline

1. Introduction
   - About DRMs
   - Approaching the DRM
2. SEH Basics
   - Overview
   - Exception Dispatching
3. Anti-RE Techniques
   - Overview
   - Trampolines
   - Debug Registers
   - P-Code Machine
4. Decrypting the Content
   - The Algorithm
   - Demo

About this Talk

What it *IS*

- Show some not-so-common Anti-RE techniques
- Strategies to defeat Anti-RE

What it is *NOT*

- How to hack the DRM from *********
- No tutorial for writing decryption tools

What's a DRM?

- "Digital Rights Management"
  - Restrict access to content
  - Content encrypted
  - Decrypt online

What's a DRM?

- "Digital Rights Management"
  - Restrict access to content
  - Content encrypted
  - Decrypt online
- Key often bound to user/hardware
  - Prevents copying
  - Change hardware -> new license

What's a DRM?

- "Digital Rights Management"
  - Restrict access to content
  - Content encrypted
  - Decrypt online
- Key often bound to user/hardware
  - Prevents copying
  - Change hardware -> new license
- Media key, hardware key, player key, content key...

What's a DRM?

- "Digital Rights Management"
  - Restrict access to content
  - Content encrypted
  - Decrypt online
- Key often bound to user/hardware
  - Prevents copying
  - Change hardware -> new license
- Media key, hardware key, player key, content key...
- Obviously: every DRM can be broken

Possible Strategies (1)

Ultimate Goal

Find code for content decryption and the associated key setup

Obvious Approach

1. BPs on file I/O APIs (CreateFile, ReadFile, MMF)
2. Set BPM on filebuffer
   - either stops on copy operation
   - or breaks on decryption

Possible Strategies (1)

Ultimate Goal

Find code for content decryption and the associated key setup

Obvious Approach

1. BPs on file I/O APIs (CreateFile, ReadFile, MMF)
2. Set BPM on filebuffer
   - either stops on copy operation
   - or breaks on decryption

Obvious approach impossible!

DRM System prevents this strategy by blocking the debug registers!

Possible Strategies (2)

Code Coverage

Runtime analysis to spot relevant code by recording execution of basic blocks / functions

Code Coverage Limitation

- Here: Impossible to find DRM code itself using code coverage!
- Gives some good starting points, though

Basic Approach Summary

Our Strategy

- Use code coverage to spot some places to investigate
- Use *obvious approach* to find decryption code

Outline

1. Introduction
   - About DRMs
   - Approaching the DRM
2. SEH Basics
   - Overview
   - Exception Dispatching
3. Anti-RE Techniques
   - Overview
   - Trampolines
   - Debug Registers
   - P-Code Machine
4. Decrypting the Content
   - The Algorithm
   - Demo

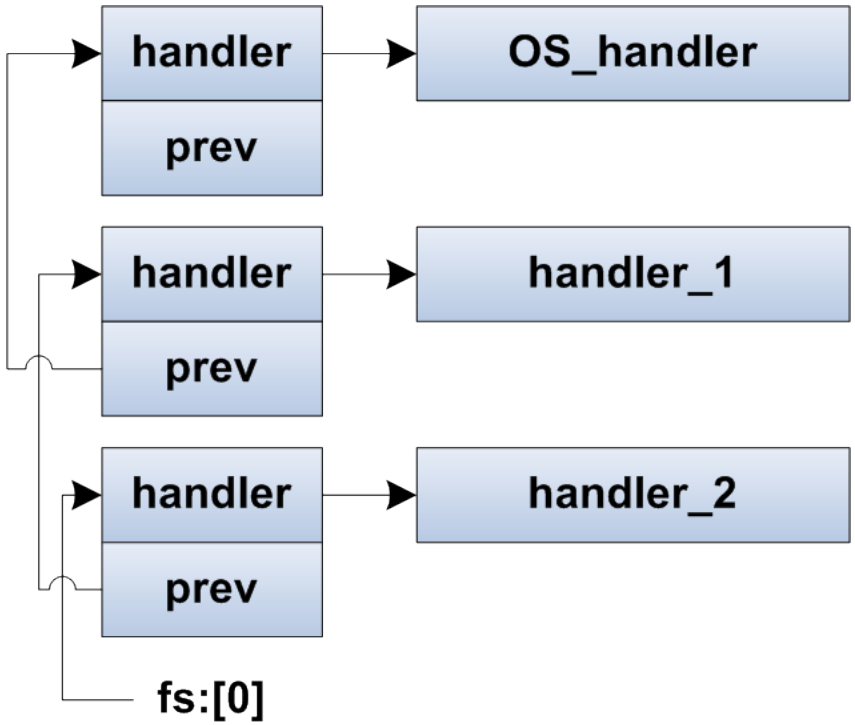
Windows SEH

Structured Exception Handling

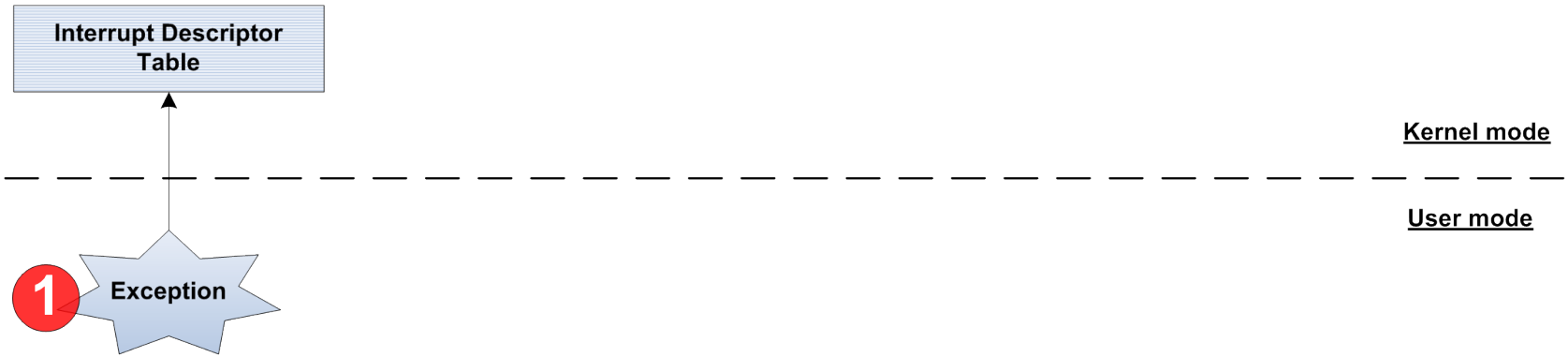
- Dispatch exception on a per-thread-basis
- Linked list of handlers starting at `fs:[0]`
- On exception OS walks list of faulting thread
- When called, a handler can:
  - Handle exception and ask OS to continue execution
  - Refuse to handle exception

SEH Handler

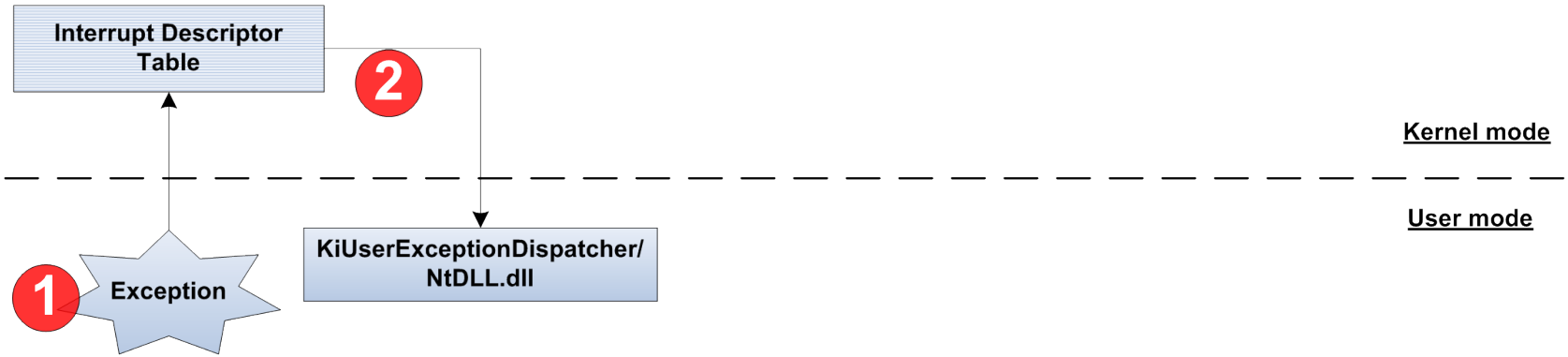
```
EXCEPTION_DISPOSITION _except_handler(_EXCEPTION_RECORD* ExceptionRecord,
                                      void* EstablisherFrame,
                                      _CONTEXT* ContextRecord,
                                      void* DispatcherContext);
```



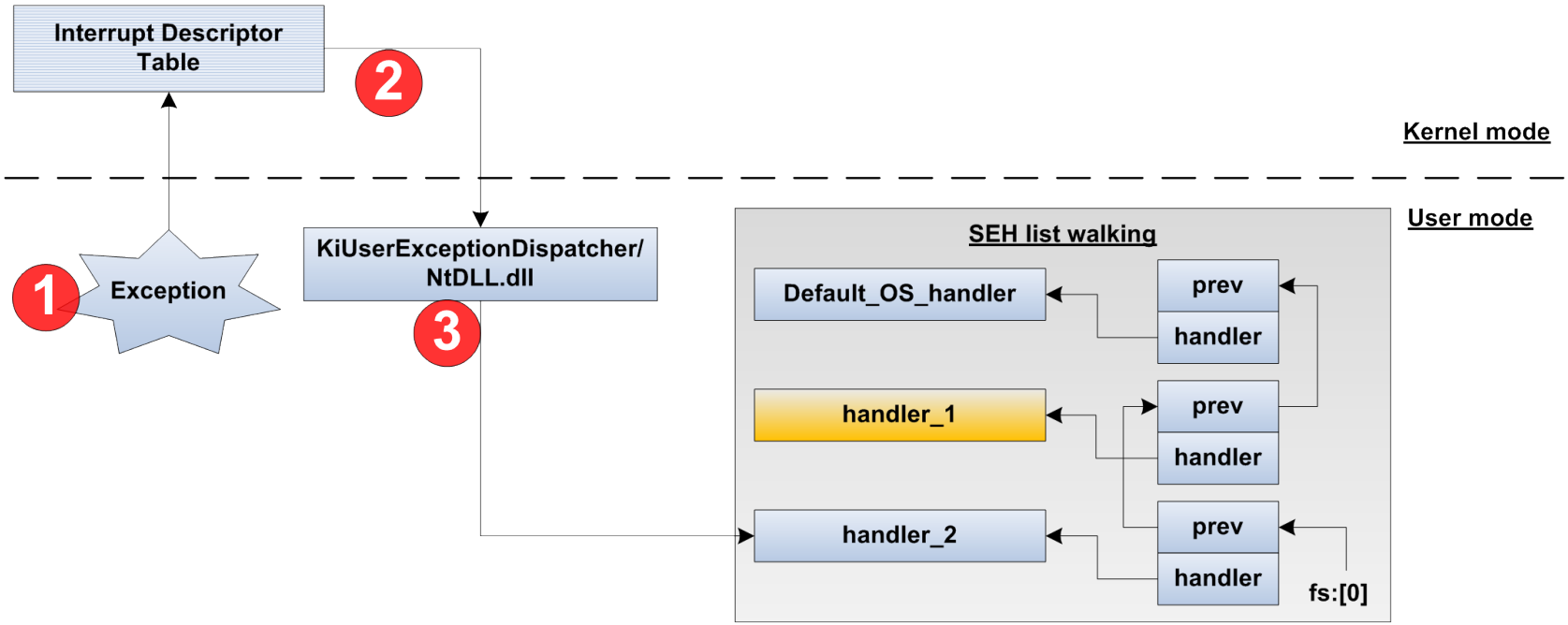
SEH Handler Invocation



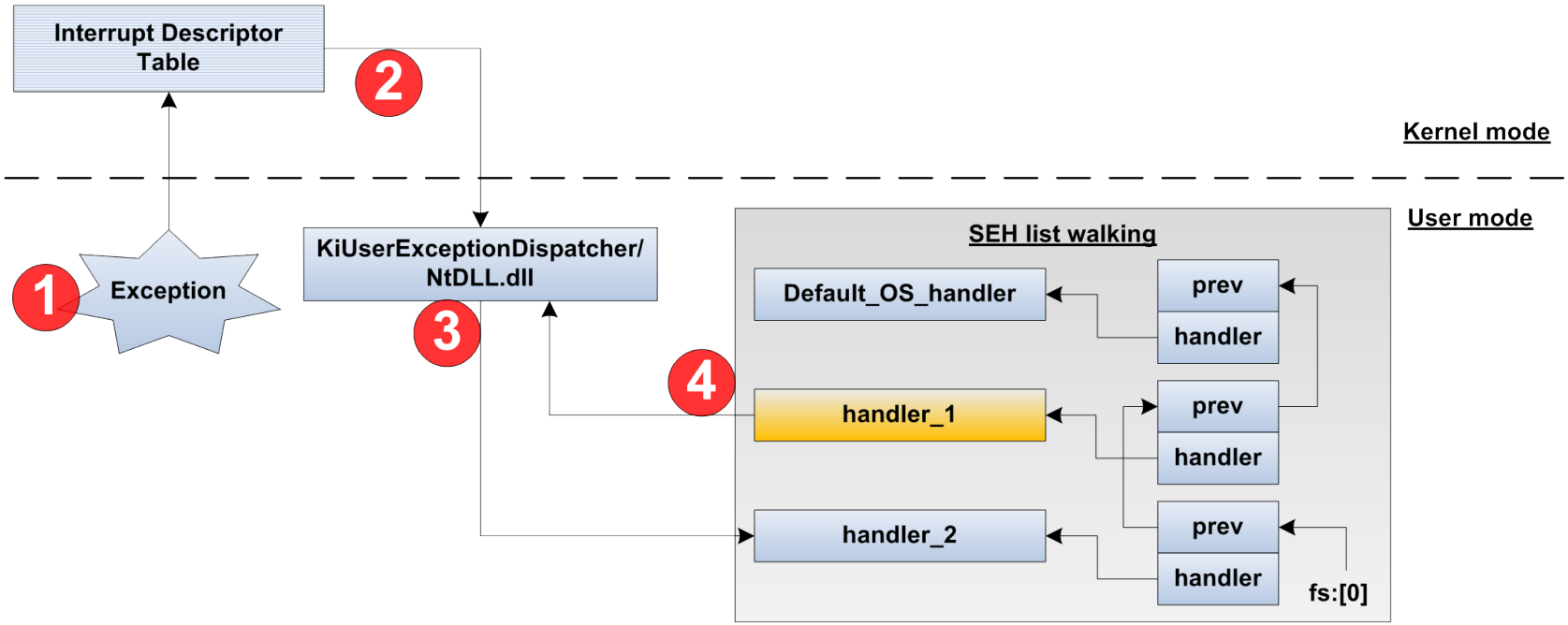
SEH Handler Invocation



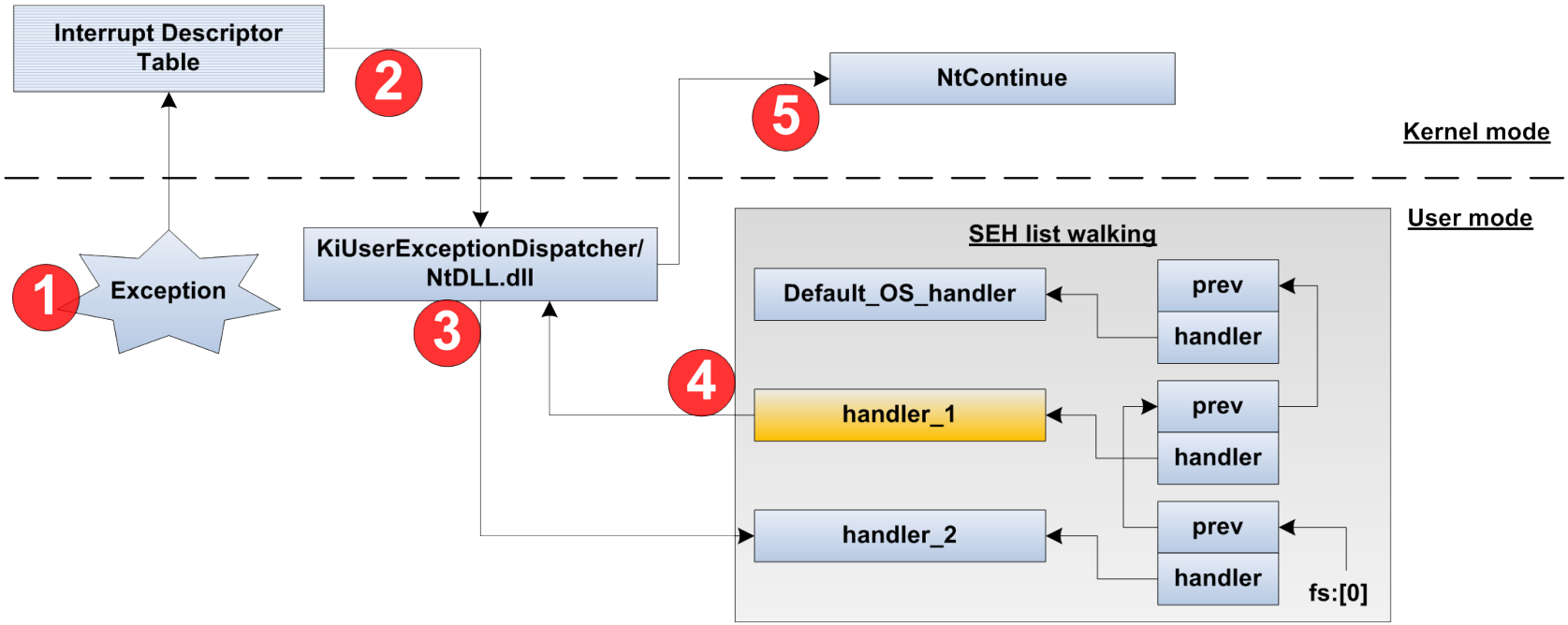
SEH Handler Invocation



SEH Handler Invocation



SEH Handler Invocation



Remarks

SEH Handler Invocation

Simplified view, because

- No stack unwinding
- No collided unwind, nested exceptions
- Handler can decide not to return (C++, try...except)

But good enough for our analysis!

Outline

1. Introduction
   - About DRMs
   - Approaching the DRM
2. SEH Basics
   - Overview
   - Exception Dispatching
3. Anti-RE Techniques
   - Overview
   - Trampolines
   - Debug Registers
   - P-Code Machine
4. Decrypting the Content
   - The Algorithm
   - Demo

The DRM Protection (1)

Control Flow Obfuscation

- Use fake exceptions to interrupt control flow
- Handlers change thread context
- Inter-/intra-modular calls through call tables
- Use dynamically allocated trampolines
- P-Code machine

The DRM Protection (2)

Anti-Debugging

- Check PEB flag
- Scan APIs for `0xCC`
- Usage of debug registers (no BPM/BPX)
- Special files contain code uncompressed at runtime
- Use fake exceptions to detect debugger

Trampolines Overview

Trampoline Definition

- Copy code at runtime to randomized location (`RDTSC`), execute from there

Trampoline Execution

1. Change control flow via fake exceptions (single step exception)
2. Exception handler modifies `EIP` based on debug register values
3. Execution resumes at next trampoline

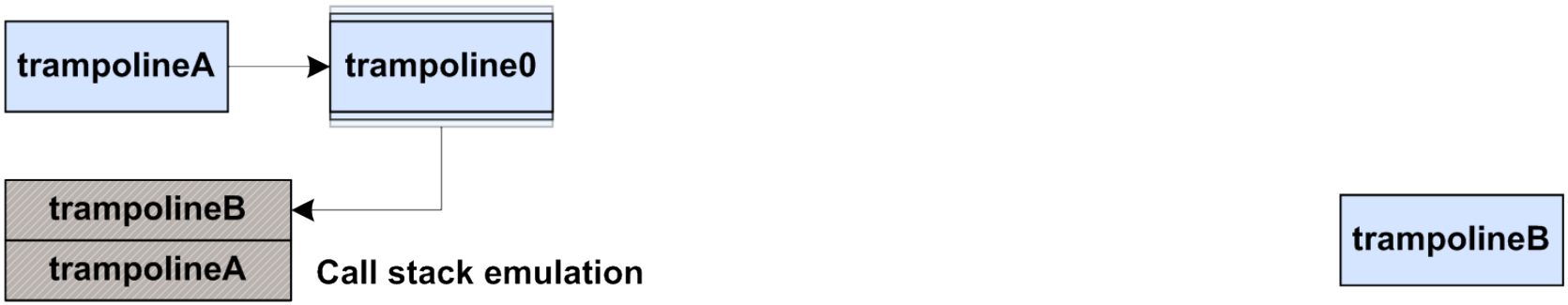
Trampoline Details



Trampoline Control Flow

- Trampoline A transfers control flow to trampoline B
- Control flow entirely depends on jumps and exceptions
- No `call` or `ret` instructions, no direct control flow between trampolines
- Therefore, a call hierarchy emulation is implemented

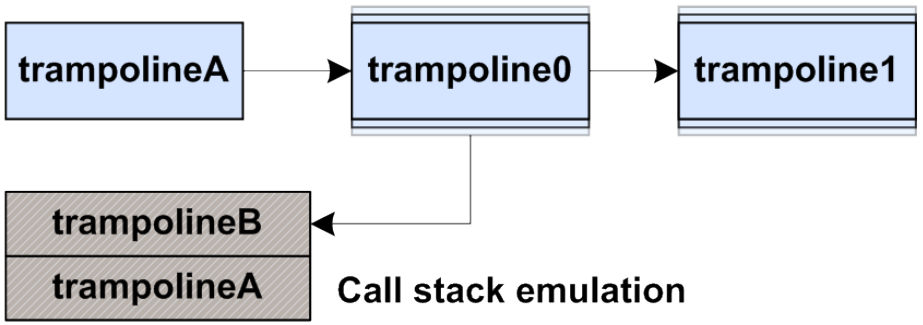
Trampoline Details



Trampoline0

- TrampolineA copies trampoline0 and jumps to it
- Trampoline0 manages internal call hierarchy
- Put destination trampoline on stack
- Copies next trampoline to random location

Trampoline Details

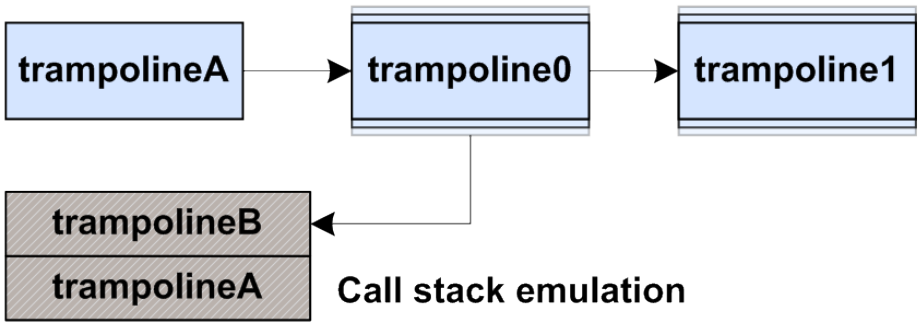




Trampoline1

- Copy trampoline0 to random location
- Install SEH frame and trigger single step exception

Trampoline Details





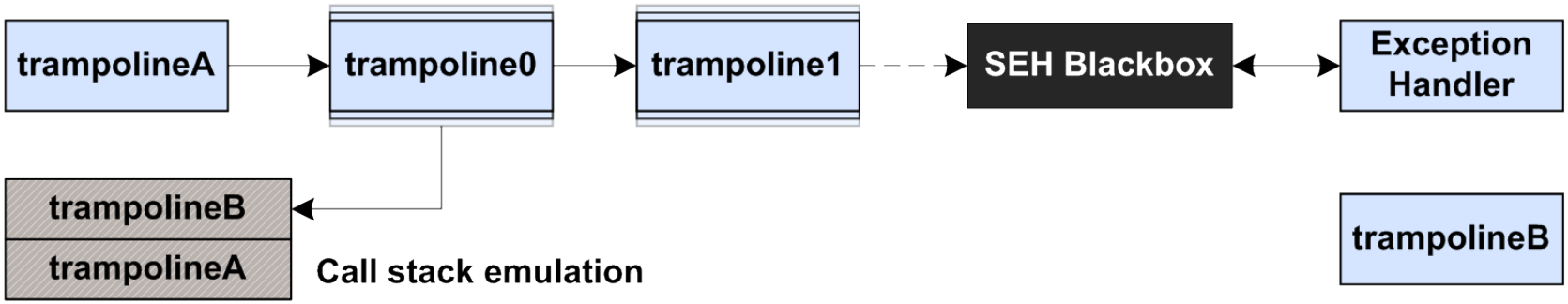
Trampoline1

- Copy trampoline0 to random location
- Install SEH frame and trigger single step exception

Trigger Exception

```
pushf
pop eax
or eax, 100h
push eax
popf
```

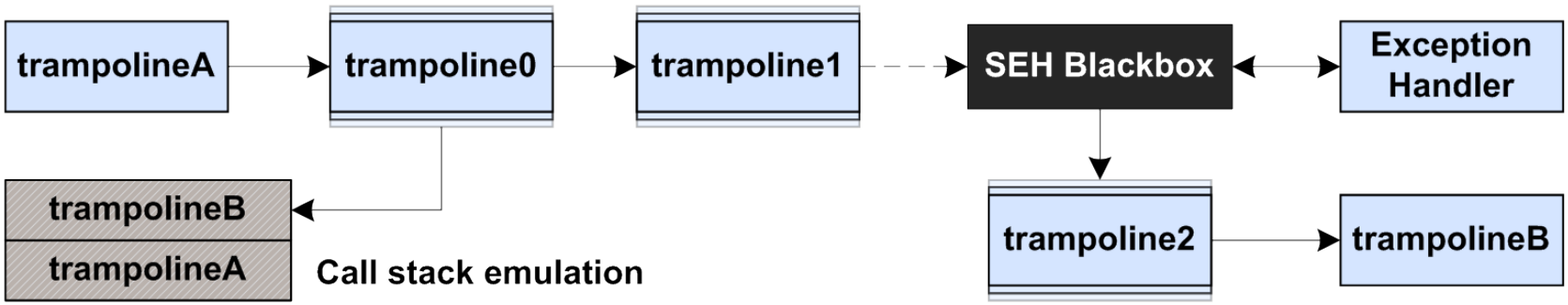
Trampoline Details



Exception Handler

- Changes EIP based on debug register values
- Clear TF bit, remove SEH frame, clean stack

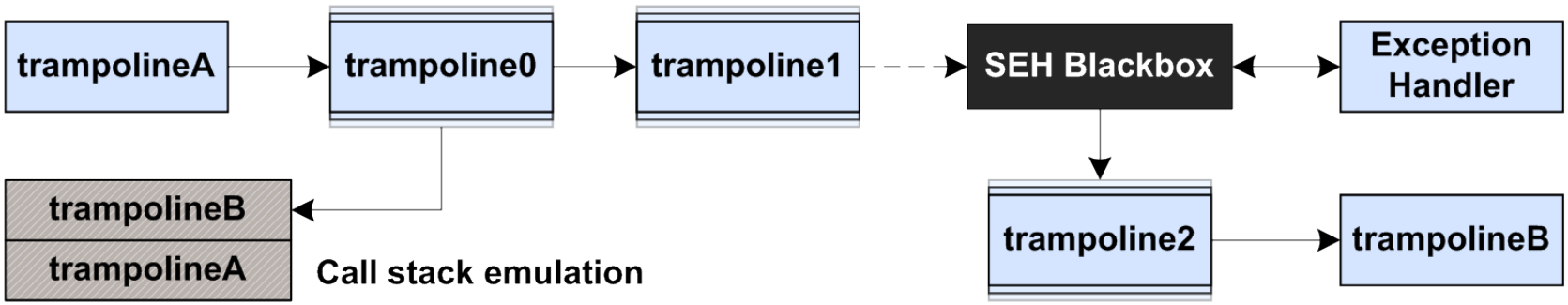
Trampoline Details



Trampoline2

- Copy destination trampoline
- Jump to destination trampoline

Trampoline Details



Call Stack Emulation

The `ret` instruction is emulated by a similar mechanism!

- Special exception handler removes trampoline from internal call stack
- Modifies context, execution resumes

More Trampoline Details

Use of the Debug Registers

- DR0 and DR6 are zeroed out
- DR1 contains pointer to a shared stack area to pass data between trampolines
- DR2 holds trampoline address, which is used to perform return emulation
- DR3 holds the address of the starting trampoline (trampoline0)
- DR7 is used to turn hardware breakpoints on and very frequently

Impact of Trampolines

Impact on RE

- Debugging pretty annoying, trampoline addresses jitter
- Control flow depends on DRs, so no BPM/BPX
- No call stack, i.e. back tracing difficult
- We can´t *execute until return*, difficult to tell who called us
- No direct *call* between subs, less X-Refs
- Absence of `ret` instructions confuses disassembler

Impact of Trampolines

Impact on RE

- Debugging pretty annoying, trampoline addresses jitter
- Control flow depends on DRs, so no BPM/BPX
- No call stack, i.e. back tracing difficult
- We can´t *execute until return*, difficult to tell who called us
- No direct *call* between subs, less X-Refs
- Absence of `ret` instructions confuses disassembler
- But: Once understood we get perfect call stack

Ease Impact of Trampolines

Idea

- *Fix* trampoline addresses
- Use kernel mode driver

Ease Impact of Trampolines

Idea

- *Fix* trampoline addresses
- Use kernel mode driver

Driver

1. Turn `RDTSC` into privileged instruction (`TSD` flag, `CR4` register)
2. Hook `IDT`
3. Return zero upon exception if
   - Exception from user mode
   - Instruction was `RDTSC`

   else jump to original handler

Reclaiming the Debug Registers (1)

Usage of DRs

- DRM system uses DRs for storage
- Uses `SetThreadContext` API
- Debugger cannot use hardware breakpoints (crash or no break)

Reclaiming the Debug Registers (1)

Usage of DRs

- DRM system uses DRs for storage
- Uses `SetThreadContext` API
- Debugger cannot use hardware breakpoints (crash or no break)

Strategy

But we need BPMs for our strategy!

Reclaiming the Debug Registers (2)

Use API Hooking

- Hook into Set/GetThreadContext API
- Redirect modifications to internal storage
- DRM System cannot modify DRs anymore!
- Debugger can use DRs

Reclaiming the Debug Registers (2)

Use API Hooking

- Hook into Set/GetThreadContext API
- Redirect modifications to internal storage
- DRM System cannot modify DRs anymore!
- Debugger can use DRs

Really?

- Hardware breakpoints still don't work!
- Why?

Context Emulation

Problem

- Modification of `EIP` depends on DRs
- Two thread contexts: kernel mode vs. internal storage

Context Emulation

Problem

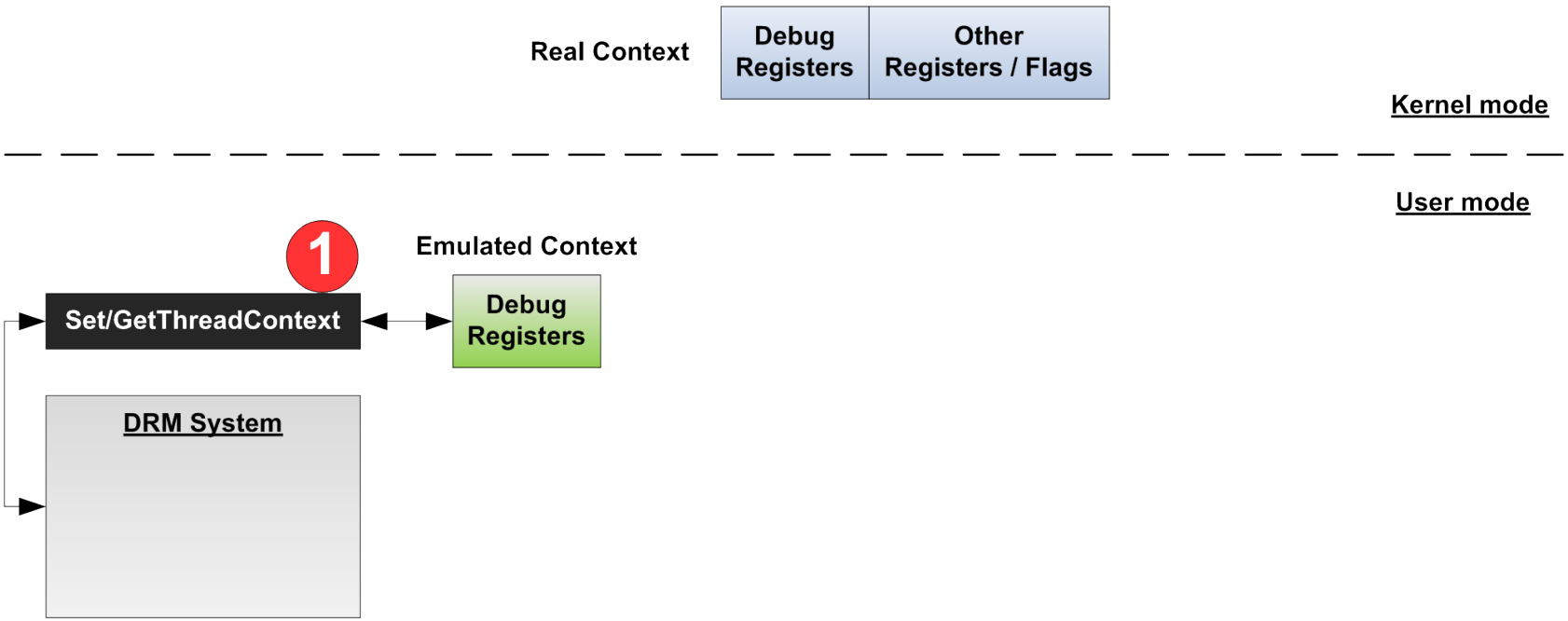
- Modification of `EIP` depends on DRs
- Two thread contexts: kernel mode vs. internal storage

Hook KiUserExceptionDispatcher

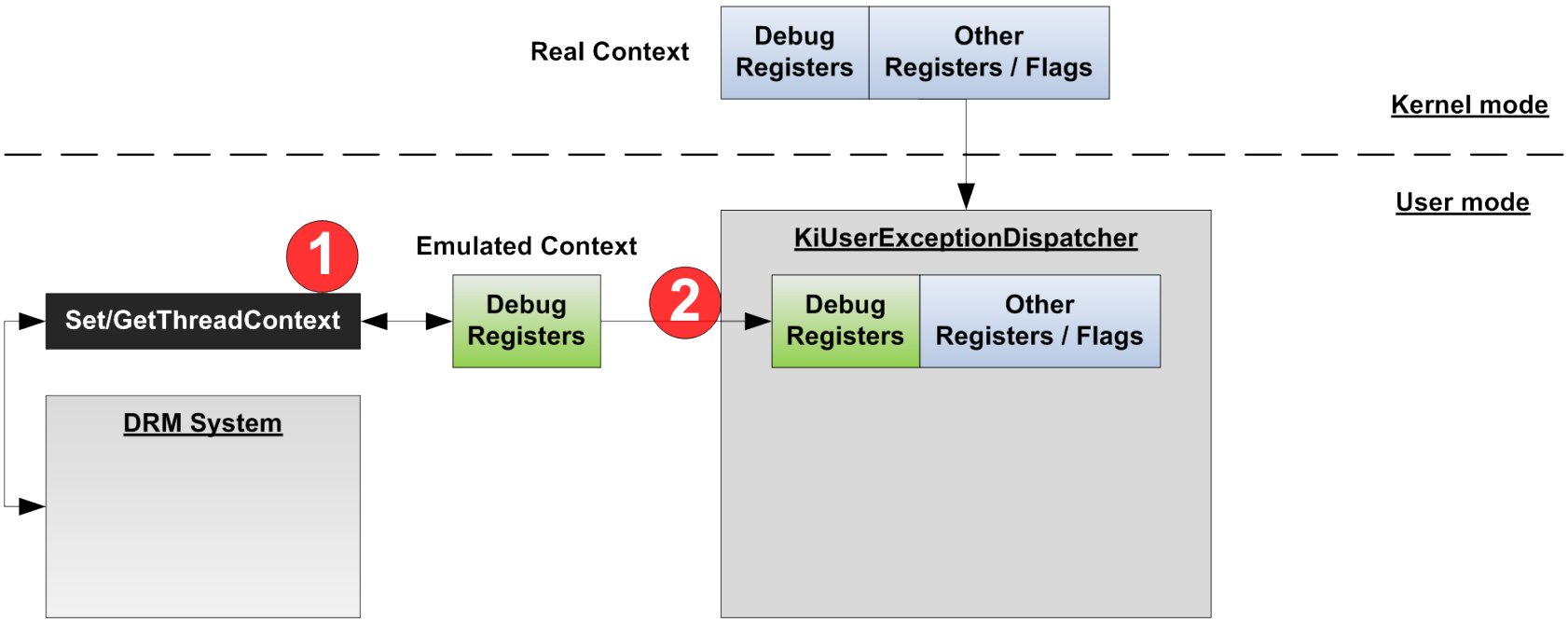
If *fake exception*, execute re-implemented KiUserExceptionDispatcher:

1. Pass fake context, DR values from internal storage
2. On return copy modifications to real context
3. Apply context via `NtContinue`

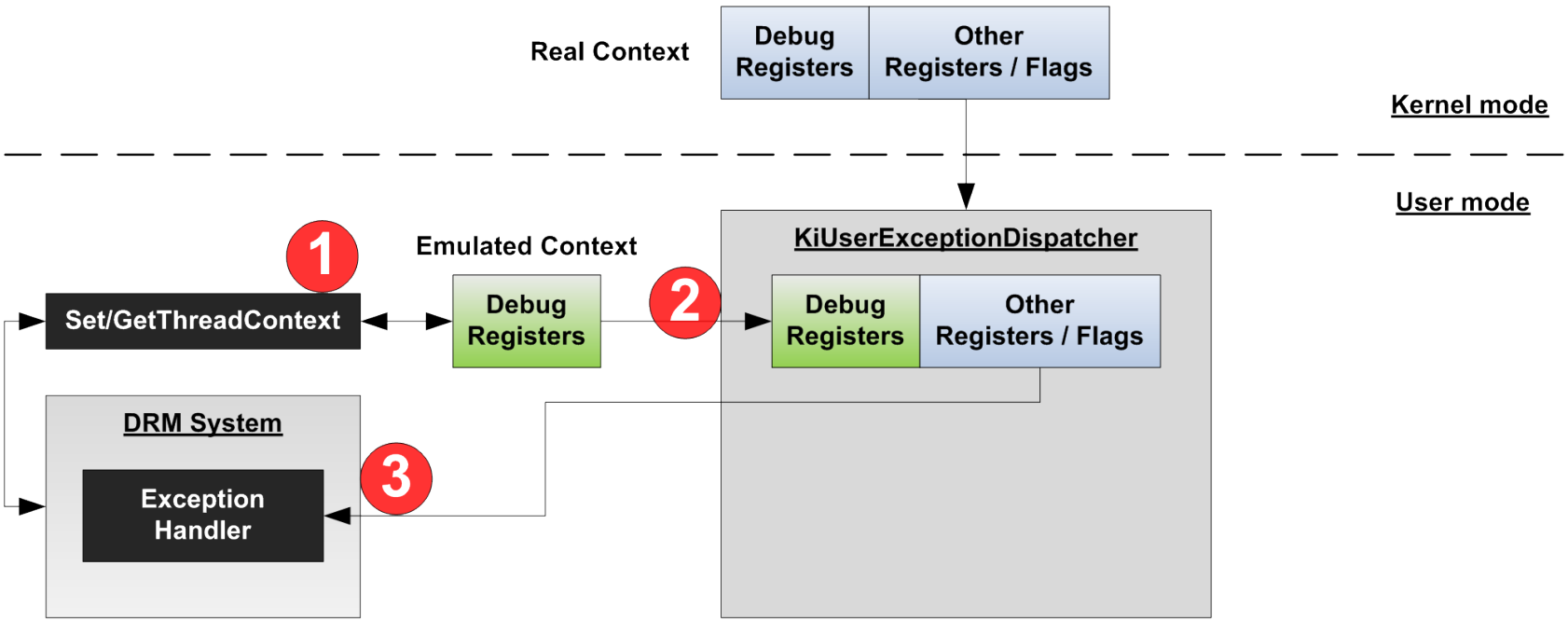
KiUserExceptionDispatcher - Re-implemented



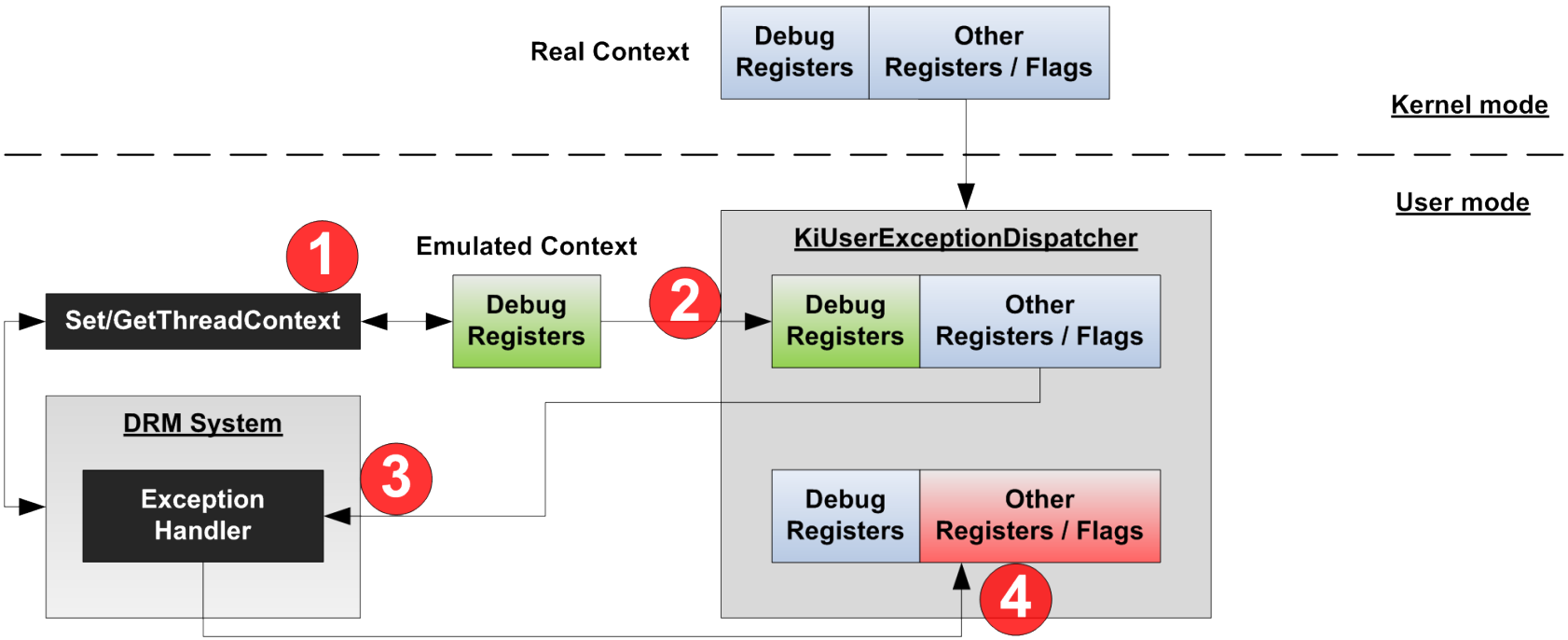
KiUserExceptionDispatcher - Re-implemented



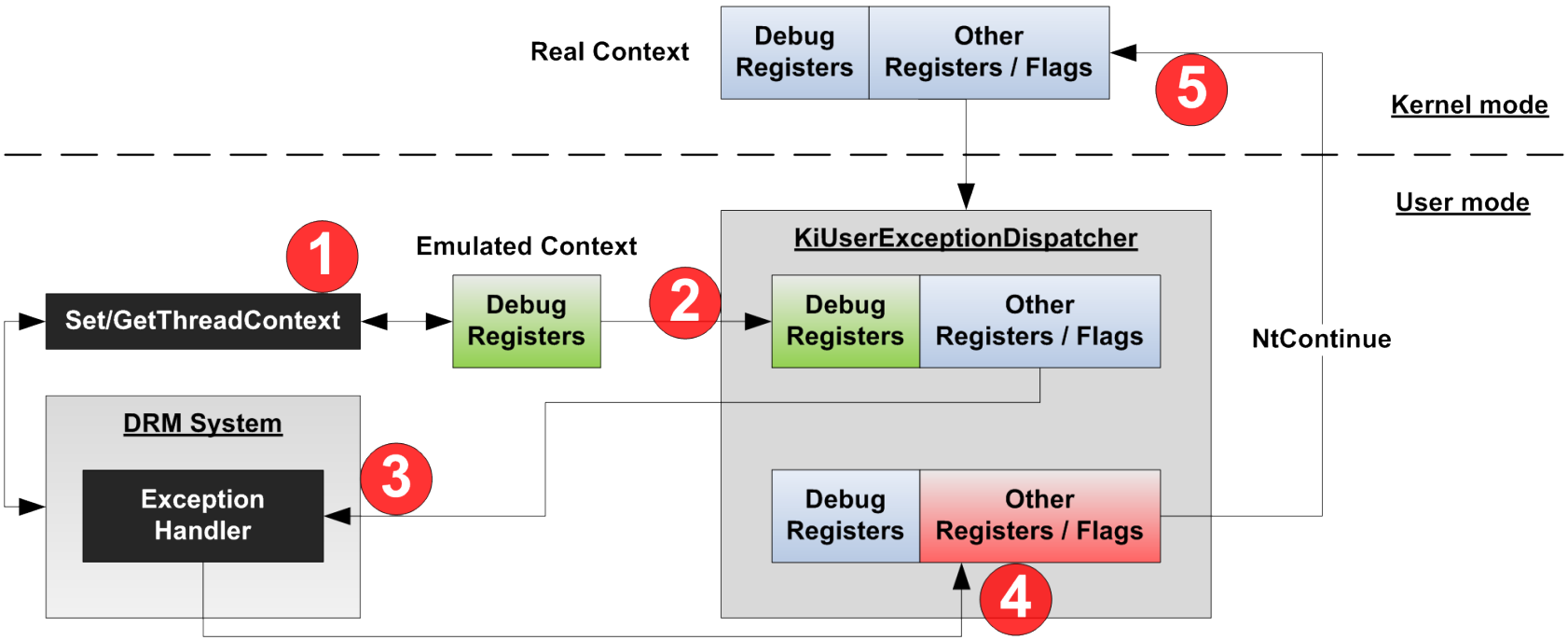
KiUserExceptionDispatcher - Re-implemented



KiUserExceptionDispatcher - Re-implemented



KiUserExceptionDispatcher - Re-implemented



Summary

Situation after Countermeasures

- DRM System cannot modify DRs - API hook
- Exception handler gets expected values - KiUserExceptionDispatcher patch
- Our debugger can use hardware breakpoints!

P-Code Machine Overview

Machine Properties

- Stack based with result register
- 256 fixed size opcodes (1 byte)
- Opcodes interleaved with data (ASN.1)
- Allocate memory in host machine
- High-level opcodes (load opcodes, call into other modules, music decoding)
- Low level opcodes, emulate virtual CPU

Loading of Opcodes

Opcode Module Files

- Special module which contains P-Code machine
- Contain native code + opcodes
- Decompressed at runtime
- No PE, no IAT, no sections, etc.
- Relocation table + some fixed imports (MSVCRT)

Obfuscation in the P-Code Machine

Executing Opcodes

- Per-module random pool
- Randomize opcode <-> opcode handler
- Descramble opcodes with PRNG in machine
- Garbage data interleaved with opcodes
- Data parsed via ASN.1

Impact of the P-Code Machine

Static RE Difficult

- Understand machine itself first
- Different meaning of opcodes per module
- ASN.1 parsing

Impact of the P-Code Machine

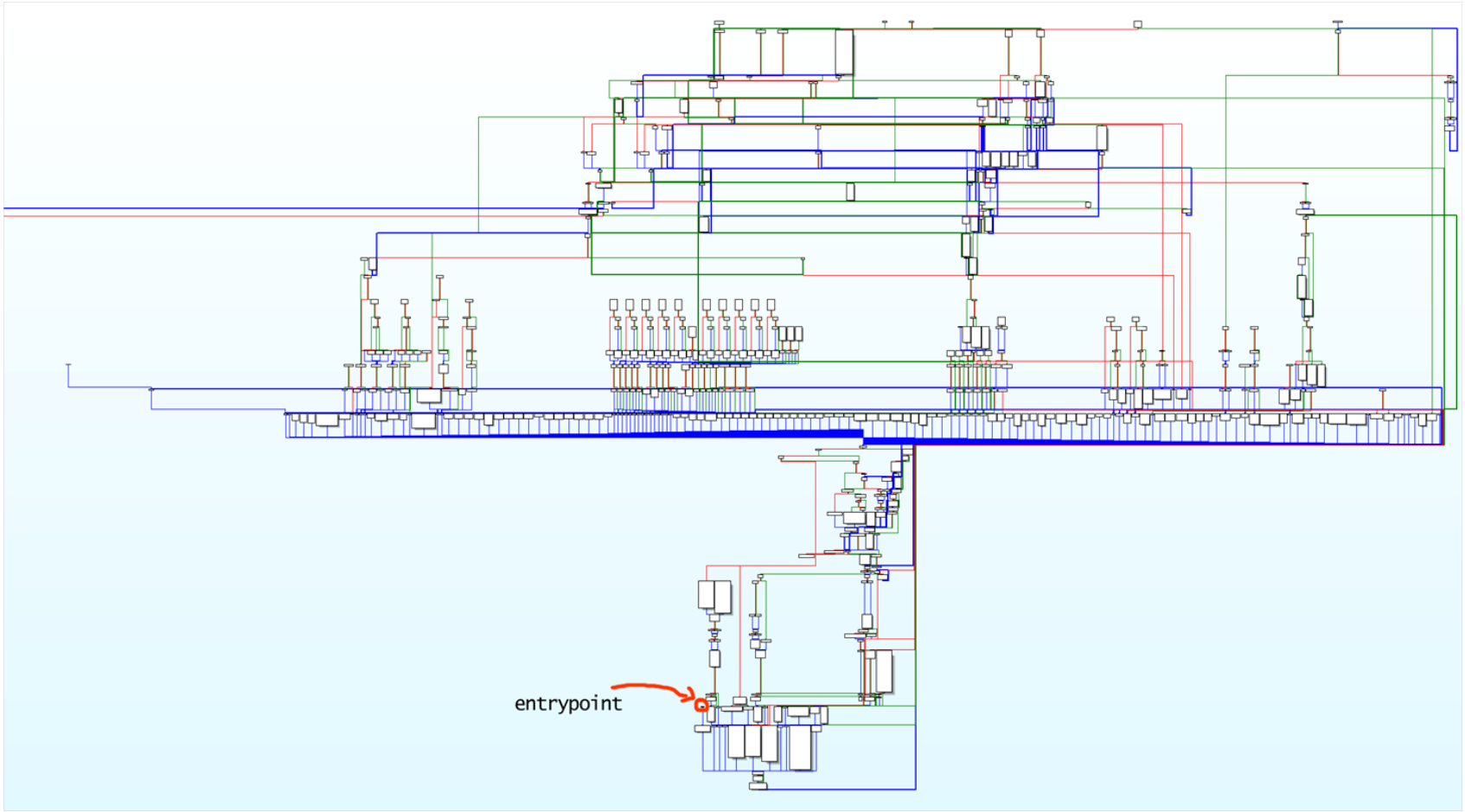
Static RE Difficult

- Understand machine itself first
- Different meaning of opcodes per module
- ASN.1 parsing

Debugging Difficult

- Low signal to noise ratio (big "handler loop")
- Even lower due to opcode descrambling

P-Code Machine in IDA



Strategies to find Decryption Algorithm + Keysetup

1. Write custom Disassembler (*Expensive Strategy*)
   - Many handlers
   - Long and complex high level handlers
   - Re-assemble randomization, descrambling, garbage instructions, ASN.1

Strategies to find Decryption Algorithm + Keysetup

1. Write custom Disassembler (*Expensive Strategy*)
   - Many handlers
   - Long and complex high level handlers
   - Re-assemble randomization, descrambling, garbage instructions, ASN.1
2. Trace until key written to memory (*Brute Force Strategy*)
   - Single-step via debugger script
   - Slow, but reaches code writing key
   - Not so clever

Strategies to find Decryption Algorithm + Keysetup

1. Write custom Disassembler (*Expensive Strategy*)
   - Many handlers
   - Long and complex high level handlers
   - Re-assemble randomization, descrambling, garbage instructions, ASN.1
2. Trace until key written to memory (*Brute Force Strategy*)
   - Single-step via debugger script
   - Slow, but reaches code writing key
   - Not so clever
3. Use emulation (*Cool Strategy*)
   - Use CPU emulation (PyEmu, x86 Emu for IDA, ...)
   - Fast, very flexible

Strategies to find Decryption Algorithm + Keysetup

4. Use BPMs / Attack machine memory (*Lazy Strategy*)
   - Use what we have
   - Exploit machine memory management
   - Filebuffer size 0x1800, DES keyschedule size 0x80
   - Set BP, fire when keysetup memory allocated
   - Set BPM, fire when keysetup written
   - Back-trace from there

Strategies to find Decryption Algorithm + Keysetup

4. Use BPMs / Attack machine memory (*Lazy Strategy*)
   - Use what we have
   - Exploit machine memory management
   - Filebuffer size 0x1800, DES keyschedule size 0x80
   - Set BP, fire when keysetup memory allocated
   - Set BPM, fire when keysetup written
   - Back-trace from there

Keen Disappointment

Decryption and keysetup in native code! High-level handlers!

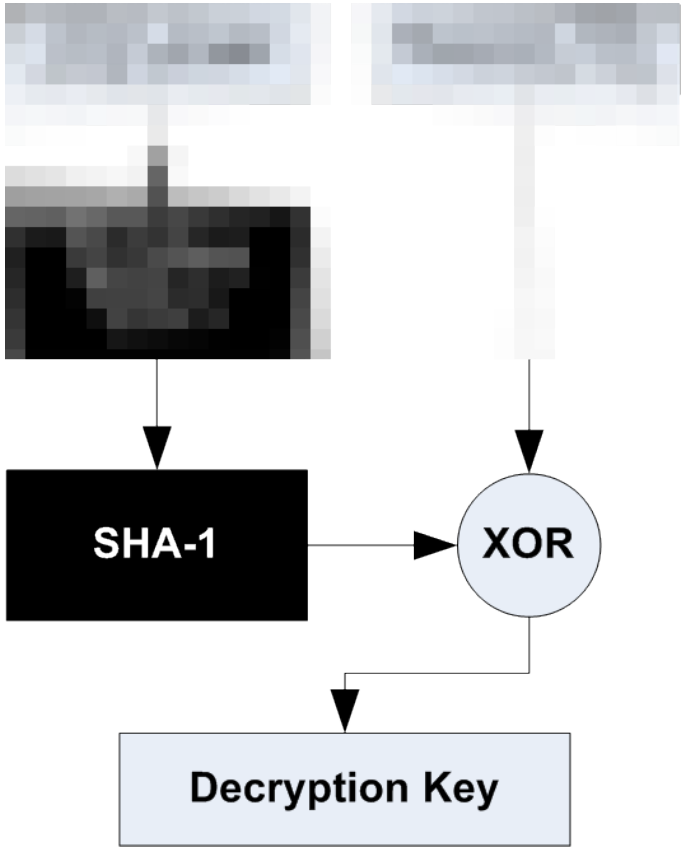
Outline

1. Introduction
   - About DRMs
   - Approaching the DRM
2. SEH Basics
   - Overview
   - Exception Dispatching
3. Anti-RE Techniques
   - Overview
   - Trampolines
   - Debug Registers
   - P-Code Machine
4. Decrypting the Content
   - The Algorithm
   - Demo

Keysetup Algorithm

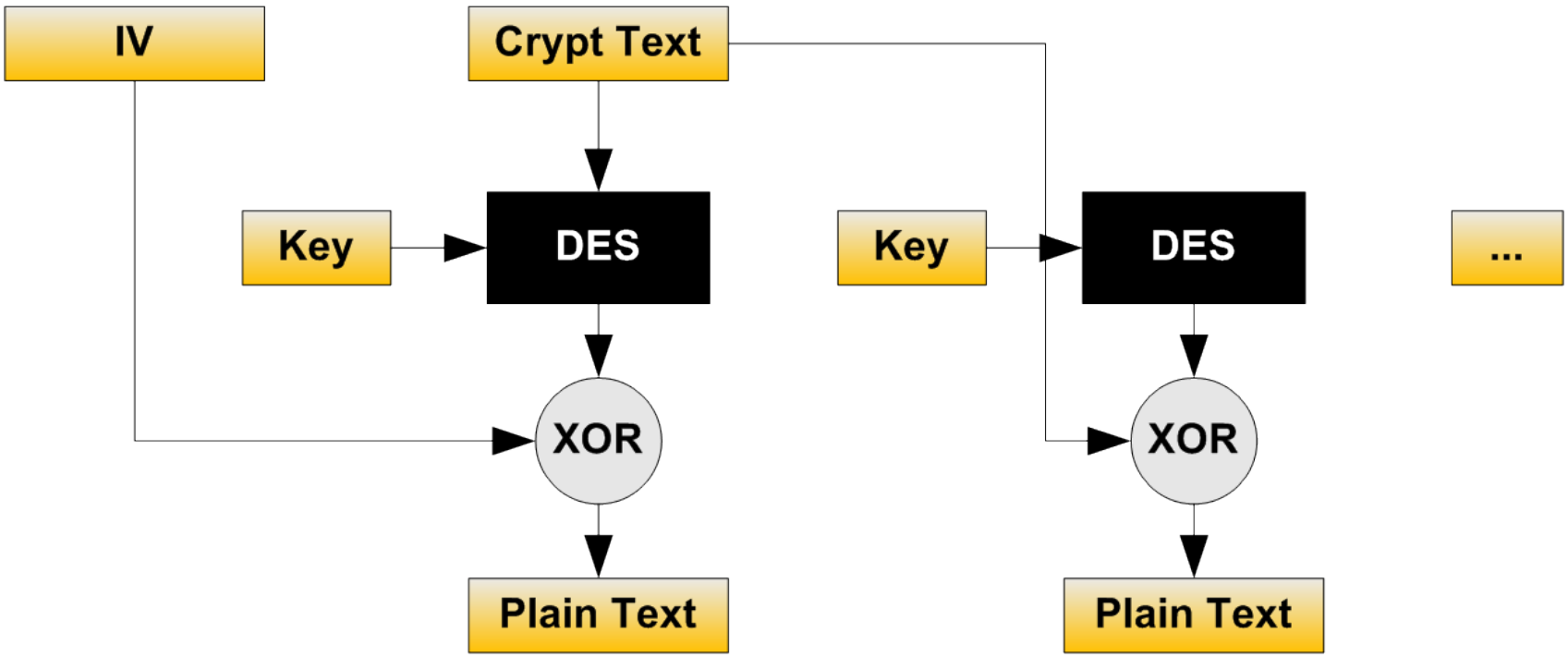
Key Derivation

- Hash some files
- Use different hash algorithms
- Different key for every music file



Decryption Algorithm

- Decrypt content with DES-CBC (Cipher Block Chaining)
- IV from DRM file



Demo

Demo

Conclusion

Summary

- Overall: good protection
- BPMs led us to success, P-Code machine almost useless!
- Implementation weaknesses

Conclusion

Summary

- Overall: good protection
- BPMs led us to success, P-Code machine almost useless!
- Implementation weaknesses

Room for Improvements

1. Transform more native code to P-Code
2. Make P-Code machine more complex (nesting, more obfuscation, ...)
3. Improve (very) weak debugger detection
4. *Use* DRs, let control flow depend on BPM/BPX
5. ...

Thanks for your Attention!

Questions?