

NTrace

Dynamic Function Boundary Tracing for Windows on IA-32

Oct 13, 2009

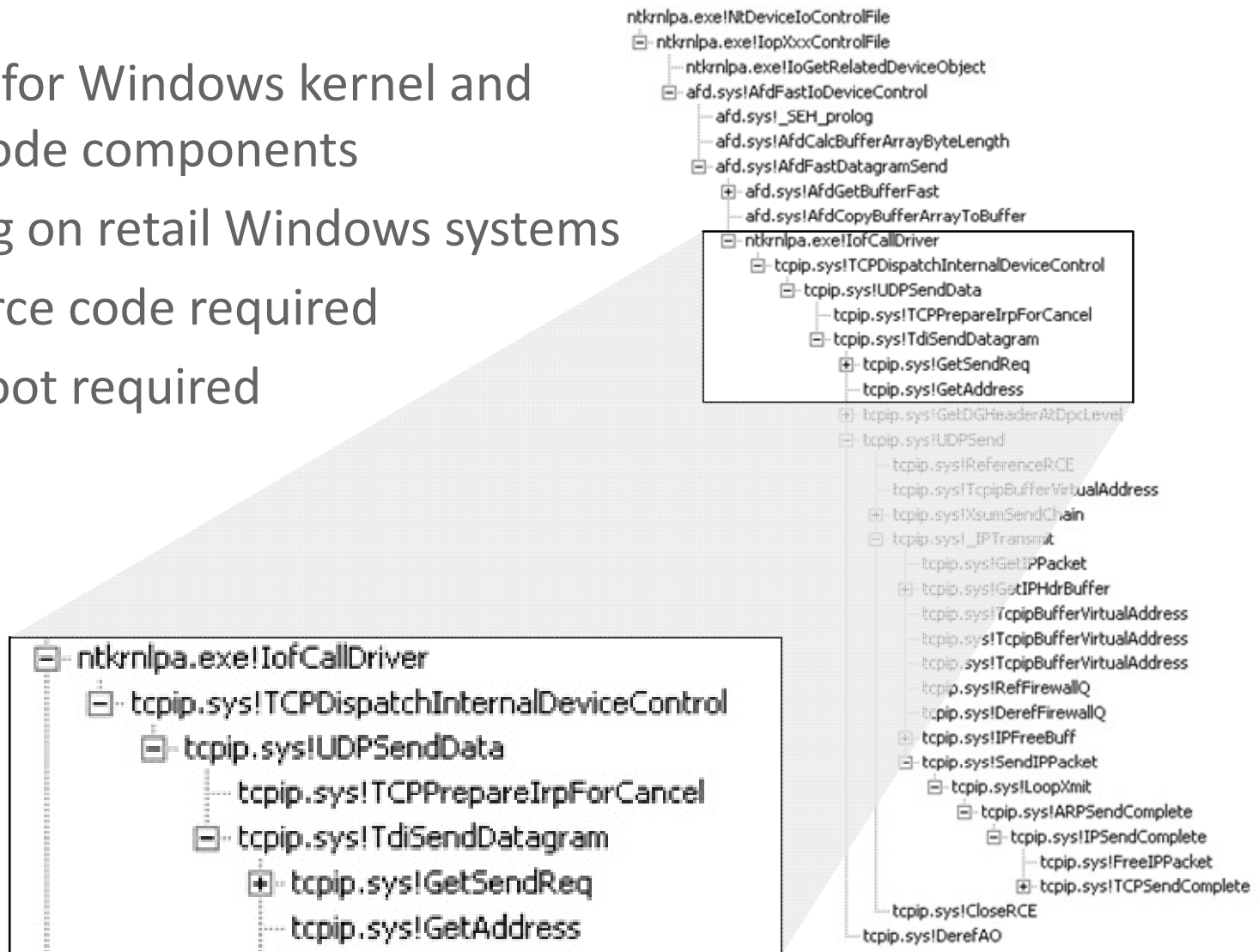
16th Working Conference on Reverse Engineering

Johannes Passing, M.Sc.

Hasso Plattner Institute at U Potsdam, Germany

NTrace at a glance

- Tracing for Windows kernel and user mode components
- Running on retail Windows systems
- No source code required
- No reboot required



Agenda

- **Dynamic Function Boundary Tracing**
 - Overview
 - Implementation Strategies
- NTrace
 - How NTrace captures events
 - Implementation highlights
- Performance Evaluation

Dynamic Function Boundary Tracing

- Function Boundary Tracing
 - Tracing function entry and exit
- Key Benefits of *Dynamic* Function Boundary Tracing
 - No source code and rebuild required
 - ‘On the fly’ instrumentation
- Key areas of application
 - Debugging production systems
 - Foundation for reverse engineering tools

What makes a good dynamic FBT tool?

- Low Runtime Overhead
 - Usage on production systems requires minimal slowdown
 - Influences timing behavior
- High Coverage
 - Percentage of functions which can be instrumented
- Low Risk
 - Intrusiveness, danger of introducing faults

Implementation strategies (1/3)

- Leveraging CPU features
 - E.g. *Branch Trace Storage, Instruction Based Sampling*
 - High coverage, low risk, but...

→ High overhead (too fine grained)

- Modifying the environment
 - E.g. *IAT/SSDT/Vtbl patching, OSR IRP Tracker*
 - Low risk, low overhead, but...

→ Low coverage

Implementation strategies (2/3)

- Dynamic compilation/translation
 - E.g. *Shade, DynamoRIO, Pin, Valgrind*
 - Low overhead, may achieve high coverage, but...

→ High risk (reliance on IA-32 disassembly)
- Injecting trap-generating instructions (e.g. *int 3*)
 - E.g. *DTrace (IA-32), KernInst (IA-32)*
 - High coverage, low risk, but...

→ High overhead (traps are expensive)

Implementation strategies (3/3)

- In-place code modifications
 - E.g. *DTrace (SPARC)*, *KernInst (SPARC)*, *Detours*
 - Potentially high coverage, low risk, low overhead, but...

- not easy to get right on IA-32 (and Windows)

Key challenges of IA-32

- IA-32 uses a variable-length instruction set
 - Disassembly is error-prone
 - Smallest *jmp* occupies 2 bytes, *push ebp* only 1
 - Shifting instruction boundaries is *very* dangerous due to preemption and interruption
- Need to adhere to Intel's strict requirements on runtime code modification
 - Issue serializing instruction (barrier) after code modification
 - Stall other cores/CPU's in an multiprocessor environment

Key challenges of Windows NT

- Kernel is re-entrant and preemptive
- Heavily multi-threaded
- Structured Exception Handling (SEH)
 - Ubiquitous in both user and kernel mode
 - Unwinds are function exits, too!
- Only public debugging symbols available
- No kernel modifications possible

Agenda

- Dynamic Function Boundary Tracing
 - Overview
 - Implementation Strategies
- **NTrace**
 - How NTrace captures events
 - Implementation highlights
- Performance Evaluation

Overview

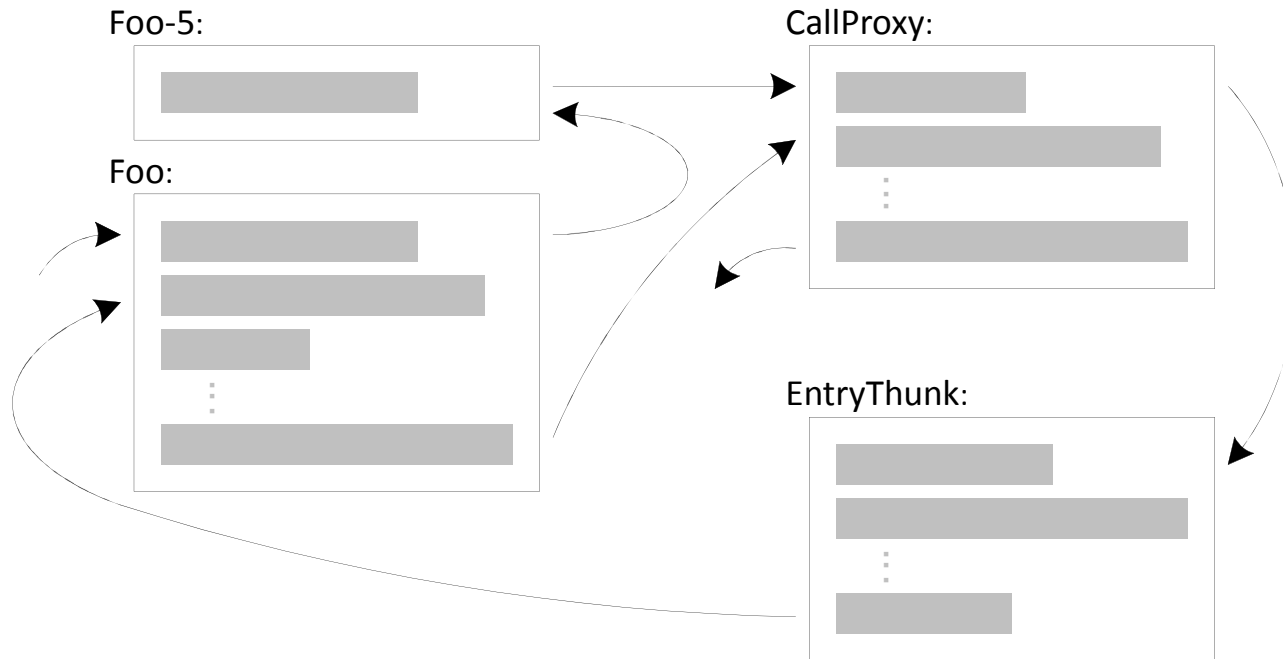
- Idea
 - Leverage properties of hotpatchable images
 - Use in-place code modification to capture function entries
 - Replace return address to capture function exits
 - Integrates with SEH infrastructure to capture unwinds
- Supports both kernel and user mode
- Supports SMP
- Runs on retail Windows versions (Svr03 SP1 onwards)
- Public debugging symbols suffice

Hotpatchable images

- Part of Hotpatching
- Introduced in Svr03 SP1 (DDK 3790.1830)
- Almost all OS images are hotpatchable
- Both user and kernel mode
- Compiler/Linker support: /hotpatch, /functionpadmin

```
...
retn    10
nop
nop
nop
nop
jmp    PatchedNtfs...
NtfsPinMappedData:
jmp  edi, edi
push   ebp
mov    ebp, esp
mov    ecx, [ebp+18h]
mov    edx, [ebp+0ch]
...
```

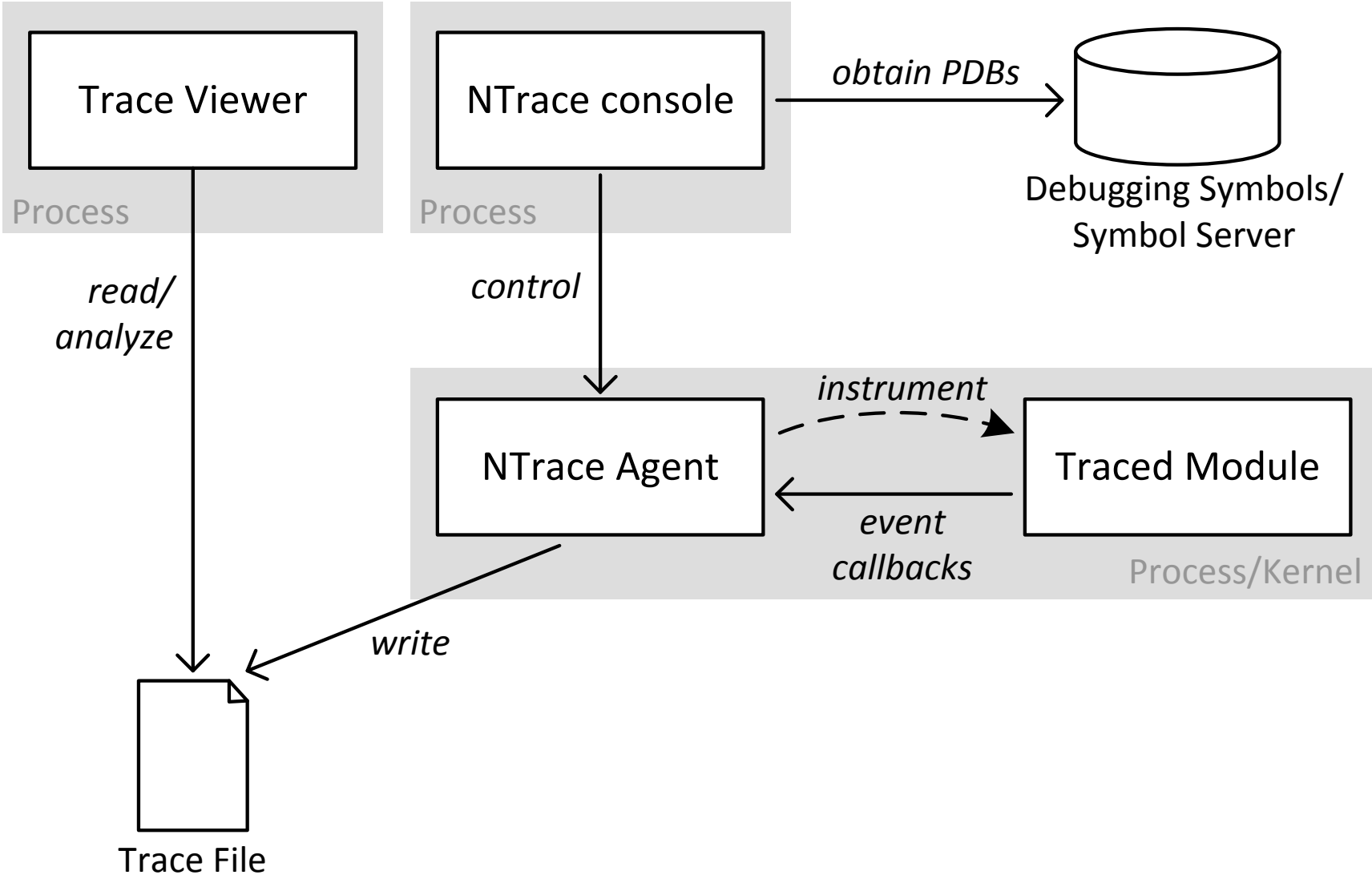
Capturing function entry and exit



SEH: Capturing unwinds

- Structured Exception Handling implements programmatic, stack based exception handling on IA-32
 - Linked list of EXCEPTION_REGISTRATION_RECORDs (ERR)
 - Each *must* be allocated on the stack (thorough validation)
- To be notified about unwinds, a registration is needed
- But: Allocating an ERR is not feasible
- Solution: Modify existing ERR, use combination of two exception handlers to 'emulate' an exception registration

NTrace Architecture



Agenda

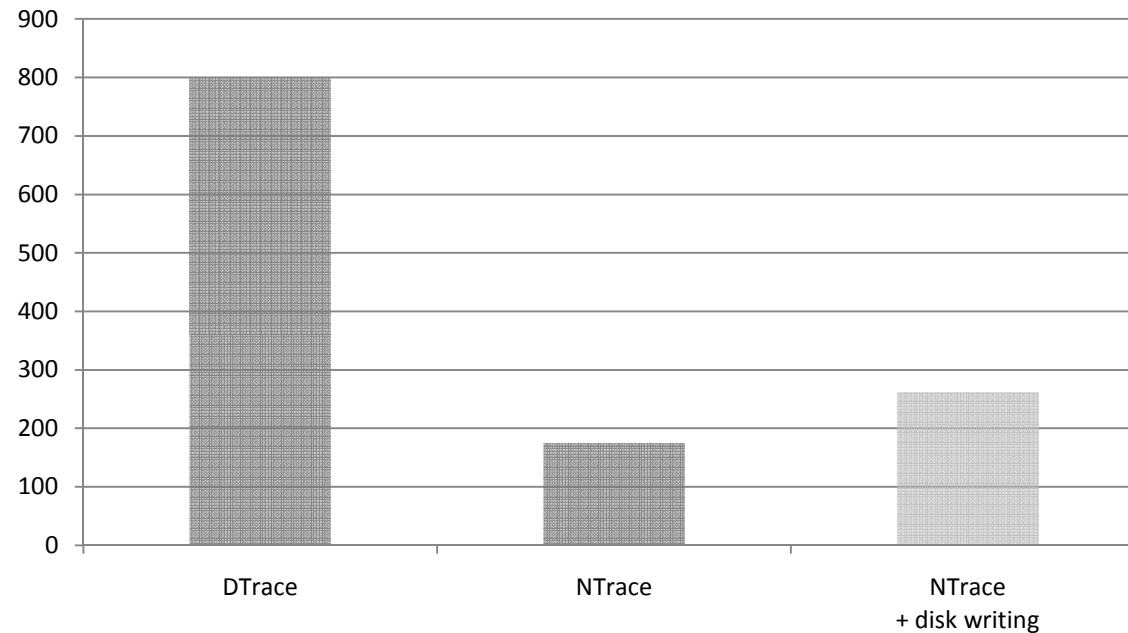
- Dynamic Function Boundary Tracing
 - Overview
 - Implementation Strategies
- NTrace
 - How NTrace captures events
 - Implementation highlights
- **Performance Evaluation**

NTrace vs. DTrace (IA-32)

- Micro benchmark
 - 500 million calls to getpid/NtIsProcessInJob
 - Very simple system call
- IA-32 machine, 4 cores
 - ... running NTrace/Windows Server 2003 SP2
 - ... running DTrace FBT/OpenSolaris 10
- Empty Probe

NTrace vs. DTrace (IA-32): Results

- Absolute slowdown per traced system call in nanoseconds



- NTrace is significantly faster than DTrace FBT (175ns vs. 798ns)
- ...even when writing all data to disk (262ns vs. 798ns)
- ...roughly on par with DTrace SPARC

Conclusion

- NTrace can trace both user and kernel mode Windows NT
- Integrates with Structured Exception Handling
- Overcomes IA-32 and Windows challenges, achieves
 - low runtime overhead
 - low risk
 - decent to high coverage
- Future work: May investigate AMD64 support
- D Script-like KTrace frontend is in the works

Operating Systems and Middleware Group

Hasso-Plattner-Institut

at University of Potsdam, Germany

<http://www.dcl.hpi.uni-potsdam.de/>

More about NTrace on

<http://ntrace.org/>