Win at Reversing

API Tracing and Sandboxing through Inline Hooking

Nick Harbour

Agenda

- Reverse Engineering Primer
- Approaches to Dynamic Analysis
- Inline Hooks
- Advantages Over Other Techniques
- Usages

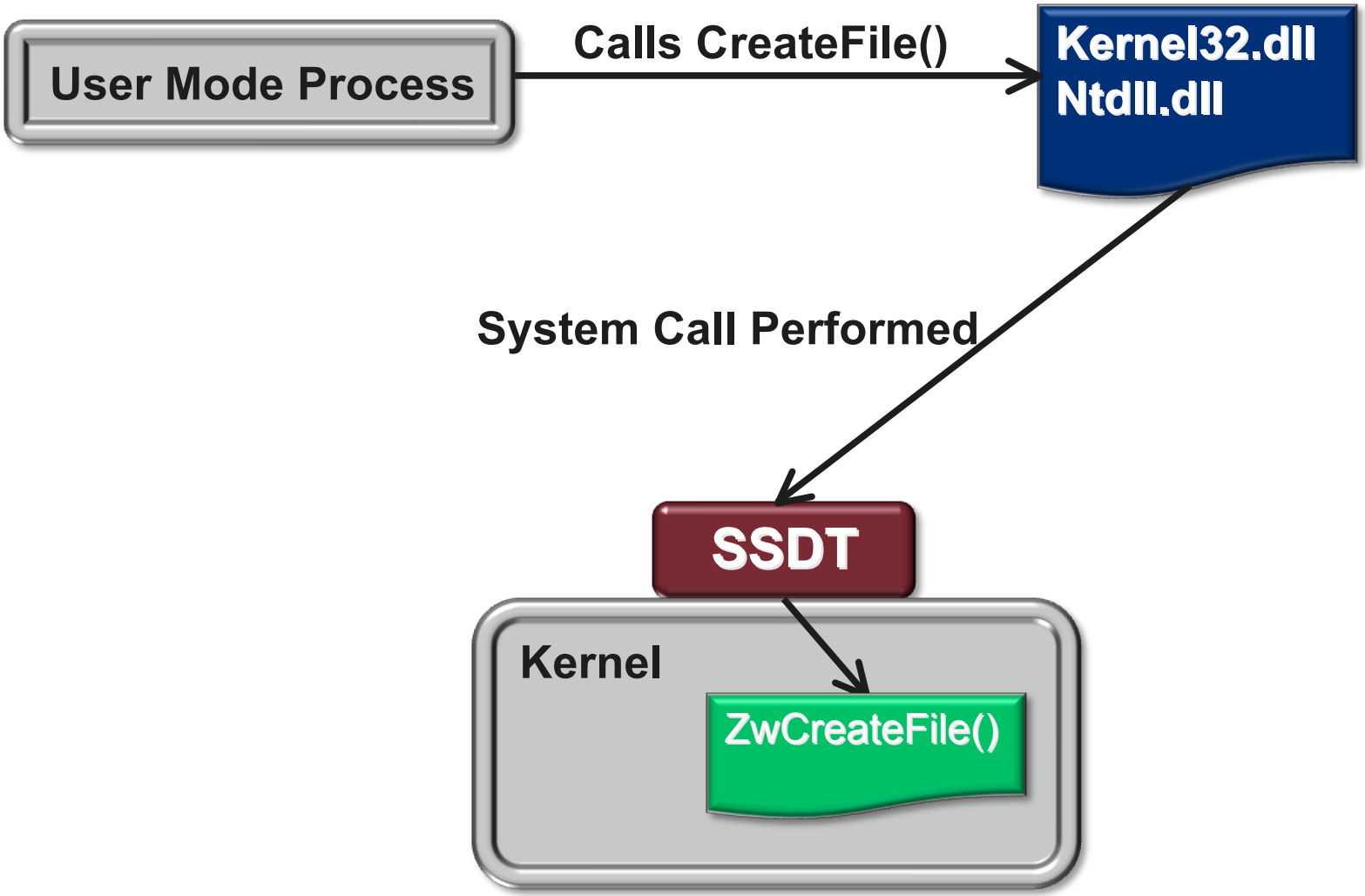
Reverse Engineering Primer

- Reverse Engineering techniques can be devided into two categories: Static and Dynamic Analysis
- Static Analysis
  - Techniques which do not involve running the code
  - Disassembly, file structure analysis, strings, etc.
- Dynamic Analysis
  - Techniques which involve running the code
  - Behavioral analysis

Approaches to Dynamic Analysis

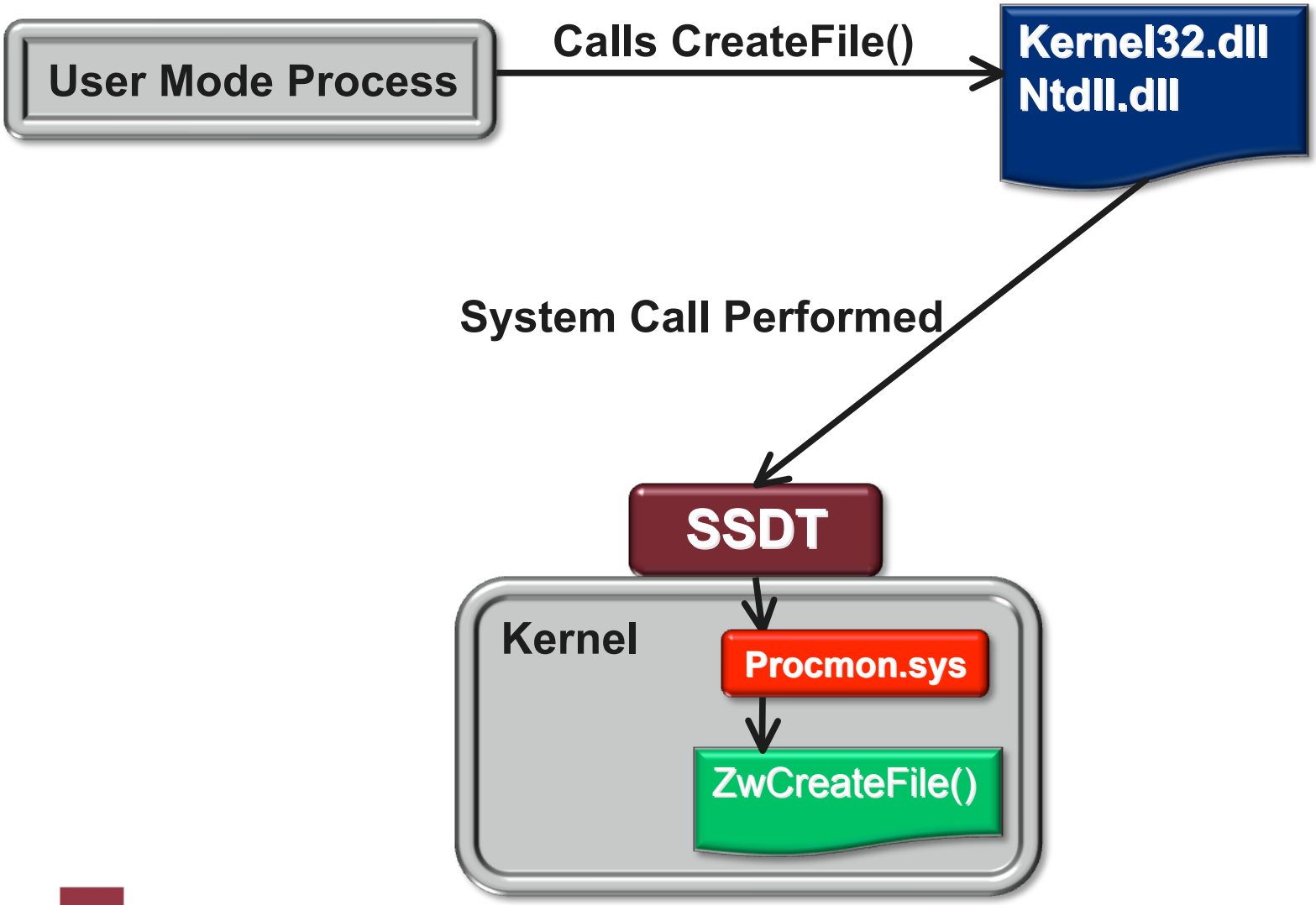
- Network Monitoring
  - Isolated Physical Networks
  - Virtual Networks
- Hardware Emulation
  - Norman Sandbox et al.
- Kernel-Level Monitoring (SSDT hooks)
  - Sysinternals' Process Monitor
- Debuggers

Kernel-Level Monitoring



MANDIANT™

Kernel-Level Monitoring

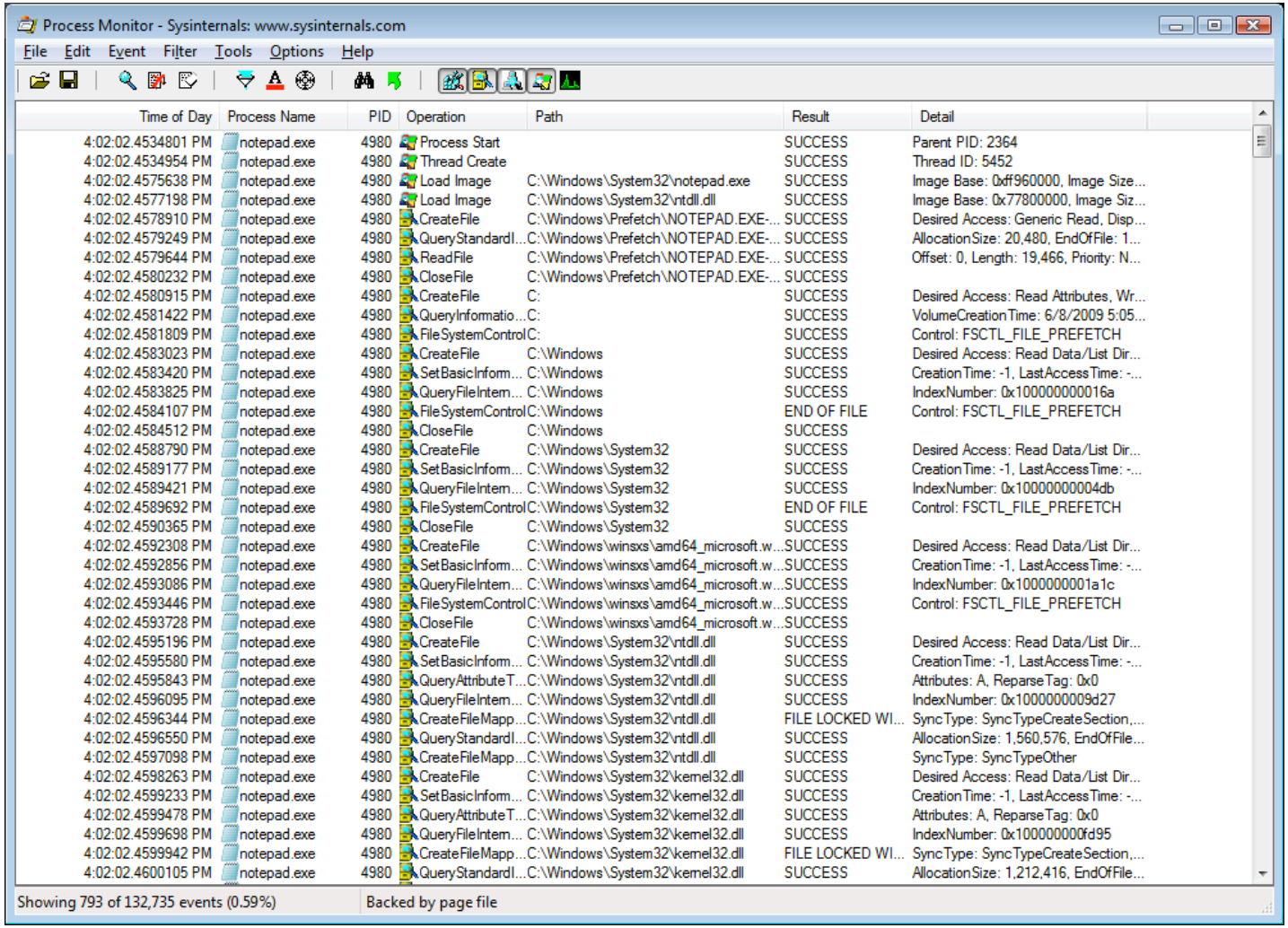


MANDIANT™

Kernel-Level Monitoring

- Advantages
  - Captures every system call
  - Can’t be avoided from userland
- Disadvantages
  - Only captures functions implemented as system calls
  - Not every important function call in the Win32 API is implemented as a system call
  - Tools don’t differentiate between process housekeeping and calls from usercode
  - Calls to internal DLL’s cannot be observed

Process Monitor



MANDIANT™

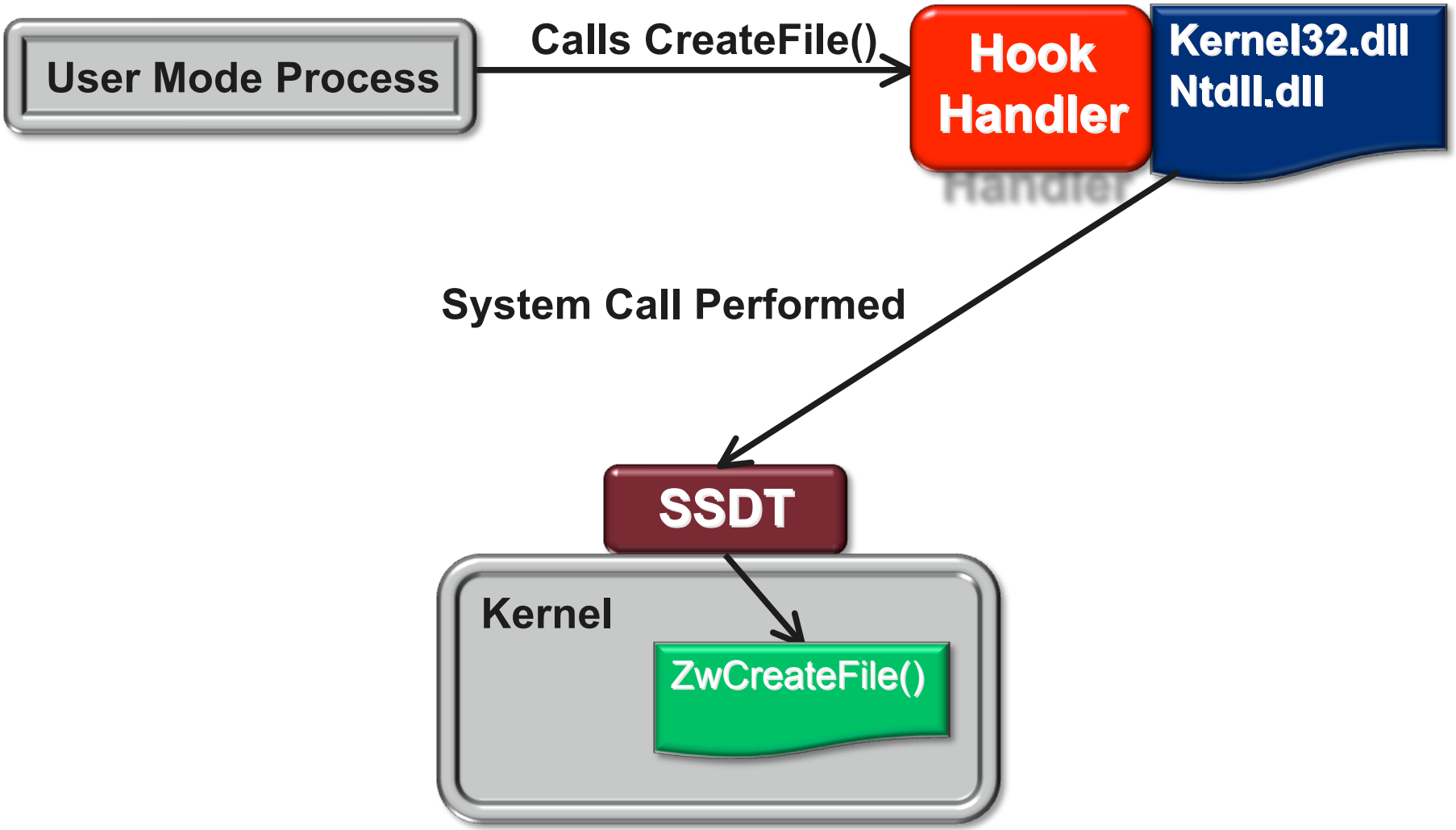
Process Monitoring via Debugging

- Advantages
  - Debugger can trap any function call, not just system calls
  - Trapped calls are more likely to be highly relevant to the program's operation
- Disadvantages
  - Have to act as a debugger
  - Susceptible to countless anti-debugger techniques

Inline Hooks

- Advantages
  - Can trap any function call, not just system calls
  - Trapped calls are more likely to be highly relevant to the program's operation
  - Not operating as a debugger
  - No device driver required
- Disadvantages
  - More of a pain in the #@! to implement

Monitoring with Inline Hooks



MANDIANT™

Implementing Inline Hooks

1. Find a function of interest
2. Disassemble the beginning of the function
3. If possible, overwrite the beginning bytes of the function with a jump or call instruction
4. Implement a handler for the hooked function

Why Disassemble?

- If you attempt to hook every function from a DLL, for example, you might run into a function such as the one below
- Inserting a 5 byte jump or call would write beyond the end of the function. ☹

```
somefunction:

31 C0    xor eax, eax
C3       retn
```

A Successful Hook Install

```
original_function:
55                          push ebp
89 E5                       mov  ebp, esp
81 EC 18 00 00 00           sub  esp, 24
31 C9                       xor  ecx, ecx
…

hooked_function:
E9 E4 7C FF FF              jmp <handler>
18 00 00 00                 ;unused
31 C9                       xor  ecx, ecx
```

MANDIANT™

What to do with hooked functions.

- Observe and Report
  - Collect data about the current function call by gathering data from stack and report to console
  - Execute any instructions overwritten from the hook
  - Jump back to the next instruction in the hooked function
- Intercept and Emulate
  - Perform a specified action *Instead* of calling the intended function

Roll-your-own Sandbox

- Trap gethostbyname() to always return a fixed IP address.
- A pseudo-handle interface to allow fake reads and writes to files and netwok sockets.
  - Trap connect() to connection to a pseudo-socket.
  - CreateFile(), ReadFile(), WriteFile(), MapViewOfFile()…

API Thief

- Launches target process in a suspended state
- Injects a DLL into the process.
- The Injected DLL hooks all Win32 API functions before the target process is resumed
- API Call monitoring can be used simply with a process monitor-style console
- Imbedded python can be used to write custom handlers for specific hooked functions

- Obtain API Thief at www.mandiant.com

API Thief Demonstration

- Basic Process Monitoring
- Basic Interception (gethostbyname)
- Pseudo-Handles demonstration
- Automated Unpacking with API Thief

Questions?

nick.harbour@mandiant.com

nickharbour@gmail.com