
# The beast within – Evading dynamic malware analysis using Microsoft COM

## Black Hat USA 2016

**Ralf Hund**

**Credits: Martin Goll, Emre Güler, Andreas Maaß**

# Outline

- Introduction
  - Dynamic Malware Analysis
  - Microsoft COM & Malware
- Case Studies
  - Self-crafted COM tests
  - Analyzed with various sandboxes
- Dynamic Analysis of COM Malware
  - How do sandboxes work and why is there a problem
- Alternative Approach

# Dynamic Malware Analysis

# The Problem

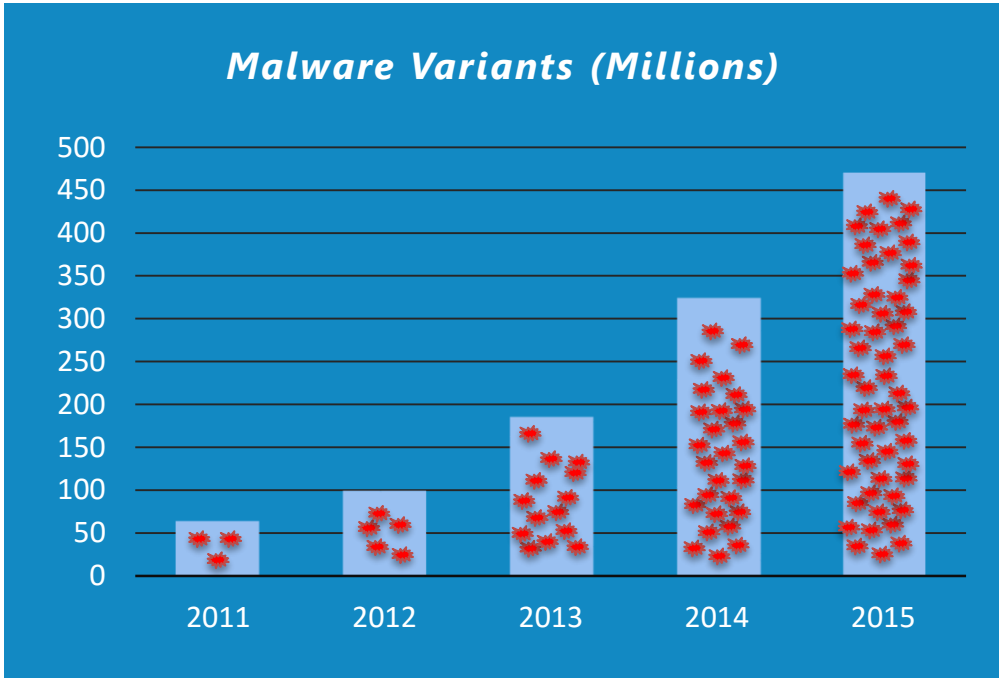
## Cyber Threat Trends

### Exponential Volume Growth

- 2015: >450K new variants / day
- 2015: >150M total

### Increasing Complexity

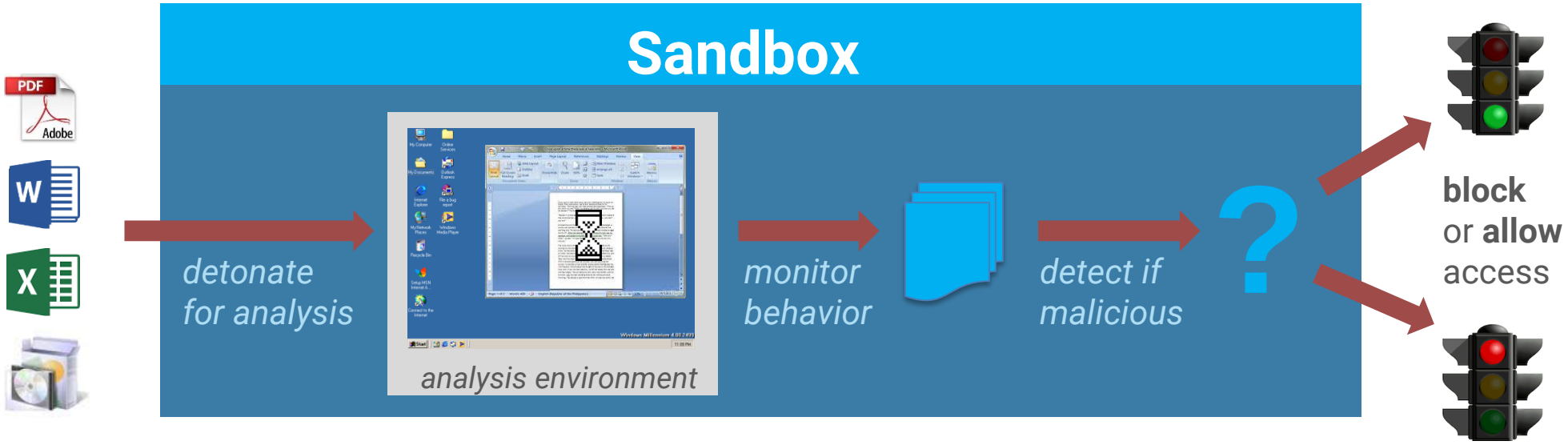
- More evasive malware
- Targeted attacks
- Advanced persistent threats (APT)




*https://www.av-test.org/de/statistiken/malware*

Signature based approaches have shortcomings given quantity and quality of today's malware.
**Dynamic malware analysis** is widely accepted solution to cope with this problem.

# Comprehensive Threat Detection with Sandboxing




*Unknown files and URLs (e.g. Word, PDF, Installer)*
*from arbitrary sources (e.g. Webbrowsing, Email, Download, USB device)*

# Microsoft COM

# Microsoft Component Object Model (COM)

- Binary interface standard for software components


- Standard Win32 API provides procedural „C“ interface
  - Maybe use C++?
  - C++ poses many problems with binary interface


- COM is the solution
  - Provides binary standard C++ lacks
  - Language neutral: Can be used in C++, VB, C#, etc.

- COM objects provide interfaces and methods
  - Example: IWebBrowser2::Navigate

# COM Today

- Still used in many current technologies
  - DirectX
  - Windows Scripting Host (VBScript, JScript, VBA)
  - Microsoft Office
  - PowerShell
  - .NET / WinRT



- Popular interfaces for malware are:
  - Internet Explorer: Download files in background
  - Shell Link: Create, delete, modify, etc. files
  - WBEM (WMI): Query for installed AV products, etc.
  - Firewall Manager: Create firewall exceptions
  - Task Scheduler: Create new Windows tasks

# COM Malware Statistics

- Some statistics from internal sharing programs:
  - ~20 % of all samples use COM interface
  - Mix of executables, MS Office files, etc.
    - Executables ~10 %
    - MS Office files ~90 %

- Tons of COM interfaces exist in Windows
  - Create files
  - Access the registry
  - Download data from remote server
  - …

# Case Studies

# Case Studies Motivation

- Let‘s see how well sandboxes *perform* with COM samples…
- *5* different self-crafted test programs
- Inspired by *typical* malware behavior
  - Persistence
  - C&C communication
  - Evasion
  - …

# COM Test Programs

1. *Autostart*
   - Create autostart entry using *CLSID_ShellLink* interface
2. *Browser*
   - Receives C&C commands using *CLSID_InternetExplorer* interface
3. *Firewall*
   - Disables Windows Firewall using *CLSID_NetFwPolicy2* interface
4. *Filesystem*
   - Copy file to Windows folder using *CLSID_FileOperation* interface
5. *New Process*
   - Create new process using *CLSID_WbemLocator* interface (WMI)

# Case Study Results

- Submitted all of these tests to four different sandboxes
  - Open source sandbox
  - Public version of a commercial sandbox
  - Two non-public commercial sandboxes

## Detection results

worst case

| | #1 Autostart | #2 Browser | #3 Firewall | #4 Filesystem | #5 New Process |
|---|---|---|---|---|---|
| SB #1 | ✗ | ✗ | ✗ | ✗ | ✗ |
| SB #2 | ✓ | ! | ✗ | ✗ | ✗ |
| SB #3 | ✓ | ! | ✗ | ! | ✗ |
| SB #4 | ✓ | ! | ✓ | ✓ | ✓ |

# Observation: Noise

- Sandboxes that detect *something* also log a *noise*

- SB #2
  - Wrong IOCs (host names, files, etc.)

- SB #3
  - False alerts: Anti-reverse engineering, suspicious imports, …

- SB #4
  - Report contains 136 events (files, process, hosts, etc.)
  - 32 are *actually* test behavior ➔ almost *80%* is noise
  - „Opens TCP port“, „code injection“, „tampers with explorer“, …

# Dynamic Analysis of COM Malware

# Excursion: Classic Sandbox Classification

- Approach #1: **Hooking** based
  - Install hooks at various memory locations
  - Quite fast, close to native performance
  - Can be detected/evaded

- Approach #2: **Emulation** based
  - Executes malware in full system emulator
  - Can theoretically see every machine instruction executed
  - Very slow (a lot of overhead only for CPU emulation)

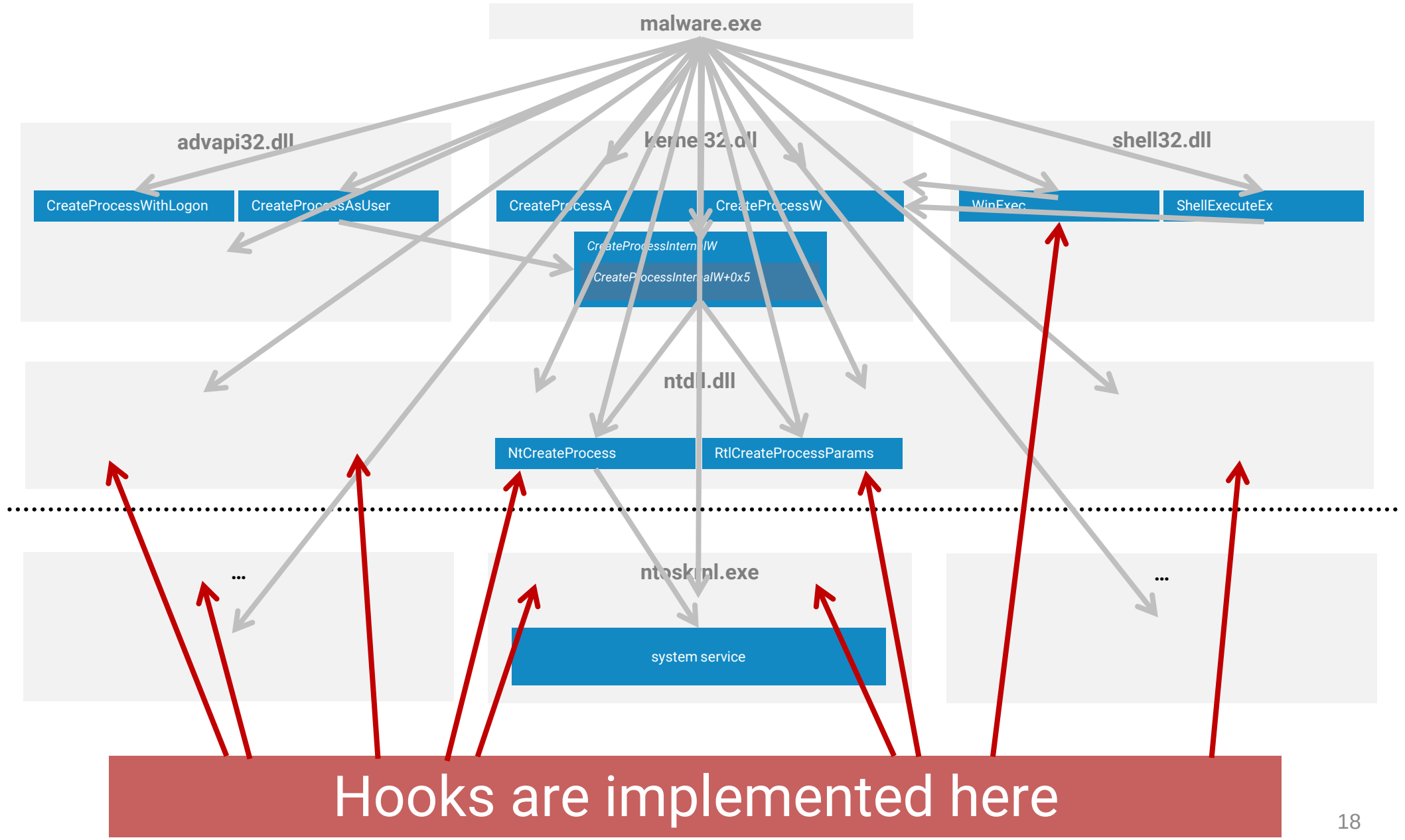
- Approach #3: **Transition** based
  - See later …

# Design Goals

1. ***No evasion***: All behavior must be reported
2. ***No noise***: Reports must not be inflated with noise
3. ***Stealthiness***: Do not leave (a big) footprint in the system
4. ***Stability***: Do not crash due to buggy hooks
5. ***Performance***: Do not slow down the system too much

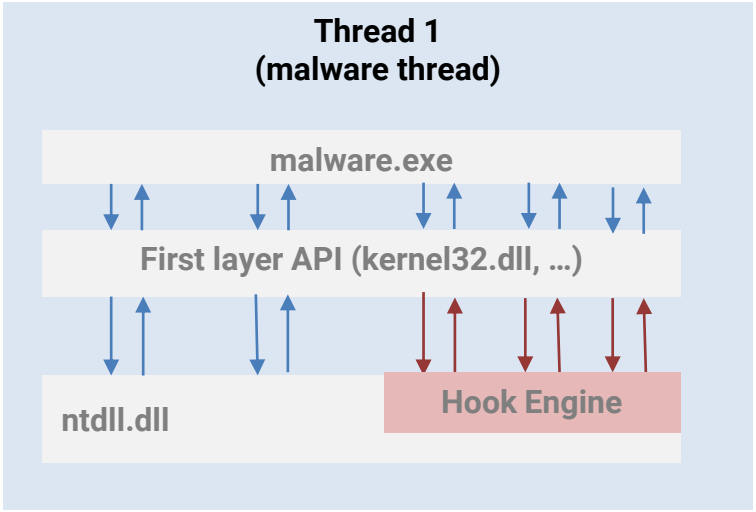
Goals 3, 4 & 5 can only be achieved by *limiting* the amount of hooks

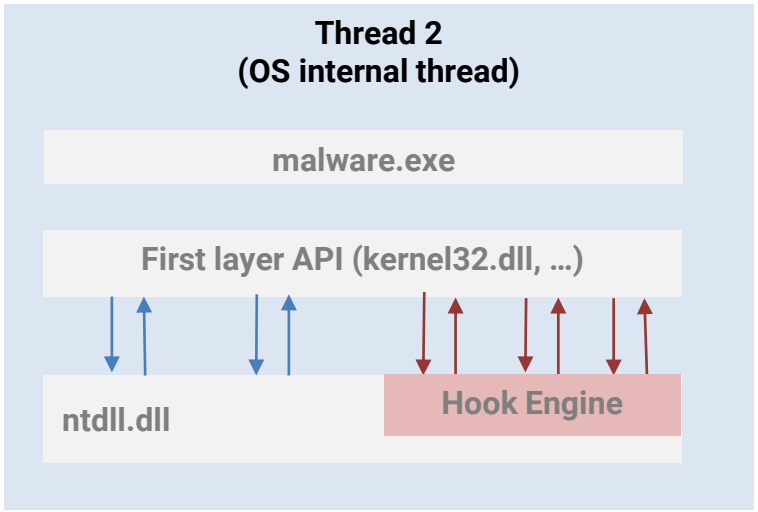
# Challenge #1: Where to Place Hooks?



# Challenge #2: Handling Noise

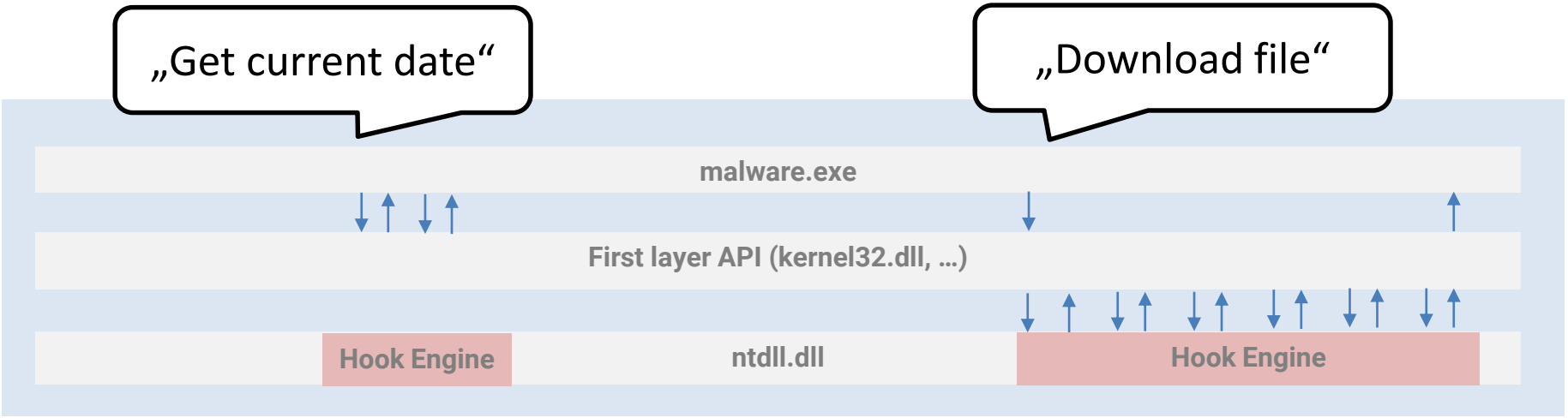
- Must *filter* out irrelevant hooked calls
- OS and apps generate *unrelated* calls as *side-effect*




- Is hooked call *relevant* or not?
- Image you hook inside Internet Explorer, MS Word, ...
- Not easy to solve …

# Challenge #3: Limited Visibility

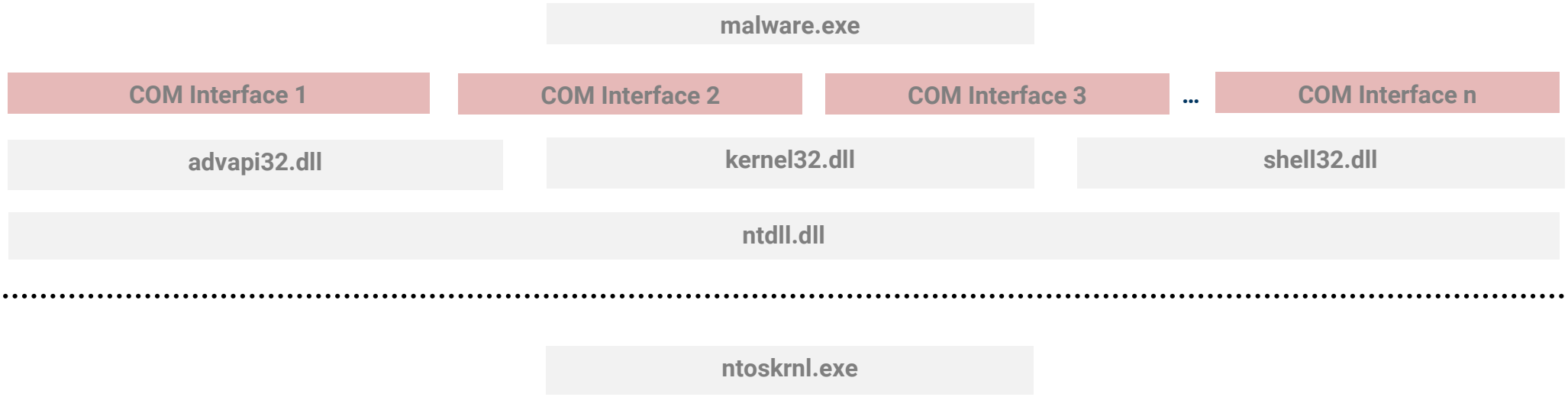



**See too little: Calls do not end in NTDLL**

**See too much (avalanche effect)**

# COM Issue #1: Additional API Layer

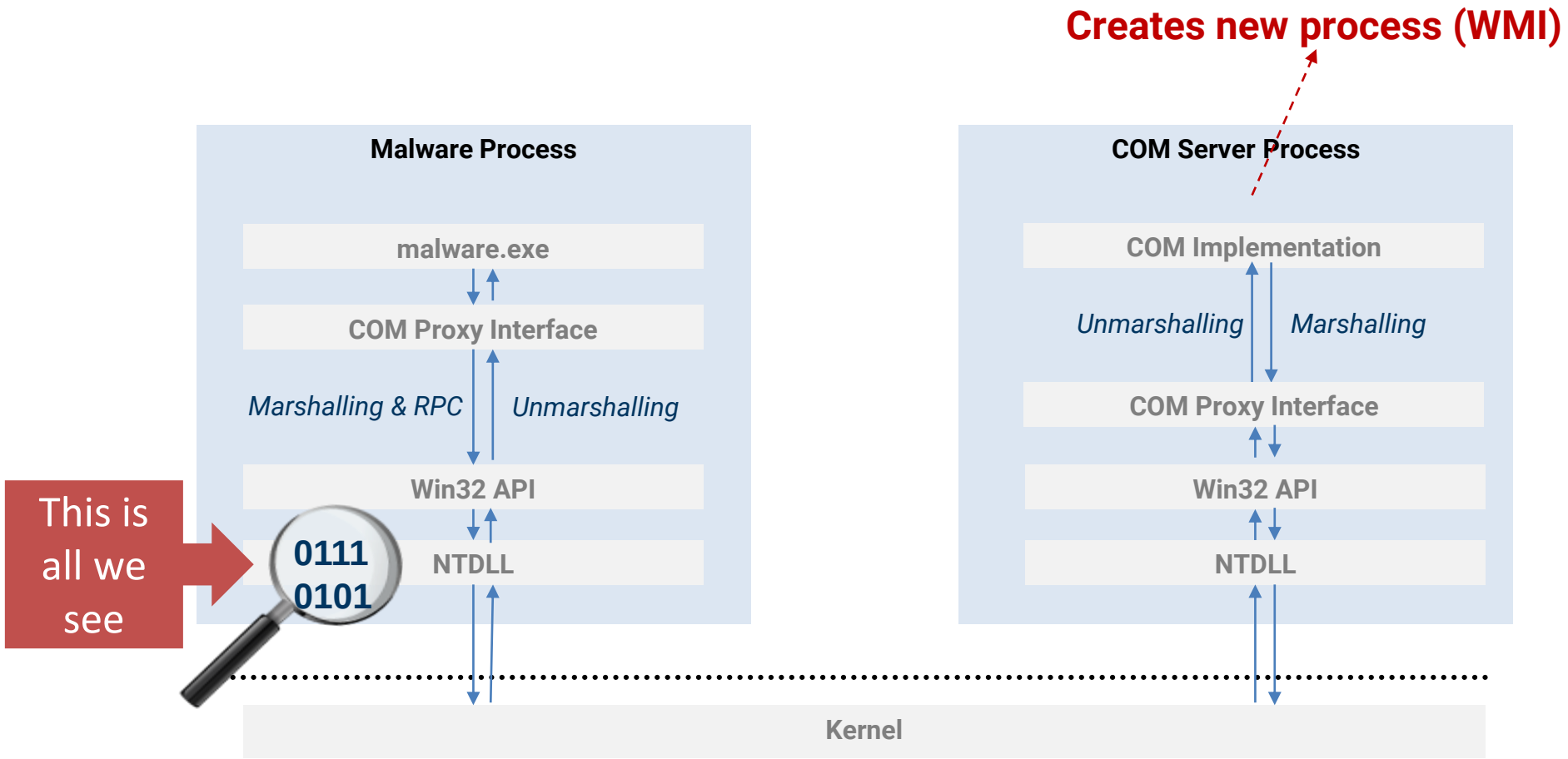
COM provides yet another (inflated) API layer




1. Must filter out even *more* noise
2. Even *more* calls go unnoticed
3. Avalanche effect even *worse*

# COM Issue #2: RPC

- COM supports *remote procedure calls* (RPC)
- Method calls are executed in another process



# RPC Madness

- Only *marshalled* data seen at NTDLL layer
  - Which *method* is executed?
  - What are the *parameters*?

- Interpretation needs *internal* knowledge of COM runtime
  - Mostly *non-documented* information
  - Lots of *reversing* necessary
  - Microsoft is free to adjust and/or change runtime at any time

- Let‘s just monitor *COM server processes* then
  - How to *filter* out COM server process noise?
  - How to *filter* out COM calls from irrelevant processes?

# Summary

- Don‘t want sandbox to be *evaded* with one COM call
- Don‘t want sandbox which cannot be evaded but contains tons of *noise*
- Remember noise in SB #4?
  - „Opens TCP port“ ➔ This is the Internet Explorer COM process
  - „Code injection“➔ This is COM runtime doing RPC
  - „Tampers explorer“➔ This is the *CLSID_FileOperation* interface

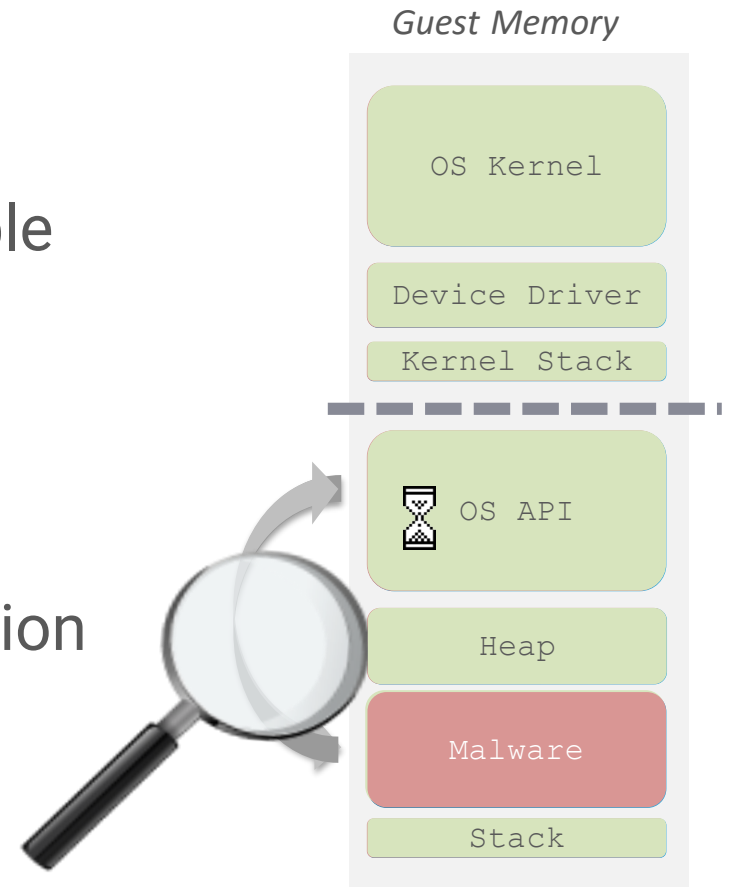
# Alternative Approach

# Intermodular Transition Monitoring (ITM)

1. Use VT MMU to partition memory
   - Current module: X executable
   - Remaining memory: NX non-executable

2. Run malware in VM
   - With bare metal performance
   - Interrupts only on intermodular transition

3. Monitor is automatically invoked
   - Read guest memory
   - Readjust partitioning
   - Continue execution
   - Until return to calling malware




**IWebBrowser2:Navigate (**
url=„https://www.vmray.com“,
Flags=0x123,
TargetFrameName=„_blank“,
PostData=NULL,
Headers=„…“)

# Challenges

- Need to parse a lot of information
  - Interface and method names
  - Parameters: Integers, strings, variants, byref, byvalue, …
- „Dynamic“ binding of COM interfaces
  - Many different variations exist (*QueryInterface*, *Invoke*, …)
- Need to understand what each COM method does
- Lots of work but at least it‘s public and documented!

# Summary

ITM fixes all disadvantages mentioned previously:

1. No noise filtering necessary
2. No missing first layer calls
3. No avalanche effect
4. No need for special handling of RPCs

**Thank you for your attention!**

**Happy to answer any questions!**