Hypervisor-Based, Hardware-Assisted System Monitoring

VB2013
October 2-4, 2013 – Berlin

Carsten Willems, Ralf Hund, Thorsten Holz
Ruhr-University Bochum, Horst Görtz Institute for IT-Security
VMRay GmbH

Existenzgründungen aus der Wissenschaft

EUROPÄISCHE UNION

Europäischer Sozialfonds für Deutschland

Gefördert durch:
Bundesministerium für Wirtschaft und Technologie
aufgrund eines Beschlusses des Deutschen Bundestages

Malware Analysis Methods

- Static analysis
  - Complete results, but time-consuming & complicated
  - Countermeasures: obfuscation, encryption, vmprotect, …
- Dynamic analysis
  - *Execute* sample to get register & memory values
  - Speeds up analysis, but only *one* execution path
  - Side-Effect: automatic unpacking, deobfuscation, …
  - Countermeasures: anti-debug/emulate/dump/hook/…
- **Behavior analysis**
  - Automated dynamic analysis
  - Only monitor interaction between sample ⇔ system

Analysis System Requirements

- Need to cope with sophisticated malware today
  - Kernel rootkits, targeted attacks, APT, …
- Need better behavior analysis systems, which provide:
  - *Transparency*
  - Isolation
  - Soundness
  - Monitoring Granularity
  - Performance
  - OS independence

Transparency

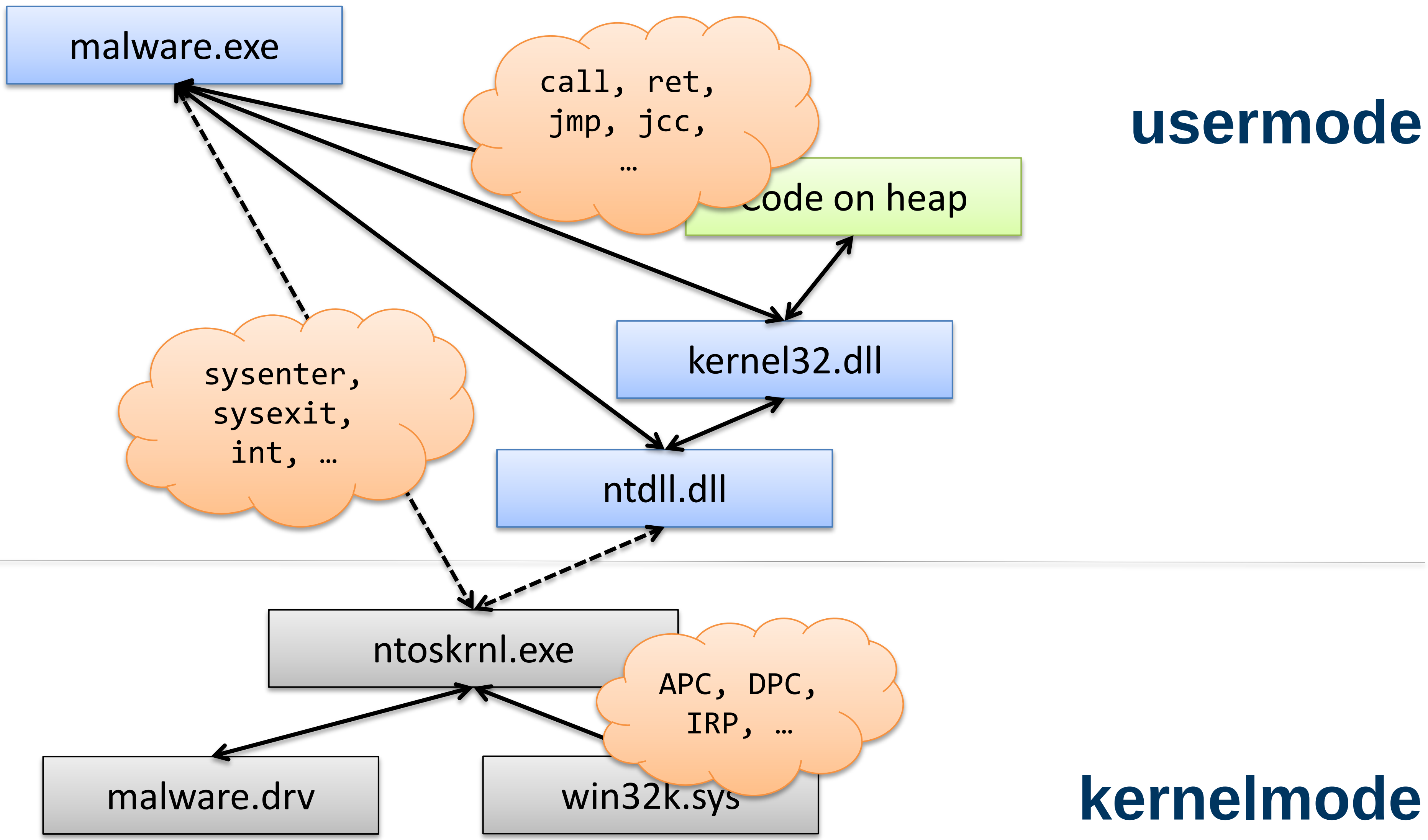
- Are emulators the solution?
  - Big performance overhead
  - CISC architecture hard/impossible to emulate
  - Easy to *detect*
- Even worse: emulators can be fooled
  - instruction sequence with different *semantics* in emulator ⇔ native machine
  - Code inherently acts *benign* in emulator and *malicious* on native machine
  - No compare or conditional jump!

➔ Run analysis on *native* hardware

Design

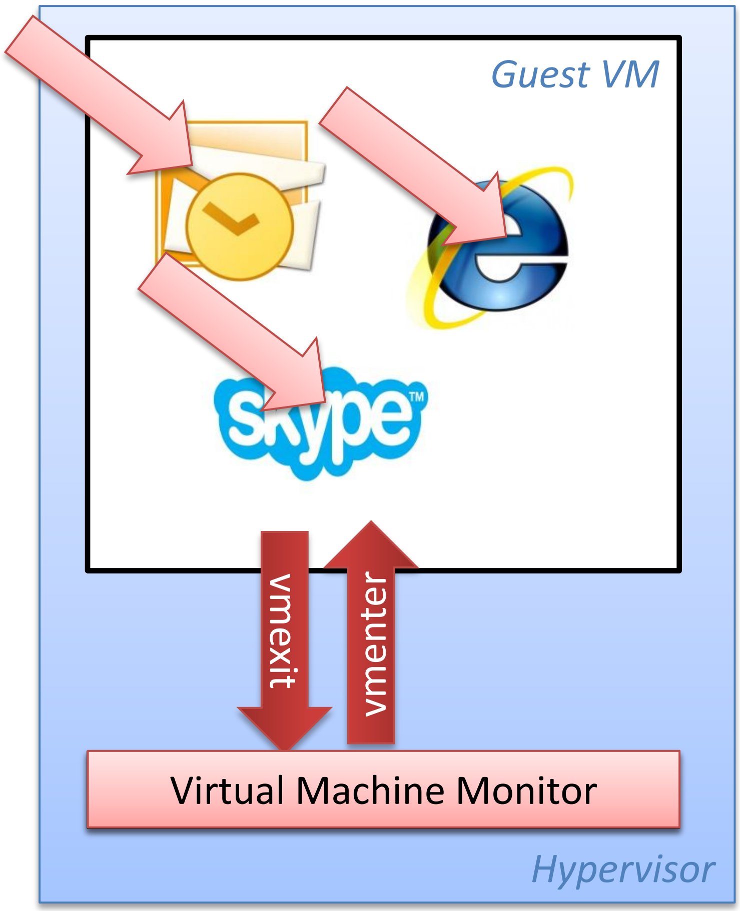
Virtualization-Based Behavior Analysis

Monitoring Module Transitions



vmexit & vmenter

- Most time guest VM runs independent from hypervisor



- Certain events cause `vmexit`
  - Direct hardware access, external Interrupts, critical faults, certain privileged instructions, …
- After handling the situation
  - Hypervisor calls `vmenter`
  - Guest VM remains execution
  - Hypervisor becomes inactive until next vmexit

Instrumenting the Hypervisor

- Hypervisor not designed for program analysis
  - ➔ how to instrument to control & monitor guest VM ?
  - ➔ how to enforce `vmexit` on ***interesting*** operations ?

- Possible methods
  - Single Stepping ➔ very slow
  - Binary Instrumentation ➔ detectable
  - PTE Instrumentation ➔ detectable
  - Invalid configuration ➔ detectable
    - e.g. invalid syscall/interrupt/context
  - **Two Dimensional Paging (TDP)**

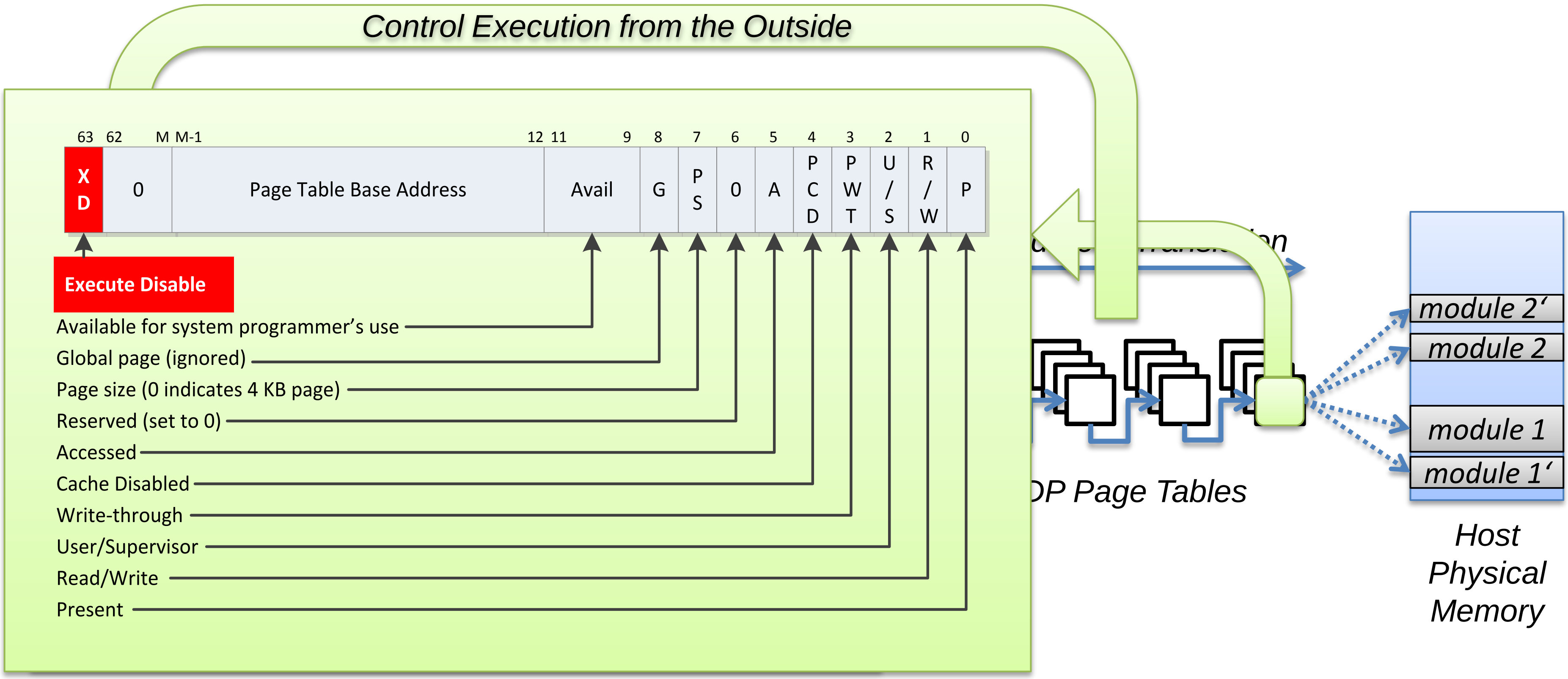
Technical Background

Two-Dimensional Paging

Guest / Host Memory Isolation

- VMMs need to ensure memory isolation/containment
  - Protect host memory from guest
  - Protect guest memory from other guests
- In the past: *Shadow Page Tables (SPT)*
  - Intercept guest accesses to page tables & CR3
  - Slow, but transparent to guest
- Today: *Two Dimensional Paging (TDP)*
  - Intel: Extended Page Tables (EPT)
  - AMD: Nested Page Tables (NPT)

Two Dimensional Paging (TDP)

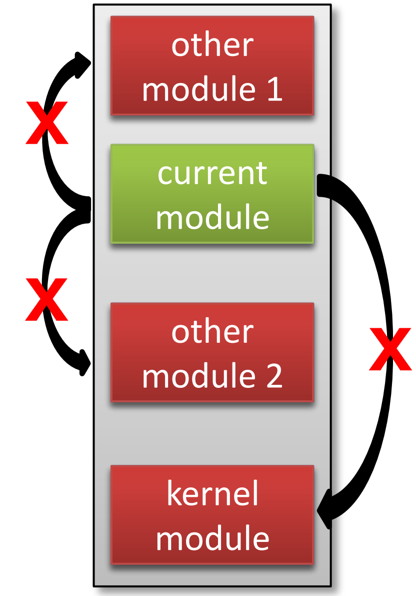


TDP adds **one additional** address translation layer:

Guest Physical Memory ➔ Host Physical Memory

TDP to Monitor Module Transitions

- Modify TDP paging structures
  - Memory of current module = executable
  - Remaining memory = non-executable
- To intercept transitions between modules
  - Function/system calls and returns
  - Obtain function name and parameters
- Completely **transparent** to guest VM
  - Only datastructures of hypervisor are touched
  - Nothing inside the guest is changed

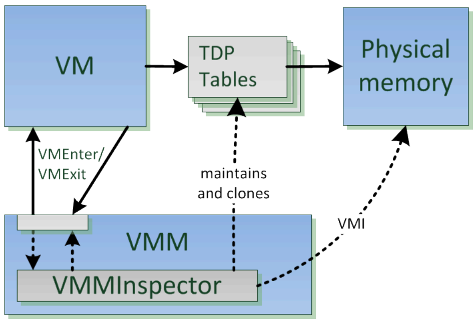


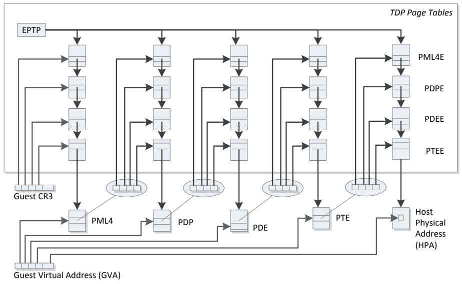
Virtual Memory

Prototype Description

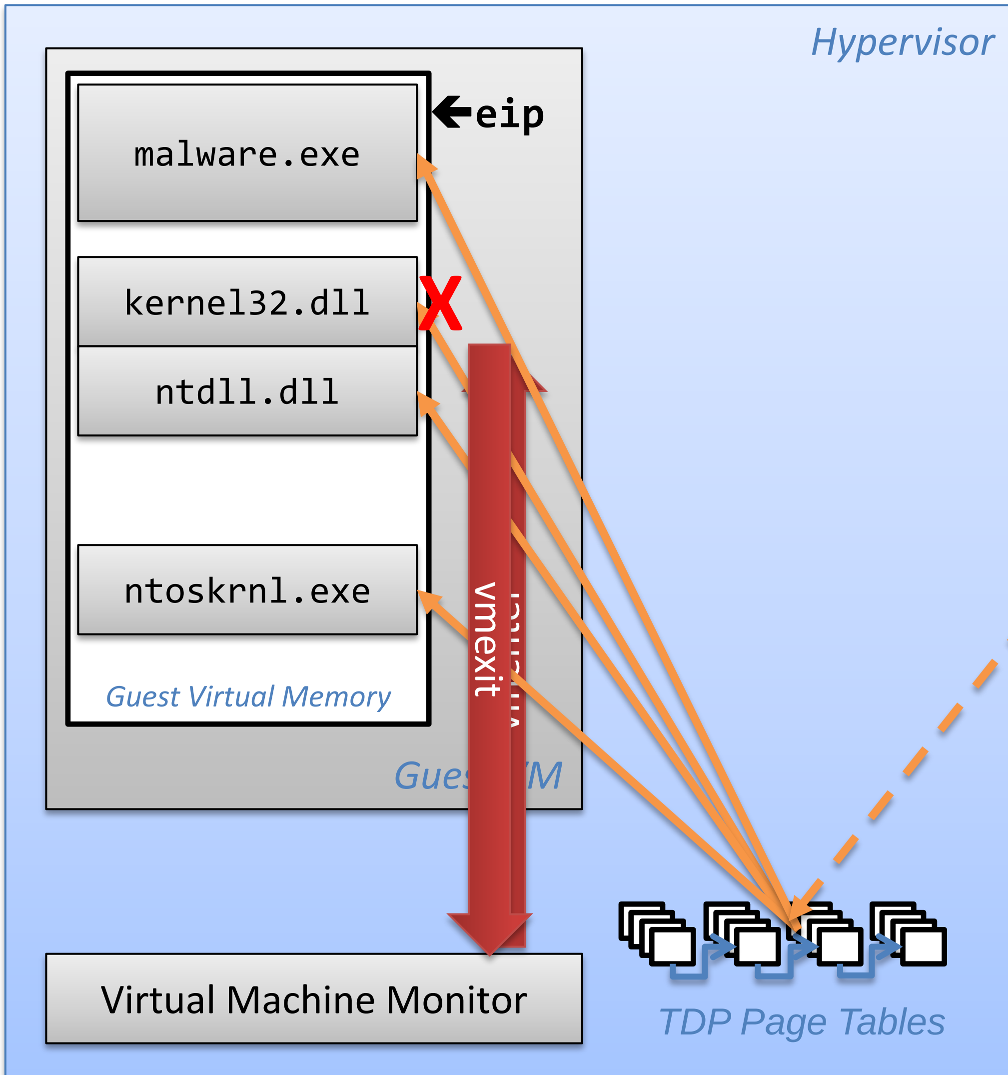
CXPInspector

- CXPInspector (academic prototype)
  - Host: Based on KVM
  - Guest: Windows 7, 64-Bit version
    - Also support 32-bit processes
    - Minimum effort to use Windows 8
  - Performance overhead 1.5x - 5x
    - Depends on configuration
    - Improvable by new VT instructions
- Main characteristics
  - Hardware VT + TDP extension
  - Monitoring module transitions





Operation Overview



on vmexit:

- check what function is called
- use prototype information to get function parameters from stack
- change guest X/NX settings
  - NX for old module
  - X for new module
- perform vmenter
- continues operation in new module
  - until next module transition

execution starts
no interception as
long as eip stays
vmexit when
other module is
called

Feature Summary

- Monitor usermode & kernelmode code
  - Principle is the identical
- Monitor 32 & 64-bit processes
- Monitor operating system
  - Kernel / Driver routines
- Monitor function / API / system calls
  - All Windows API functions detected

Evaluation

- TDSS/TDL4
  - 64bit kernel rootkit for Windows 7
  - Modifies MBR to be loaded before OS
  - Disables Patchguard
  - Disables kernel debugger (by replacing kdcom.dll)
  - Maintains its own encrypted filesystem
  - Installs hooks on the kernel to
    - hide and protect MBR and hidden filesystem
    - inject code into new processes / loaded images

Code Injection

Horst Görtz Institut für IT-Sicherheit

VMRAY

```
from=fffff800026216e9   to=fffff800027d51d  :EtwTraceContextSwap+92
from=fffff800026da15f   to=fffffa8
from=ffffa8             fffff8               penProcess+0
from=
from                                         ccess=0x1fffff
fr
fr
fr                       tring=0x            b900, SourceString="\\device\\
from                     \{752492e1-ed26-91d2-750b-04be2c7925eb" )
from=f                _snprintf(
from=ffffa8            str=0xfffff88002fab050, size=0x00000103 (259), format="%.*S" )
from=fffff80
from=ffffa8           _snprintf returns 0x0000002d (45) (
from=fffff80           str="{752492e1-ed26-91d2-750b-04be2c7925eb}\\phdata" )
from=ffffa8
from=fffff800026e2acb   to=ffffa800
from=ffffa8002949158    to=fffff800              llocateIrp+0
from=fffff800026dc115   to=ffffa8
from=ffffa800294917a    to=ffff                  ocateMdl+0
from=fffff800026dcc9d   to=ffff
from=ffffa8002949191    to=ffff                  MmProbeAndLockPages+0
from=fffff800026ed388   to=ffffa800
from=ffffa80029491a0    to=fffff800026d8be0      skrnl.exe:KeInitializeEvent+0
from=fffff800026d8bfc   to=ffffa80029491a6      ???:+1a6
```

kernel function called on process creation

generate filename of hidden configuration file

filesystem device

get object of hidden filesystem device

Discussion

- Summary
  - Virtualization-based malware analysis
  - Monitor module transitions
  - Utilize Two-Dimensional Paging
  - Can analyze user- and kernelmode code
  - No changes to analysis system
- Future
  - Commercial product „VMRay“
  - Currently rewriting prototype
  - Available mid 2014