Mixed-Mode Malware and Its Analysis

Shabnam Aboughadareh[1], Christoph Csallner[1], Mehdi Azarmi[2]
[1]University of Texas at Arlington
[2]Purdue University

PPREW-4
New Orleans, Louisiana

UNIVERSITY OF
TEXAS
ARLINGTON

PURDUE
UNIVERSITY

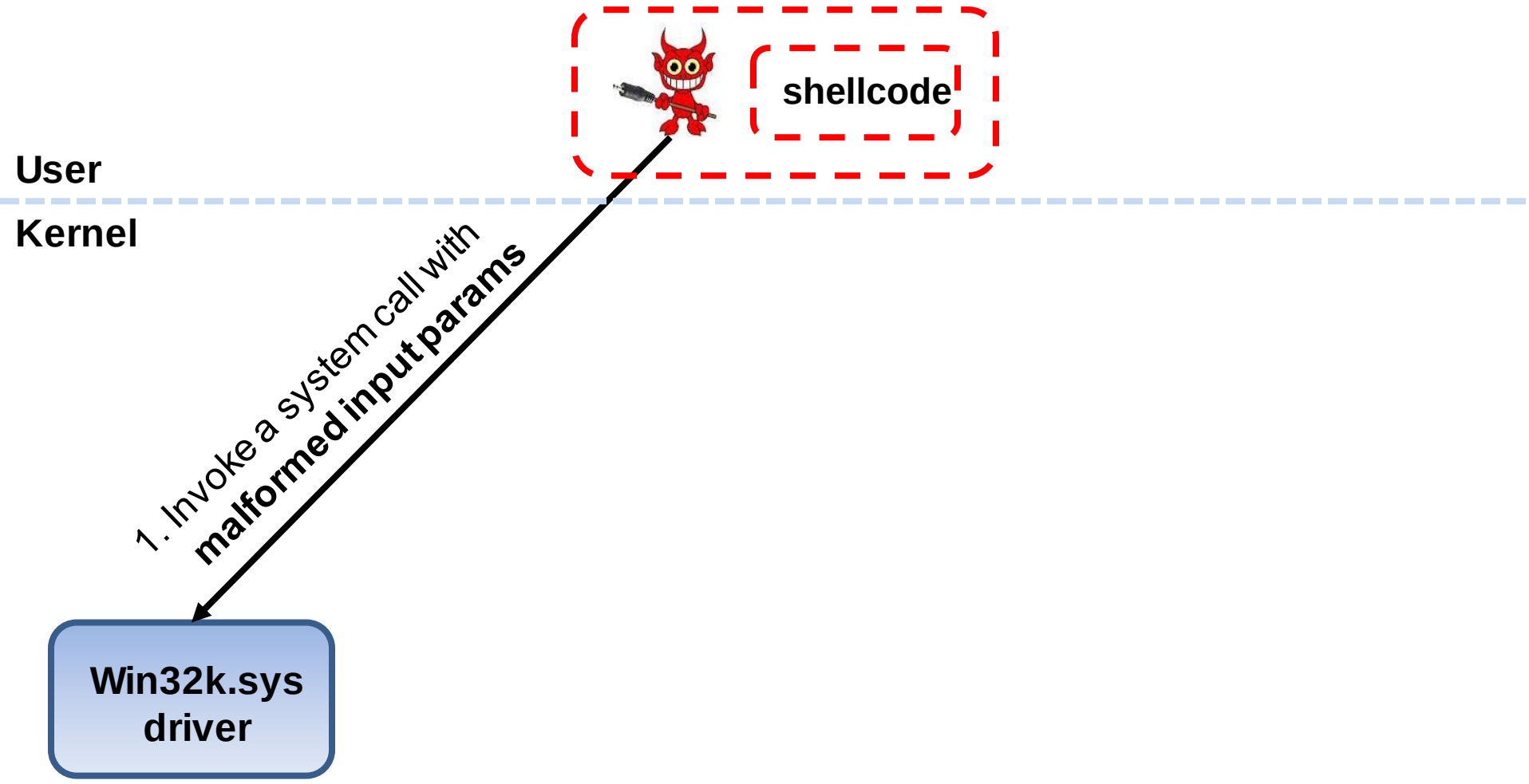
Mixed-Mode Malware

- Performs both user and kernel level actions
  - One familiar example: Kernel/device driver exploits
    - User-mode malware exploits a kernel vulnerability to execute a shellcode with kernel privilege
- User and kernel actions are **inter-dependant**
  - Example: malware adjusts its user-mode actions based on kernel data modifications done by kernel-mode actions

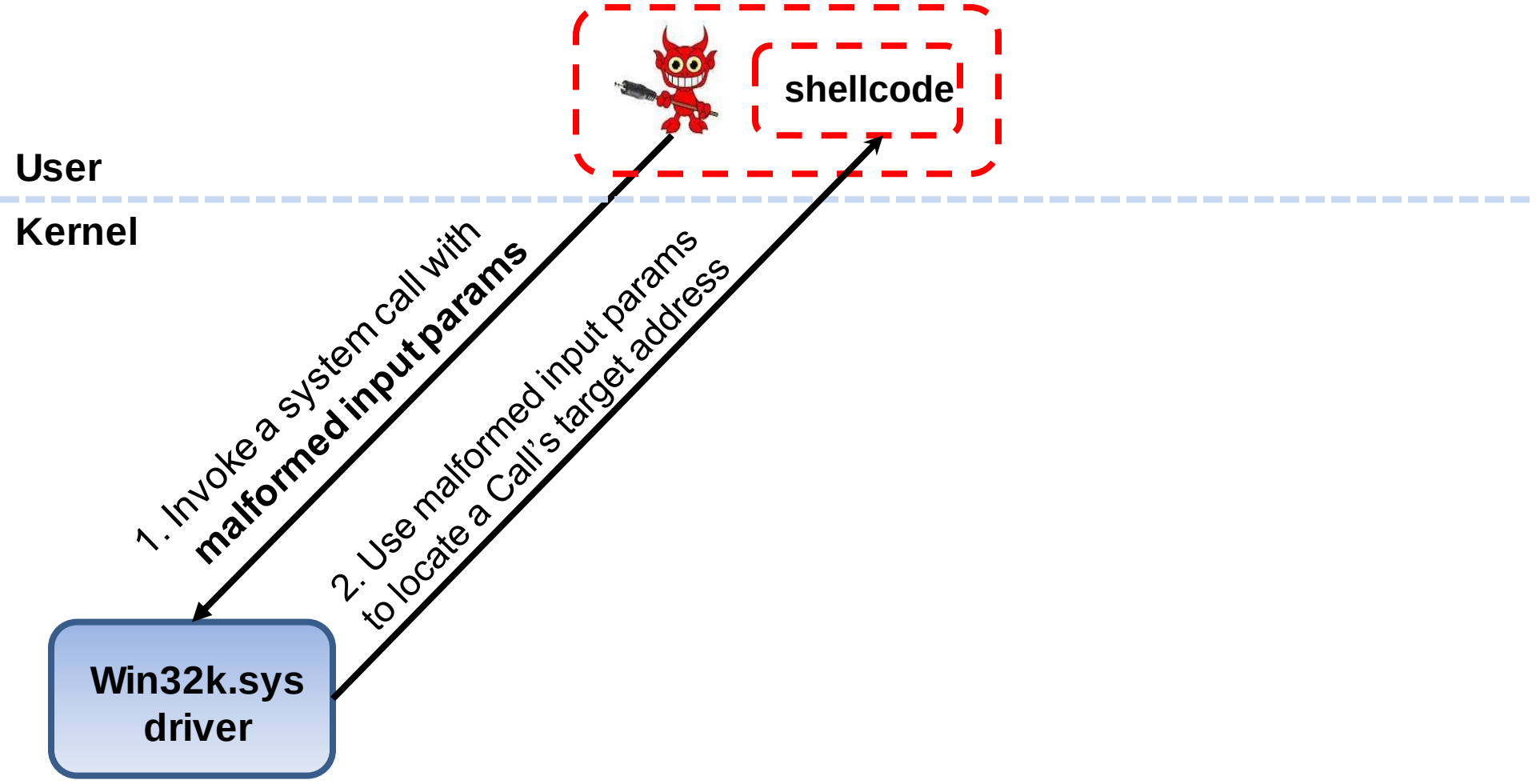
Mixed-Mode Malware Sample 1: Stuxnet Kernel Exploit

- User-mode malware
- Runs a **kernel-mode shellcode** to perform "**privilege escalation**" attack:
  - Access the list of process objects (EPROCESS) in kernel
  - Overwrite the token value (a member field) of malware's process object with "System" process object's token value
  - Such **kernel data modification** makes user-mode instructions execute with **administrator** privilege
- Kernel exploits can freely manipulate **any** OS data and code **besides** the common usage for privilege escalation (e.g., escalation attack + process hiding)

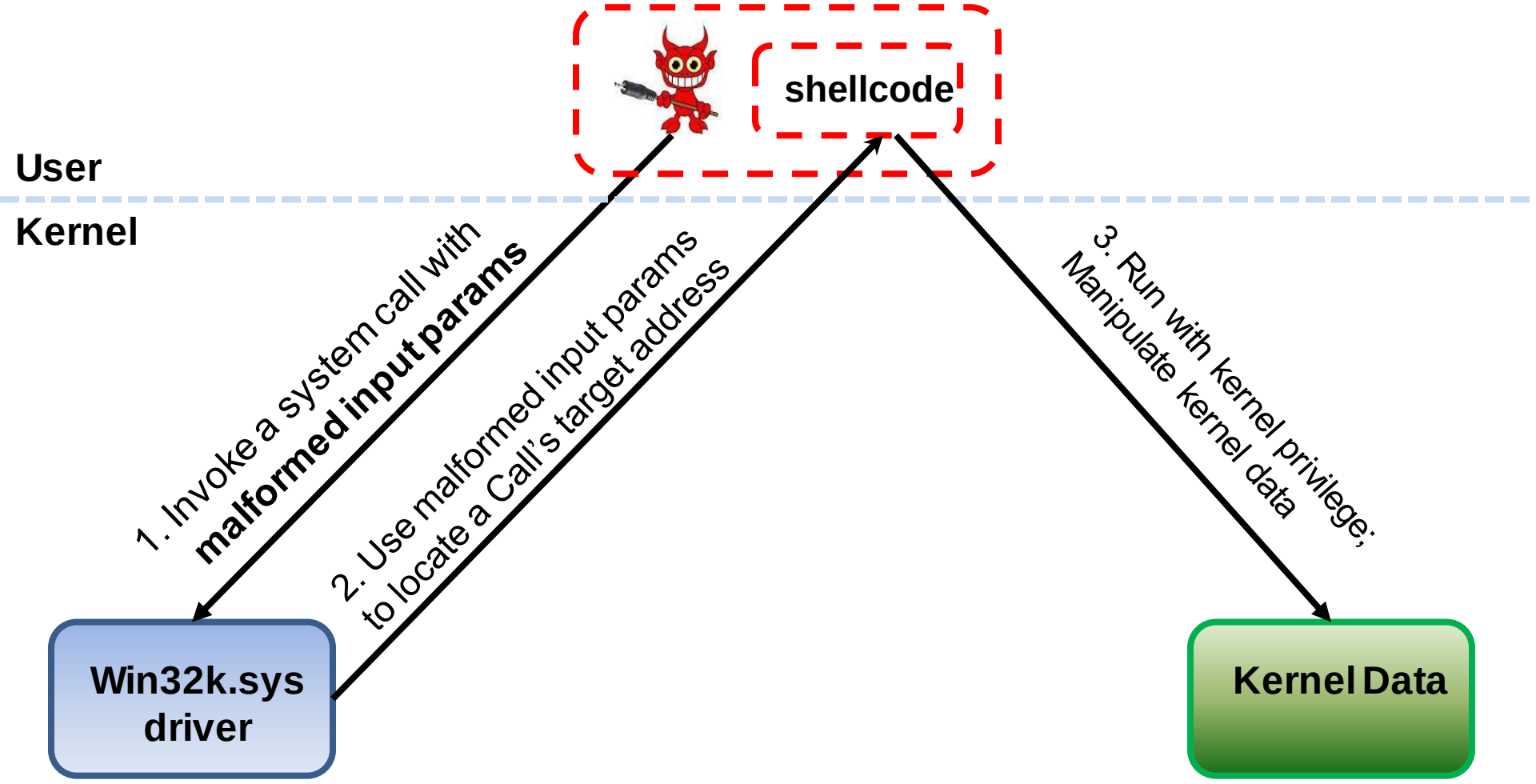
Mixed-Mode Malware
Sample 1: Stuxnet Kernel Exploit



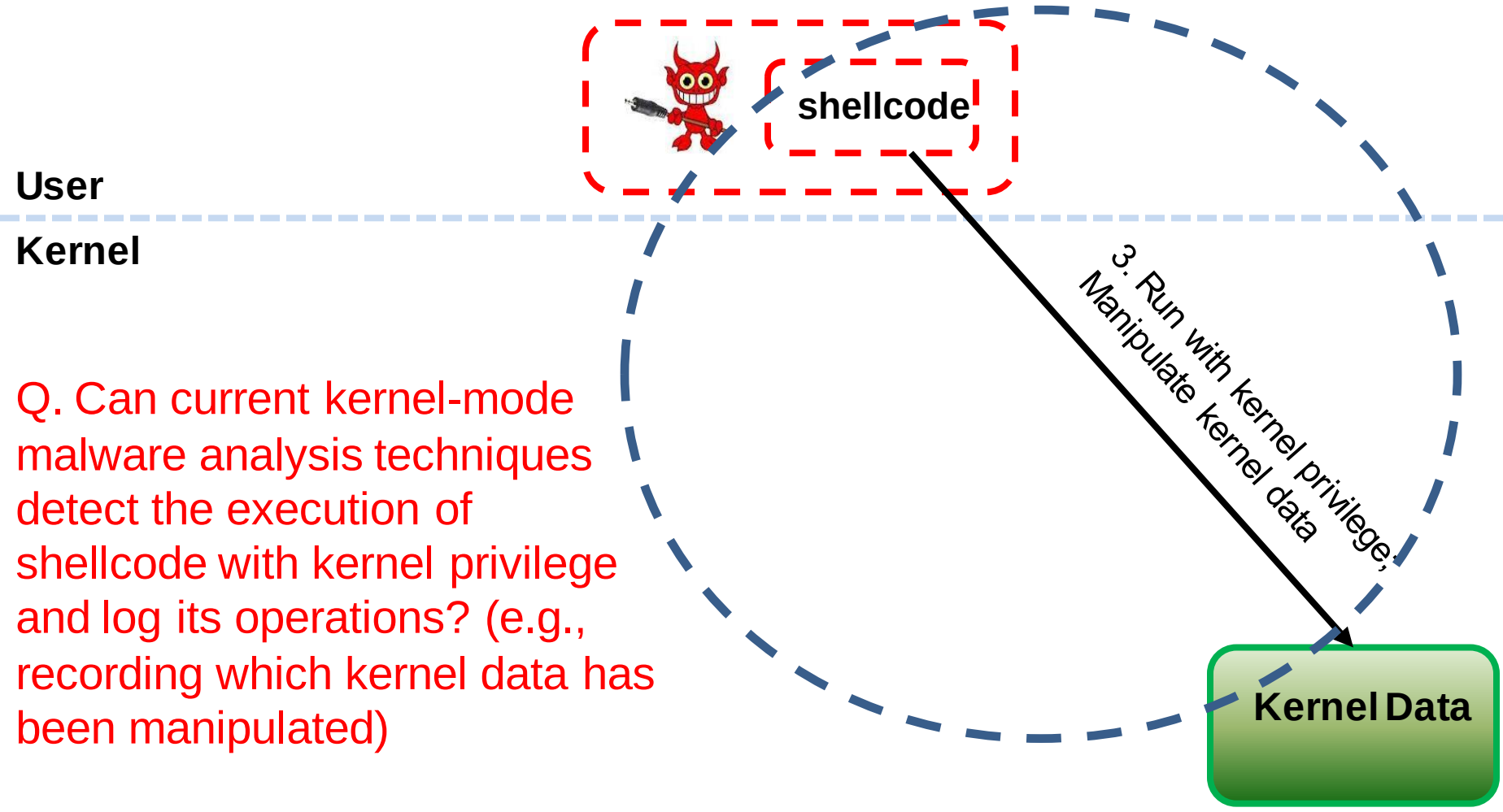
Mixed-Mode Malware
Sample 1: Stuxnet Kernel Exploit



Mixed-Mode Malware
Sample 1: Stuxnet Kernel Exploit



Mixed-Mode Malware
Sample 1: Stuxnet Kernel Exploit



Q. Can current kernel-mode malware analysis techniques detect the execution of shellcode with kernel privilege and log its operations? (e.g., recording which kernel data has been manipulated)

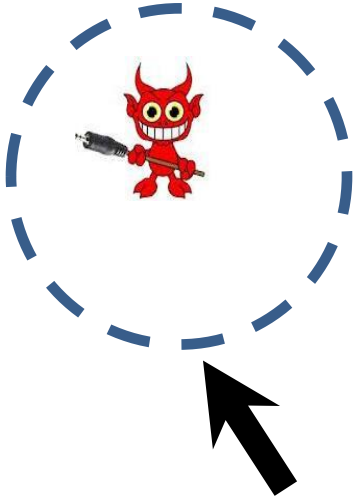
dAnubis Kernel Analysis Tool Against Sample 1

- Part of Anubis malware analysis tool
- Effective for analysis of **standalone** kernel-mode malware samples (e.g., malicious drivers or kernel-mode rootkits)
- Not effective for analysis of kernel exploits such as sample 1

dAnubis Kernel Analysis Tool Against Sample 1

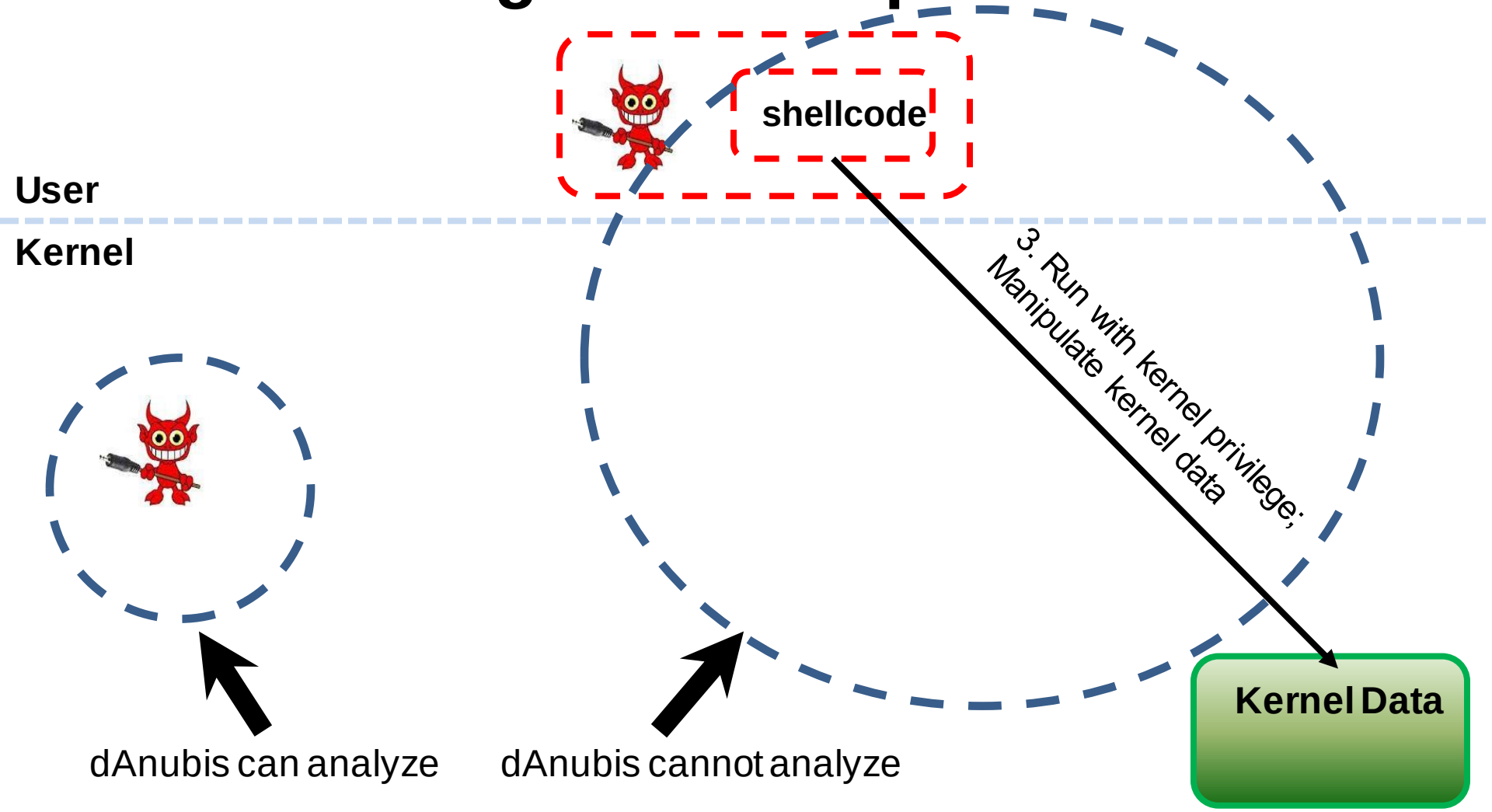
User

Kernel



dAnubis can analyze

dAnubis Kernel Analysis Tool Against Sample 1



Mixed-Mode Malware Analysis
Design Key 1

- Effective mixed-mode analysis technique must **detect** and **analyze** the execution of **untrusted kernel code** (e.g., the shellcode in sample 1)

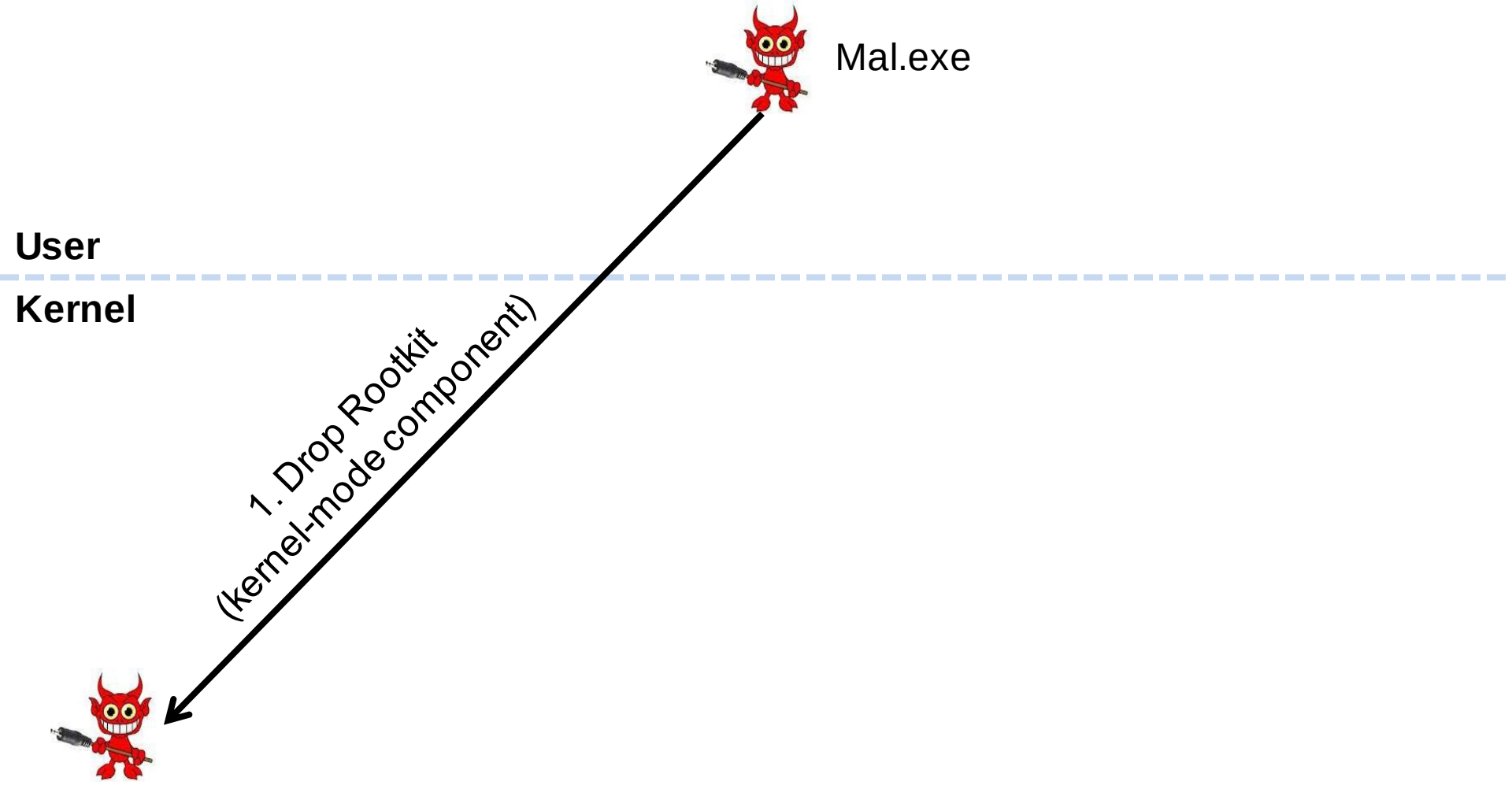
Mixed-Mode Malware
Sample 2: Inter-dependant user and kernel components

- A bit background about common **non mixed-mode malware** samples:
  - Many malware samples have user and kernel mode components (e.g., flame, duqu)
  - The kernel mode component is usually a rootkit that hides malware trace (e.g., hiding malware process in user mode) and **does not participate** in actual attack

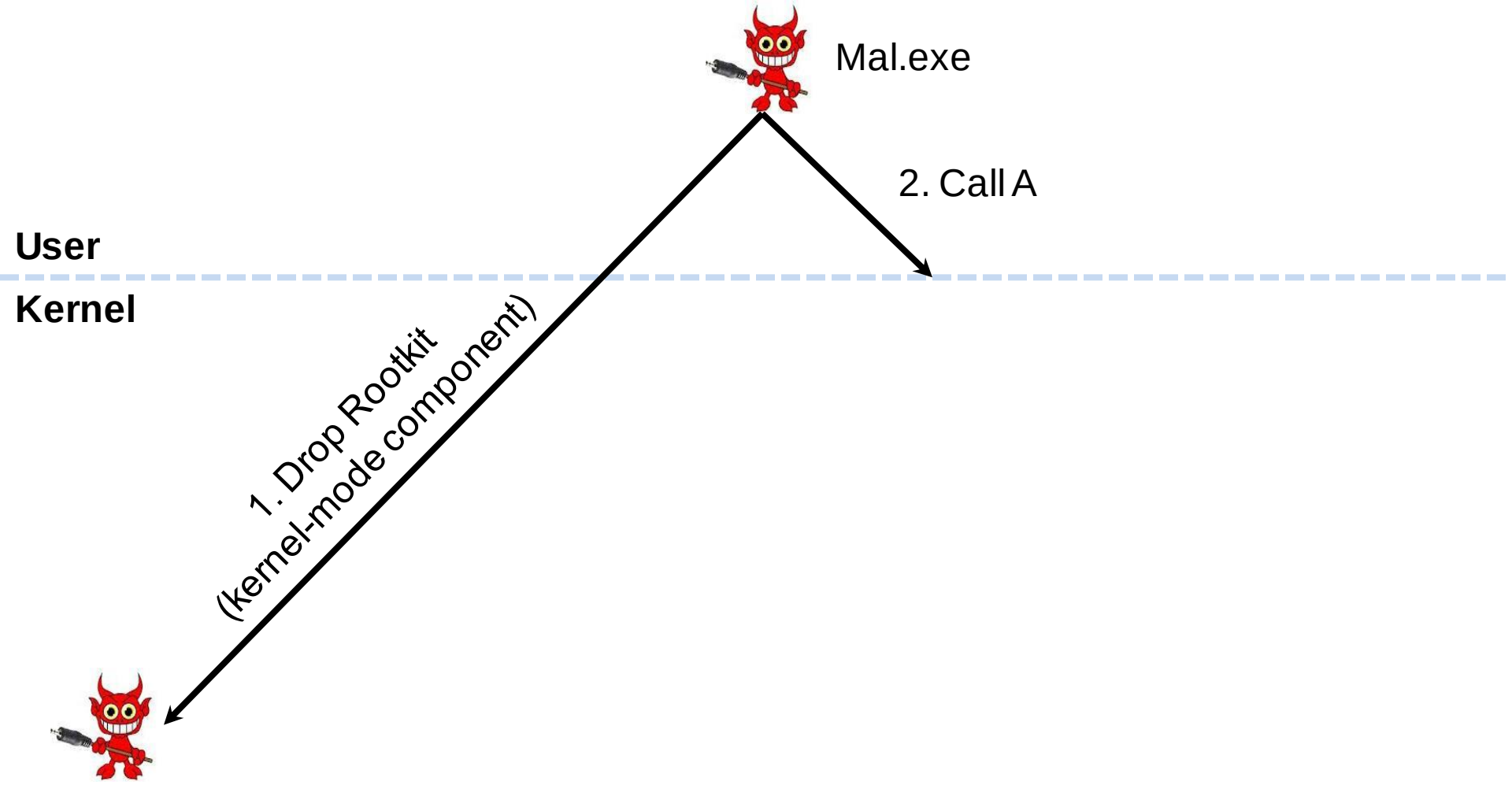
Mixed-Mode Malware
Sample 2: Inter-dependant user and kernel components

- **System call swapping attack**: Malware uses a kernel-mode component to **change the semantic** of system calls it invokes
  - Malware invokes a benign sequence of system calls in user-mode but a malicious sequence of system services executes in kernel
  - Tracking system calls of applications is an important technique for malware detection and analysis
  - Such mixed-mode malware can evade intrusion detection systems or malware analysis techniques

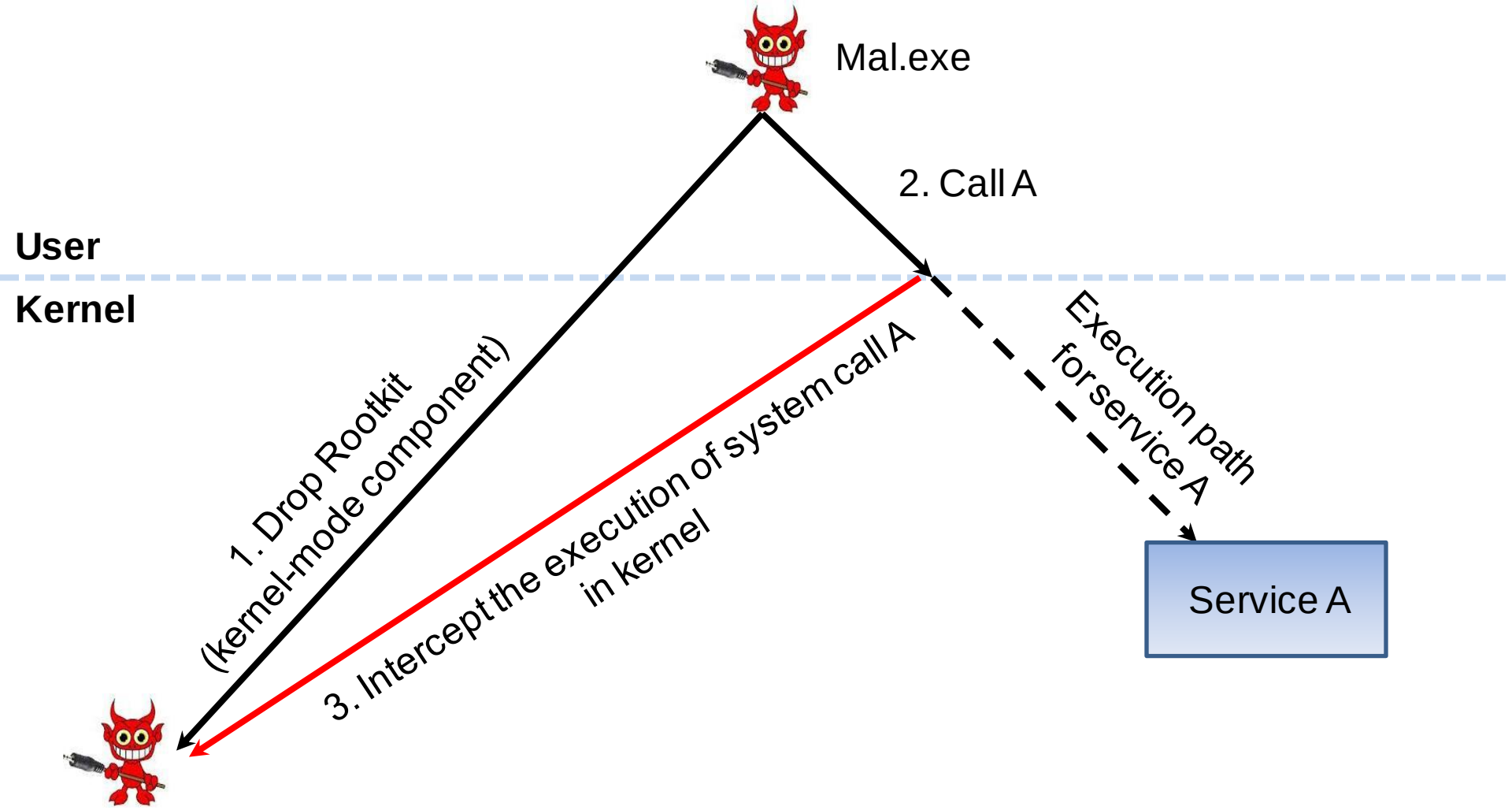
Mixed-Mode Malware
Sample 2: Inter-dependant user and kernel components



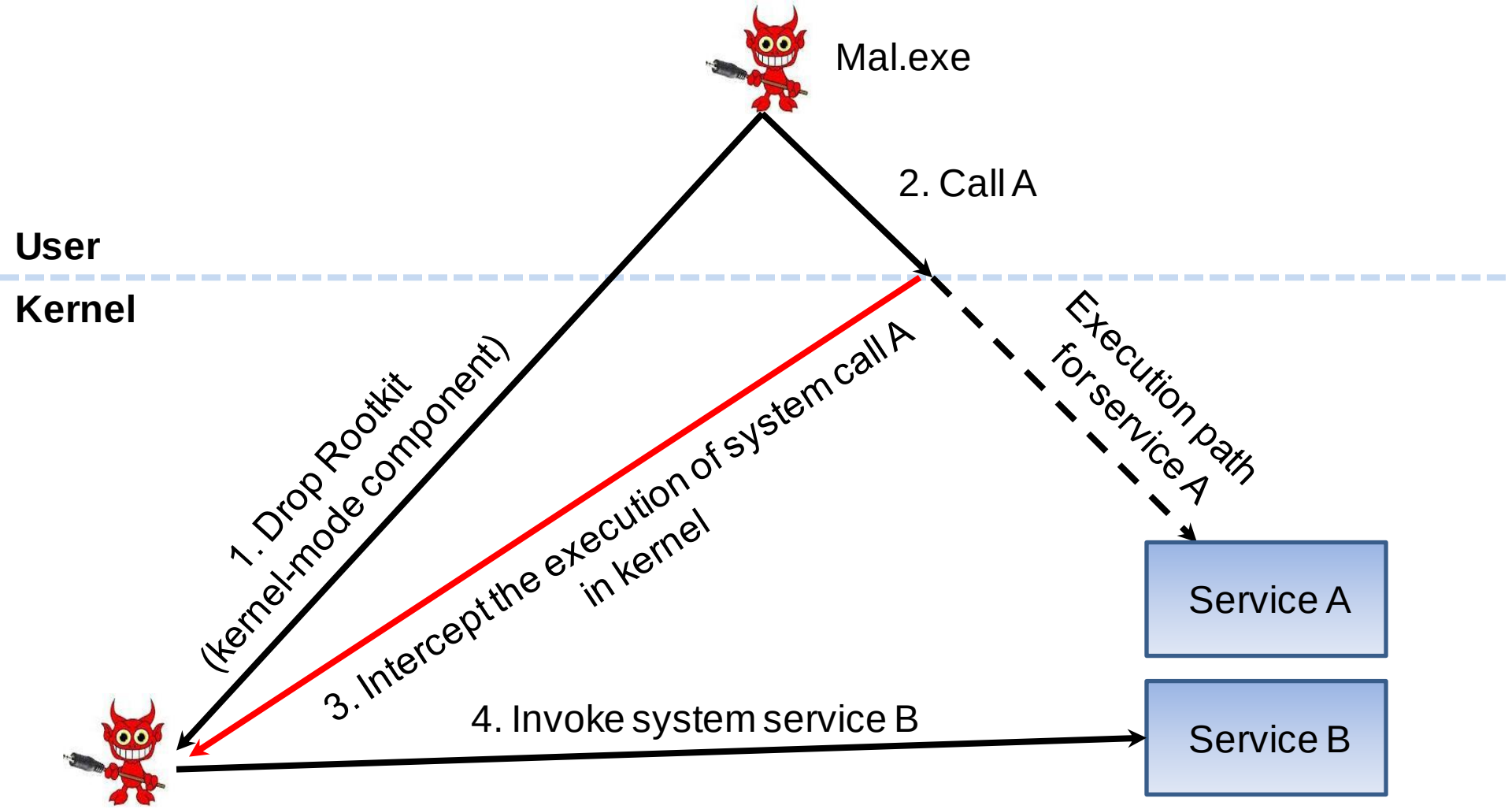
Mixed-Mode Malware
Sample 2: Inter-dependant user and kernel components



Mixed-Mode Malware
Sample 2: Inter-dependant user and kernel components



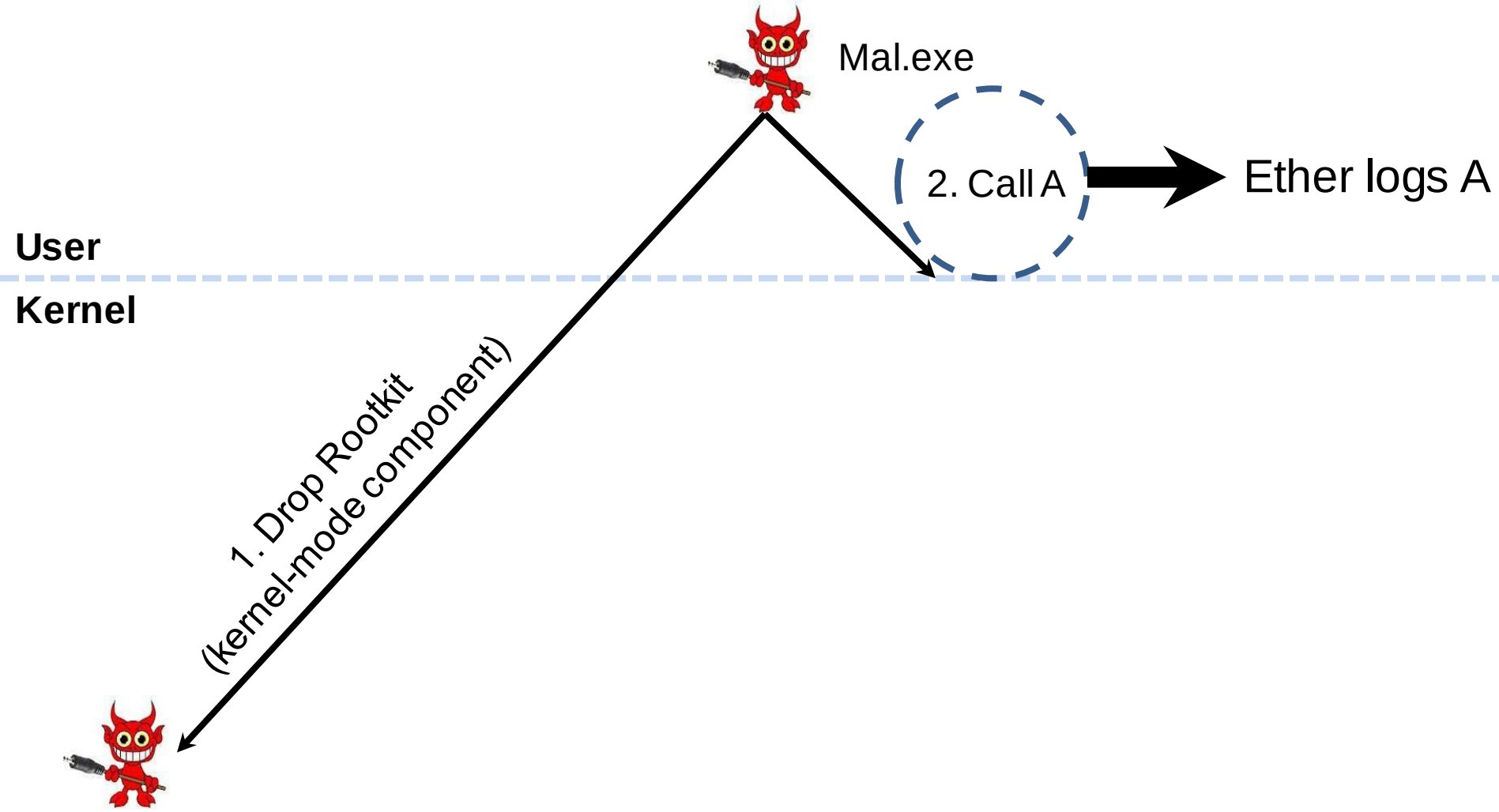
Mixed-Mode Malware
Sample 2: Inter-dependant user and kernel components



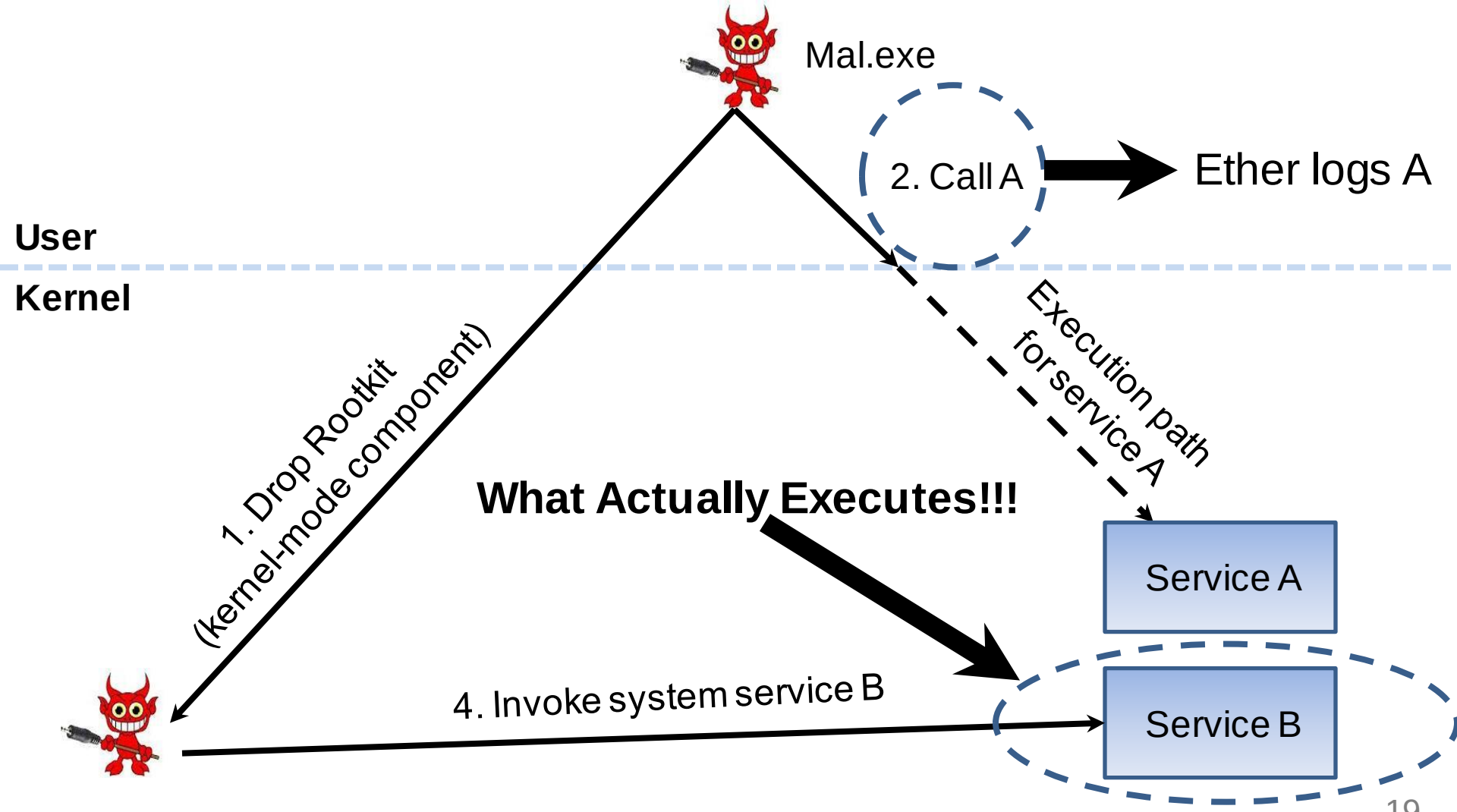
Ether Malware Analysis Tool Against Sample 2

- Proof-of-concept malware analysis tool built on basis of hardware virtualization technology (Xen)
- **Single-mode** analysis approach:
  - Trust on integrity of kernel during the analysis
  - Inaccurate system call tracking log for analysis of sample 2

Ether Malware Analysis Tool Against Sample 2



Ether Malware Analysis Tool Against Sample 2



Mixed-Mode Malware Analysis
Design Key 2

- Effective mixed-mode analysis technique must operate in **both** user and kernel mode and track the execution of kernel during the analysis

Mixed-Mode Malware Analysis

- **Design Key 1**: Detect and Log the execution of untrusted kernel code

- **Design Key 2**: Operate in both user and kernel mode

Q. Can we customize a current malware analysis tool for mixed-mode analysis?

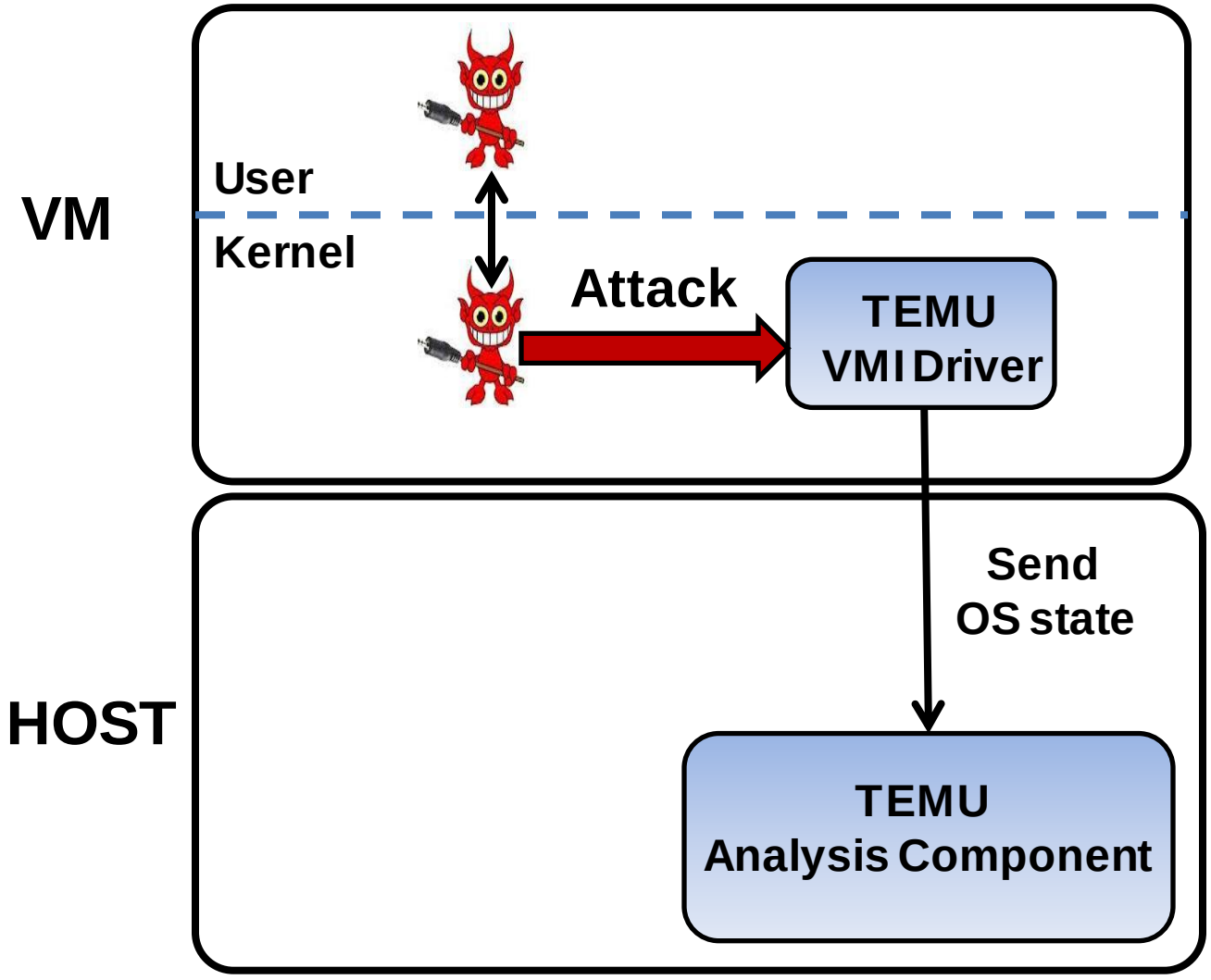
TEMU Malware Analysis Tool Against Mixed-Mode Malware

- Part of Bitblaze project
- Uses software virtualization technology (QEMU)
- Uses a **kernel-mode driver** for Virtual Machine Introspection (VMI)

  *VMI: Obtain the OS state (e.g., current process) from hardware state (memory bytes and register values) of virtual machine

- A mixed-mode malware can compromise the execution of the analysis components residing in its execution domain (e.g., manipulating the OS functions used for VMI such as GetCurrentProcess)

TEMU Malware Analysis Tool Against Mixed-Mode Malware



Mixed-Mode Malware Analysis
Design Key 3

- Malware Analysis components should reside **outside** the domain of mixed-mode malware

Mixed-Mode Malware Analysis
Approach 1

- **Design Key 1**: Detect and Log the execution of untrusted kernel code
- **Approach 1:**
  - **Before malware execution:** Get OS state (address range of trusted kernel code and kernel data)
  - **After malware execution:** Detect the execution of untrusted kernel code using the obtained **trusted code address range**

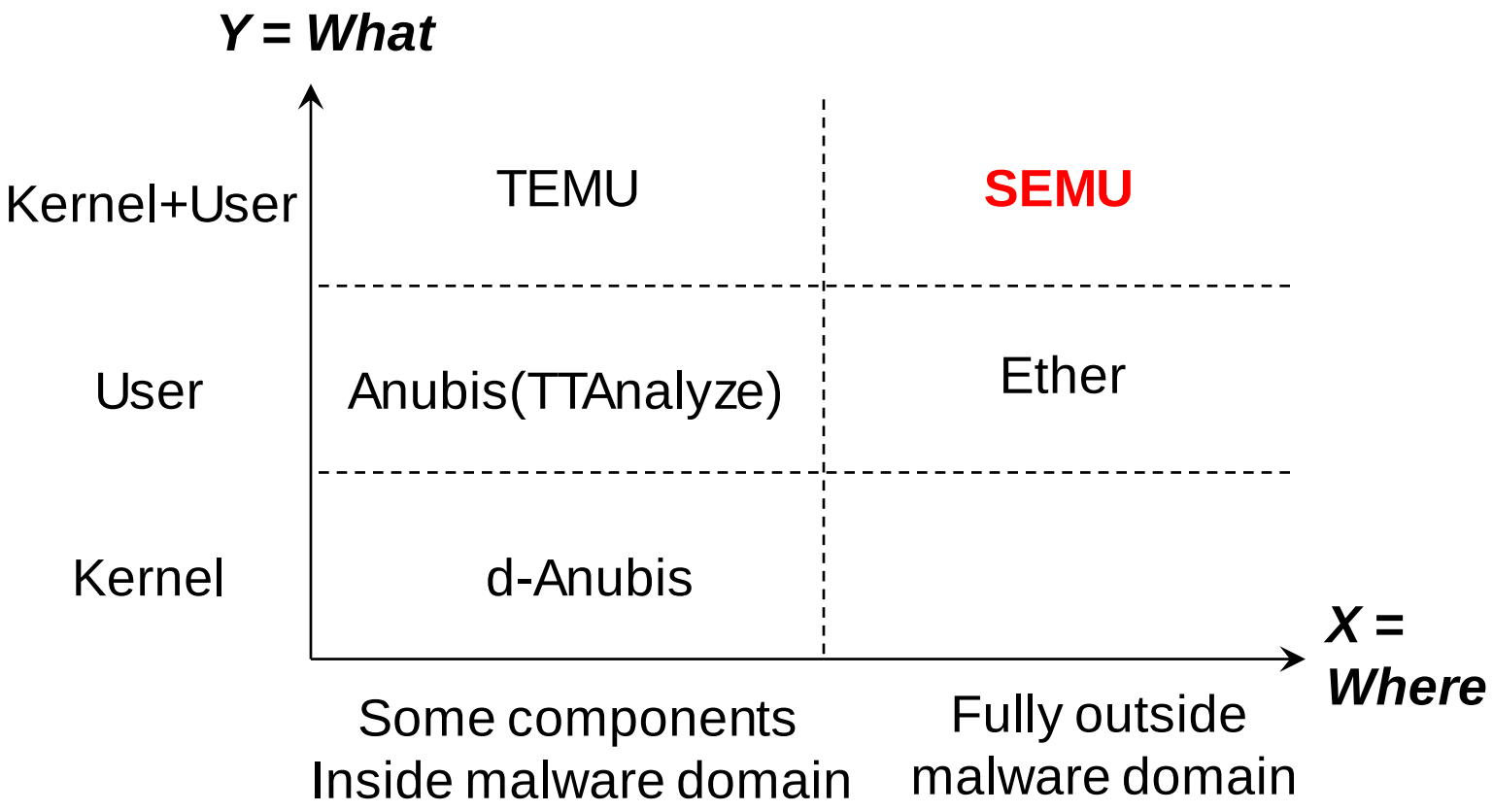
Mixed-Mode Malware Analysis
Approach 2

- **Design Key 2**: Operate in both user and kernel mode
- **Approach 2:**
  - Get **whole-system execution** trace of malware (even for user-mode samples) :
    - Control flow of kernel after system call occurs
    - Access (read/write) of kernel data

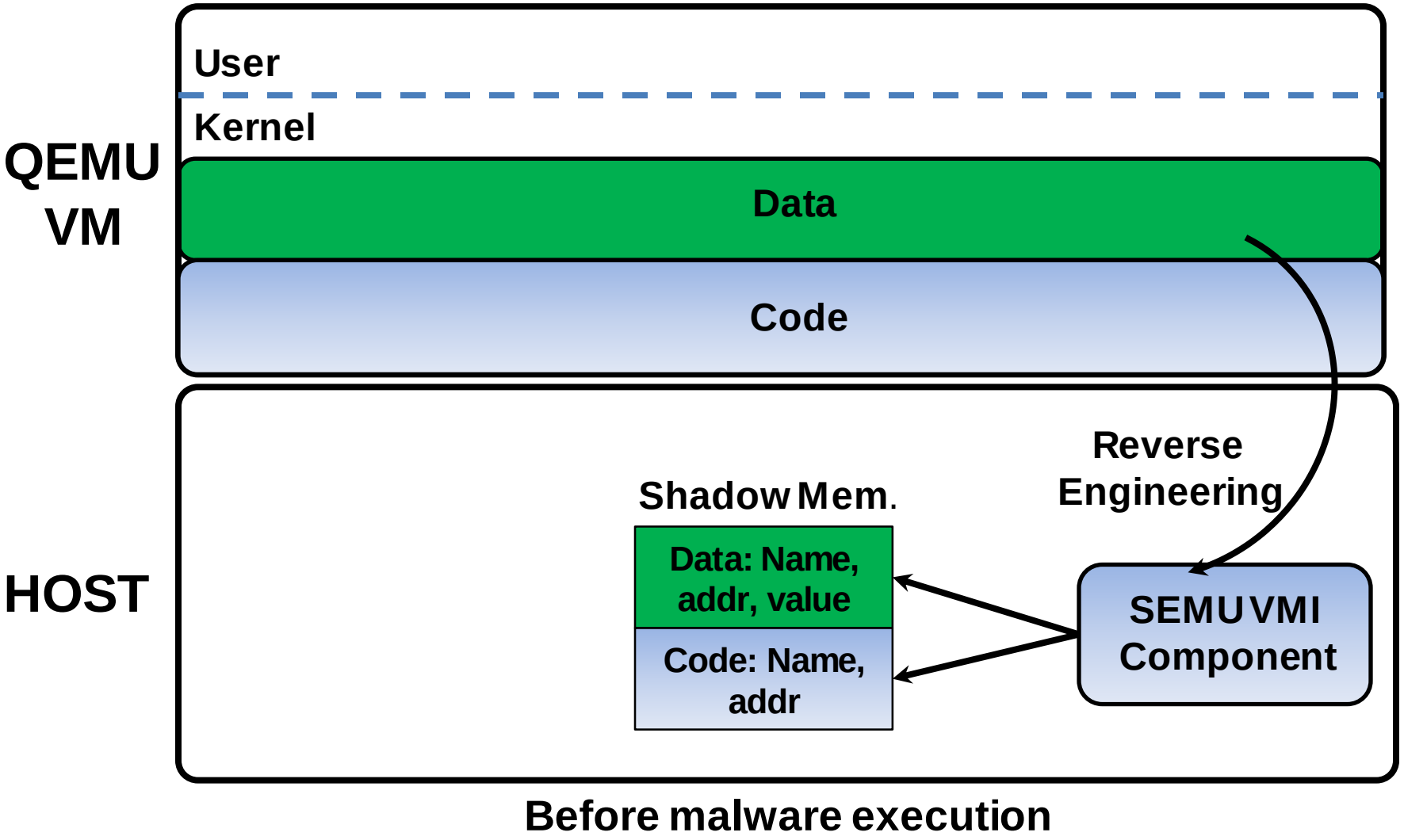
Mixed-Mode Malware Analysis
Approach 3

- **Design Key 3**: Put analysis components outside the mixed-mode malware's domain
- **Approach 3:**
  - Perform VMI by **reverse engineering** of OS from **outside** of the VM
    - For Windows OS, use of PDB (program database) files as debug symbols:
      - Obtain **name** and **address** of **many** OS data and codes

SEMU Malware Analysis Tool

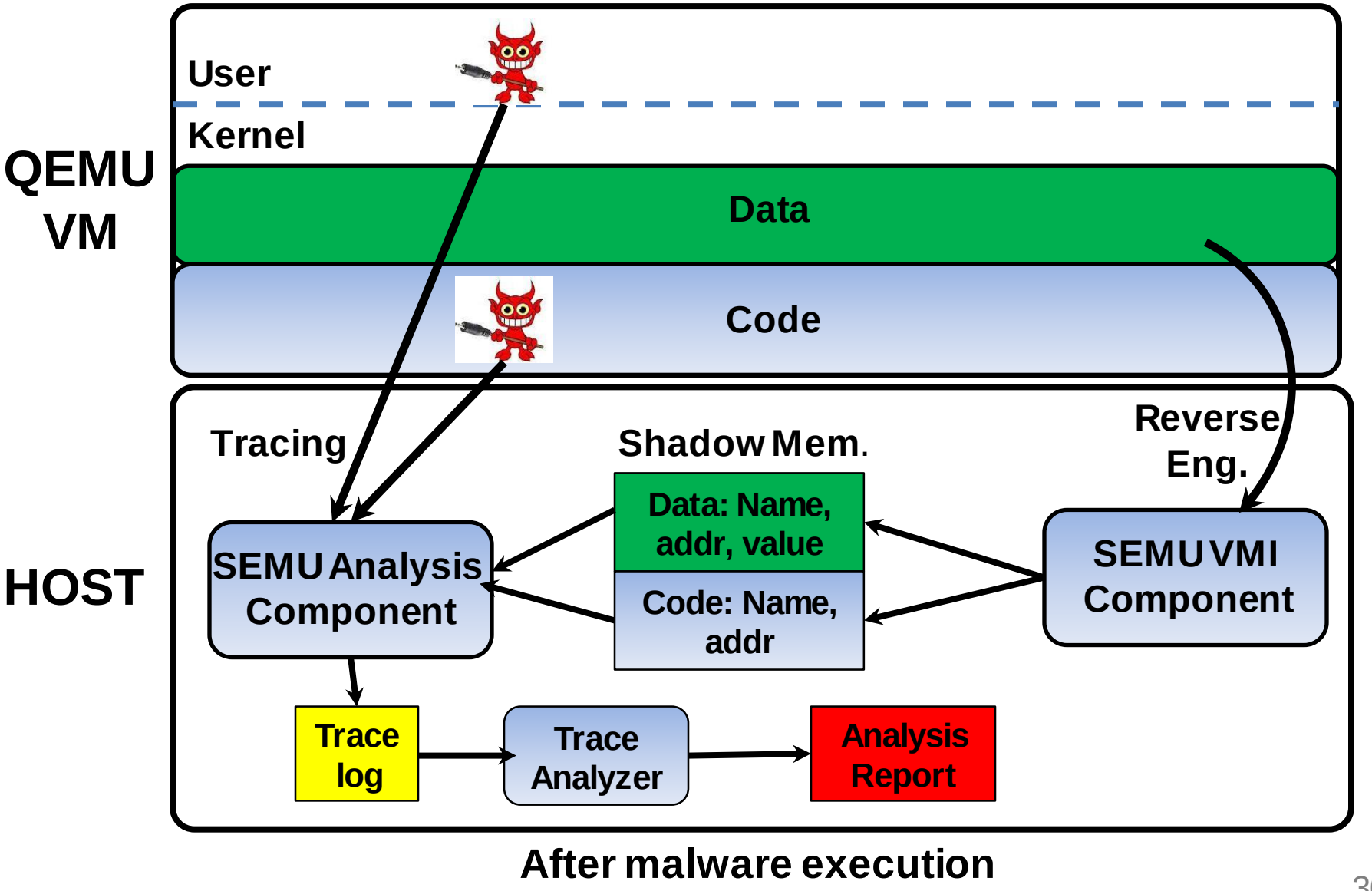


SEMU Malware Analysis Tool



Before malware execution

SEMU Malware Analysis Tool



After malware execution

Evaluation

- Research Question 1: Can SEMU analyze mixed-mode malware that cannot be fully analyzed by current state-of-the-art approaches?

Mixed-Mode Malware Analysis Result

Description	Affected Object	Via OS function	Kernel LOC	User LOC	Slow-down
Modify System calls	KTHREAD	No	370	1,684	35.3
Modify system calls (MDL)	SSDT	Yes	417	1,684	38.7
DKOM object hiding	EPROCESS DRIVER_OBJECT	No	96	451	28.2
DKSM renaming	EPROCESS	No	111	451	20.6
Privilege escalation	EPROCESS	No	0	149	25.2
User-mode unhook	SSDT	Yes	0	710	29.1

Evaluation

- Research Question 2: Is the SEMU execution time competitive with current state-of-the-art approaches?

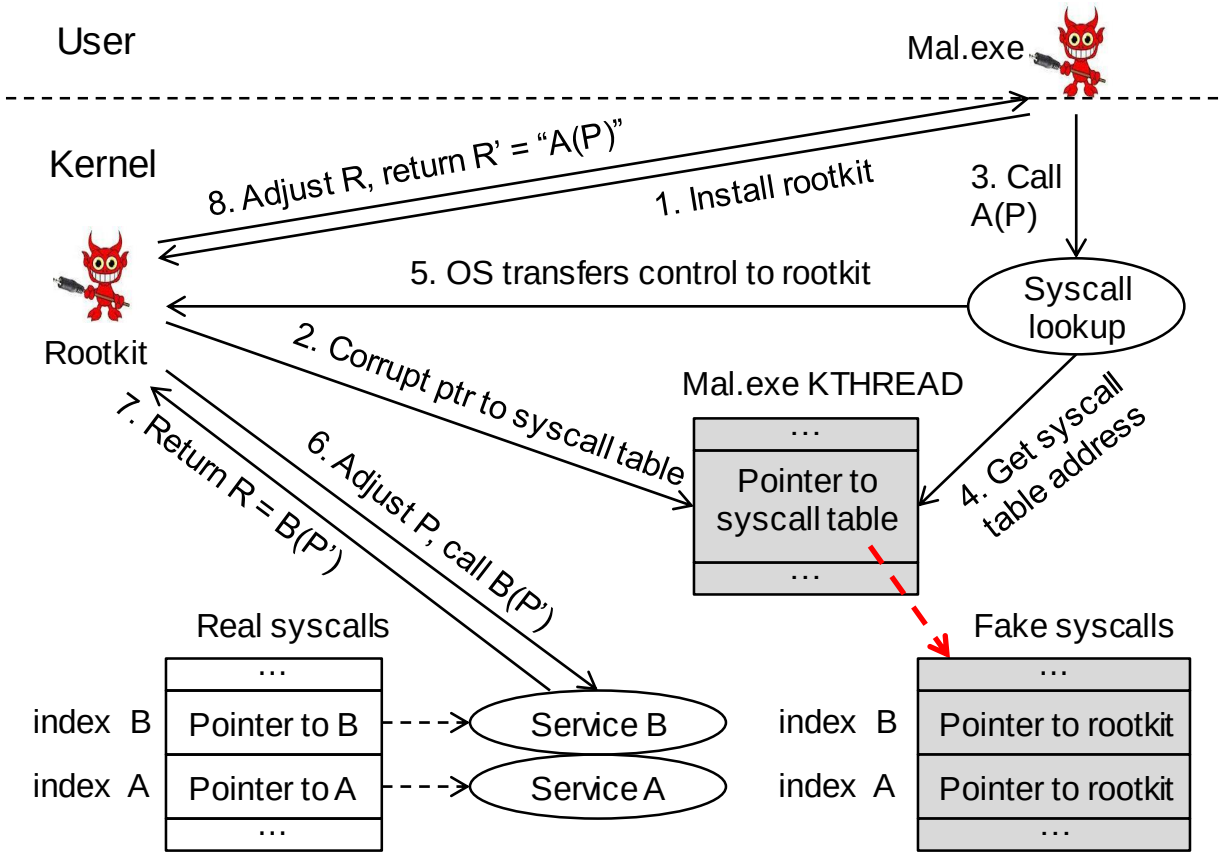
Fine-grained VMI: Instruction tracing in Ether and SEMU (S)

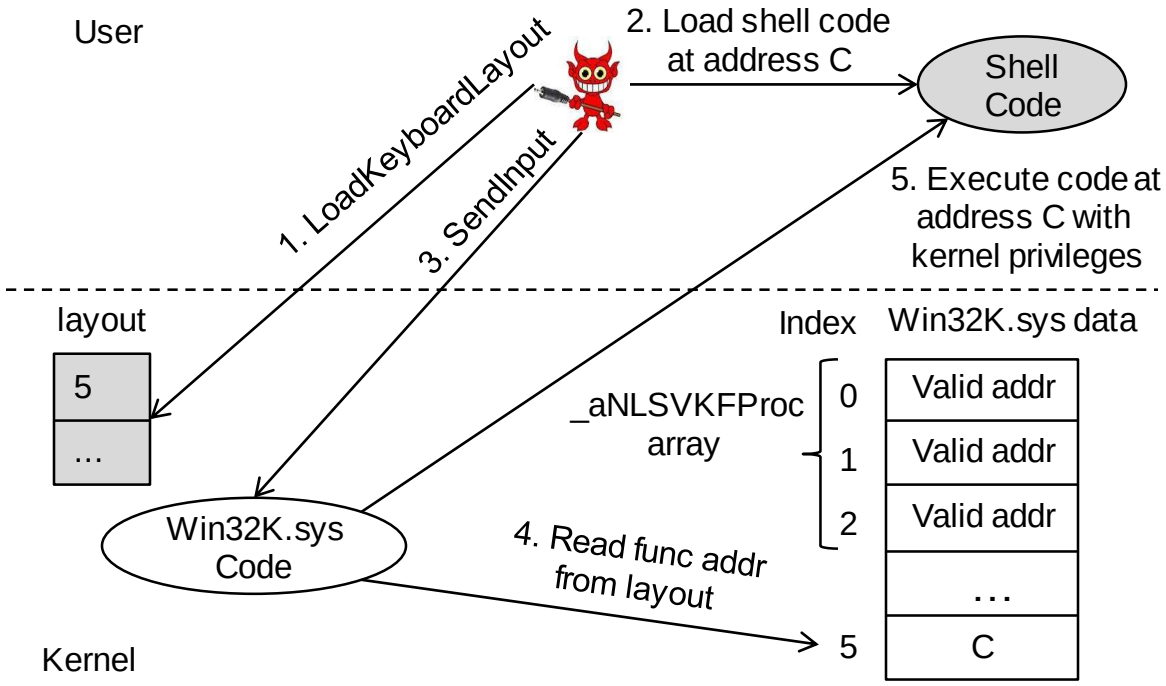
Subject	w/o VMI [s]		fine VMI [s]		Slowdown	
	Ether	S	Ether	S	Ether	S
Efsinfo	0.63	2.42	20.54	21.39	32	8
Timezone	0.05	0.79	4.41	13.03	87	16
Whoami	0.03	0.72	4.49	19.83	149	27
UPX	0.32	9.00	45.58	322.60	141	35
RAR a	0.15	3.07	45.16	302.93	300	98

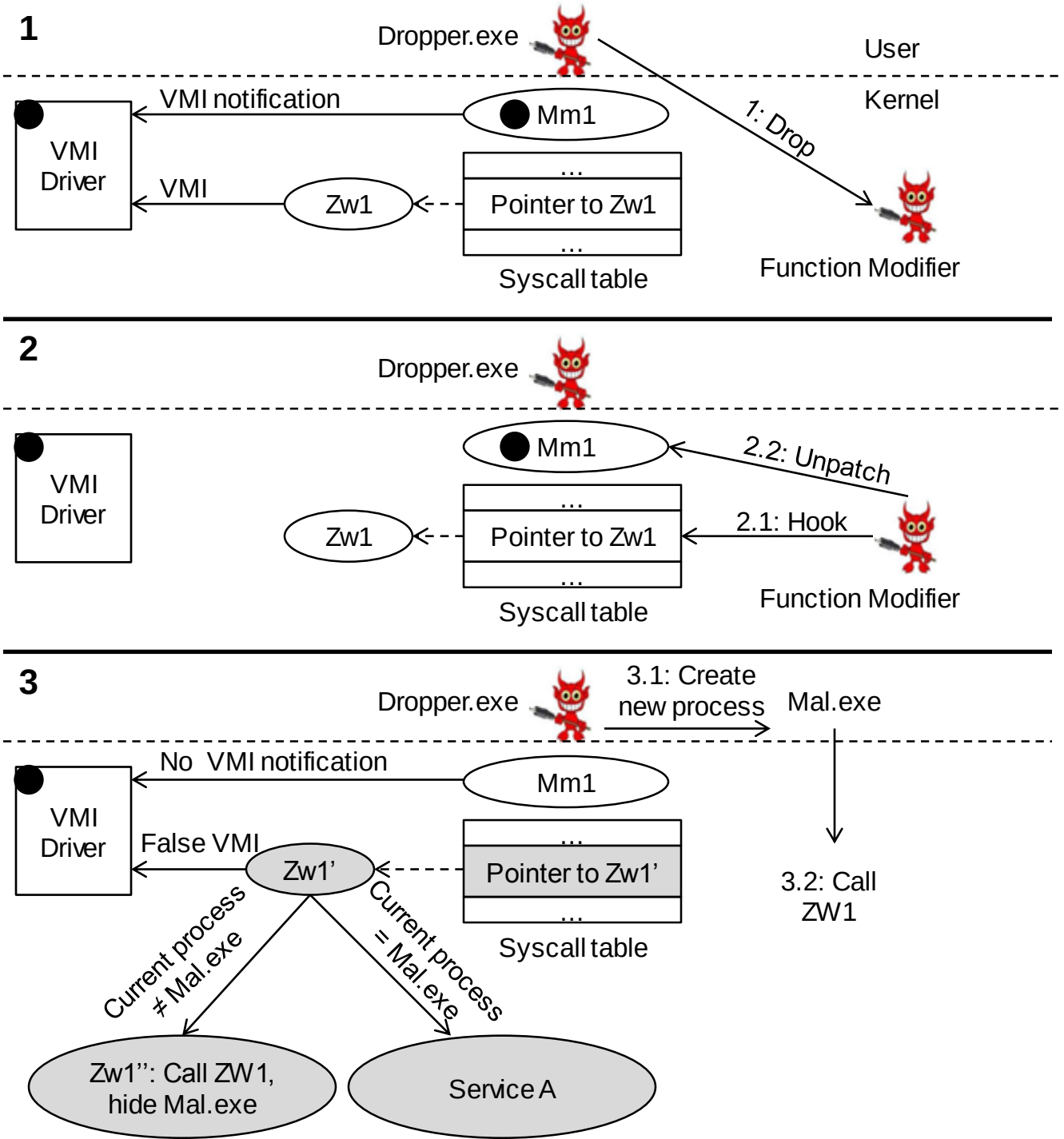
Inside-the-guest VMI in TEMU (T) and Outside-the-guest VMI in SEMU (S)

Subject	w/o VMI [s]		Coarse [s]		% O/H	
	T	S	T	S	T	S
PsGetsid	1.68	0.56	3.44	1.09	105	95
Pslist -t	3.19	1.03	4.69	1.31	47	27
Psinfo -s	5.76	2.88	9.79	4.78	70	66
Coreinfo	1.70	0.65	3.75	1.07	121	63
ListDLLs	3.20	2.58	5.01	3.75	57	45

Backup


User
Mal.exe
Kernel
8. Adjust R, return R' = "A(P)"
1. Install rootkit
3. Call A(P)
5. OS transfers control to rootkit
Syscall lookup
Rootkit
2. Corrupt ptr to syscall table
Mal.exe KTHREAD
…
Pointer to syscall table
…
4. Get syscall table address
7. Return R = B(P')
6. Adjust P, call B(P')
Real syscalls
…
index B
Pointer to B
Service B
index A
Pointer to A
Service A
…
Fake syscalls
…
index B
Pointer to rootkit
index A
Pointer to rootkit
…


User
2. Load shell code at address C
Shell Code
1. LoadKeyboardLayout
3. SendInput
5. Execute code at address C with kernel privileges
layout
5
...
Index
Win32K.sys data
_aNLSVKFProc array
0
1
2
Valid addr
Valid addr
Valid addr
...
C
Win32K.sys Code
4. Read func addr from layout
5
Kernel


1
Dropper.exe
User
Kernel
VMI notification
Mm1
1: Drop
VMI Driver
VMI
Zw1
...
Pointer to Zw1
...
Syscall table
Function Modifier
2
Dropper.exe
Mm1
2.2: Unpatch
VMI Driver
...
Zw1
Pointer to Zw1
2.1: Hook
...
Syscall table
Function Modifier
3
Dropper.exe
3.1: Create new process
Mal.exe
No VMI notification
Mm1
VMI Driver
False VMI
Zw1'
...
Pointer to Zw1'
...
Syscall table
3.2: Call ZW1
Current process ≠ Mal.exe
Current process = Mal.exe
Zw1'': Call ZW1, hide Mal.exe
Service A