symantec™

Confidence in a connected world.

Code Injection From the Hypervisor:
Removing the need for in-guest agents

Matt Conover & Tzi-cker Chiueh

Core Research Group, Symantec Research Labs

SADE:
SteAlthy Deployment and Execution

- Introduction
  - A typical enterprise environment, what's wrong with it, and how SADE improves it
- Implementation
  - How SADE is using VMsafe
- Performance
  - In-guest agents versus SADE injected agents

SADE:
SteAlthy Deployment and Execution

- This project is focused on enterprise environments.
- Let me start with a higher-level vision…

- A simplified enterprise environment (pre-SADE):
  - Workstation: one desktop computer per employee
  - Agents: software components (anti-virus, update programs, etc.) installed on each workstation
  - Domain: all enterprise workstations are part of a domain (e.g., Active Directory)
  - Domain server: controls authentication, policies, and software updates of workstations
  - Domain administrator: maintains all of the workstations

SADE: Introduction

What's wrong with this model?

1. Administrative headache
2. Wasted resources (disk space, electricity, CPU time, etc.)
3. Security risk

SADE: Introduction

- Administrative headache
  - Domain administrator needs to keep all machines updated
  - Need to install separate agents for everything (an anti-virus agent, a software update agent, etc.)
  - Less-than-seamless: if the user gets infected with a virus, it may disable the anti-virus. Then what? Administrator needs to manually clean the machine

SADE: Introduction

- Wasted resources
  - Why does each desktop need an update program when the enterprise desktops are all fairly homogenous?
  - Antivirus scans all files at least once per workstation, although each workstation mostly has the same files. The agent of each workstation is working in isolation.
  - Having the same software installed on each machine wastes disk space
  - Performing the same scans on each machine wastes electricity and CPU resources

SADE: Introduction

- Security risk
  - The classic problem of security software and threats operating at the same privilege level
  - If the security agent lives on the workstation, it can be disabled by undetected malware.
  - There is no way to *real* way to remediate this except to boot from a rescue CD

SADE: Project Goal

- Eliminate the need for “agents” running on the user’s machine
  - Instead of having agents everywhere, do all of these steps from a central location
  - Make “targeted deployments” when necessary
- How?
  - Use virtual machines instead of workstations
  - Do software updates from the hypervisor
  - Do security checks from the hypervisor
  - Do file scans from the hypervisor

SADE: Project Goal

- Benefits
  - Simplifies the whole design
  - Don’t need to maintain agents in each workstation
  - Scanning files can be done once globally from the hypervisor rather than once per machine

SADE: Project Goal

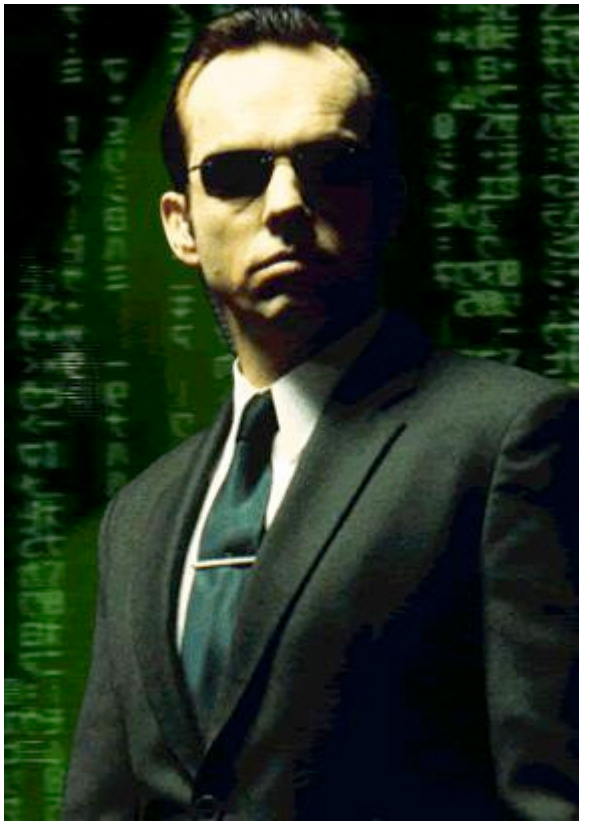
- You might know that VMWare already has a tool to load an executable file inside the guest virtual machine…

- Why not just use VMware tools to load an executable?
  - This is not meant to be used in a hostile environment.
  - It will mount the program as an ISO (use the CDROM) and run a user-mode executable from the CD
  - This is very easy for a malware to detect and prevent (i.e., kernel-mode rootkit hooking NtCreateProcess).

  - Our approach never touches the disk of the guest. The code runs directly from kernel-mode and doesn't require the OS driver loader.
  - This is a much better approach for a hostile environment…

SADE: Project Goal

- Benefits
  - Malware on the workstation can't disable the agents, because they aren't even there. They can pop in, at anytime, unexpectedly…



SADE: Project Goal

- A simplified enterprise environment (post-SADE):
  - Workstation: each desktop computer replaced by a virtual machine
  - Domain: all virtual machines run under a hypervisor
  - Agents: stored in a central repository of the domain and deployed to the workstation only when necessary

Team Symantec!

SADE in a Nutshell

- The agent only exists in the guest while it is executing
  - Once the agent finishes executing, it is removed from the guest and the memory is wiped clean.
- Can be completed in less than second
  - The window for malware to detect or disable our agent is very small

SADE in a Nutshell

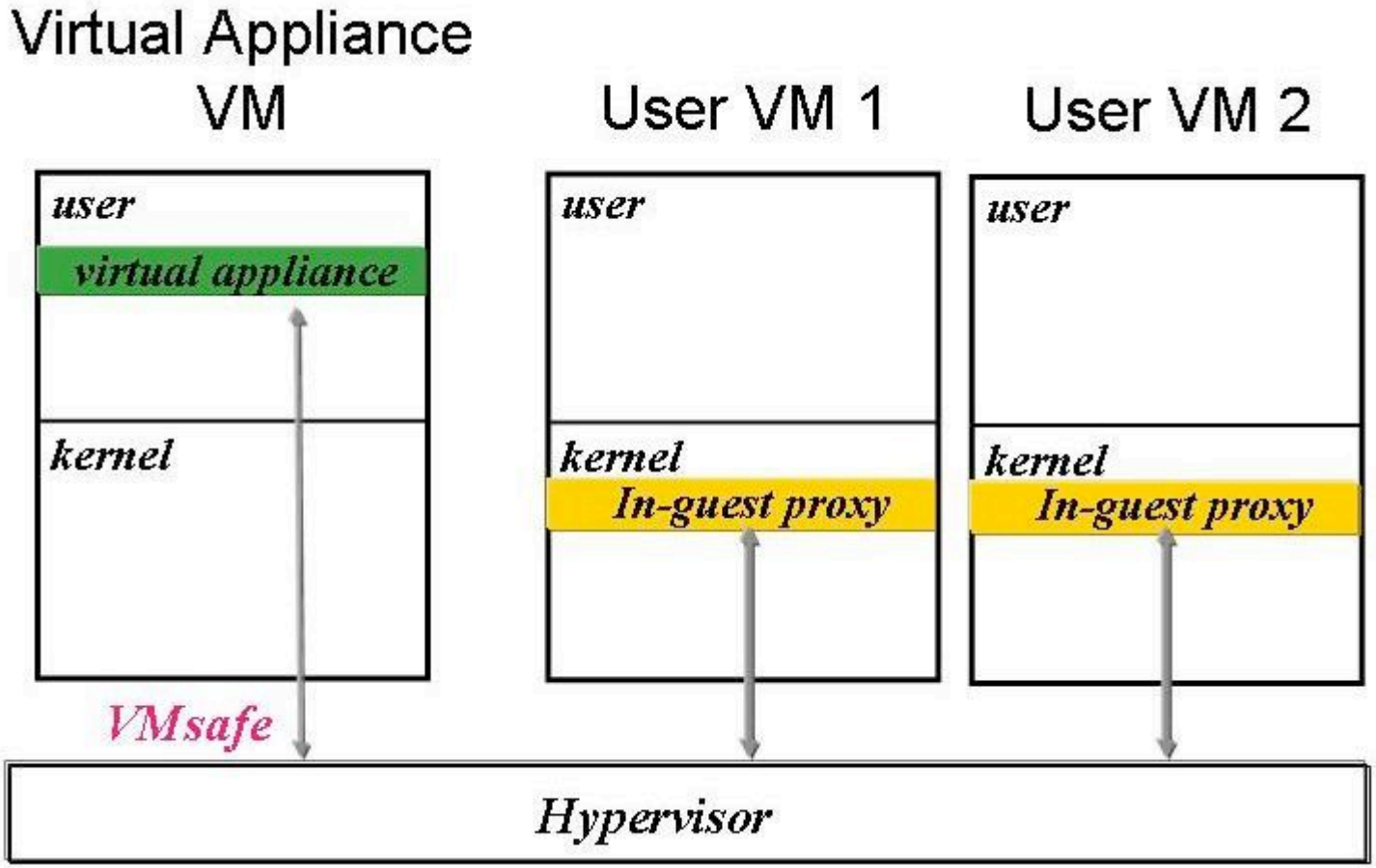
- SADE can inject an agent into the guest virtual machine without the help of the OS.
- SADE will load the driver itself, it does not rely on the native OS driver loader
- Development is easy
  - The agent is a standard Windows kernel driver compiled using standard tools (Windows DDK, written in C)
  - The agent can use all the standard kernel APIs like DbgPrint

Our Prototype

- In our implementation we used VMware's ESX server as our hypervisor and VMsafe APIs to interface with the hypervisor
- VMsafe gives us a way to detect when a memory page is about to be read, written, or executed.
- Our prototype: Implemented an anti-virus scanner on the hypervisor which then injects a remediation driver into the guest virtual machine to remove a virus once detected.

Our Prototype

- Our prototype protecting two virtual machines (User VM 1&2)



Scenario

- Here's the scenario I'll be describing during the rest of the talk..
- Using anti-virus definitions running on security VM to scan the user VMs for malware.
  - Use memory scanning rather than file scans
- A virus (W32.Gammima) is run in the user VM and detected.
  - We want to remediate this virus by terminating the process
  - We'll inject code into the guest to do this.
  - To be absolutely safe, we'll do the remediation in kernel-mode (protect against kernel rootkits)

Step 1: Detect the Threat

- Uses page execution trigger on all memory pages to detect when a page is about to be executed
- Scans the memory page
- If the page is clean, remove the execution trigger from that page and replace it with a write trigger
- No future attempts to execute that page will trigger the page execution trigger
- If the page is modified, the page write trigger will be executed and we'll again scan the page.

Step 2: Prepare the Agent

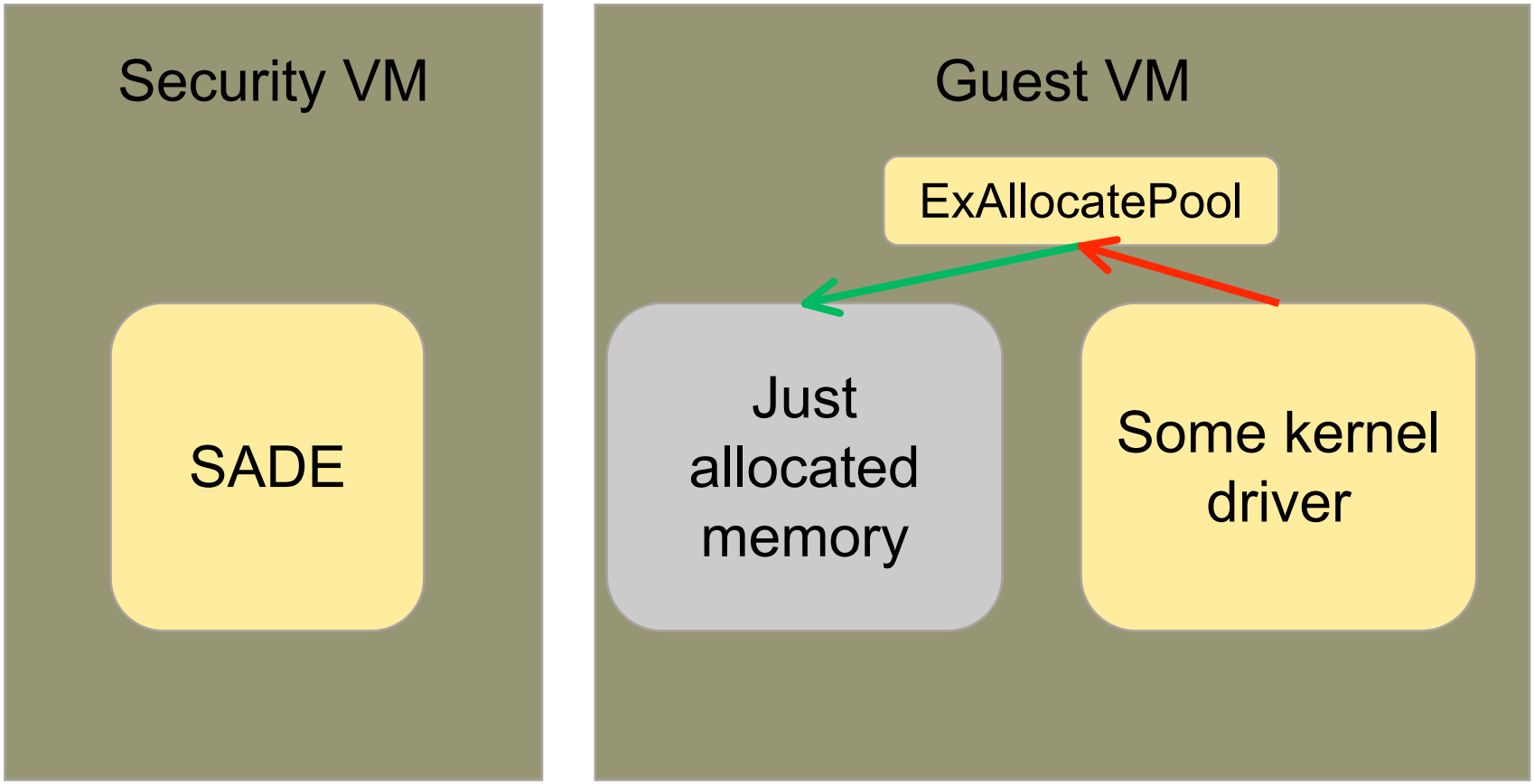
- Read the agent driver into memory from disk
- This a Windows Portable Executable (PE) format driver
- The imports of the agent need to be resolved.
  - Read the import table of the agent.
  - For each API used (such as DbgPrint), we need to find the runtime address of the API in the guest.
  - Locate the export tables of the guest kernel (NTOSLRKNL and HAL).

Step 3: Find Memory for the Agent

- We need to inject the agent into the guest virtual machine
- Where should we put the agent? None of the memory inside the guest virtual machine “belongs” to us
- Use a trick: put a page execution trigger on `ExAllocatePool`
  - The equivalent of `kmalloc` on Windows
  - When EIP register (the instruction pointer) is at the RET instruction, look at the functions return value (in the EAX register)
  - This points to memory just allocated, but not yet used
  - Temporarily hijack this memory, inject bootstrap code to allocate “permanent” memory.
  - After bootstrap code finishes, restore control to `ExAllocatePool`

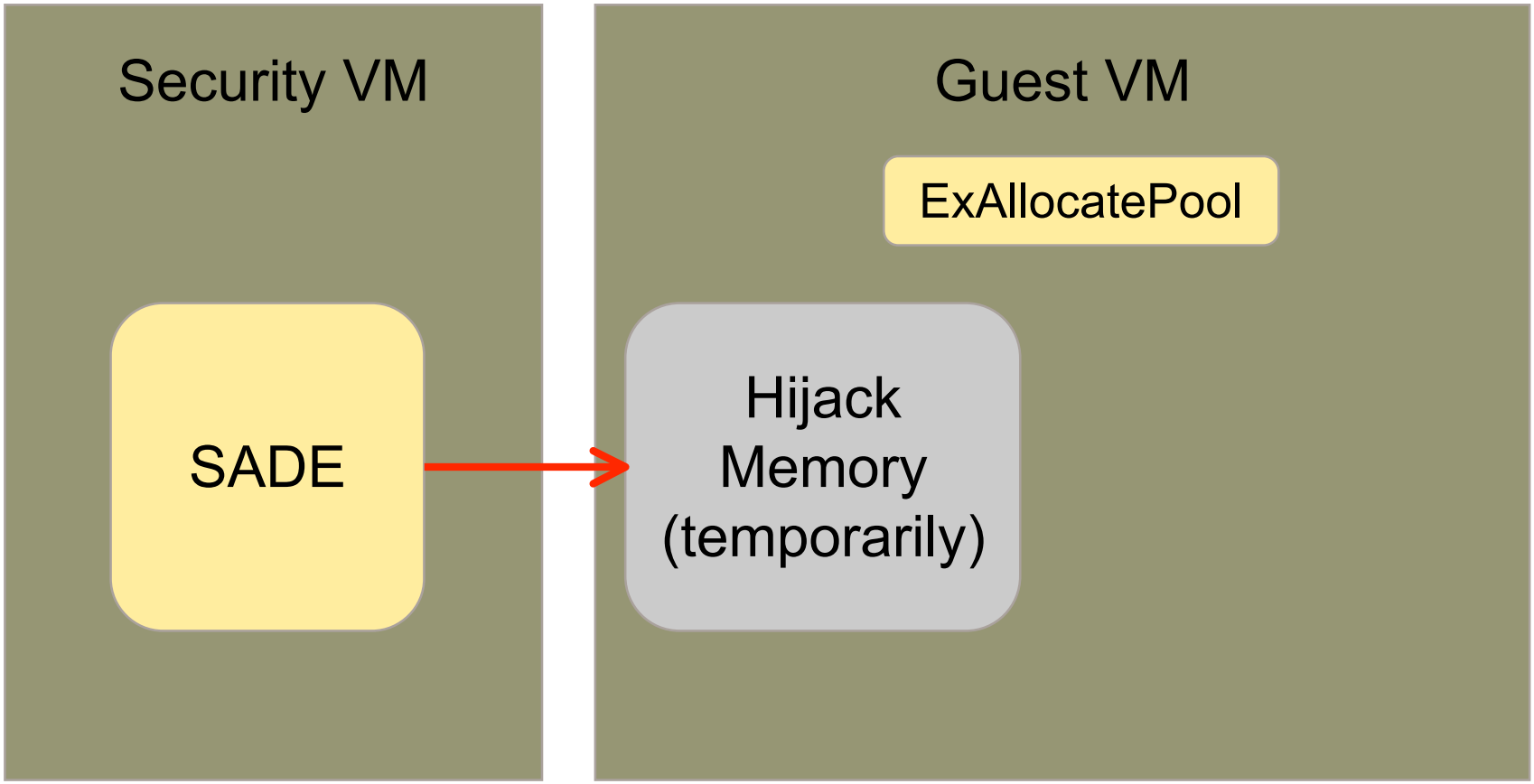
Step 3: Find Memory for the Agent

#1: Detect when ExAllocatePool API is about to return



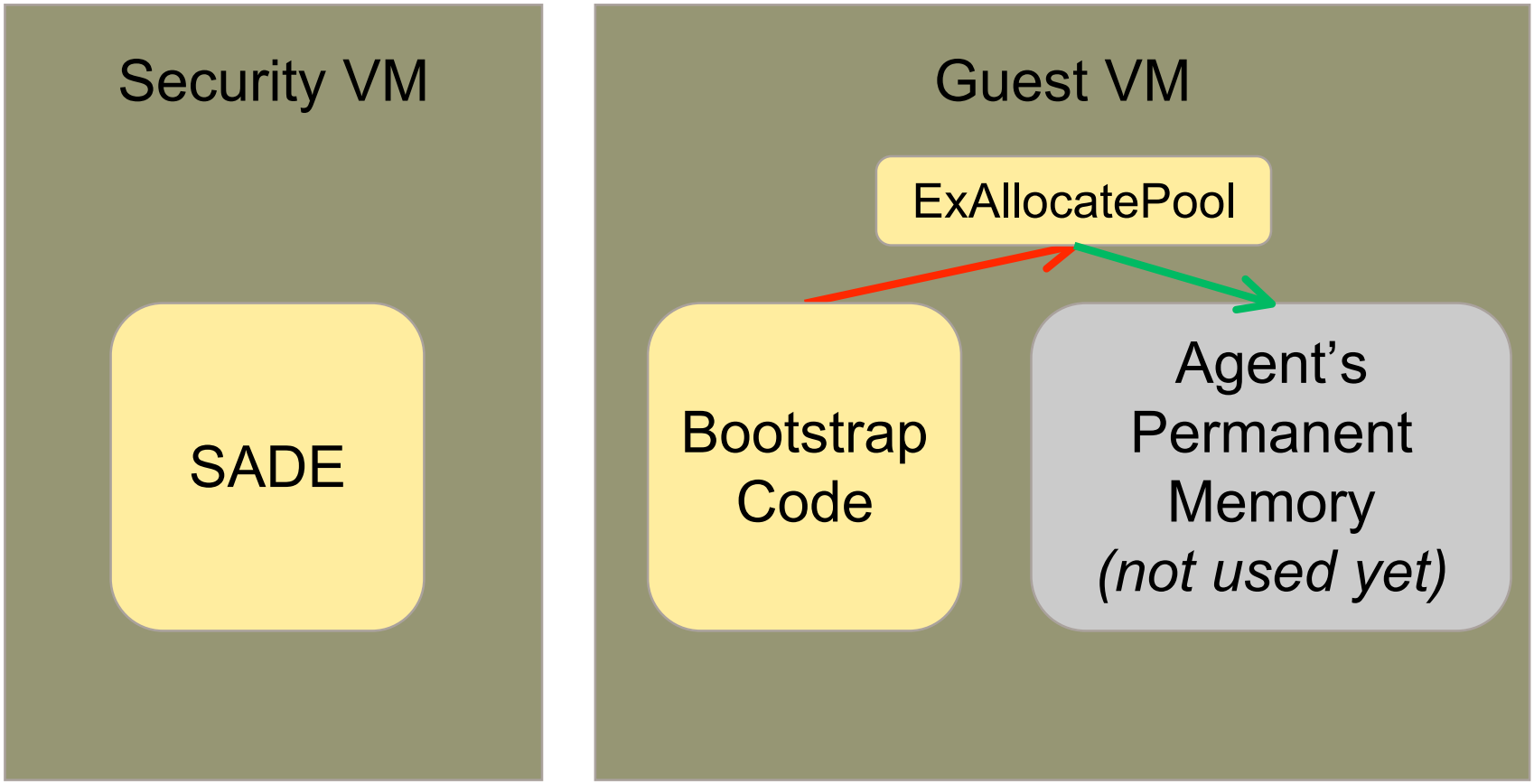
Step 3: Find Memory for the Agent

#2 Insert our bootstrap (allocation) code into the hijacked memory



Step 3: Find Memory for the Agent

#3 Allocate agent's permanent memory using bootstrap



Step 4: Invoke the Agent

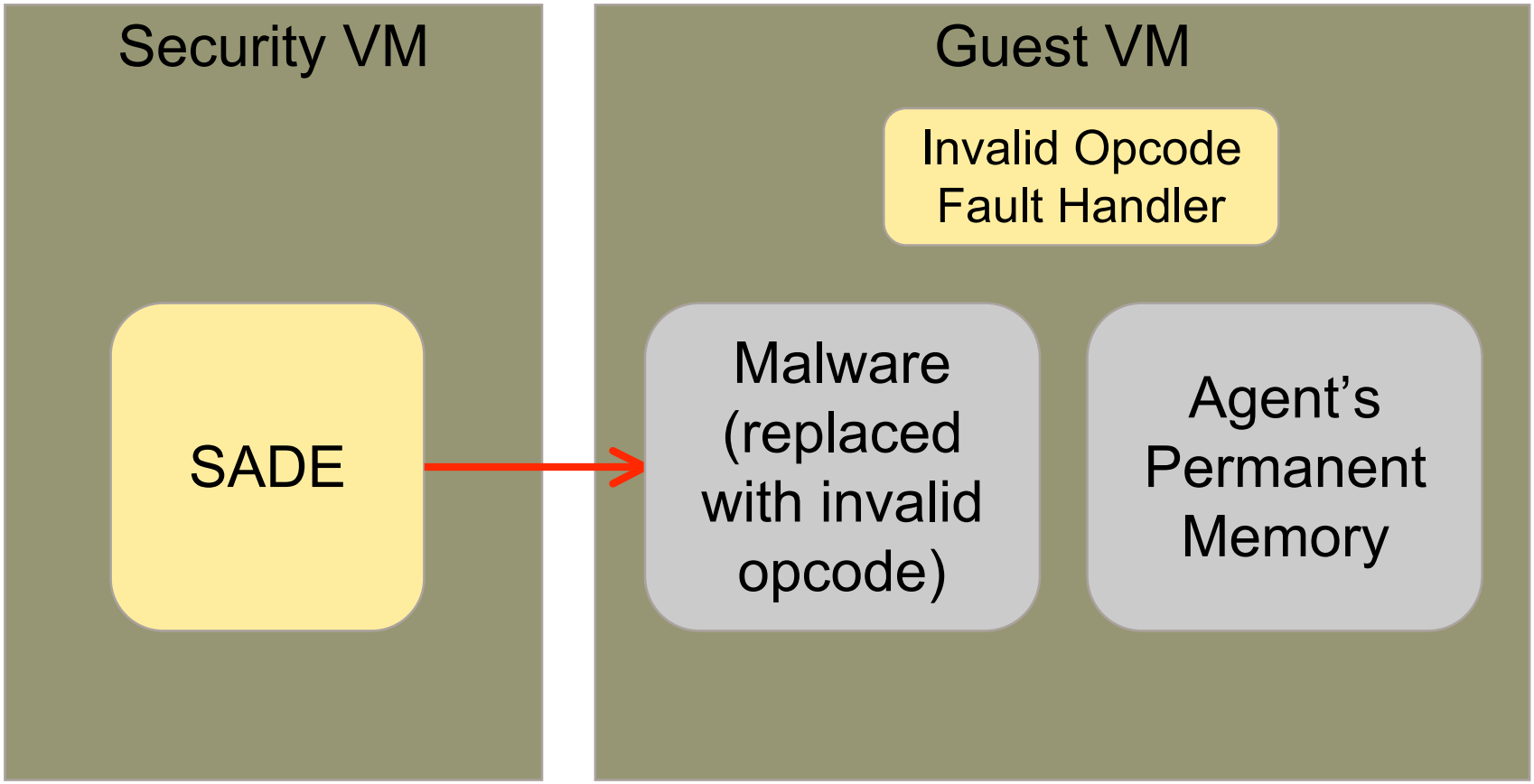
- At the time the malware is detected, there are two possible execution states:
  - If the malware was running in ring 0 (a kernel mode rootkit), we can just directly change EIP to point to the where the injected agent driver is located.
  - If the malware is running at ring 3 (which is usually the case), this won't work. User-mode code obviously cannot access kernel-mode APIs or memory. In this case, we need to use a trick to force an immediate transition to ring 0
- We force a fault (CPU exception) to force this transition

Step 4: Invoke the Agent

- Insert an invalid opcode at EIP (points into the malware page).
- Place an execution trigger on the invalid opcode fault handler.
- When the guest VM resumes execution, instead of executing the malware, it will immediately produce an invalid opcode fault.
- Now the guest is running at ring 0, change EIP to point to the agent's code

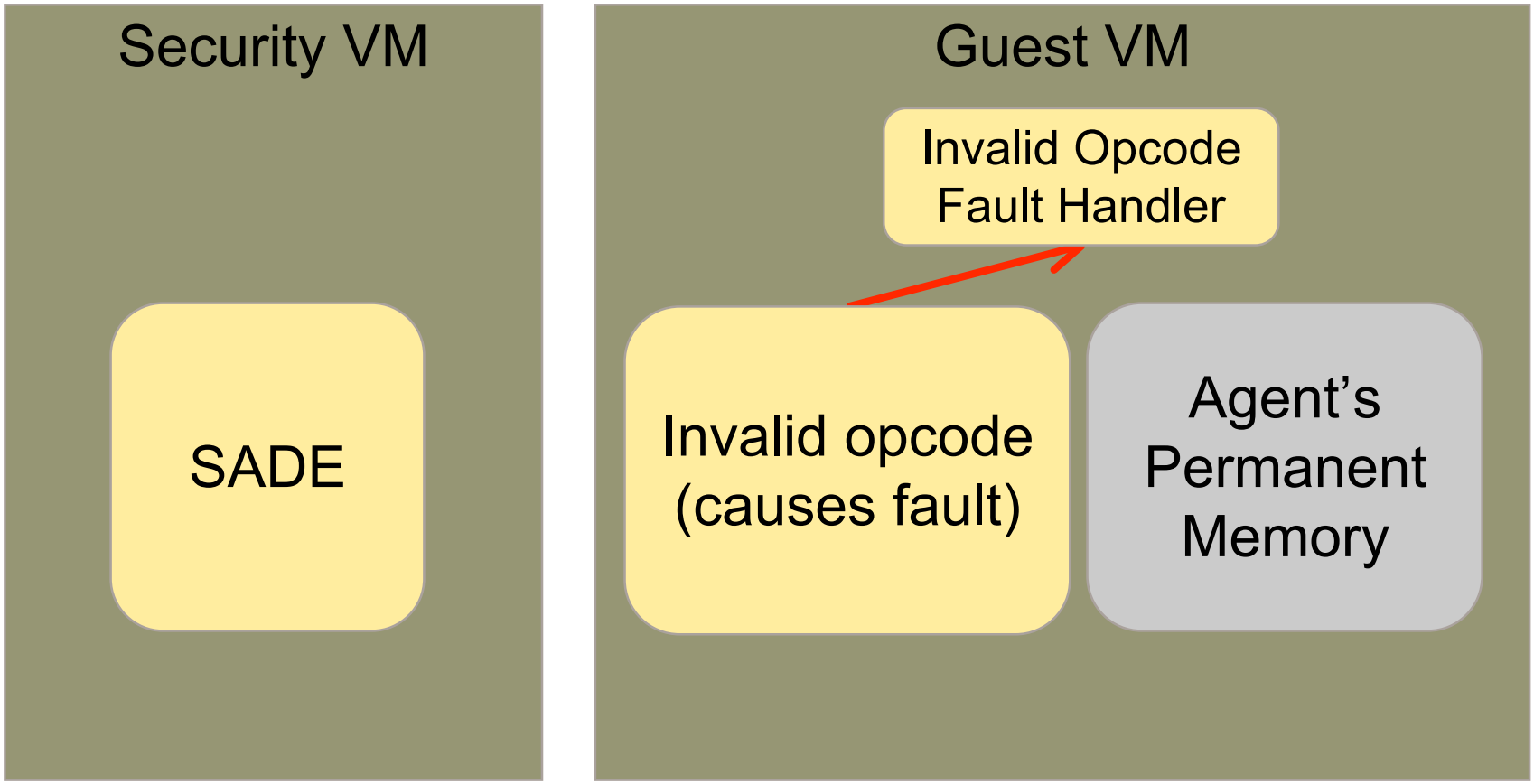
Step 4: Invoke the Agent

- Overwrite the malware code with an invalid opcode



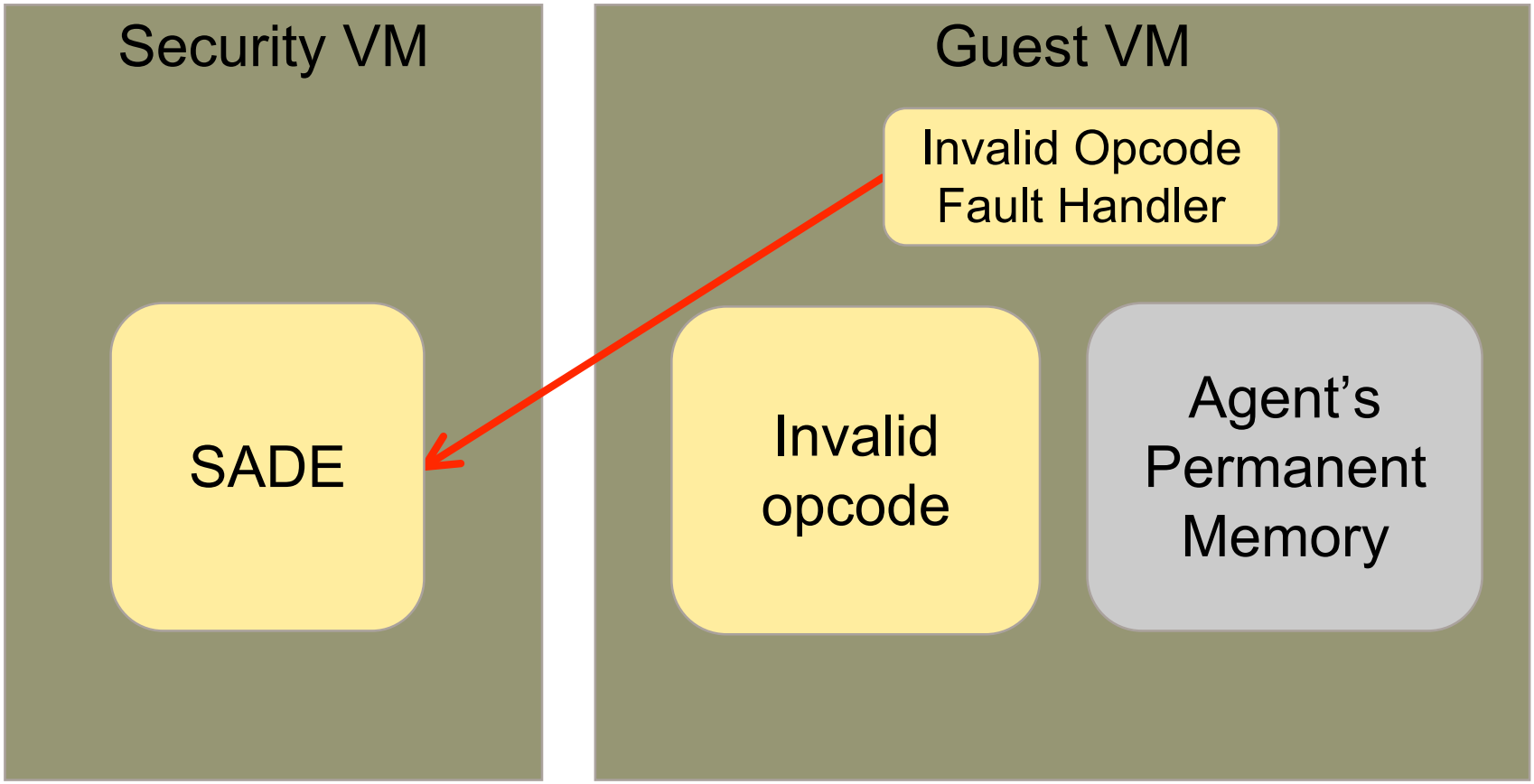
Step 4: Invoke the Agent

- Guest VM causes an invalid opcode fault



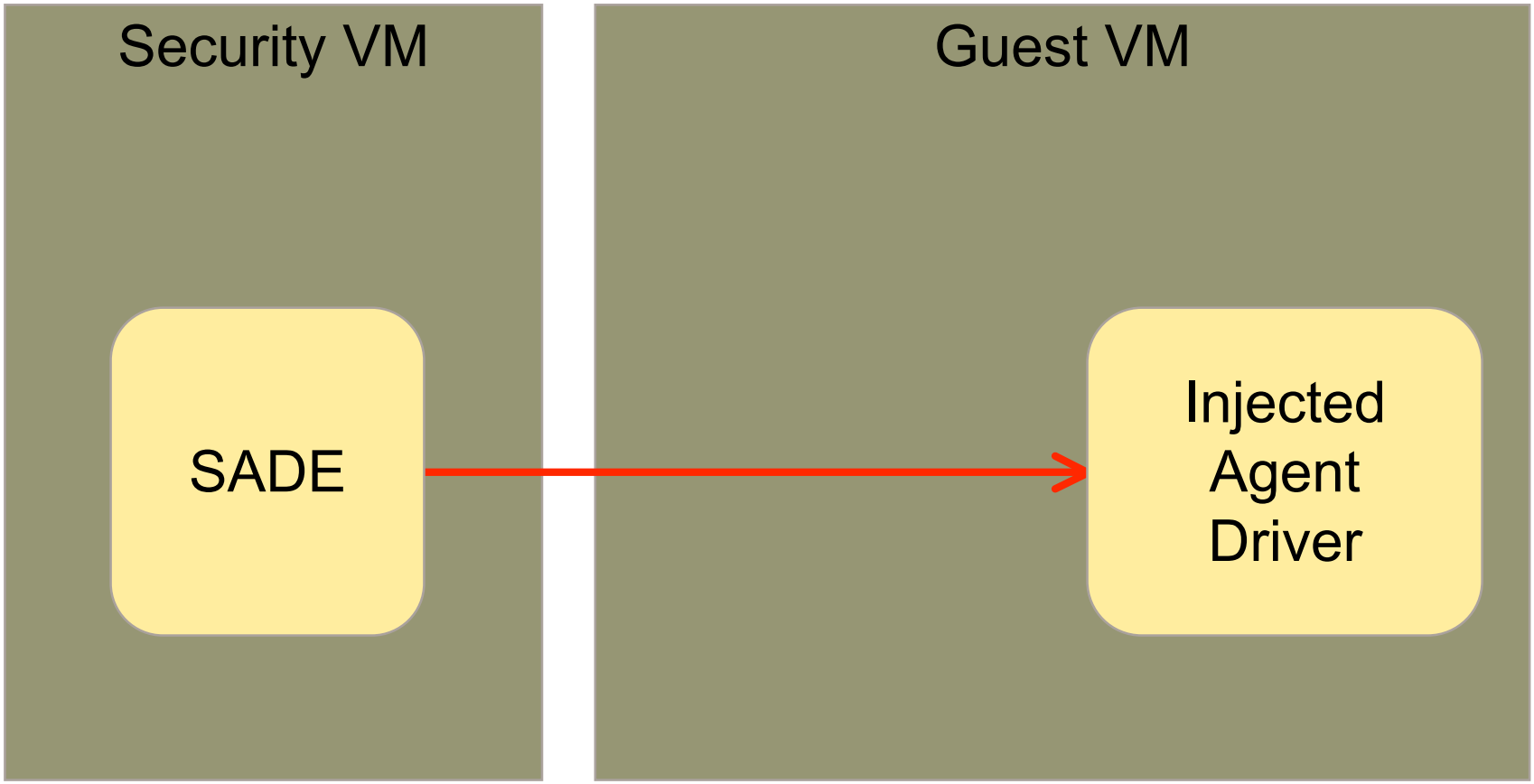
Step 4: Invoke the Agent

- Execution event on invalid opcode handler triggered



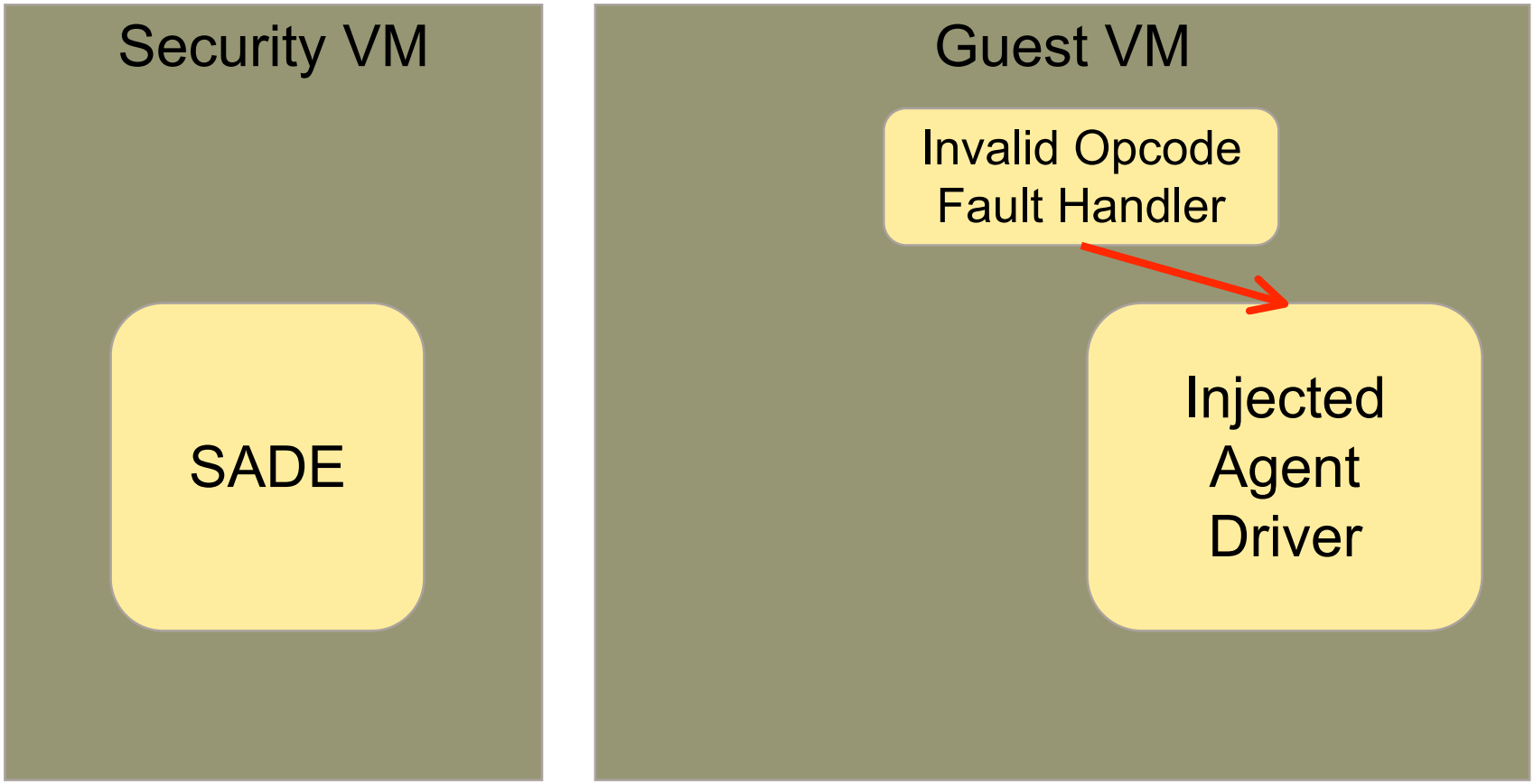
Step 4: Invoke the Agent

- Inject agent code into the allocated memory



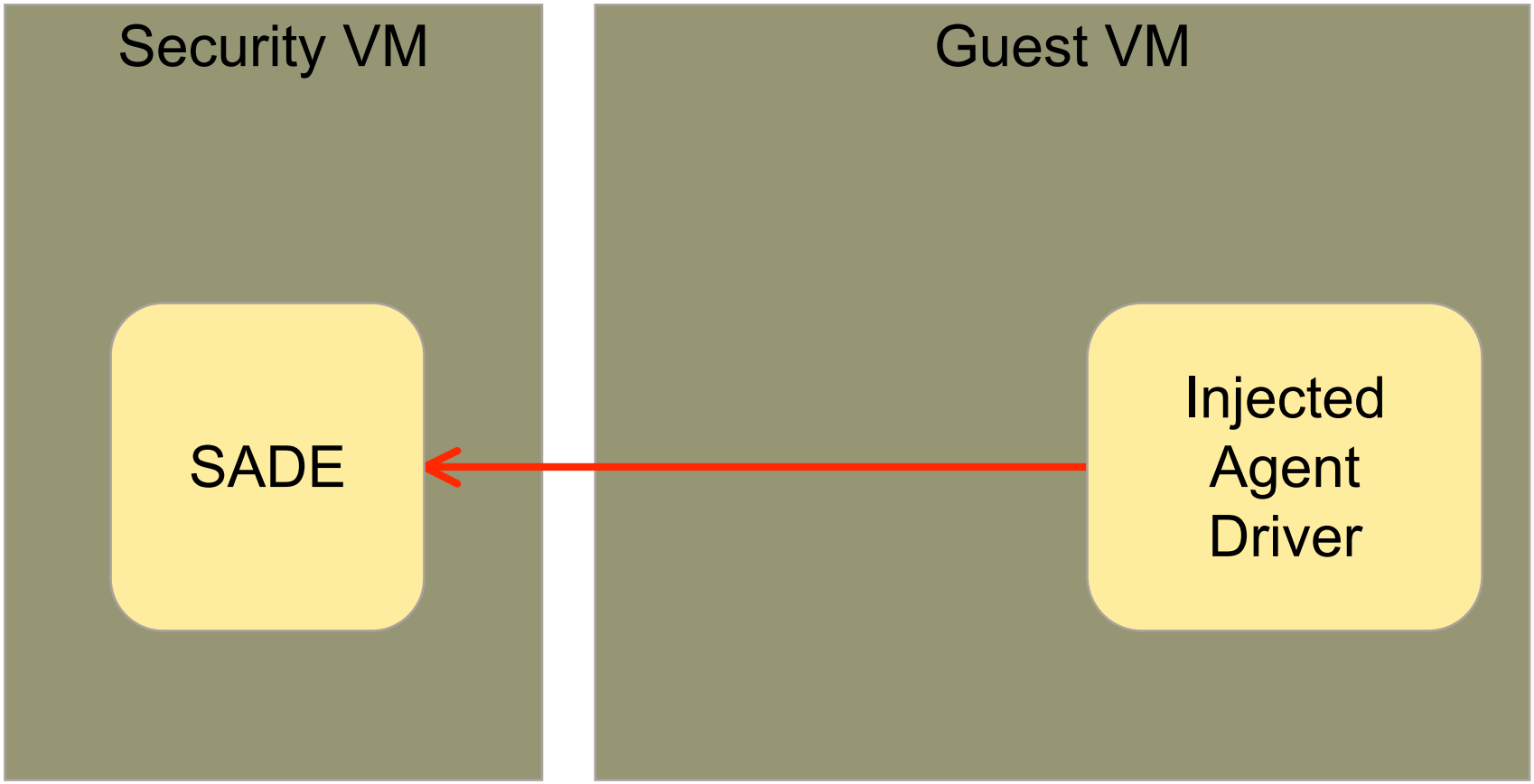
Step 4: Invoke the Agent

- Invalid opcode fault handler is hijacked to execute agent



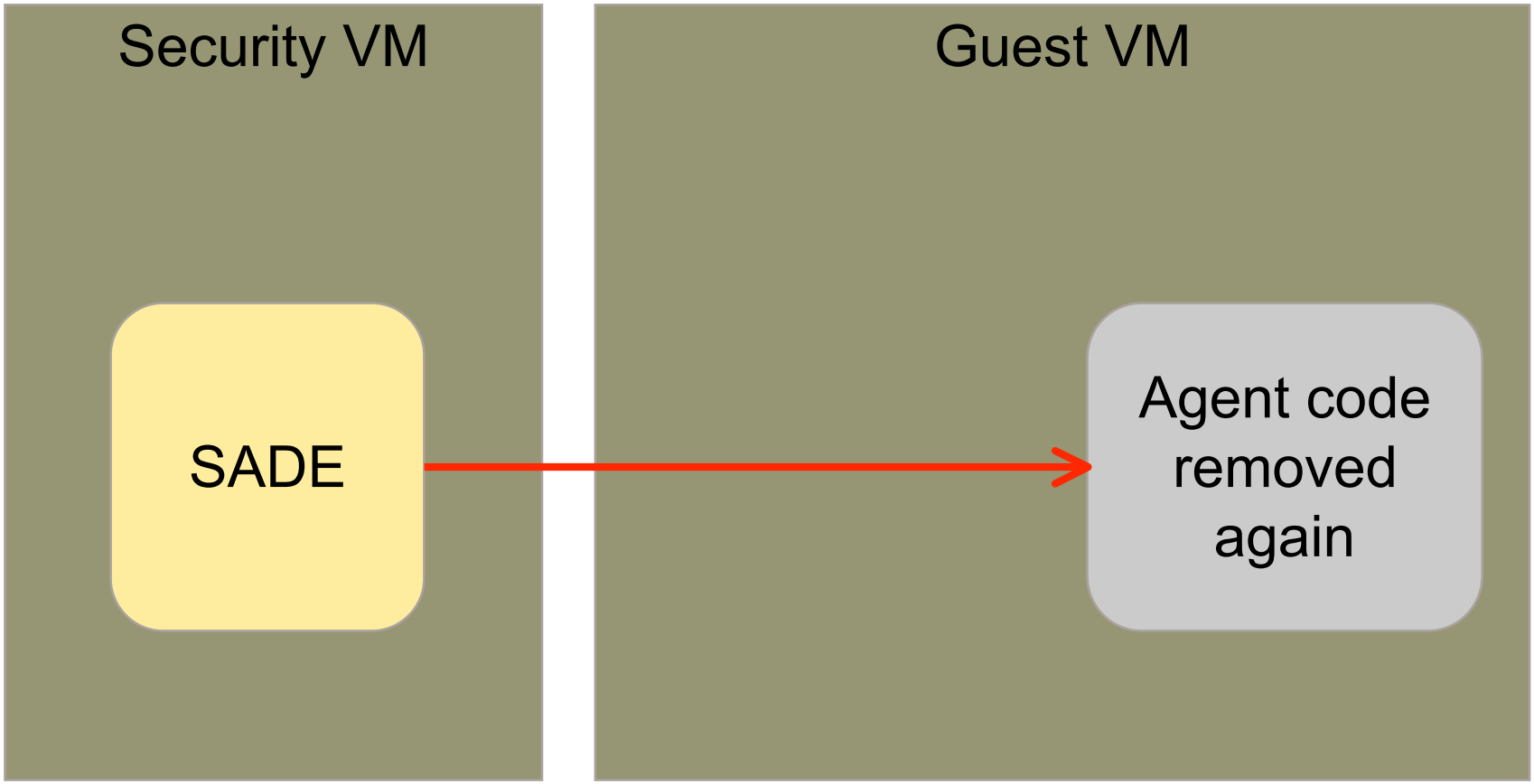
Step 5: Return from the Agent

- Hypervisor detects when agent is finished



Step 5: Return from the Agent

- Machine is back to the original state (no agent present)



Demo

- Run W32.Gammima virus
- Detected by SADE
- Inject remediation driver
- Remediation driver calls
  `NtTerminateProcess(NtCurrentProcess())`

Performance

- Startup (1 time cost)
  - 17 ms: Discover NTOSKRNL and parse export table
  - 1 ms: Install bootstrap code (calls `ExAllocatePool`)
  - 1.1 ms: Execute bootstrap code
  - 2 ms: Relocate and load agent driver
- Inject and execute agent driver (for each malware event)
  - 4.7 ms: Trigger and handle invalid opcode exception
  - 0.1 ms: Ring3-to-ring0 transition
  - 1 ms: In-guest function execution
- Restore original state (for each malware event)
  - 1.9 ms: Restore original program context
  - 0.1 ms: Ring0-to-ring3 transition
- Total time (for each malware event): 7.8 milliseconds

- Disclaimer: These numbers are specific to our prototype's implementation. This is not a VMware benchmark.

Closing Remarks

- The prototype is finished, stable, and works like a charm! This prototype:

- Can be used to inject a legacy driver into the guest.
  - It can handle a “hostile” guest virtual machine.
  - It doesn’t eliminate the possibility of the agent being detected/disabled, but it makes the window very small

- Significantly raises the bar for malware running in a virtualized environment to detect or disable security agents
  - This prototype demonstrates one of the security benefits of virtualization over legacy hardware

Questions?
Thanks!

matthew_conover
@
symantec.com