AVAR 2009@kyoto

FFR GreenKiller

Automatic kernel-mode malware analysis system

Junichi Murakami,
murakami@fourteenforty.jp
Director of Research,
Fourteenforty Research Institute, Inc.

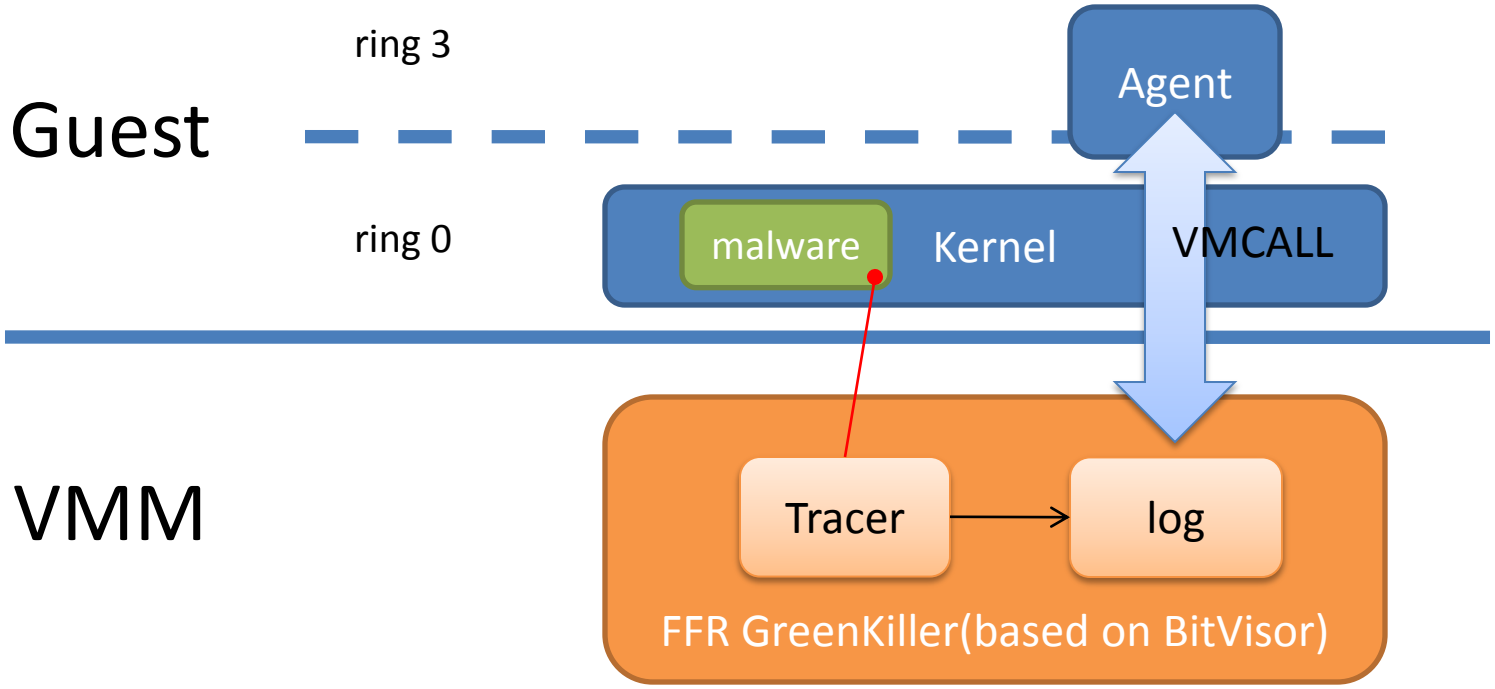
Who am I?

- MURAKAMI Junichi(*村上 純一*)
  - Reverse Engineer, both Windows and Linux kernel development
  - BlackHat speaker at US and Japan 08
  - Security & Programming Camp Instructor(2006 -)
  - ITPro ホワイトハッカー道場(White Hacker Dojo)
    http://itpro.nikkeibp.co.jp/article/COLUMN/20070927/283156/

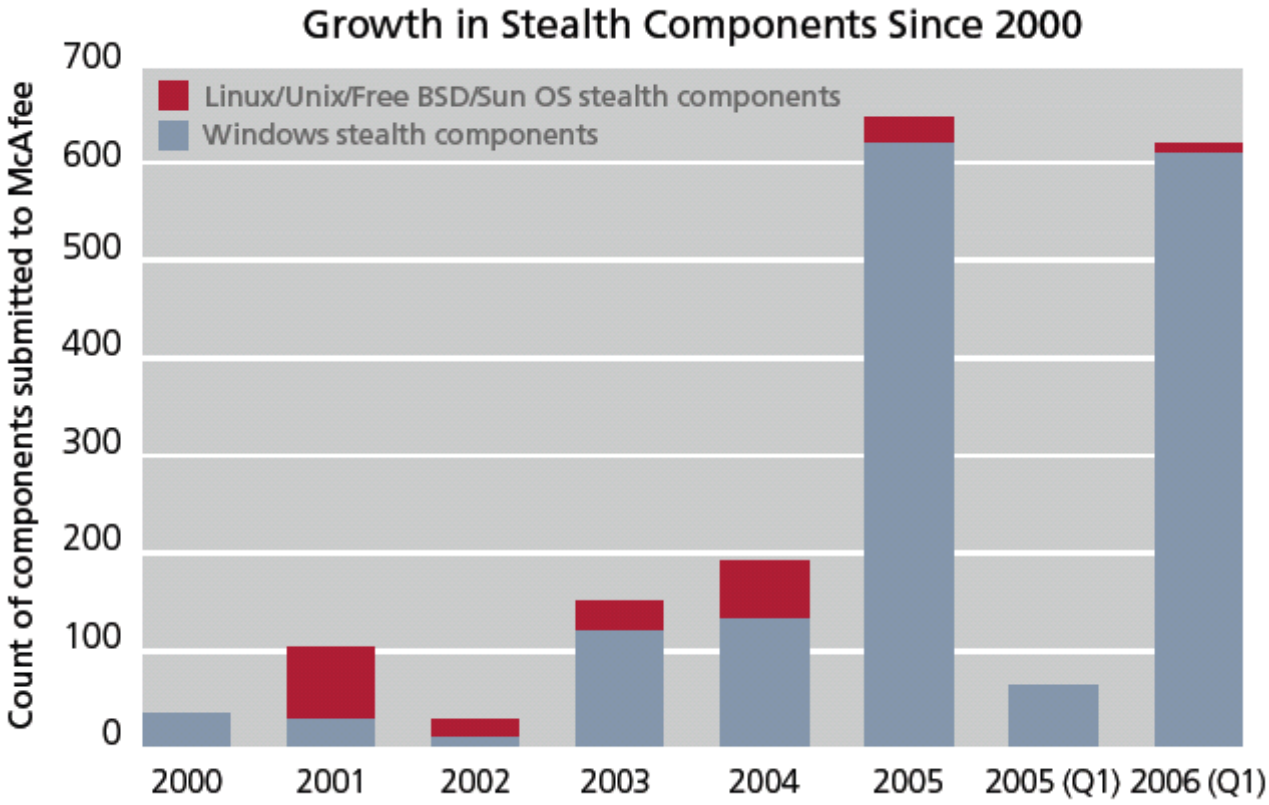
FFR GreenKiller

version 0.0.1

Analyzing malicious driver from VMM



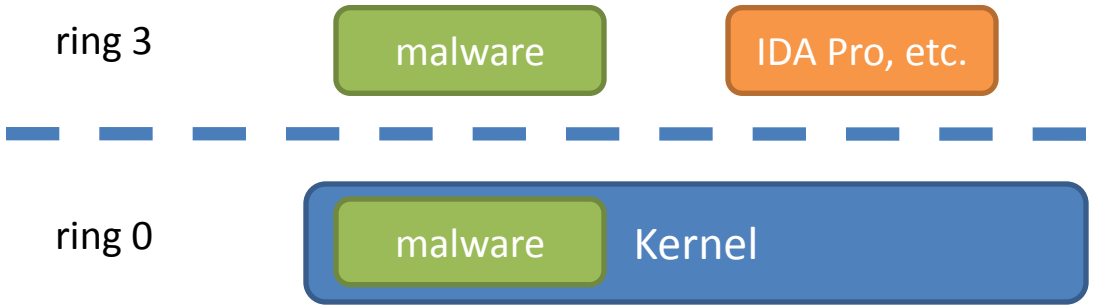
Background



Rootkits, Part1of 3: The Growing Threat(McAfee Avert Labs)
http://www.mcafee.com/us/local_content/white_papers/threat_center/wp_akapoor_rootkits1_en.pdf

Why kernel-mode (is troublesome) ?

User-mode malware is a process,
Kernel-mode malware is a part of system

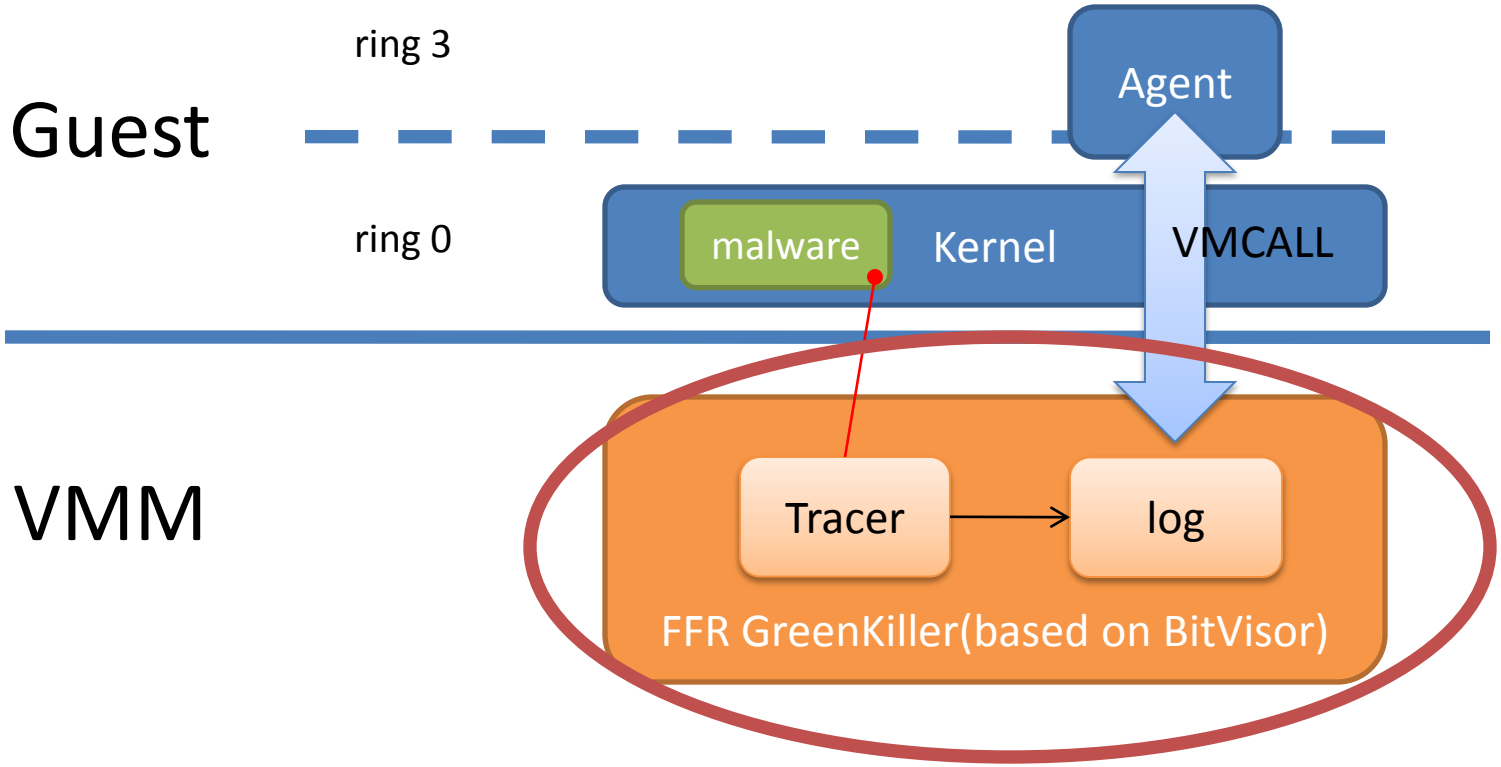


BSOD is very welcomed ☺

FFR GreenKiller

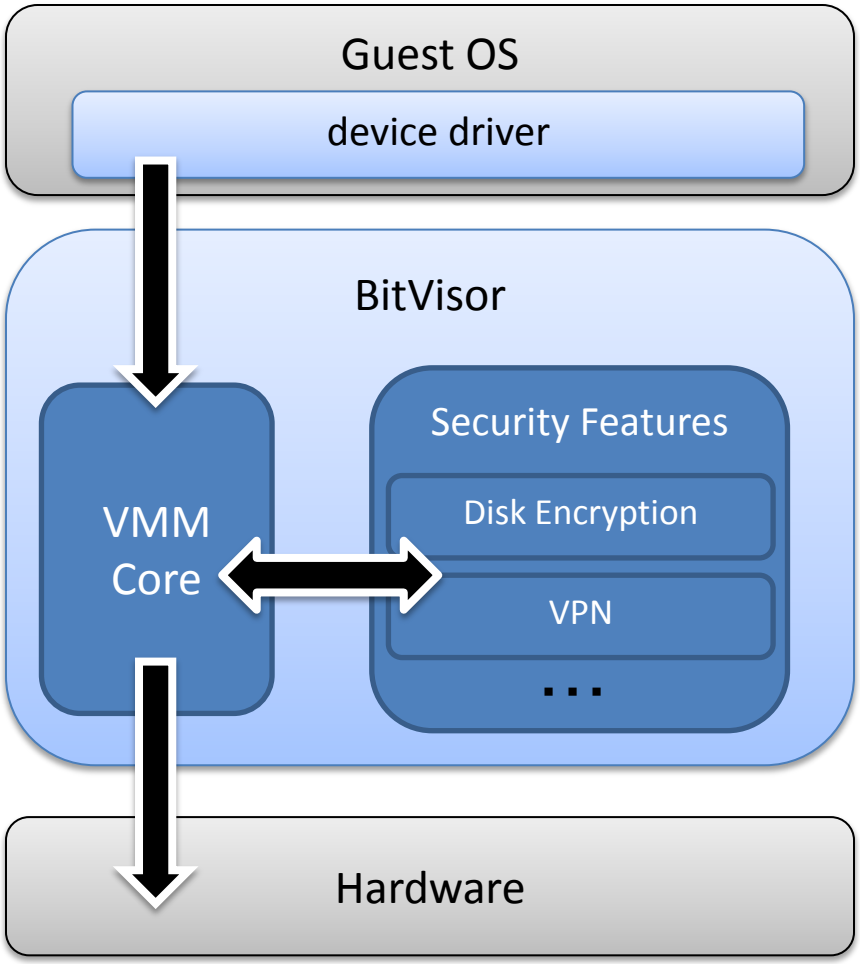
version 0.0.1

Analyzing malicious driver from VMM

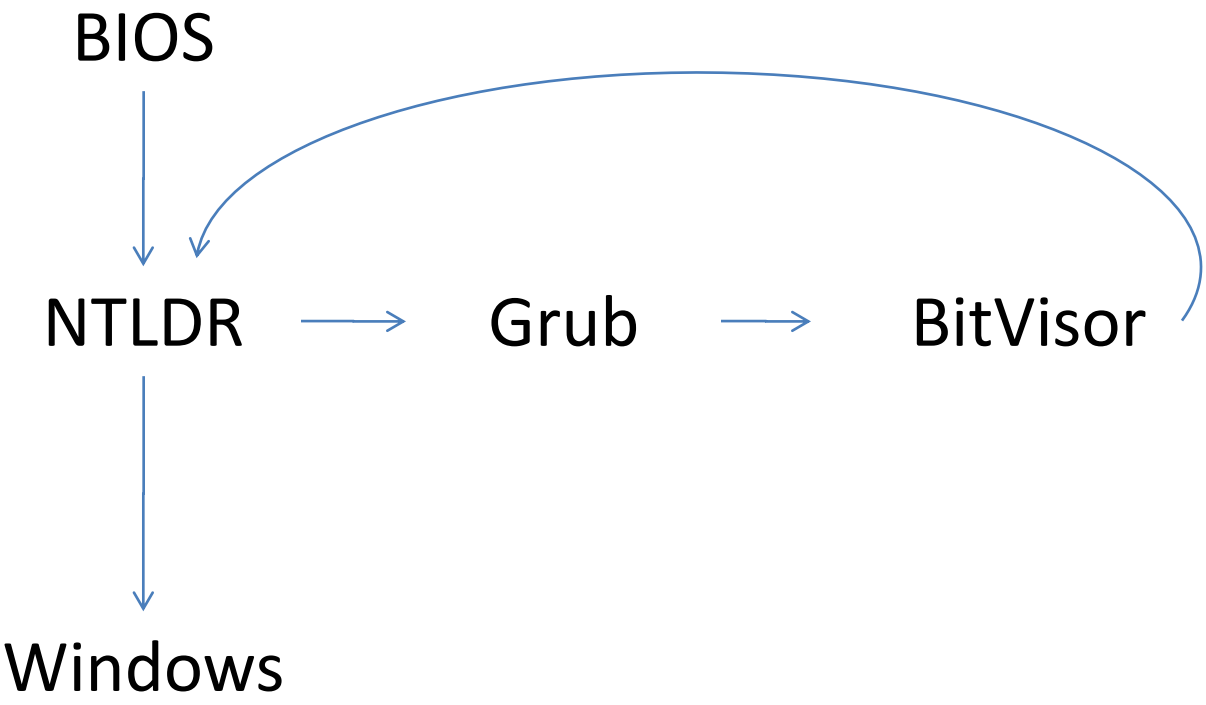


BitVisor - http://www.bitvisor.org

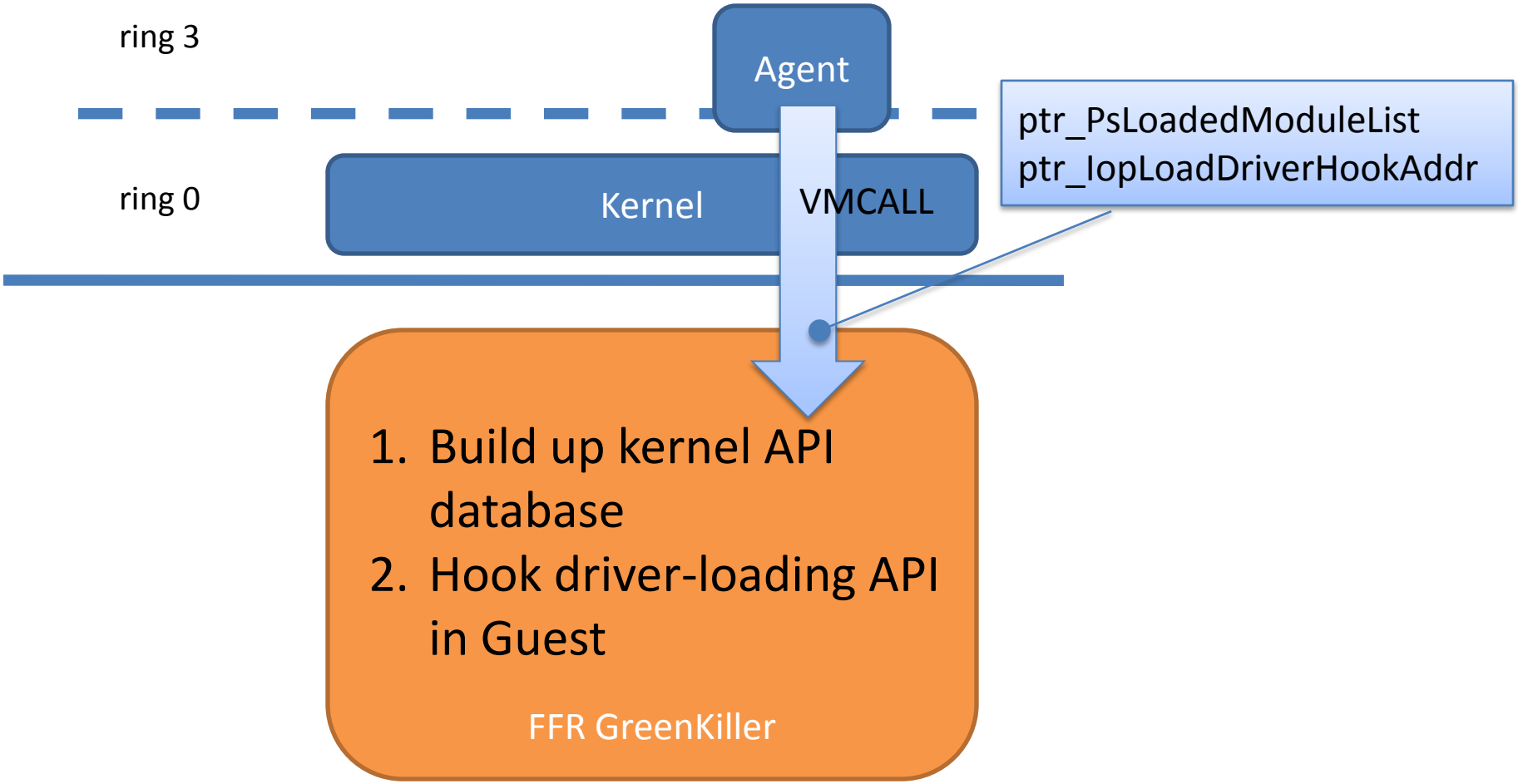
- Open-source VMM software (BSD License)
- Secure VM Project (National Project in Japan)
- Intel-VT and AMD-v supported
- GreenKiller uses BitVisor as a VMM framework ☺



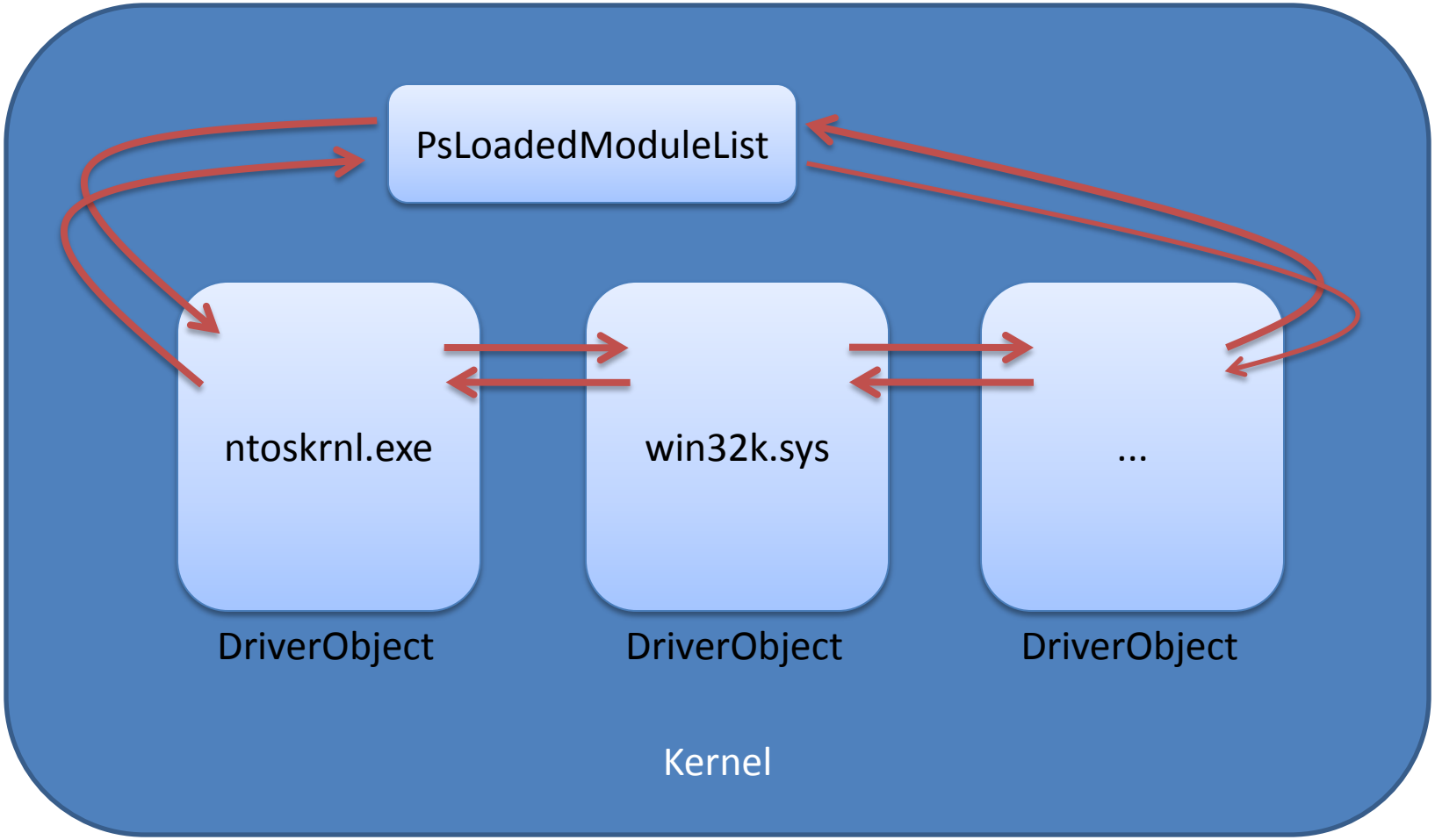
BitVisor (cont.)



Initialization

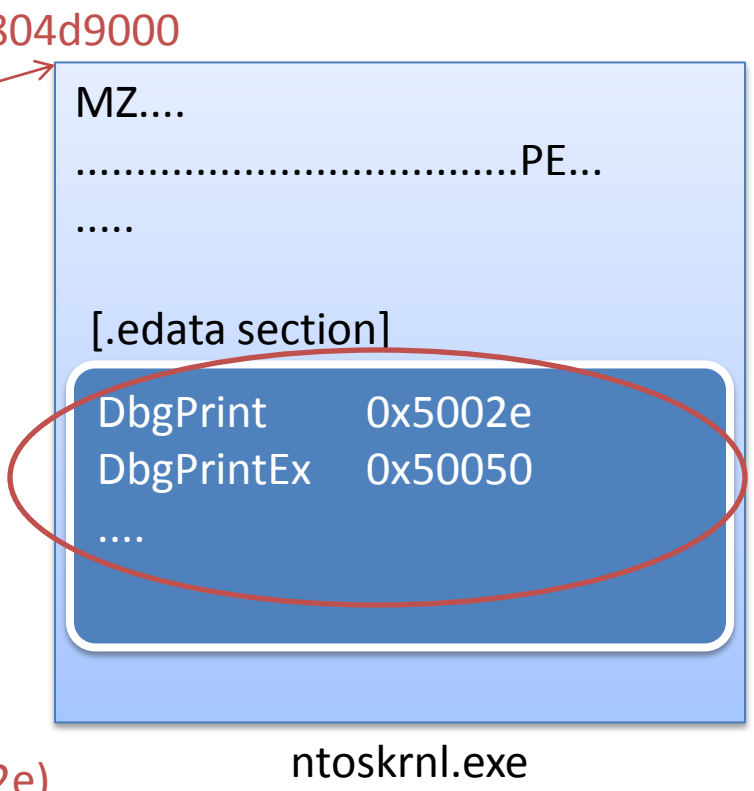


PsLoadedModuleList



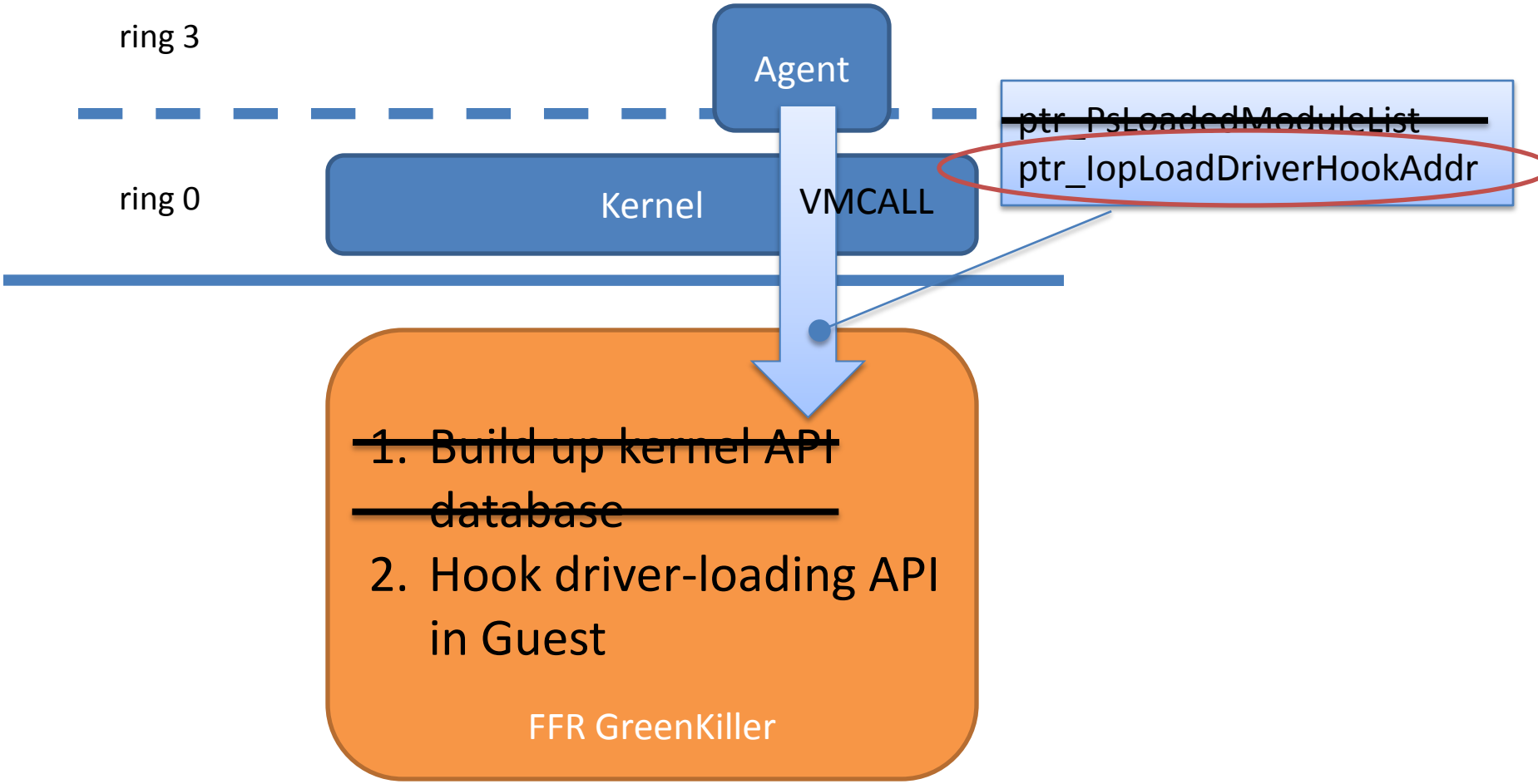
DriverObject

```
kd> dt nt!_DRIVER_OBJECT
   +0x000 Type             : Int2B
   +0x002 Size             : Int2B
   +0x004 DeviceObject     : Ptr32 _DEVICE_OBJECT
   +0x008 Flags            : Uint4B
   +0x00c DriverStart      : Ptr32 Void
   +0x010 DriverSize       : Uint4B
   +0x014 DriverSection    : Ptr32 Void
   +0x018 DriverExtension  : Ptr32 _DRIVER_EXTENSION
   +0x01c DriverName       : _UNICODE_STRING
   +0x024 HardwareDatabase : Ptr32 _UNICODE_STRING
   +0x028 FastIoDispatch   : Ptr32 _FAST_IO_DISPATCH
   +0x02c DriverInit       : Ptr32     long
   +0x030 DriverStartIo    : Ptr32     void
   +0x034 DriverUnload     : Ptr32     void
   +0x038 MajorFunction    : [28] Ptr32     long
```

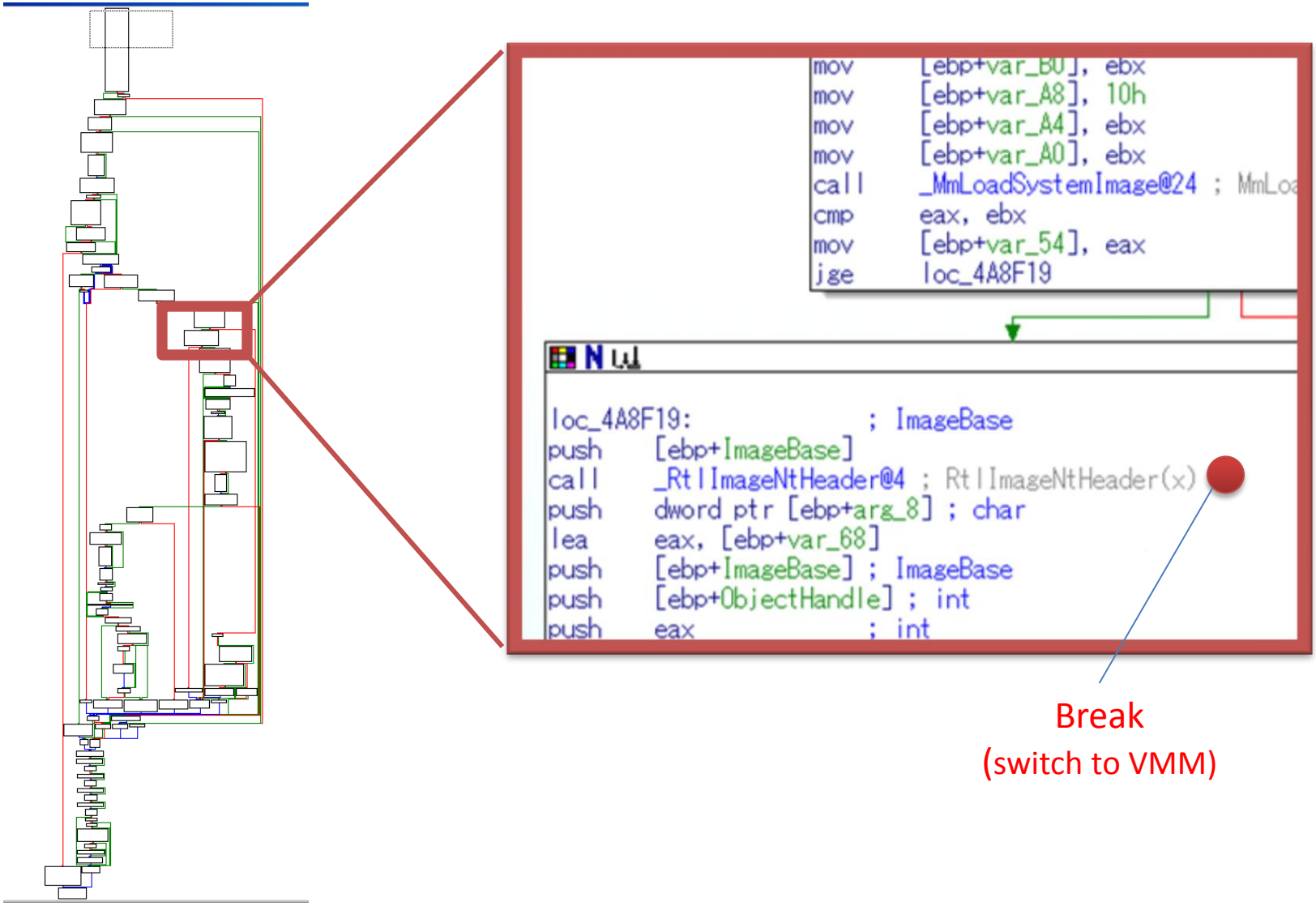


DbgPrint 0x8052902e (0x804d9000 + 0x5002e)
DbgPrintEx 0x80529050 (0x804d9000 + 0x50050)
...
...

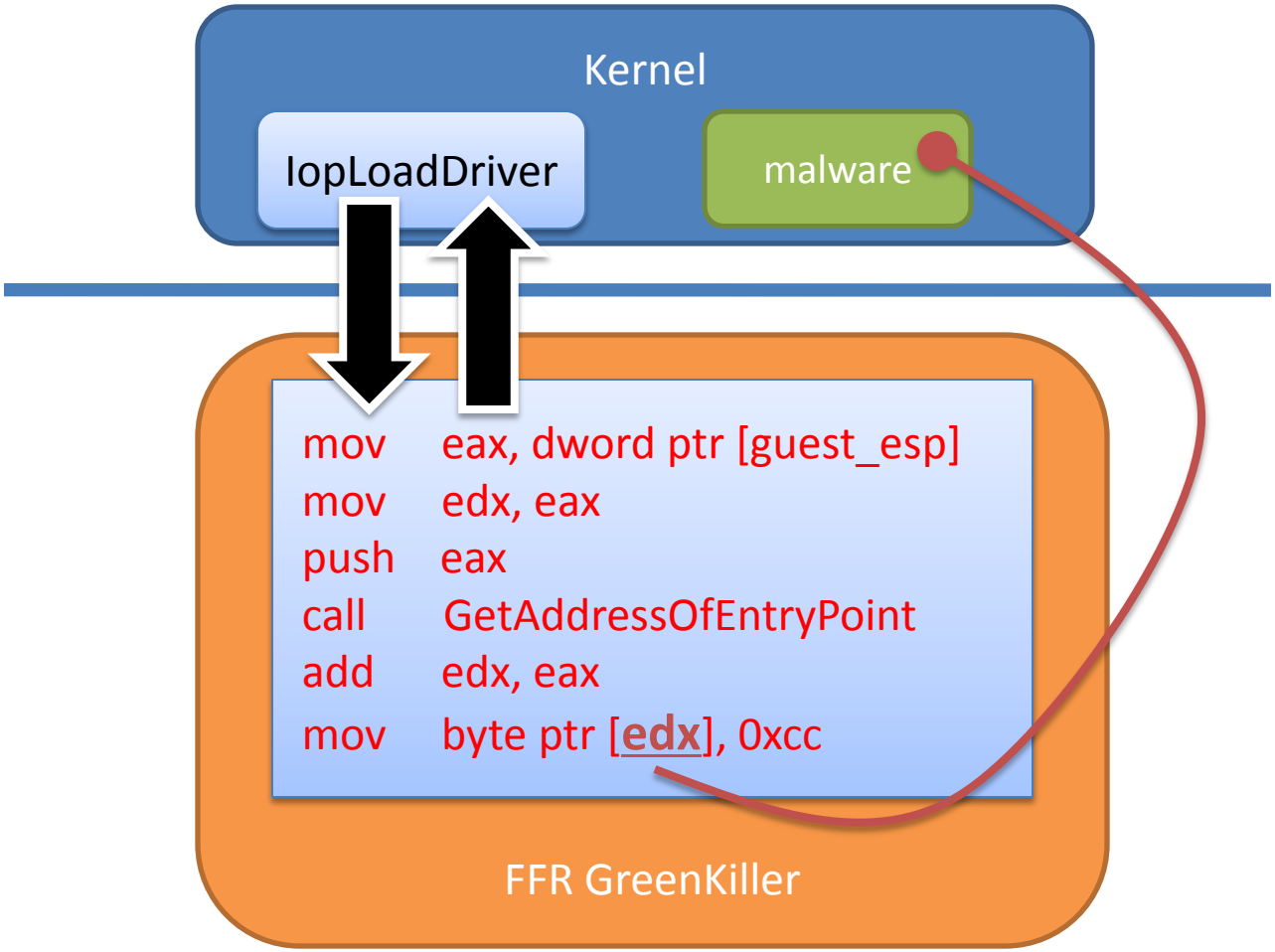
Initialization



IopLoadDriver



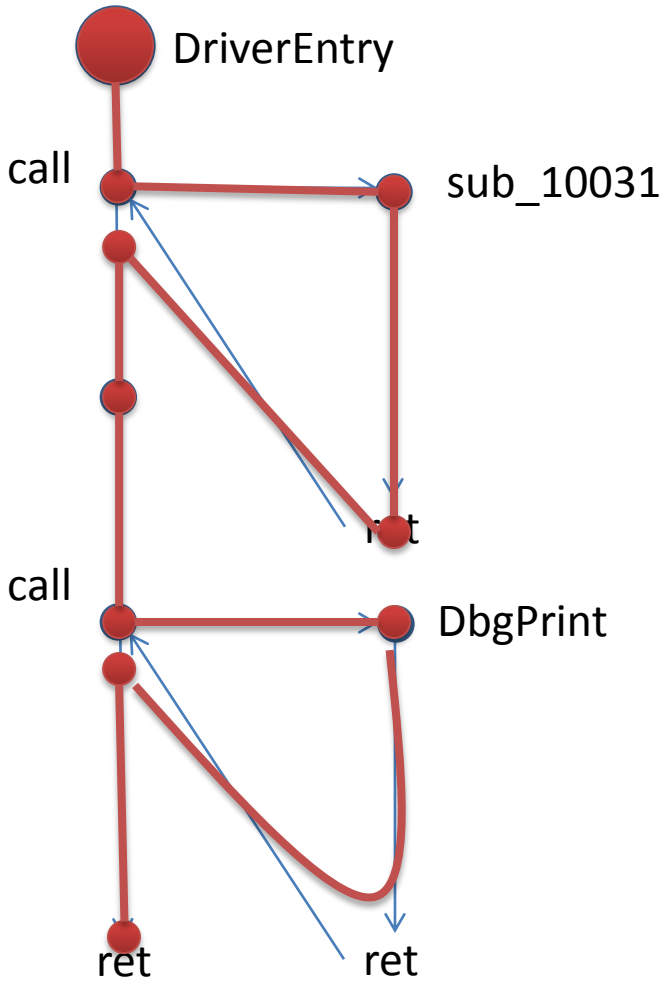
Break at DriverEntry



Software BP vs. Hardware BP

- Software BP
  - Replace original byte with int3(0xcc)
    - Processor generates an #BP when int3 is hit
  - No limit on number of BPs
- Hardware BP
  - Use Debug Register(DR0-DR3)
  - Needless to modify original code
  - Only 4 BPs simultaneously

Code Tracing

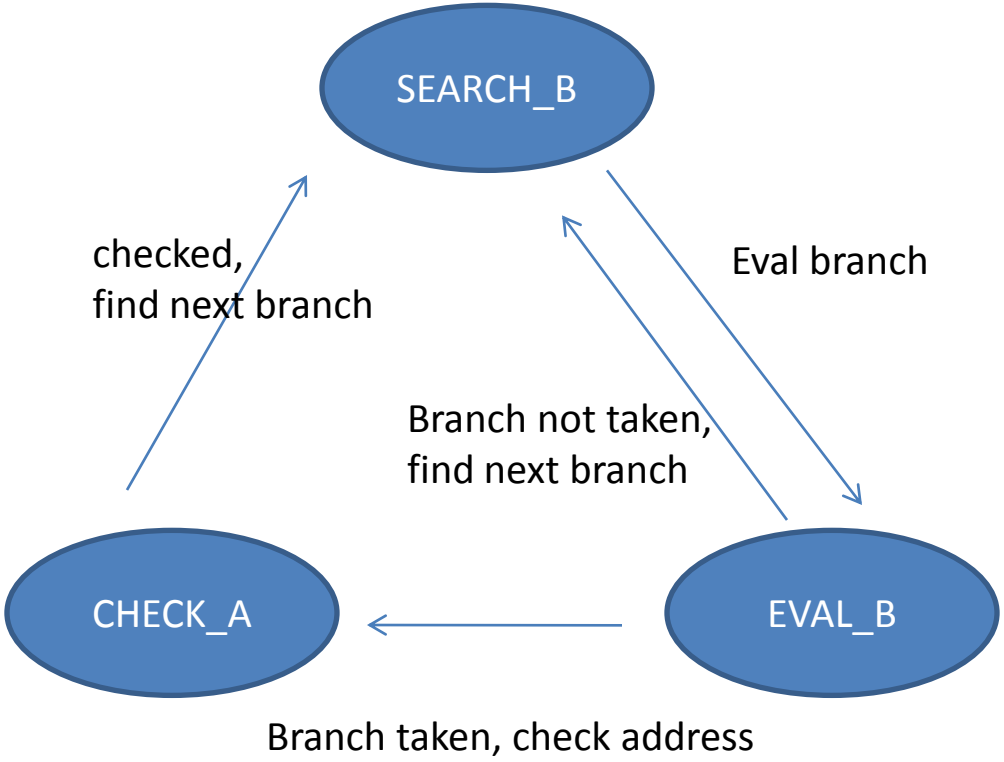


A) Insert '0xcc' into closest branch by disassembling the code from DriverEntry

B) Execution is suspended on the branch
Check whether execution takes the branch according to EFLAGS's value
- If true: Insert '0xcc' into branched addr.
- If not: Working the same as A.

C) Execution is suspended on the branched addr.
Check whether it is an entry of kernel API
- if true: Insert '0xcc' into retaddr
- if not: Working the same as A.

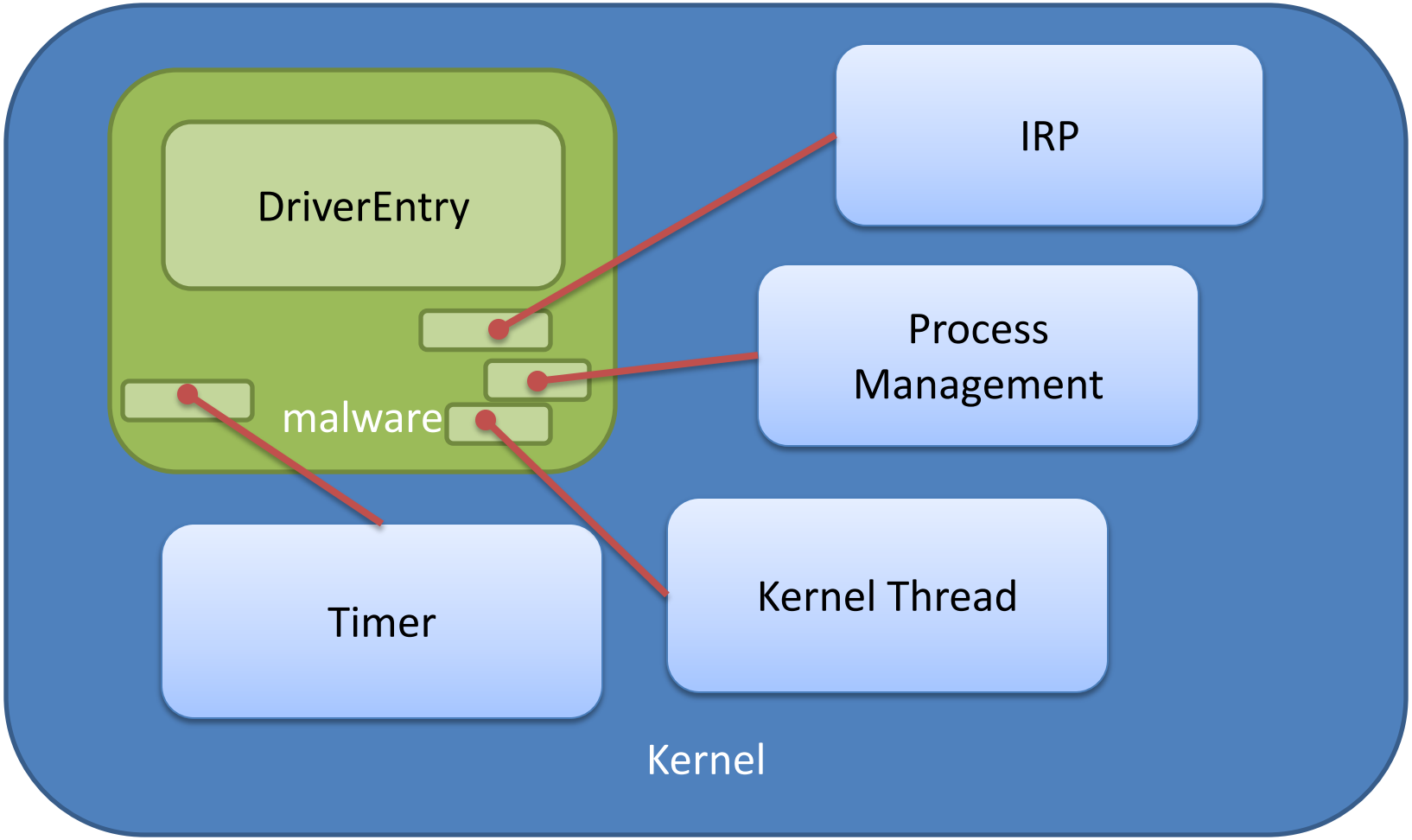
Definition of an FSM



That’s all?

- No, we have to consider multiple context for callbacks
- Driver is a plug-in for the kernel

Callbacks


DriverEntry
malware
IRP
Process Management
Timer
Kernel Thread
Kernel

Ex)PsSetCreateProcessNotifyRoutine

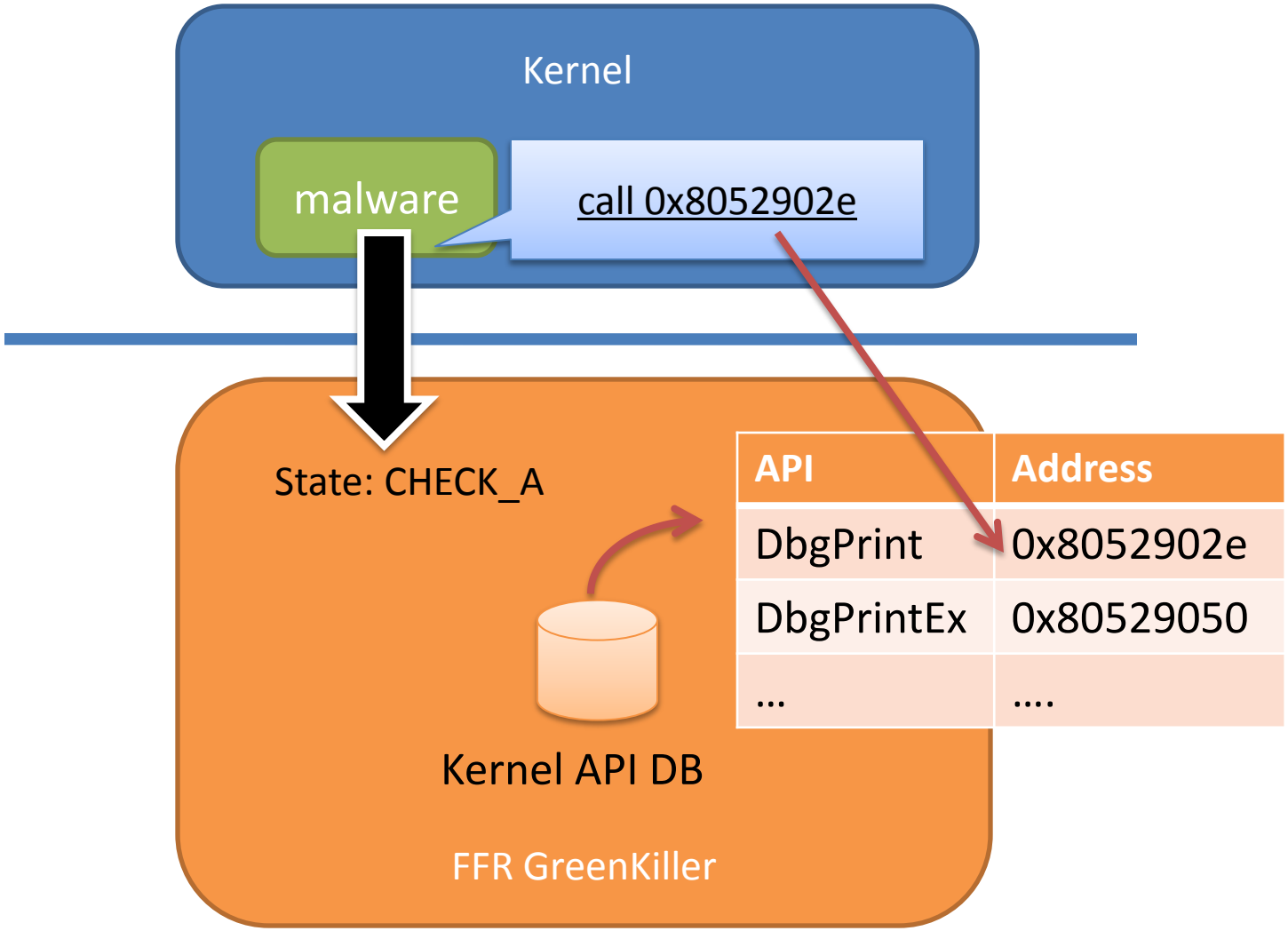
```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject,
                     IN PUNICODE_STRING RegistryPath)
{
    …
    PsSetCreateProcessNotifyRoutine(
        CreateProcessNotifyCallback, ; NotifyRoutine
        FALSE                        ; Remove
    );
    …
}

VOID CreateProcessNotifyCallback(IN HANDLE ParentId, IN HANDLE ProcessId,
                                 IN BOOLEAN Create)
{
    DbgPrint( “Pid %d is %s¥n”,
              ProcessId, Create ? “created” : “exit” );
}
```

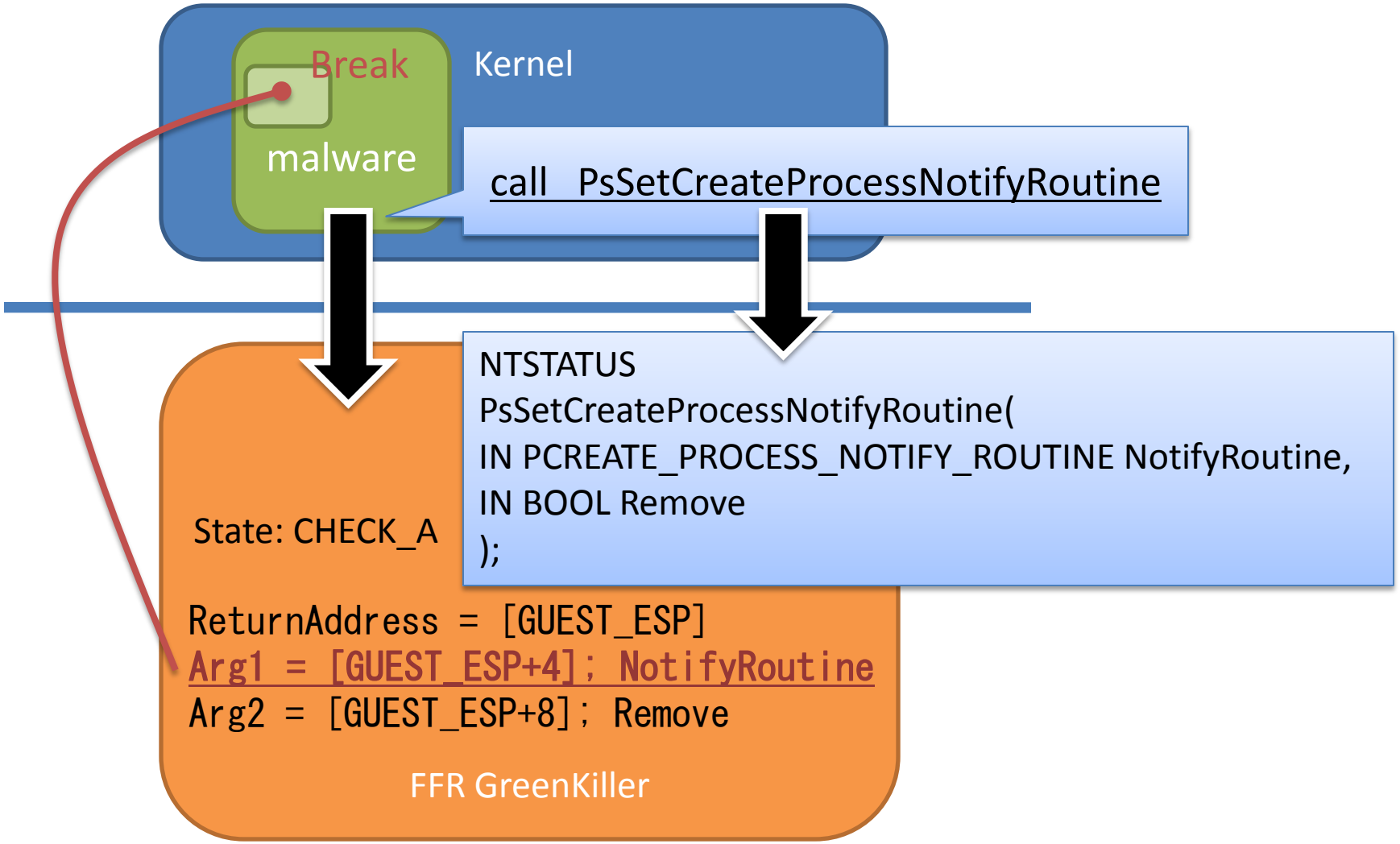
Callbacks(cont.)

- IRP Handlers
- DPC Routines
- Registry Callbacks
- Fs/Ndis/Tdi filters
- PsSetCreateProcessNotifyRoutine
- PsSetLoadImageNotifyRoutine
- PsCreateSystemThread
- etc.

Lookup Kernel API



Lookup Kernel API(cont.)



Output

[DriverEntry]
0xfd965012: DbgPrint(“DriverEntry: 0x%lx¥n”)
0xfd065054: IoCreateDevice(“¥¥Device¥m3z1”)
0xfd065067: IoCreateSymbolicLink(“¥¥Device¥m3z1”, “¥¥DosDevices¥m3z1”)
0xfd06508c: PsSetCreateProcessNotifyRoutine(CB#1)

[DriverUnload]
0xfd9650c0: IoDeleteSymbolicLink(“¥¥DosDevices¥m3z1”)
0xfd9650cc: IoDeleteDevice

[CB#1: PsSetCreateProcessNotityRoutine]
0xfd965134: PsGetCurrentProcessId
0xfd965140: DbgPrint(“CreateProcessCallback %d %d %d¥n”)
0xfd96710c: ExFreePool

Limitation

- Require Intel-VT
- Require an Agent in guest's ring3
  - Request for quick hack :)
- Limited API support
  - Callback, Argument recognization

Related Work

We can detect memory-patching by malware

- Viton, Hypervisor IPS(BlackHat USA 08)
  - Manipulate Guest PageTable/PageTableEntry,
  - To protect memory patching by malicious driver

JP: http://www.fourteenforty.jp/research/research_papers/bh-japan-08-Murakami.pdf

EN: http://www.fourteenforty.jp/research/research_papers/bh-usa-08-Murakami.pdf

Q & A


FFR
http://www.fourteenforty.jp
Fourteenforty Research Institute, Inc.