

Reverse Engineering Malware Dynamic Analysis of Binary Malware II

Advanced dynamic analysis

- Debugger scripting
- Hooking and library injection
- Instrumentation frameworks
- Emulators and virtualization
- Memory forensics

Debugger automation with scripting

- Debuggers can be extended with flexible scripting languages like python
- Any debugging task can be automated: unpacking, decrypting strings, etc.
- Debuggers that support python scripting:
 - Immunity debugger
 - GDB
 - IDA debugger
- Python debugger module for Windows:
 - PaiMei, reverse engineering framework includes "PyDbg" module
 - F-secure proprietary python Win32 debugger using ctypes

Example debugger script: Sober.Y URL's

- Sober was a family of email-worms, written in Visual Basic
- It updated itself using a set of dynamically generated URL's
- Reversing the URL generation algorithm was very challenging
- Developing an automated debugging script was much faster

DEMO: Case Sober.Y

```
File Edit Jump Search View Debug Options Window v AU: idle READY
[.] IDA View-A
Seg001:0042889D call vbaDerefAry1
seg001:004288A2 mov byte ptr [eax], 6Eh
seg001:004288A5 mov [ebp+IGNORE], 6Ch
seg001:004288AC push 68h
seg001:004288AE push dword 42E09C
seg001:004288B4 call vbaDerefAry1
seg001:004288B9 mov byte ptr [eax], 20h
seg001:004288BC mov [ebp+IGNORE], 6Dh
seg001:004288C3 push dword 42E028
seg001:004288C9 push [ebp+var_80]
seg001:004288CC call vbaStrCat
seg001:004288D1 mov dword ptr [ebp+var_80+4], eax
seg001:004288D7 mov [ebp+var_80], 8
seg001:004288E1 lea eax, [ebp+var_80]
seg001:004288E7 push eax
seg001:004288E8 call sub_405090
seg001:004288ED lea ecx, [ebp+var_80]
seg001:004288F3 call vbaFreeVar
seg001:004288F8 mov [ebp+IGNORE], 6Eh
seg001:004288FF call sub_40504E
seg001:00428904 mov [ebp+IGNORE], 6Fh
seg001:0042890B mov dword ptr [ebp+var_80+4], 80020004h
0042889D: downLoadAndExec+D29
F1 Help C Code D Data N Name Alt-X Quit F10 Menu DISK: 27G
```

Hooking and Library Injection

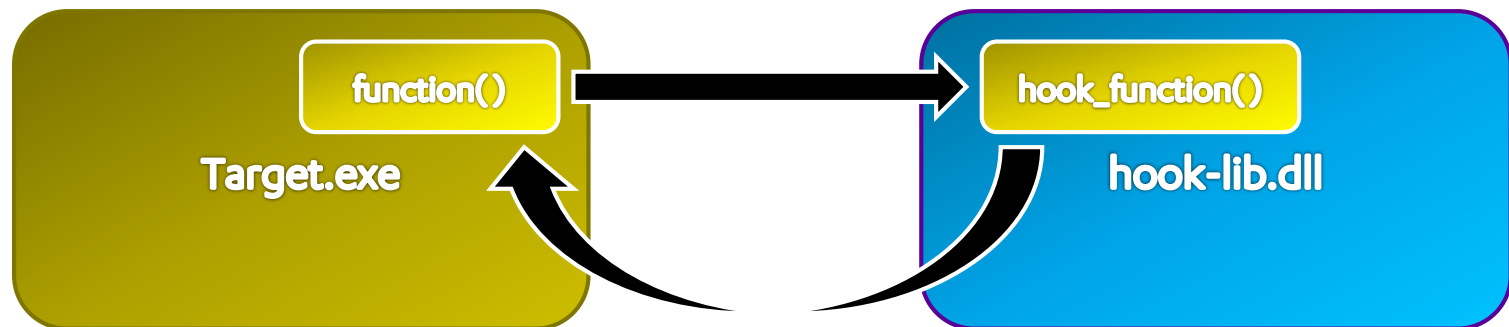
- Basic tracing and troubleshooting techniques
- Can be used in dynamic analysis and reverse engineering very effectively
- Hooks are not using debug API:
 - Fast execution
 - Not confused by anti-debugging tricks
- Intrusive (will modify the target address space)
- Tools can be quite complicated to use (notable exception: Frida)
- Executed on real hardware!
- Example use: Trace file I/O, registry and networking for analyzing program functionality
- Example use: Detect dynamically generated code sections for unpacking

Inline hooking

- Simple way to instrument binaries dynamically, without need to recompile programs
- Usually done by inserting branches to the hooked (target) functions
- Hooking function gets the same parameters as target function
- After the analysis, target function gets back the control
- Analysis code in the hooking function needs memory space
- Position-independent shellcode can be inserted anywhere in the memory space, but more common method is to provide code by injecting dynamically loaded library

Library injection

- Target process is forced to load extra module containing the instrumentation code
- Interesting functions in the target process are hooked
- Hooking module does the necessary processing and returns back to hooked function



PE IAT hooking

- Basic idea: hook by replacing the pointer in import address table (IAT) to the hooking function
- IAT can be easily parsed from the PE headers
- Hooking function is inserted in the address space by library injection

Example hooking library: Detours

- Microsoft Research hooking/injection library for x86/amd64/ia64 Windows
- Uses flexible inline hooking technique
- Understands native functions and managed code (MSIL)
- Detours DLL is loaded with library injection:
 - Dynamically with library injection
 - Statically, by modifying the target import table for loading the Detours DLL before target entry point executes
(**DetourCreateProcessWithDLL()**)
- Hooks are inserted conveniently with **DetourAttach()** and removed with **DetourDetach()**

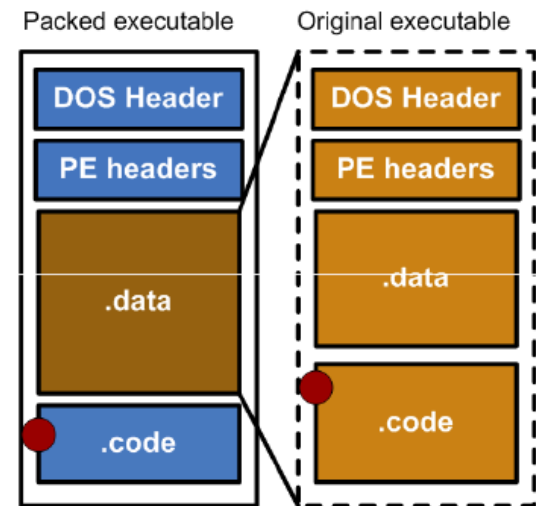
Example hooking library: Frida

- Hook library functions on Windows, OSX, iOS, Linux and Android
- Injected code written in JavaScript
- Internally, Frida injects V8 engine and builds transparent code transitions from native to JS and back



Example tool: FIST

- FIST: F-secure Interactive System Trace
- Proprietary tool for generic unpacking on x86 Windows
- Hooks most kernel32.dll, advapi32.dll, msvcrt.dll, shell32.dll and user32.dll functions
- Hook functions compare the code in return address to the disk image
- If the return address was modified, the code is possibly near the original entry point
- Based on the fact that most non-trivial programs need to use Win32 API's



Instrumentation frameworks

- Dynamic manipulation of programs using binary instrumentation
- Good for profiling and debugging, but also writing reverse engineering tools
- More flexible than function hooking:
 - Instrument instructions, basic blocks
 - Instrument system calls
 - Inspect memory read/write
 - ... And much more
- Major frameworks: DynamoRIO, Pin, Valgrind
- Problems with some packed files!

Using emulators for tracing and instrumentation

- Emulators can be used for tracing by instrumenting code outside the OS
- By definition, it is non-intrusive
- Target executed on emulated hardware, more safe than debugging and hooking
- Instrumentation API:
 - Interface for hooking up instructions, exceptions etc.
 - Example: bochs instrumentation API
- Debugging API:
 - Emulator can export standard debugger API, such as GDB
 - Example: qemu GDB stub

Emulator types

- Hardware virtualization
 - Emulator is sharing the hardware resources with the host machine
 - CPU instructions run directly on real CPU
 - Good performance
 - Examples: VMWare, VirtualBox, Xen, KVM (Linux kernel VM)
- Software
 - Emulator implemented purely using software
 - CPU instructions are interpreted or translated dynamically
 - Can be quite slow
 - Examples: Bochs, Qemu

The HW/SW distinction is not really that clear, for example all HW virtualization solutions will fallback to software emulation in certain situations, like real-mode. Also Xen and KVM use qemu for hardware emulation.

Emulator example: Bochs instrumentation API

- Bochs: open-source PC (x86/amd64) emulator
- Uses interpretation for emulating the instructions
- Interpretation makes Bochs very portable, it runs on any C++ environment
- Supports powerful instrumentation with C++
- Callbacks for
 - CPU events, like interrupts and exceptions
 - CPU instructions
- Support functions, such as memory I/O

Emulator example: Using the Qemu GDB stub

- Qemu: open-source multi-platform emulator
- Uses dynamic code translation for speeding up the emulation
- Supports debugging via the built-in GDB stub
- Qemu GDB stub features:
 - Non-intrusive
 - Breakpoints are implemented in the stub ("hardware")
 - VM time stops when the stub is waiting for input
- GDB supports python scripting
- Flexible system-level tracing tools

Attacking emulators

- Malware has a lot of ways to detect emulators, roughly categorized as:
 - Timing attacks
 - OS implementation
 - Hardware implementation
 - Emulator-related software inside the OS, for example VMWare tools
- Emulators can also be attacked with denial of service attack:
 - Execute massive amount of instructions
 - Emulators in AV engines cannot give too much clock cycles for the emulator
- Most dangerous attack on emulators is to escape from the emulated environment by using a bug in the emulator software

Detecting emulators: OS implementation

- If the emulator is not running full-blown OS, its API emulation can be easily detected
- Windows has huge amount of documented API's and undocumented, still quite solid API's
- Emulators try to return something even for unsupported API's, just to keep execution ongoing
- Current malware uses a lot of API-related tricks to detect emulators
- Some examples:
 - Call API's with bogus or unsupported parameters, verify return values
 - Use of callback functions in the API's for doing something useful
 - Observe side-effects of API's (register values, traces in stack etc.)

Detecting emulators: Hardware

- Implementing a CPU emulator is a very complicated task:
 - Intel x86/amd64 instruction set consists of ~500 instructions
 - Paging and exception handling is complicated
- Full-blown PC emulator needs to implement a fair amount of hardware devices to be convincing
- Some examples for detecting emulator hardware:
 - Detect missing CPUID information or inconsistencies (*)
 - Check implementation of complicated instructions, like CMPXCHG8B (*)
 - Check non-zero Local Descriptor Table (LDT) to detect VMWare (*)
 - Detect VMWare devices, for example "VMWare PCI Express Root Port"

(*) Peter Ferrie: Attacks on More Virtual Machine Emulators (<http://pferrie.tripod.com/papers/attacks2.pdf>)

Emulator detection

example: CPUID instruction

- CPUID is used to get the processor information:
 - Vendor identification string, for example "GenuineIntel"
 - CPU type, family, model and stepping
 - Supported instruction sets
 - Other features, such as thermal and power management
- Software emulator needs to be consistent in CPUID return values and features it can emulate
- Attacker can also check if such a CPU is really available in reality
- Almost all software emulators fail to be consistent

CPUID continued

From Intel 64 and IA-32 Architectures Software Developer's Manual:

Initial EAX Value	Information Provided about the Processor	
Basic CPUID Information		
0H	EAX EBX ECX EDX	Maximum Input Value for Basic CPUID Information (see Table 3-18) "Genu" "ntel" "inel"
01H	EAX EBX ECX EDX	Version Information: Type, Family, Model, and Stepping ID (see Figure 3-5) Bits 07-00: Brand Index Bits 15-08: CLFLUSH line size (Value * 8 = cache line size in bytes) Bits 23-16: Maximum number of addressable IDs for logical processors in this physical package*. Bits 31-24: Initial APIC ID Feature Information (see Figure 3-6 and Table 3-20) Feature Information (see Figure 3-7 and Table 3-21)

Information Provided about the Processor
Bits 07-00: Cache level starts at 1 Bit 08: Set indicating cache level (does not need Set indicator) Bit 09: Fully Associative cache Bits 13-12: Reserved Bits 25-14: Maximum number of addressable IDs for logical processors sharing this cache* Bits 31-26: Maximum number of addressable IDs for processor cores in this physical package* Bits 11-00: System Coherency Line Size* Bits 21-12: # Physical Line Partitions* Bits 31-02: # Value of associativity* Bits 31-05: # Number of Sets* Bit 0: Value Back Invariant Indicator Bit 1: Valid/Invalid from threads sharing this cache acts upon lower-level caches for threads sharing this cache Bit 2: Valid/Invalid is not guaranteed to act upon lower-level caches of non-sharing threads sharing this cache Bit 3: Cache Incoherencies Bit 4: Cache is not inclusive of lower cache levels Bit 5: Cache is inclusive of lower cache levels Bit 6: Complex Cache Indexing Bit 7: Direct-mapped cache Bit 8: Complex Function is used to index the cache, potentially using all addresses Bits 31-08: Reserved = 0 NOTES: * Add one to the return value to get the result. ** The nearest power-of-2 integer that is not smaller than (1) + (EAX[25:14]) is the number of unique initial APIC IDs reserved for addressing different logical processors sharing this cache. *** The nearest power-of-2 integer that is not smaller than (1) + (EAX[31:26]) is the number of unique Core IDs reserved for addressing different processor cores in a physical package. Core ID is a sub-set of bits of the initial APIC ID. **** The required value is constant for valid initial values in EAX; valid EAX values start from 0.

Initial EAX Value	Information Provided about the Processor
00H	EAX: Bits 15-00: Smallest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0 EBX: Bits 15-00: Largest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0 ECX: Bit 00: Enumeration of Monitor-Mwait extensions (beyond EAX and EBX registers) supported Bit 01: Supports treating interrupts as break-event for Mwait, even when interrupts disabled Bits 31-02: Reserved EDX: Bits 03-00: Number of C0P sub-C-states supported using Mwait Bits 07-04: Number of C1+ sub-C-states supported using Mwait Bits 11-08: Number of C2+ sub-C-states supported using Mwait Bits 15-12: Number of C3+ sub-C-states supported using Mwait Bits 19-16: Number of C4+ sub-C-states supported using Mwait Bits 31-20: Reserved = 0 NOTES: * The definition of C0 through C4 states for Mwait extension are processor-specific C-states, not ACPI C-states.
00H	EAX: Bit 00: Digital temperature sensor is supported if set Bit 01: Intel Turbo Boost Technology Available (see description of IA32_TURBO_CTL[38]) Bit 02: ADAPT: APC-TIME-always-running feature is supported if set Bit 03: Reserved Bit 04: PPL: Power limit notification controls are supported if set Bit 05: ECD: Clock modulation duty cycle extension is supported if set Bit 06: PTH: Package thermal management is supported if set Bits 31-07: Reserved EDX: Bits 03-00: Number of Interrupt Thresholds in Digital Thermal Sensor Bits 31-04: Reserved

Initial EAX Value	Information Provided about the Processor
00H	ECX: Bit 00: Hardware Correction Feedback Capability (Presence of IA32_HFTR and IA32_HFTR1). The capability to provide a measure of observed processor performance (as a percentage of the observed processor performance at frequency specified in CPUID Brand String) Bits 02-01: Reserved = 0 Bit 03: The processor supports performance-energy bias preference if CPUID_ECX[Bit 03] is set and it also implies the presence of a new architecture MSR called IA32_ARCH_PPL_BIAS (100H) Bits 31-04: Reserved = 0 Reserved = 0 Structured Extended Feature Flags Enumeration Leaf (Output depends on EAX input value) 00H: Sub-leaf 0 (Input EAX = 0) EAX: Bits 31-00: Reports the maximum number of supported leaf 7 sub-features EBX: Bit 00: FSGSBASE: Supports RDPBSBASE/RDPBSBASE+RFBPBASE/page 1 Bit 01: Reserved Bit 02: SMPL: Supports Supervisor Mode Execution Protection if 1 Bit 03: Supports Enhanced REP MOV/STOBE if 1 Bit 04: BPU/CD: If 1, supports BPU/CD instruction for system software that manages process context identifiers Bit 31-11: Reserved ECX: Reserved EDX: Reserved
00H	EAX: Value of bits 01-03 of IA32_PLATFORM_CAP MSR address (100H) EBX: Reserved ECX: Reserved EDX: Reserved Architecture Performance Monitoring Leaf

Initial EAX Value	Information Provided about the Processor
00H	EAX: Bits 07-00: Version ID of architectural performance monitoring Bits 15-08: Number of general-purpose performance monitoring counters per logical processor Bits 23-16: Bit width of general-purpose performance monitoring counter Bits 31-24: Length of EBX bit vector to enumerate architectural performance monitoring events EBX: Bit 00: Core cycle event not available if 1 Bit 01: Instruction retired event not available if 1 Bit 02: Branch instruction retired event not available if 1 Bit 03: Last-level cache reference event not available if 1 Bit 04: Last-level cache miss event not available if 1 Bit 05: Branch instruction retired event not available if 1 Bit 06: Branch instruction retired event not available if 1 Bit 07-00: Reserved = 0 ECX: Reserved = 0 EDX: Bits 04-00: Number of fixed-function performance counters (if supported) in 10 Bits 12-05: Bit width of fixed-function performance counters (if supported) in 10 Bits 31-05: Reserved = 0
00H	Extended Topology Enumeration Leaf NOTES: * Bit of Leaf EBX output depends on the initial value in EAX. EBX output does not vary with initial value in EAX. ECX/EDX output always reflect initial value in EAX. An other output value for an invalid initial value in EAX are 0. Leaf EBX events if EBX[15:0] is not zero. Leaf EBX events if EBX[15:0] is not zero. Bits 04-00: Number of bits right on xAPIC ID to get a unique topology ID of the next level type. All logic processors with the same next level ID share current level. Bits 31-05: Reserved EBX: Bits 15-00: Number of logical processors at this level type. The number reflects configuration as reported by user. Bits 31-18: Reserved ECX: Bits 07-00: Last-level cache line size in bytes Bits 15-08: Last-level cache line size in bytes Bits 31-18: Reserved

Initial EAX Value	Information Provided about the Processor
00H	ECX: Bits 31-00: xAPIC ID of the current logical processor. NOTES: * Software should use this field (EAX[40]) to enumerate processor topology of the system. ** Software must not use EBX[15:0] to enumerate processor topology of the system. This value in the EBX[15:0] is only intended for display/diagnostic purposes. The actual number of logical processors available to BIOS/OS applications may be different from the value of EBX[15:0], depending on software and platform hardware configuration. *** The value of the "leaf type" field is not related to level numbers in any way; higher "leaf type" values do not mean higher levels. Leaf type field has the following encoding: 0: Invalid 1: SHIT 2: Core 3-255: Reserved Processor Extended State Enumeration Main Leaf (EAX = 00H, ECX = 0) 00H: Leaf EBX main leaf (EAX = 0) EAX: Bits 31-00: Reports the valid bit fields of the lower 32 bits of XCR0 if a bit is 1, the corresponding bit field in XCR0 is reserved. Bit 00: legacy x87 Bit 01: SSE Bit 02: SSE-64 Bit 03: AVX Bits 31-03: Reserved EBX: Bits 21-00: Maximum size (bytes) from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than EAX if some features at the end of the XSAVE save area are not enabled. ECX: Bits 31-00: Maximum size (bytes) from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processor. i.e. all the valid bit fields in XCR0. EDX: Bits 31-00: Reports the valid bit fields of the upper 32 bits of XCR0. If a bit is 0, the corresponding bit field in XCR0 is reserved. Processor Extended State Enumeration Sub-leaf (EAX = 00H, ECX = 1) 00H: Leaf EBX sub-leaf (EAX = 0, ECX = 1) EAX: Bits 31-00: Reports the valid bit fields of the lower 32 bits of XCR0 if a bit is 1, the corresponding bit field in XCR0 is reserved. Bit 00: legacy x87 Bit 01: SSE Bit 02: SSE-64 Bit 03: AVX Bits 31-03: Reserved EBX: Bits 21-00: Maximum size (bytes) from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than EAX if some features at the end of the XSAVE save area are not enabled. ECX: Bits 31-00: Maximum size (bytes) from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processor. i.e. all the valid bit fields in XCR0. EDX: Bits 31-00: Reports the valid bit fields of the upper 32 bits of XCR0. If a bit is 0, the corresponding bit field in XCR0 is reserved.

Initial EAX Value	Information Provided about the Processor
00H	EAX: Bits 31-00: Reserved Bit 00: XSAVEOPT is available EBX: Reserved ECX: Reserved EDX: Reserved Processor Extended State Enumeration Sub-leaf (EAX = 00H, ECX = 1) 00H: Leaf EBX sub-leaf (EAX = 0, ECX = 1) EAX: Bits 31-00: Reports the valid bit fields of the lower 32 bits of XCR0 if a bit is 1, the corresponding bit field in XCR0 is reserved. Bit 00: legacy x87 Bit 01: SSE Bit 02: SSE-64 Bit 03: AVX Bits 31-03: Reserved EBX: Bits 21-00: Maximum size (bytes) from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than EAX if some features at the end of the XSAVE save area are not enabled. ECX: Bits 31-00: Maximum size (bytes) from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processor. i.e. all the valid bit fields in XCR0. EDX: Bits 31-00: Reports the valid bit fields of the upper 32 bits of XCR0. If a bit is 0, the corresponding bit field in XCR0 is reserved.
40000000H	Unimplemented CPUID Leaf Functions If the leaf EBX sub-leaf (EAX = 0, ECX = 1) is supported, the maximum physical address number supported should come from this field. #FFFFFFFFH: Extended Function CPUID information 80000000H: EAX: Maximum input value for Extended Function CPUID information (see Table 3-18) EBX: Reserved ECX: Reserved EDX: Reserved

Initial EAX Value	Information Provided about the Processor
80000001H	EAX: Extended Processor Signature and Feature Bits ECX: Reserved EBX: Bit 00: Leaf EBX available in 64-bit mode Bit 01: Reserved EDX: Bits 10-00: Reserved Bit 11: SERIAL_PSTATE: Available (when in 64-bit mode) Bits 19-12: Reserved = 0 Bit 20: Branch-Check-Bit available Bits 29-21: Reserved = 0 Bit 30: 1: 64-bit pages are available if 1 Bit 31: 2: 64-bit pages are available if 1 Bit 32: 2: 64-bit pages are available if 1 Bit 33: 2: 64-bit pages are available if 1 Bit 34: 2: 64-bit pages are available if 1 Bit 35: 2: 64-bit pages are available if 1 Bit 36: 2: 64-bit pages are available if 1 Bit 37: 2: 64-bit pages are available if 1 Bit 38: 2: 64-bit pages are available if 1 Bit 39: 2: 64-bit pages are available if 1 Bit 40: 2: 64-bit pages are available if 1 Bit 41: 2: 64-bit pages are available if 1 Bit 42: 2: 64-bit pages are available if 1 Bit 43: 2: 64-bit pages are available if 1 Bit 44: 2: 64-bit pages are available if 1 Bit 45: 2: 64-bit pages are available if 1 Bit 46: 2: 64-bit pages are available if 1 Bit 47: 2: 64-bit pages are available if 1 Bit 48: 2: 64-bit pages are available if 1 Bit 49: 2: 64-bit pages are available if 1 Bit 50: 2: 64-bit pages are available if 1 Bit 51: 2: 64-bit pages are available if 1 Bit 52: 2: 64-bit pages are available if 1 Bit 53: 2: 64-bit pages are available if 1 Bit 54: 2: 64-bit pages are available if 1 Bit 55: 2: 64-bit pages are available if 1 Bit 56: 2: 64-bit pages are available if 1 Bit 57: 2: 64-bit pages are available if 1 Bit 58: 2: 64-bit pages are available if 1 Bit 59: 2: 64-bit pages are available if 1 Bit 60: 2: 64-bit pages are available if 1 Bit 61: 2: 64-bit pages are available if 1 Bit 62: 2: 64-bit pages are available if 1 Bit 63: 2: 64-bit pages are available if 1 Bit 64: 2: 64-bit pages are available if 1 Bit 65: 2: 64-bit pages are available if 1 Bit 66: 2: 64-bit pages are available if 1 Bit 67: 2: 64-bit pages are available if 1 Bit 68: 2: 64-bit pages are available if 1 Bit 69: 2: 64-bit pages are available if 1 Bit 70: 2: 64-bit pages are available if 1 Bit 71: 2: 64-bit pages are available if 1 Bit 72: 2: 64-bit pages are available if 1 Bit 73: 2: 64-bit pages are available if 1 Bit 74: 2: 64-bit pages are available if 1 Bit 75: 2: 64-bit pages are available if 1 Bit 76: 2: 64-bit pages are available if 1 Bit 77: 2: 64-bit pages are available if 1 Bit 78: 2: 64-bit pages are available if 1 Bit 79: 2: 64-bit pages are available if 1 Bit 80: 2: 64-bit pages are available if 1 Bit 81: 2: 64-bit pages are available if 1 Bit 82: 2: 64-bit pages are available if 1 Bit 83: 2: 64-bit pages are available if 1 Bit 84: 2: 64-bit pages are available if 1 Bit 85: 2: 64-bit pages are available if 1 Bit 86: 2: 64-bit pages are available if 1 Bit 87: 2: 64-bit pages are available if 1 Bit 88: 2: 64-bit pages are available if 1 Bit 89: 2: 64-bit pages are available if 1 Bit 90: 2: 64-bit pages are available if 1 Bit 91: 2: 64-bit pages are available if 1 Bit 92: 2: 64-bit pages are available if 1 Bit 93: 2: 64-bit pages are available if 1 Bit 94: 2: 64-bit pages are available if 1 Bit 95: 2: 64-bit pages are available if 1 Bit 96: 2: 64-bit pages are available if 1 Bit 97: 2: 64-bit pages are available if 1 Bit 98: 2: 64-bit pages are available if 1 Bit 99: 2: 64-bit pages are available if 1 Bit 100: 2: 64-bit pages are available if 1 Bit 101: 2: 64-bit pages are available if 1 Bit 102: 2: 64-bit pages are available if 1 Bit 103: 2: 64-bit pages are available if 1 Bit 104: 2: 64-bit pages are available if 1 Bit 105: 2: 64-bit pages are available if 1 Bit 106: 2: 64-bit pages are available if 1 Bit 107: 2: 64-bit pages are available if 1 Bit 108: 2: 64-bit pages are available if 1 Bit 109: 2: 64-bit pages are available if 1 Bit 110: 2: 64-bit pages are available if 1 Bit 111: 2: 64-bit pages are available if 1 Bit 112: 2: 64-bit pages are available if 1 Bit 113: 2: 64-bit pages are available if 1 Bit 114: 2: 64-bit pages are available if 1 Bit 115: 2: 64-bit pages are available if 1 Bit 116: 2: 64-bit pages are available if 1 Bit 117: 2: 64-bit pages are available if 1 Bit 118: 2: 64-bit pages are available if 1 Bit 119: 2: 64-bit pages are available if 1 Bit 120: 2: 64-bit pages are available if 1 Bit 121: 2: 64-bit pages are available if 1 Bit 122: 2: 64-bit pages are available if 1 Bit 123: 2: 64-bit pages are available if 1 Bit 124: 2: 64-bit pages are available if 1 Bit 125: 2: 64-bit pages are available if 1 Bit 126: 2: 64-bit pages are available if 1 Bit 127: 2: 64-bit pages are available if 1 Bit 128: 2: 64-bit pages are available if 1 Bit 129: 2: 64-bit pages are available if 1 Bit 130: 2: 64-bit pages are available if 1 Bit 131: 2: 64-bit pages are available if 1 Bit 132: 2: 64-bit pages are available if 1 Bit 133: 2: 64-bit pages are available if 1 Bit 134: 2: 64-bit pages are available if 1 Bit 135: 2: 64-bit pages are available if 1 Bit 136: 2: 64-bit pages are available if 1 Bit 137: 2: 64-bit pages are available if 1 Bit 138: 2: 64-bit pages are available if 1 Bit 139: 2: 64-bit pages are available if 1 Bit 140: 2: 64-bit pages are available if 1 Bit 141: 2: 64-bit pages are available if 1 Bit 142: 2: 64-bit pages are available if 1 Bit 143: 2: 64-bit pages are available if 1 Bit 144: 2: 64-bit pages are available if 1 Bit 145: 2: 64-bit pages are available if 1 Bit 146: 2: 64-bit pages are available if 1 Bit 147: 2: 64-bit pages are available if 1 Bit 148: 2: 64-bit pages are available if 1 Bit 149: 2: 64-bit pages are available if 1 Bit 150: 2: 64-bit pages are available if 1 Bit 151: 2: 64-bit pages are available if 1 Bit 152: 2: 64-bit pages are available if 1 Bit 153: 2: 64-bit pages are available if 1 Bit 154: 2: 64-bit pages are available if 1 Bit 155: 2: 64-bit pages are available if 1 Bit 156: 2: 64-bit pages are available if 1 Bit 157: 2: 64-bit pages are available if 1 Bit 158: 2: 64-bit pages are available if 1 Bit 159: 2: 64-bit pages are available if 1 Bit 160: 2: 64-bit pages are available if 1 Bit 161: 2: 64-bit pages are available if 1 Bit 162: 2: 64-bit pages are available if 1 Bit 163: 2: 64-bit pages are available if 1 Bit 164: 2: 64-bit pages are available if 1 Bit 165: 2: 64-bit pages are available if 1 Bit 166: 2: 64-bit pages are available if 1 Bit 167: 2: 64-bit pages are available if 1 Bit 168: 2: 64-bit pages are available if 1 Bit 169: 2: 64-bit pages are available if 1 Bit 170: 2: 64-bit pages are available if 1 Bit 171: 2: 64-bit pages are available if 1 Bit 172: 2: 64-bit pages are available if 1 Bit 173: 2: 64-bit pages are available if 1 Bit 174: 2: 64-bit pages are available if 1 Bit 175: 2: 64-bit pages are available if 1 Bit 176: 2: 64-bit pages are available if 1 Bit 177: 2: 64-bit pages are available if 1 Bit 178: 2: 64-bit pages are available if 1 Bit 179: 2: 64-bit pages are available if 1 Bit 180: 2: 64-bit pages are available if 1 Bit 181: 2: 64-bit pages are available if 1 Bit 182: 2: 64-bit pages are available if 1 Bit 183: 2: 64-bit pages are available if 1 Bit 184: 2: 64-bit pages are available if 1 Bit 185: 2: 64-bit pages are available if 1 Bit 186: 2: 64-bit pages are available if 1 Bit 187: 2: 64-bit pages are available if 1 Bit 188: 2: 64-bit pages are available if 1 Bit 189: 2: 64-bit pages are available if 1 Bit 190: 2: 64-bit pages are available if 1 Bit 191: 2: 64-bit pages are available if 1 Bit 192: 2: 64-bit pages are available if 1 Bit 193: 2: 64-bit pages are available if 1 Bit 194: 2: 64-bit pages are available if 1 Bit 195: 2: 64-bit pages are available if 1 Bit 196: 2: 64-bit pages are available if 1 Bit 197: 2: 64-bit pages are available if 1 Bit 198: 2: 64-bit pages are available if 1 Bit 199: 2: 64-bit pages are available if 1 Bit 200: 2: 64-bit pages are available if 1 Bit 201: 2: 64-bit pages are available if 1 Bit 202: 2: 64-bit pages are available if 1 Bit 203: 2: 64-bit pages are available if 1 Bit 204: 2: 64-bit pages are available if 1 Bit 205: 2: 64-bit pages are available if 1 Bit 206: 2: 64-bit pages are available if 1 Bit 207: 2: 64-bit pages are available if 1 Bit 208: 2: 64-bit pages are available if 1 Bit 209: 2: 64-bit pages are available if 1 Bit 210: 2: 64-bit pages are available if 1 Bit 211: 2: 64-bit pages are available if 1 Bit 212: 2: 64-bit pages are available if 1 Bit 213: 2: 64-bit pages are available if 1 Bit 214: 2: 64-bit pages are available if 1 Bit 215: 2: 64-bit pages are available if 1 Bit 216: 2: 64-bit pages are available if 1 Bit 217: 2: 64-bit pages are available if 1 Bit 218: 2: 64-bit pages are available if 1 Bit 219: 2: 64-bit pages are available if 1 Bit 220: 2: 64-bit pages are available if 1 Bit 221: 2: 64-bit pages are available if 1 Bit 222: 2: 64-bit pages are available if 1 Bit 223: 2: 64-bit pages are available if 1 Bit 224: 2: 64-bit pages are available if 1 Bit 225: 2: 64-bit pages are available if 1 Bit 226: 2: 64-bit pages are available if 1 Bit 227: 2: 64-bit pages are available if 1 Bit 228: 2: 64-bit pages are available if 1 Bit 229: 2: 64-bit pages are available if 1 Bit 230: 2: 64-bit pages are available if 1 Bit 231: 2: 64-bit pages are available if 1 Bit 232: 2: 64-bit pages are available if 1 Bit 233: 2: 64-bit pages are available if 1 Bit 234: 2: 64-bit pages are available if 1 Bit 235: 2: 64-bit pages are available if 1 Bit 236: 2: 64-bit pages are available if 1 Bit 237: 2: 64-bit pages are available if 1 Bit 238: 2: 64-bit pages are available if 1 Bit 239: 2: 64-bit pages are available if 1 Bit 240: 2: 64-bit pages are available if 1 Bit 241: 2: 64-bit pages are available if 1 Bit 242: 2: 64-bit pages are available if 1 Bit 243: 2: 64-bit pages are available if 1 Bit 244: 2: 64-bit pages are available if 1 Bit 245: 2: 64-bit pages are available if 1 Bit 246: 2: 64-bit pages are available if 1 Bit 247: 2: 64-bit pages are available if 1 Bit 248: 2: 64-bit pages are available if 1 Bit 249: 2: 64-bit pages are available if 1 Bit 250: 2: 64-bit pages are available if 1 Bit 251: 2: 64-bit pages are available if 1 Bit 252: 2: 64-bit pages are available if 1 Bit 253: 2: 64-bit pages are available if 1 Bit 254: 2: 64-bit pages are available if 1 Bit 255: 2: 64-bit pages are available if 1 Bit 256: 2: 64-bit pages are available if 1 Bit 257: 2: 64-bit pages are available if 1 Bit 258: 2: 64-bit pages are available if 1 Bit 259: 2: 64-bit pages are available if 1 Bit 260: 2: 64-bit pages are available if 1 Bit 261: 2: 64-bit pages are available if 1 Bit 262: 2: 64-bit pages are available if 1 Bit 263: 2: 64-bit pages are available if 1 Bit 264: 2: 64-bit pages are available if 1 Bit 265: 2: 64-bit pages are available if 1 Bit 266: 2: 64-bit pages are available if 1 Bit 267: 2: 64-bit pages are available if 1 Bit 268: 2: 64-bit pages are available if 1 Bit 269: 2: 64-bit pages are available if 1 Bit 270: 2: 64-bit pages are available if 1 Bit 271: 2: 64-bit pages are available if 1 Bit 272: 2: 64-bit pages are available if 1 Bit 273: 2: 64-bit pages are available if 1 Bit 274: 2: 64-bit pages are available if 1 Bit 275: 2: 64-bit pages are available if 1 Bit 276: 2: 64-bit pages are available if 1 Bit 277: 2: 64-bit pages are available if 1 Bit 278: 2: 64-bit pages are available if 1 Bit 279: 2: 64-bit pages are available if 1 Bit 280: 2: 64-bit pages are available if 1 Bit 281: 2: 64-bit pages are available if 1 Bit 282: 2: 64-bit pages are available if 1 Bit 283: 2: 64-bit pages are available if 1 Bit 284: 2: 64-bit pages are available if 1 Bit 285: 2: 64-bit pages are available if 1 Bit 286: 2: 64-bit pages are available if 1 Bit 287: 2: 64-bit pages are available if 1 Bit 288: 2: 64-bit pages are available if 1 Bit 289: 2: 64-bit pages are available if 1 Bit 290: 2: 64-bit pages are available if 1 Bit 291: 2: 64-bit pages are available if 1 Bit 292: 2: 64-bit pages are available if 1 Bit 293: 2: 64-bit pages are available if 1 Bit 294: 2: 64-bit pages are available if 1 Bit 295: 2: 64-bit pages are available if 1 Bit 296: 2: 64-bit pages are available if 1 Bit 297: 2: 64-bit pages are available if 1 Bit 298: 2: 64-bit pages are available if 1 Bit 299: 2: 64-bit pages are available if 1 Bit 300: 2: 64-bit pages are available if 1 Bit 301: 2: 64-bit pages are available if 1 Bit 302: 2: 64-bit pages are available if 1 Bit 303: 2: 64-bit pages are available if 1 Bit 304: 2: 64-bit pages are available if 1 Bit 305: 2: 64-bit pages are available if 1 Bit 306: 2: 64-bit pages are available if 1 Bit 307: 2: 64-bit pages are available if 1 Bit 308: 2: 64-bit pages are available if 1 Bit 309: 2: 64-bit pages are available if 1 Bit 310: 2: 64-bit pages are available if 1 Bit 311: 2: 64-bit pages are available if 1 Bit 312: 2: 64-bit pages are available if 1 Bit 313: 2: 64-bit pages are available if 1 Bit 314: 2: 64-bit pages are available if 1 Bit 315: 2: 64-bit pages are available if 1 Bit 316: 2: 64-bit pages are available if 1 Bit 317: 2: 64-bit pages are available if 1 Bit 318: 2: 64-bit pages are available if 1 Bit 319: 2: 64-bit pages are available if 1 Bit 320: 2: 64-bit pages are available if 1 Bit 321: 2: 64-bit pages are available if 1 Bit 322: 2: 64-bit pages are available if 1 Bit 323: 2: 64-bit pages are available if 1 Bit 324: 2: 64-bit pages are available if 1 Bit 325: 2: 64-bit pages are available if 1 Bit 326: 2: 64-bit pages are available if 1 Bit 327: 2: 64-bit pages are available if 1 Bit 328: 2: 64-bit pages are available if 1 Bit 329: 2: 64-bit pages are available if 1 Bit 330: 2: 64-bit pages are available if 1 Bit 331: 2: 64-bit pages are available if 1 Bit 332: 2: 64-bit pages are available if 1 Bit 333: 2: 64-bit pages are available if 1 Bit 334: 2: 64-bit pages are available if 1 Bit 335: 2: 64-bit pages are available if 1 Bit 336: 2: 64-bit pages are available if 1 Bit 337: 2: 64-bit pages are available if 1 Bit 338: 2: 64-bit pages are available if 1 Bit 339: 2: 64-bit pages are available if 1 Bit 340: 2: 64-bit pages are available if 1 Bit 341: 2: 64-bit pages are available if 1 Bit 342: 2: 64-bit pages are available if 1 Bit 343: 2: 64-bit pages are available if 1 Bit 344: 2: 64-bit pages are available if 1 Bit 345: 2: 64-bit pages are available if 1 Bit 346: 2: 64-bit pages are available if 1 Bit 347: 2: 64-bit pages are available if 1 Bit 348: 2: 64-bit pages are available if 1 Bit 349: 2: 64-bit pages are available if 1 Bit 350: 2: 64-bit pages are available if 1 Bit 351: 2: 64-bit pages are available if 1 Bit 352: 2: 64-bit pages are available if 1 Bit 353: 2: 64-bit pages are available if 1 Bit 354: 2: 64-bit pages are available if 1 Bit 355: 2: 64-bit pages are available if 1 Bit 356: 2: 64-bit pages are available if 1 Bit 357: 2: 64-bit pages are available if 1 Bit 358: 2: 64-bit pages are available if 1 Bit 359: 2: 64-bit pages are available if 1 Bit 360: 2: 64-bit pages are available if 1 Bit 361: 2: 64-bit pages are available if 1 Bit 362: 2: 64-bit pages are available if 1 Bit 363: 2: 64-bit pages are available if 1 Bit 364: 2: 64-bit pages are available if 1 Bit 365: 2: 64-bit pages are available if 1 Bit 366: 2: 64-bit pages are available if 1 Bit 367: 2: 64-bit pages are available if 1 Bit 368: 2: 64-bit pages are available if 1 Bit 369: 2: 64-bit pages are available if 1 Bit 370: 2: 64-bit pages are available if 1 Bit 371: 2: 64-bit pages are available if 1 Bit 372: 2: 64-bit pages are available if 1 Bit 373: 2: 64-bit pages are available if 1 Bit 374: 2: 64-bit pages are available if 1 Bit 375: 2: 64-bit pages are available if 1 Bit 376: 2: 64-bit pages are available if 1 Bit 377: 2: 64-bit pages are available if 1 Bit 378: 2: 64-bit pages are available if 1 Bit 379: 2: 64-bit pages are available if 1 Bit 380: 2: 64-bit pages are available if 1 Bit 381: 2: 64-bit pages are available if 1 Bit 382: 2: 64-bit pages are available if 1 Bit 383: 2: 64-bit pages are available if 1 Bit 384: 2: 64-bit pages are available if 1 Bit 385: 2: 64-bit pages are available if 1 Bit 386: 2: 64-bit pages are available if 1 Bit 387: 2: 64-bit pages are available if 1 Bit 388: 2: 64-bit pages are available if 1 Bit 389: 2: 64-bit pages are available if 1 Bit 390: 2: 64-bit pages are available if 1 Bit 391: 2: 64-bit pages are available if 1 Bit 392: 2: 64-bit pages are available if 1 Bit 393: 2: 64-bit pages are available if 1 Bit 394: 2: 64-bit pages are available if 1 Bit 395: 2: 64-bit pages are available if 1 Bit 396: 2: 64-bit pages are available if 1 Bit 397: 2: 64-bit pages are available if 1 Bit 398: 2: 64-bit pages are available if 1 Bit 399: 2: 64-bit pages are available if 1 Bit 400: 2: 64-bit pages are available if 1 Bit 401: 2: 64-bit pages are available if 1 Bit 402: 2: 64-bit pages are available if 1 Bit 403: 2: 64-bit pages are available if 1 Bit 404: 2: 64-bit pages are available if 1 Bit 405: 2: 64-bit pages are available if 1 Bit 406: 2: 64-bit pages are available if 1 Bit 407: 2: 64-bit pages are available if 1 Bit 408: 2: 64-bit pages are available if 1 Bit 409: 2: 64-bit pages are available if 1 Bit 410: 2: 64-bit pages are available if 1 Bit 411: 2: 64-bit pages are available

Memory forensics

- Idea: let the malware run freely and analyze the memory
- Why is it effective:
 - All interesting is in memory (executable images, file I/O buffers, OS structures and objects etc.)
 - Analysis is done outside the running environment
- Possible drawbacks:
 - Slowness
 - When to dump?
 - On physical machines, dump software can be tampered



Volatility

- Open-source framework for physical memory analysis
- Support for 32 –and 64-bit Windows, Linux and OSX
- Gather information from processes, virtual memory, OS structures, OS objects and more
- Dump images and memory pages on disk
- Lots of useful plugins, like "malfind"
- Typical workflow: pause a VM, give volatility the physical memory as a flat file

DEMO

Volatility example: Equation/GRAYFISH

```
> volatility -f grayfish.vmem malfind
```

```
Process: services.exe Pid: 716 Address: 0xb90000
```

```
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
```

```
Flags: CommitCharge: 8, PrivateMemory: 1, Protection: 6
```

```
0x00b90000  68 00 00 00 00 68 17 00 b9 00 68 d5 1f 82 7c 68  h....h....h...|h
0x00b90010  fa 13 b0 00 ff 24 24 8b c5 83 c0 11 c7 00 29 16  ....$$.....).
0x00b90020  80 7c 81 c0 ad ff ff ff bf 6a 15 b0 00 33 ed ff  .|.....j...3..
0x00b90030  64 24 fc 90 90 90 90 90 90 90 90 90 90 90 90  d$.....
```

```
0xb90000 6800000000      PUSH DWORD 0x0
0xb90005 681700b900      PUSH DWORD 0xb90017
0xb9000a 68d51f827c      PUSH DWORD 0x7c821fd5
0xb9000f 68fa13b000      PUSH DWORD 0xb013fa
0xb90014 ff2424      JMP DWORD [ESP]
```

Equation: The Death Star of Malware Galaxy - Kaspersky Labs' Global Research & Analysis Team

Equation/GRAYFISH

```
> volatility -f grayfish.vmem vaddump -pid 716
```

```
> hexdump -C services.exe.1f64550.0x00b00000-0x00b0afff.dmp
```

00000000	0f 5e 30 00 61 00 00 00	2c 00 00 00 35 35 00 00	.^0.a...,...55..
00000010	e8 00 00 00 00 00 00 00	c0 00 00 00 00 00 00 00
00000020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000030	00 00 00 00 00 00 00 00	00 00 00 00 50 00 00 00P...
00000040	1a 95 7e 1a 00 bc 23 8f	2b e8 cb 44 8f 2b 9c 78	..~...#.+.D.+x
00000050	43 31 60 d0 66 05 ad 66	eb 6f 60 81 eb 3a 3a 05	C1`.f..f.o`...:..
00000060	fc 60 b6 17 60 66 c7 3a	60 43 3a 60 ec a5 d1 60	.`..`f.:`C:`...`
00000070	6f 05 4c 17 7a 4f 4f ee	8c 00 00 00 00 00 00 00	o.L.z00.....
00000080	99 ac eb 4a 85 97 e5 8f	85 97 e5 8f 85 97 e5 8f	...J.....
00000090	a4 73 cb 8f ef 97 e5 8f	80 3e f7 8f 59 97 e5 8f	.s.....>..Y...
000000a0	80 3e b9 8f ef 97 e5 8f	13 a1 f7 8f 62 97 e5 8f	.>.....b...
000000b0	a4 ef 06 8f 24 97 e5 8f	85 97 1a 8f 7d 97 e5 8f\$......}...
000000c0	13 a1 2c 8f 97 97 e5 8f	b0 37 b9 8f ba 97 e5 8f	.,.....7.....
000000d0	06 43 81 78 85 97 e5 8f	00 00 00 00 00 00 00 00	.C.x.....
000000e0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000f0	70 b7 00 00 44 cb 2c 00	e5 16 93 4d 00 00 00 00	p...D.,....M....

Equation/GRAYFISH

```
> volatility -f grayfish.vmem vaddump -pid 716
```

```
> hexdump -C services.exe.1f64550.0x00b00000-0x00b0afff.dmp
```

```
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ.....|
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |.....@.....|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 f0 00 00 00  |.....|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |.....!..L.!Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.....|
00000080  ab 84 61 9e ef e5 0f cd  ef e5 0f cd ef e5 0f cd  |..a.....|
00000090  6c f9 01 cd ed e5 0f cd  80 fa 05 cd eb e5 0f cd  |l.....|
000000a0  80 fa 0b cd ed e5 0f cd  d9 c3 05 cd e6 e5 0f cd  |.....|
000000b0  6c ed 52 cd ec e5 0f cd  ef e5 0e cd d7 e5 0f cd  |l.R.....|
000000c0  d9 c3 04 cd e5 e5 0f cd  10 c5 0b cd ee e5 0f cd  |.....|
000000d0  52 69 63 68 ef e5 0f cd  00 00 00 00 00 00 00 00  |Rich.....|
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
000000f0  50 45 00 00 4c 01 04 00  0f 82 59 47 00 00 00 00  |PE..L.....YG....|
```

KAN: Forensics memory tracing

- Memory tracing engine KAN:
 - Build on top of KVM, the Linux kernel VM
 - Presented in Recon 2014
- Instead of a single memory snapshot, take a series of snapshots
- Create a coherent overall picture of system behavior (much like debugging/system tracing)
- Capture transient memory data, like
 - Obfuscated code and data
 - Self-modifying code
 - Crypto keys and buffers
 - Short-lived data, like URL's, networking buffers, configuration data, etc.
- More information coming from Endre Bangerter, Security Engineering Lab, Bern University of Applied Sciences

Summing it all: Cuckoo sandbox

- Cuckoo has it all: instrumentation with hooks, emulators, memory forensics, network traffic analysis, automated reporting
- Can utilize several different VM platforms: VMWare, KVM, VirtualBox, or use a custom platform, for example real HW
- Support of volatility analysis after sample run
- Nice reporting
- Used by VirusTotal and many others



**SWITCH
ON
FREEDOM**