Check Point
SOFTWARE TECHNOLOGIES LTD.

Enhancing malware analysis with

Automated Memory Analysis

black hat USA 2014

Adi Hayon
Tomer Teller

Why are we here? (one of many reasons)

- A malicious program:
  - Allocates memory in a remote process (and write to it)
  - Executes the code in that memory region
  - Frees the code
- Memory dump taken at the end of execution
  - **No malicious artifacts found in post-mortem analysis**

Why are we here? (one of many reasons)

- Snake/Uroburos rootkit (MD5: 626576e5f0f85d77c460a322a92bb267)
- Inline interrupt hooks
- Zeroed image header
  - **This evades file carving**

Setting the Context

- Automated system analyzes a new sample
  - Static Analysis - no significant results
  - Dynamic Analysis - no significant results
  - Memory Analysis – limited results
- Evasion tricks are out of scope
- Focus is on memory analysis enhancement

Static Analysis Challenges

- Time consuming
- 35%~ of malicious samples are packed*
- 90%~ of packed files are protected
- Obfuscation, Cryptors, Encrypted Resources

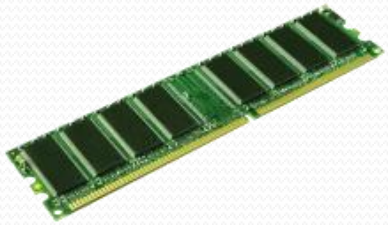
* *https://media.blackhat.com/bh-us-12/Briefings/Branco/BH_US_12_Branco_Scientific_Academic_Slides.pdf*

Dynamic Analysis Challenges

- "What you see is what you get"
- Subverting API functions is easy. APIs Lie.
- Calling undocumented/native functions
- Custom WinAPI function implementations
- Reminder: evading dynamic analysis is out of scope

Memory Analysis Advantages

- Discovers system inconsistencies that might indicate a rootkit
- Collects hidden artifacts that cannot be retrieved using OS-provided API
- Advanced malware operates solely in memory
- Identifies system activity and overall machine state



Memory Analysis Disadvantages

- Current solutions require manual inspection (not scalable)
- Interpreting analysis tools output requires in-depth knowledge of OS internals
- Anti-Forensics tools exist* to:
  - Prevent grabbing of memory dumps
  - Plant fake artifacts in memory as decoys
- Artifacts from a single memory dump lack context, since there is no baseline to compare it with
- Taking memory dumps requires accurate timing as memory is volatile

* *http://scudette.blogspot.co.il/2014/02/anti-forensics-and-memory-analysis.html*

Current Automated Approach

- Execute a sample in a sandbox
- Terminate execution after X minutes
- Grab a memory dump of the machine
- Analyze the memory dump offline
- Detect malicious/suspicious artifacts in-memory
- Revert, Rinse, Repeat

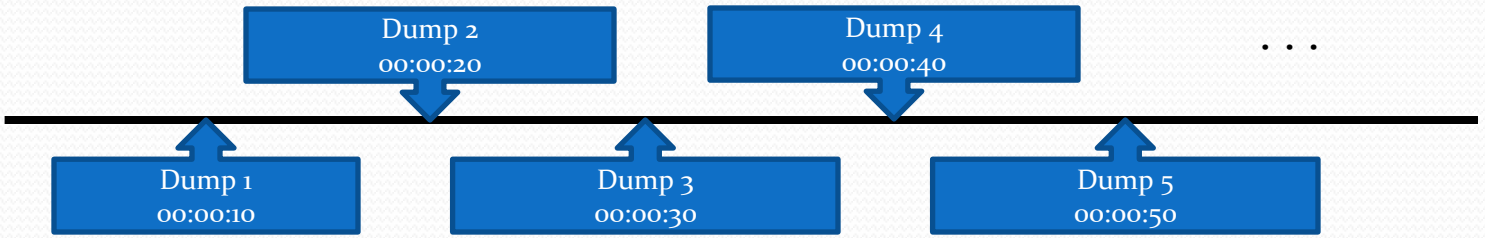
Memory Dump Timing Challenge

- Post-mortem memory dumps (after the program terminates) risks “missing in on the action”
  - Malicious artifacts may appear and disappear intermittently
- Example:
  - Memory region is allocated with RWE permissions
  - Code is written to that region and executed
  - Malware unload itself

  ➔ Detecting the additionally code at the end will fail

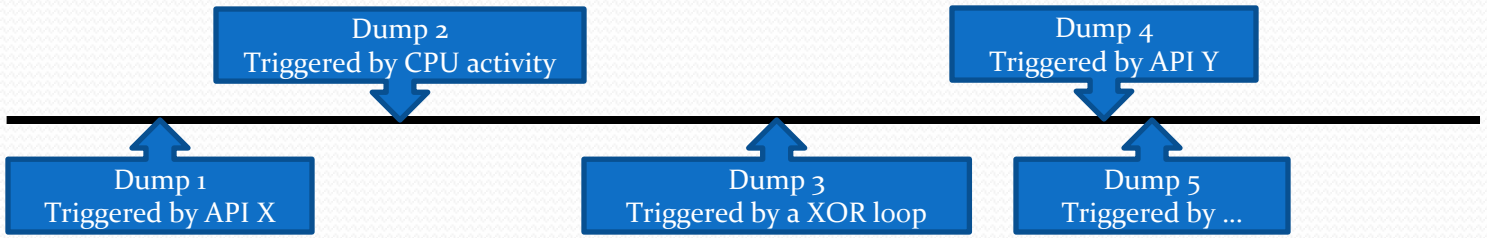
Possible Solution

- **Interval-Based memory dump**
- Grab a memory dump every X seconds
- Analyze each dump - search for malicious artifacts
- Does it solve the problem? **No**
  - Malware can slip between the intervals
  - Many dumps to analyze make it inefficient (Time/Space)



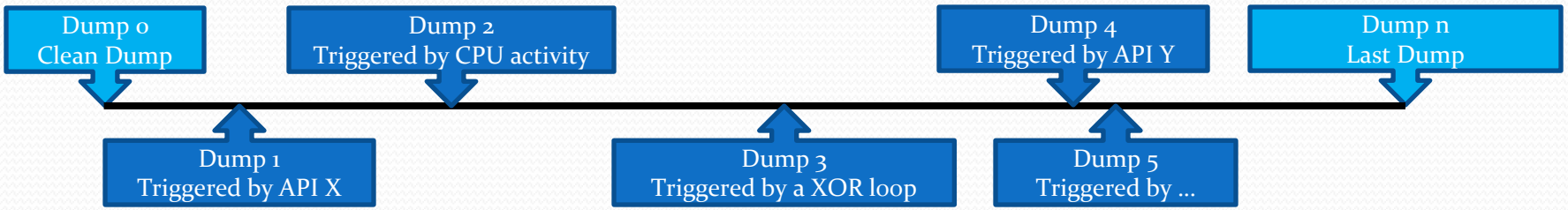
Better Solution

- **Trigger-Based memory dump**
- Dump memory when something “interesting” happens
- “Interesting” points in time:
  - Known malicious API-sequence (behaviors) in user/kernel mode
    (e.g. Code injection, hollow process)
  - Evidence cleaning attempts
    (e.g. Process Termination, Un-mapping memory, etc.)
  - “Heavy” mathematical computation
    (e.g. unpacking in progress)
  - Sampling CPU performance counters for abnormal process activity



Differential Analysis

- Analyze each dump for malicious artifacts
- Diff all dump analysis results from last to clean
  - Clean: Taken before Malware execution
  - Last: Taken when time exceeded
- Produce a list of New/Modified/Deleted artifacts
- Visualize!



Our Approach

- Execute a sample in a controlled environment (CE)
- Trace and monitor execution
- When a trigger is detected
  - Suspend CE -> Dump Memory -> Resume CE
- Before the sample terminates
  - Suspend CE -> Dump Memory -> Terminate CE
- Differential Analysis
  - Clean Dump vs. Dump #1 vs. Dump #2, .. vs. Final Dump
- Generate Report

DEMO #1 - Showcase Malware

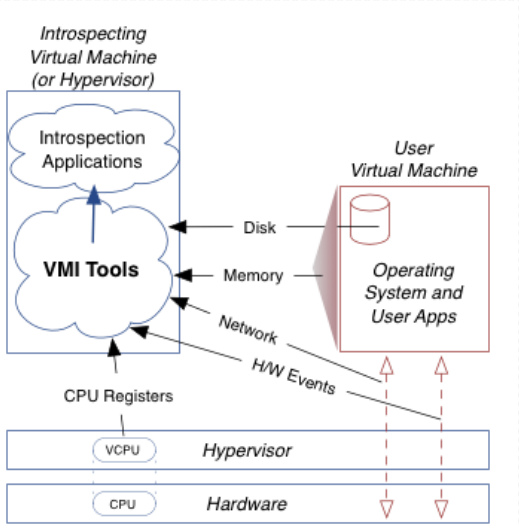
- Trigger-Based vs. Interval-Based
- Differential analysis
- Visualization

Differential Analysis Plugins

- Process Heap Entropy checker
  - Check for entropy changes over time
- Anti Virus Strings
  - Check for new unpacked strings
- Hybrid Data Extractor
  - Comparing code in-memory (dynamic) against the code on disk (static) to detect unpacked code/data
- Modified PE Header
  - Monitor PE header modification and reconstruct it on-the-fly

Taking a (memory) Dump

- Live Memory Introspection (*libVMI*/*pyVMI*)
  - Suspend CE
  - Query memory directly
  - Resume CE
- Offline Memory Dump (*libvirt*)
  - Suspend CE
  - Dump memory to disk
  - Resume CE



https://code.google.com/p/vmitools/

DEMO #2 - Advanced Features

- Trigger-based analysis with VMI
- Hybrid Analysis (Dynamic + Memory)
- Artifact dumper

DEMO #3 – SNAKE/Uroburos Rootkit

- Kernel Triggers
- PE header reconstruction
- Artifact dumper

Implementation

- Modified Cuckoo Sandbox v1.1
  - Modified Cuckoo/CuckooMon components
  - New hooks in User/Kernel Mode
  - New static analysis scripts
  - IDA integration (e.g. calculate MD5/ssdeep per function/section)
  - PinTool integration for DBI
- New Volatility plugins for differential analysis

The techniques are generic and can be applied to any sandbox - Read the WP

https://github.com/djteller/MemoryAnalysis

Future Work

- Brainstorming & Implementing new triggers
- Automatic verdict (malicious/benign)
- Plug-in framework
- Optimization (e.g. grabbing mini-dumps)
- Extend (non-intrusive) VMI capabilities
- Define new operations for misbehavior analysis
- Port solution to other automated malware systems

Thank You

- Slides
- White Paper
- Code

https://github.com/djteller/MemoryAnalysis

@djteller  @adihayon1