Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System

Tamas K. Lengyel
@tklengyel
tamas@tklengyel.com
Agenda

1. Motivation
2. System design
   a. Scalability
   b. Fidelity
   c. Stealth
3. Experimental results
4. Conclusion
Motivation

- Large and continuously growing set of malware samples day-by-day
- Human analysts are expensive and limited
- Static analysis is thwarted by packers / metamorphic malware
- Malware is trying to stay under the radar
Motivation

- Dynamic analysis
  - Observe the execution of the sample in a sandbox
  - Collect behavioral characteristics
  - Collect artifacts (PCAP, unpacked binaries, etc.)
  - Analyst can quickly find the outliers

- Linear hardware requirements
- Rootkits
- Sandbox detection
System design

- Mitigate linear hardware requirements with on-demand resource allocation
  - Scalability

- Monitor execution of both user- and kernel-mode malware without the potential of tampering
  - Fidelity

- Hide monitoring environment from the executing malware
  - Stealth
System design

- Built on Xen and LibVMI
- No manual patches required
- Detection of monitoring is separate from the detection of virtualization!
- Xen is widely deployed
Scalability

Use copy-on-write (CoW) disk and memory
  • Extra resources allocated only when used
VLAN isolation
  • Avoid IP/MAC address collisions
Strong isolation of analysis sessions
Fidelity

Detect when

- New process is scheduled, syscalls executed, files accessed / created / deleted, etc.

Monitor guest OS execution and memory utilization with hardware virtualization extensions

- Breakpoint injection
- Extended Page Tables (EPT) violations
- Monitor Trap Flag (MTF) singlestepping
- MOV-TO-CR trapping
Tamper resistance

Map OS without using heuristics
- Use debug data to find internal kernel functions
- FS/GS $\rightarrow$ _KPCR $\rightarrow$ Kernel!
- Syscalls vs. internal functions
- Allows monitoring kernel-mode rootkits as well

Monitor kernel heap allocations
- Kernel meta-info about processes, files, threads, mutexes, etc.
- Direct state reconstruction without scanning!
Heap tracing

1. Heap allocation request trapped
2. Extract return address from stack
3. Trap when return address is called
4. Retrieve structure address
5. Monitor structure initialization with EPT permissions and MTF singlestepping
Stealth

No in-guest agents
  - No in-guest artifacts

Protect injected breakpoints with EPT permissions

How to start the execution of the sample?
  - Hijack a running a process!
  - Can use any active process
  - No artifacts left afterwards
Experimental results

Malwares analyzed
- Zeus, TDL4, SpyBot, Careto, etc.
- 1000 samples from ShadowServer

Heap allocations are clearly in the fast-path of the kernel execution

In-memory only files
- Never flushed to disk
- Only accessible via memory
- Only for a short period of time
Experimental results

Overhead
- Doesn’t matter as long as the malware can’t detect the monitoring environment
- We don’t know how long we need to execute a sample anyway

Throughput
- Doubled the number of sessions!
- Can be further optimized with run-time memory deduplication
Future work

- DRAKVUF specific
  - Multi-vCPU support
  - Linux support

- General dynamic malware analysis problems
  - Stalling code
  - Record & Replay
  - Branch exploration
  - Timing attacks
Conclusion

- Dynamic analysis can aid in sorting through large sets of malware
- Existing tools are limited and can be tampered with
- DRAKVUF raises the bar
  - Open source (GPLv2)
  - [http://drakvuf.com](http://drakvuf.com)

- Thanks! Questions?