Tracking Rootkit Footprints with a Practical Memory Analysis System

Weidong Cui (MSR)
Marcus Peinado (MSR)
Zhilei Xu (MIT)
Ellick Chan (UIUC)

21st USENIX Security Symposium

Kernel Rootkit Footprints

Memory changes a kernel rootkit makes for

- Hijacking code execution
- Hiding its activities

Kernel Rootkit Hooking

- Directly modify code
  - E.g., insert a JMP instruction
  - Easy to check
- Manipulate a function pointer in a data structure
  - Easy to check static data
  - Dynamic data is the challenge!
- Hooking a *single* function pointer may be enough for an attack
- We need to check all function pointers

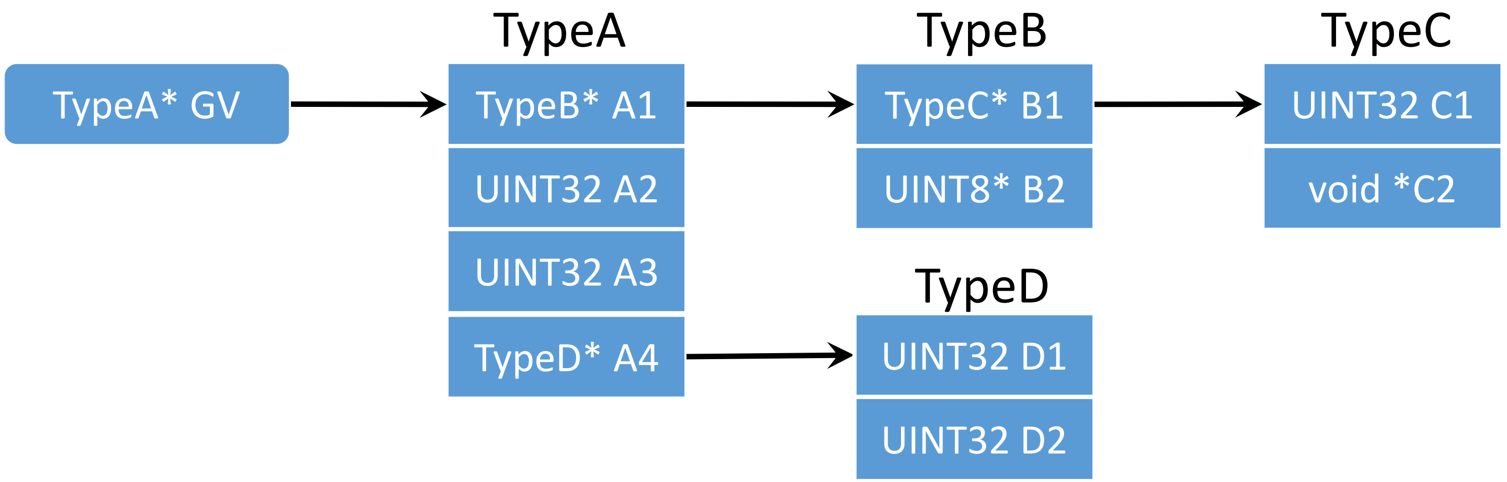
Challenge: Identify all dynamic data to check all function pointers

Kernel Rootkit in Memory

- A needle in a haystack!
- A typical Windows 7 kernel has
  - 100+ loaded modules
  - 100K to 1M+ data objects
  - 100K+ function pointers

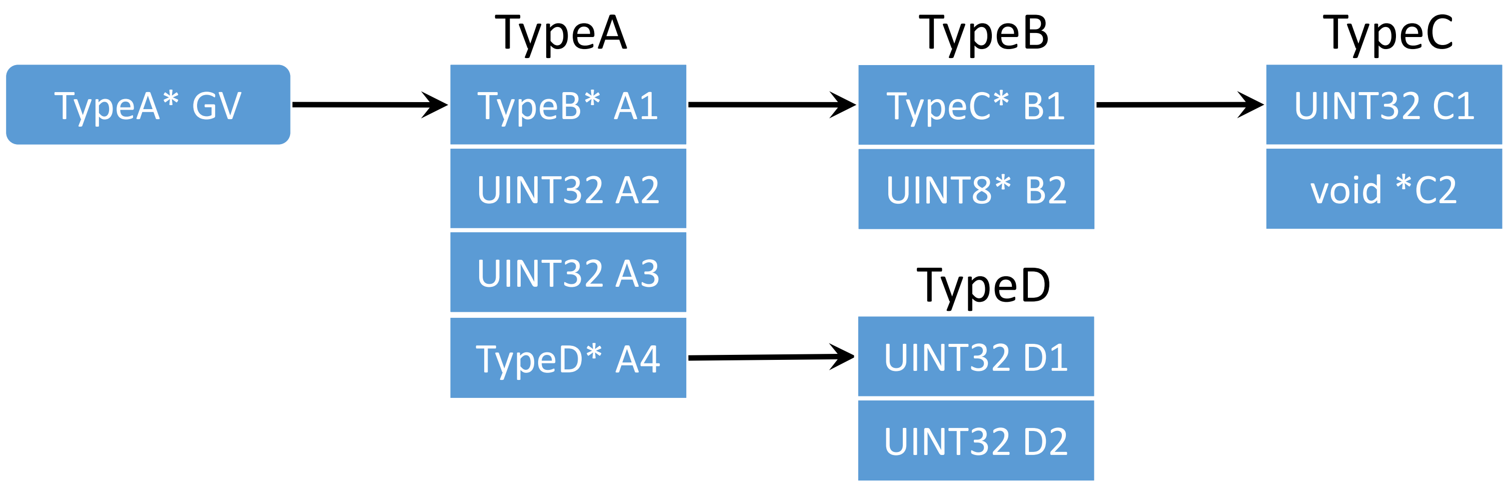
How to find all the data and function pointers?

Basic Memory Traversal



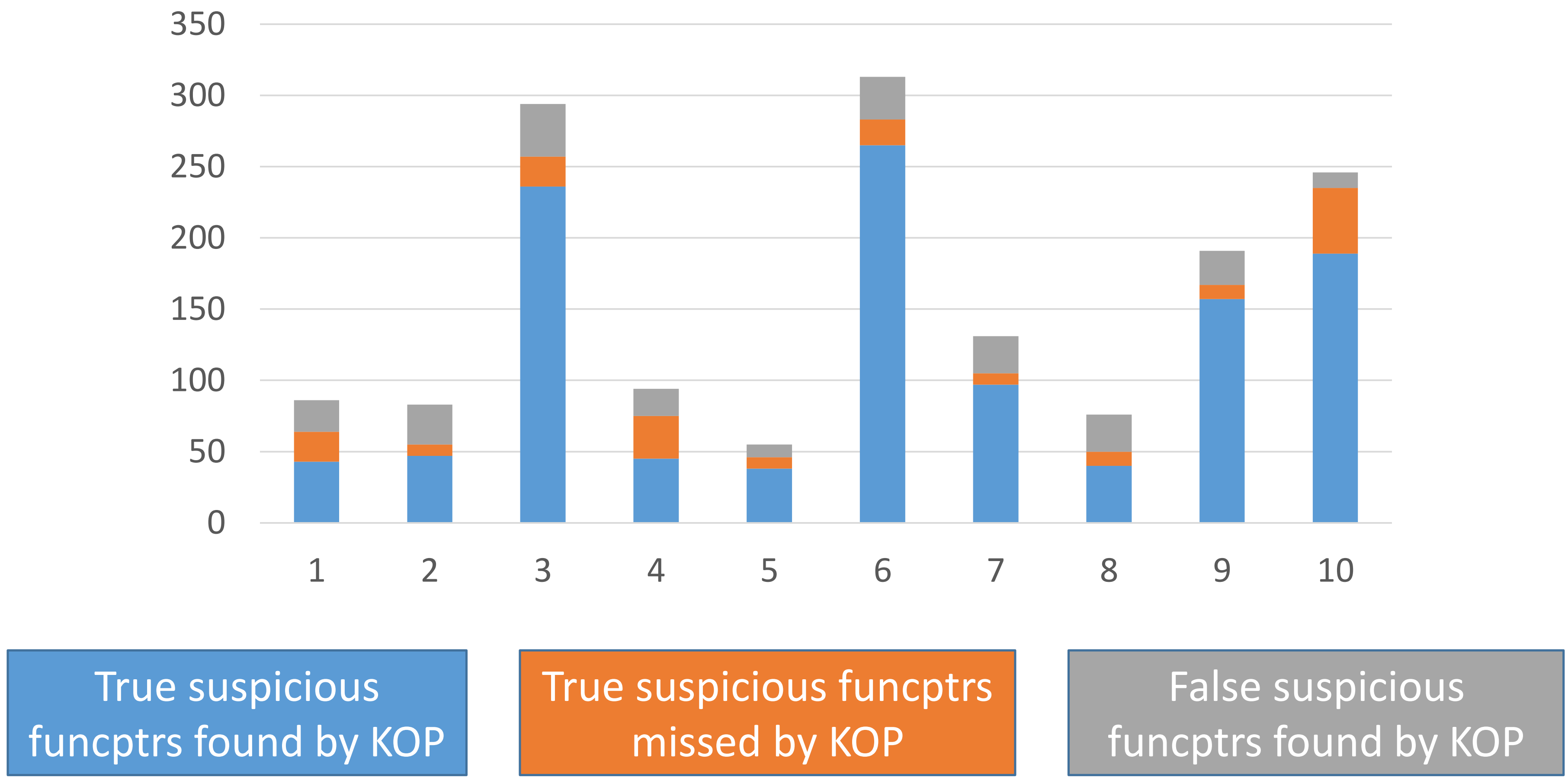
- [SBCFI: Petroni07], [Gibraltar: Baliga08], [KOP: Carbone09]
- KOP uses static analysis to infer types for generic pointers

What if a Pointer is Invalid?



Errors are propagated and accumulate!

KOP on 10 Real-World Crash Dumps



Pointer Uncertainty is Unavoidable

- Invalid pointers
  - Uninitialized pointers
  - Corrupted pointers

- Ambiguous pointers
  - Pointers in unions
  - Generic pointers with multiple candidate types

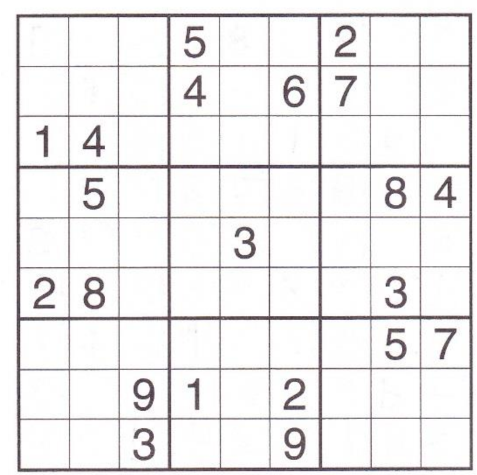
We must handle pointer uncertainty effectively!

MAS: A Practical Memory Analysis System

- Accurate: find all rootkit footprints
- Robust: handle real-world snapshots
- Fast: finish in just minutes

How does MAS Handle Pointer Uncertainty?

- Identify data objects without following pointers (as much as possible)
- Ignore pointers with ambiguous types
- Check all available constraints before following a pointer
- Support error correction in memory traversal



Fast and Precise Pointer Analysis

- Demand-driven
- Partially flow sensitive (SSA)
- Context-sensitive
- Field-sensitive

Identifying Data Objects by Pool Tags

- Pool tags are a feature of Windows
  - Similar features available in Linux
- Many pool tags are associated with a unique data type
  - E.g., “Irp “ is for IRP
- Use static analysis to infer relationship between pool tags and data types

```
FOO* f = (FOO*) ExAllocatePoolWithTag( NonPagedPool, sizeof( FOO ), ‘DooF’ );
```

Ignoring Ambiguous Pointers

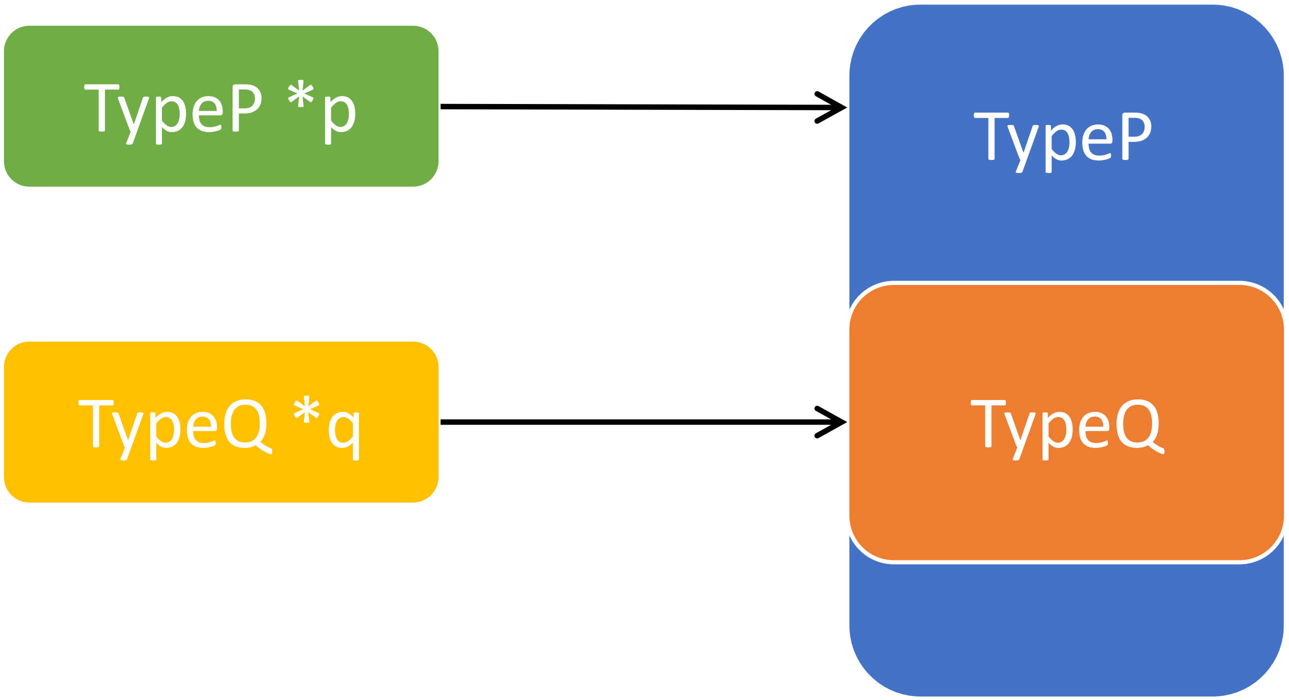
- Resolving type ambiguities with heuristics is bound to have errors
- Only follow pointers with unique types

Constraints for Data Objects

- Size constraint
- Pointer constraint
- Enum constraint
- Pool tag constraint

Type Constraint

- The type layouts of two overlapped data objects must match



What if they don’t match?

Error Correction

- If two overlapped data objects have type mismatch
  - If one object was found without following pointers, keep it
  - Otherwise, keep the larger object
- When removing an existing object
  - Remove all the objects that are only reachable from the removed object

Integrity Checking

- Code Integrity
- Function Pointer Integrity
- Visibility Integrity

Implementation

- Static analysis
  - 12K lines of C++ code
  - Developed a PREfast plugin to extract information
- Memory traversal and integrity checking
  - 24K lines of C/C++ code
  - Worked as a debugger extension for WinDbg

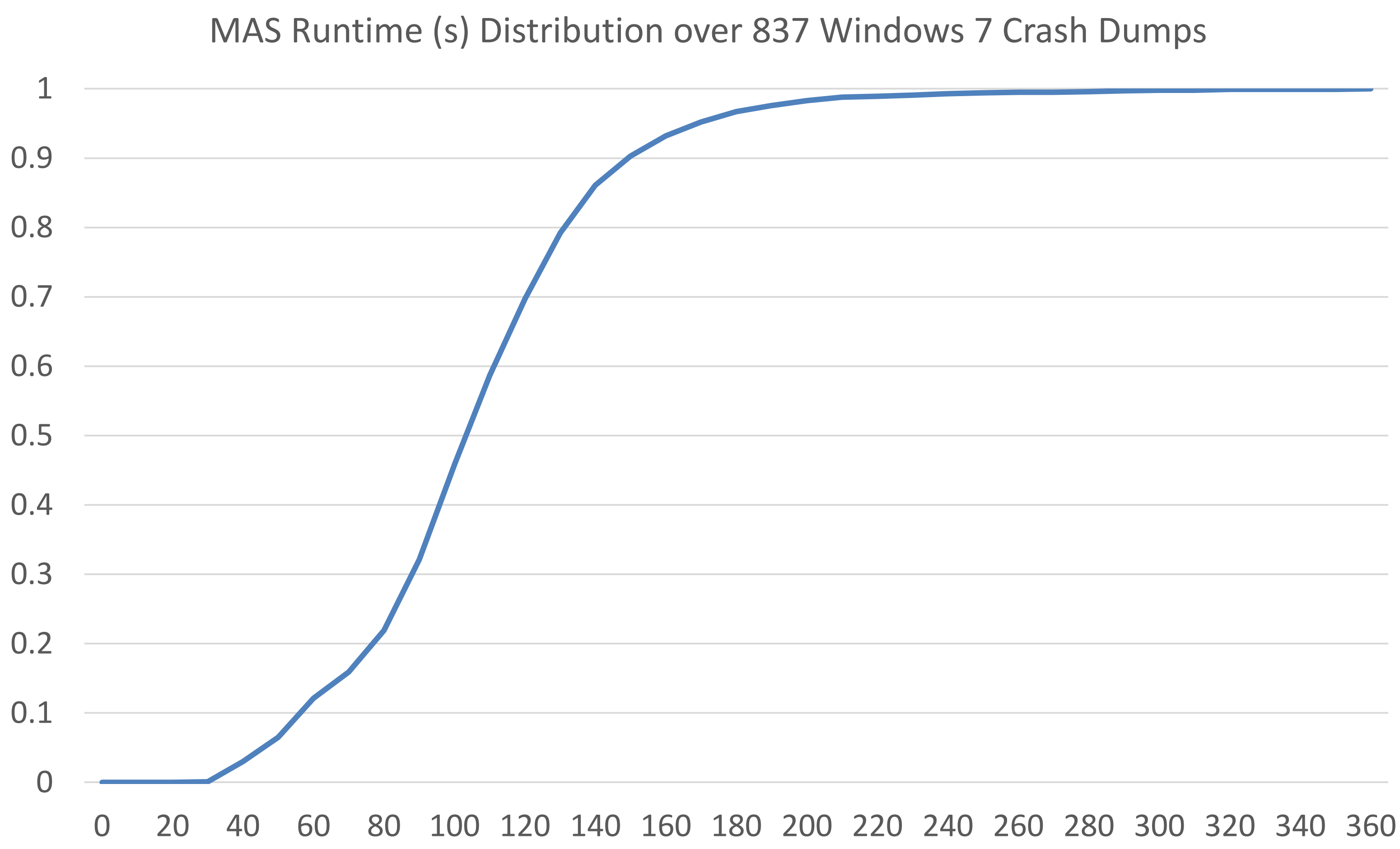
Real-World Data Sets

- 11 Windows Vista SP1 crash dumps
- 837 Window 7 crash dumps
- 154,768 kernel malware samples

Accuracy

- For 10 Windows Vista SP1 crash dumps
  - All suspicious function pointers found by MAS are true function pointers
  - All true suspicious function pointers found by KOP are found by MAS
- For 837 Windows 7 crash dumps
  - We verified that all but 24 out of 400K suspicious function pointers are true function pointers

Performance



Detecting Rootkits in Crash Dumps

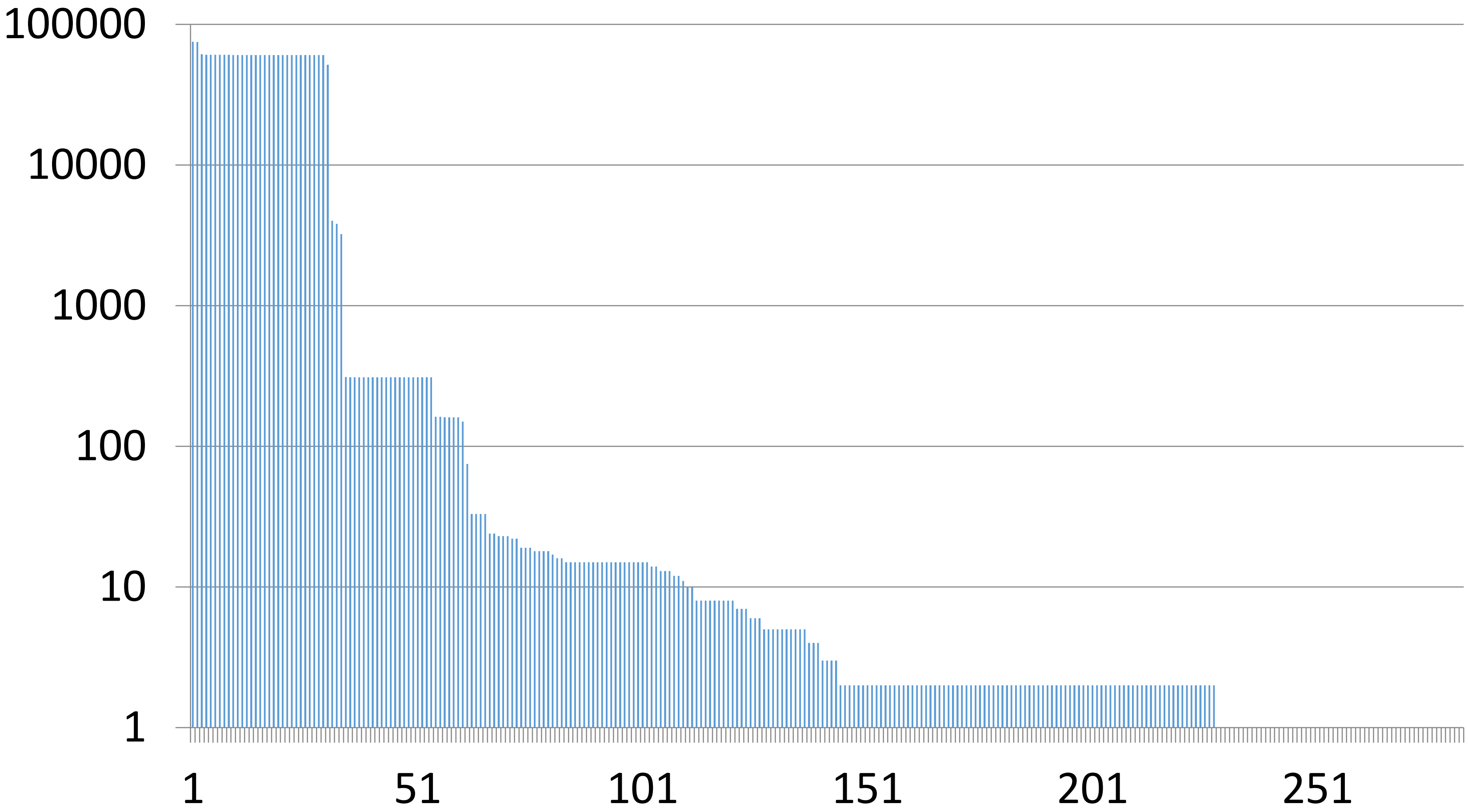
- Cannot fully automate it because of third-party drivers
  - Ignore suspicious function pointers to unknown modules
  - Took one hour of manual effort

	# of Crash Dumps
Total	848
Only funcptrs to unknown modules	664
Anti-virus software	84
Rootkits	95
Corrupted	5

Malware Study

- 191 unique function pointers
- 31 different data structures
- NTOS kernel + 5 different modules

Malware Clustering



Summary

- MAS is a practical memory analysis system for detecting and analyzing kernel rootkits
  - Handles pointer uncertainty effectively
- Applied MAS to 848 real-world crash dumps
  - Found 95 of them have rootkits
- Large-scale study of 150K malware samples
  - Hooked 191 unique functions pointers in 31 data structures of 6 modules