On Correctness, Robustness and Coverage of Memory Analysis

Heng Yin
Assistant Professor

Department of Electrical Engineering and Computer Science,
Syracuse University

What is memory analysis?

- Input:
  - A memory snapshot (or dump) of a running physical machine or virtual machine
- Analysis:
  - Scan data structure signatures
  - Traverse data structures
- Output:
  - Semantic knowledge extracted from the snapshot
  - Examples: running processes, loaded modules, network connections, …

Existing Work

- Signature Scanning
  - Signature tools in Volatility
  - Robust signature [Dolan-Gavitt et al. CCS 2009]
  - Graph signature [Lin et al. NDSS 2011]
- Data Structure Traversal
  - Traversal tools in Volatility
  - KOP/MAS from Microsoft Research

Why is it useful?

- Digital Forensics
  - Collect crime evidence
- Virtual Machine Introspection
  - Monitor VM activities from outside, desired for IaaS cloud providers
- Malware Detection
  - Find malware (especially kernel rootkit) footprint in memory

Unique Advantage: Code and data must be loaded into memory to be executed and accessed

What are the challenges?

- Semantic Gap
  - Data structure definitions (Limited documentation for closed-source OSes)
  - Global variables (The roots for traversal)
  - Generic pointers (void *, struct list_head, LIST_ENTRY)
  - Consequence: low coverage and reduced accuracy
- Memory Manipulation Attacks
  - Pointer manipulation (add/remove/change a pointer, unlink an object)
  - Value manipulation (obtain fraudulent info)
  - Consequence: low robustness and low trustworthiness

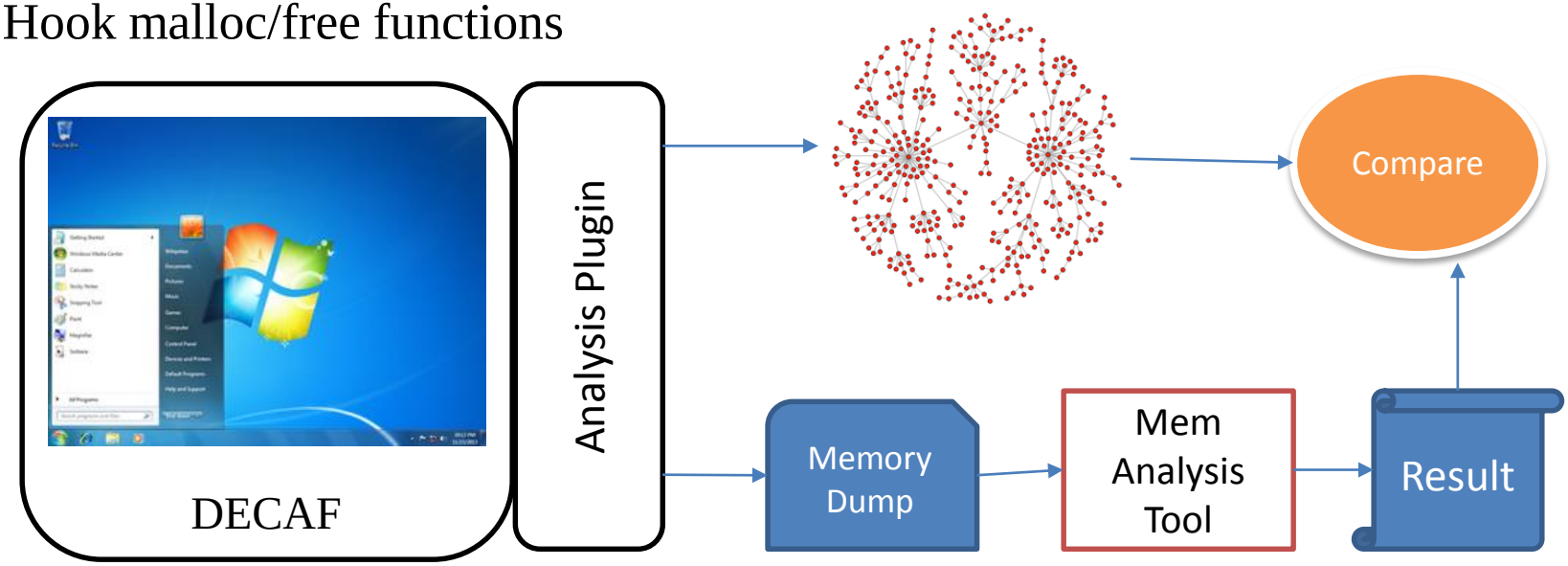
Questions to Investigate

- Q1: Correctness (in non-deceptive settings)
  - How can we know if a memory analysis tool produces correct results, especially for closed-source OSes?
- Q2: Robustness (in deceptive settings)
  - To what extent a memory analysis tool can still produce correct results, if the memory snapshot has been compromised?
- Q3: Improving the state-of-the-art
  - Can we develop a better memory analysis tool (Higher coverage, more robust)?

We focus on OS kernel space. User-level memory analysis is beyond the scope for now.

Q1: Correctness Study

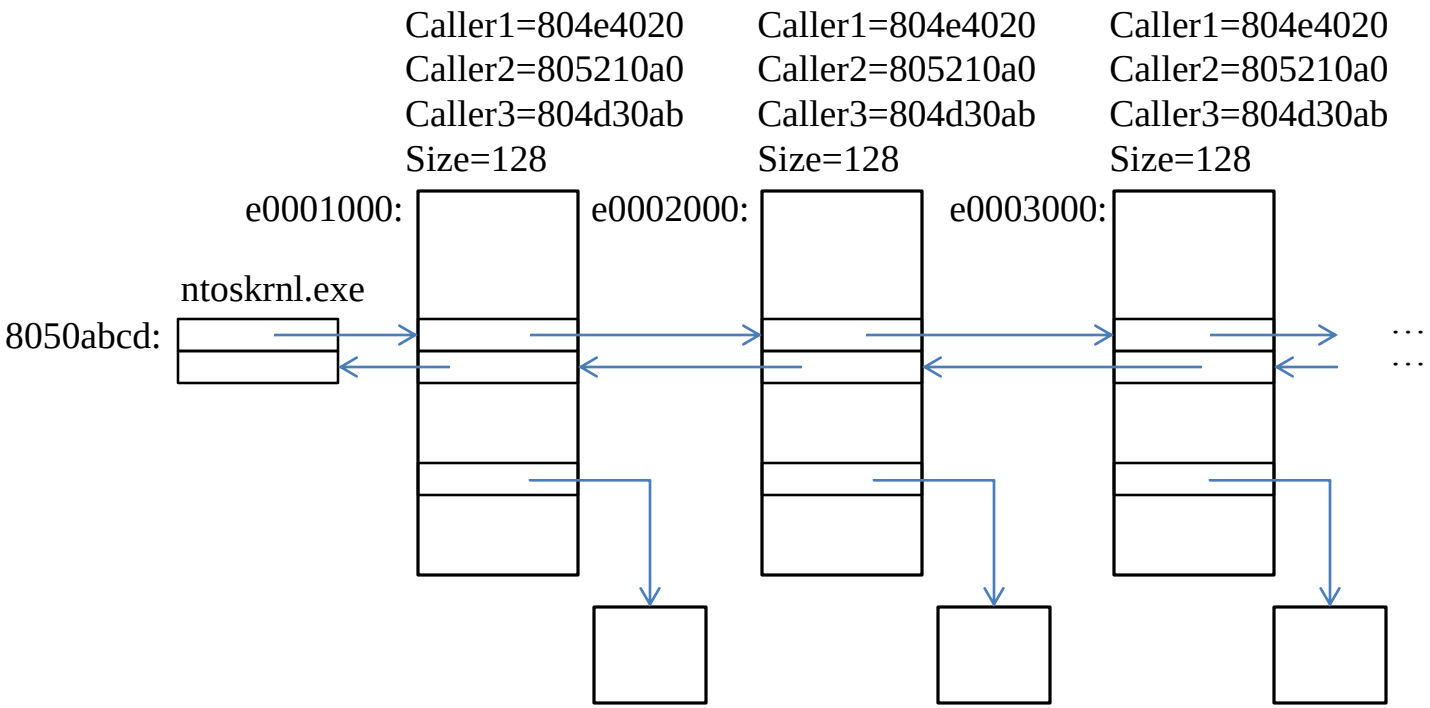
- How to obtain the ground truth: Kernel Data Structure Graph
- We construct it on the fly
  - Whole-system dynamic binary analysis
  - Use DECAF, an open-source platform
  - Hook malloc/free functions



https://code.google.com/p/decaf-platform/

An example graph

- How do we map these objects into their types?



Mapping Caller List to Type

- Rationale: Objects allocated at the same calling context have the same type

Caller List to Type Mapping in Windows 7

Object	Caller 3	Caller 2	Caller 1	Size	in Volatility	in DSG
_EPROCESS	PspCreateProcess	ObpCreateObject	ObpAllocateObject	728	2	2
_EPROCESS	PspCreateProcess	ObpCreateObject	ObpAllocateObject	744	30	30
_ETHREAD	PspAllocateThread	ObpCreateObject	ObpAllocateObject	736	413	413
_DRIVER_OBJECT	IoCreateDriver	ObpCreateObject	ObpAllocateObject	252	1	1
_DRIVER_OBJECT	IoCreateDriver	ObpCreateObject	ObpAllocateObject	236	2	3
_DRIVER_OBJECT	IopLoadDriver	ObpCreateObject	ObpAllocateObject	236	64	64
_DRIVER_OBJECT	EtwpStartAutoLogger	EtwpAllocateTraceBufferPool	EtwpAllocateFreeBuffers	65536	1	6
_DRIVER_OBJECT	IopInitializeBuiltinDriver	ObpCreateObject	ObpAllocateObject	236	32	32
_DRIVER_OBJECT	NtCreateMutant	ObpCreateObject	ObpAllocateObject	88	24	24
_KMUTANT	NtCreateMutant	ObpCreateObject	ObpAllocateObject	72	234	234
_FILE_OBJECT	IoCreateStreamFileObjectLite	ObpCreateObject	ObpAllocateObject	160	43	67
_FILE_OBJECT	IoCreateStreamFileObjectLite	ObpCreateObject	ObpAllocateObject	176	136	179
_FILE_OBJECT	IopAllocateRealFileObject	ObpCreateObject	ObpAllocateObject	176	2025	2135
_FILE_OBJECT	IopAllocateRealFileObject	ObpCreateObject	ObpAllocateObject	160	182	321
_FILE_OBJECT	EtwpStartAutoLogger	EtwpAllocateTraceBufferPool	EtwpAllocateFreeBuffers	65536	3	6

We obtain debug symbols to map these callers to kernel function names

Correctness for Basic Volatility Tools

Commands	WinXP-SP3				Win7-SP0			
	DSG	Vol.	FN	FP	DSG	Vol.	FN	FP
pslist	22	22	0	0	32	32	0	0
psscan	22	22	0	0	32	32	0	0
pstree	22	22	0	0	32	32	0	0
sockets	21	21	3	3	-	-	-	-
sockscan	21	22	0	1	-	-	-	-
connections	6	6	0	0	-	-	-	-
connscan	6	6	0	0	-	-	-	-
filescan	1586	1807	6	0	2709	2398	0	0
driverscan	74	74	0	0	106	100	0	0
thrdscan	325	326	0	1	413	422	0	9
mutantscan	149	149	0	0	258	258	0	0
netscan	-	-	-	-	68	70	0	2

- False positives are quite common in several tools, because signature patterns are not strong enough.
- 6 false negatives in filescan are due to an implementation error. We reported a patch to Volatility.

Correctness and Efficiency for Robust Signatures

Tool	512 MB			1 GB		
	Time	Objs	FN/FP	Time	Objs	FN/FP
SigField	641s	25	0/0	1283s	28	0/0
SigField_opt	341s	25	0/0	715s	28	0/0
SigGraph	494s	25	0/0	1006s	28	0/0

- We did not observe any errors
- Efficiency is quite low: spend several minutes to scan one memory dump, over 10 minutes for big dumps (1GB)
  - Cannot use these schemes for realtime VM scanning
- Basic signature tools are much faster
  - Only a few seconds
  - Skip large memory regions based on unreliable heuristics

Q1: Summary

- We obtained the ground truth by
  - dynamically monitoring kernel memory allocation and de-allocation
  - performing type inference to map these objects to their types
- Signature tools often raise false positives
  - Devise a good signature is hard
  - How to determine if an object is dead?
- Even one tool has an implementation error
  - Due to incorrect understanding of kernel data structure definition
- Robust signature schemes are slow!
  - Has to examine every byte at least once

Q2: Robustness Study

- Deceptive scenario
  - The OS kernel has been compromised
  - The attacker can write to arbitrary memory locations to deceive security analysis
  - The attacker does not want to crash the system
- Questions
  - Which data structure fields can be modified (mutable)?
  - How can these mutations affect memory analysis tools?

Q2: Our Approach

- Automatic Mutation Testing
  1. Start a test program inside VM
  2. Save and pause the VM
  3. Locate a kernel data structure belonging to that test program
  4. Select a field to mutate
  5. Resume the VM to finish the test program

     (system may crash or the program may terminate prematurely)
  6. Go to Step 2

Q2: Test Cases and Mutation Rules

No	Test Case
1	Start test program Test Point 1: mutate process & thread related values Run other test cases
2	Load a user DLL Test Point 2: mutate DLL related values Call a function in the DLL repeatedly Unload the DLL
3	Load a kernel module Test Point 3: mutate kernel module values Send IO requests to the kernel module Unload the kernel module
4	Open two files, one each for read and write Test Point 4: mutate file values Read and write the two files repeatedly close the files
5	Open a TCP connection Test Point 5: mutate values related to this connection Send and receive data through this connection Close the connection
6	Open a registry key (Windows only) Test Point 6: Mutate registry key related values Read and write this registry key repeatedly Close the key

Type	Mutation Rules
ID	0, copy from another ID, increment or decrement by a small constant
Size/Offset	0, increment or decrement by a small constant
String	“”, copy from another string, mutate one character

Q2: Our Findings

Category: Structures	Semantic Field	Mutability
Process: struct EPROCESS	UniqueProcessId, ExitStatus, ImageFileName, CreateTime, GrantedAccess, InheritedFromUniqueProcessId, ObjectTable.HandleCount, ObjectHeader.ObjectType	✓
	ActiveThreads, Flags	✗
	Token	*p*
Thread: struct ETHREAD	StartAddress, Cid.UniqueThread, ObjectHeader.ObjectType	✓
	Cid.UniqueProcess	✗
DLL & Kernel Module: struct LDR_DATA_TABLE_ENTRY	DllBase, EntryPoint, FullDllName, BaseDllName, Flags, LoadCount, PatchInfo	✓
Registry Key, CM_KEY_NODE	Name, NameLength, LastWriteTime, SubkeyCounts, Flags, Signature, Parent	✓
	Security	✗
Network	TCPT_OBJECT.RemoteIpAddress, TCPT_OBJECT.RemotePort, TCPT_OBJECT.LocalAddress, TCPT_OBJECT.LocalPort, TCPT_OBJECT.Pid	✗
	TCP_LISTENER.AddressFamily, TCP_LISTENER.Owner, TCP_LISTENER.CreateTime, TCP_ENDPOINT.State	✓
Memory Pool: struct POOL_HEADER	PoolTag, BlockSize	✓

Windows XP SP3

Category: Structures	Semantic Field	Mutability
Process: struct task_struct	state, flags, comm, start_time, stime, exit_code	✓
	fds	✗
	pid	*p*
File: struct dentry struct inode	task_struct.files.fd[i].f_owner, task_struct.files.fd[i].f_mode, task_struct.files.fd[i].f_pos, d_name, d_iname, d_flags, d_time, i_uid, i_gid, i_size, i_atime, i_ctime, i_mtime	✓
Module: struct module, struct vm_area_start	name, num_syms, state, core_size, core_text_size, num_kp, vm_flags	✓
	vm_start, vm_end,	✗
Network: struct inet_sock, struct sock_common, struct sock	saddr, daddr, sport, dport	✗
	skc_family, skc_refcount, skc_state, sk_protocol, sk_flags, sk_type, sk_err	✓

Linux 2.6

Many fields are mutable, and thus untrustworthy. Bad news!

Q2: Insights

- Network related values are not mutable
  - Local and remote IP addresses and ports
- Process ID is mutable
  - In Windows, PID in EThread is not mutable
- Strings are mutable
  - Process name, file name, module name, pool tag, etc.
- Timestamps are mutable
  - E.g. process creation and termination time

Q3: Improving the state-of-the-art

- Q3.1: Can we leverage information redundancy to defeat value manipulation attacks?
  - E.g., PID appears in multiple data structures. Attackers need to modify all these copies for complete deception
- Q3.2: Can we improve the coverage and accuracy of data structure identification from a global view?
  - Prior approaches evaluate each data structure individually

Q3.1: How to find duplicate values in kernel space?

- Trace kernel execution
- For each insn, perform bidirectional data flow analysis to track duplicate values

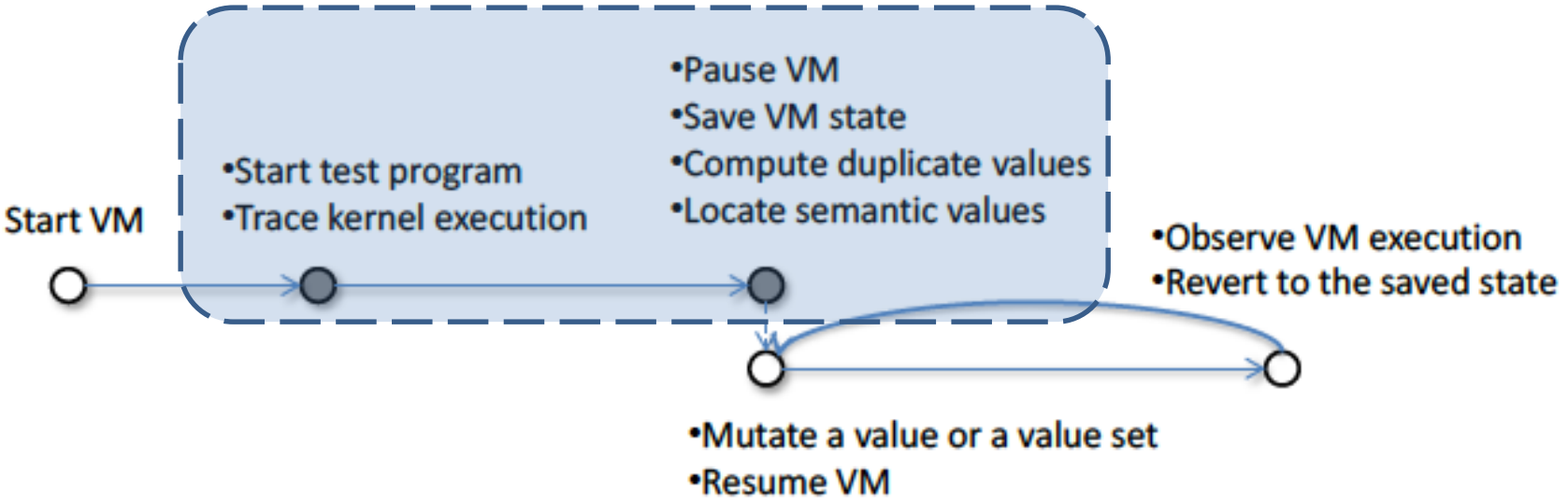
Algorithm 1 Dynamic Duplicate Value Analysis

```
procedure DynValueAnalysis(Trace t)
    for all instruction i ∈ t do
        if i.type is assignment operation then
            for each src & dst byte pair(u, v) do
                DoAssign(u, v)
            end for
        else
            for each byte v in the dst operand do
                DoRemove(v)
            end for
        end if
    end for
end procedure

procedure DoAssign(u, v)
    for all variable r ∈ S_v do
        S_r ← S_r − {v}
    end for
    for all variable r ∈ S_u do
        S_r ← S_r + {v}
    end for
    S_v ← S_u
end procedure

procedure DoRemove(v)
    for all variable r ∈ S_v do
        S_r ← S_r − {v}
    end for
end procedure
```

Q3.1: Testing Procedure



Q3.1: Mutating duplicate values in Windows XP

Primary Field	# of Dups	Type of Duplicates	Immutable Duplicates	Set Mutability
_EPROCESS.UniqueProcessId	36	_ETHREAD.Cid.UniqueProcess, _HANDLE_TABLE.UniqueProcessId, _CM_KEY_BODY.ProcessId, _EPROCESS.InheritedFromUniqueProcessId, _ETIMER.Lock, _TEB.ClientId, _TEB.RealClientId, 0x9b57b6d0, 0x9ccdaef0, 0x9cce697c...	_ETHREAD.Cid.UniqueProcess	✗
_EPROCESS.ImageFileName	4	_OBJECT_NAME_INFORMATION.Name, _RTL_USER_PROC_PARAMS.ImagePathName, _SE_AUDIT_PROCESS_INFO.ImageFileName	None	✓
_EPROCESS.CreateTime	2	_ETHREAD.CreateTime	None	✓
_EPROCESS.ActiveThreads	2	_EPROCESS.ActiveThreadsHighWatermark	None	✓
_HANDLE_TABLE.HandleCount	2	_HANDLE_TABLE.HandleCountHighWatermark	None	✓
_FILE_OBJECT.FileName (Data file)	7	0x003a948e, 0x822df33a, 0x822df35c, ...	None	✓
_LDR_DATA_TABLE_ENTRY.FullDllName	3	_LDR_DATA_TABLE_ENTRY.BaseDllName, _FILE_OBJECT.FileName	None	✓
_LDR_DATA_TABLE_ENTRY.BaseDllName	3	_LDR_DATA_TABLE_ENTRY.FullDllName, _FILE_OBJECT.FileName	None	✓
_CM_KEY_NODE.LastWriteTime	2	0x9b43ea60	None	✓
_CM_KEY_NODE.Parent	4	0x94d20a20, 0x9adc7940, 0x9adc7948	None	✓
_CM_KEY_NODE.Security	2	0x822c7880	_CM_KEY_NODE.Security	✗
_ETHREAD.StartAddress	2	_SECTION_OBJECT.StartingVa	_SECTION_OBJECT.StartingVa	✗

Q3.1: Mutating duplicate values in Linux

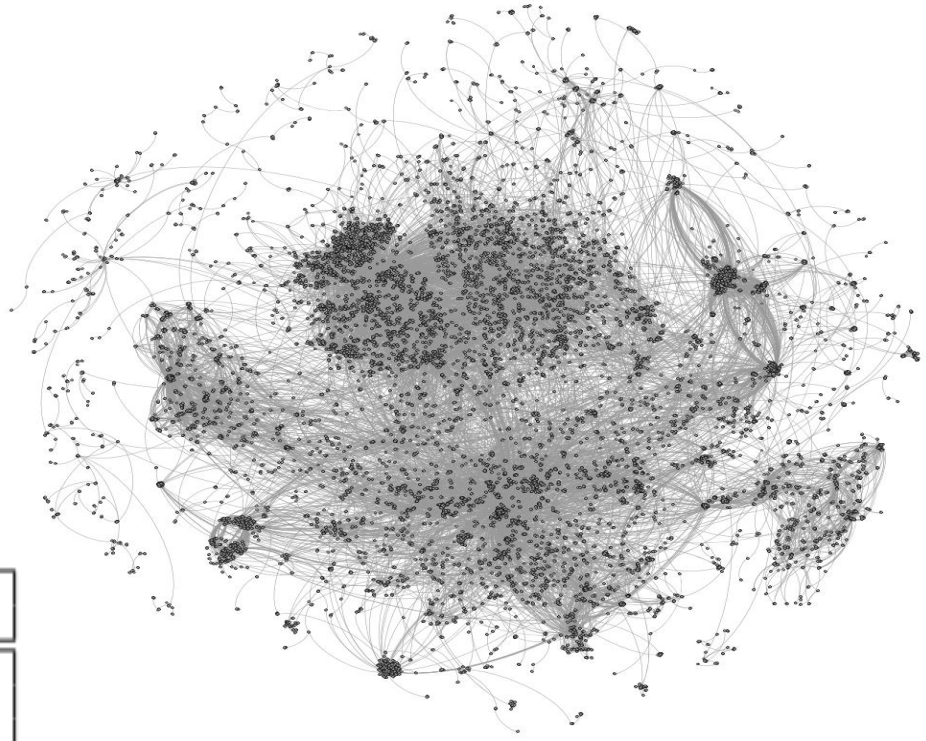
Primary Field	# of Dups	Type of Duplicates	Immutable Duplicates	Set Mutability
task_struct.pid	4	task_struct.t_gid, task_struct.t_gid(lwp), 0xf63916dc	None	✓
task_struct.comm	2	task_struct.comm(lwp)	None	✓
task_struct.static_prio	3	task.parent.static_prio, task.static_prio (lwp)	None	✓
task_struct.exit_code	3	task.parent.exit_code, task.exit_code (lwp)	None	✓
task_struct.fds	3	0xf7179080, 0xf61bae84	0xf7179080, task.fds	✗
module.name	2	0xd93c524c	None	✓
module.num_syms	12	module.num_kp, 0xe086c15c, 0xe086c170...	None	✓
vma.vm_start	2	vma.vm_end	vma.vm_start	✗
vma.vm_end	2	vma.vm_start	vma.vm_end	✗
dentry.d_name	2	0xf583f0d8	None	✓
inet_sock.saddr	24	inet_sock.rcv_saddr inet_sock.daddr 0xde49147c 0xde49148c ...	inet_sock.rcv_saddr inet_sock.daddr 0xde49147c 0xde49148c ...	✗
inet_sock.daddr	24	inet_sock.rcv_saddr inet_sock.saddr 0xde49147c 0xde49148c ...	inet_sock.rcv_saddr inet_sock.daddr 0xde49147c 0xde49148c ...	✗

Q3.1: Observations

- Duplicate values do exist for many important semantic values
- Unfortunately, most of these duplicate values are mutable both collectively and individually
- In very limited cases, checking duplicate values can be helpful to defeat value manipulation attacks

Q3.2: A Network Perspective

- Kernel data structure graph is a small-world network



A kernel data structure graph for Windows XP

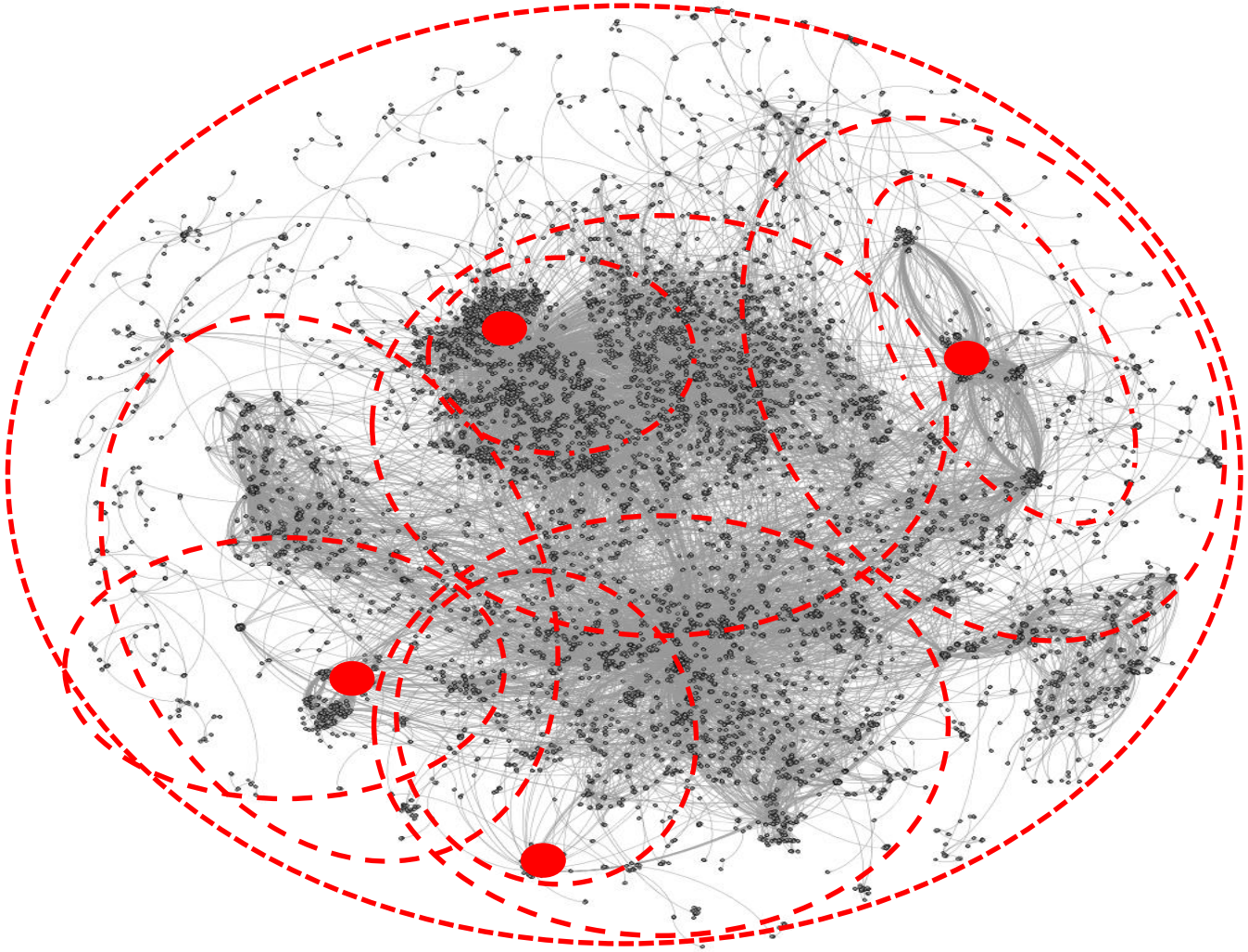
5 dumps for Windows XP

	Nodes	Links	Diameter	Center	Clustering Coefficient
1	18249	173153	2.81	ntoskrnl.exe	0.070
2	22713	205683	3.77	ntoskrnl.exe	0.067
3	20078	221224	2.81	ntoskrnl.exe	0.074
4	29811	222231	3.11	CMCa	0.061
5	33277	200816	5.01	CMCa	0.053

Q3.2: A Network Perspective (cont'd)

- Points-to relation reveals the objects' types
  - Deterministically (for typed pointers)
  - Probabilistically (for generic pointers)
- We perform supervised learning
  - Use DECAF to collect labeled memory dumps
  - Construct a pointer-constraint model
- We perform network-based inference
  - Adapt random surfer model (a page rank algorithm)

Q3.2: Scan, Traverse and Infer



More details…

Will be presented in Thursday 11am

10:00-10:30	☑ **Break** (Foyer)	
10:30-12:00	**Orleans A**	**Orleans B**
	☐ **Panel: The Attacker Among Us: Insider Threats Within the Energy Sector** Moderator: Dr. William (Bill) Claycomb, CERT Insider Threat Center- Carnegie Mellon University Panelists (listed alphabetically): Dr. Nader Mehravari, Cyber Security Solutions, Software Engineering Institute Dr. Shawn Taylor, Sandia National Laboratories Mr. Randy Trzeciak, CERT Insider Threat Center - Carnegie Mellon University	☐ **Securing Memory and Storage** *Chair:*Cristina Serban *SEER: Practical Memory Virus Scanning as a Service* Jason Gionta, North Carolina State University; Ahmed Azab, Samsung Electronics Co., Ltd.; William Enck, North Carolina State University; Peng Ning, North Carolina State University; Xiaolan Zhang, Google Inc *MACE: High-Coverage and Robust Memory Analysis For Commodity Operating Systems* Qian Feng, Syracuse University; Aravind Prakash, Syracuse University; Heng Yin, Syracuse University; Zhiqiang Lin, University of Texas at Dallas *Assisted Deletion of Related Content* Hubert Ritzdorf, ETH Zurich; Nikolaos Karapanos, ETH Zurich; Srdjan Capkun, ETH Zurich
12:00-13:30	☑ **Lunch** (Lafitte AB)	

Q3.2: Result Highlights

- Achieve > 95% coverage for WinXP/Win7
  - Include both documented and undocumented objects
- With 80% typed pointers removed, our coverage degradation is negligible
  - Thanks to the small-world network
- Outperform Volatility
  - Detect rootkit footprint in undocumented objects
  - Resilient against pool tag manipulations
  - In contrast, volatility heavily relies on pool tags and thus is completely defeated

Summary

- We conducted a systematic study on memory analysis
  - With respect to correctness and robustness
  - Explore ways to improve it
- We made the following conclusions
  - The existing tools may produce erroneous results, and may even have implementation errors.
  - The robustness of these tools are questionable, given that attackers can freely manipulate many semantic values
  - Exploiting duplicate values can improve the robustness, but the improvement is marginal
  - Exploiting pointer relations is a promising direction. It can improve coverage and defeat pointer manipulation attacks.

For more details, please read

- “On the Trustworthiness of Memory Analysis --- An Empirical Study from the Perspective of Binary Execution”, IEEE Transactions on Dependable and Secure Computing
- “MACE: High-Coverage and Robust Memory Analysis for Commodity Operating Systems”, ACSAC 2014.

Questions?