# An OS-agnostic Approach to Memory Forensics

Andrea Oliveri[1], Matteo Dell'Amico[2], Davide Balzarotti[1]

[1]EURECOM [2]University of Genoa

@IridiumXOR

oliveri@eurecom.fr

EURECOM
Sophia Antipolis

UNIVERSITÀ
DEGLI STUDI
DI GENOVA

# Memory forensics 101: OS Profiles

- Non-trivial forensics tools use ***profiles***.

# Memory forensics 101: OS Profiles

- Non-trivial forensics tools use ***profiles***.
- A profile:
    - describes locations, field, and links of kernel data structures.

# Memory forensics 101: OS Profiles

- Non-trivial forensics tools use *profiles*.
- A profile:
  - describes locations, field, and links of kernel data structures.
  - is based on a deep knowledge of the OS internals.

# Memory forensics 101: OS Profiles

- Non-trivial forensics tools use ***profiles***.
- A profile:
  - describes locations, field, and links of kernel data structures.
  - is based on a deep knowledge of the OS internals.
- Different OS releases and kernel configurations require different profiles.

# The *“easy”* scenario…

An analyst acquires a memory dump of an *“ordinary”* system which:

# The *“easy”* scenario…

An analyst acquires a memory dump of an *“ordinary”* system which:

- is based on Intel or ARM CPUs

# The *“easy”* scenario…

An analyst acquires a memory dump of an *“ordinary”* system which:

- is based on Intel or ARM CPUs
- runs an OS that is supported by forensics tools

# The *“easy”* scenario…

An analyst acquires a memory dump of an *“ordinary”* system which:

- is based on Intel or ARM CPUs
- runs an OS that is supported by forensics tools
- requires a profile that is available to the analyst

# The *“easy”* scenario…

An analyst acquires a memory dump of an *“ordinary”* system which:

- is based on Intel or ARM CPUs
- runs an OS that is supported by forensics tools
- requires a profile that is available to the analyst

=> Existing tools can be used to extract artifacts

# …and the **hard** one.

An analyst acquires a memory dump of a *generic* device which:

- is based on specialized CPU architectures (industrial devices, printers…)

# …and the **hard** one.

An analyst acquires a memory dump of a *generic* device which:

- is based on specialized CPU architectures (industrial devices, printers…)
- uses a completely unknown/uncommon OS (network devices, IoT devices…)

# …and the **hard** one.

An analyst acquires a memory dump of a *generic* device which:

- is based on specialized CPU architectures (industrial devices, printers…)
- uses a completely unknown/uncommon OS (network devices, IoT devices…)
- cannot be rehosted or instrumented to take multiple snapshots.

# …and the **hard** one.

An analyst acquires a memory dump of a ***generic*** device which:

- is based on specialized CPU architectures (industrial devices, printers…)
- uses a completely unknown/uncommon OS (network devices, IoT devices…)
- cannot be rehosted or instrumented to take multiple snapshots.

=> No compatible forensics tools => No structured analysis possible.

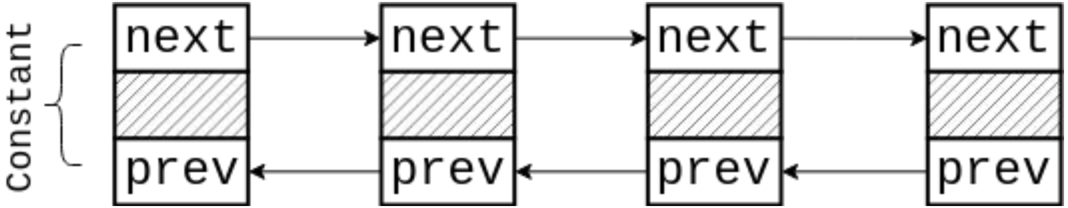
¯_(ツ)_/¯

# Memory forensics, differently

- *Profiles*-based approaches focus on OSs implementations

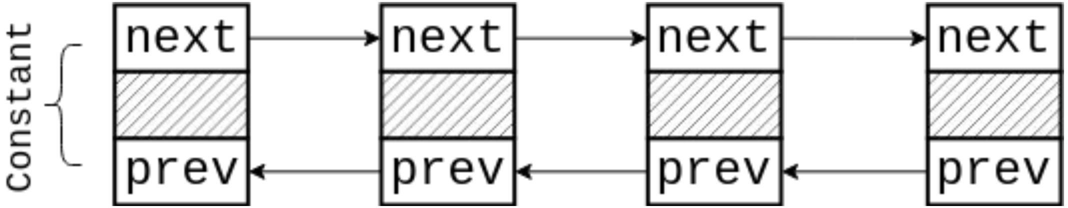
# Memory forensics, differently

- *Profiles*-based approaches focus on OSs implementations
- Data structures have universal topological constraints.



# Memory forensics, differently

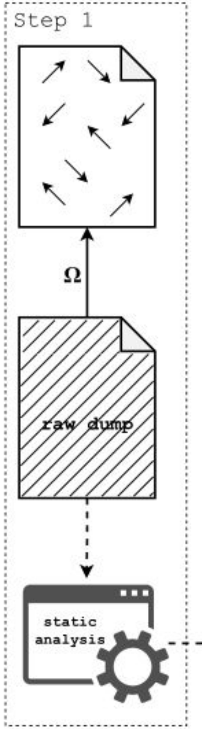
- *Profiles*-based approaches focus on OSs implementations
- Data structures have universal topological constraints.




=> CORE IDEA: Identify data structures without **any** knowledge of the OS using topological constraints.

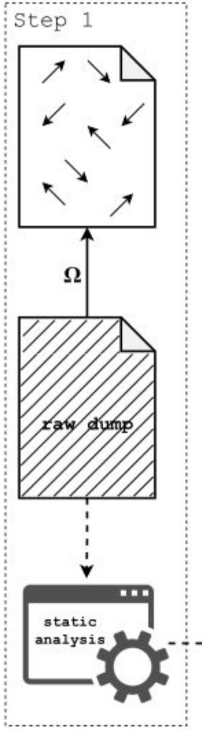
# Step 1: pointers and global variables

- Extract pointers in an OS-agnostic way.



# Step 1: pointers and global variables

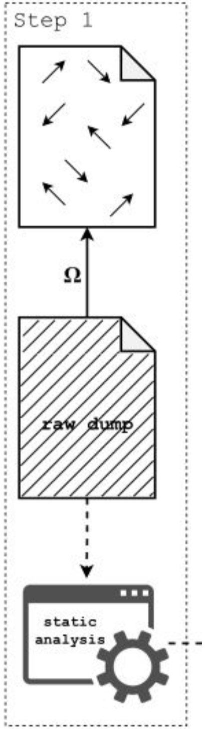
- Extract pointers in an OS-agnostic way.
  - Use MMU constraints*




*A.Oliveri et D.Balzarotti “In the Land of MMUs: Multiarchitecture OS-Agnostic Virtual Memory Forensics”

# Step 1: pointers and global variables

- Extract pointers in an OS-agnostic way.
  - Use MMU constraints*
  - Kernel address space as pointer validator.

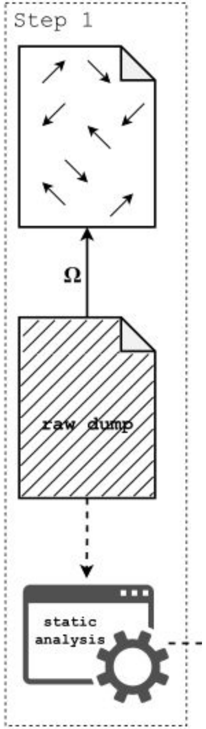



*A.Oliveri et D.Balzarotti “In the Land of MMUs: Multiarchitecture OS-Agnostic Virtual Memory Forensics”

# Step 1: pointers and global variables

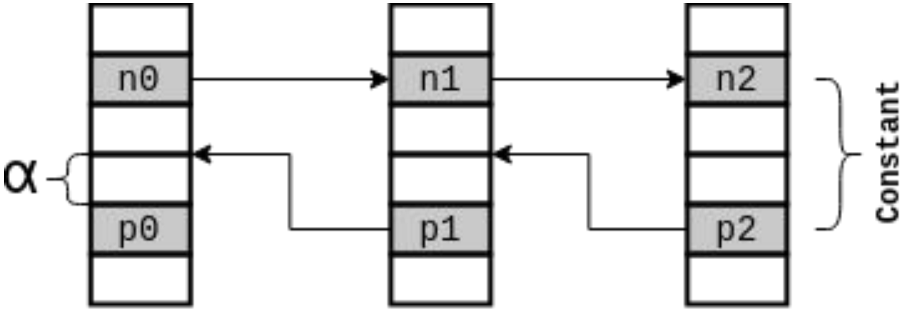
- Extract pointers in an OS-agnostic way.
  - Use MMU constraints*
  - Kernel address space as pointer validator.

- Static analysis tool to extract global variables.

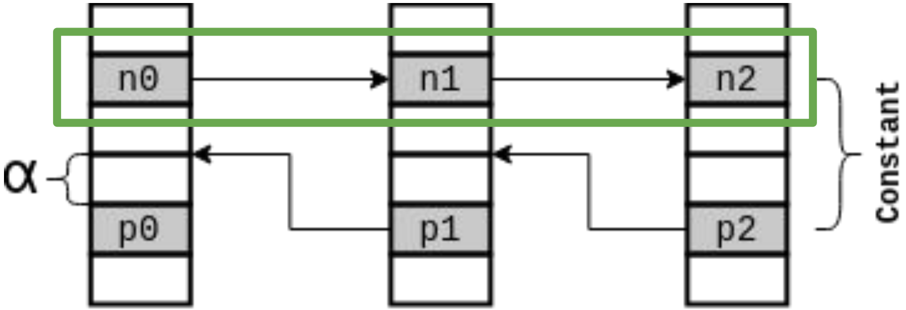



*A.Oliveri et D.Balzarotti "In the Land of MMUs: Multiarchitecture OS-Agnostic Virtual Memory Forensics"

# Step 2: pointers chains

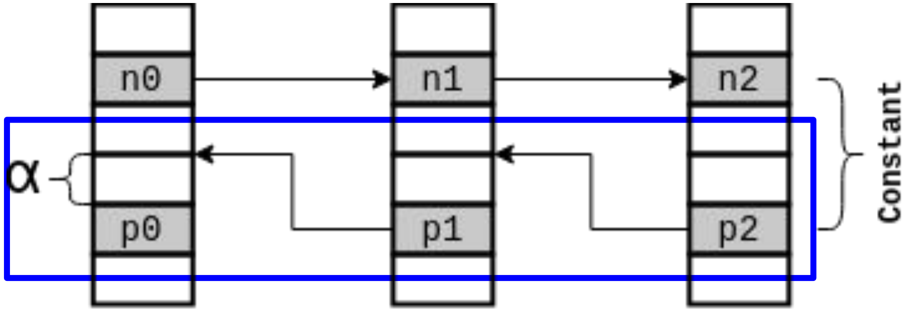


# Step 2: pointers chains



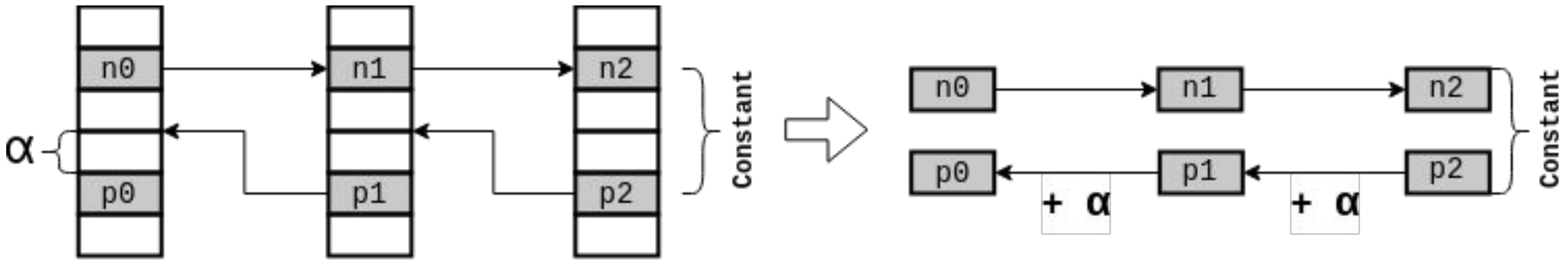

$$n_{i+1} = *n_i$$

# Step 2: pointers chains

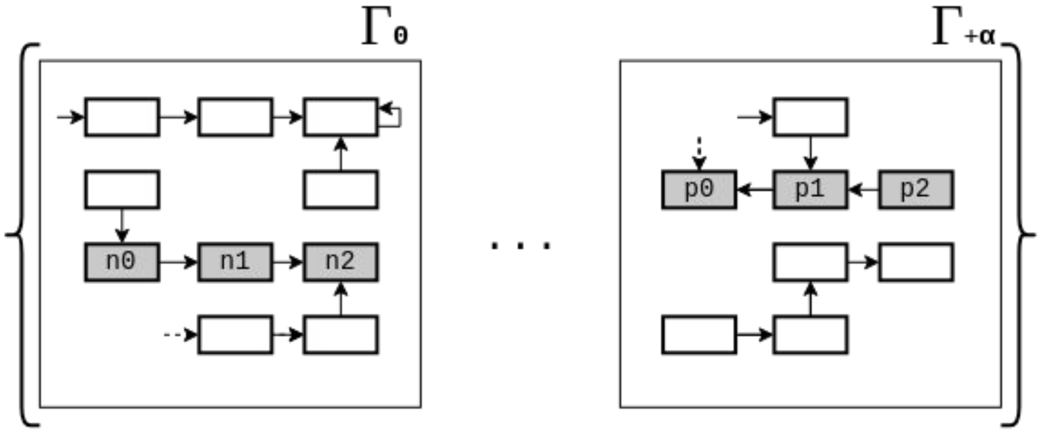



$$p_i = *p_{i+1} + \boldsymbol{\alpha}$$

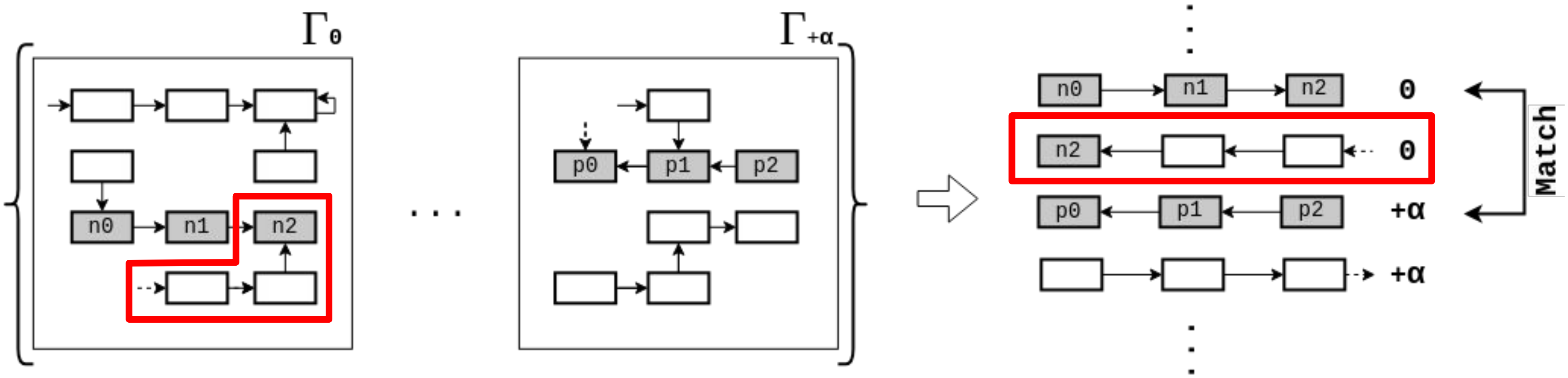
# Step 2: pointers chains



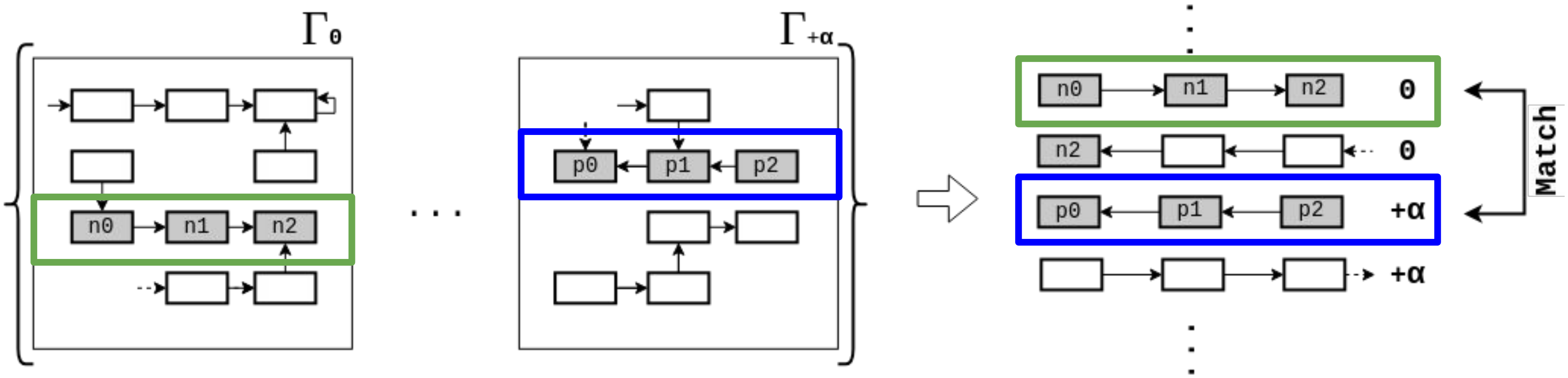
# Step 2: pointers chains



# Step 2: pointers chains

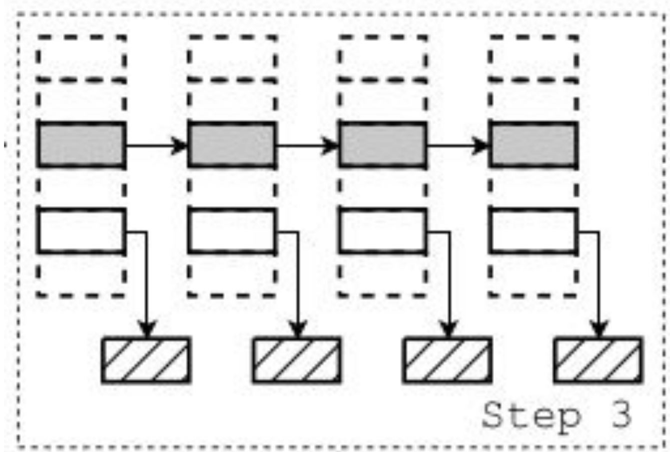


# Step 2: pointers chains



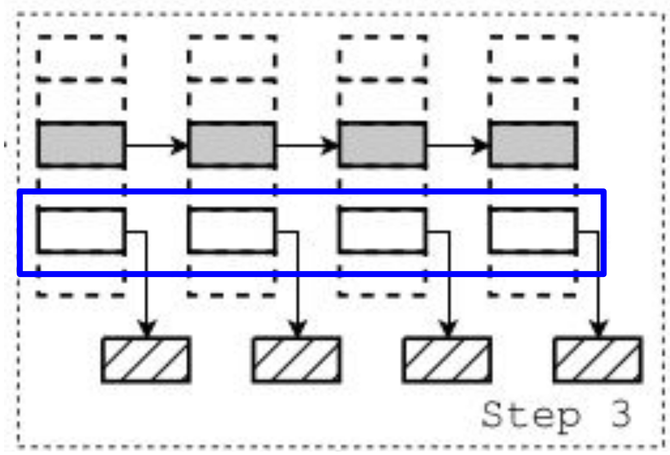
# Step 3: nodes size

- Skeleton pointers are at the <u>same offset in each struct node.</u>



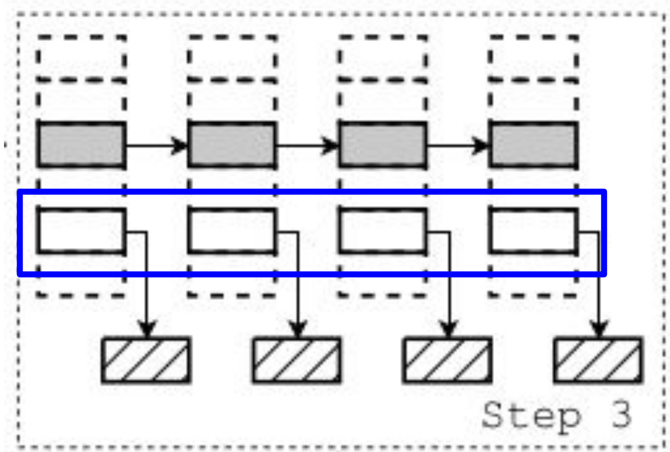
# Step 3: nodes size

- Skeleton pointers are at the <u>same offset in each struct node.</u>
    - Data located at the same offset must have the same type.



# Step 3: nodes size

- Skeleton pointers are at the <u>same offset in each struct node.</u>
    - Data located at the same offset must have the same type.
- => Extend nodes by including fields containing the same data type.

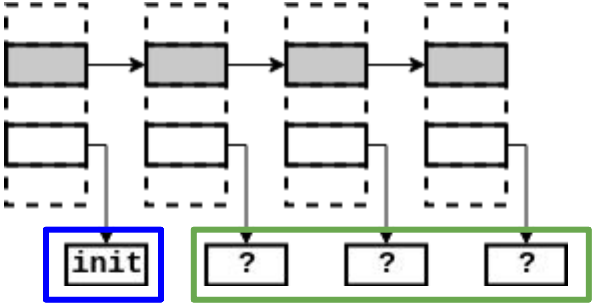


# Step 4: results filtering

- ***Seed***: a piece of forensics information known *a priori* or easily carved
    - a name of a process or a kernel module
    - an IP address
    - a file name etc.

# Step 4: results filtering

- ***Seed***: a piece of forensics information known *a priori* or easily carved
  - a name of a process or a kernel module
  - an IP address
  - a file name etc.
- **Used as anchors** to extract forensic relevant information



# Fossil

- Proof-of-concept in Python

# Fossil

- Proof-of-concept in Python
- It extracts:
  - linked lists
  - doubly-linked lists
  - binary trees
  - arrays of pointers to structures
  - dereference of linked nodes

# Fossil

- Proof-of-concept in Python
- It extracts:
  - linked lists
  - doubly-linked lists
  - binary trees
  - arrays of pointers to structures
  - dereference of linked nodes
- It uses strings as seeds:
  - immediately recognizable by an analyst
  - often present in fundamental forensics data structures.

# Experiments

- 14 different OSs on VMs with 4GB of RAM.

| OS | Kernel type[1] | Open-source | Main language | Pointers size ($\alpha$) |
|---|---|---|---|---|
| Darwin | H | ● | C | 8 |
| Embox | R | ● | C | 4 |
| FreeBSD | M | ● | C | 8 |
| HaikuOS | H | ● | C++ | 4 |
| HelenOS | m | ● | C | 8 |
| iOS (AArch64) | H | ● | C | 8 |
| Linux | M | ● | C | 8 |
| Linux (AArch64) | M | ● | C | 8 |
| NetBSD | M | ● | C | 4 |
| ReactOS | m | ● | C | 4 |
| ToaruOS | H | ● | C | 8 |
| vxWorks | R | ○ | C | 8 |
| Windows XP | H | ○ | C | 8 |
| Windows 10 | H | ○ | C | 8 |

H: hybrid, m: micro, M: monolithic, R: real-time

# Experiments

- 14 different OSs on VMs with 4GB of RAM.
- Different..
  - **..CPU architecture**

| OS | Kernel type[1] | Open-source | Main language | Pointers size ($\alpha$) |
|---|---|---|---|---|
| Darwin | H | ● | C | 8 |
| Embox | R | ● | C | 4 |
| FreeBSD | M | ● | C | 8 |
| HaikuOS | H | ● | C++ | 4 |
| HelenOS | m | ● | C | 8 |
| iOS (AArch64) | H | ● | C | 8 |
| Linux | M | ● | C | 8 |
| Linux (AArch64) | M | ● | C | 8 |
| NetBSD | M | ● | C | 4 |
| ReactOS | m | ● | C | 4 |
| ToaruOS | H | ● | C | 8 |
| vxWorks | R | ○ | C | 8 |
| Windows XP | H | ○ | C | 8 |
| Windows 10 | H | ○ | C | 8 |

H: hybrid, m: micro, M: monolithic, R: real-time

# Experiments

- 14 different OSs on VMs with 4GB of RAM.
- Different..
  - ..CPU architecture
  - **..OS type**

| OS | Kernel type[1] | Open-source | Main language | Pointers size ($\alpha$) |
|---|---|---|---|---|
| Darwin | H | ● | C | 8 |
| Embox | R | ● | C | 4 |
| FreeBSD | M | ● | C | 8 |
| HaikuOS | H | ● | C++ | 4 |
| HelenOS | m | ● | C | 8 |
| iOS (AArch64) | H | ● | C | 8 |
| Linux | M | ● | C | 8 |
| Linux (AArch64) | M | ● | C | 8 |
| NetBSD | M | ● | C | 4 |
| ReactOS | m | ● | C | 4 |
| ToaruOS | H | ● | C | 8 |
| vxWorks | R | ○ | C | 8 |
| Windows XP | H | ○ | C | 8 |
| Windows 10 | H | ○ | C | 8 |

H: hybrid, m: micro, M: monolithic, R: real-time

# Experiments

- <u>14 different OSs</u> on VMs with <u>4GB of RAM</u>.
- Different..
  - ..CPU architecture
  - **..OS type**

| OS | Kernel type[1] | Open-source | Main language | Pointers size ($\alpha$) |
|---|---|---|---|---|
| Darwin | H | ● | C | 8 |
| Embox | R | ● | C | 4 |
| FreeBSD | M | ● | C | 8 |
| HaikuOS | H | ● | C++ | 4 |
| HelenOS | m | ● | C | 8 |
| iOS (AArch64) | H | ● | C | 8 |
| Linux | M | ● | C | 8 |
| Linux (AArch64) | M | ● | C | 8 |
| NetBSD | M | ● | C | 4 |
| ReactOS | m | ● | C | 4 |
| ToaruOS | H | ● | C | 8 |
| vxWorks | R | ○ | C | 8 |
| Windows XP | H | ○ | C | 8 |
| Windows 10 | H | ○ | C | 8 |

H: hybrid, m: micro, M: monolithic, R: real-time

# Experiments

- 14 different OSs on VMs with 4GB of RAM.
- Different..
  - ..CPU architecture
  - **..OS type**

| OS | Kernel type[1] | Open-source | Main language | Pointers size ($\alpha$) |
|---|---|---|---|---|
| Darwin | H | ● | C | 8 |
| Embox | R | ● | C | 4 |
| FreeBSD | M | ● | C | 8 |
| HaikuOS | H | ● | C++ | 4 |
| HelenOS | m | ● | C | 8 |
| iOS (AArch64) | H | ● | C | 8 |
| Linux | M | ● | C | 8 |
| Linux (AArch64) | M | ● | C | 8 |
| NetBSD | M | ● | C | 4 |
| ReactOS | m | ● | C | 4 |
| ToaruOS | H | ● | C | 8 |
| vxWorks | R | ○ | C | 8 |
| Windows XP | H | ○ | C | 8 |
| Windows 10 | H | ○ | C | 8 |

H: hybrid, m: micro, M: monolithic, R: real-time

# Experiments

- 14 different OSs on VMs with 4GB of RAM.
- Different..
  - ..CPU architecture
  - **..OS type**

| OS | Kernel type[1] | Open-source | Main language | Pointers size ($\alpha$) |
|---|---|---|---|---|
| Darwin | H | ● | C | 8 |
| Embox | R | ● | C | 4 |
| FreeBSD | M | ● | C | 8 |
| HaikuOS | H | ● | C++ | 4 |
| HelenOS | m | ● | C | 8 |
| iOS (AArch64) | H | ● | C | 8 |
| Linux | M | ● | C | 8 |
| Linux (AArch64) | M | ● | C | 8 |
| NetBSD | M | ● | C | 4 |
| ReactOS | m | ● | C | 4 |
| ToaruOS | H | ● | C | 8 |
| vxWorks | R | ○ | C | 8 |
| Windows XP | H | ○ | C | 8 |
| Windows 10 | H | ○ | C | 8 |

H: hybrid, m: micro, M: monolithic, R: real-time

# Experiments

- 14 different OSs on VMs with 4GB of RAM.
- Different..
  - ..CPU architecture
  - ..OS type
  - **..kernel architectures**

| OS | Kernel type[1] | Open-source | Main language | Pointers size ($\alpha$) |
|---|---|---|---|---|
| Darwin | H | ● | C | 8 |
| Embox | R | ● | C | 4 |
| FreeBSD | M | ● | C | 8 |
| HaikuOS | H | ● | C++ | 4 |
| HelenOS | m | ● | C | 8 |
| iOS (AArch64) | H | ● | C | 8 |
| Linux | M | ● | C | 8 |
| Linux (AArch64) | M | ● | C | 8 |
| NetBSD | M | ● | C | 4 |
| ReactOS | m | ● | C | 4 |
| ToaruOS | H | ● | C | 8 |
| vxWorks | R | ○ | C | 8 |
| Windows XP | H | ○ | C | 8 |
| Windows 10 | H | ○ | C | 8 |

H: hybrid, m: micro, M: monolithic, R: real-time

# Experiments

- 14 different OSs on VMs with 4GB of RAM.
- Different..
  - ..CPU architecture
  - ..OS type
  - ..kernel architectures
  - **..programming language**

| OS | Kernel type[1] | Open-source | Main language | Pointers size ($\alpha$) |
|---|---|---|---|---|
| Darwin | H | ● | C | 8 |
| Embox | R | ● | C | 4 |
| FreeBSD | M | ● | C | 8 |
| HaikuOS | H | ● | $\boxed{\text{C++}}$ | 4 |
| HelenOS | m | ● | C | 8 |
| iOS (AArch64) | H | ● | C | 8 |
| Linux | M | ● | C | 8 |
| Linux (AArch64) | M | ● | C | 8 |
| NetBSD | M | ● | C | 4 |
| ReactOS | m | ● | C | 4 |
| ToaruOS | H | ● | C | 8 |
| vxWorks | R | ○ | C | 8 |
| Windows XP | H | ○ | C | 8 |
| Windows 10 | H | ○ | C | 8 |

H: hybrid, m: micro, M: monolithic, R: real-time

# Experiments

- 14 different OSs on VMs with 4GB of RAM.
- Different..
  - ..CPU architecture
  - ..OS type
  - ..kernel architectures
  - ..programming language
  - **..license**

| OS | Kernel type[1] | Open-source | Main language | Pointers size ($\alpha$) |
|---|---|---|---|---|
| Darwin | H | ● | C | 8 |
| Embox | R | ● | C | 4 |
| FreeBSD | M | ● | C | 8 |
| HaikuOS | H | ● | C++ | 4 |
| HelenOS | m | ● | C | 8 |
| iOS (AArch64) | H | ● | C | 8 |
| Linux | M | ● | C | 8 |
| Linux (AArch64) | M | ● | C | 8 |
| NetBSD | M | ● | C | 4 |
| ReactOS | m | ● | C | 4 |
| ToaruOS | H | ● | C | 8 |
| vxWorks | R | ○ | C | 8 |
| Windows XP | H | ○ | C | 8 |
| Windows 10 | H | ○ | C | 8 |

H: hybrid, m: micro, M: monolithic, R: real-time

# Experiments

- 14 different OSs on VMs with 4GB of RAM.
- Different..
  - ..CPU architecture
  - ..OS type
  - ..kernel architectures
  - ..programming language
  - ..license
- Static analysis tool: Ghidra

| OS | Kernel type[1] | Open-source | Main language | Pointers size ($\alpha$) |
|---|---|---|---|---|
| Darwin | H | ● | C | 8 |
| Embox | R | ● | C | 4 |
| FreeBSD | M | ● | C | 8 |
| HaikuOS | H | ● | C++ | 4 |
| HelenOS | m | ● | C | 8 |
| iOS (AArch64) | H | ● | C | 8 |
| Linux | M | ● | C | 8 |
| Linux (AArch64) | M | ● | C | 8 |
| NetBSD | M | ● | C | 4 |
| ReactOS | m | ● | C | 4 |
| ToaruOS | H | ● | C | 8 |
| vxWorks | R | ○ | C | 8 |
| Windows XP | H | ○ | C | 8 |
| Windows 10 | H | ○ | C | 8 |

H: hybrid, m: micro, M: monolithic, R: real-time

# Experiment 1: process enumeration

**Goal**: Extract the names of all processes, starting from two known ones

| OS | Process List<br>Two seeds<br>all/referenced |
|---|---|
| Darwin | 6/6 |
| Embox | 5/4 |
| FreeBSD | 4/4 |
| HaikuOS | 3/3 |
| HelenOS | 2/- |
| iOS | 3/3 |
| Linux | 3/1 |
| Linux (AArch64) | 1/1 |
| NetBSD | 2/2 |
| ReactOS | 5/4 |
| ToaruOS | 1/1 |
| vxWorks | 2/- |
| Windows XP | 3/3 |
| Windows 10 | 4/- |

[1] AS: array of pointers to structs, A1: first level auxiliary structure, CDL: circular doubly-linked list, LDL: linear doubly-linked list, L: linked list

# Experiment 1: process enumeration

**Goal**: Extract the names of all processes, starting from two known ones

| OS | Process List | |
|---|---|---|
| | Structure type[1] | Two seeds all/referenced |
| Darwin | LDL | 6/6 |
| Embox | AS | 5/4 |
| FreeBSD | L | 4/4 |
| HaikuOS | LDL | 3/3 |
| HelenOS | CDL | 2/- |
| iOS | LDL | 3/3 |
| Linux | CDL | 3/1 |
| Linux (AArch64) | CDL | 1/1 |
| NetBSD | LDL | 2/2 |
| ReactOS | CDL | 5/4 |
| ToaruOS | A1 | 1/1 |
| vxWorks | LDL | 2/- |
| Windows XP | CDL | 3/3 |
| Windows 10 | CDL | 4/- |

[1] AS: array of pointers to structs, A1: first level auxiliary structure, CDL: circular doubly-linked list, LDL: linear doubly-linked list, L: linked list

# Experiment 1: process enumeration

**Goal**: Extract the names of all processes, starting from two known ones

| OS | Process List | | |
|---|---|---|---|
| | Structure type[1] | Two seeds all/referenced | One seed all/referenced |
| Darwin | LDL | 6/6 | 9/9 |
| Embox | AS | 5/4 | 8/5 |
| FreeBSD | L | 4/4 | 34/23 |
| HaikuOS | LDL | 3/3 | 114/114 |
| HelenOS | CDL | 2/- | 5/- |
| iOS | LDL | 3/3 | 9/7 |
| Linux | CDL | 3/1 | 76/3 |
| Linux (AArch64) | CDL | 1/1 | 300/2 |
| NetBSD | LDL | 2/2 | 2/2 |
| ReactOS | CDL | 5/4 | 5/4 |
| ToaruOS | A1 | 1/1 | 47/47 |
| vxWorks | LDL | 2/- | 14/- |
| Windows XP | CDL | 3/3 | 5/3 |
| Windows 10 | CDL | 4/- | 6/- |

[1] AS: array of pointers to structs, A1: first level auxiliary structure, CDL: circular doubly-linked list, LDL: linear doubly-linked list, L: linked list

# Experiment 1: process enumeration

**Goal:** Extract the names of all processes, starting from two known ones

| OS | Process List | | |
|---|---|---|---|
| | Structure type[1] | Two seeds all/referenced | One seed all/referenced |
| Darwin | LDL | 6/6 | 9/9 |
| Embox | AS | 5/4 | 8/5 |
| FreeBSD | L | 4/4 | 34/23 |
| HaikuOS | LDL | 3/3 | 114/114 |
| HelenOS | CDL | 2/- | 5/- |
| iOS | LDL | 3/3 | 9/7 |
| Linux | CDL | 3/1 | 76/3 |
| Linux (AArch64) | CDL | 1/1 | 300/2 |
| NetBSD | LDL | 2/2 | 2/2 |
| ReactOS | CDL | 5/4 | 5/4 |
| ToaruOS | A1 | 1/1 | 47/47 |
| vxWorks | LDL | 2/- | 14/- |
| Windows XP | CDL | 3/3 | 5/3 |
| Windows 10 | CDL | 4/- | 6/- |

[1] AS: array of pointers to structs, A1: first level auxiliary structure, CDL: circular doubly-linked list, LDL: linear doubly-linked list, L: linked list

# Experiment 2: modules, pools and fs enumeration

**Goal**: Extract other data structures, starting from two known seeds

| OS | Kernel modules | Kernel pools | File systems |
|---|---|---|---|
| Darwin | ● | ● | ● |
| Embox | ● | -[1] | ○ |
| FreeBSD | ● | -[1] | ● |
| HaikuOS | ● | ● | ○ |
| HelenOS | ●[†] | ● | ●[†] |
| iOS | ○ | ● | ● |
| Linux | ●[†] | ● | ● |
| Linux (AArch64) | ●[†] | ● | ● |
| NetBSD | ● | ●[†] | ● |
| ReactOS | ○ | ● | ○ |
| ToaruOS | ● | -[1] | ● |
| vxWorks | ○ | ● | ○ |
| Windows XP | ● | ● | ○ |
| Windows 10 | ● | ● | ● |

# Experiment 2: modules, pools and fs enumeration

**Goal**: Extract other data structures, starting from two known seeds

| OS | Kernel modules | Kernel pools | File systems |
|---|---|---|---|
| Darwin | ● | ● | ● |
| Embox | ● | -[1] | ○ |
| FreeBSD | ● | -[1] | ● |
| HaikuOS | ● | ● | ○ |
| HelenOS | ●† | ● | ●† |
| iOS | ○ | ● | ● |
| Linux | ●† | ● | ● |
| Linux (AArch64) | ●† | ● | ● |
| NetBSD | ● | ●† | ● |
| ReactOS | ○ | ● | ○ |
| ToaruOS | ● | -[1] | ● |
| vxWorks | ○ | ● | ○ |
| Windows XP | ● | ● | ○ |
| Windows 10 | ● | ● | ● |

# Experiment 2: modules, pools and fs enumeration

**Goal**: Extract other data structures, starting from two known seeds

| OS | Kernel modules | Kernel pools | File systems | Other structures |
|---|---|---|---|---|
| Darwin | ● | ● | ● | • List of network devices • System locks • Kernel/user pipes • Kernel parameters |
| Embox | ● | -[1] | ○ | • List of commands |
| FreeBSD | ● | -[1] | ● | |
| HaikuOS | ● | ● | ○ | • Executable libraries • Kernel/user pipes • Semaphores |
| HelenOS | ●[†] | ● | ●[†] | |
| iOS | ○ | ● | ● | • List of network devices • System locks • Kernel/user pipes • Kernel parameters |
| Linux | ●[†] | ● | ● | • Files in `sysfs` • Network protocols |
| Linux (AArch64) | ●[†] | ● | ● | • Files in `sysfs` • Network protocols |
| NetBSD | ● | ●[†] | ● | • Kernel tasks |
| ReactOS | ○ | ● | ○ | |
| ToaruOS | ● | -[1] | ● | • Devices' list • Processes' environment |
| vxWorks | ○ | ● | ○ | • Devices' list • Open sockets |
| Windows XP | ● | ● | ○ | |
| Windows 10 | ● | ● | ● | |

# Experiment 3: blackbox scenario

**Goal**: Recover data structures without seeds

| OS | Process list | Kernel modules | Kernel pools | Filesystem |
|---|---|---|---|---|
| Darwin | 2 | 10 | 11 | 7 |
| Embox | 17 | ○ | - | - |
| FreeBSD | 24 | 31 | - | 26 |
| HaikuOS | 6 | 1 | 11 | - |
| HelenOS | 4 | 2 | 1 | 1 |
| iOS | 2 | - | 2 | 15 |
| Linux | 5 | 28 | 26 | 15 |
| Linux (AArch64) | 4 | 22 | 19 | 24 |
| NetBSD | 2 | 6 | 18 | ○ |
| ReactOS | 5 | - | 12 | - |
| ToaruOS | 3 | 2 | 3 | - |
| vxWorks | 4 | - | 2 | - |
| Windows XP | 5 | 1 | 2 | - |
| Windows 10 | 41 | ○ | ○ | ○ |

# Experiment 3: blackbox scenario

**Goal**: Recover data structures without seeds

=> **Highlights:**

- 50% in TOP5

| OS | Process list | Kernel modules | Kernel pools | Filesystem |
|---|---|---|---|---|
| Darwin | 2 | 10 | 11 | 7 |
| Embox | 17 | ○ | - | - |
| FreeBSD | 24 | 31 | - | 26 |
| HaikuOS | 6 | 1 | 11 | - |
| HelenOS | 4 | 2 | 1 | 1 |
| iOS | 2 | - | 2 | 15 |
| Linux | 5 | 28 | 26 | 15 |
| Linux (AArch64) | 4 | 22 | 19 | 24 |
| NetBSD | 2 | 6 | 18 | ○ |
| ReactOS | 5 | - | 12 | - |
| ToaruOS | 3 | 2 | 3 | - |
| vxWorks | 4 | - | 2 | - |
| Windows XP | 5 | 1 | 2 | - |
| Windows 10 | 41 | ○ | ○ | ○ |

# Experiment 3: blackbox scenario

**Goal**: Recover data structures without seeds

=> **Highlights:**

- 50% in TOP5
- ~80% in TOP20

| OS | Process list | Kernel modules | Kernel pools | Filesystem |
|---|---|---|---|---|
| Darwin | 2 | 10 | 11 | 7 |
| Embox | 17 | ○ | - | - |
| FreeBSD | 24 | 31 | - | 26 |
| HaikuOS | 6 | 1 | 11 | - |
| HelenOS | 4 | 2 | 1 | 1 |
| iOS | 2 | - | 2 | 15 |
| Linux | 5 | 28 | 26 | 15 |
| Linux (AArch64) | 4 | 22 | 19 | 24 |
| NetBSD | 2 | 6 | 18 | ○ |
| ReactOS | 5 | - | 12 | - |
| ToaruOS | 3 | 2 | 3 | - |
| vxWorks | 4 | - | 2 | - |
| Windows XP | 5 | 1 | 2 | - |
| Windows 10 | 41 | ○ | ○ | ○ |

# Profile- vs. topological- based forensics tools

## Profile based tools

✅ Fine tuned analysis.

## Topological based tools

✅ Blackbox approach.

# Profile- vs. topological- based forensics tools

## Profile based tools

✅ Fine tuned analysis.

❌ Require to support the OS.

## Topological based tools

✅ Blackbox approach.

✅ Independent from the OS.

# Profile- vs. topological- based forensics tools

## Profile based tools

✅ Fine tuned analysis.

❌ Require to support the OS.

✅ High specificity: extract data structures containing no seed.

## Topological based tools

✅ Blackbox approach.

✅ Independent from the OS.

❌ High flexibility: cannot locate complex/custom structures.

Questions?

# From a raw dump to kernel data structures

1. Extract pointers and global variables.
2. Reconstruct structures uses topological constraints
3. Estimate structs size.
4. Results filtering.