black hat USA 2024

AUGUST 7-8, 2024

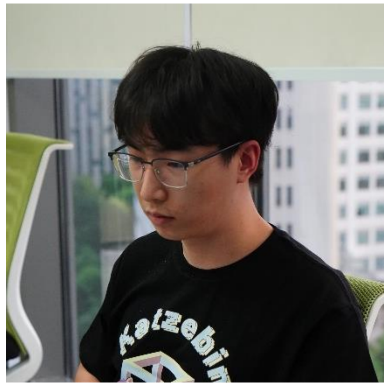
BRIEFINGS

Stop! Sandboxing Exploitable Functions and Modules Using In-Kernel Machine Learning

Presenter: Qinrun Dai

Contributors: Zicheng Wang, Tiejin Chen, Yueqi Chen, and Hua Wei

About us



Qinrun Dai

PhD Student
University of Colorado, Boulder

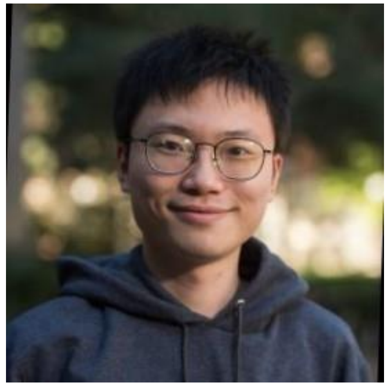
University of Colorado Boulder

南京大學
NANJING UNIVERSITY

ASU Arizona State University

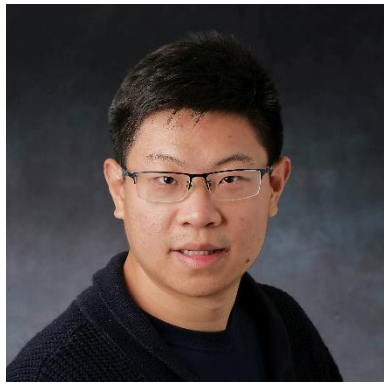
Zicheng Wang

PhD
Nanjing University



Tiejin Chen

PhD Student
Arizona State University



Yeuqi Chen

Assistant Professor
University of Colorado, Boulder

Hua Wei

Assistant Professor
Arizona State University

Agenda

- **Motivation**
  - Risky Time Window in Kernel Development
  - Existing Solutions and Limitations
  - Challenges of On-the-Fly Solution
- Challenges & Design Overview
- Example Workflow by CVE-2022-0995 & Video Demo
- Technical Details
- Evaluation

Tool is available at: https://github.com/a8stract-lab/o2c

Paper is available at: https://arxiv.org/abs/2401.05641

Risky Time Window in Kernel Development

Risky Time Window in Kernel Development

Linux Kernel Development Timeline

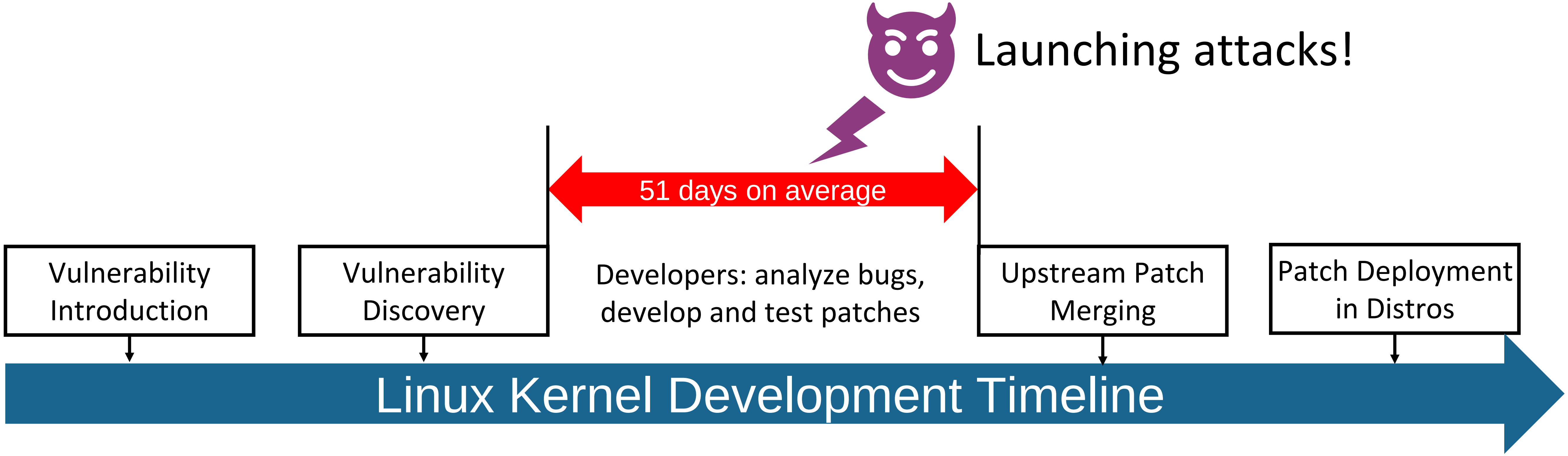
Risky Time Window in Kernel Development

Risky Time Window in Kernel Development

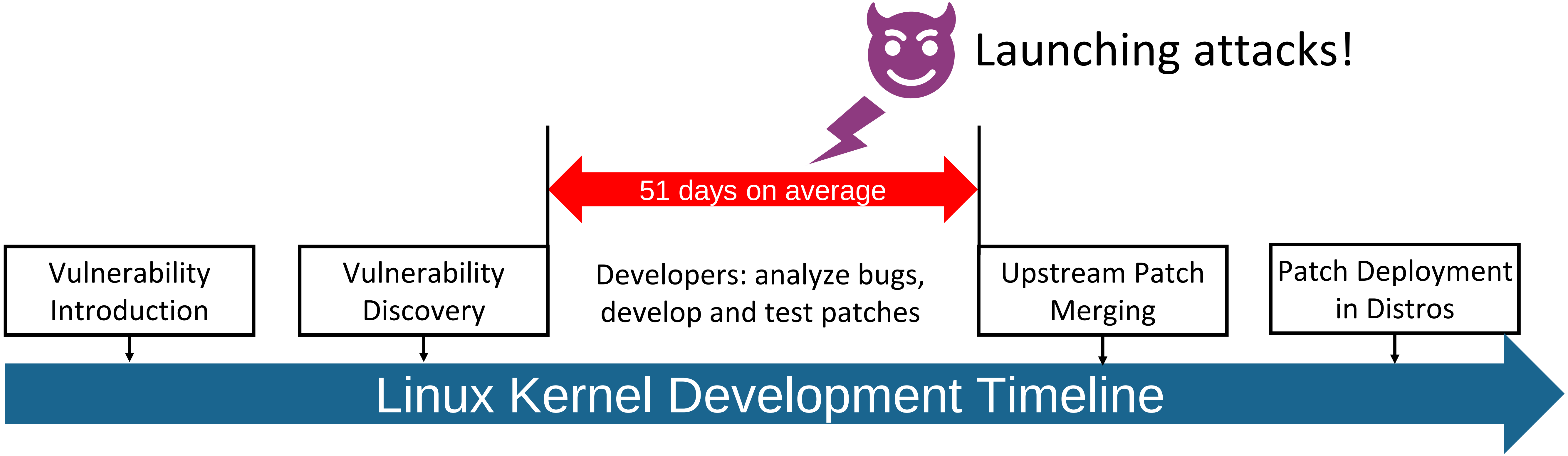
Risky Time Window in Kernel Development

Risky Time Window in Kernel Development

Risky Time Window in Kernel Development

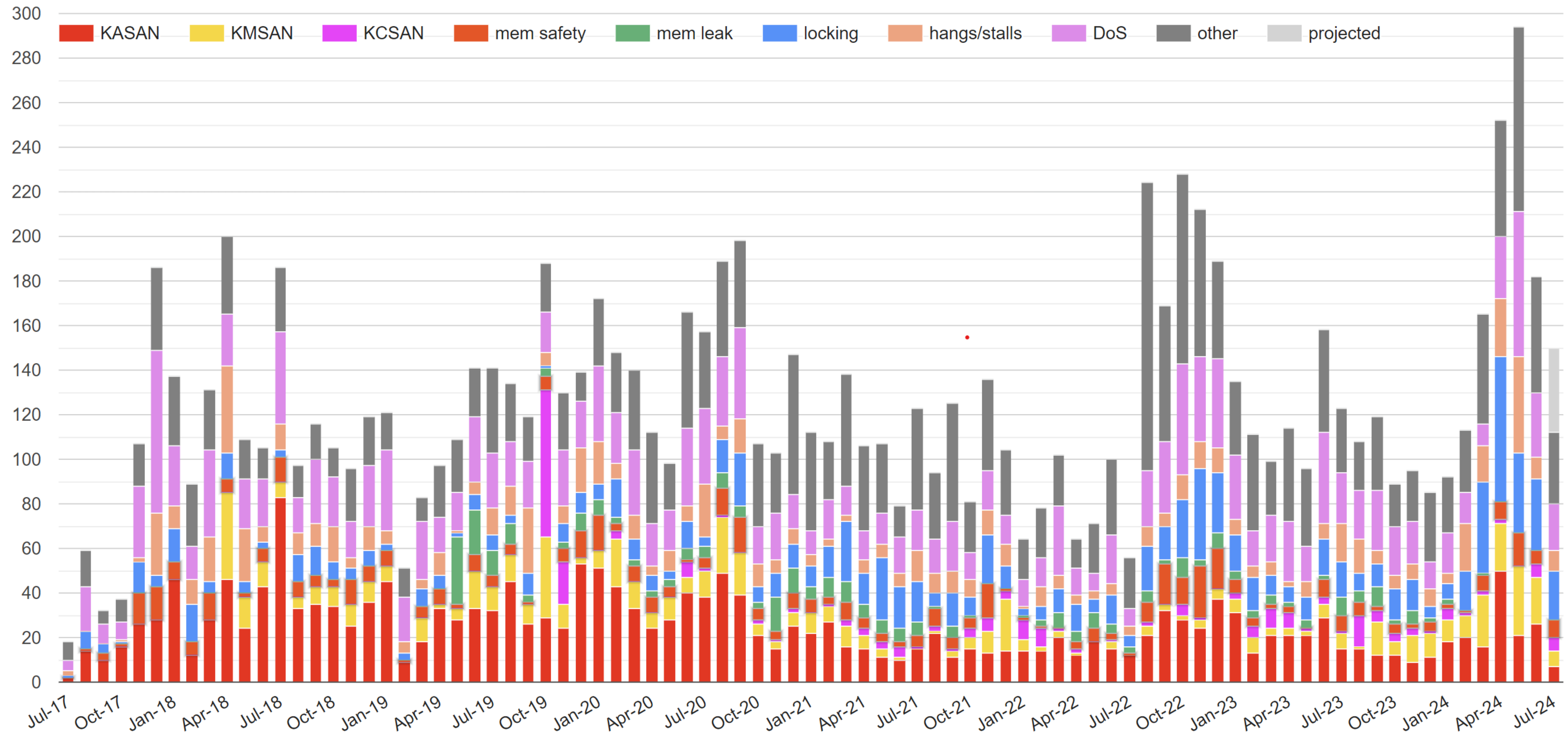


Risky Time Window in Kernel Development



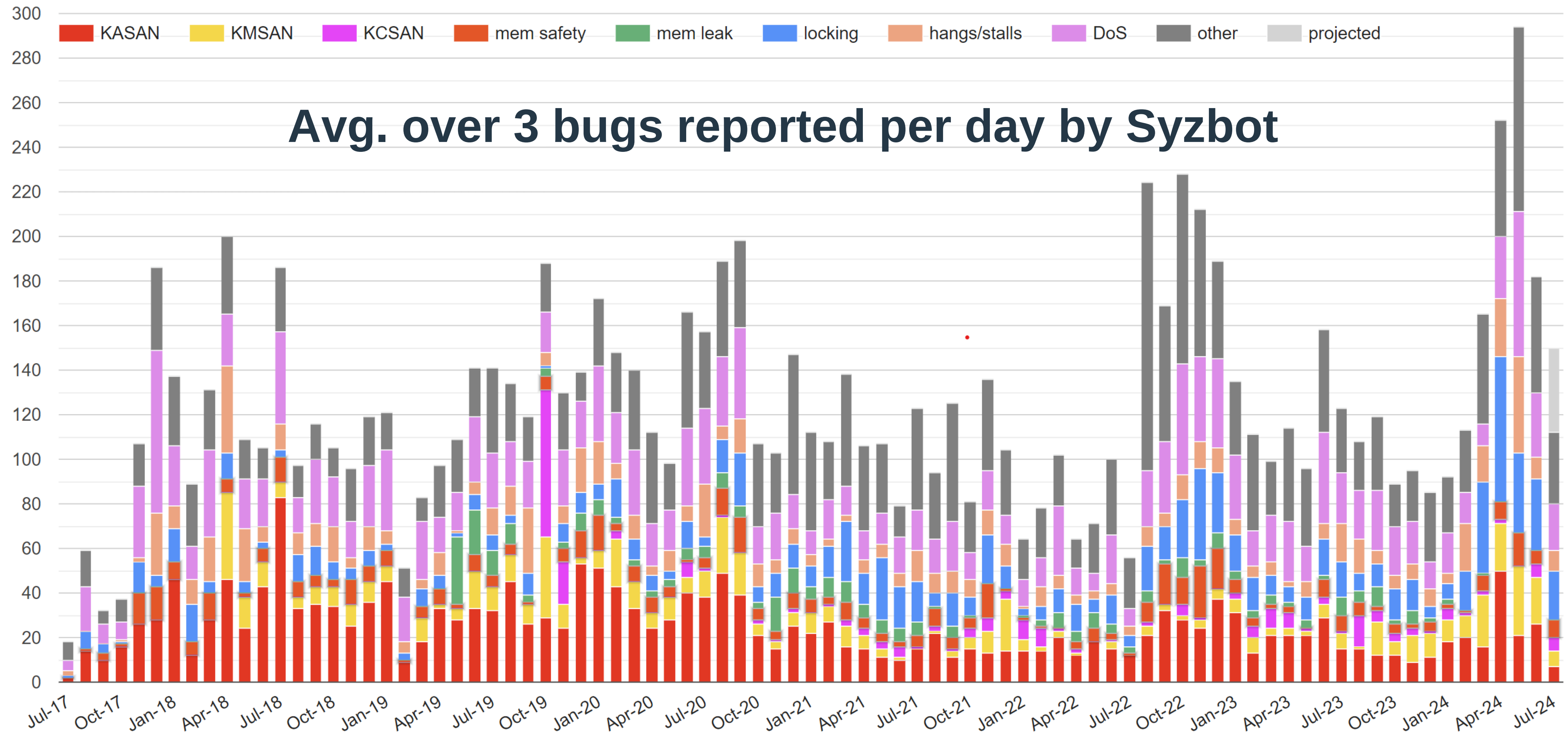
How to remediate newly discovered vulnerabilities before official patches are available?

Disruptive Solution is Unacceptable



# of bug reports by Syzbot per month

Disruptive Solution is Unacceptable



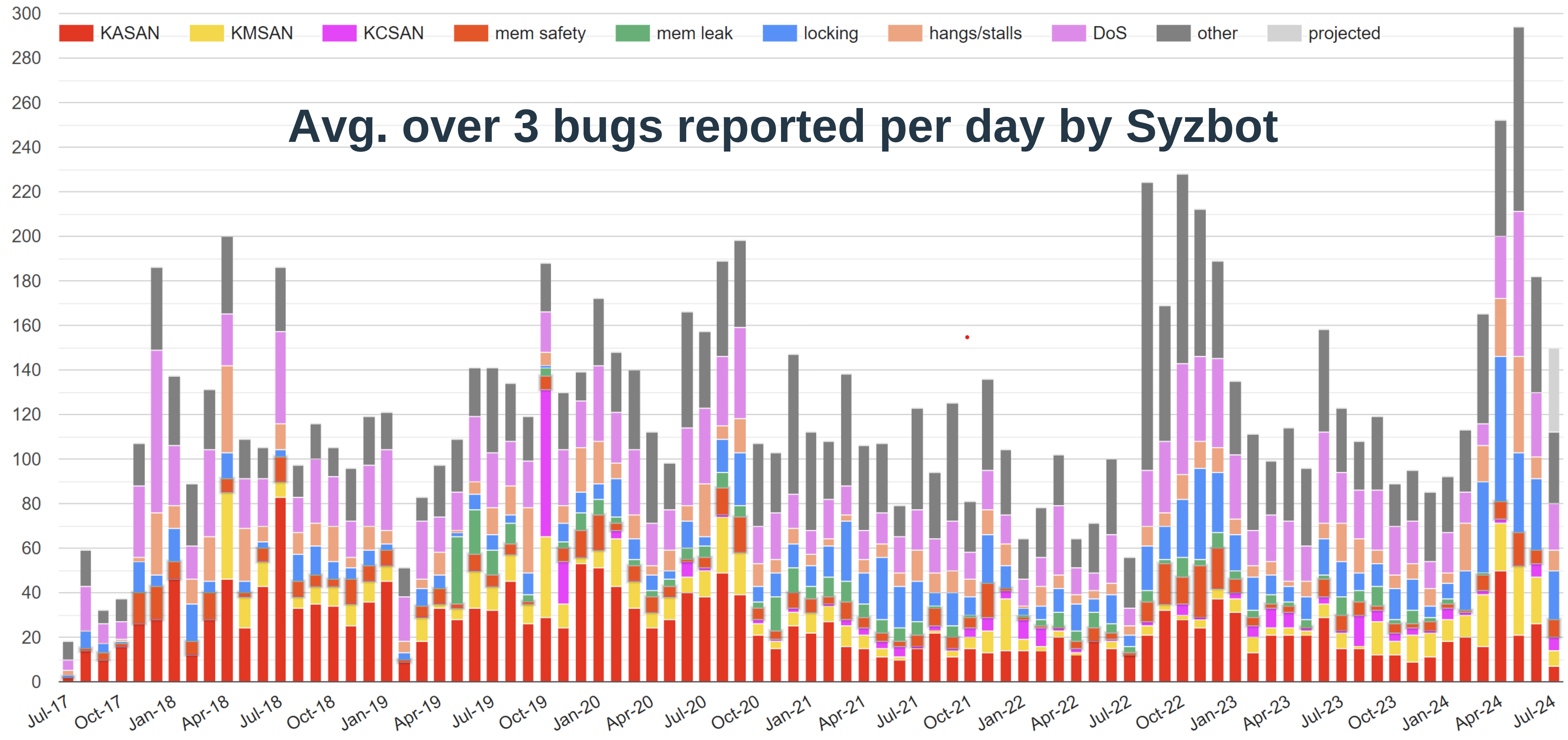
# of bug reports by Syzbot per month

Disruptive Solution is Unacceptable

Takeaway:

A disruptive solution that requires rebooting and disrupting running service is unacceptable.

Otherwise over 3 times of rebooting is needed to have a full coverage.



# of bug reports by Syzbot per month

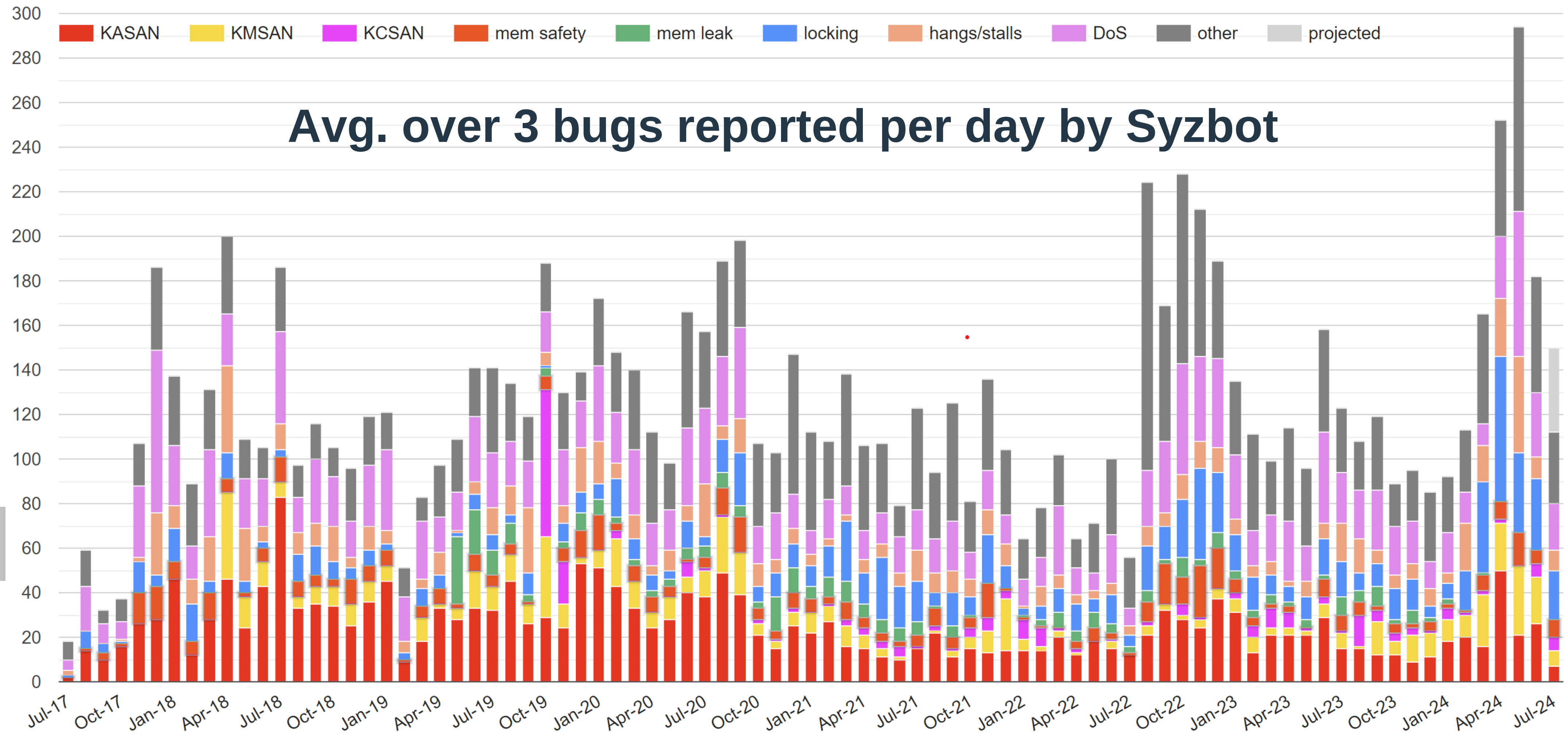
Disruptive Solution is Unacceptable

Takeaway:

A disruptive solution that requires rebooting and disrupting running service is unacceptable.

Otherwise over 3 times of rebooting is needed to have a full coverage.

An On-the-Fly solution is desired



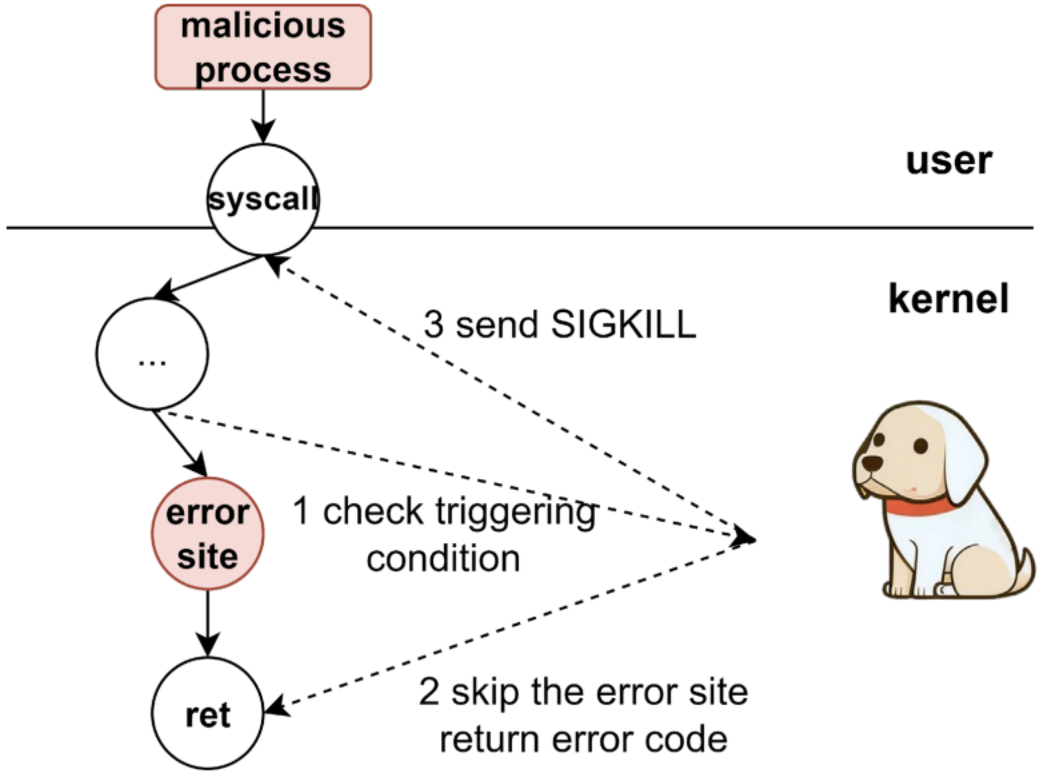
# of bug reports by Syzbot per month

Existing Solutions and Limitations

[1] PET: Prevent Discovered Errors from Being Triggered in the Linux Kernel, USENIX Security'23

Existing Solutions and Limitations

- PET[1]

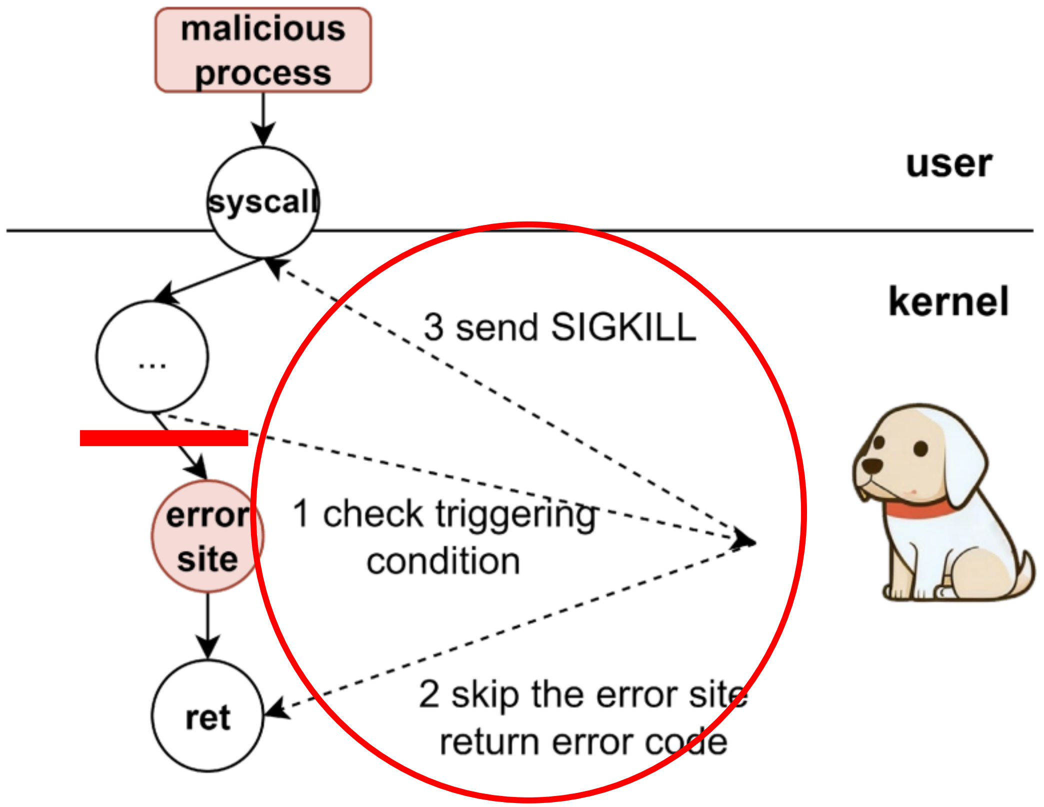


[1] PET: Prevent Discovered Errors from Being Triggered in the Linux Kernel, USENIX Security'23

USA 2024

Existing Solutions and Limitations

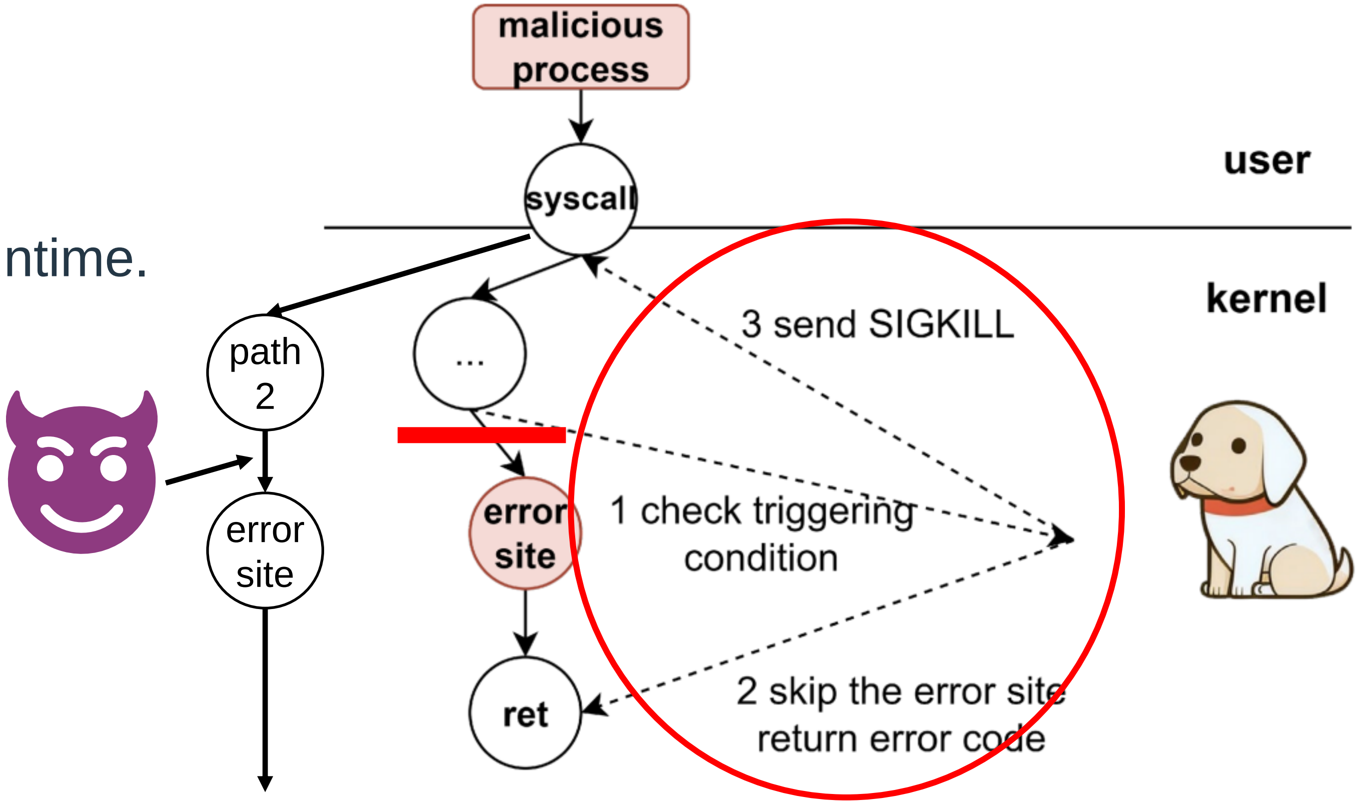
- PET[1]
- Core idea:
  - Construct triggering conditions.
  - Determine if triggering condition is met at runtime.
  - Prevent triggering if yes.



[1] PET: Prevent Discovered Errors from Being Triggered in the Linux Kernel, USENIX Security'23

Existing Solutions and Limitations

- PET[1]
- Core idea:
  - Construct triggering conditions.
  - Determine if triggering condition is met at runtime.
  - Prevent triggering if yes.
- Limitation:
  - Can be bypassed if exploits target another triggering site along a different path.



[1] PET: Prevent Discovered Errors from Being Triggered in the Linux Kernel, USENIX Security'23

Existing Solutions and Limitations (cont.)

[2] SeaK: Rethinking the Design of a Secure Allocator for OS Kernel, USENIX Security'24

Existing Solutions and Limitations (cont.)

- SeaK[2]

[2] SeaK: Rethinking the Design of a Secure Allocator for OS Kernel, USENIX Security'24

Existing Solutions and Limitations (cont.)

- SeaK[2]

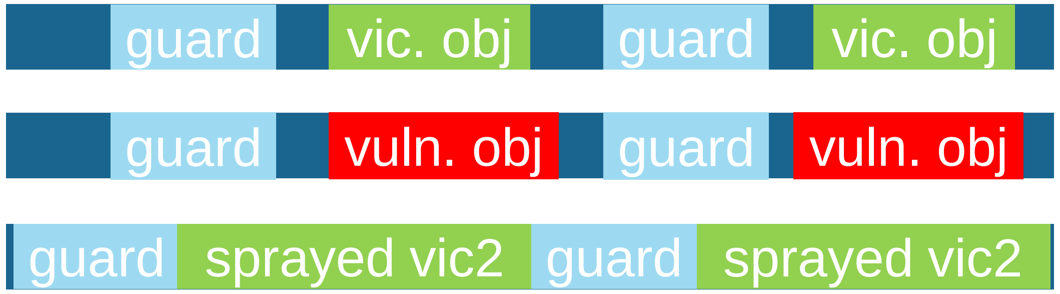
Typical memory layout of heap out-of-bound exploitation

[2] SeaK: Rethinking the Design of a Secure Allocator for OS Kernel, USENIX Security’24

Existing Solutions and Limitations (cont.)

- SeaK[2]
- Core idea:
  - Isolates vulnerable objects, victim objects, and spray objects in different regions.

Typical memory layout of heap out-of-bound exploitation



Memory layout after isolation

[2] SeaK: Rethinking the Design of a Secure Allocator for OS Kernel, USENIX Security'24

Existing Solutions and Limitations (cont.)

- SeaK[2]
- Core idea:
  - Isolates vulnerable objects, victim objects, and spray objects in different regions.
- Limitation:
  - While more general than PET, SeaK[2] can be bypassed if attackers exploit legacy objects.

Typical memory layout of heap out-of-bound exploitation

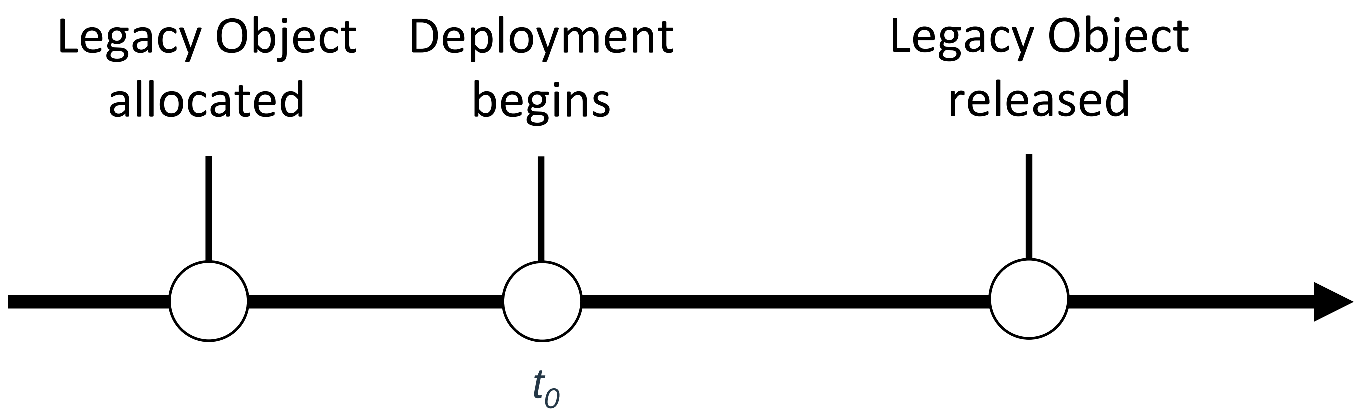
Memory layout after isolation

[2] SeaK: Rethinking the Design of a Secure Allocator for OS Kernel, USENIX Security'24

Legacy Objects Problem in Detail

Legacy Objects Problem in Detail

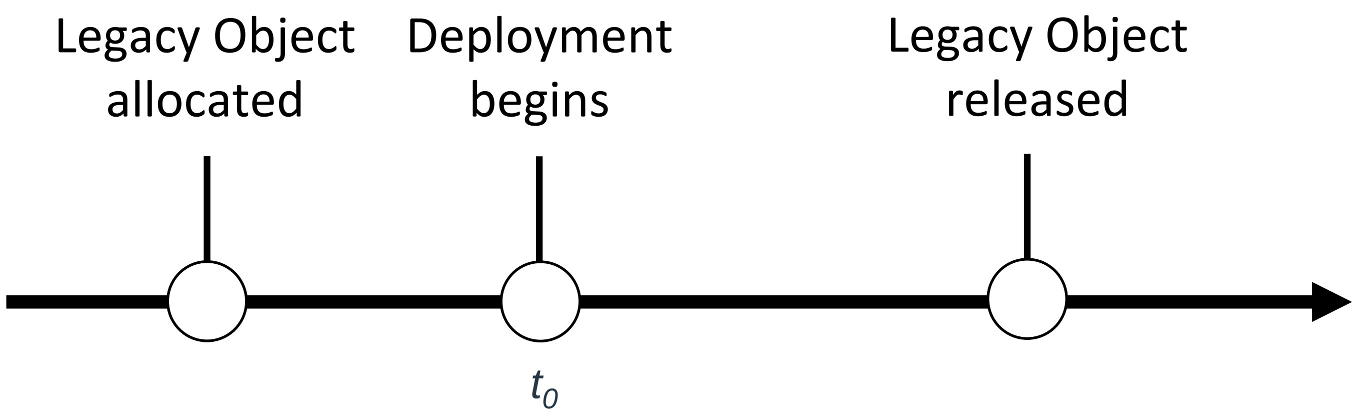
- Definition: objects allocated before protection is deployed (t_0) and released after t_0.



Legacy Objects Problem in Detail

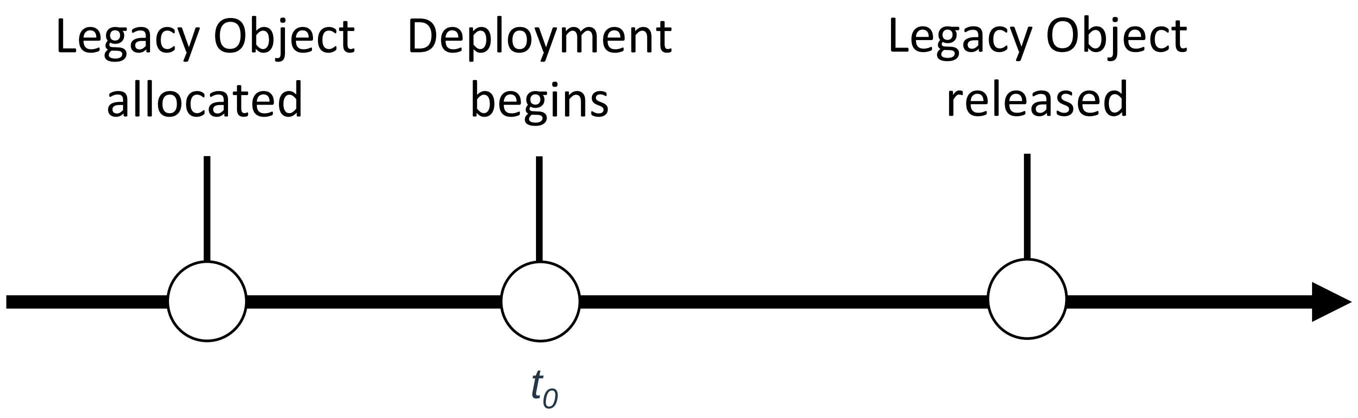
- Definition: objects allocated before protection is deployed (t_0) and released after t_0.

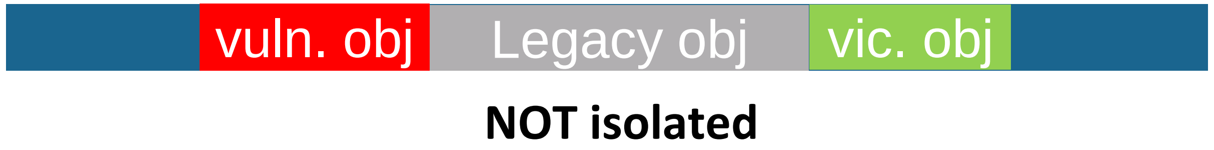
- Our statistics:
  - Lifetime of legacy objects is long: 10,862 objects last more than 10s.
  - Many chances to manipulate legacy objects: average 22.87 modifications during the object's lifetime.



Legacy Objects Problem in Detail

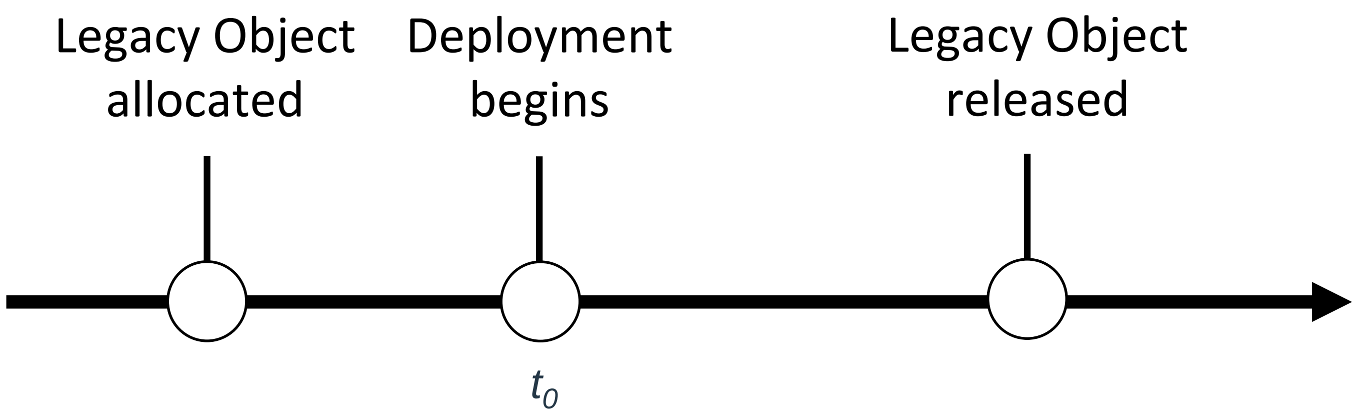
- Definition: objects allocated before protection is deployed (t_0) and released after t_0.
- Our statistics:
  - Lifetime of legacy objects is long: 10,862 objects last more than 10s.
  - Many chances to manipulate legacy objects: average 22.87 modifications during the object’s lifetime.
- What if a vulnerable / victim object is legacy?
  - Not isolated and mixed up with other objects.

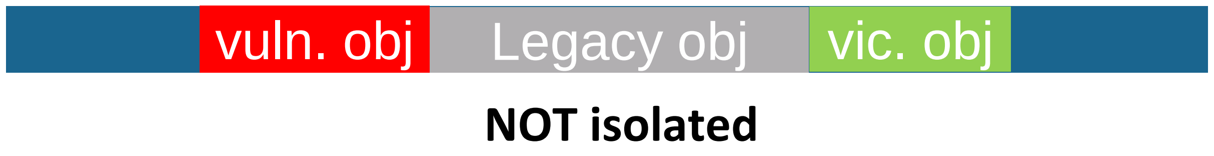




Legacy Objects Problem in Detail

- Definition: objects allocated before protection is deployed (t_0) and released after t_0.
- Our statistics:
  - Lifetime of legacy objects is long: 10,862 objects last more than 10s.
  - Many chances to manipulate legacy objects: average 22.87 modifications during the object's lifetime.
- What if a vulnerable / victim object is legacy?
  - Not isolated and mixed up with other objects.





Auditing legacy objects access is the focus of this briefing

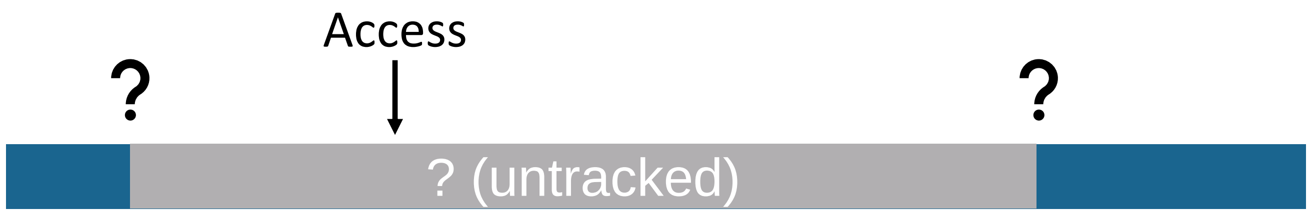
Agenda

- Motivation
- **Challenges & Design Overview**
  - Legacy Object Auditing - Challenge 1
  - Solution to Challenge 1
  - Legacy Object Auditing - Challenge 2
  - Solution to Challenge 2
  - Approach overview
- Example Workflow by CVE-2022-0995 & Video Demo
- Technical Details
- Evaluation

Legacy Object Auditing - Challenge 1

Legacy Object Auditing - Challenge 1

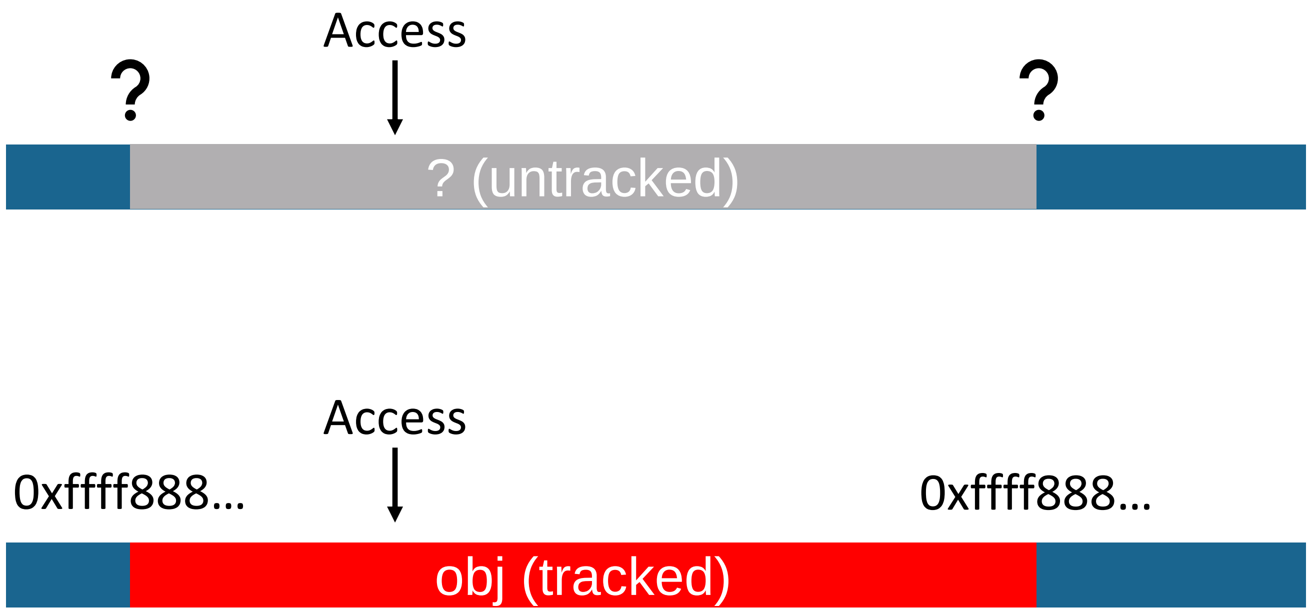
- Fact: Legacy objects are allocated before protection is enabled. We cannot record KASAN-like metadata for legacy objects.



Legacy Object Auditing - Challenge 1

- Fact: Legacy objects are allocated before protection is enabled.
  We cannot record KASAN-like metadata for legacy objects.

- Consequence: When a legacy object is accessed, start address, end address, and type are untracked.

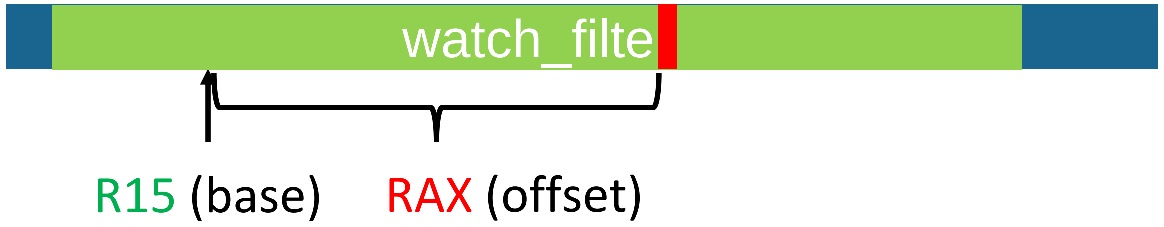


Solution to Challenge 1

Solution to Challenge 1

C: __set_bit(q->type, watch_filter->type_filter);

Asm: BTS [R15], RAX

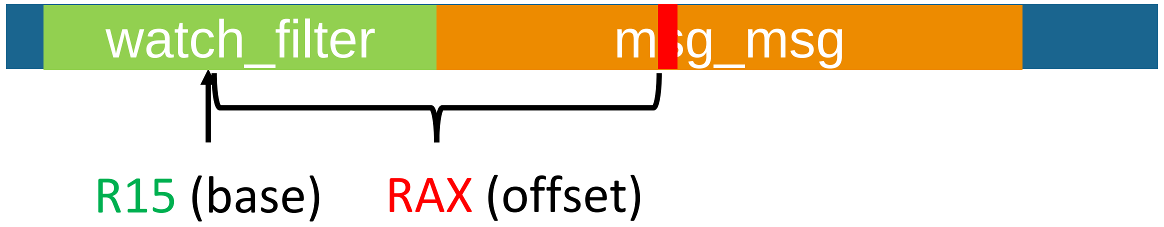


Solution to Challenge 1

- We use Machine Learning to infer the type of an accessed object, compared with access pointer type.

C: __set_bit(q->type, watch_filter->type_filter);

Asm: BTS [R15], RAX

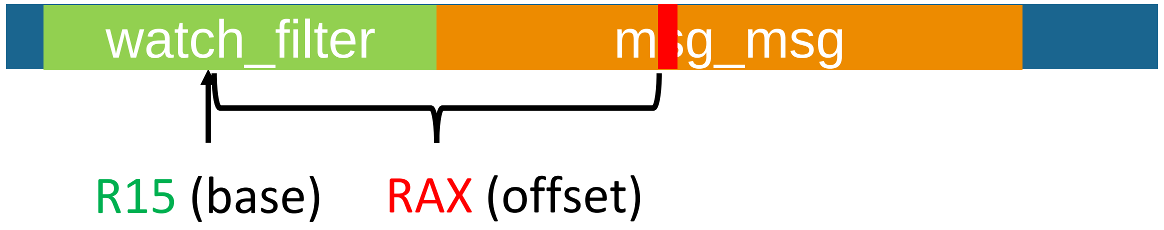


Solution to Challenge 1

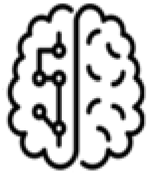
- We use Machine Learning to infer the type of an accessed object, compared with access pointer type.

C: __set_bit(q->type, watch_filter->type_filter);

Asm: BTS [R15], RAX



Human: What does these unorganized data mean?



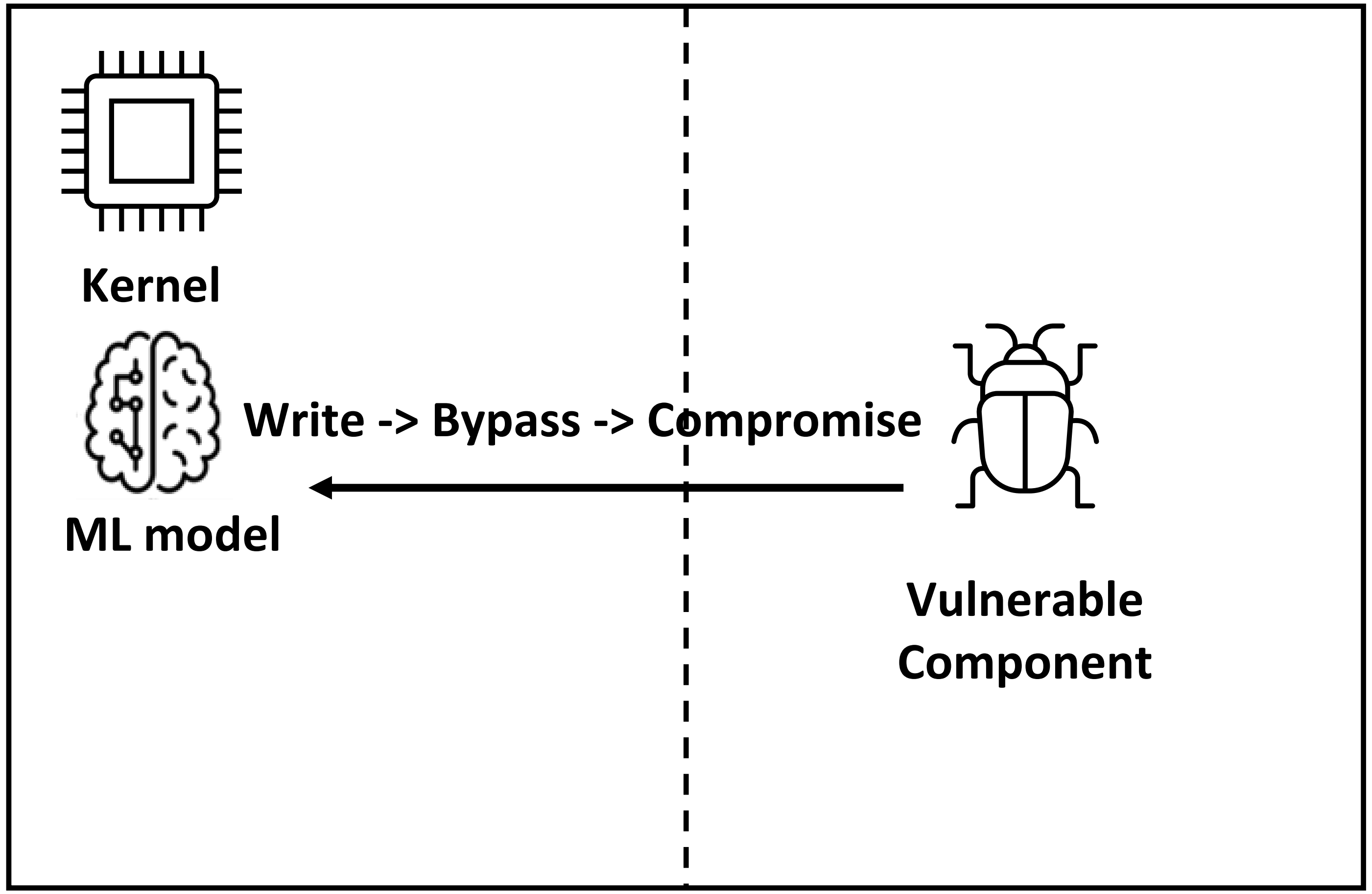
Trained AI: According to byte1, byte2, …, byteN, the object's type is inferred as msg_msg, indicating error because expected type should be watch_filter.

0xffff88810738e5c0	41 62 73 74 72 61 63 74
0xffff88810738e5c8	A0 79 04 02 81 88 FF FF
0xffff88810738e5d0	00 AC 04 02 81 88 FF FF

0xffff88810738e5c0	41 62 73 74 72 61 63 74
0xffff88810738e5c8	A0 79 04 02 81 88 FF FF
0xffff88810738e5d0	00 AC 04 02 81 88 FF FF

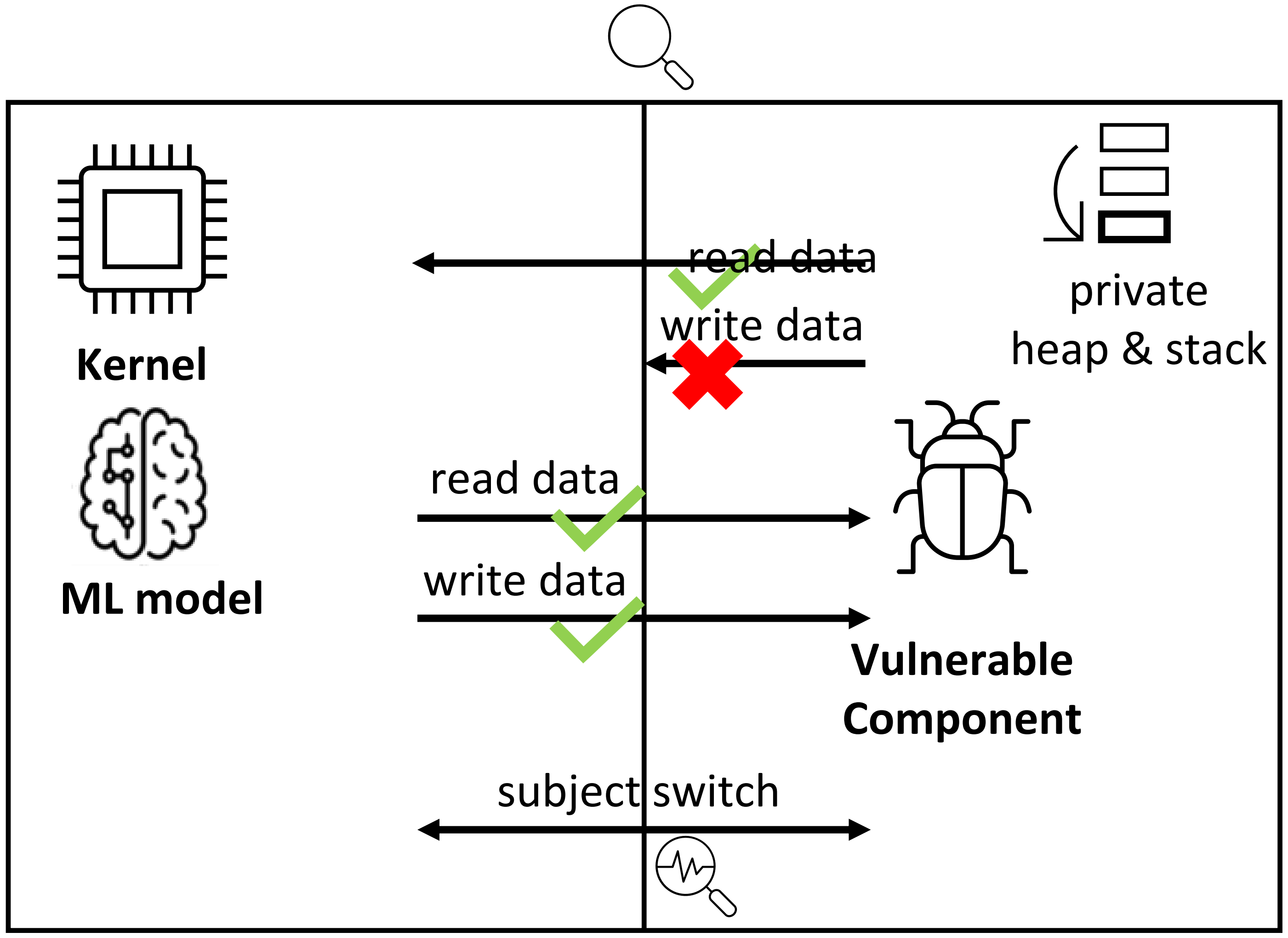
Legacy Object Auditing - Challenge 2

- Auditing integrity
- How to ensue the following integrity of auditing will not be compromised?
  - ML model integrity
  - Data-Flow integrity
  - Control-Flow integrity



Solution to Challenge 2

- Kernel Code instrumentation
  - Audit each read / write
  - Audit subject switch
- Private heap & stack
  - Vulnerable Component only use its own private data structures.

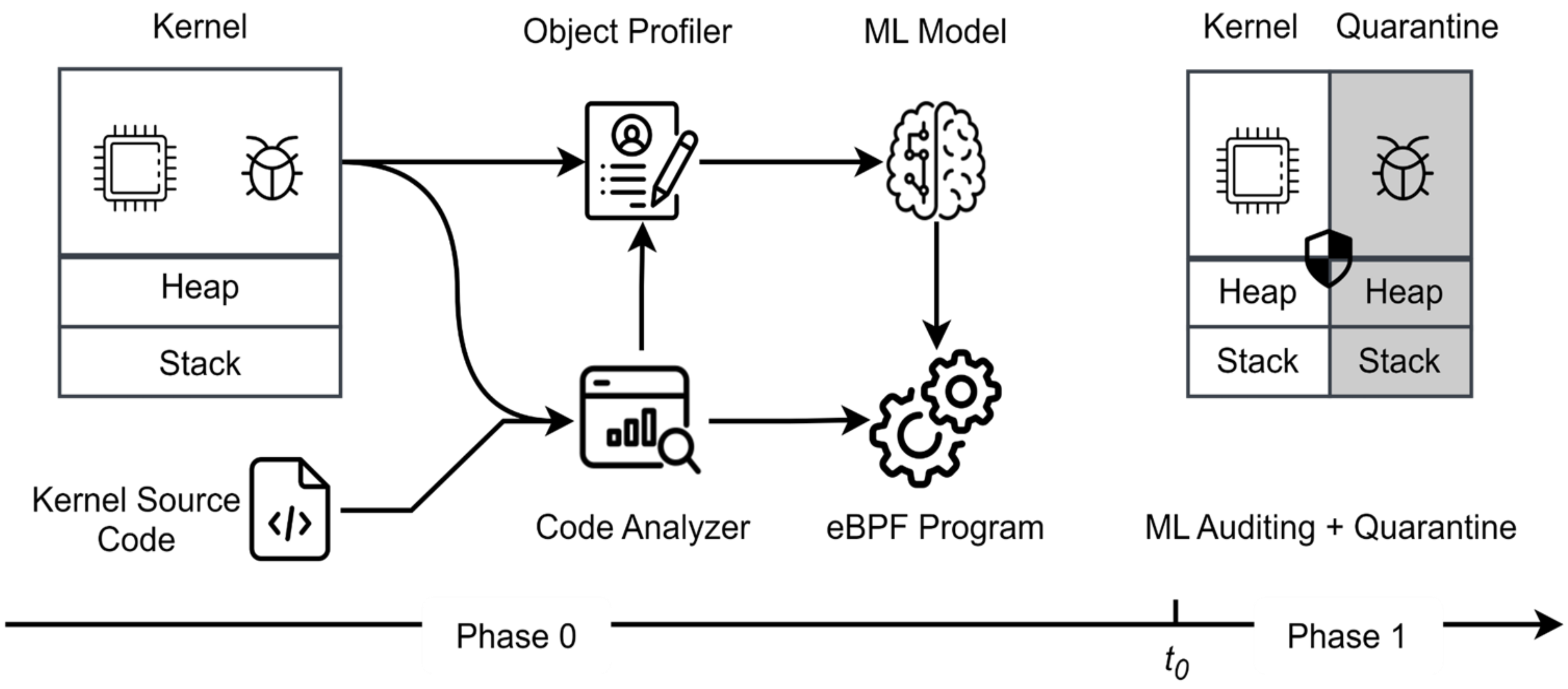


Access Auditing Policy to Challenge 2

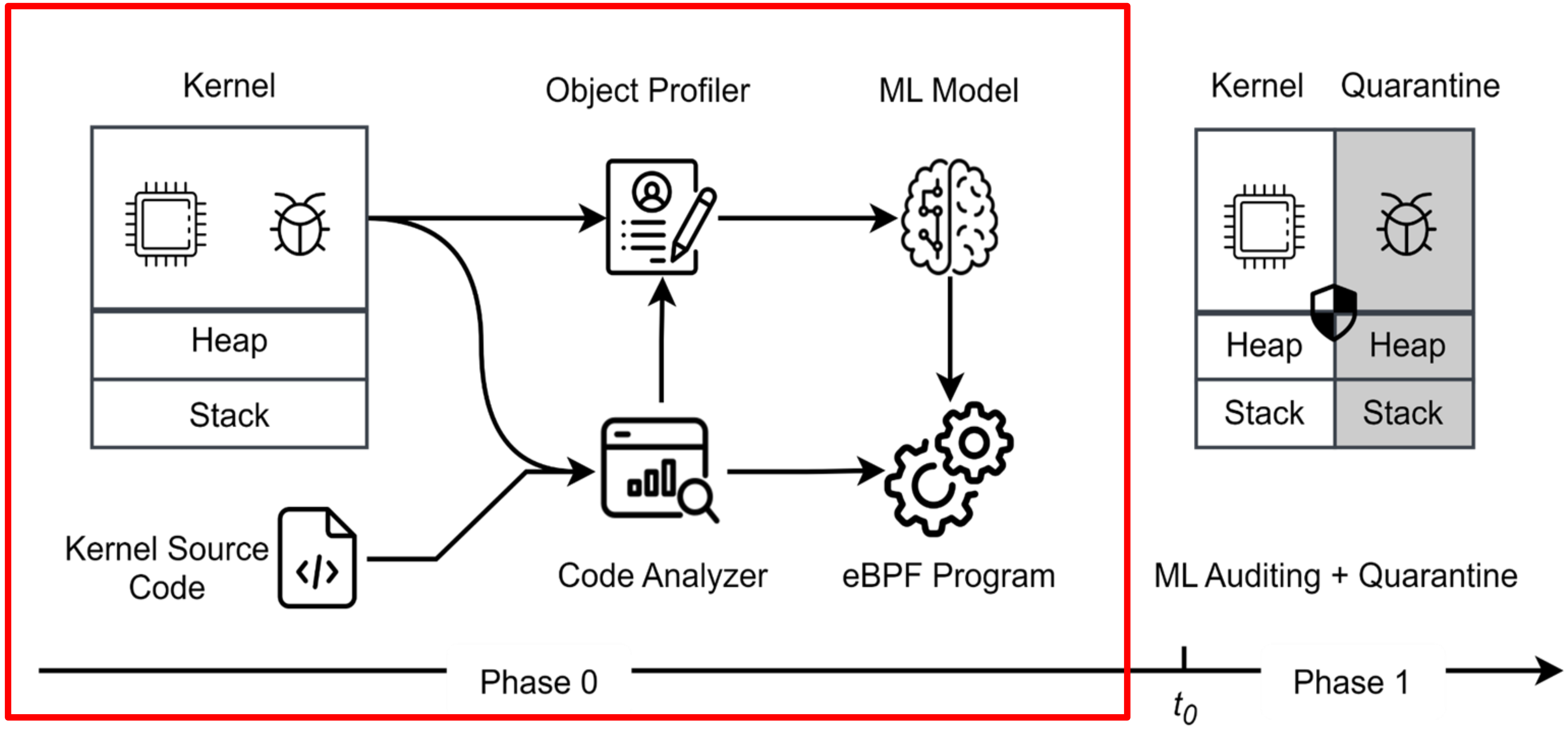
	Trusted Kernel			Untrusted component		
	read	write	exec	read	write	exec
Kernel Code	✓		✓	✓		
Kernel Data	✓	✓		✓		
Kernel Heap	✓	✓		✓		
Kernel Stack	✓	✓		✓		
Auditing mechanism	✓	✓	✓	✓		
Component Code	✓	✓	✓	✓		✓
Component Data	✓	✓		✓	✓	
Component Heap	✓	✓		✓	✓	
Component Stack	✓	✓		✓	✓	

Access auditing policy

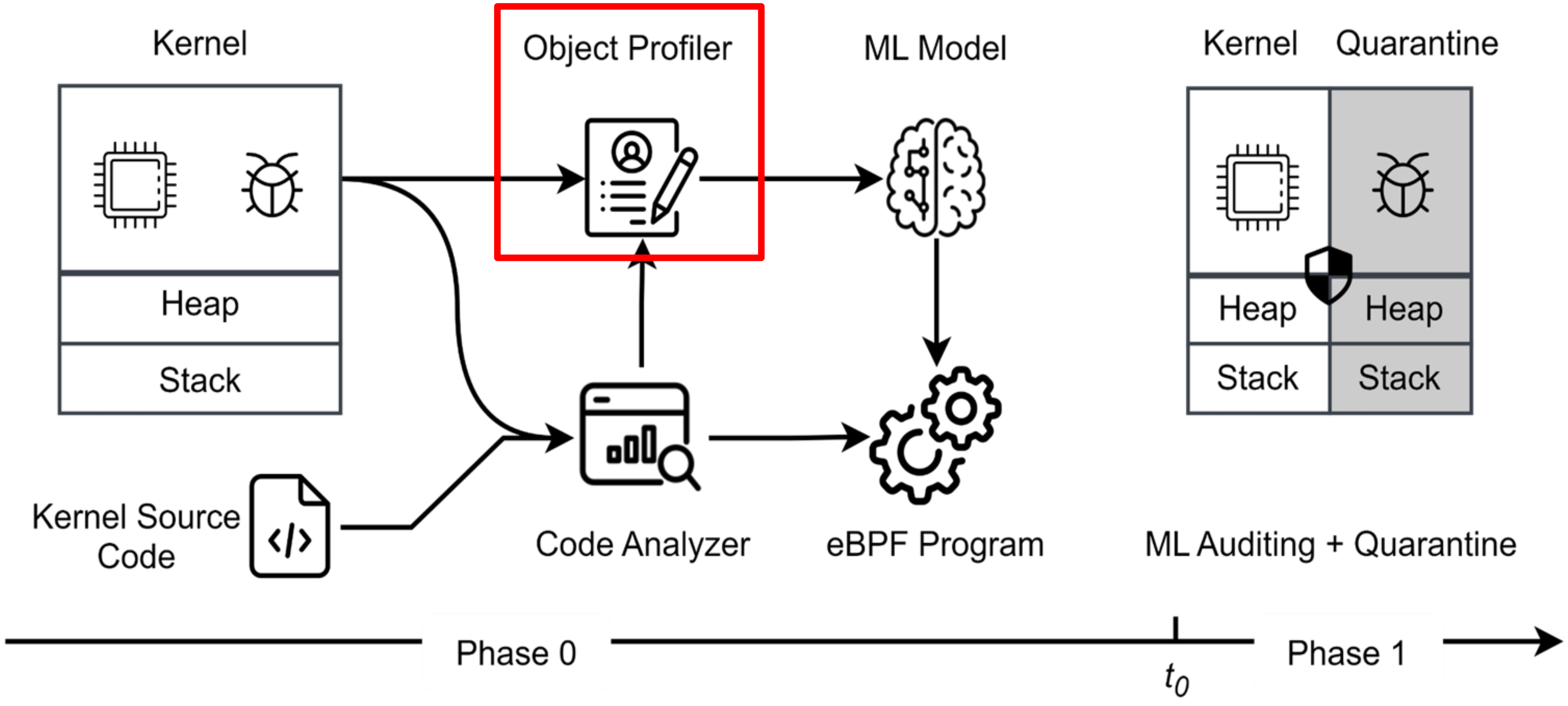
On-the-Fly Quarantine (O2Q) Overview



On-the-Fly Quarantine (O2Q) Overview

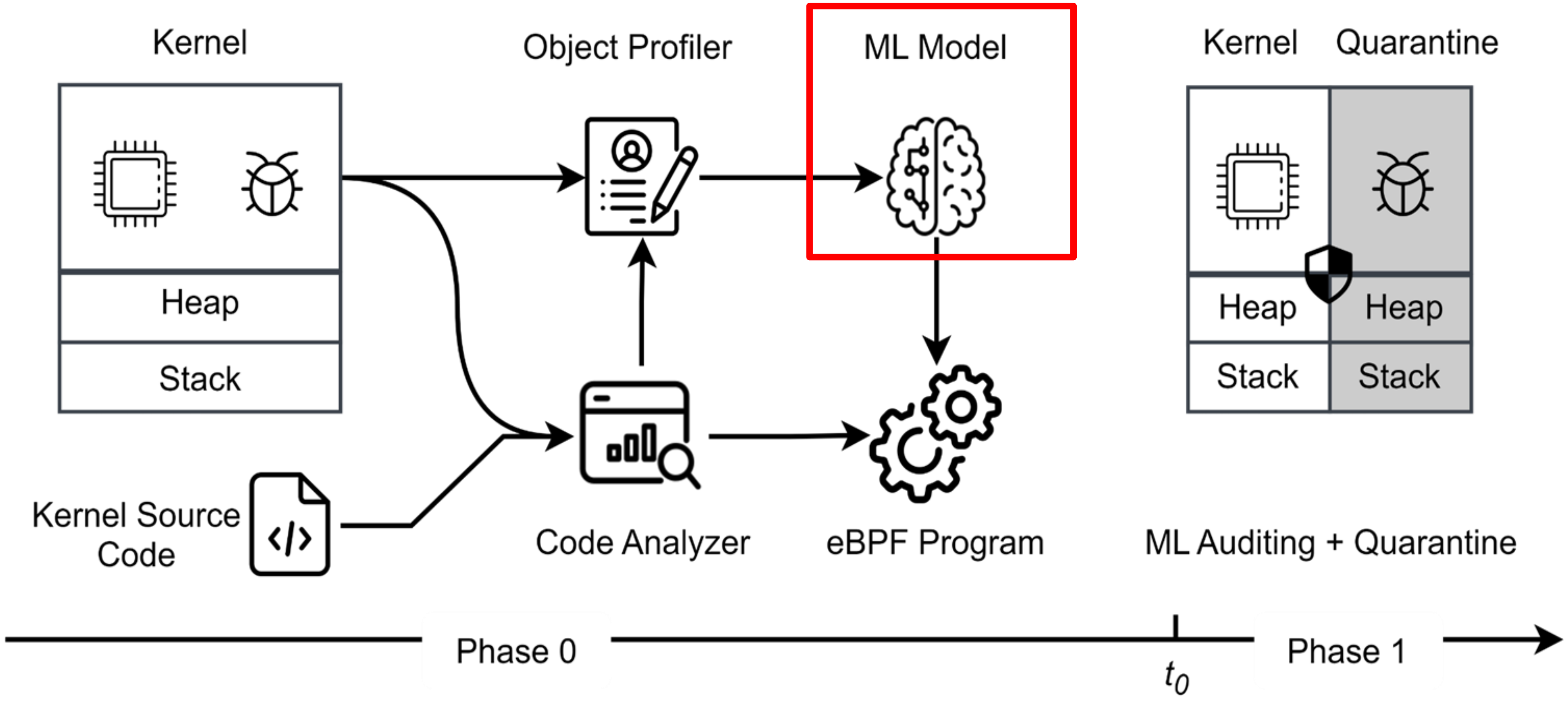


On-the-Fly Quarantine (O2Q) Overview



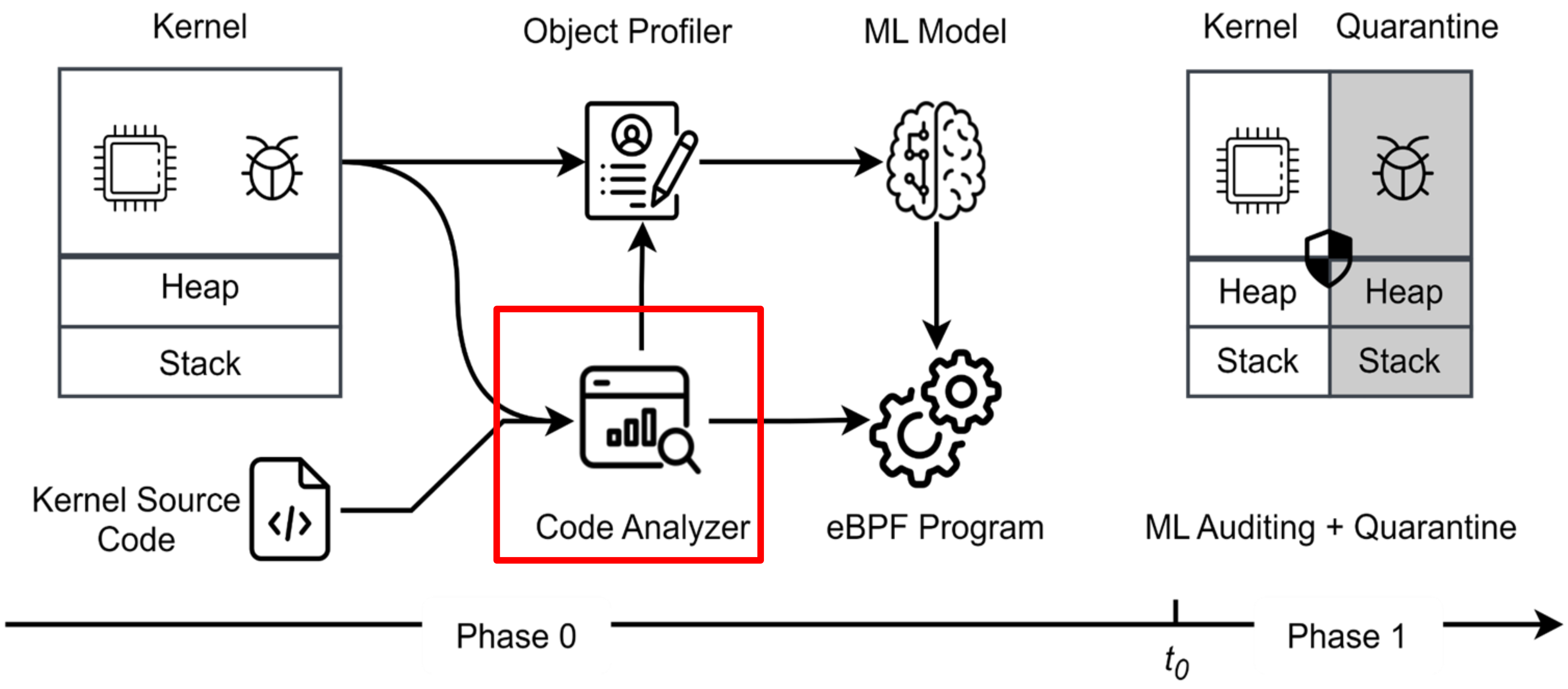
Collect data for ML model training: object's type and content

On-the-Fly Quarantine (O2Q) Overview



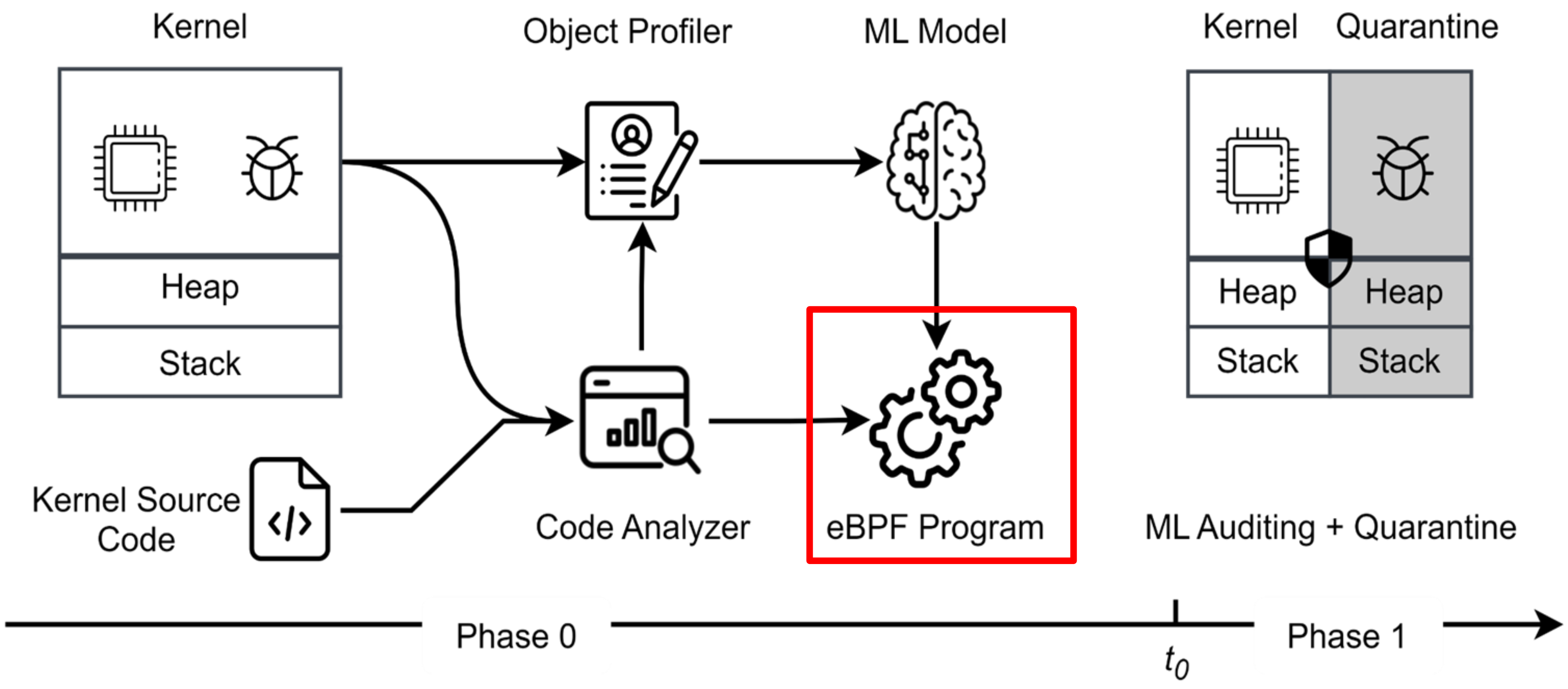
Train ML model inferring object's type based on its content

On-the-Fly Quarantine (O2Q) Overview



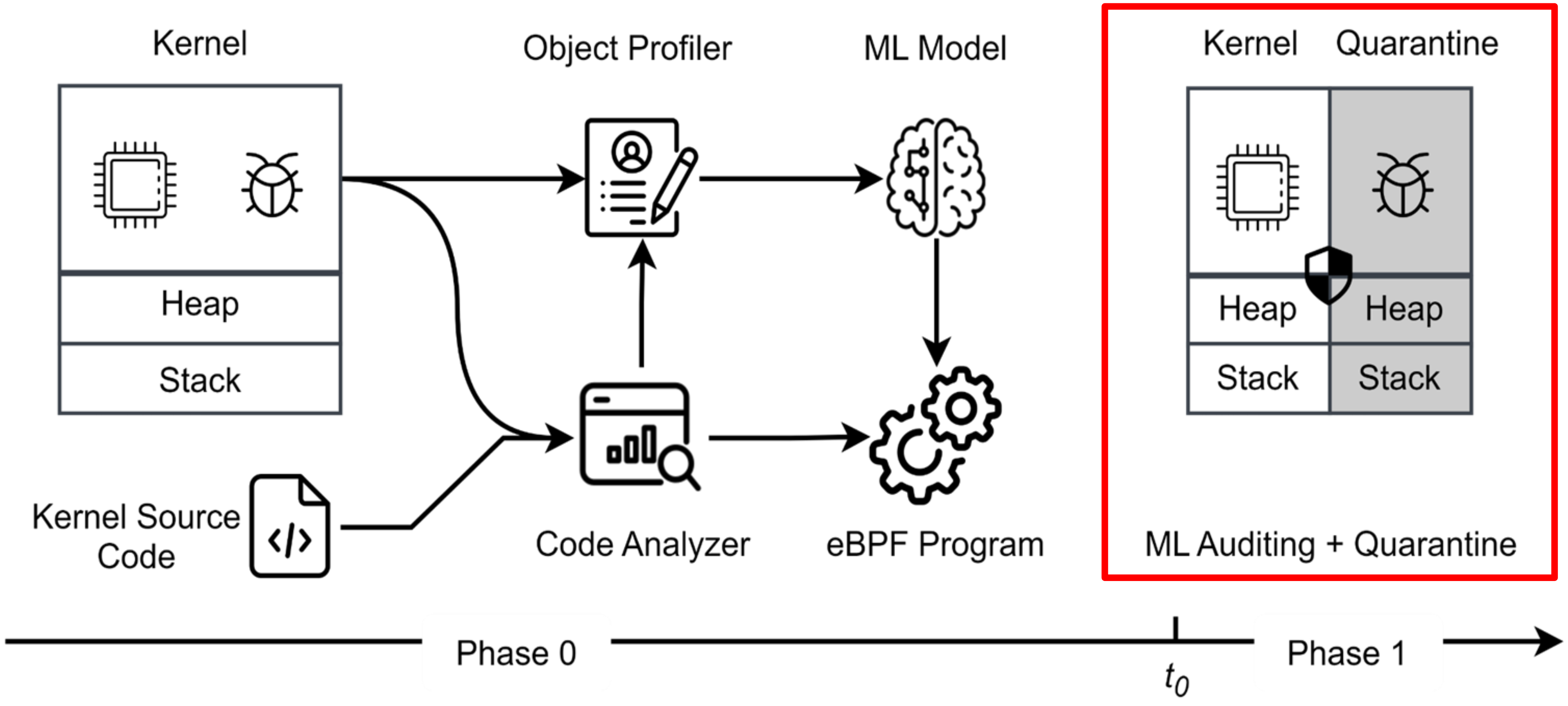
Identify instructions for instrumentation

On-the-Fly Quarantine (O2Q) Overview



Implement quarantine, examine object's type at runtime

On-the-Fly Quarantine (O2Q) Overview

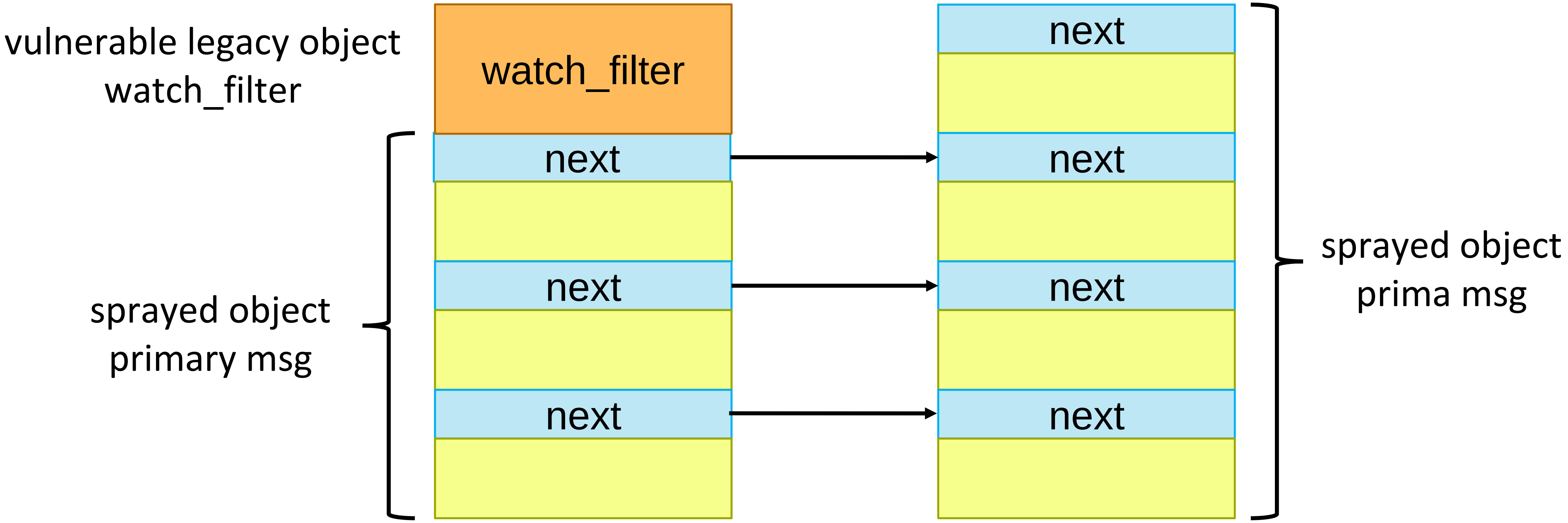


Agenda

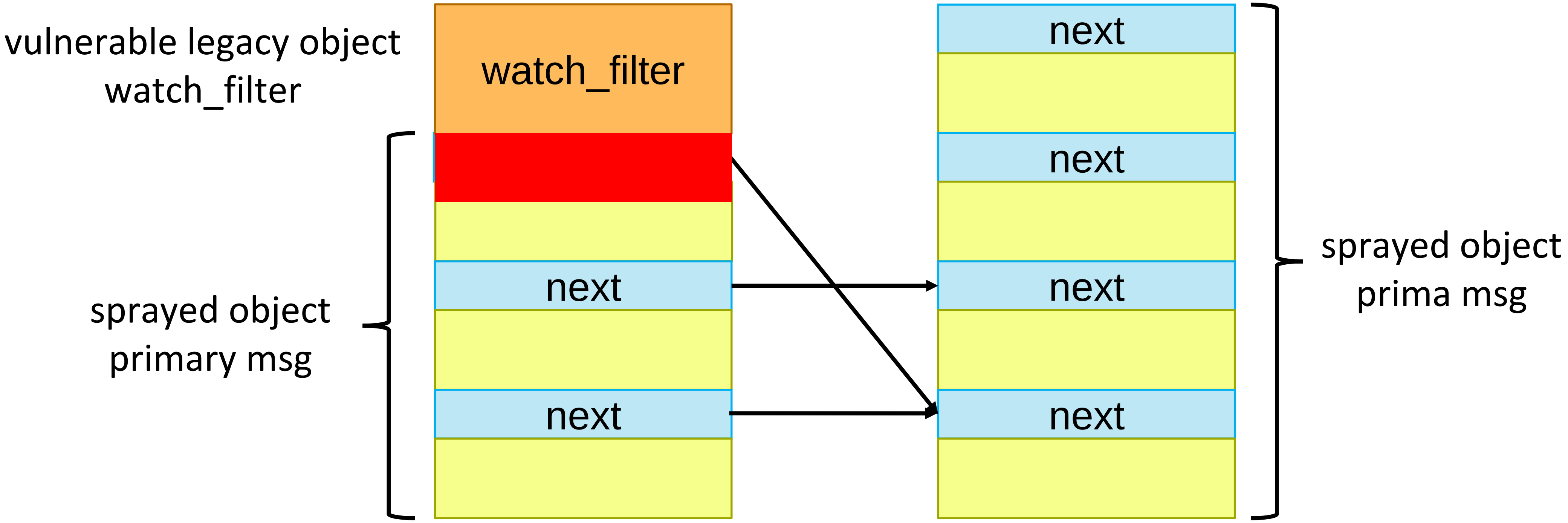
- Motivation
- Challenges & Design Overview
- **Example Workflow by CVE-2022-0995 & Video Demo**
- Technical Details
- Evaluation

A Working Example: CVE-2022-0995

A Working Example: CVE-2022-0995



A Working Example: CVE-2022-0995

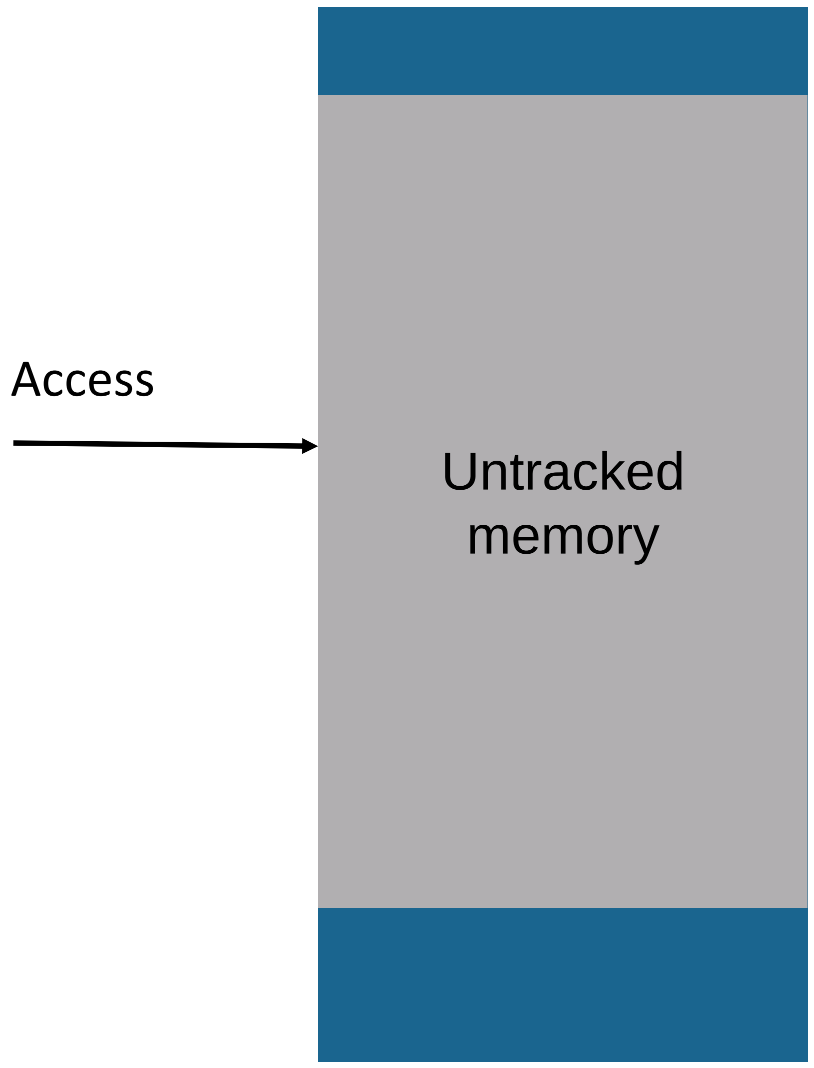


Two primary msg reference
this secondary msg.
Results in UAF

O2Q Workflow on CVE-2022-0995

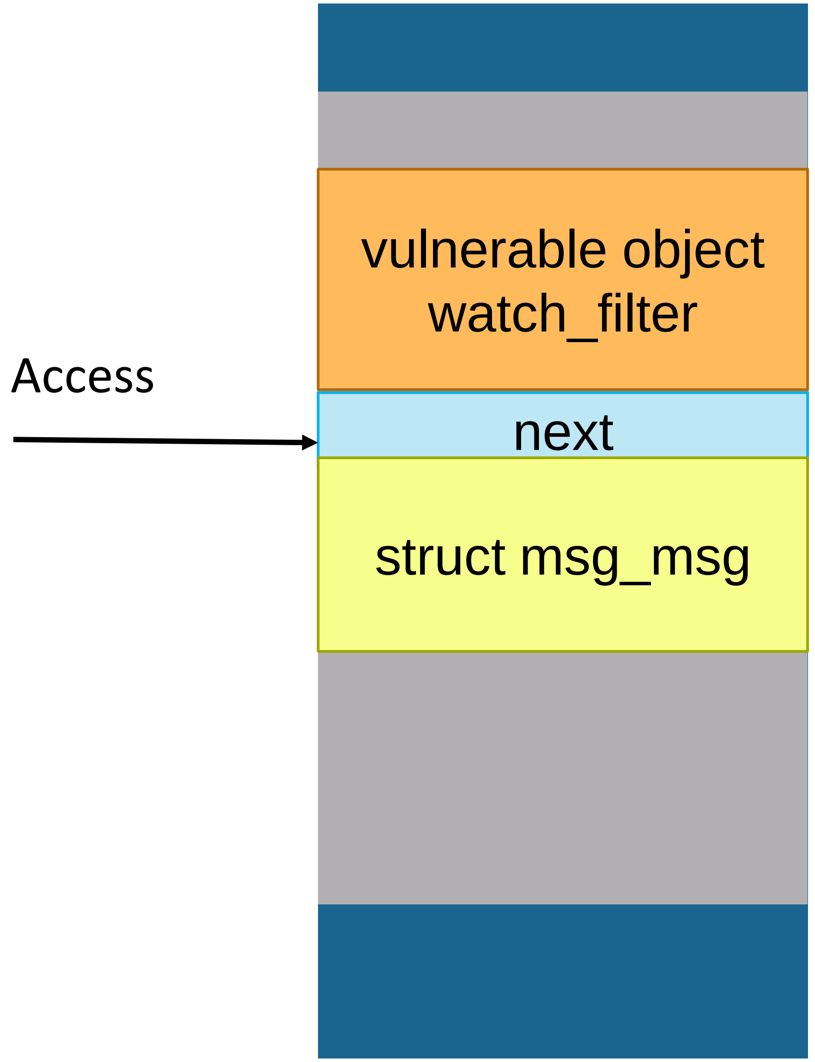
O2Q Workflow on CVE-2022-0995

- The kernel is executing vulnerable component in quarantine zone.



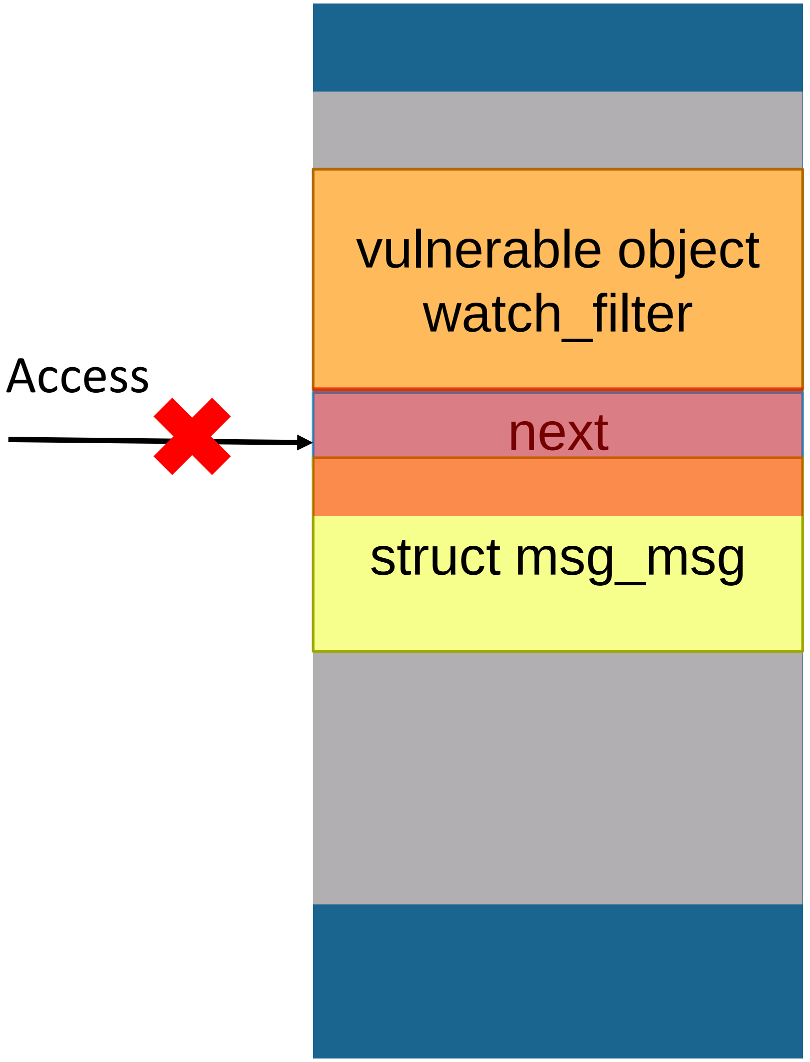
O2Q Workflow on CVE-2022-0995

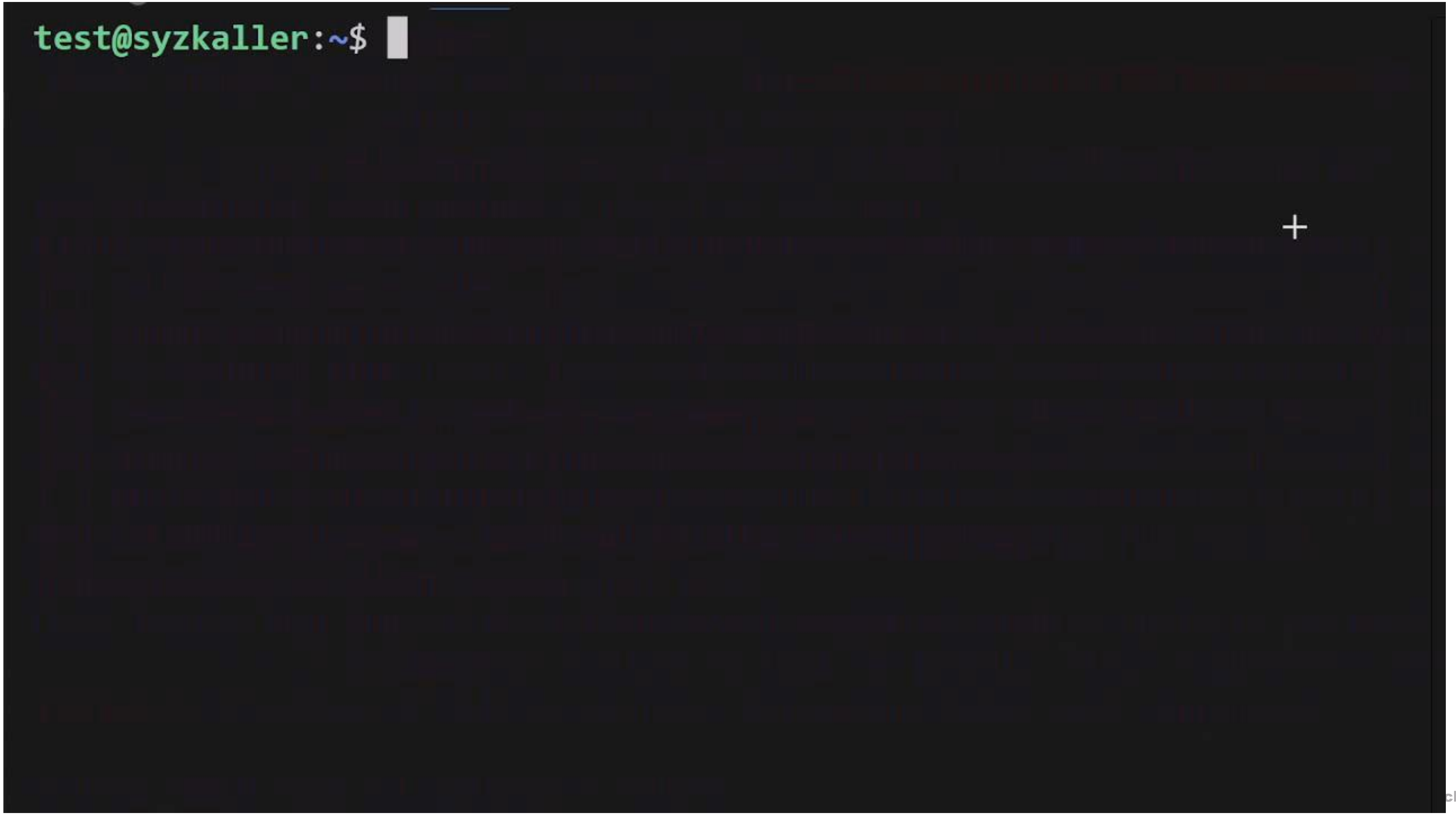
- The kernel is executing vulnerable component in quarantine zone.
- The executing instruction should access watch_filter by Code Analyzer and Object Profiler.
- The eBPF program instrumented to the executing instructions encompasses the trained ML model.



O2Q Workflow on CVE-2022-0995

- The kernel is executing vulnerable component in quarantine zone.
- The executing instruction should access watch_filter by Code Analyzer and Object Profiler.
- The eBPF program instrumented to the executing instructions encompasses the trained ML model.
- The ML model infers the accessed object is msg_msg, indicating error.



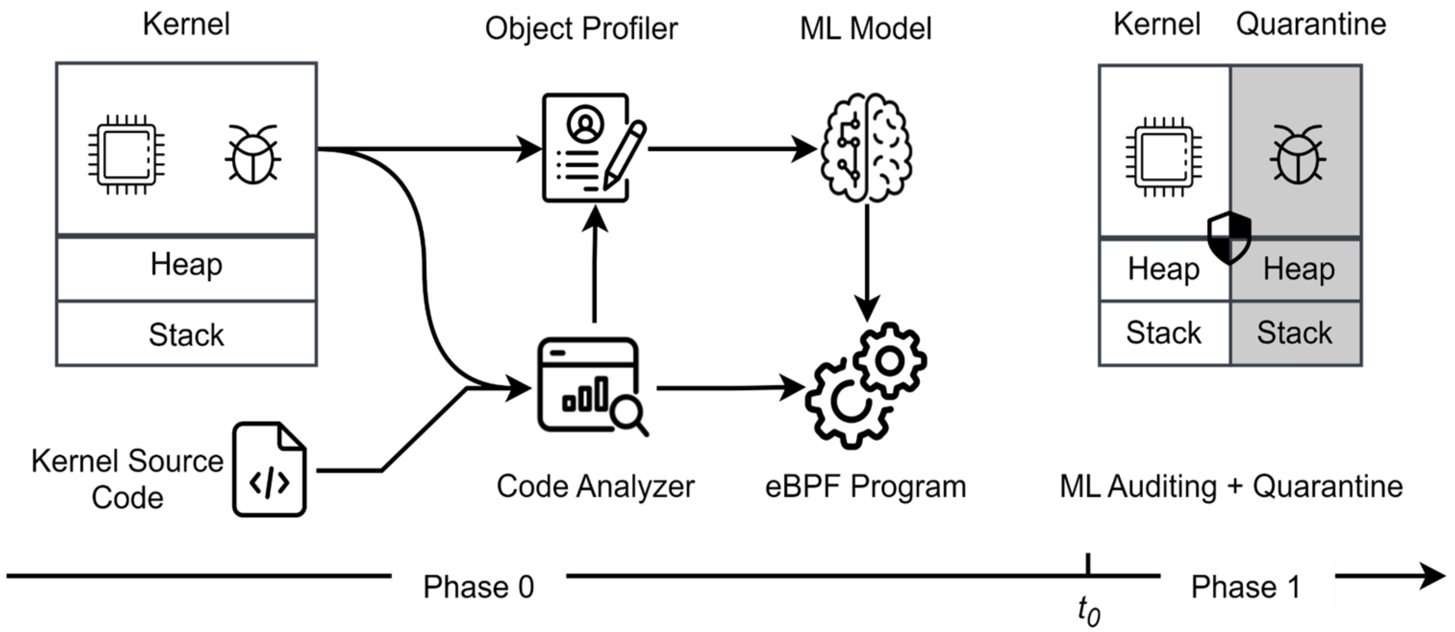

test@syzkaller:~$

```
root@syzkaller:~/bpf# ./o2q_CVE-2022-0995
```

```
test@syzkaller:~$ uname -a
```

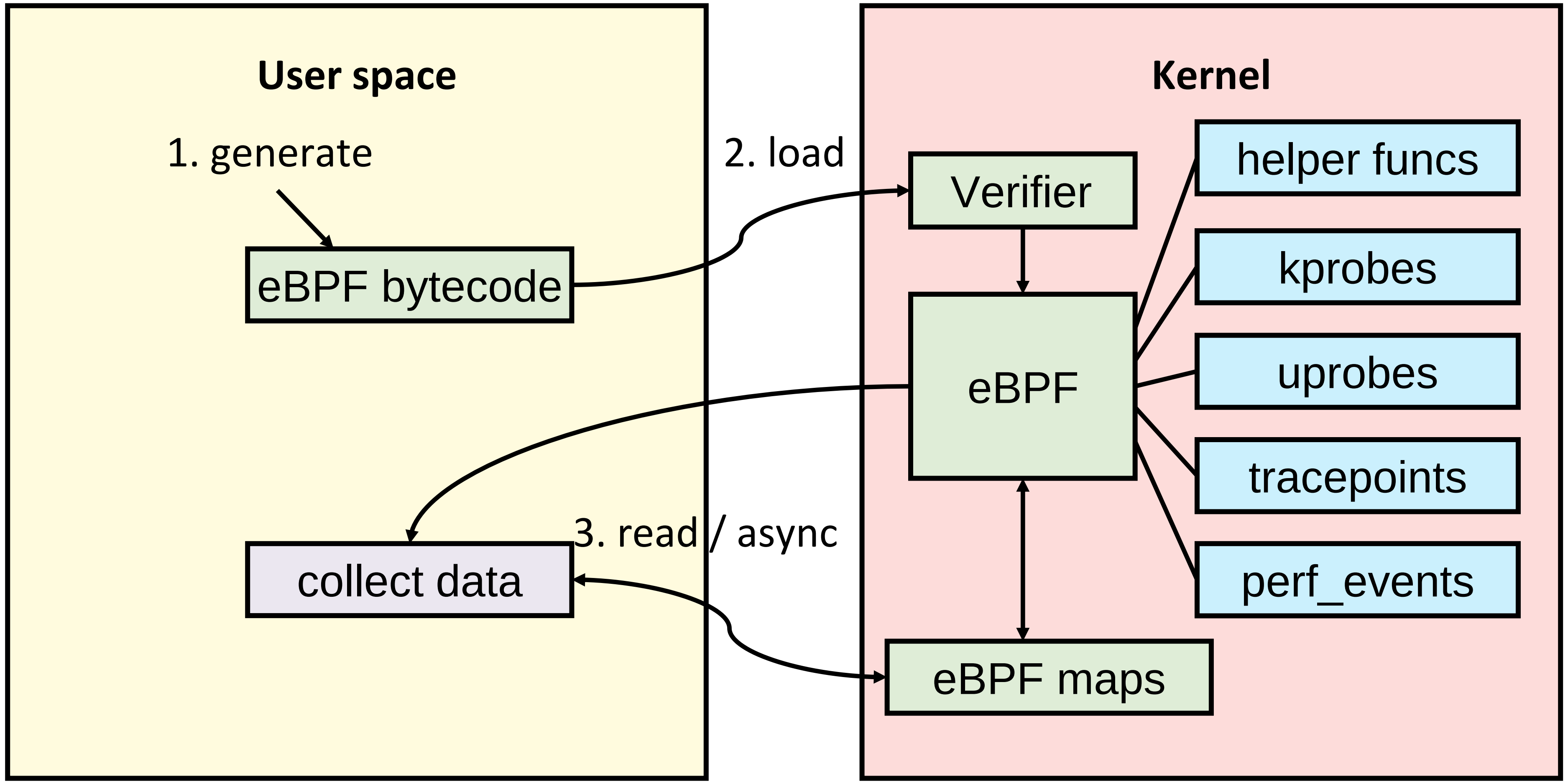
Agenda

- Motivation
- Challenges & Design Overview
- Example Workflow by CVE-2022-0995 & Video Demo
- **Technical Details**
  - Technical Backgrounds – eBPF & ML
  - O2Q Components
- Evaluation

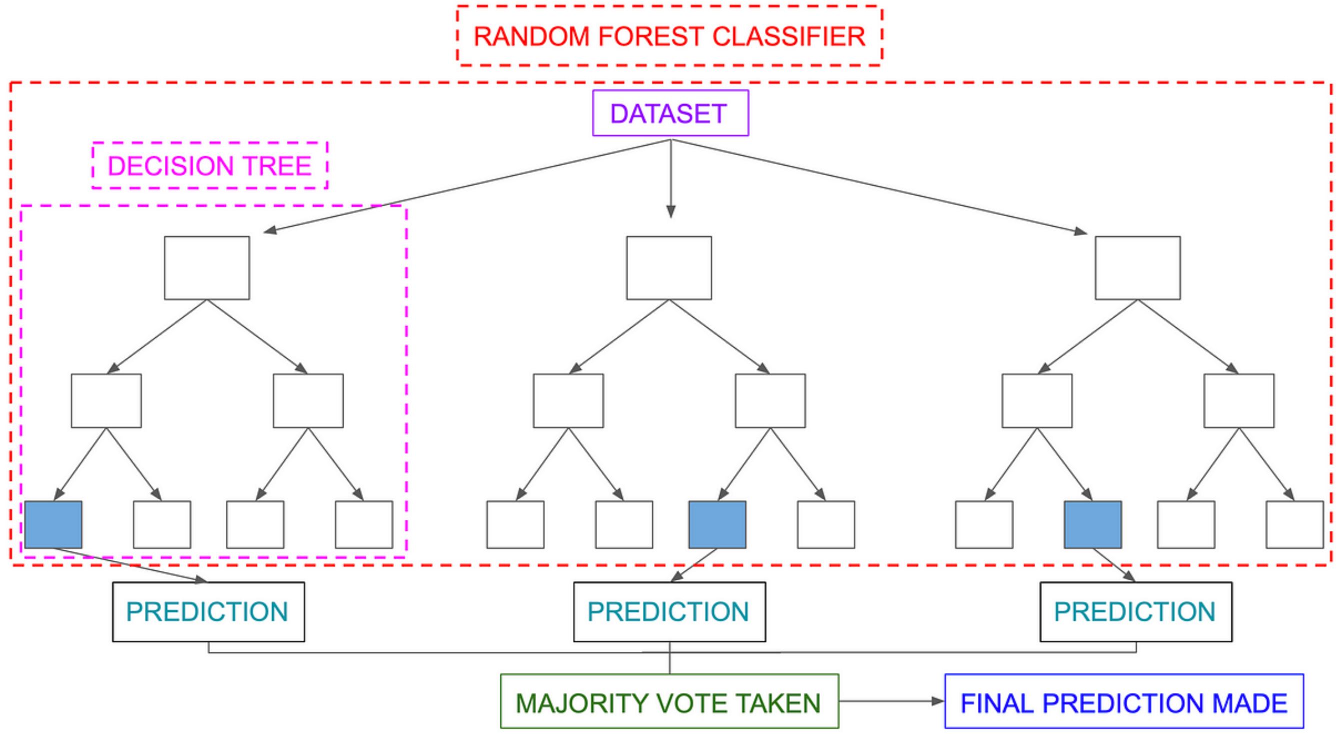


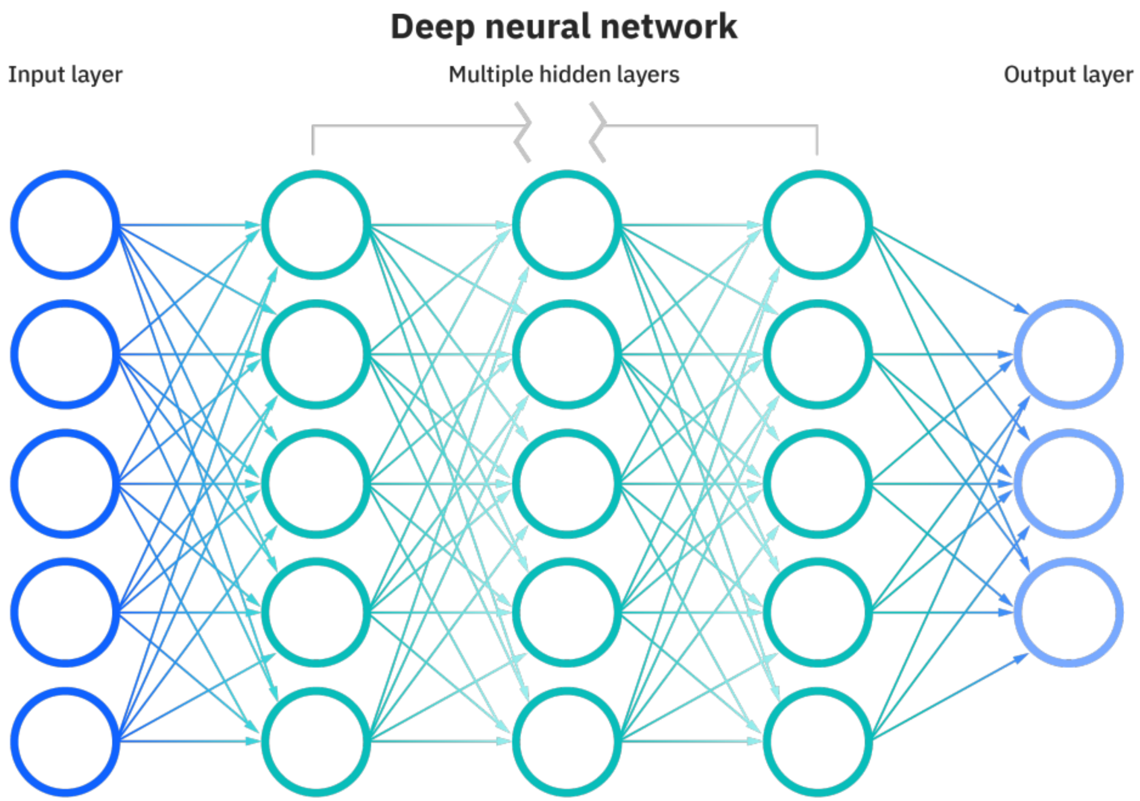
Technical Background - eBPF

- Sandbox virtual machine in kernel.
- No need to modify kernel code or load module.
- Can hook any instruction.
- Own verifier.
- High performance using JIT.
- eBPF maps for data exchange.
- eBPF helper functions.



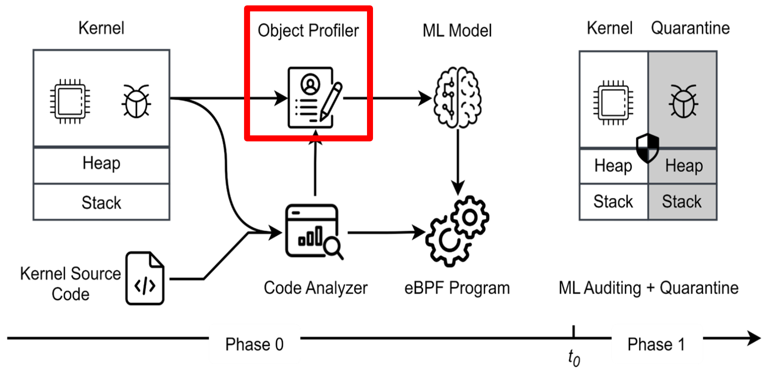
Technical Background - AI Models





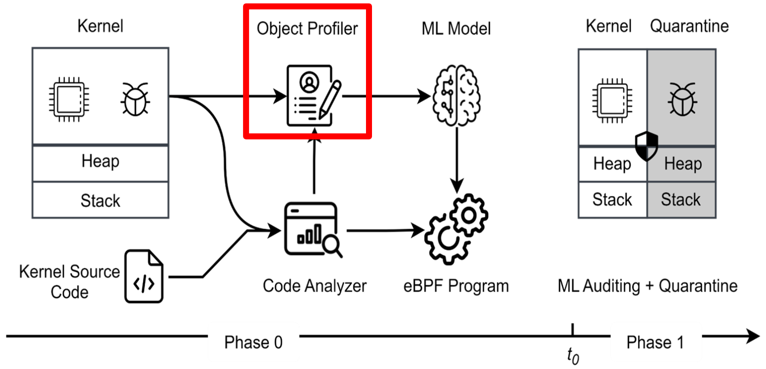
Source: https://medium.com/analytics-vidhya/machine-learning-decision-trees-and-random-forest-classifiers-81422887a544
https://www.linkedin.com/pulse/introduction-neural-networks-how-machines-process-data-lakhani

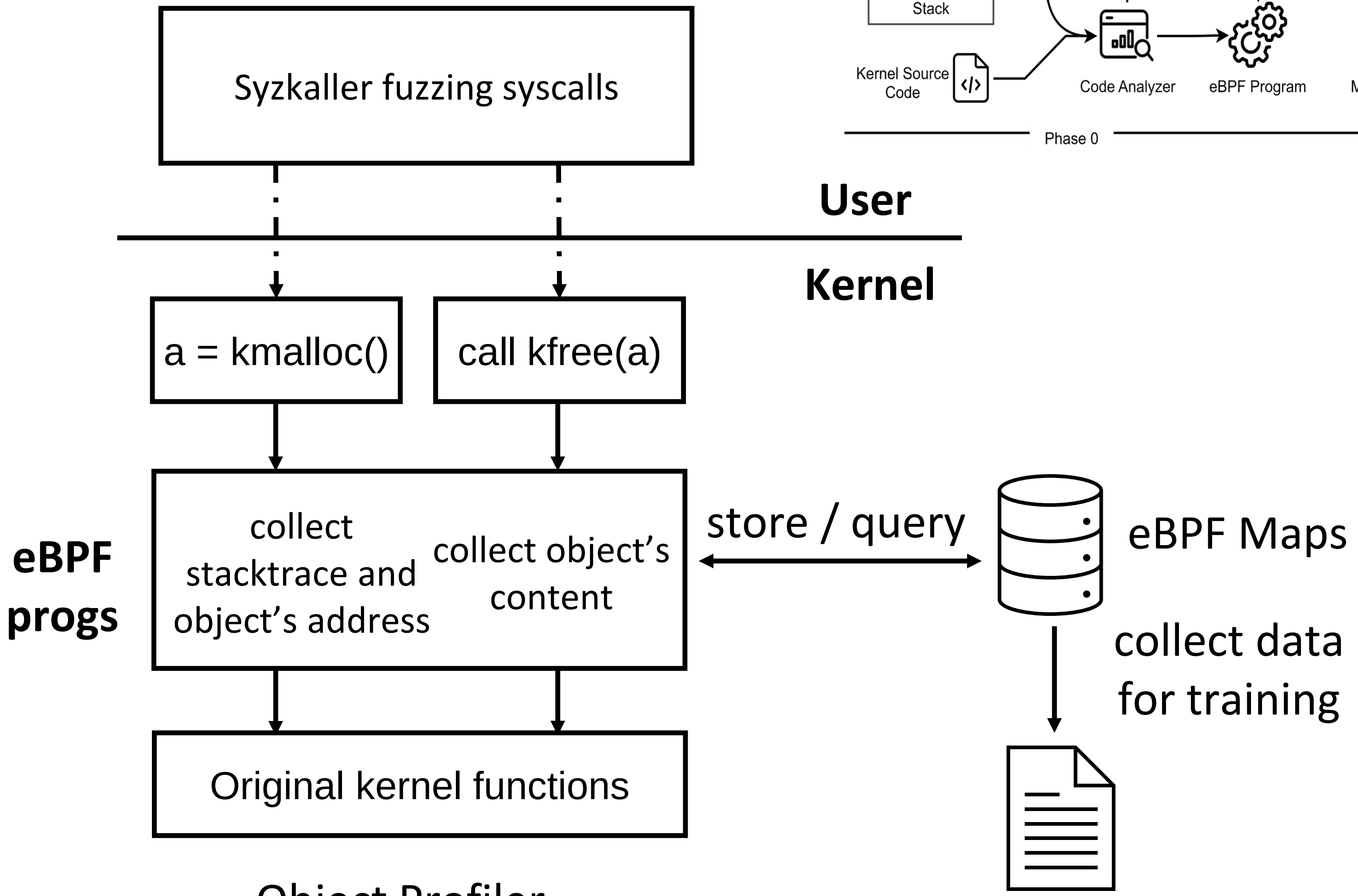
Object Profiler



Object Profiler

- Use Syzkaller to enrich data source.
- Collect each object's content and type for training.

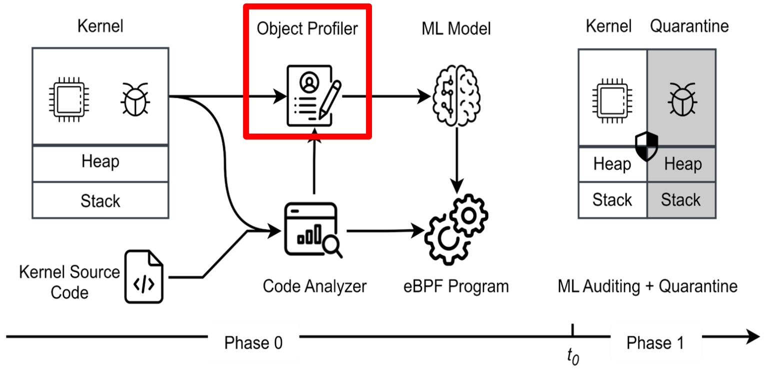


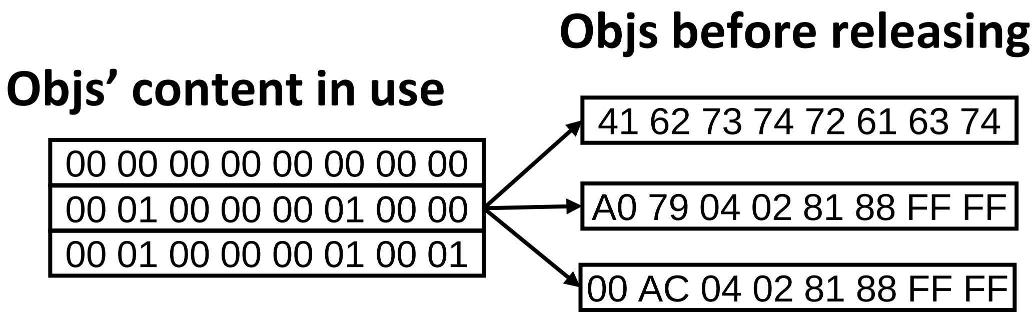


Object Profiler

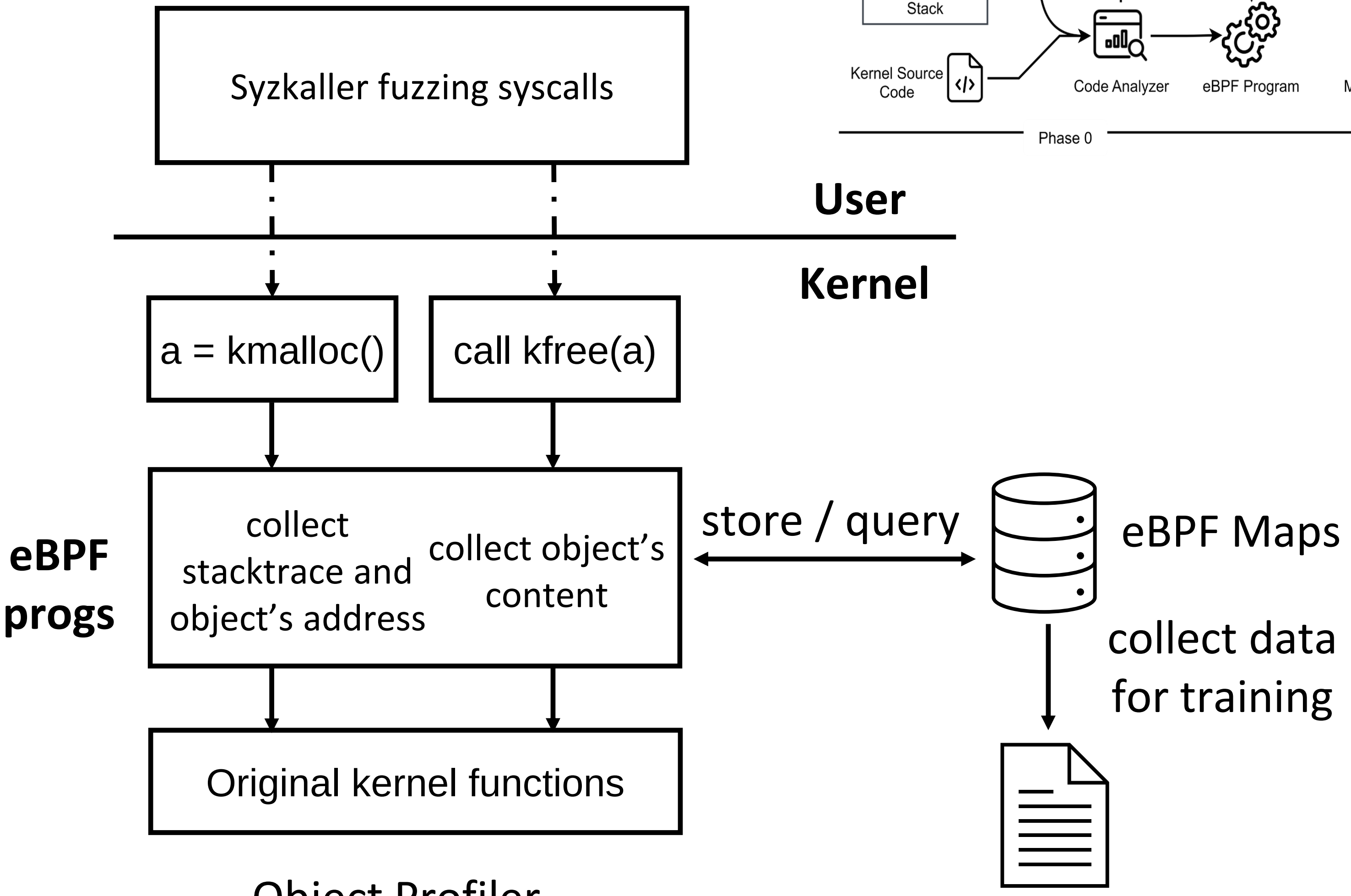
Object Profiler

- Use Syzkaller to enrich data source.
- Collect each object's content and type for training.
- Collect at object's release site: object possesses the most features that best reflect its characteristics.





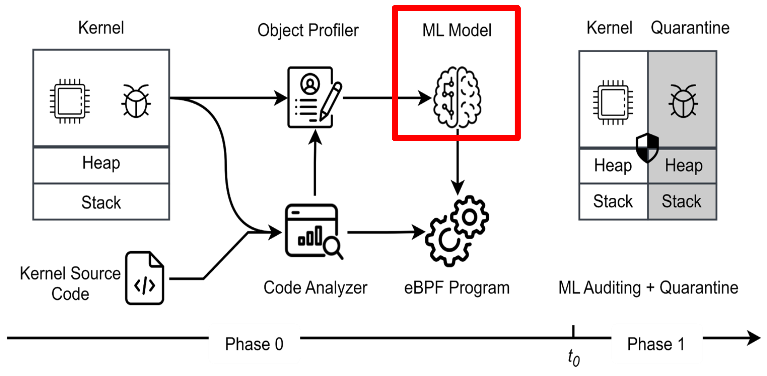
Uncharacterized vs. characterized



Object Profiler

ML Model

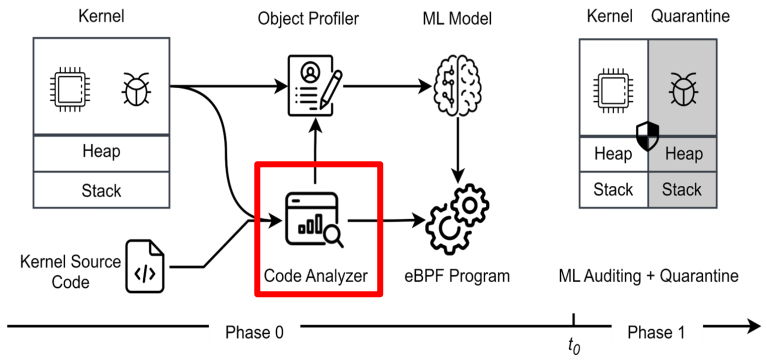
- Feature:
  - Object's data content as feature
- Label
  - Object's type and whether belongs to quarantine zone



	Tabular Data Processing	Interpretable	Defined Execution Time	Quantitative Accuracy	Convert to BPF Implementation
Decision Tree	✓	✓	✓	✓	✓
Random Forest	✓	✓	✓	✓	
Neural Network				✓	

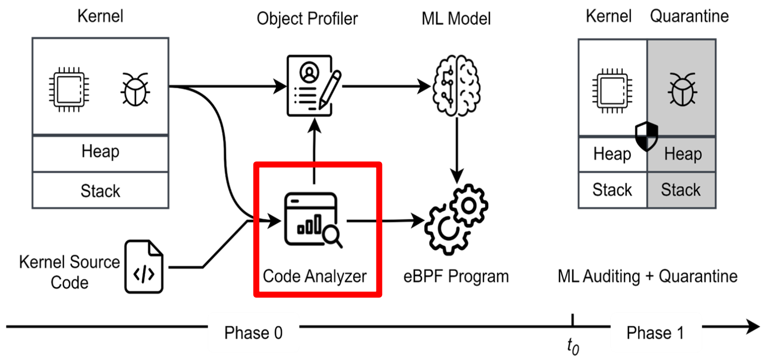
different ML model comparison

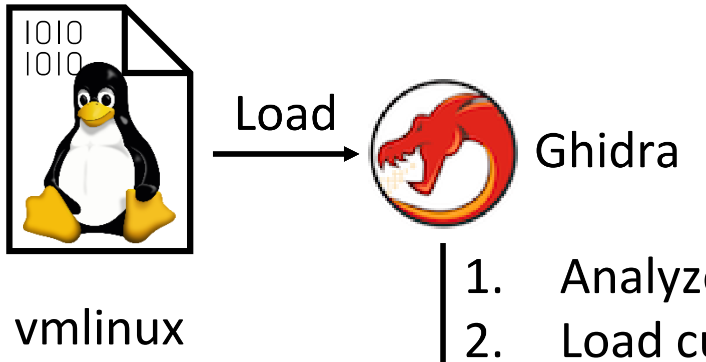
Code Analyzer



Code Analyzer

- Identify Linux Kernel’s instructions
  - Indirect jump
  - Indirect call
  - Memory write
  - Subject switch



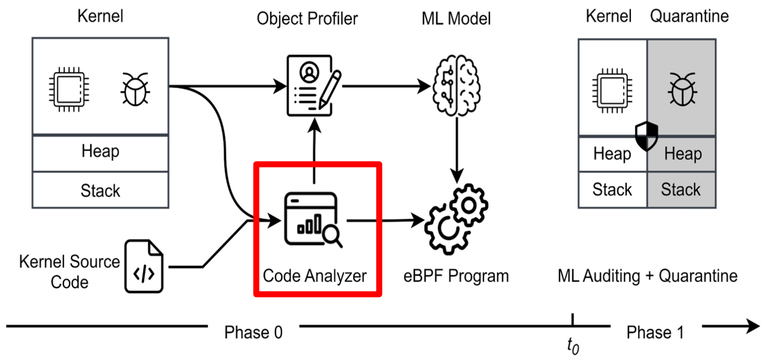


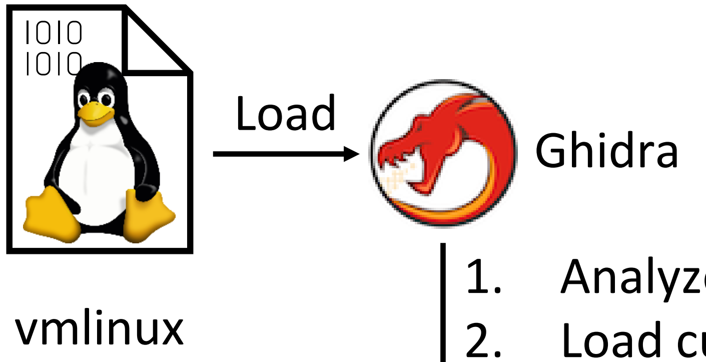
1. Analyze Kernel Binary File
2. Load custom rules
3. Identify desired instructions
4. Output entries

```
Indirect jump: call *%rax

Memory write: mov $0x0, (%rsi, %rdx, 1)

Determined address: mov off(%rip), %rax

Stack frame create: sub offset, %rsp

Stack access: mov x, off (%rsp/rbp)

Redundant check: mov  $0x0, off1(%rsi)

Redundant check: mov  $0x0, off2(%rsi)
```

Code Analyzer

- Identify Linux Kernel's instructions
  - Indirect jump
  - Indirect call
  - Memory write
  - Subject switch
- Efficiency Optimization:
  - Skip read
  - Skip determining address
  - Skip redundant check





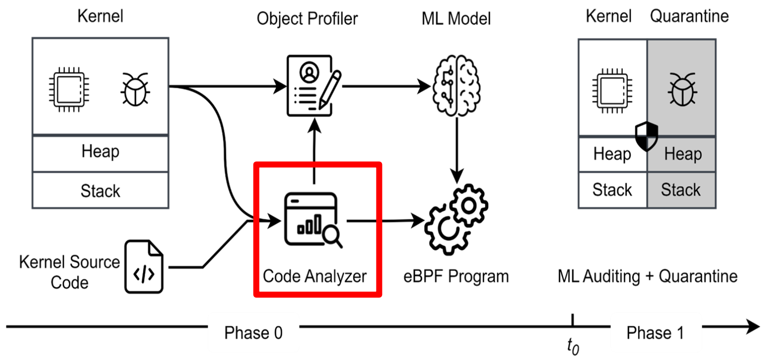
1. Analyze Kernel Binary File
2. Load custom rules
3. Identify desired instructions
4. Output entries

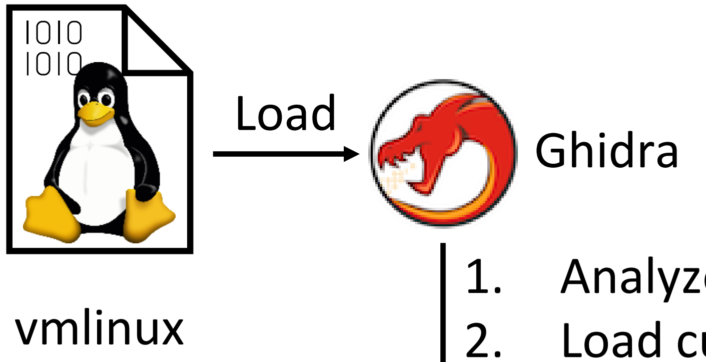
```
Indirect jump: call *%rax
Memory write: mov $0x0, (%rsi, %rdx, 1)
Determined address: mov off(%rip), %rax
Stack frame create: sub offset, %rsp
Stack access: mov x, off (%rsp/rbp)
Redundant check: mov  $0x0, off1(%rsi)
Redundant check: mov  $0x0, off2(%rsi)
```

Code Analyzer

- Identify Linux Kernel’s instructions
  - Indirect jump
  - Indirect call
  - Memory write
  - Subject switch
- Efficiency Optimization:
  - Skip read
  - Skip determining address
  - Skip redundant check

Reduced 24.07% instrument entries.



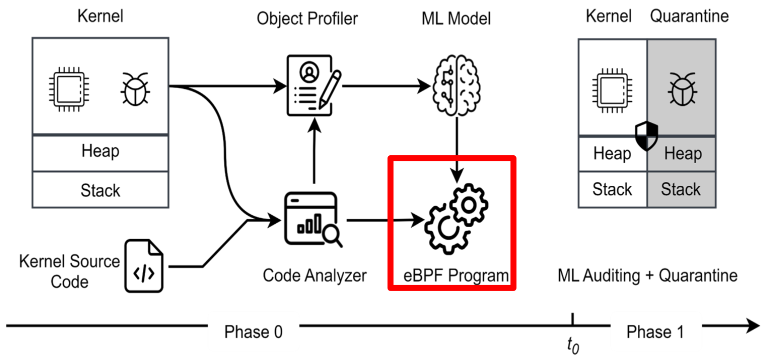


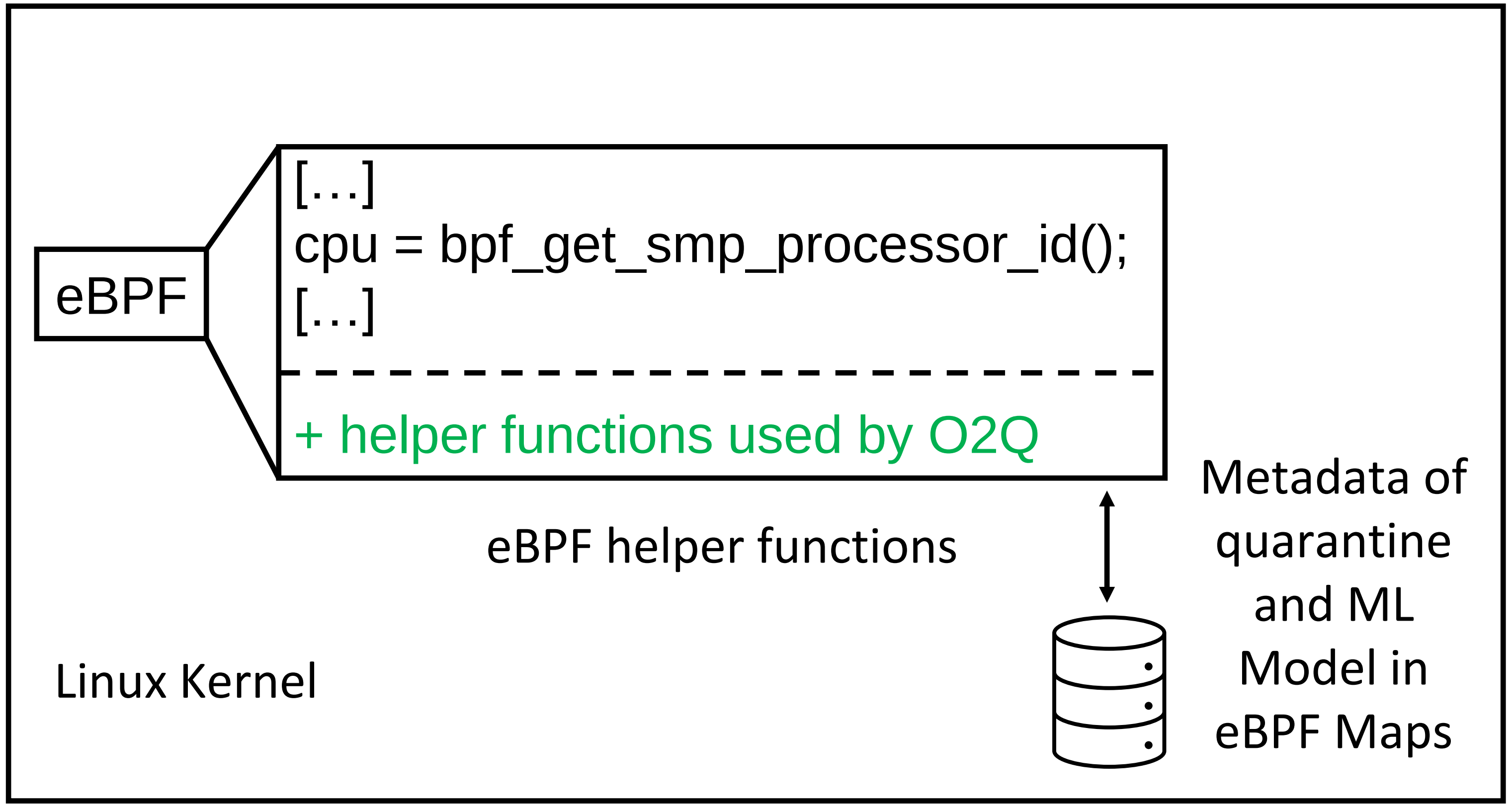
1. Analyze Kernel Binary File
2. Load custom rules
3. Identify desired instructions
4. Output entries

```
Indirect jump: call *%rax
Memory write: mov $0x0, (%rsi, %rdx, 1)
Determined address: mov off(%rip), %rax
Stack frame create: sub offset, %rsp
Stack access: mov x, off (%rsp/rbp)
Redundant check: mov  $0x0, off1(%rsi)
Redundant check: mov  $0x0, off2(%rsi)
```

eBPF Program

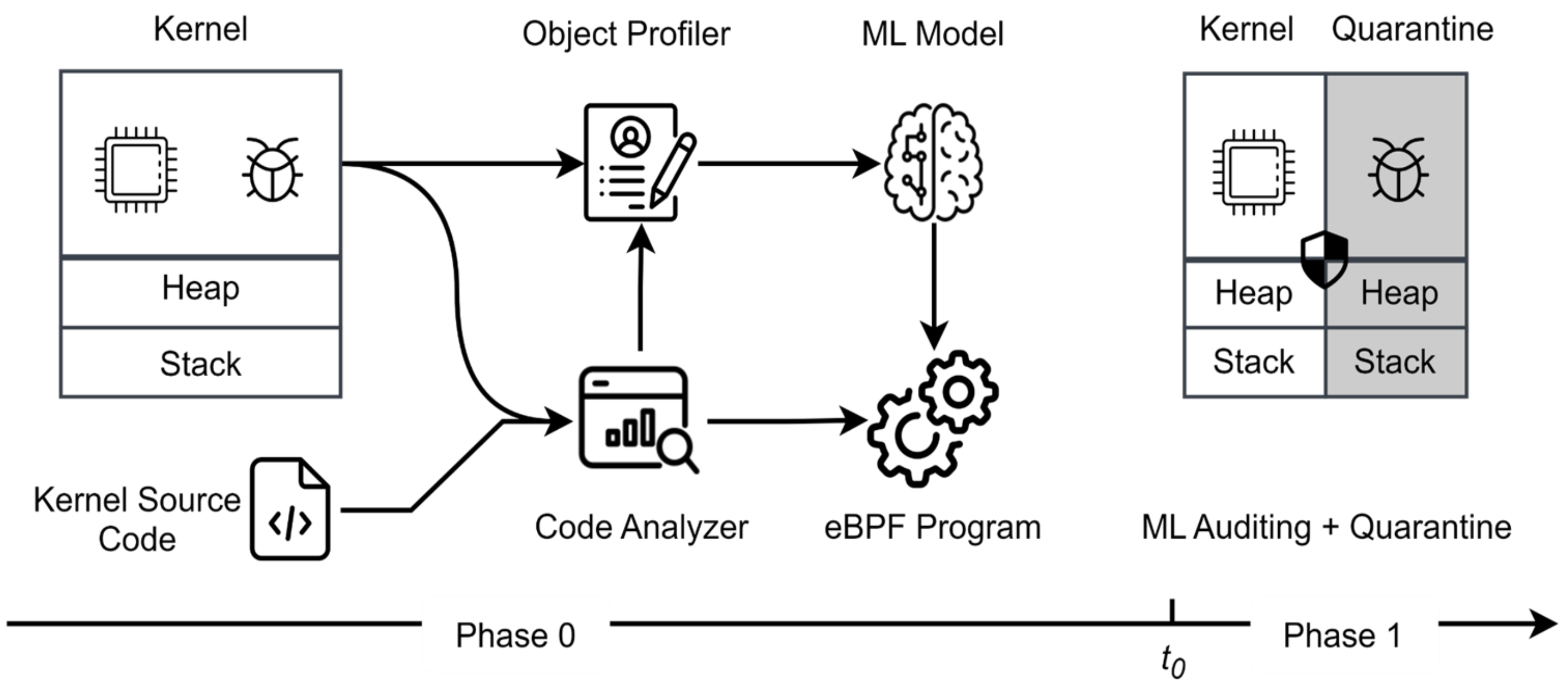
- Add extra eBPF helper functions for O2Q's functionality:
  - **bpf_set_regs()**: set register values, for switching stacks.
  - **bpf_create_slab_cache()**: creates private slab caches for the need of quarantine zone.
  - **bpf_cache_alloc() / bpf_cache_free()**: allocates from and frees to private caches.
- For better interaction with quarantine zone data:
  - **bpf_get_slab_*() / bpf_get_vm_struct()**: get the description of the slab and vmalloc directly, without traversing slab pages or vm_struct rb-trees.





O2Q Workflow Summary

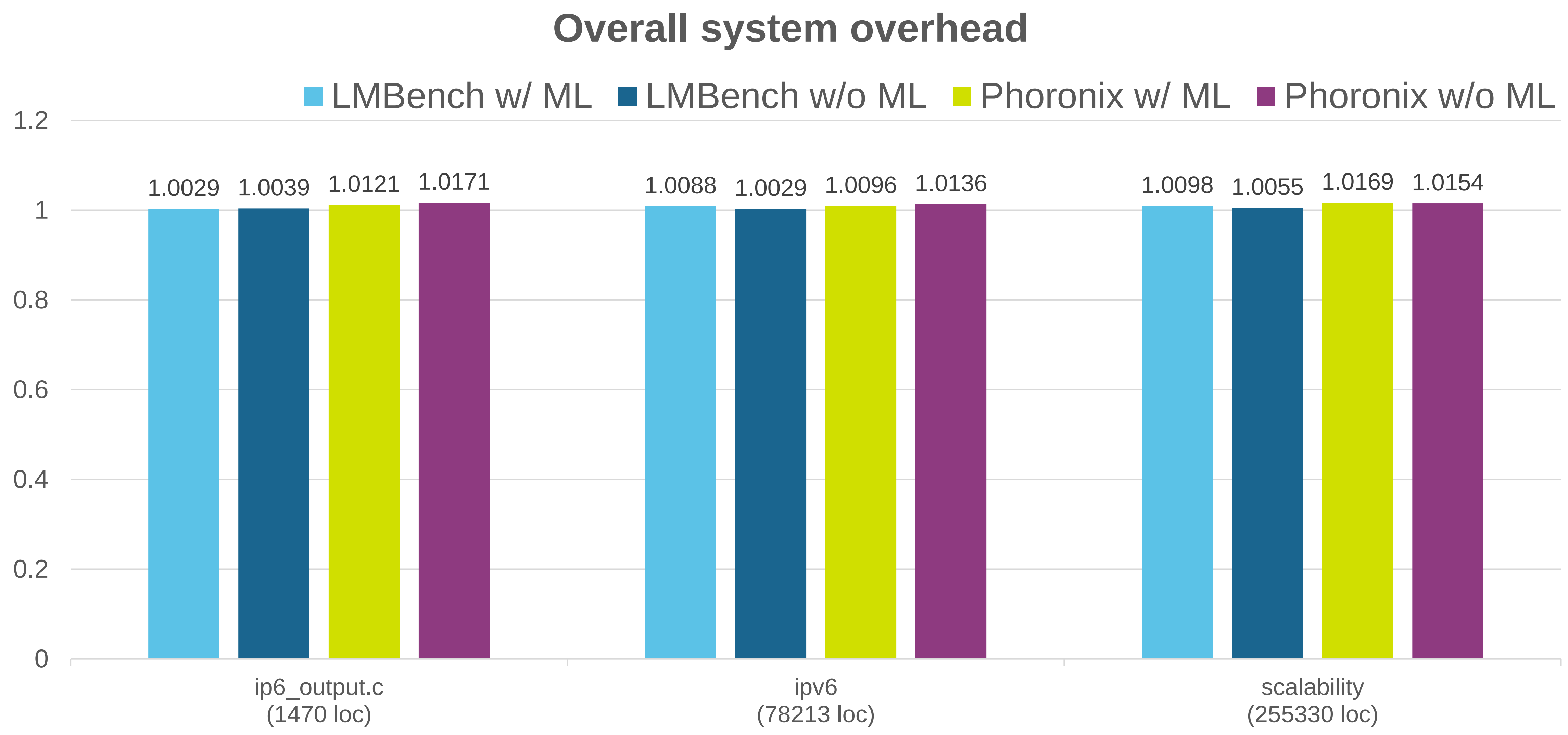
1. Object Profiler to collect object's type and content at its release site.
2. Train ML Model offline based on collected object's content, type, stacktrace and if belonging to quarantine.
3. Code Analyzer to identify code entries which need instrumentation for auditing.
4. eBPF programs to do quarantine and type examination with trained ML Model at runtime.



Agenda

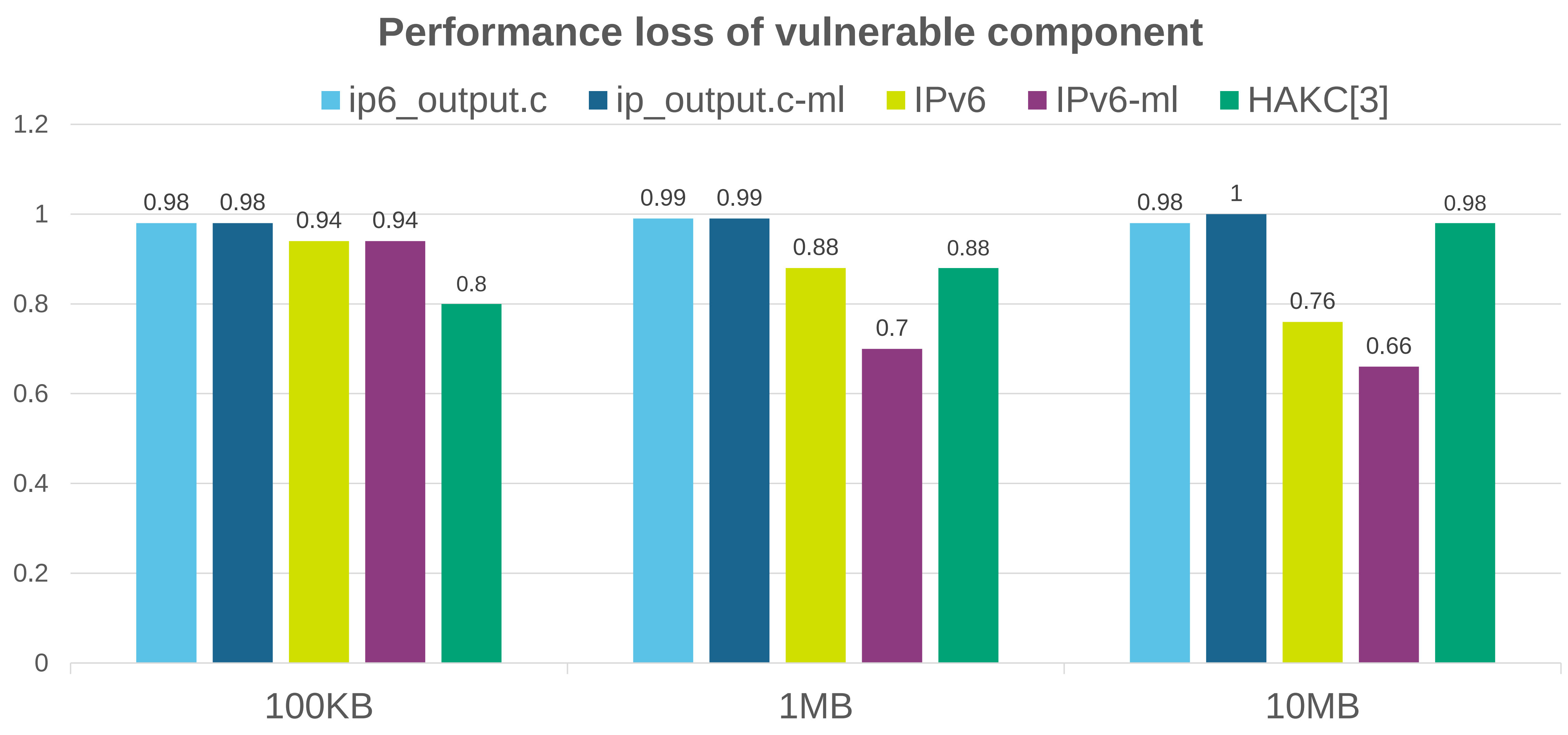
- Motivation
- Challenges & Design Overview
- Example Workflow by CVE-2022-0995 & Video Demo
- Technical Details
- **Evaluation**

Evaluation



Overall system overhead
LMBench w/ ML
LMBench w/o ML
Phoronix w/ ML
Phoronix w/o ML
1.2
1
0.8
0.6
0.4
0.2
0
1.0029
1.0039
1.0121
1.0171
1.0088
1.0029
1.0096
1.0136
1.0098
1.0055
1.0169
1.0154
ip6_output.c (1470 loc)
ipv6 (78213 loc)
scalability (255330 loc)

Evaluation



[3] HAKC: Preventing Kernel Hacks with HAKCs, NDSS'22

Evaluation

	Per Type		Per Component	
	Accuracy	Macro F1	Accuracy	Macro F1
IPV6				
Decision Tree	96.88 ± 0.65	75.56 ± 1.84	99.99 ±0.02	99.98 ± 0.03
Random Forest	96.91 ± 0.63	78.81 ± 0.73	100 ± 0.01	99.99 ± 0.01
Neural Network	89.63 ± 1.29	38.76± 2.70	99.99 ± 0.01	99.99 ±0.01
Sched				
Decision Tree	80.48 ± 0.76	71.04 ± 1.77	99.93 ± 0.14	97.74 ± 4.22
Random Forest	80.61 ± 0.69	76.28 ± 0.49	100 ± 0	99.99 ± 0.01
Neural Network	65.98 ± 6.91	39.18 ± 1.48	99.66±0.03	89.47±1.20
Netfilter				
Decision Tree	89.47 ± 0.23	78.17 ± 4.88	99.92 ± 0.07	99.51 ± 0.46
Random Forest	89.54 ± 0.15	81.87 ± 1.86	99.96 ± 0.05	99.77 ± 0.29
Neural Network	72.9 ± 2.23	37.98 ± 2.83	97.16 ±0.17	74 ± 2.56

performance of ML auditing

black hat USA 2024

Evaluation

	Per Type		Per Component	
	Accuracy	Macro F1	Accuracy	Macro F1
IPV6				
Decision Tree	96.88 ± 0.65	75.56 ± 1.84	99.99 ±0.02	99.98 ± 0.03
Random Forest	96.91 ± 0.63	78.81 ± 0.73	100 ± 0.01	99.99 ± 0.01
Neural Network	89.63 ± 1.29	38.76± 2.70	99.99 ± 0.01	99.99 ±0.01
Sched				
Decision Tree	80.48 ± 0.76	71.04 ± 1.77	99.93 ± 0.14	97.74 ± 4.22
Random Forest	80.61 ± 0.69	76.28 ± 0.49	100 ± 0	99.99 ± 0.01
Neural Network	65.98 ± 6.91	39.18 ± 1.48	99.66±0.03	89.47±1.20
Netfilter				
Decision Tree	89.47 ± 0.23	78.17 ± 4.88	99.92 ± 0.07	99.51 ± 0.46
Random Forest	89.54 ± 0.15	81.87 ± 1.86	99.96 ± 0.05	99.77 ± 0.29
Neural Network	72.9 ± 2.23	37.98 ± 2.83	97.16 ±0.17	74 ± 2.56

performance of ML auditing

Evaluation

	Per Type		Per Component	
	Accuracy	Macro F1	Accuracy	Macro F1
IPV6				
Decision Tree	96.88 ± 0.65	75.56 ± 1.84	99.99 ±0.02	99.98 ± 0.03
Random Forest	96.91 ± 0.63	78.81 ± 0.73	100 ± 0.01	99.99 ± 0.01
Neural Network	89.63 ± 1.29	38.76± 2.70	99.99 ± 0.01	99.99 ±0.01
Sched				
Decision Tree	80.48 ± 0.76	71.04 ± 1.77	99.93 ± 0.14	97.74 ± 4.22
Random Forest	80.61 ± 0.69	76.28 ± 0.49	100 ± 0	99.99 ± 0.01
Neural Network	65.98 ± 6.91	39.18 ± 1.48	99.66±0.03	89.47±1.20
Netfilter				
Decision Tree	89.47 ± 0.23	78.17 ± 4.88	99.92 ± 0.07	99.51 ± 0.46
Random Forest	89.54 ± 0.15	81.87 ± 1.86	99.96 ± 0.05	99.77 ± 0.29
Neural Network	72.9 ± 2.23	37.98 ± 2.83	97.16 ±0.17	74 ± 2.56

performance of ML auditing

Evaluation

	Per Type		Per Component	
	Accuracy	Macro F1	Accuracy	Macro F1
IPV6				
Decision Tree	96.88 ± 0.65	75.56 ± 1.84	99.99 ±0.02	99.98 ± 0.03
Random Forest	96.91 ± 0.63	78.81 ± 0.73	100 ± 0.01	99.99 ± 0.01
Neural Network	89.63 ± 1.29	38.76± 2.70	99.99 ± 0.01	99.99 ±0.01
Sched				
Decision Tree	80.48 ± 0.76	71.04 ± 1.77	99.93 ± 0.14	97.74 ± 4.22
Random Forest	80.61 ± 0.69	76.28 ± 0.49	100 ± 0	99.99 ± 0.01
Neural Network	65.98 ± 6.91	39.18 ± 1.48	99.66±0.03	89.47±1.20
Netfilter				
Decision Tree	89.47 ± 0.23	78.17 ± 4.88	99.92 ± 0.07	99.51 ± 0.46
Random Forest	89.54 ± 0.15	81.87 ± 1.86	99.96 ± 0.05	99.77 ± 0.29
Neural Network	72.9 ± 2.23	37.98 ± 2.83	97.16 ±0.17	74 ± 2.56

performance of ML auditing

Evaluation

	Per Type		Per Component	
	Accuracy	Macro F1	Accuracy	Macro F1
IPV6				
Decision Tree	96.88 ± 0.65	75.56 ± 1.84	99.99 ±0.02	99.98 ± 0.03
Random Forest	96.91 ± 0.63	78.81 ± 0.73	100 ± 0.01	99.99 ± 0.01
Neural Network	89.63 ± 1.29	38.76± 2.70	99.99 ± 0.01	99.99 ±0.01
Sched				
Decision Tree	80.48 ± 0.76	71.04 ± 1.77	99.93 ± 0.14	97.74 ± 4.22
Random Forest	80.61 ± 0.69	76.28 ± 0.49	100 ± 0	99.99 ± 0.01
Neural Network	65.98 ± 6.91	39.18 ± 1.48	99.66±0.03	89.47±1.20
Netfilter				
Decision Tree	89.47 ± 0.23	78.17 ± 4.88	99.92 ± 0.07	99.51 ± 0.46
Random Fore[illegible]	[illegible]	[illegible]	[illegible]	99.77 ± 0.29

Neural Network is not good enough

Random Forest is too heavy to be embedded via eBPF

performance of ML auditing

Evaluation

	Accuracy	Macro F1	Accuracy	Macro F1
Feature Length				
32	88.40 ± 0.42	73.97 ± 3.83	98.75 ± 0.41	91.91 ± 2.32
64	89.15 ± 0.33	77.24 ± 4.21	99.91 ± 0.07	99.47 ± 0.45
128	89.18 ± 0.29	77.44 ± 4.33	99.85 ± 0.1	99.46 ± 0.64
256	89.26 ± 0.29	77.34 ± 5.06	99.92 ± 0.08	99.51 ± 0.49
1024	89.47 ± 0.23	78.17 ± 4.88	99.92 ± 0.07	99.51 ± 0.46
Max Depth				
3	61.18 ± 2.45	1.72 ± 0.19	97.47 ± 0.4	79.34 ± 3.03
7	76.59 ± 2.38	8.48 ± 0.58	99.44 ± 0.21	96.44 ± 1.32
10	83.54 ± 2.19	21.06 ± 2.19	99.65 ± 0.14	97.78 ± 0.86
14	89.47 ± 0.23	78.17 ± 4.88	99.92 ± 0.07	99.51 ± 0.46

Performance of tuning decision tree feature length and depth

Evaluation

	Accuracy	Macro F1	Accuracy	Macro F1
Feature Length				
32	88.40 ± 0.42	73.97 ± 3.83	98.75 ± 0.41	91.91 ± 2.32
64	89.15 ± 0.33	77.24 ± 4.21	99.91 ± 0.07	99.47 ± 0.45
128	89.18 ± 0.29	77.44 ± 4.33	99.85 ± 0.1	99.46 ± 0.64
256	89.26 ± 0.29	77.34 ± 5.06	99.92 ± 0.08	99.51 ± 0.49
1024	89.47 ± 0.23	78.17 ± 4.88	99.92 ± 0.07	99.51 ± 0.46
Max Depth				
3	61.18 ± 2.45	1.72 ± 0.19	97.47 ± 0.4	79.34 ± 3.03
7	76.59 ± 2.38	8.48 ± 0.58	99.44 ± 0.21	96.44 ± 1.32
10	83.54 ± 2.19	21.06 ± 2.19	99.65 ± 0.14	97.78 ± 0.86
14	89.47 ± 0.23	78.17 ± 4.88	99.92 ± 0.07	99.51 ± 0.46

Performance of tuning decision tree feature length and depth

Evaluation

	Accuracy	Macro F1	Accuracy	Macro F1
Feature Length				
32	88.40 ± 0.42	73.97 ± 3.83	98.75 ± 0.41	91.91 ± 2.32
64	89.15 ± 0.33	77.24 ± 4.21	99.91 ± 0.07	99.47 ± 0.45
128	89.18 ± 0.29	77.44 ± 4.33	99.85 ± 0.1	99.46 ± 0.64
256	89.26 ± 0.29	77.34 ± 5.06	99.92 ± 0.08	99.51 ± 0.49
1024	89.47 ± 0.23	78.17 ± 4.88	99.92 ± 0.07	99.51 ± 0.46
Max Depth				
3	61.18 ± 2.45	1.72 ± 0.19	97.47 ± 0.4	79.34 ± 3.03
7	76.59 ± 2.38	8.48 ± 0.58	99.44 ± 0.21	96.44 ± 1.32
10	83.54 ± 2.19	21.06 ± 2.19	99.65 ± 0.14	97.78 ± 0.86
14	89.47 ± 0.23	78.17 ± 4.88	99.92 ± 0.07	99.51 ± 0.46

Performance of tuning decision tree feature length and depth

Takeaway

- Our work revealed the legacy object problem, which is critical to protect the kernel on-the-fly before patches are available.
- We demonstrated how embedding machine learning into the kernel can effectively solve the legacy object problem.
- Limitation: ML model accuracy is not 100%, only sufficing as a temporary remediation before patches are available.
- Future work:
  - Mature the prototype implementation and solution to corner cases in ML model. Expecting collaboration.
  - Reduce overhead using PKS like hardware feature.

Qinrun Dai (@2st___)

Looking for

2025 summer internship!