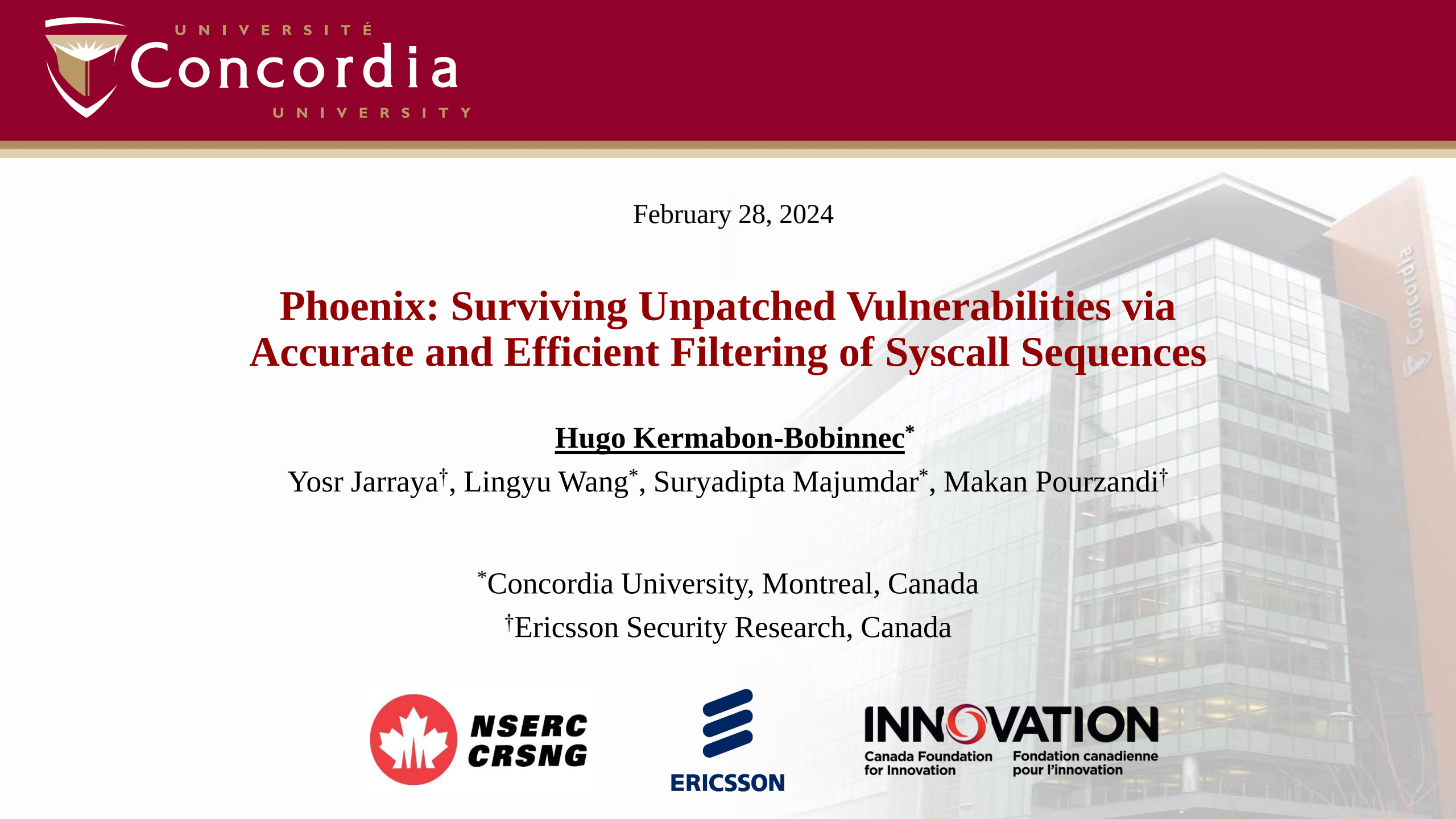


February 28, 2024

Phoenix: Surviving Unpatched Vulnerabilities via Accurate and Efficient Filtering of Syscall Sequences

Hugo Kermabon-Bobinnec[*]

Yosr Jarraya[†], Lingyu Wang[*], Suryadipta Majumdar[*], Makan Pourzandi[†]

[*]Concordia University, Montreal, Canada

[†]Ericsson Security Research, Canada

NSERC CRSNG

ERICSSON

INNOVATION Canada Foundation for Innovation Fondation canadienne pour l'innovation

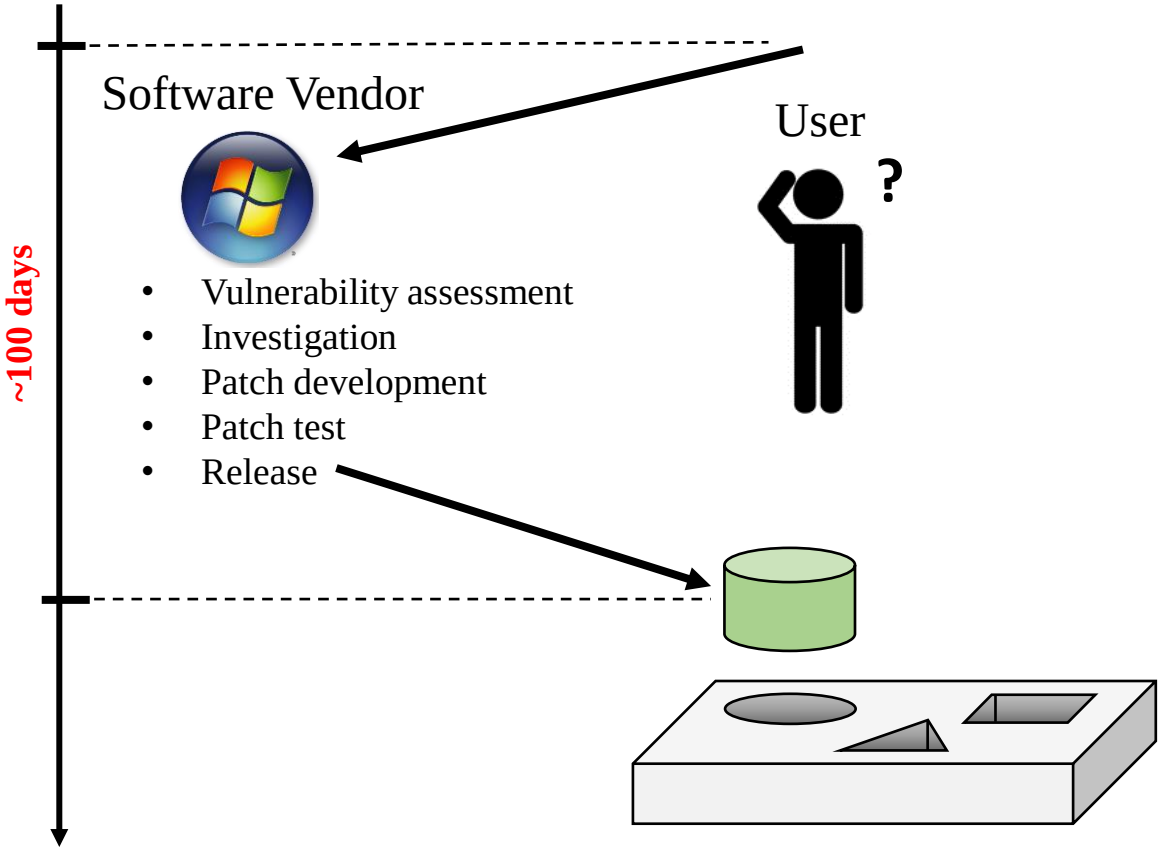
Outline

- Intro
  - Motivation
  - Related Work
- Methodology
  - Key Ideas & Overview
  - Malicious Sequence Identification
  - Dynamic Runtime Protection
- Implementation
- Experimental Results
  - Security
  - Performance
  - Provenance Analysis
- Conclusion

Outline

- **Intro**
  - **Motivation**
  - **Related Work**
- Methodology
  - Key Ideas & Overview
  - Malicious Sequence Identification
  - Dynamic Runtime Protection
- Implementation
- Experimental Results
  - Security
  - Performance
  - Provenance Analysis
- Conclusion

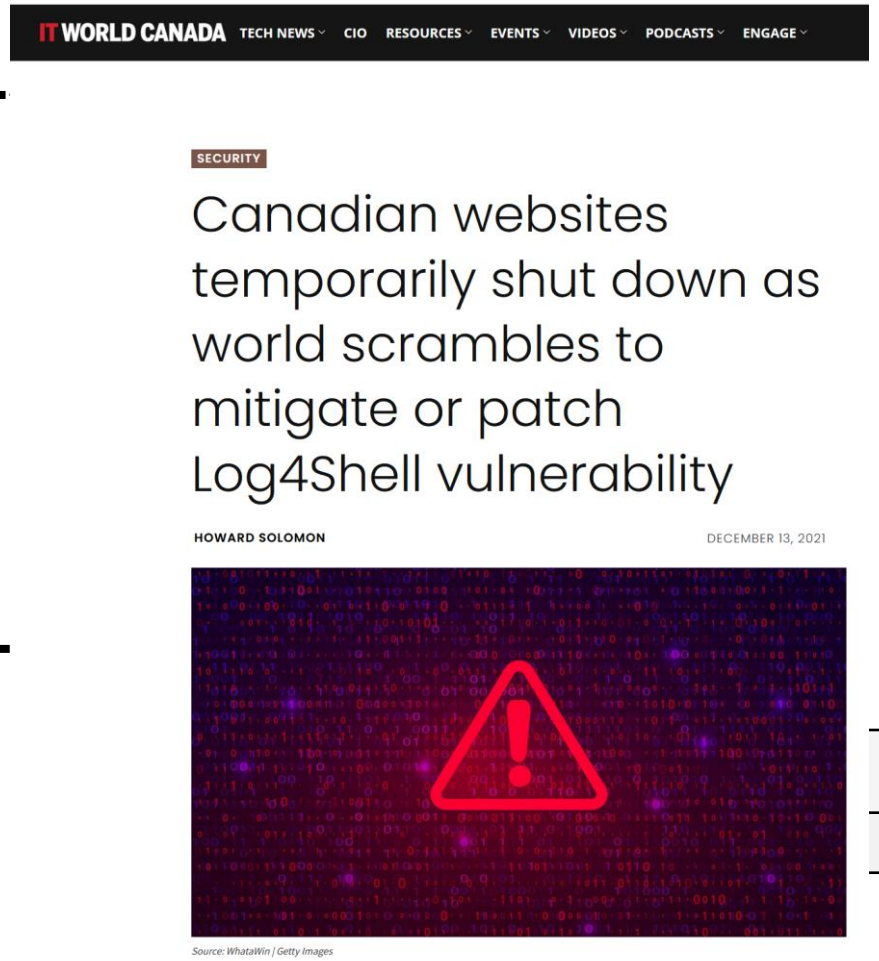
Current approach to security patching

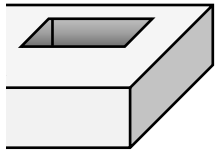


Motivation

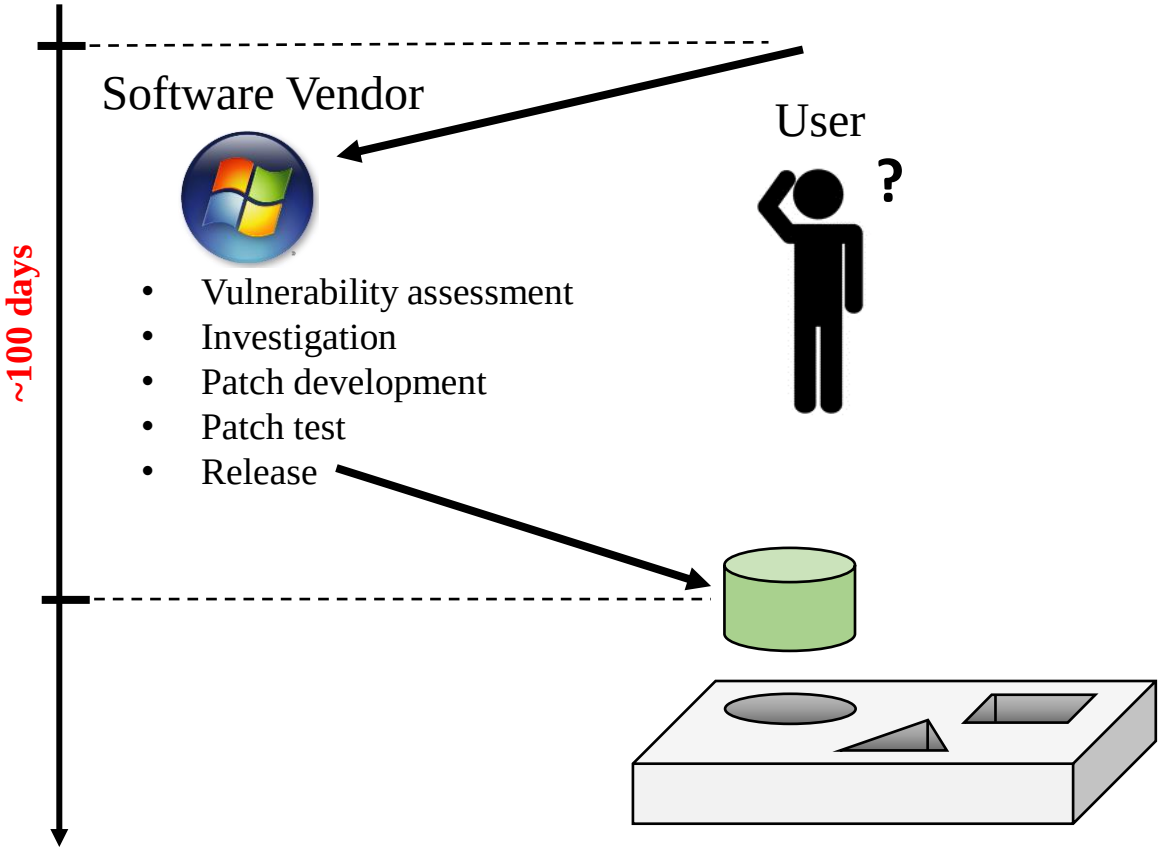
Current approach to security patching

~100 days

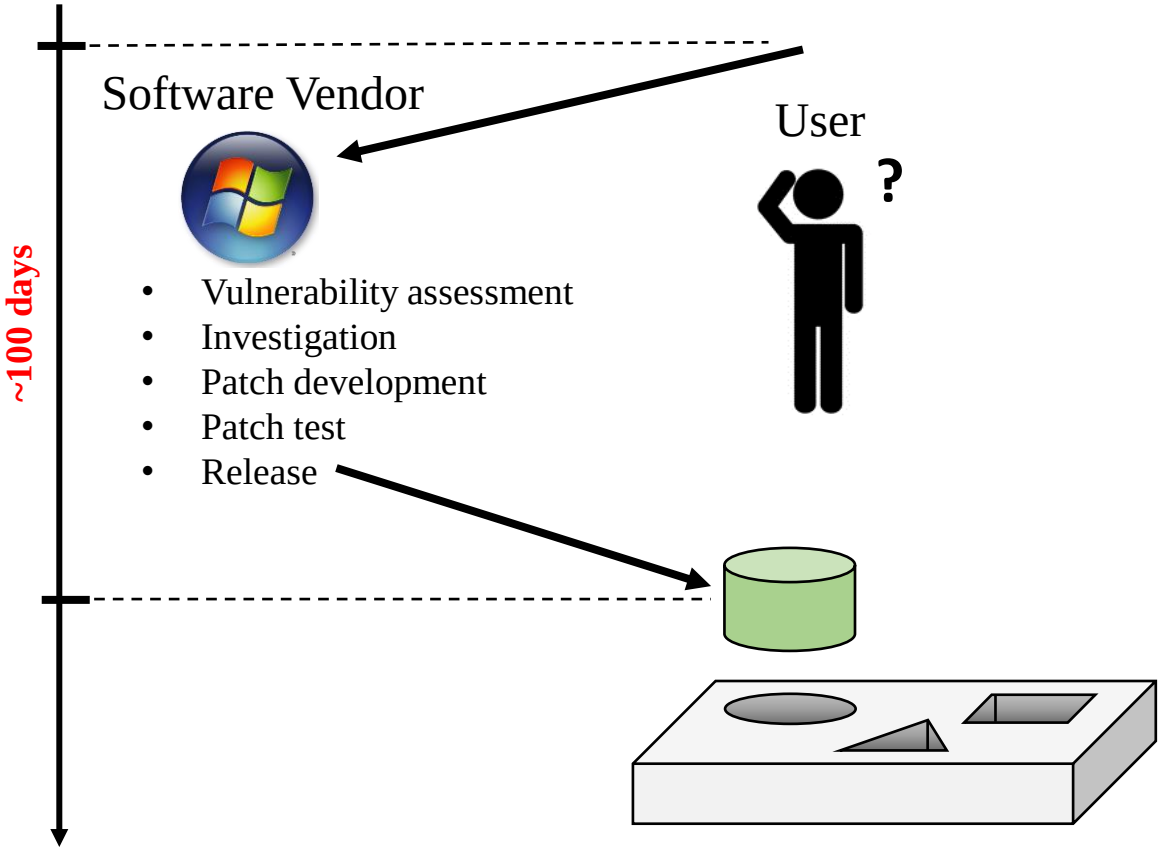




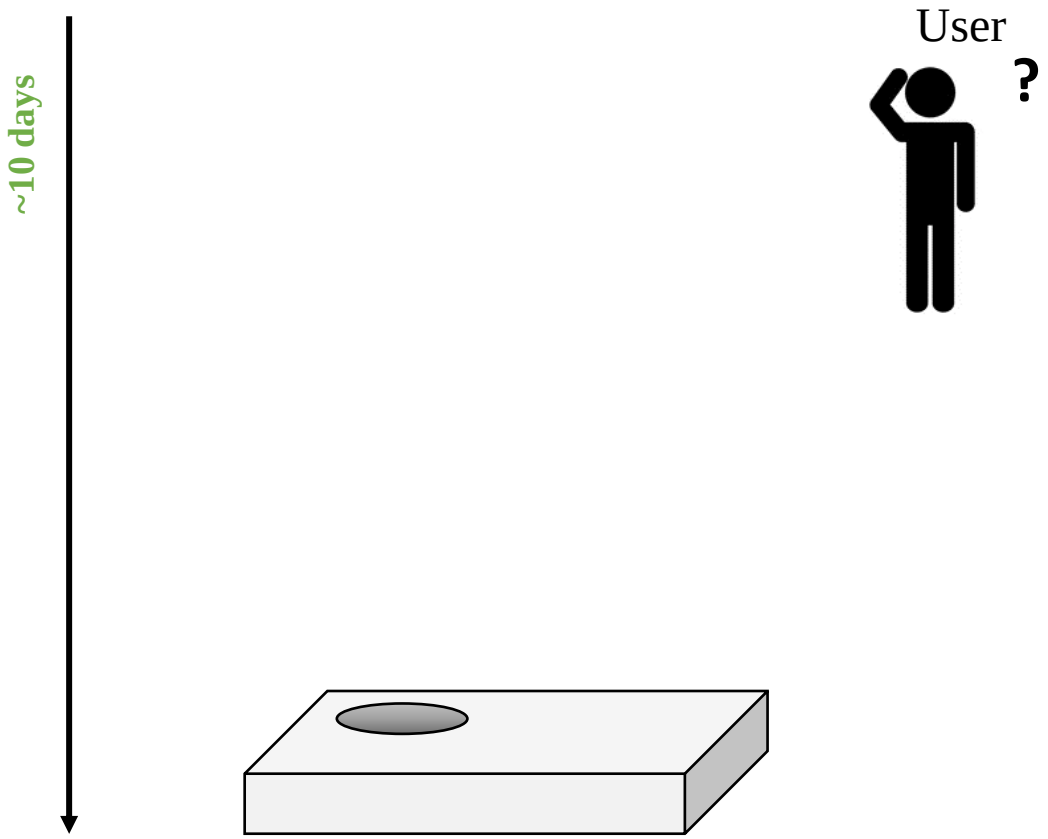
Current approach to security patching



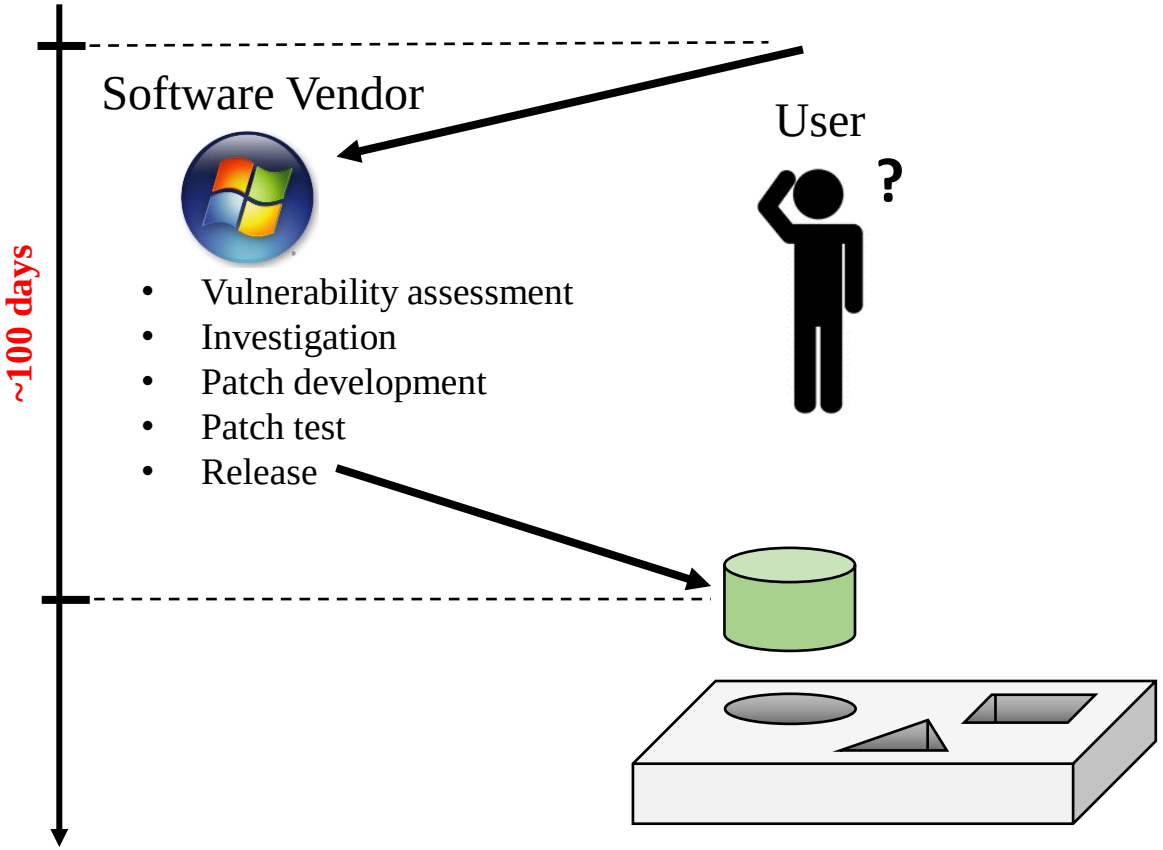
Current approach to security patching



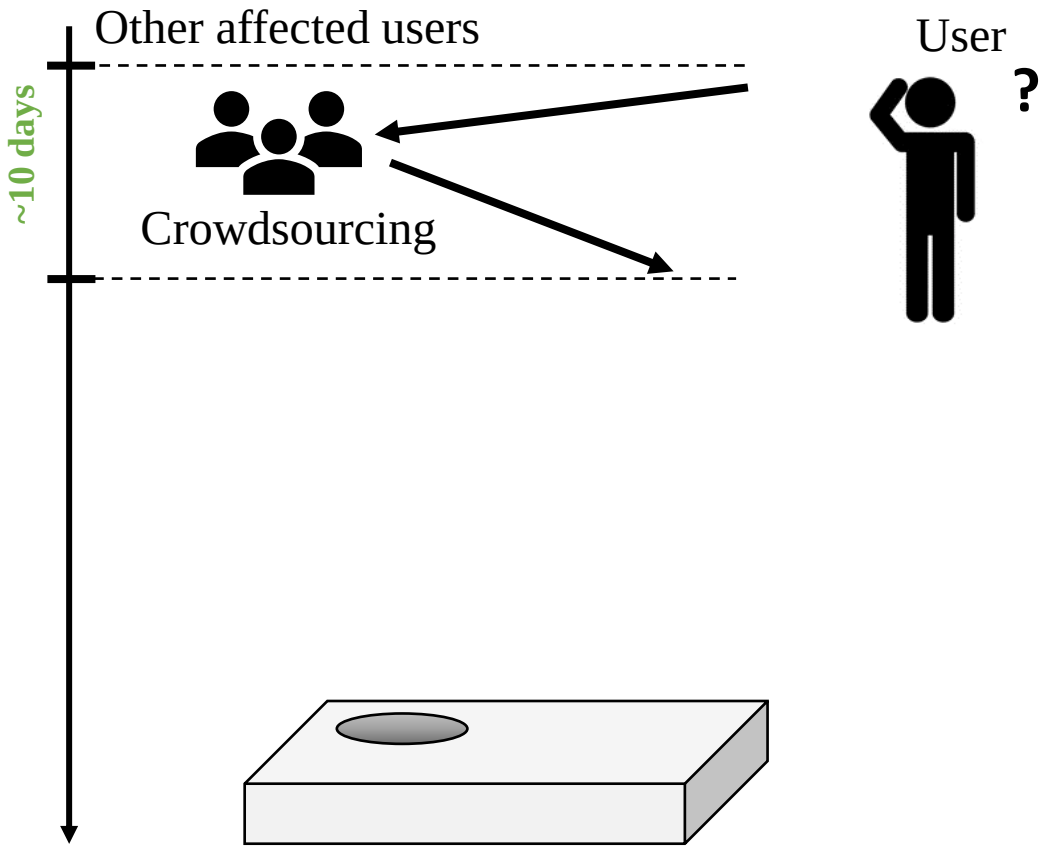
Our approach to security patching



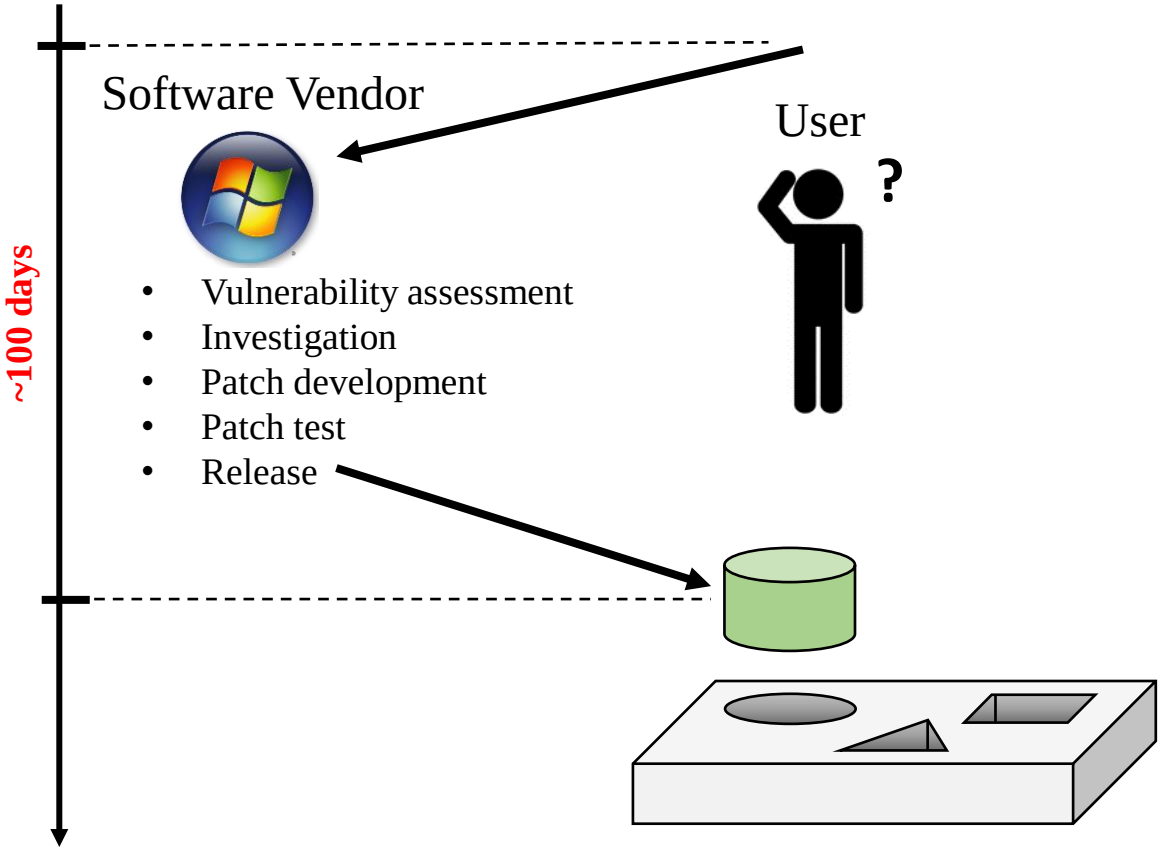
Current approach to security patching



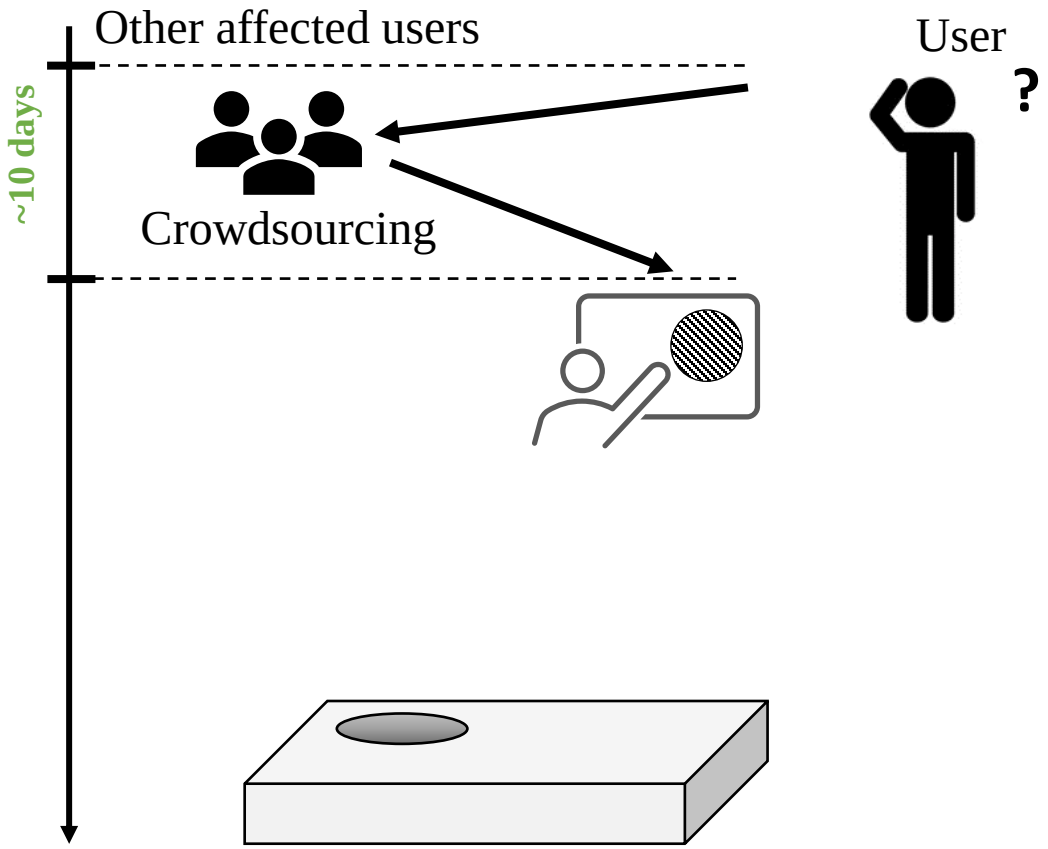
Our approach to security patching



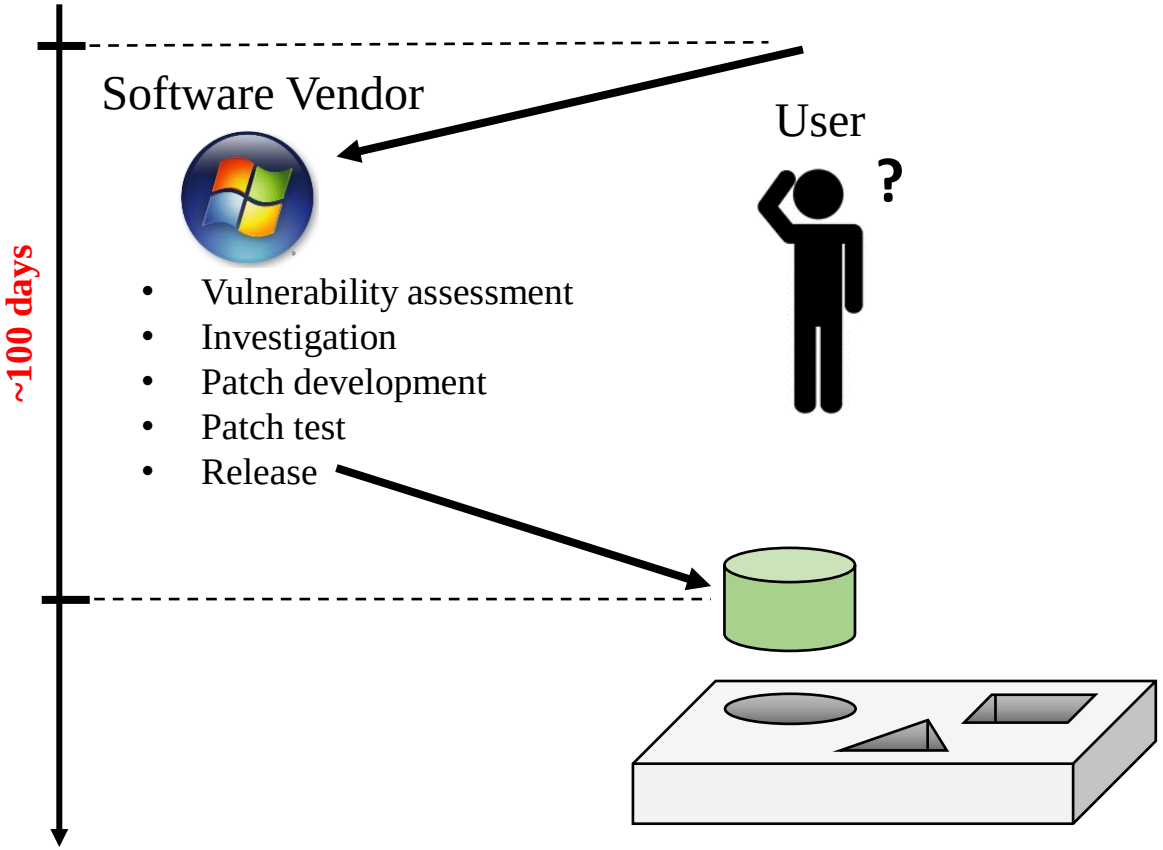
Current approach to security patching



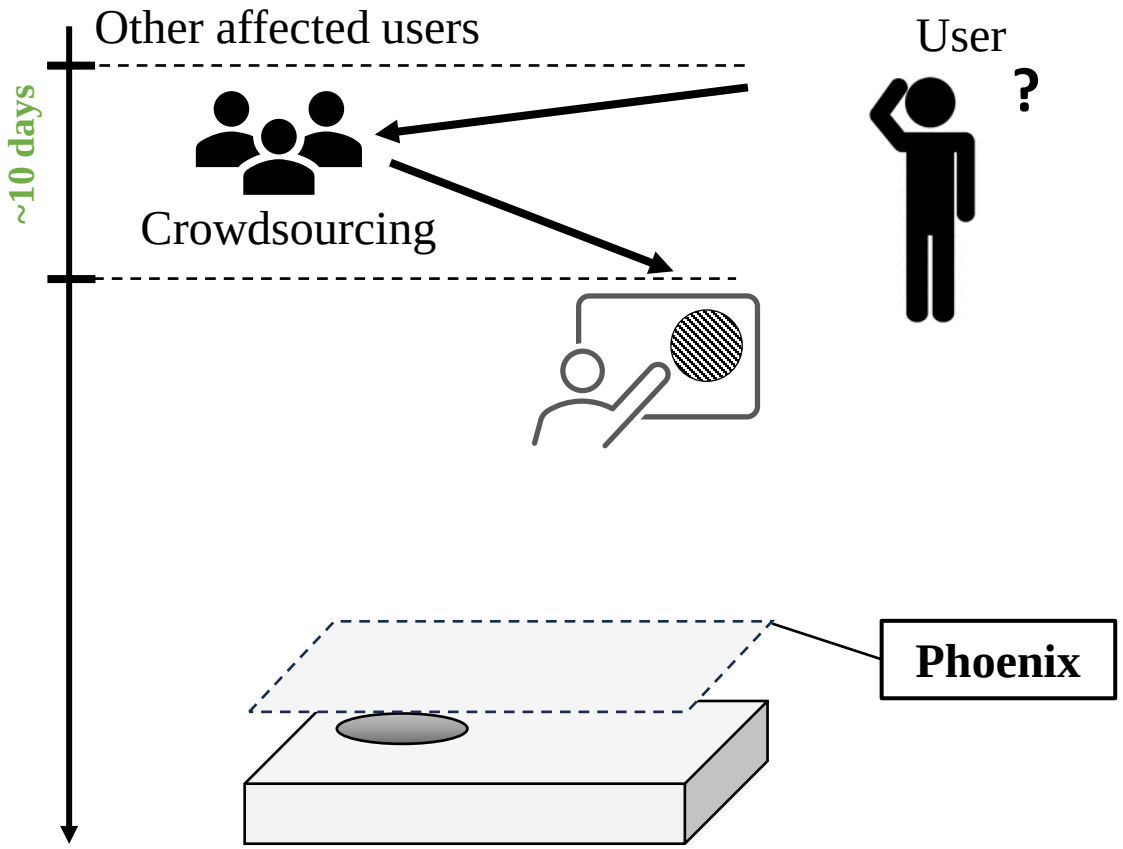
Our approach to security patching



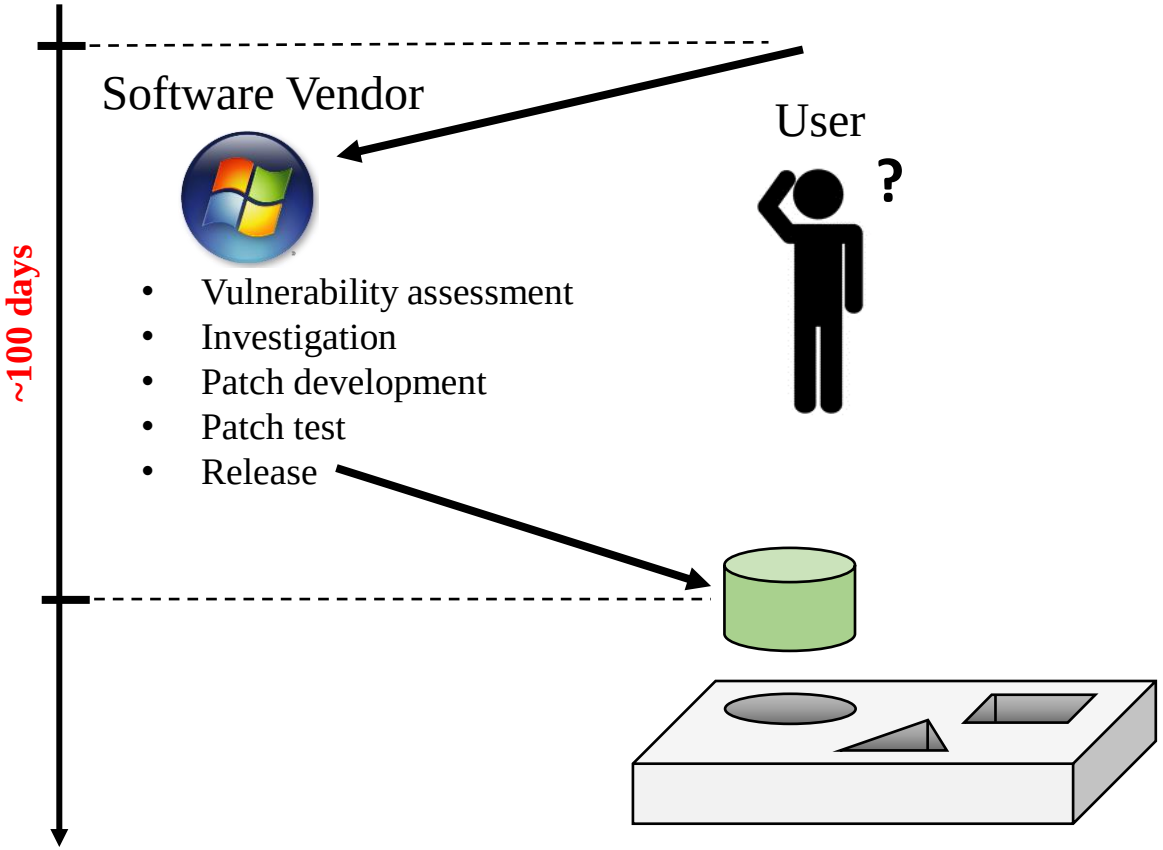
Current approach to security patching



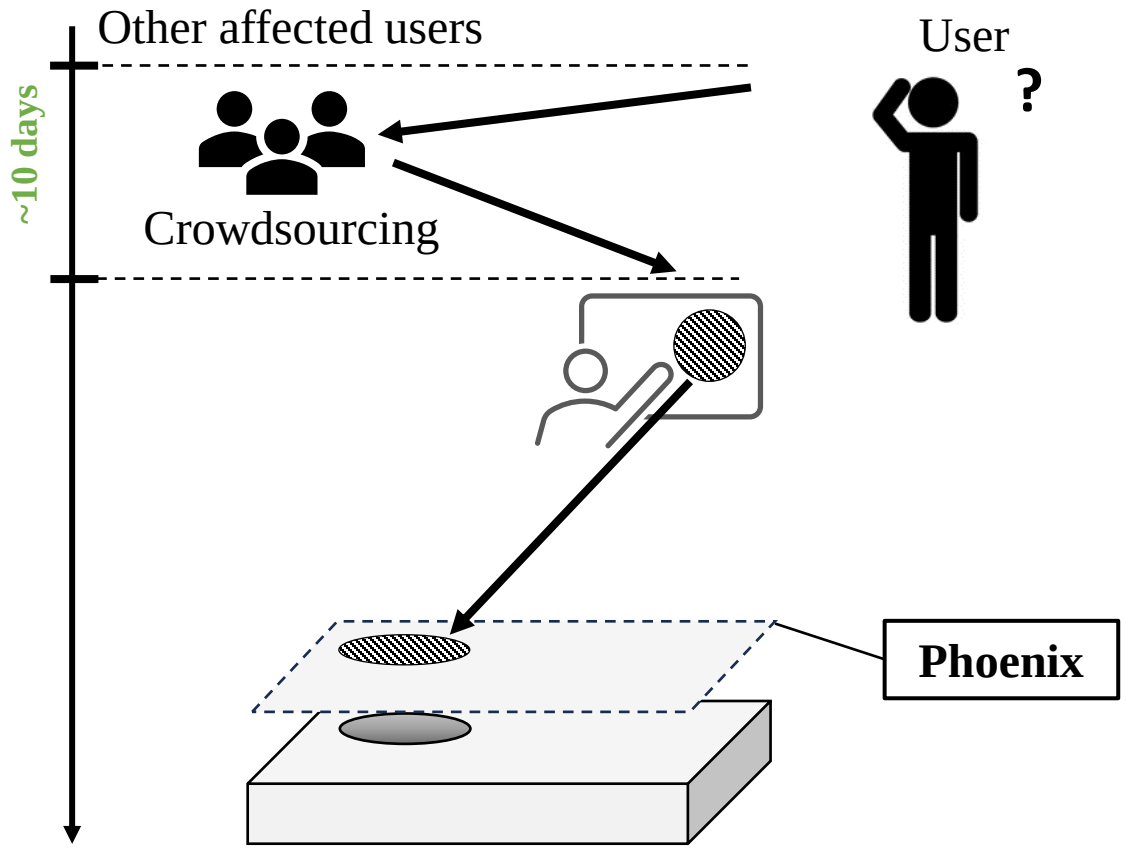
Our approach to security patching



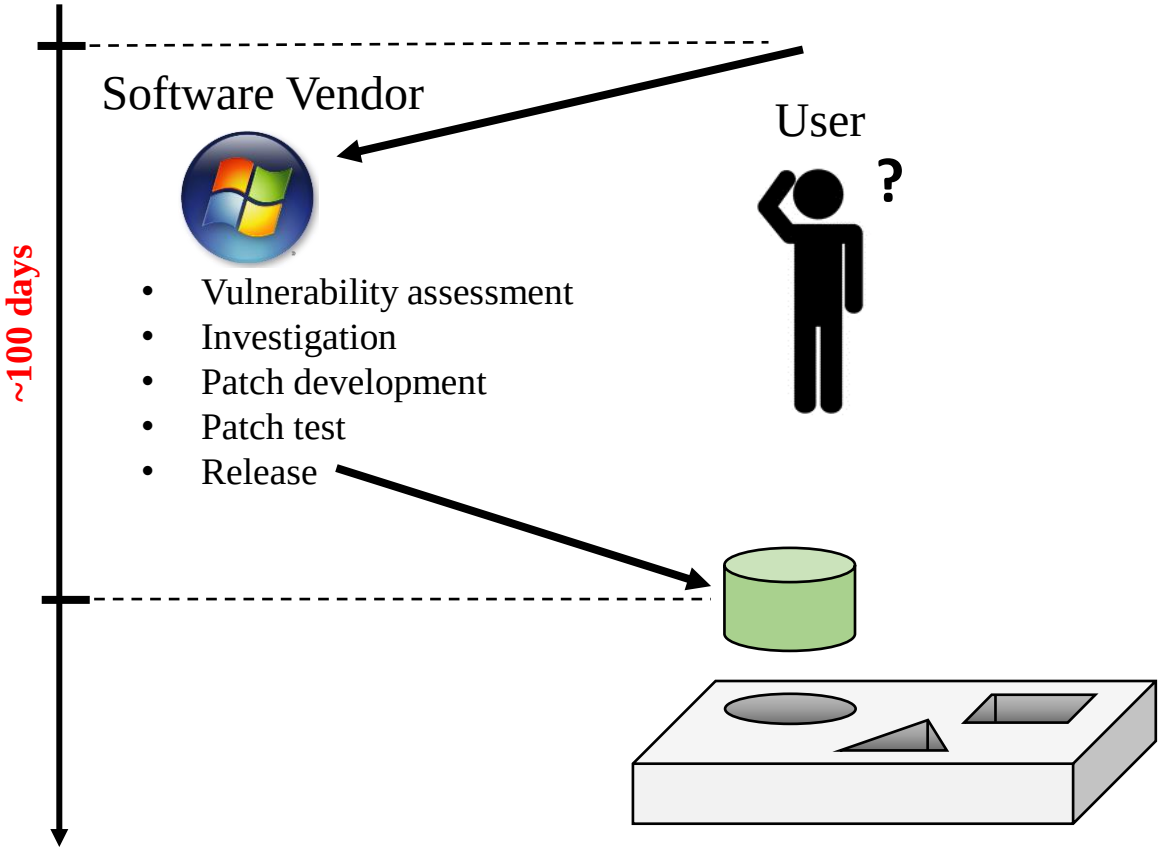
Current approach to security patching



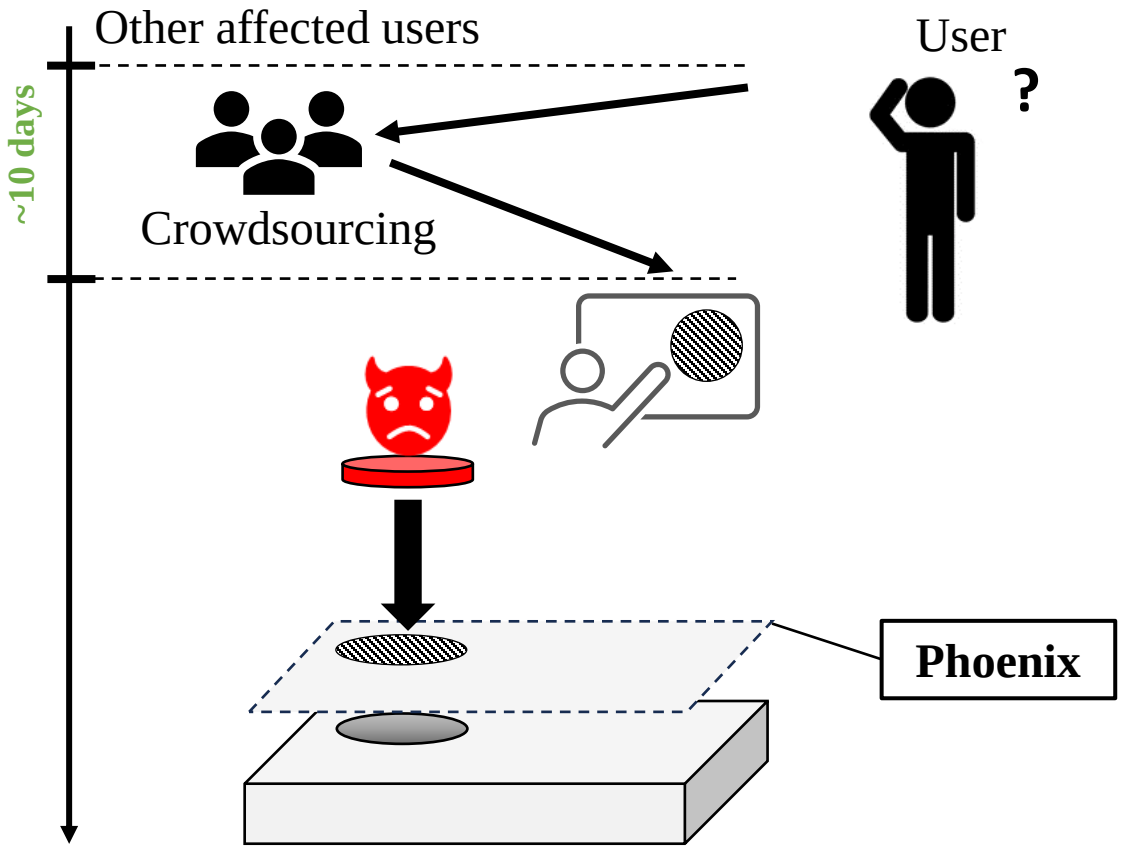
Our approach to security patching



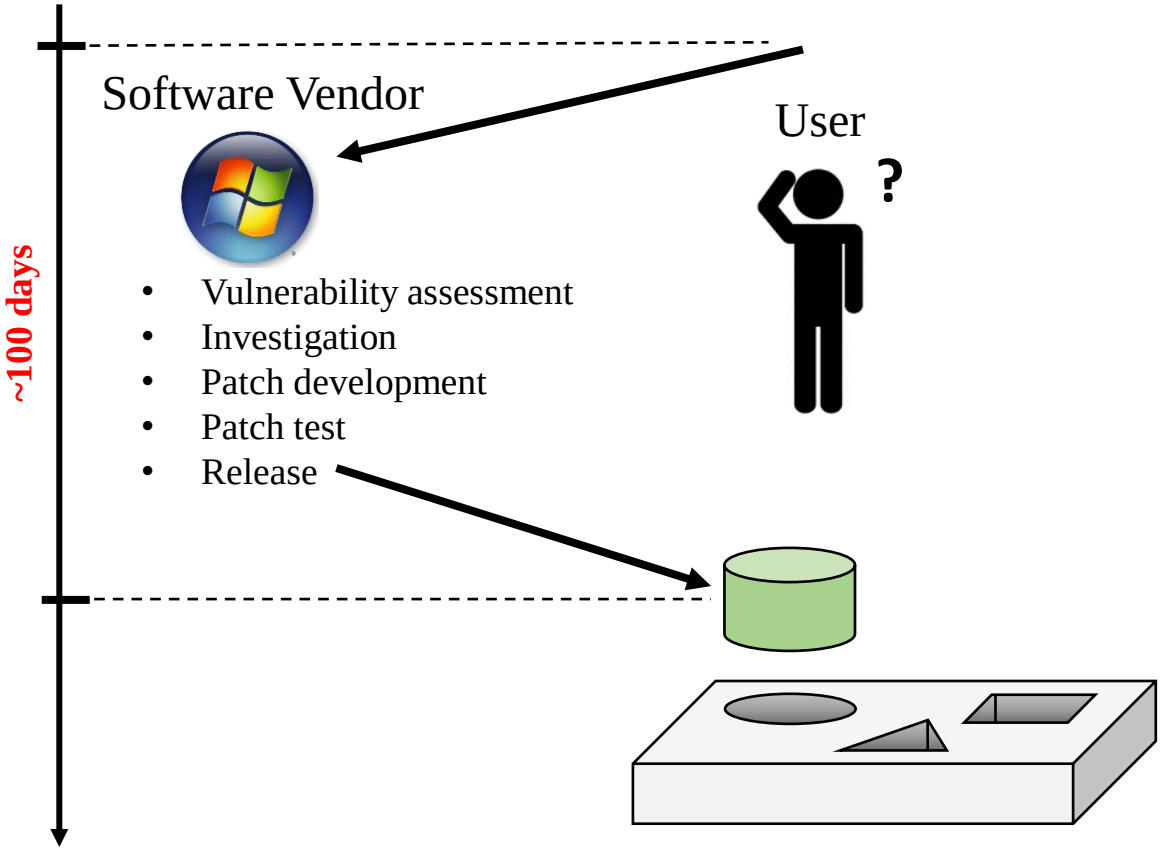
Current approach to security patching



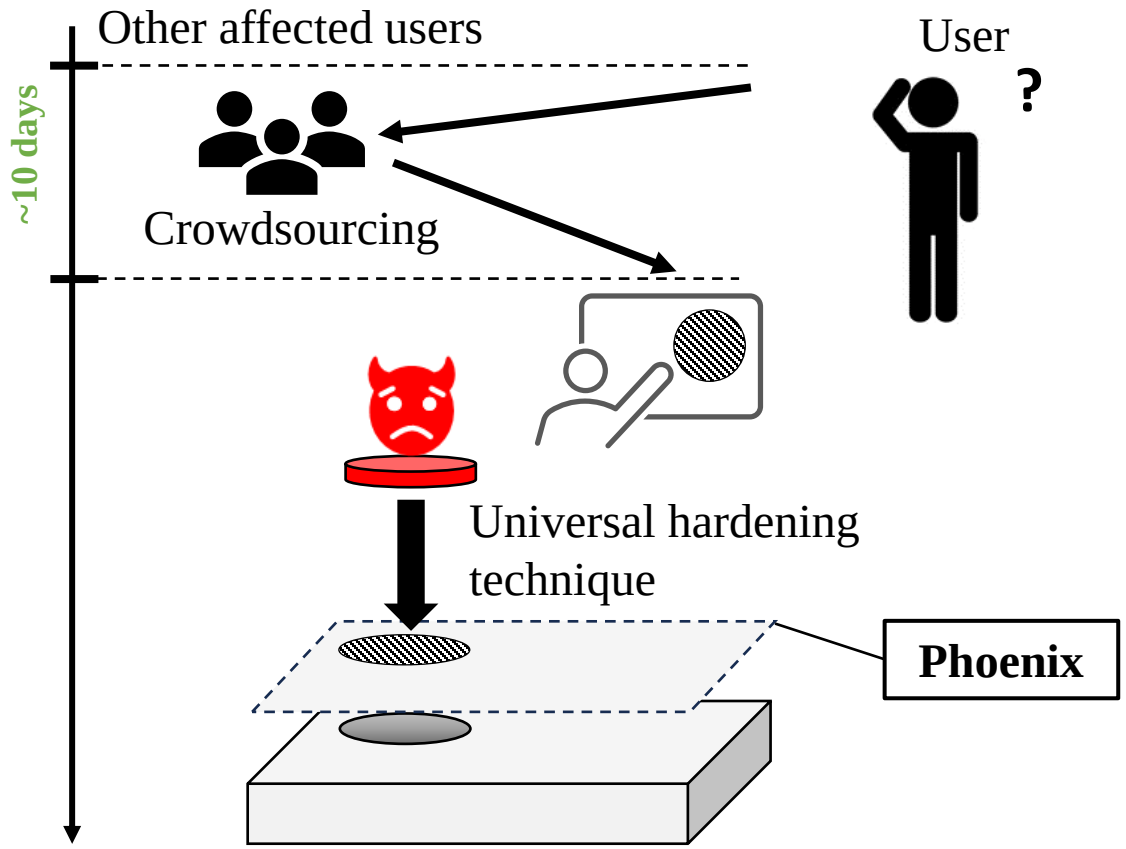
Our approach to security patching



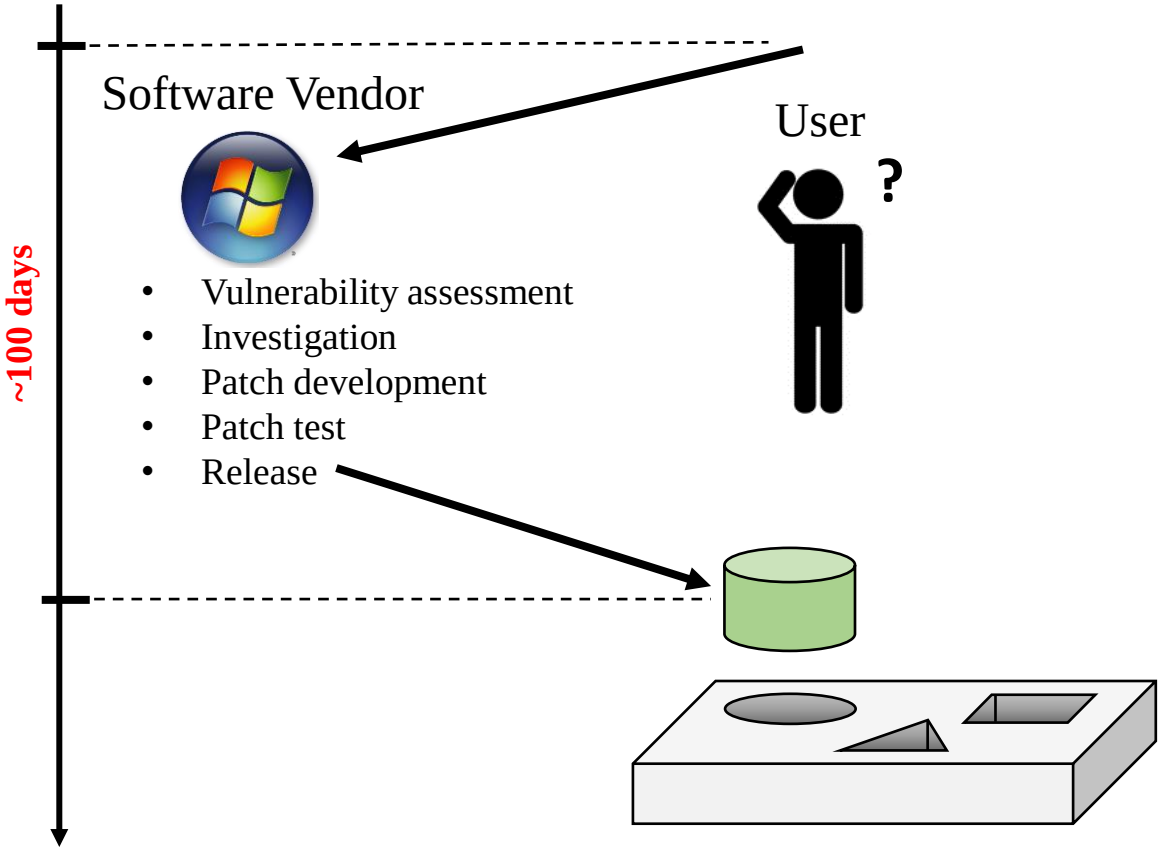
Current approach to security patching



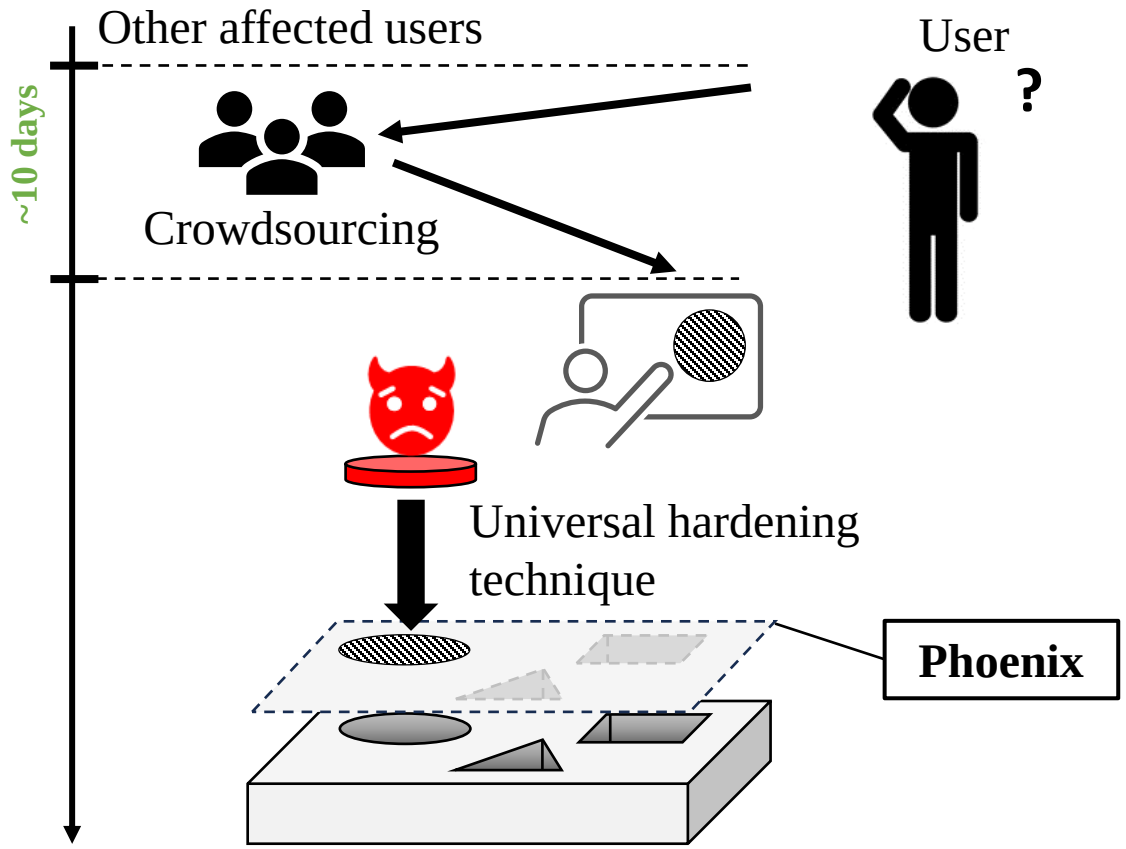
Our approach to security patching



Current approach to security patching



Our approach to security patching



Motivation

Current approach to security patching

Software Vendor

- Vulnerability assessmen
- Investigation
- Patch development
- Patch test
- Release

~100 days

Our approach to security patching

User

?

hardening

Phoenix

Forensic Analysis and Attack Detection

Existing works

CLARION
Provenance graph collection for containers

Atlas/RapSheet
Provenance graph analysis

Falco
Attack detection and alert

Limitations

- Manual inspection and enforcement

Forensic Analysis and Attack Detection

Existing works

CLARION
Provenance graph collection for containers

Atlas/RapSheet
Provenance graph analysis

Falco
Attack detection and alert

Limitations
- Manual inspection and enforcement

Container Hardening

Existing works

Related Work

Forensic Analysis and Attack Detection

Existing works

CLARION
Provenance graph collection for containers

Atlas/RapSheet
Provenance graph analysis

Falco
Attack detection and alert

Limitations
- Manual inspection and enforcement

Container Hardening

Existing works

Confine/SysFilter/C2C
Static system call filtering

SPEAKER
Phase-split filtering

Seccomp
System call filtering

Ptrace
Process observation

Related Work

Forensic Analysis and Attack Detection

Existing works

CLARION
Provenance graph collection for containers

Atlas/RapSheet
Provenance graph analysis

Falco
Attack detection and alert

Limitations
- Manual inspection and enforcement

Container Hardening

Existing works

Confine/SysFilter/C2C
Static system call filtering

SPEAKER
Phase-split filtering

Seccomp
System call filtering

Ptrace
Process observation

Limitations
- Only works for single system calls the app does not need
- Prevent unused syscalls and hope for the best

Related Work

Forensic Analysis and Attack Detection

Existing works

CLARION
Provenance graph collection for containers

Atlas/RapSheet
Provenance graph analysis

Falco
Attack detection and alert

Limitations
- Manual inspection and enforcement

"Can we bridge the gap?"

Container Hardening

Existing works

Confine/SysFilter/C2C
Static system call filtering

SPEAKER
Phase-split filtering

Seccomp
System call filtering

Ptrace
Process observation

Limitations
- Only works for single system calls the app does not need
- Prevent unused syscalls and hope for the best

Related Work

Forensic Analysis and Attack Detection

Existing works

CLARION
Provenance graph collection for containers

Atlas/RapS
Provenance graph analysis

Falco
Attack detection and alert

Limitations
- Manual inspection and enforcement

Dig to the root cause …

Proactive Forensics

"Can we bridge the gap?"

Container Hardening

Existing works

Confine/SysFilter/C2C
Static system call filtering

SPEAKER
Phase-split filtering

Seccomp
System call filtering

Ptrace
Process observation

Limitations
- Only works for single system calls the app does not need
- Prevent unused syscalls and hope for the best

Forensic Analysis and Attack Detection

Existing works

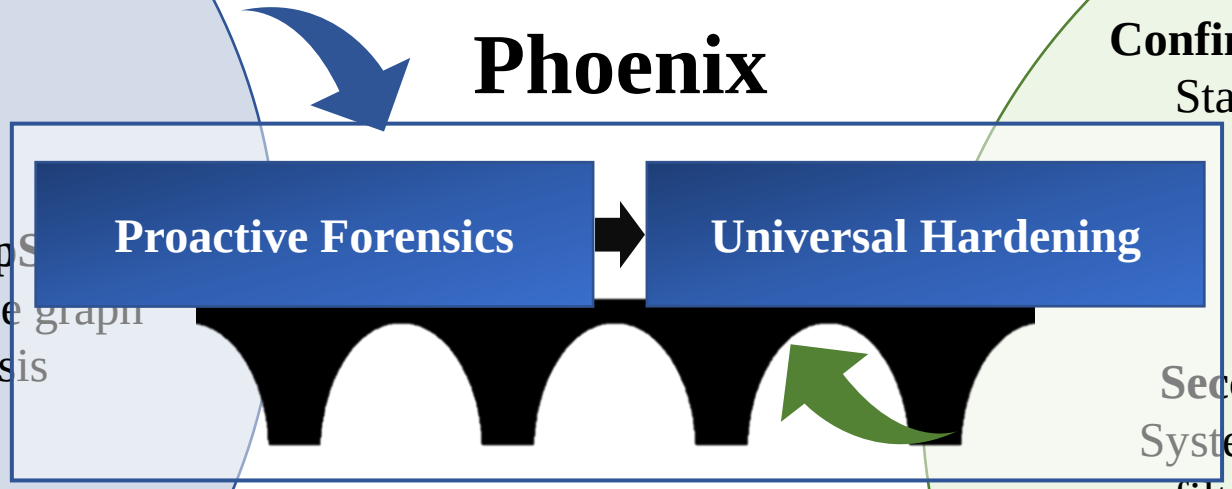
CLARION
Provenance graph collection for containers

Atlas/RapSheet
Provenance graph analysis

Falco
Attack detection and alert

Limitations
- Manual inspection and enforcement

Dig to the root cause …



… and fix it

"Can we bridge the gap?"

Container Hardening

Existing works

Confine/SysFilter/C2C
Static system call filtering

SPEAKER
Phase-split filtering

Seccomp
System call filtering

Ptrace
Process observation

Limitations
- Only works for single system calls the app does not need
- Prevent unused syscalls and hope for the best

Outline

- Intro
  - Motivation
  - Related Work
- **Methodology**
  - **Key Ideas & Overview**
  - **Malicious Sequence Identification**
  - **Dynamic Runtime Protection**
- Implementation
- Experimental Results
  - Security
  - Performance
  - Provenance Analysis
- Conclusion

Key Ideas

Key Ideas

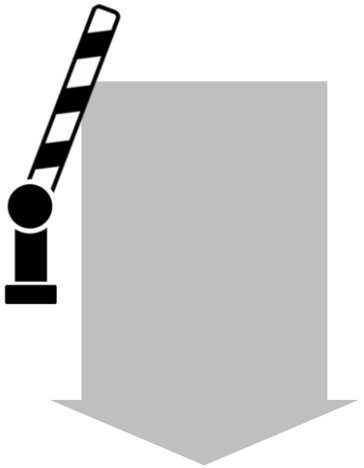
Limitations in Existing Solutions

Limitations in Existing Solutions

Insecure

Block syscalls unused by the container

Seccomp

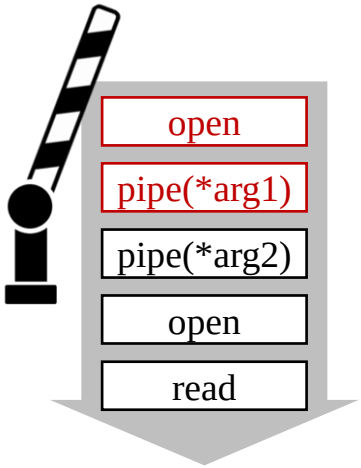


Limitations in Existing Solutions

Insecure

Block syscalls unused by the container

Seccomp



Limitations in Existing Solutions

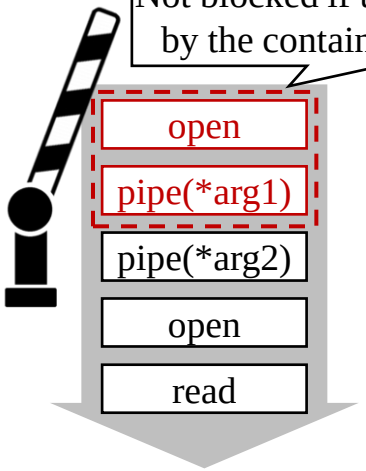
Insecure

Block syscalls unused by the container

Seccomp

Insecure

Not blocked if used by the container



Limitations in Existing Solutions

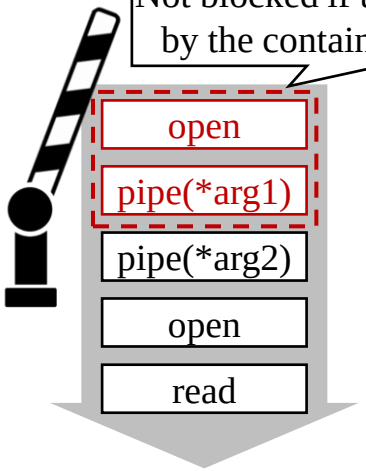
Insecure

Block syscalls unused by the container

Seccomp

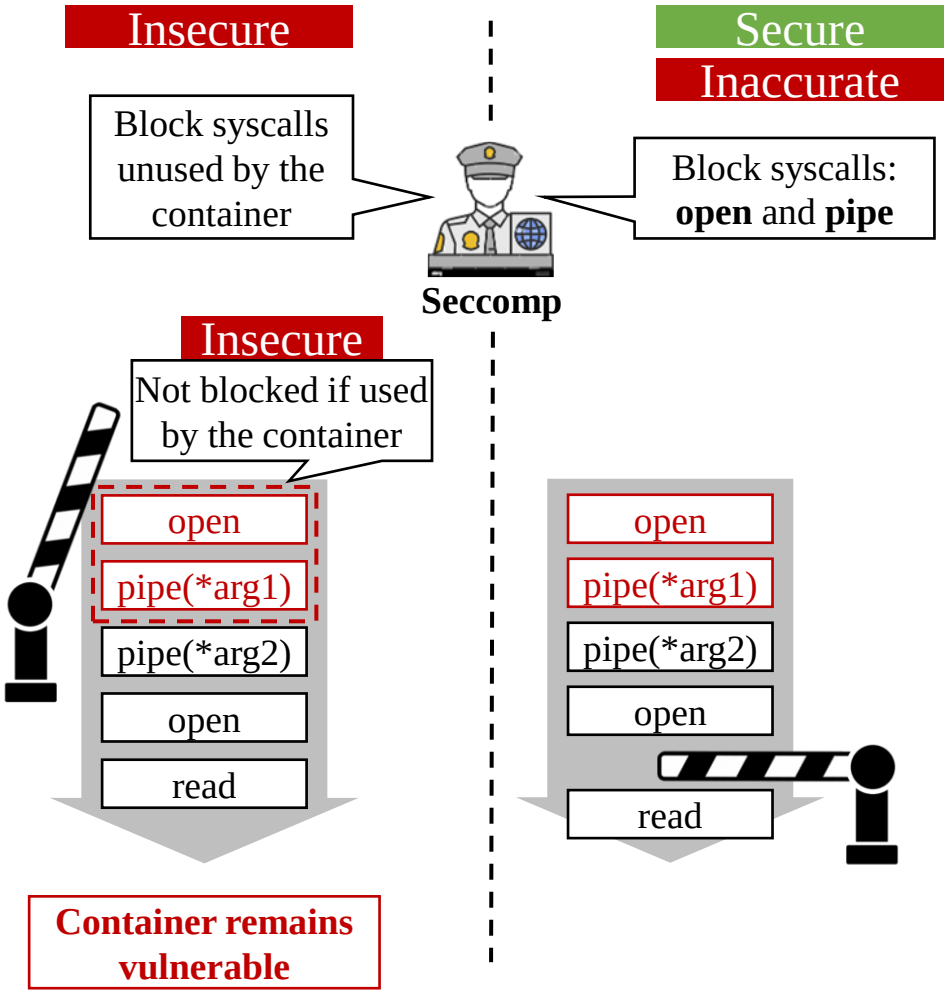
Insecure

Not blocked if used by the container



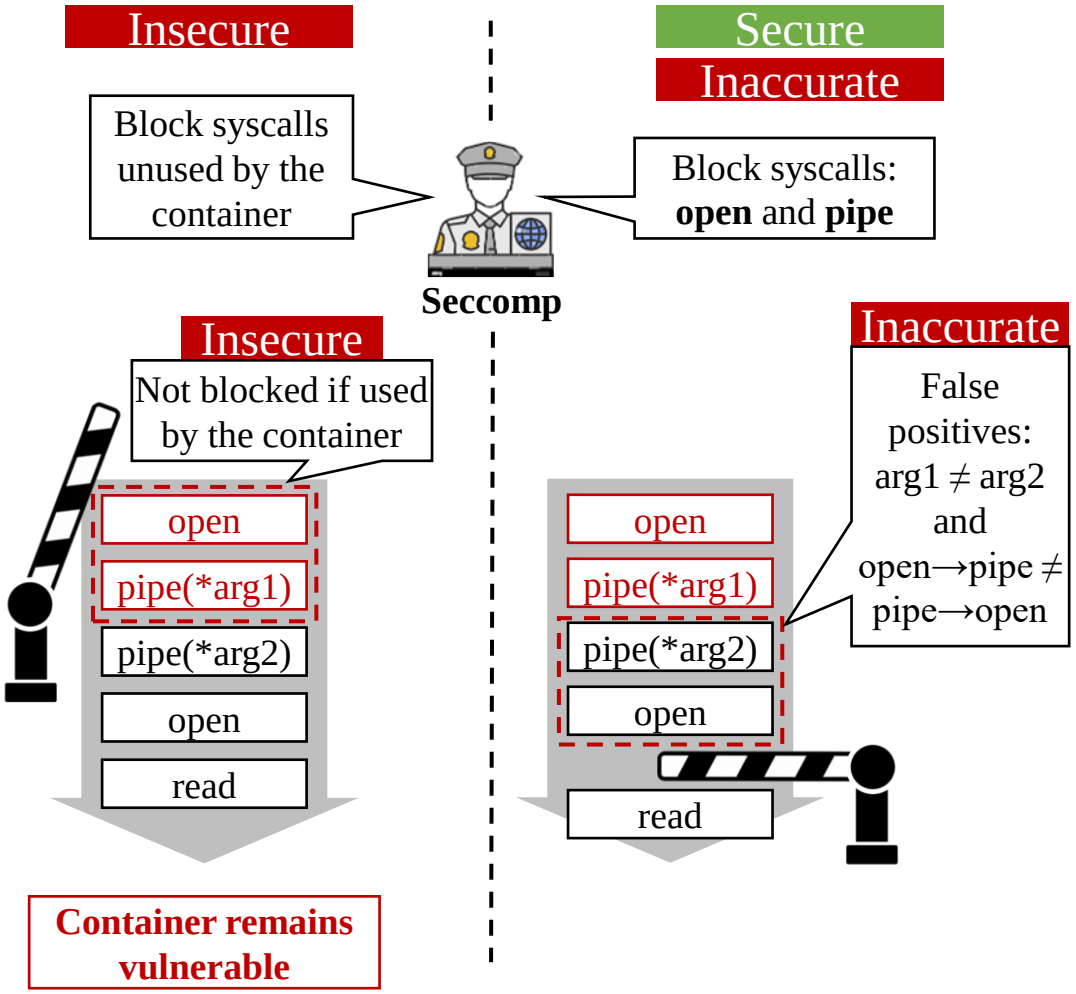
Container remains vulnerable

Limitations in Existing Solutions

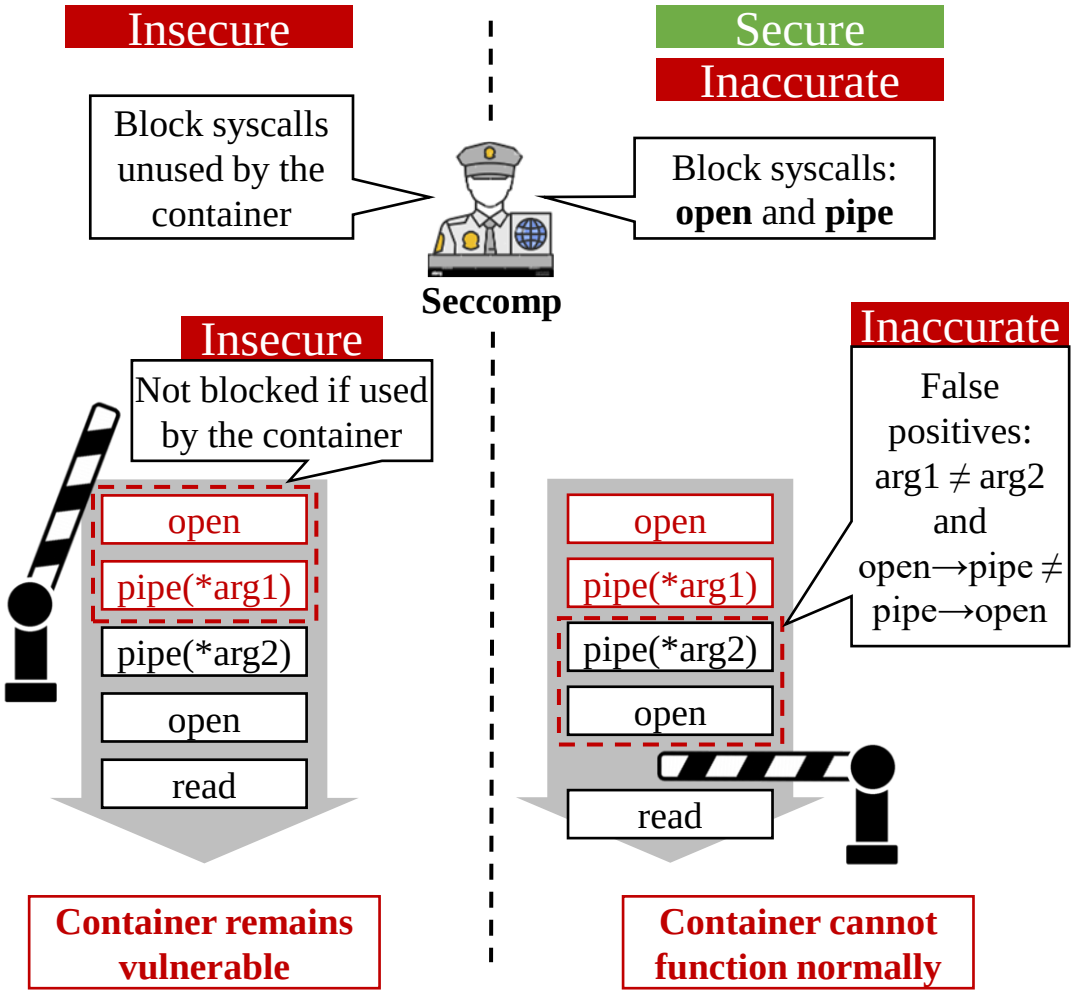


Key Ideas

Limitations in Existing Solutions



Limitations in Existing Solutions



Key Ideas

Limitations in Existing Solutions

Insecure

Block syscalls unused by the container

Seccomp

Secure
Inaccurate

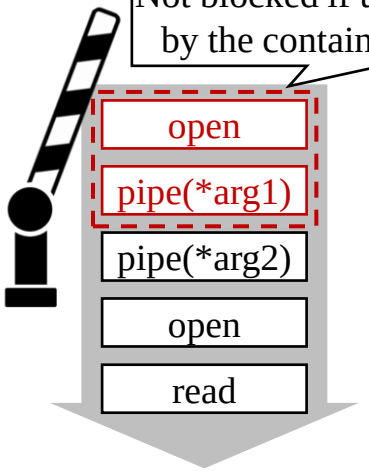
Block syscalls: **open** and **pipe**

Secure
Accurate
Inefficient

Ptrace

Insecure

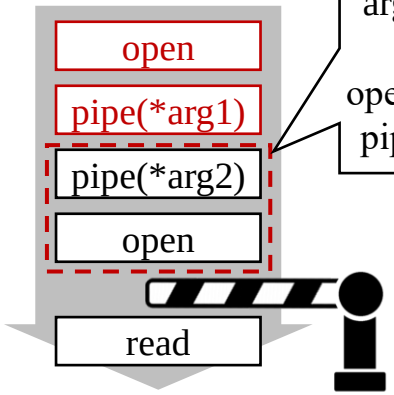
Not blocked if used by the container



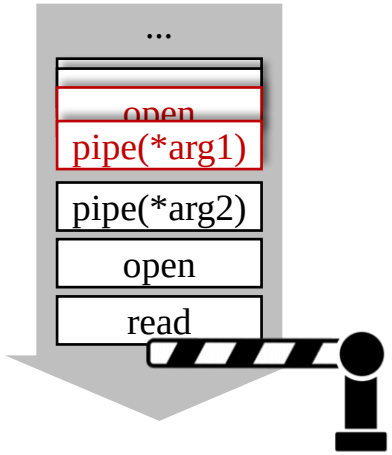
Container remains vulnerable

Inaccurate

False positives: arg1 ≠ arg2 and open→pipe ≠ pipe→open

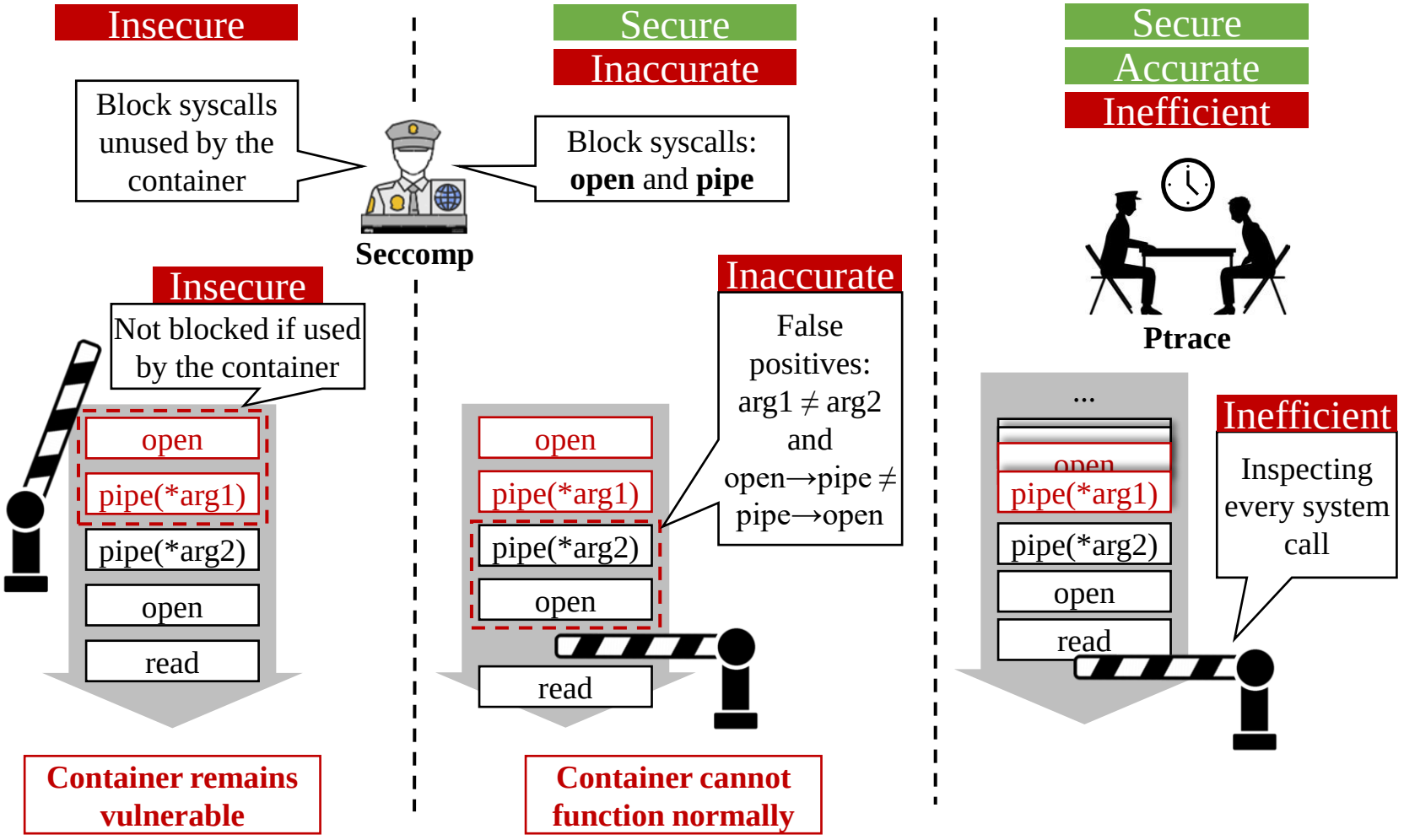


Container cannot function normally

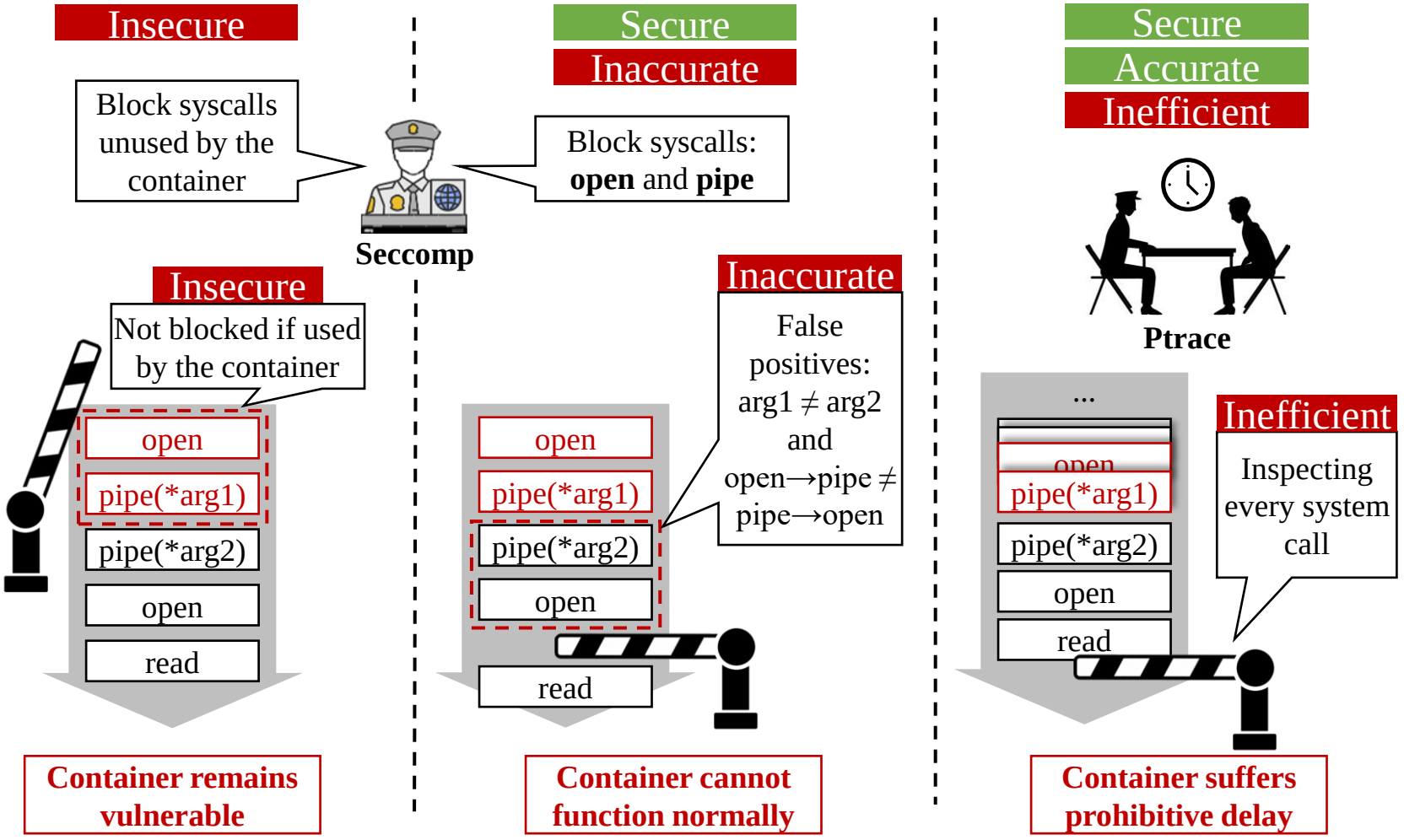


Key Ideas

Limitations in Existing Solutions



Limitations in Existing Solutions



Key Ideas

Limitations in Existing Solutions

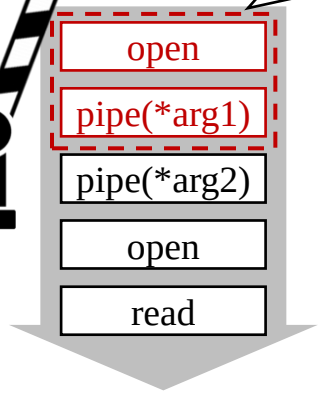
Insecure

Block syscalls unused by the container

Seccomp

Insecure

Not blocked if used by the container



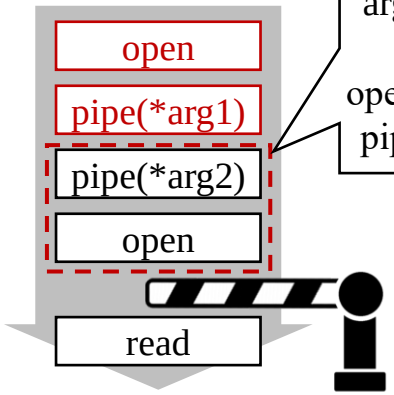
Container remains vulnerable

Secure
Inaccurate

Block syscalls: **open** and **pipe**

Inaccurate

False positives: arg1 ≠ arg2 and open→pipe ≠ pipe→open



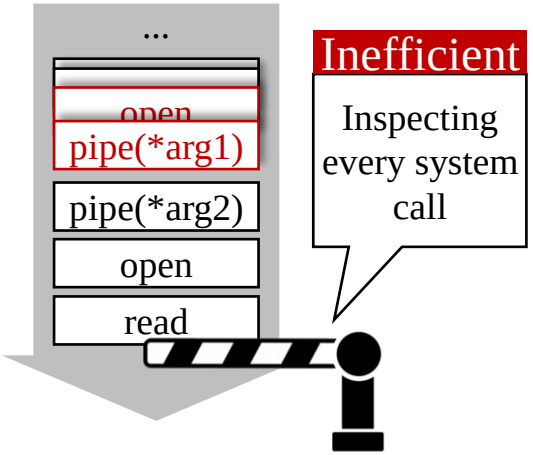
Container cannot function normally

Secure
Accurate
Inefficient

Ptrace

Inefficient

Inspecting every system call



Container suffers prohibitive delay

Our Solution: *Phoenix*

Secure
Accurate
Efficient

Our Ideas

Key Ideas

Limitations in Existing Solutions

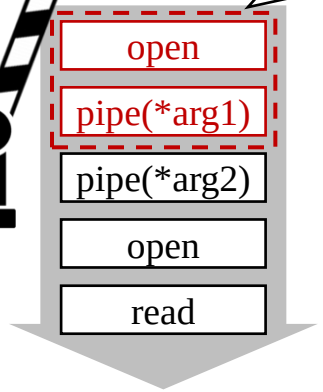
Insecure

Block syscalls unused by the container

Seccomp

Insecure

Not blocked if used by the container



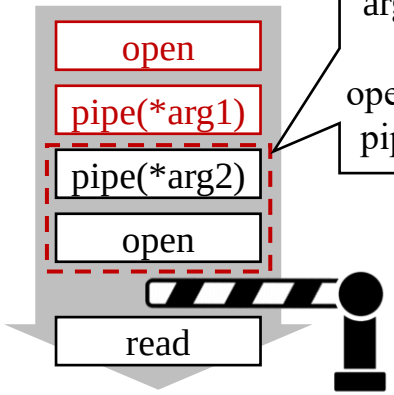
Container remains vulnerable

Secure
Inaccurate

Block syscalls: **open** and **pipe**

Inaccurate

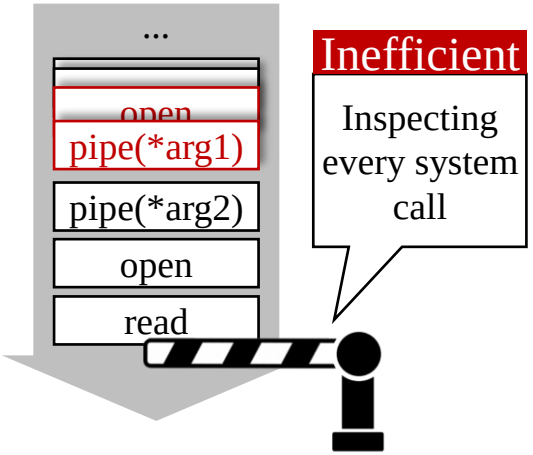
False positives: arg1 ≠ arg2 and open→pipe ≠ pipe→open



Container cannot function normally

Secure
Accurate
Inefficient

Ptrace



Inefficient

Inspecting every system call

Container suffers prohibitive delay

Our Solution: *Phoenix*

Our Ideas

Secure
Accurate
Efficient

pipe(*arg1)

Key Ideas

Limitations in Existing Solutions

Insecure

Block syscalls unused by the container

Seccomp

Secure
Inaccurate

Block syscalls: **open** and **pipe**

Insecure

Not blocked if used by the container

open
pipe(*arg1)
pipe(*arg2)
open
read

Container remains vulnerable

Inaccurate

False positives: arg1 ≠ arg2 and open→pipe ≠ pipe→open

open
pipe(*arg1)
pipe(*arg2)
open
read

Container cannot function normally

Secure
Accurate
Inefficient

Ptrace

...
open
pipe(*arg1)
pipe(*arg2)
open
read

Inefficient

Inspecting every system call

Container suffers prohibitive delay

Our Solution: *Phoenix*

Our Ideas

Secure
Accurate
Efficient

pipe(*arg1)

Is it X?
(X=pipe)

Key Ideas

Limitations in Existing Solutions

Insecure

Block syscalls unused by the container

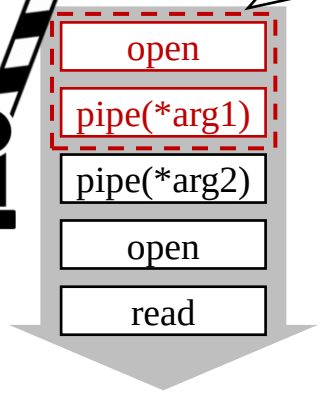
Seccomp

Secure
Inaccurate

Block syscalls: **open** and **pipe**

Insecure

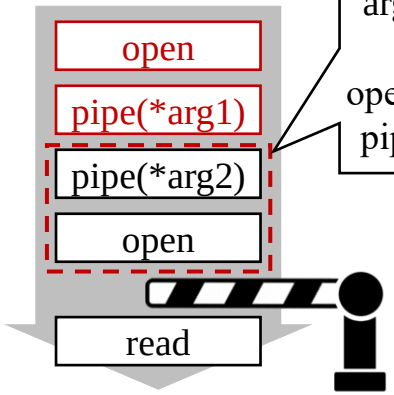
Not blocked if used by the container



Container remains vulnerable

Inaccurate

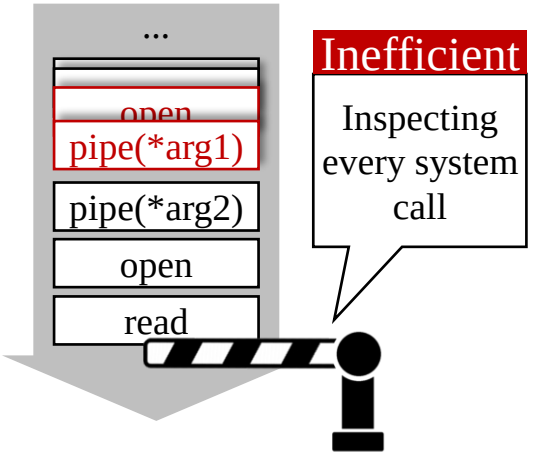
False positives: arg1 ≠ arg2 and open→pipe ≠ pipe→open



Container cannot function normally

Secure
Accurate
Inefficient

Ptrace



Inefficient

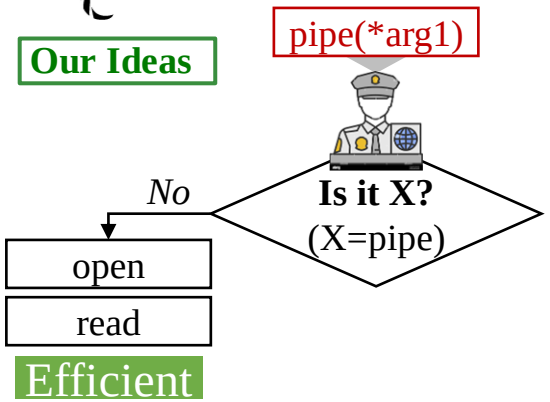
Inspecting every system call

Container suffers prohibitive delay

Our Solution: *Phoenix*

Secure
Accurate
Efficient

Our Ideas



Efficient

Key Ideas

Limitations in Existing Solutions

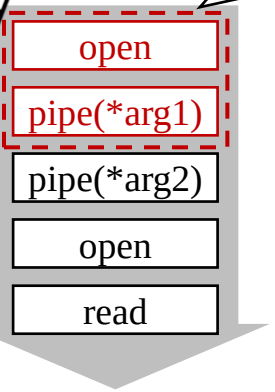
Insecure

Block syscalls unused by the container

Seccomp

Insecure

Not blocked if used by the container



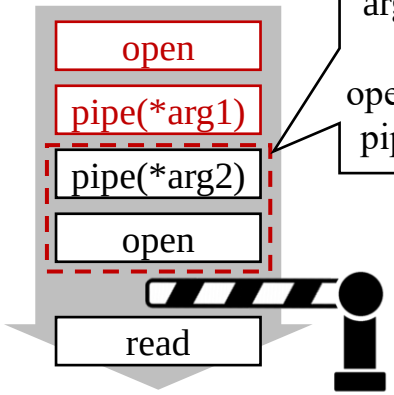
Container remains vulnerable

Secure
Inaccurate

Block syscalls: **open** and **pipe**

Inaccurate

False positives: arg1 ≠ arg2 and open→pipe ≠ pipe→open



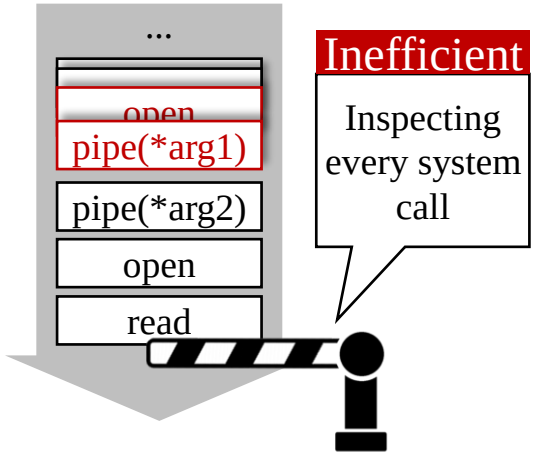
Container cannot function normally

Secure
Accurate
Inefficient

Ptrace

Inefficient

Inspecting every system call

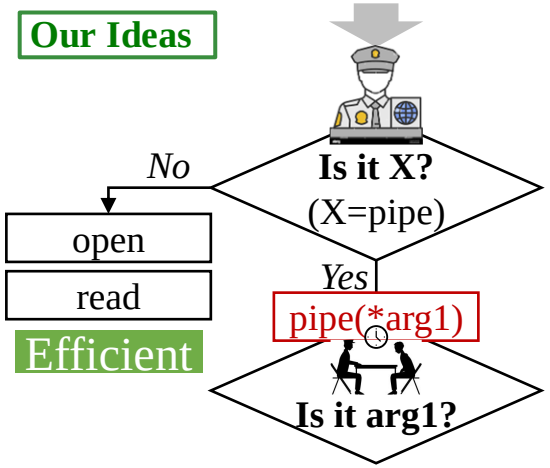


Container suffers prohibitive delay

Our Solution: *Phoenix*

Secure
Accurate
Efficient

Our Ideas



Key Ideas

Limitations in Existing Solutions

Insecure

Block syscalls unused by the container

Seccomp

Secure
Inaccurate

Block syscalls: **open** and **pipe**

Insecure

Not blocked if used by the container

open
pipe(*arg1)
pipe(*arg2)
open
read

Container remains vulnerable

Inaccurate

open
pipe(*arg1)
pipe(*arg2)
open
read

False positives: arg1 ≠ arg2 and open→pipe ≠ pipe→open

Container cannot function normally

Secure
Accurate
Inefficient

Ptrace

...
open
pipe(*arg1)
pipe(*arg2)
open
read

Inefficient

Inspecting every system call

Container suffers prohibitive delay

Our Solution: *Phoenix*

Secure
Accurate
Efficient

Our Ideas

No

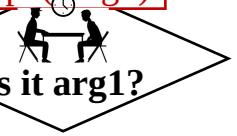
Is it X? (X=pipe)

Yes

open
read

Efficient

pipe(*arg1)



Is it arg1?

pipe(*arg2)

Key Ideas

Limitations in Existing Solutions

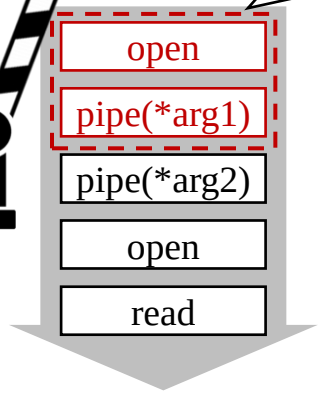
Insecure

Block syscalls unused by the container

Seccomp

Insecure

Not blocked if used by the container



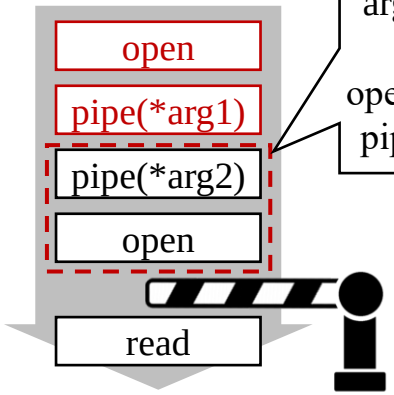
Container remains vulnerable

Secure
Inaccurate

Block syscalls: **open** and **pipe**

Inaccurate

False positives: arg1 ≠ arg2 and open→pipe ≠ pipe→open



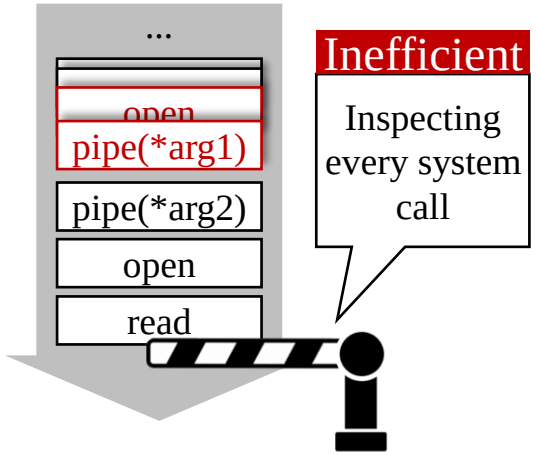
Container cannot function normally

Secure
Accurate
Inefficient

Ptrace

Inefficient

Inspecting every system call

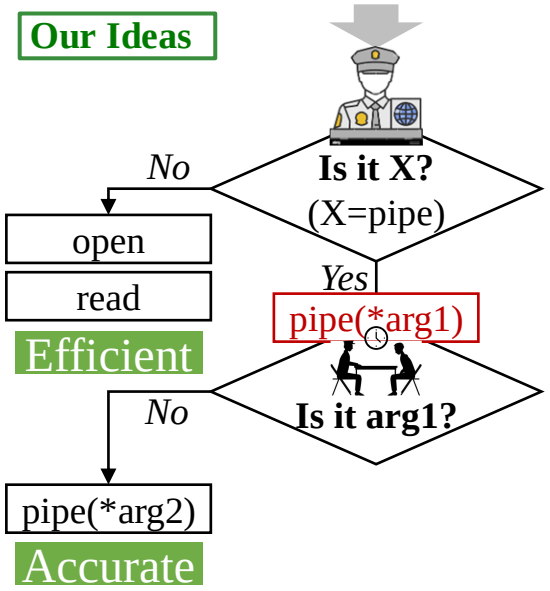


Container suffers prohibitive delay

Our Solution: *Phoenix*

Secure
Accurate
Efficient

Our Ideas



Key Ideas

Limitations in Existing Solutions

Insecure

Block syscalls unused by the container

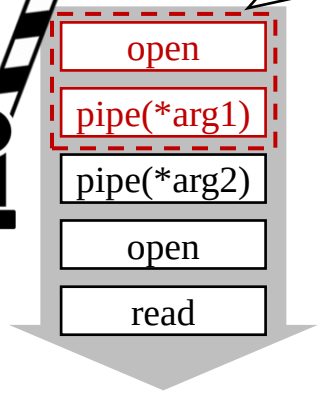
Seccomp

Secure
Inaccurate

Block syscalls: **open** and **pipe**

Insecure

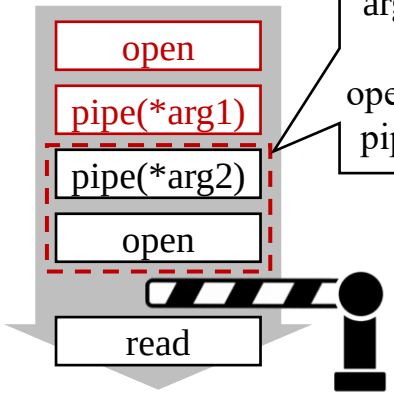
Not blocked if used by the container



Container remains vulnerable

Inaccurate

False positives: arg1 ≠ arg2 and open→pipe ≠ pipe→open



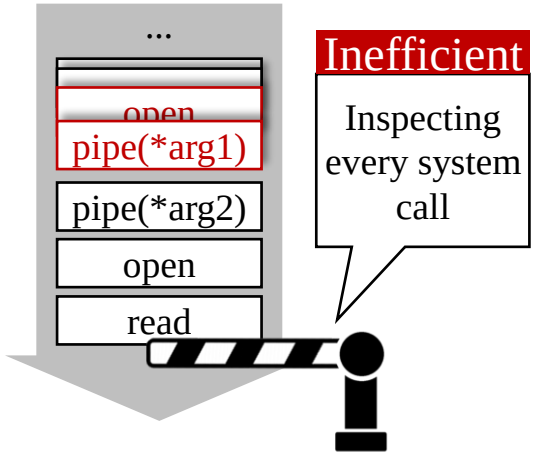
Container cannot function normally

Secure
Accurate
Inefficient

Ptrace

Inefficient

Inspecting every system call

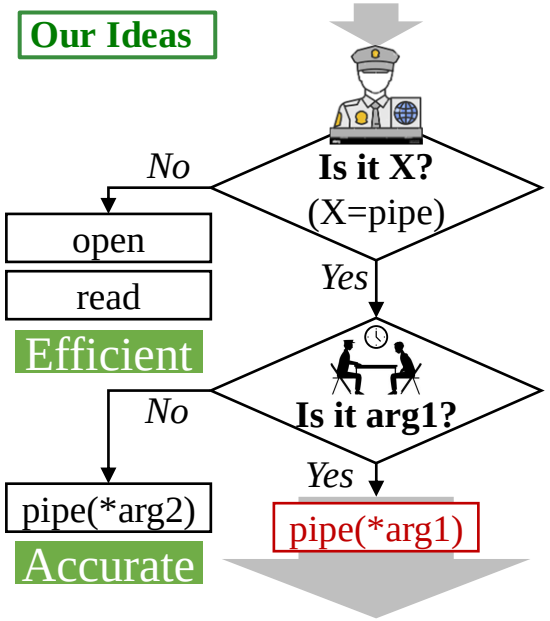


Container suffers prohibitive delay

Our Solution: *Phoenix*

Secure
Accurate
Efficient

Our Ideas



Key Ideas

Limitations in Existing Solutions

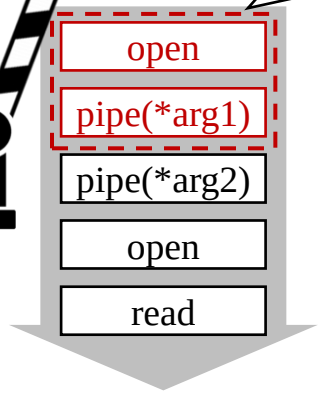
Insecure

Block syscalls unused by the container

Seccomp

Insecure

Not blocked if used by the container



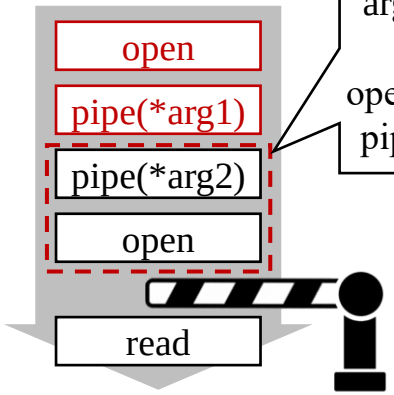
Container remains vulnerable

Secure
Inaccurate

Block syscalls: **open** and **pipe**

Inaccurate

False positives: arg1 ≠ arg2 and open→pipe ≠ pipe→open



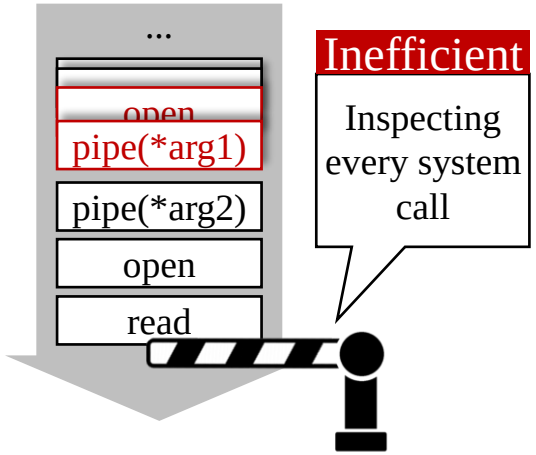
Container cannot function normally

Secure
Accurate
Inefficient

Ptrace

Inefficient

Inspecting every system call

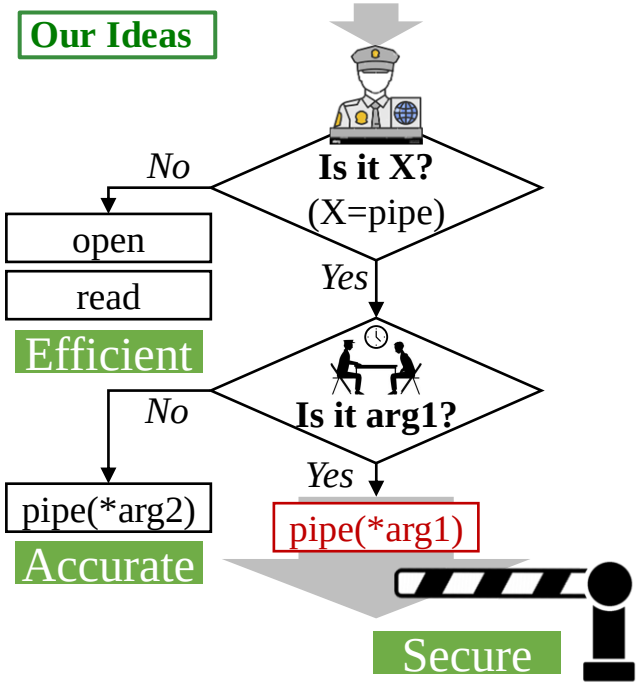


Container suffers prohibitive delay

Our Solution: *Phoenix*

Secure
Accurate
Efficient

Our Ideas



Key Ideas

Limitations in Existing Solutions

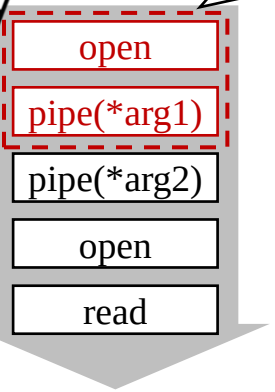
Insecure

Block syscalls unused by the container

Seccomp

Insecure

Not blocked if used by the container



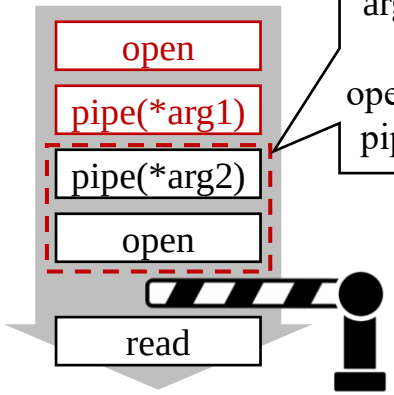
Container remains vulnerable

Secure
Inaccurate

Block syscalls: **open** and **pipe**

Inaccurate

False positives: arg1 ≠ arg2 and open→pipe ≠ pipe→open



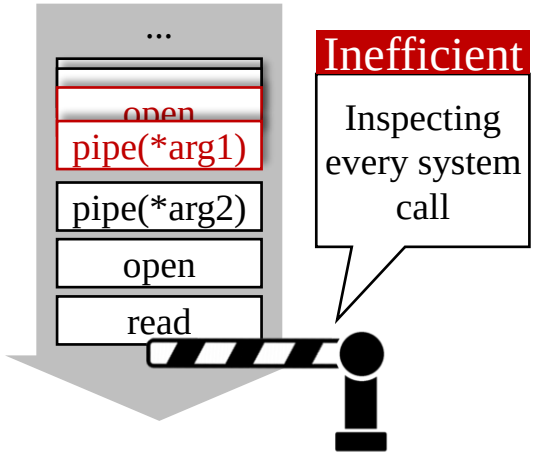
Container cannot function normally

Secure
Accurate
Inefficient

Ptrace

Inefficient

Inspecting every system call

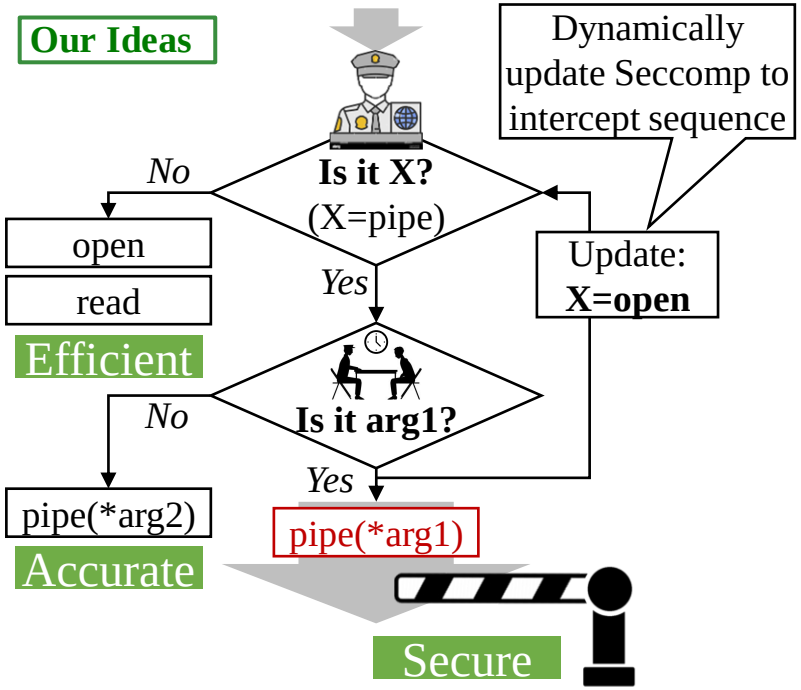


Container suffers prohibitive delay

Our Solution: *Phoenix*

Secure
Accurate
Efficient

Our Ideas



Key Ideas

Limitations in Existing Solutions

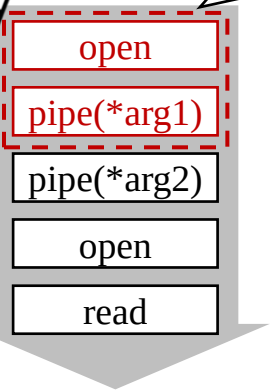
Insecure

Block syscalls unused by the container

Seccomp

Insecure

Not blocked if used by the container



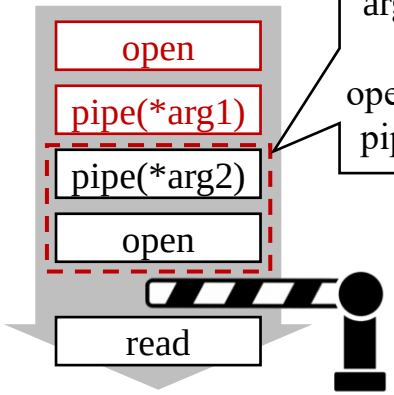
Container remains vulnerable

Secure
Inaccurate

Block syscalls: **open** and **pipe**

Inaccurate

False positives: arg1 ≠ arg2 and open→pipe ≠ pipe→open



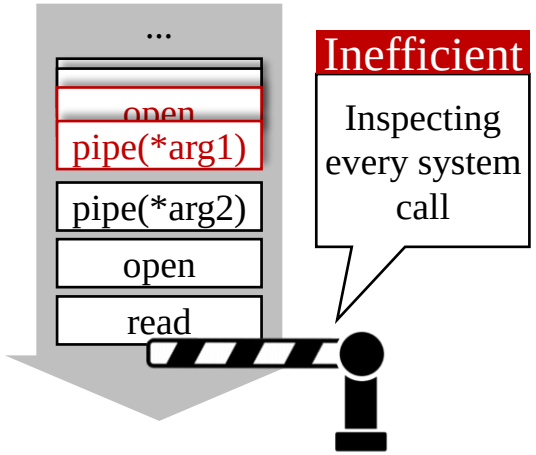
Container cannot function normally

Secure
Accurate
Inefficient

Ptrace

Inefficient

Inspecting every system call

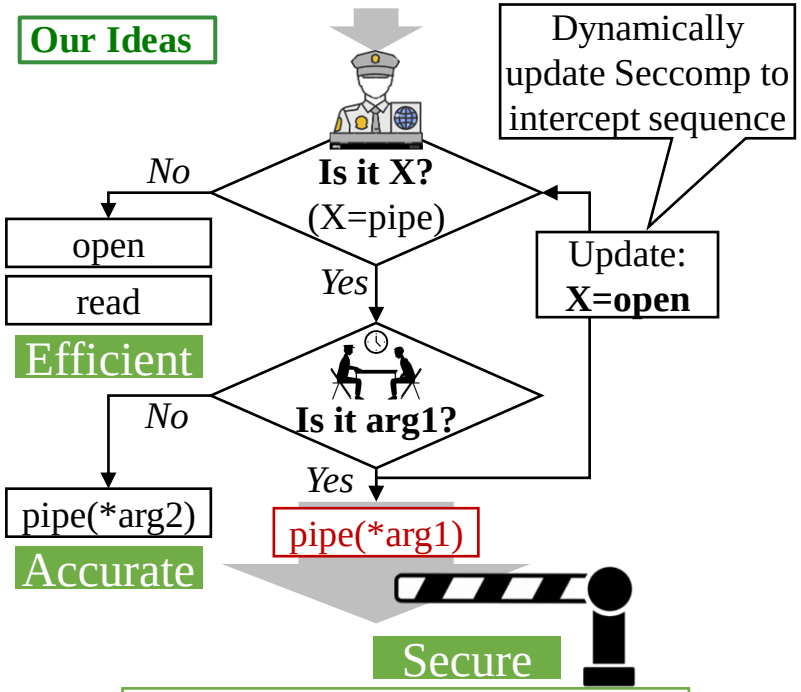


Container suffers prohibitive delay

Our Solution: *Phoenix*

Secure
Accurate
Efficient

Our Ideas



Container is protected, functions normally, and with minimal delay

Key Ideas

Limitations in Existing Solutions

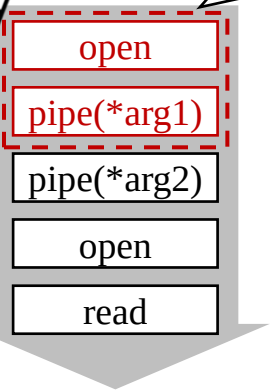
Insecure

Block syscalls unused by the container

Seccomp

Insecure

Not blocked if used by the container



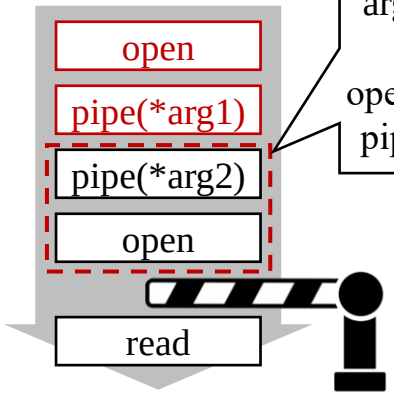
Container remains vulnerable

Secure
Inaccurate

Block syscalls: **open** and **pipe**

Inaccurate

False positives: arg1 ≠ arg2 and open→pipe ≠ pipe→open



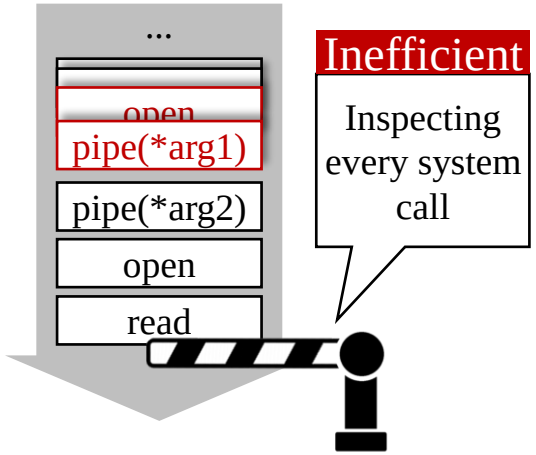
Container cannot function normally

Secure
Accurate
Inefficient

Ptrace

Inefficient

Inspecting every system call

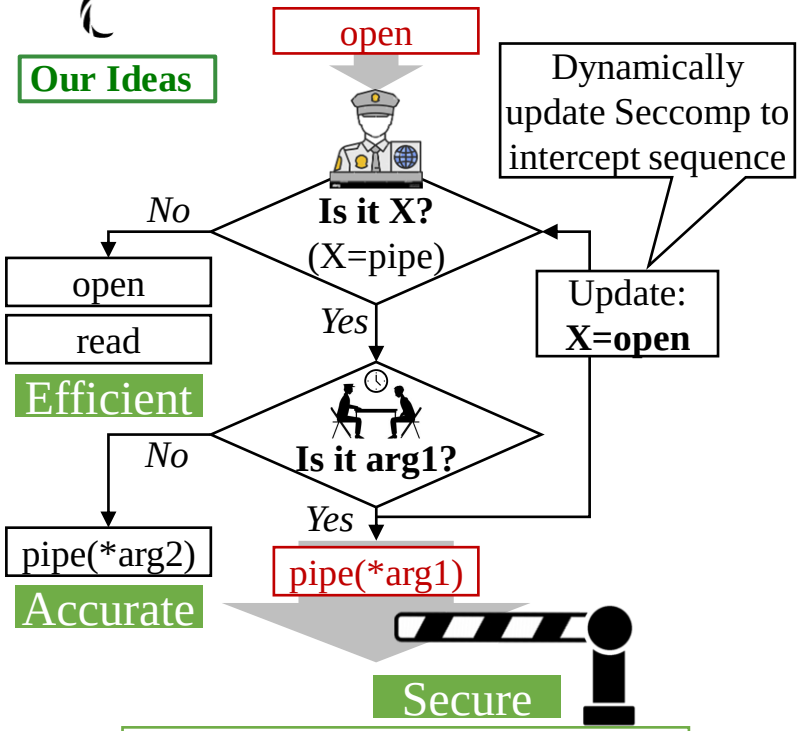


Container suffers prohibitive delay

Our Solution: *Phoenix*

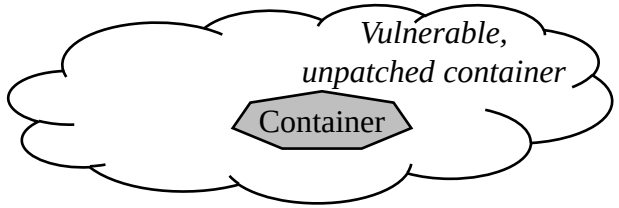
Our Ideas

Secure
Accurate
Efficient

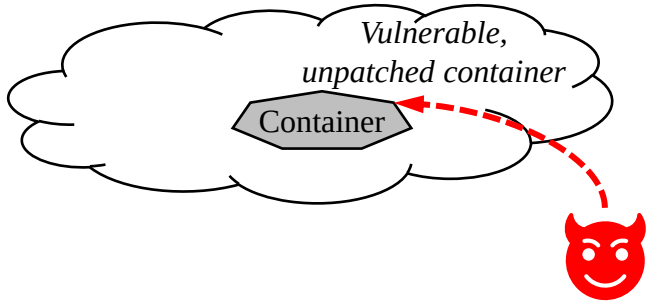


Container is protected, functions normally, and with minimal delay

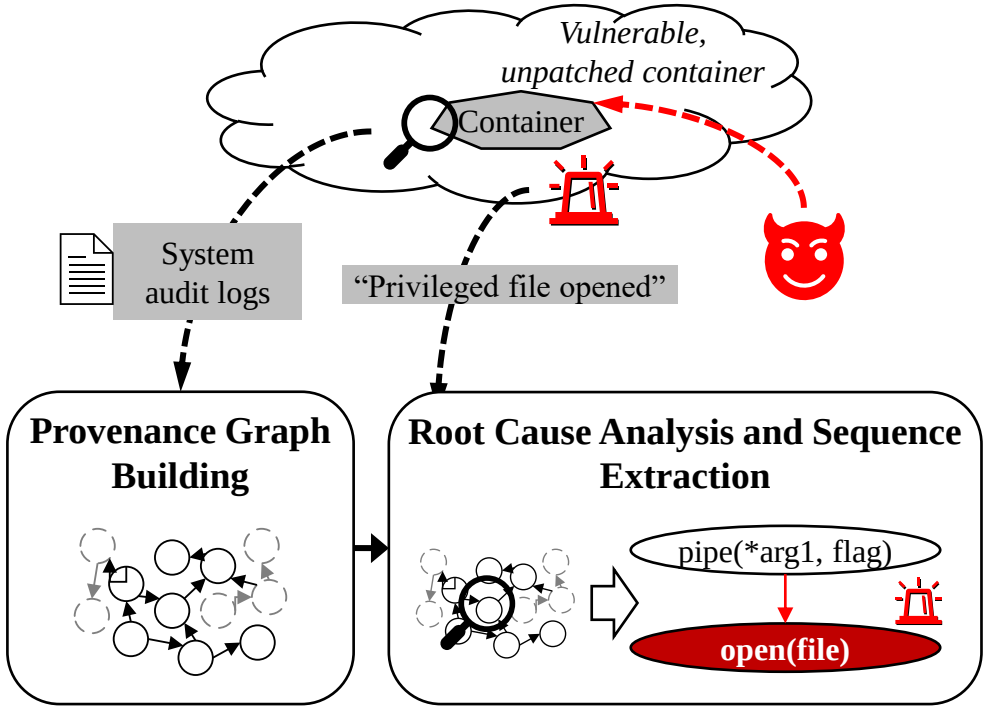
Methodology



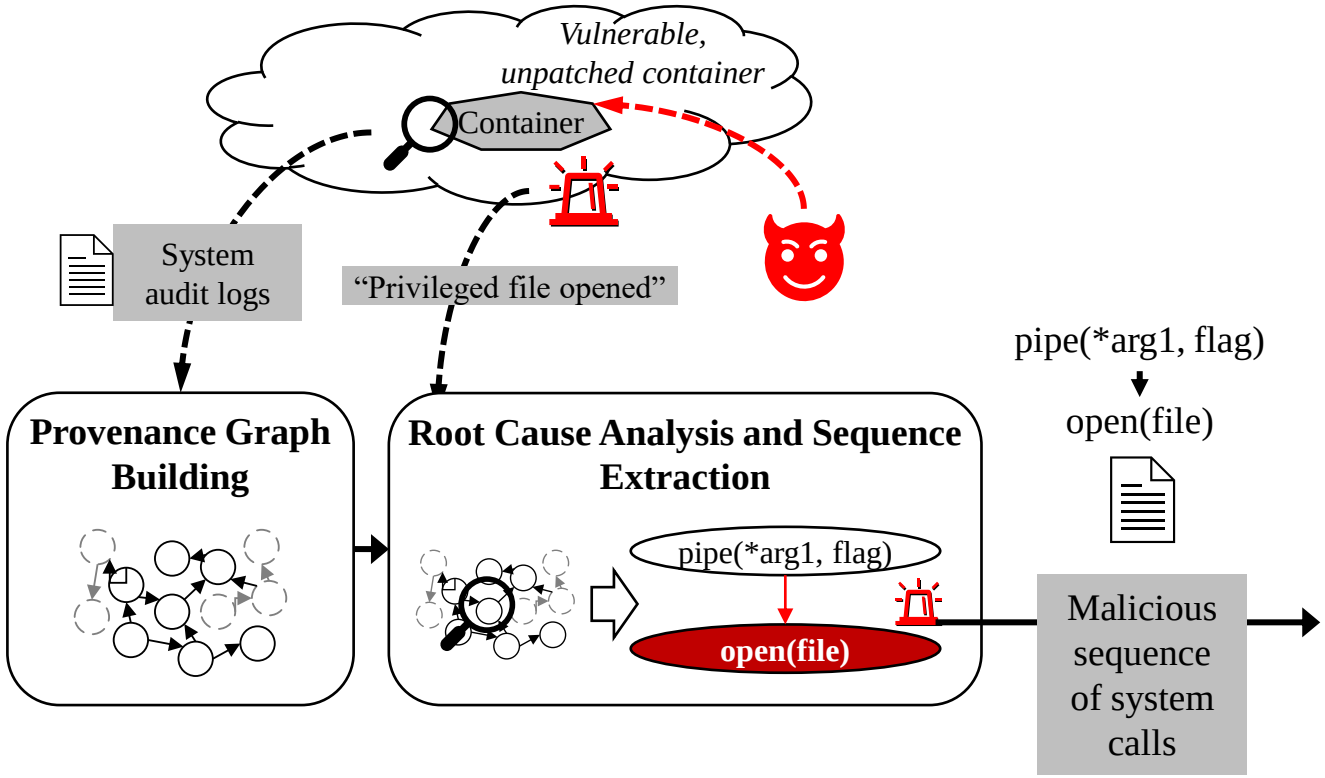
Methodology



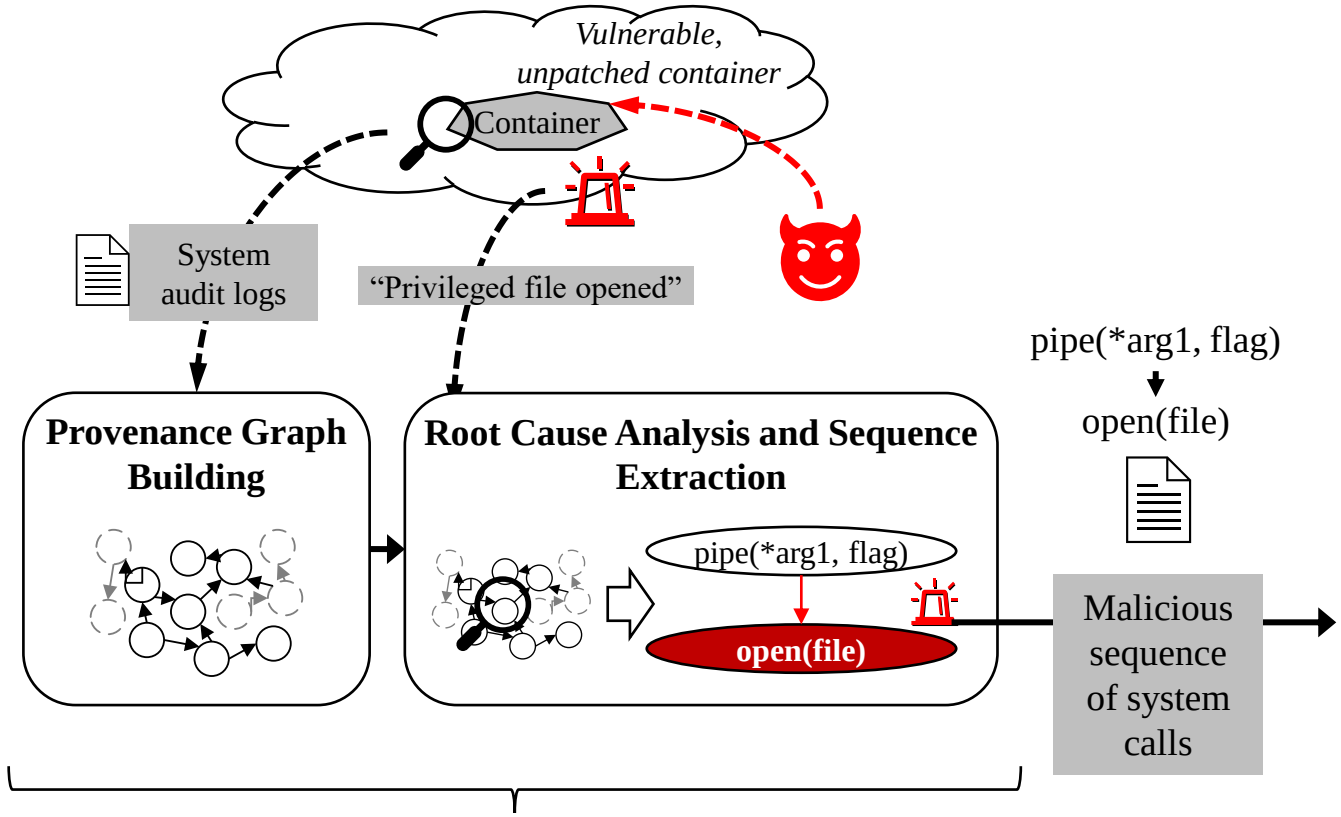
Methodology



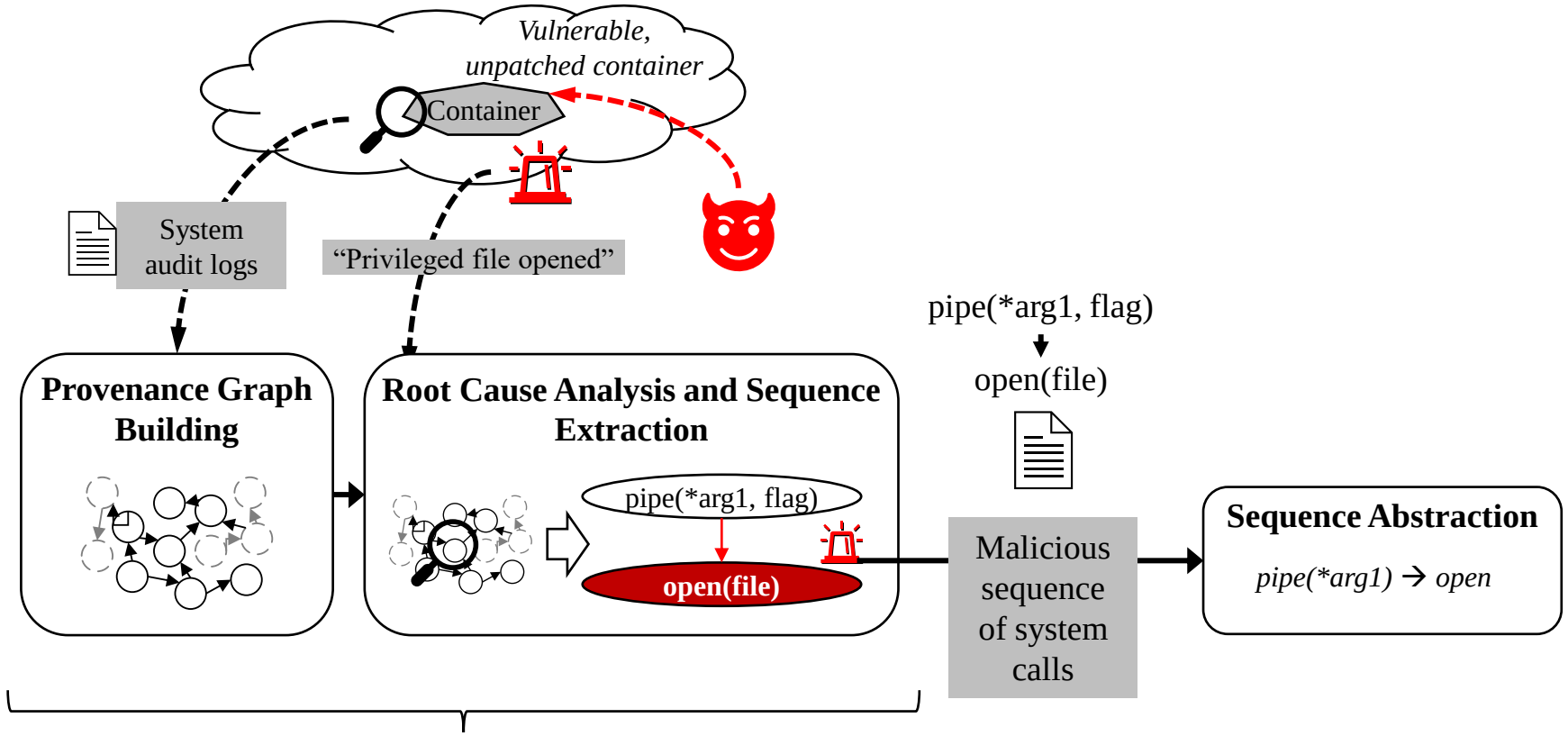
Methodology



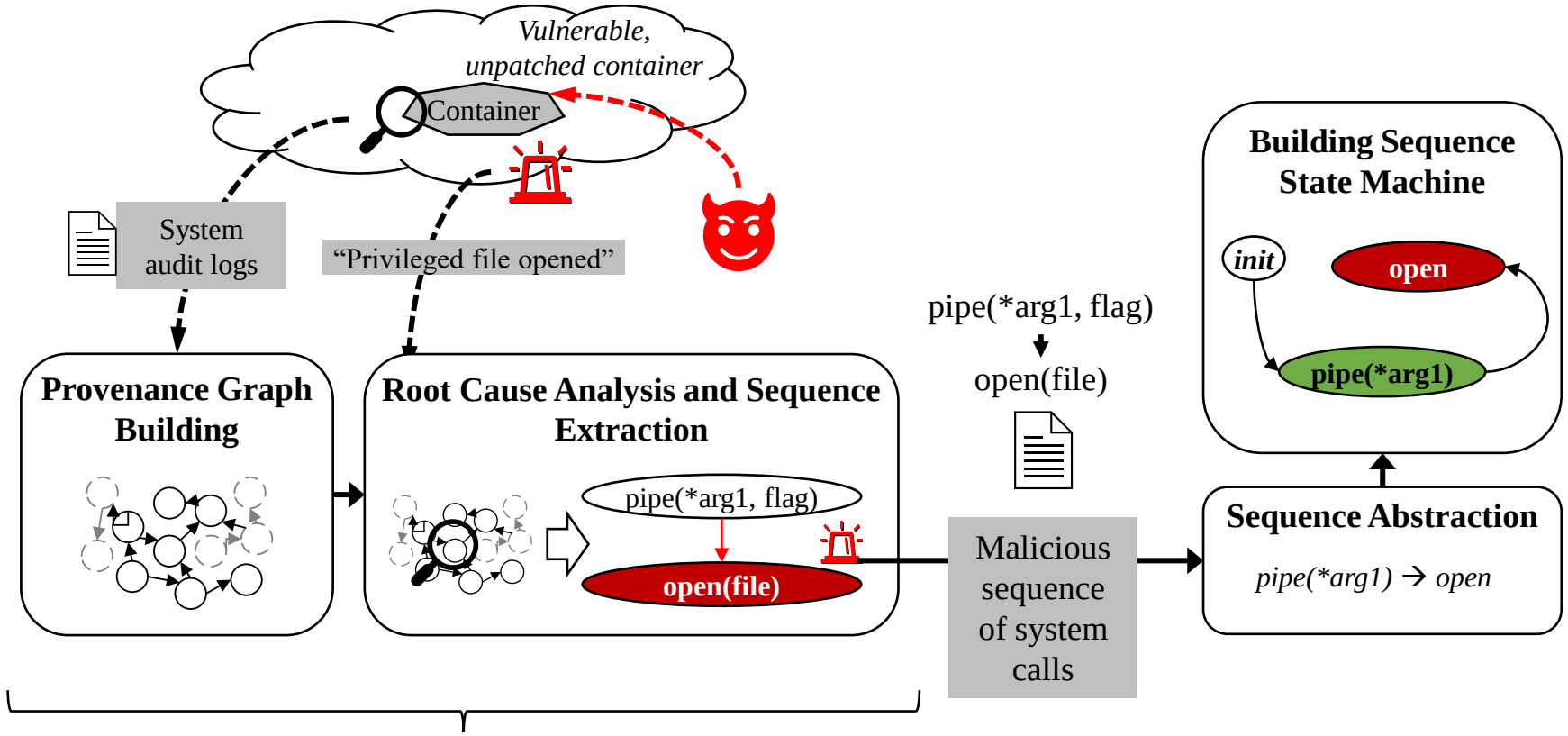
Methodology



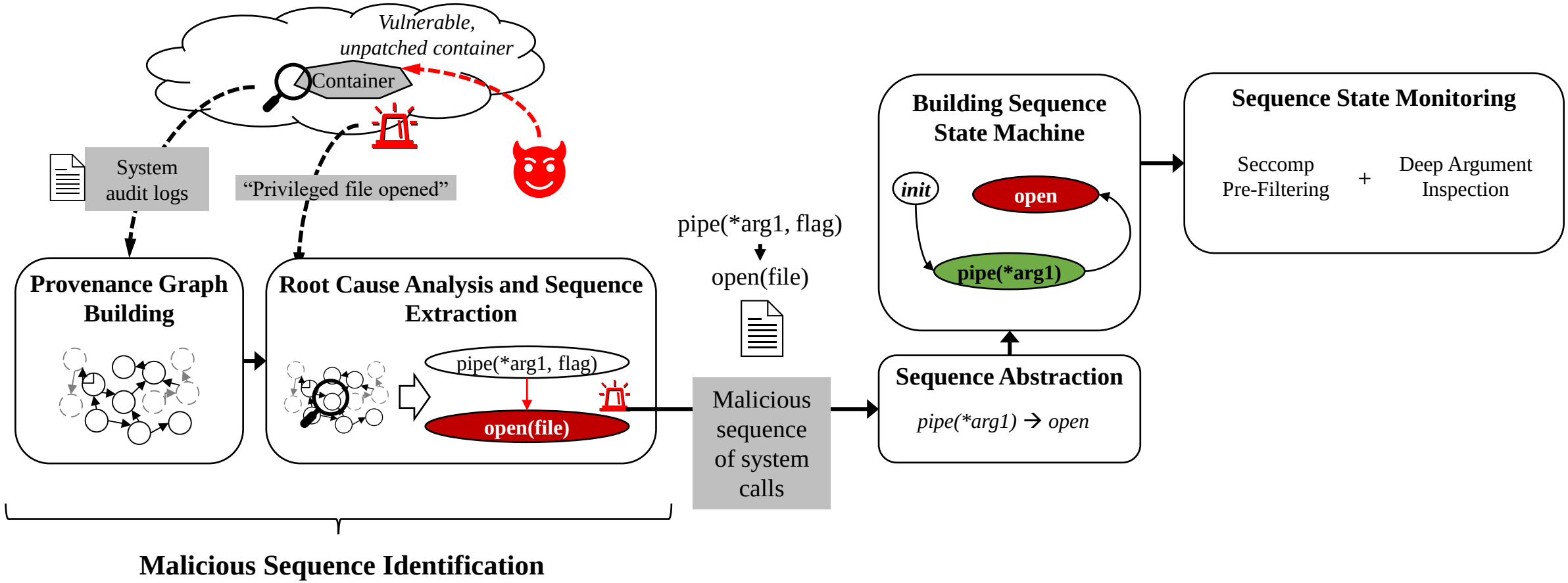
Methodology



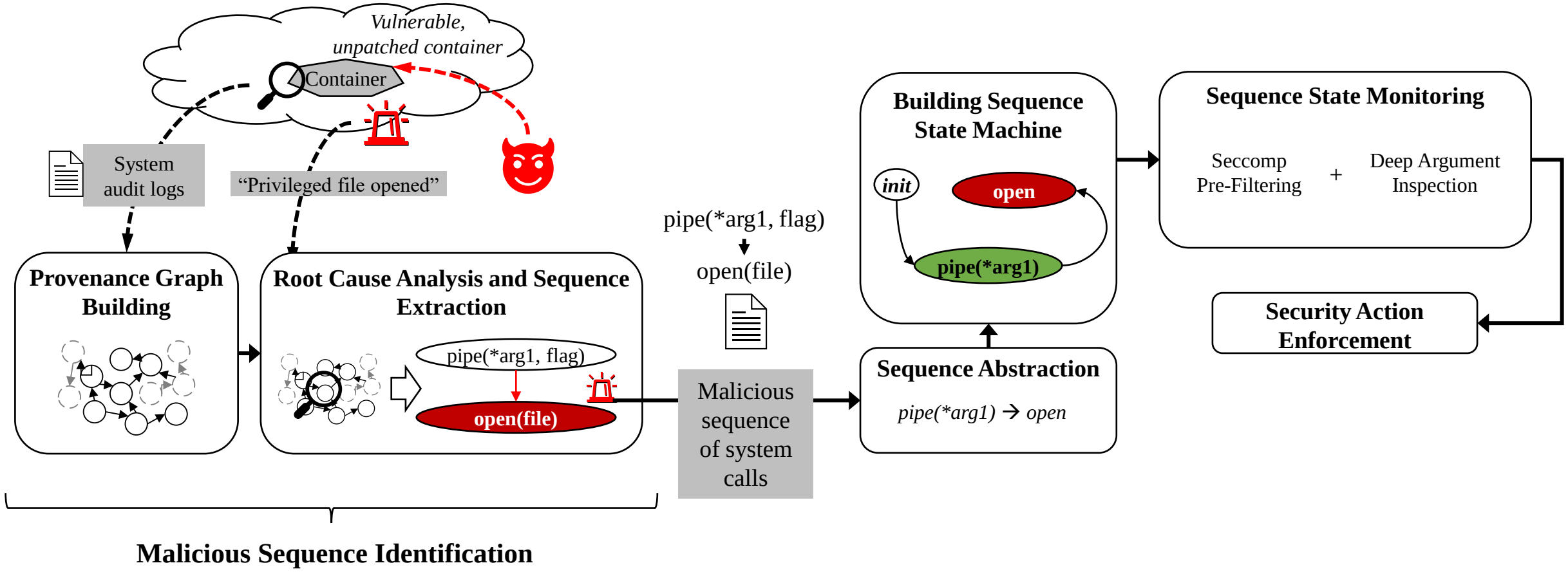
Methodology



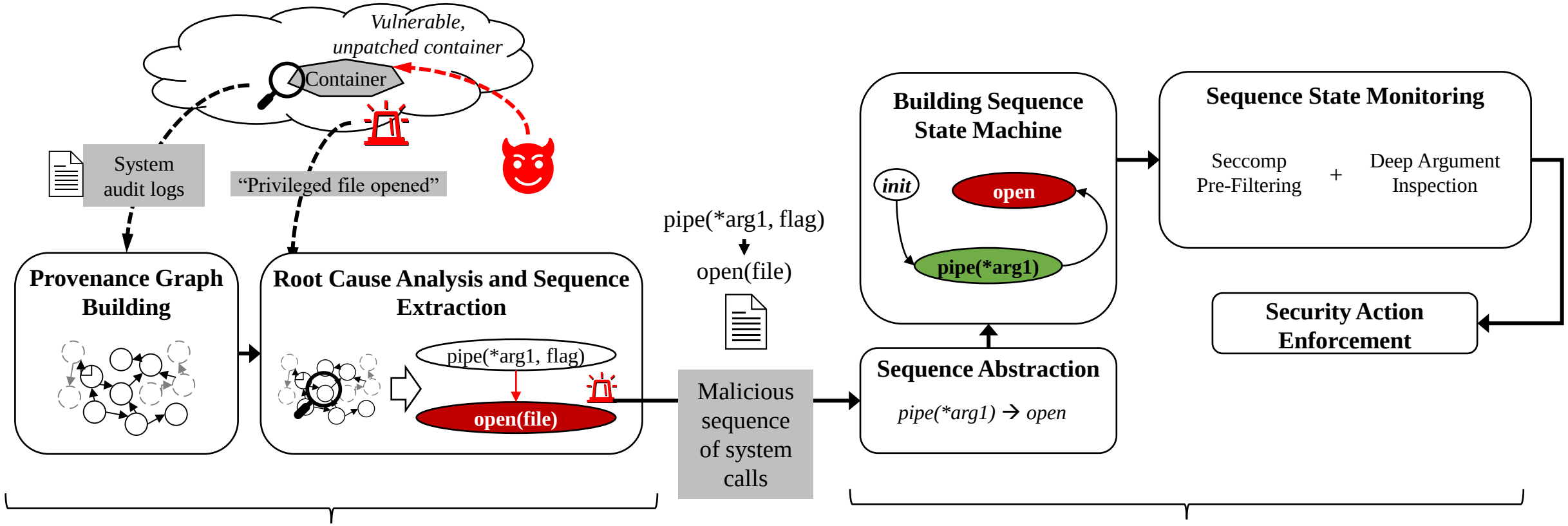
Methodology



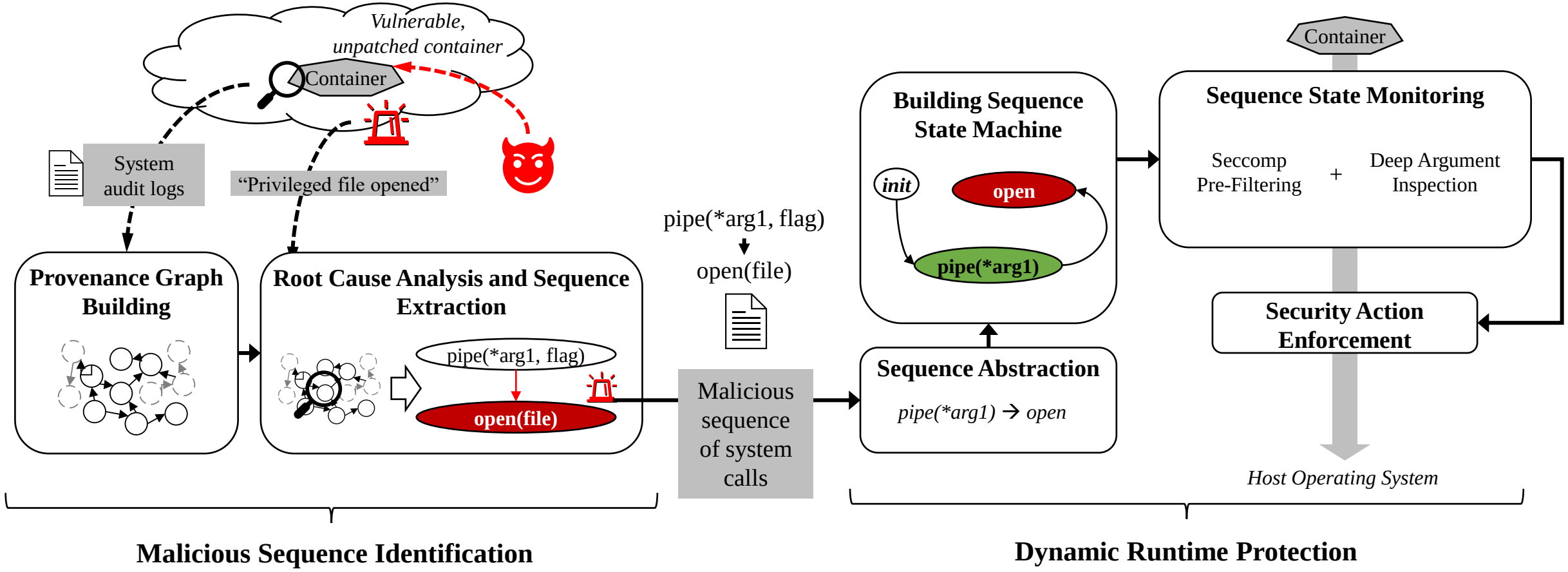
Methodology



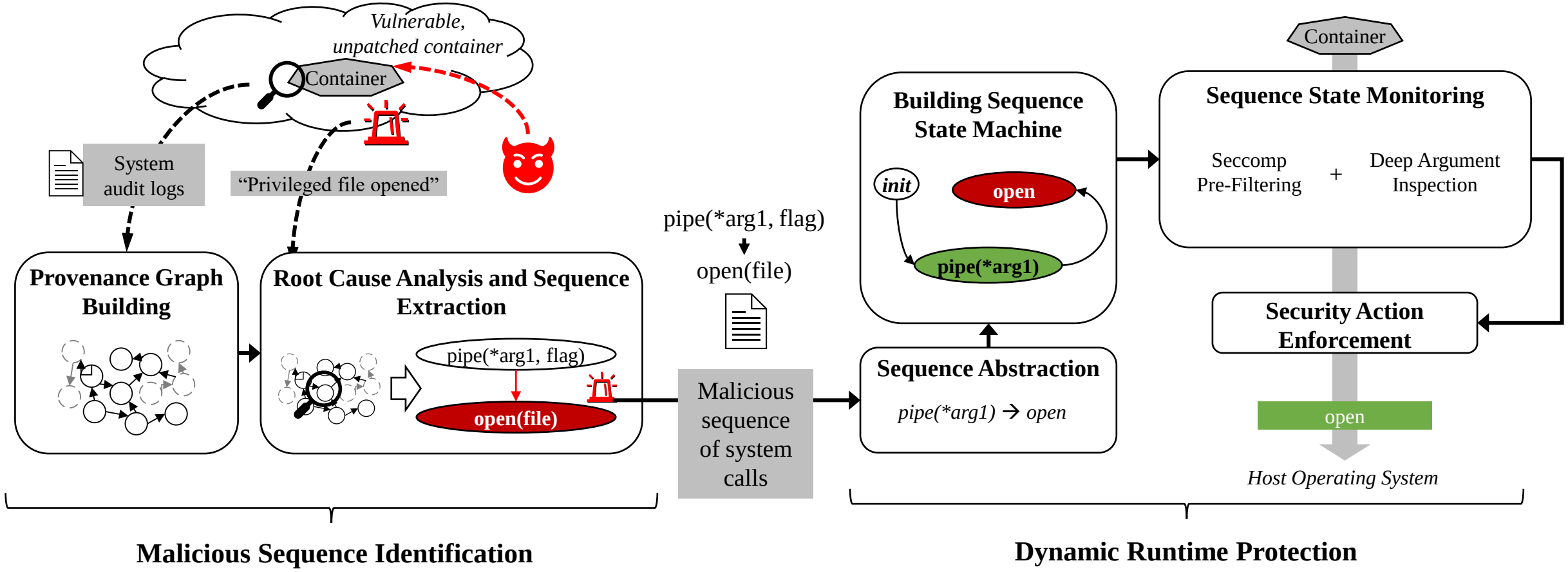
Methodology



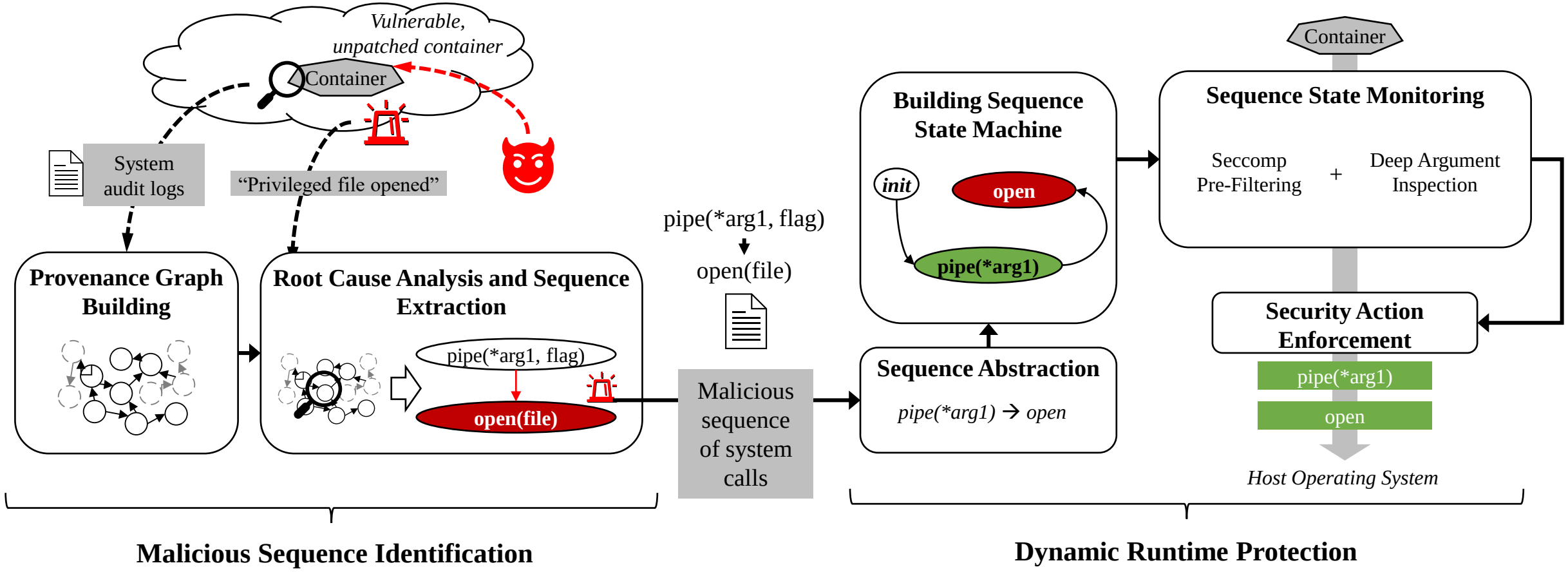
Methodology



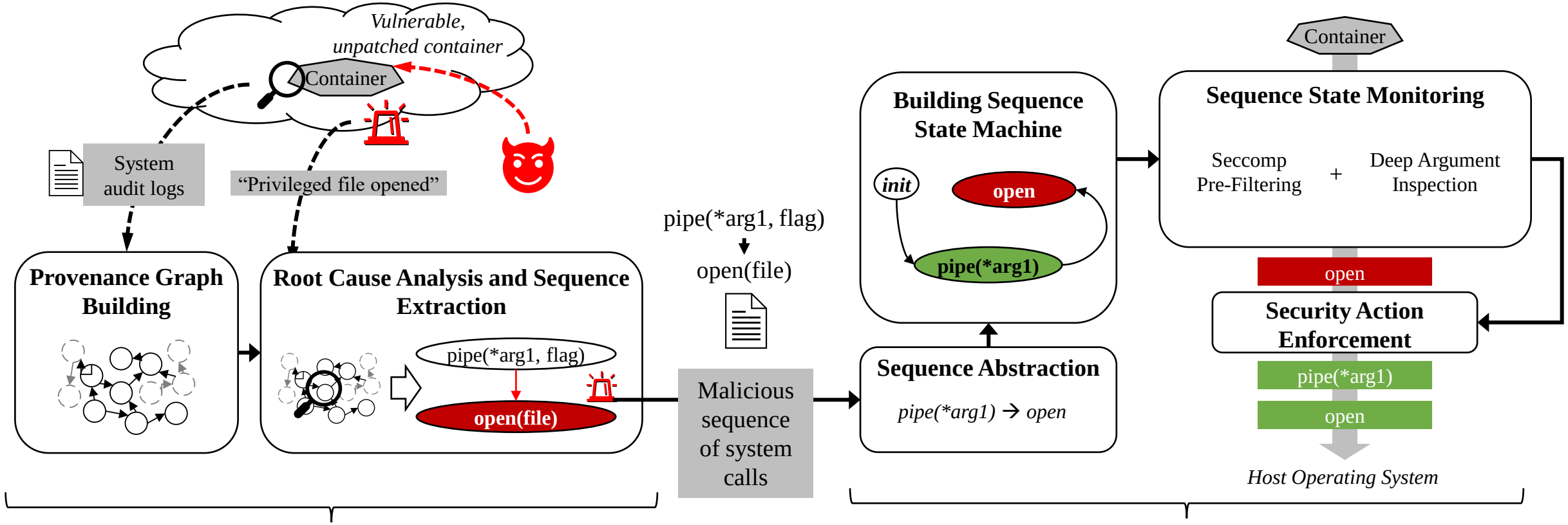
Methodology



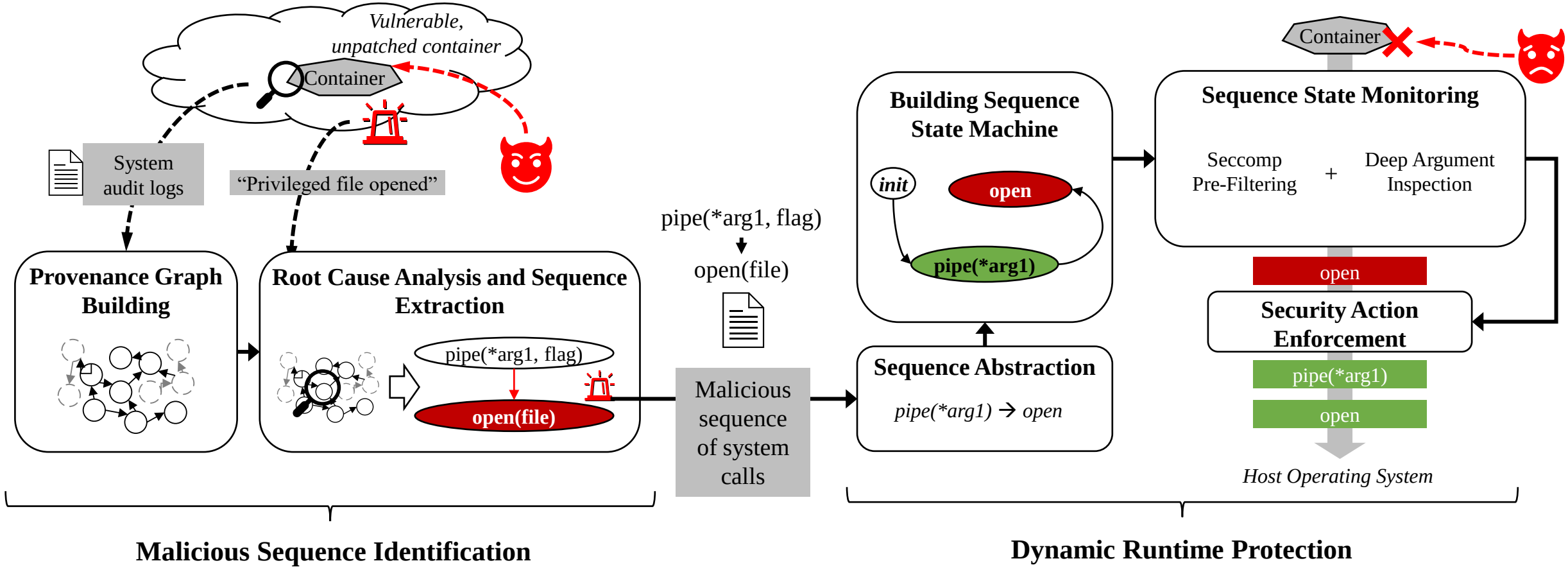
Methodology



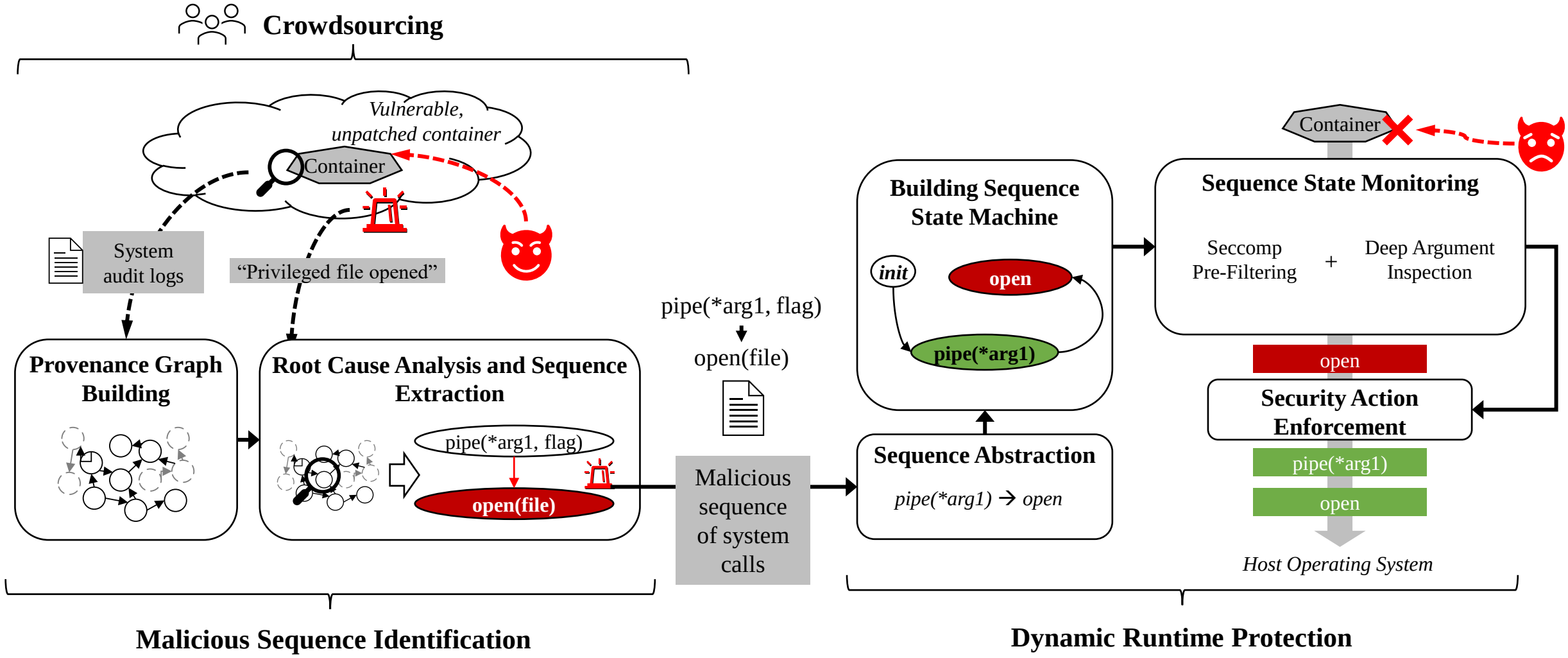
Methodology



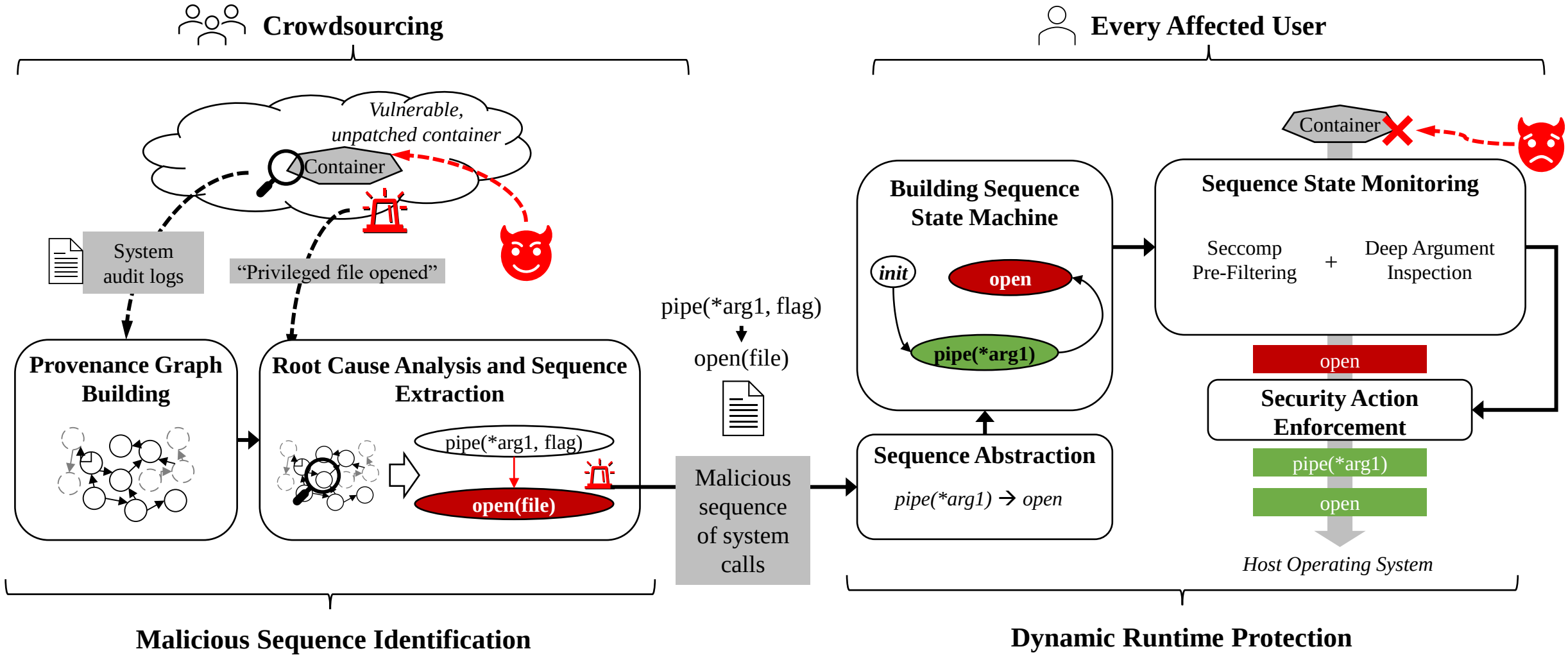
Methodology



Methodology

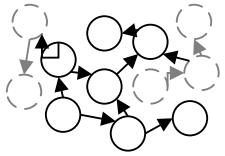


Methodology



Malicious Sequence Identification

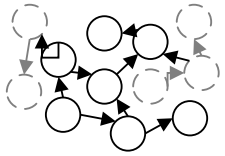
Provenance Graph Building



→ SPADE
→ Clarion

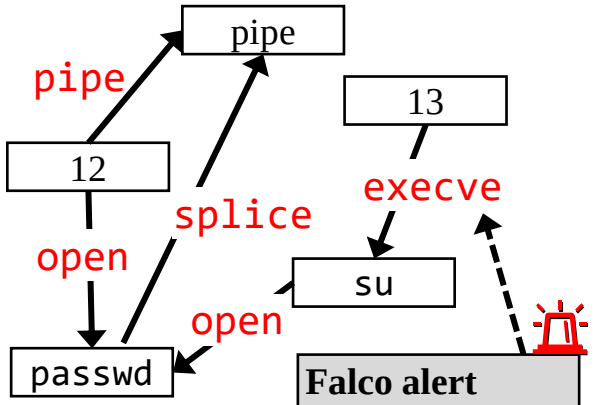
Malicious Sequence Identification

Provenance Graph Building



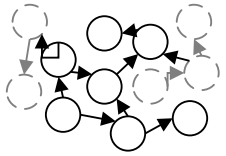
→ SPADE
→ Clarion

Root Cause Analysis



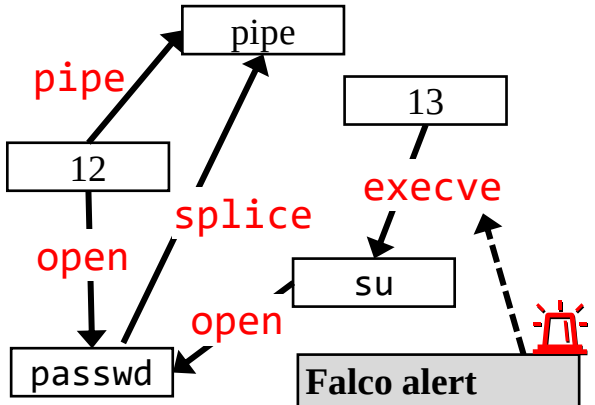
Malicious Sequence Identification

Provenance Graph Building



→ SPADE
→ Clarion

Root Cause Analysis

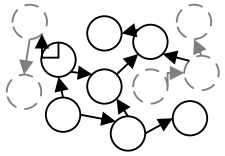


Sequence Extraction

```
...
12 open(“passwd”, RDONLY) = 4
...
12 pipe(5,6) = 0
12 splice(4,6) = 1
13 execve(“su”) = 0
13 open(“passwd”, RDONLY)
```

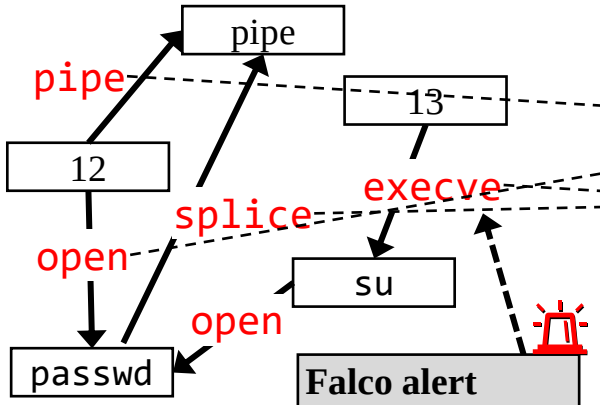
Malicious Sequence Identification

Provenance Graph Building



→ SPADE
→ Clarion

Root Cause Analysis

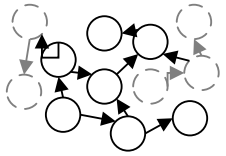


Sequence Extraction

```
...
12 open(“passwd”, RDONLY) = 4
...
12 pipe(5,6) = 0
12 splice(4,6) = 1
13 execve(“su”) = 0
13 open(“passwd”, RDONLY)
```

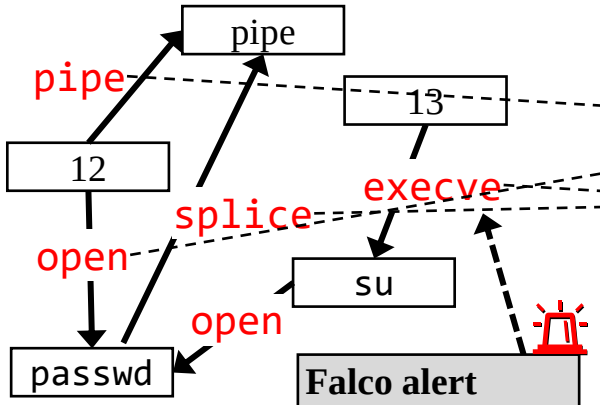
Malicious Sequence Identification

Provenance Graph Building



→ SPADE
→ Clarion

Root Cause Analysis

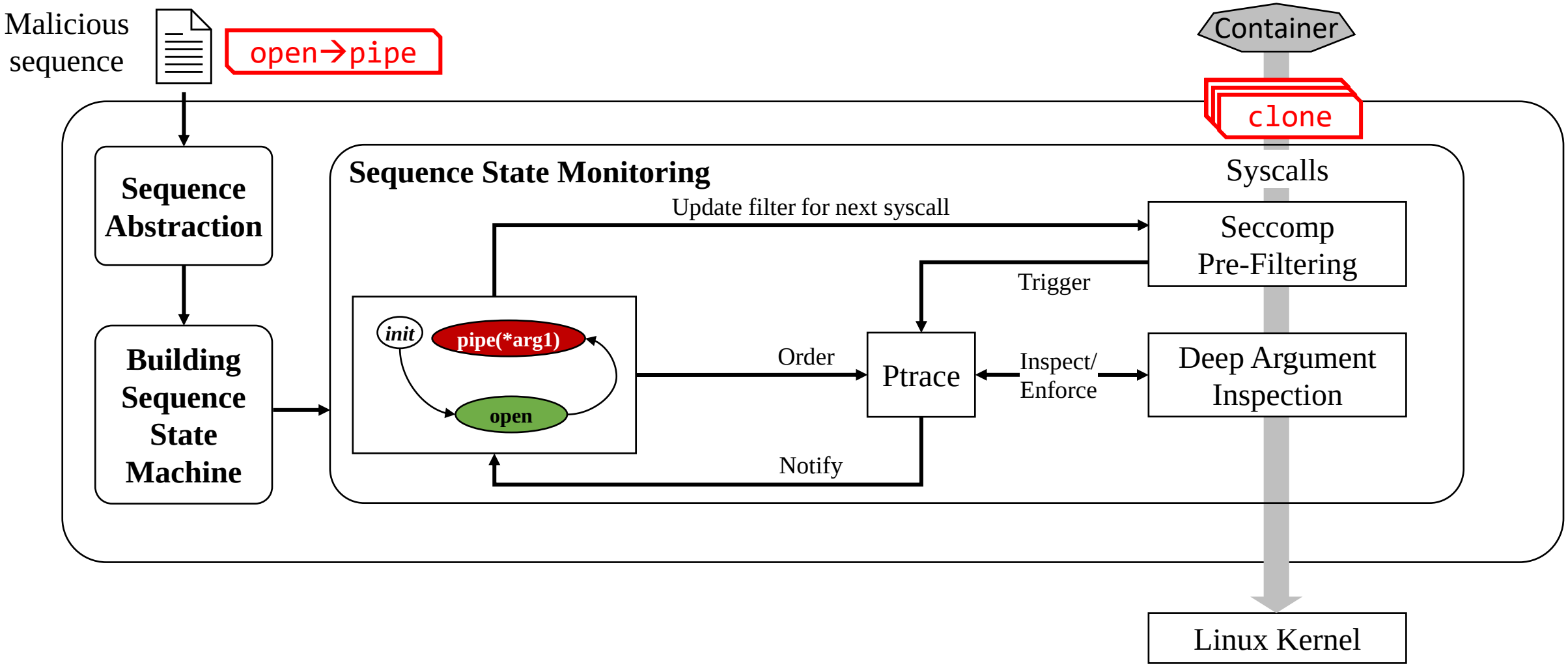


Sequence Extraction

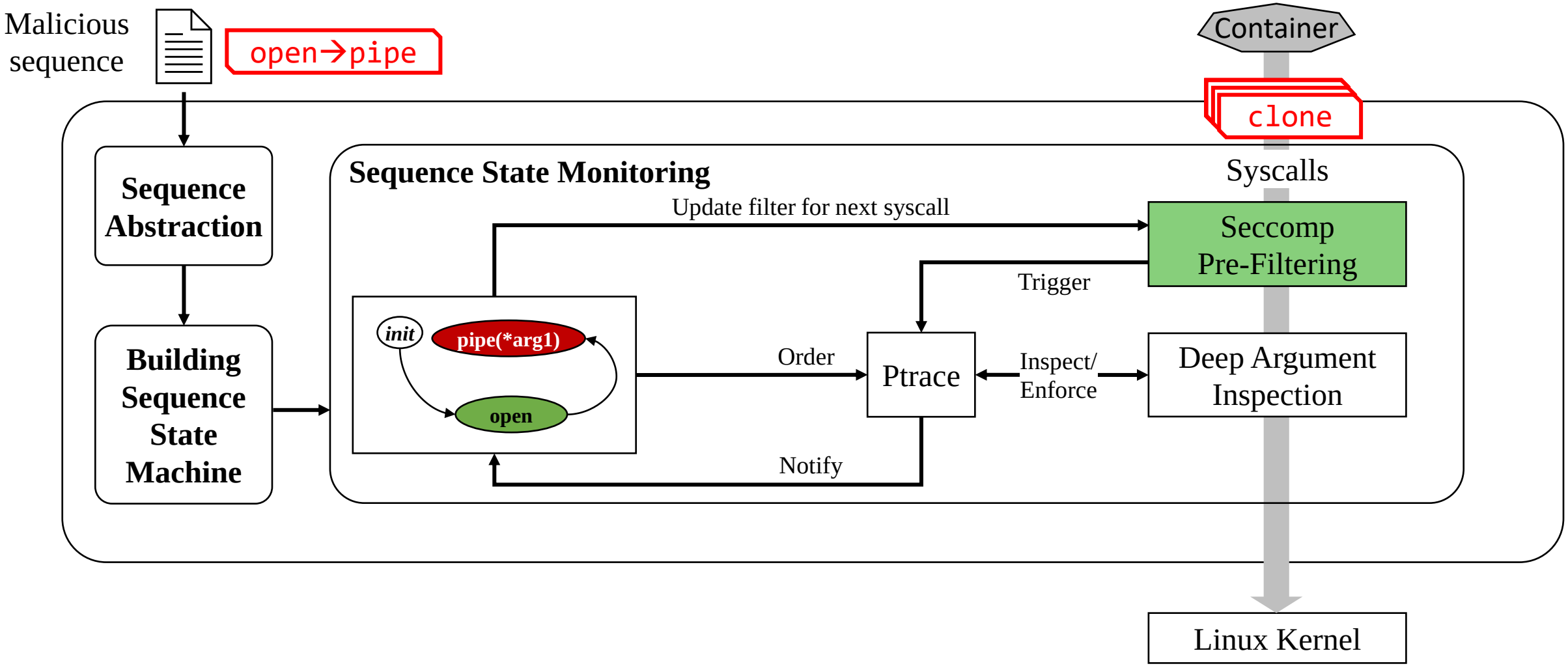
```
...
12 open(“passwd”, RDONLY) = 4
...
12 pipe(5,6) = 0
12 splice(4,6) = 1
13 execve(“su”) = 0
13 open(“passwd”, RDONLY)
```

#	Syscall	PID	Arg1	Arg2	Retcode
1	open	12	passwd	RDONLY	4
2	pipe	12	5	6	0
3	splice	12	4	6	1
4	execve	13	su		0

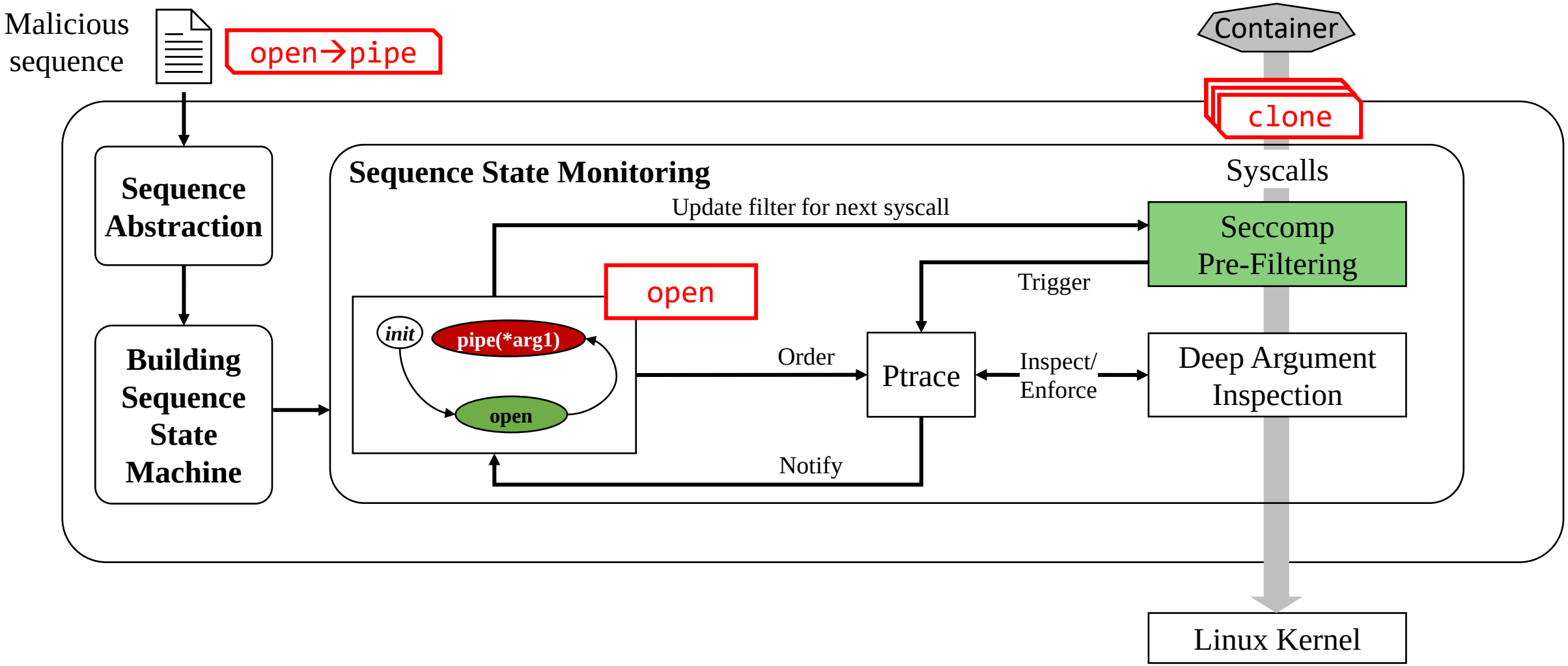
Dynamic Runtime Protection



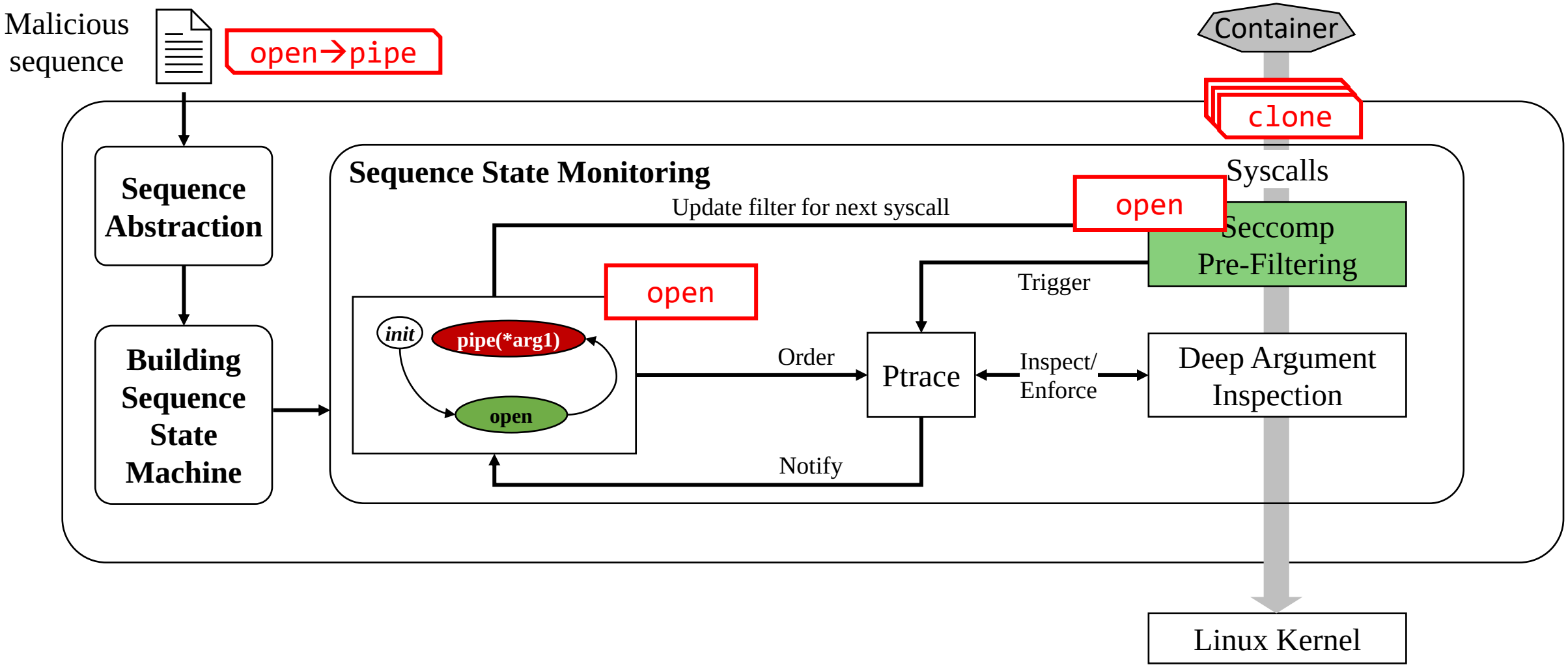
Dynamic Runtime Protection



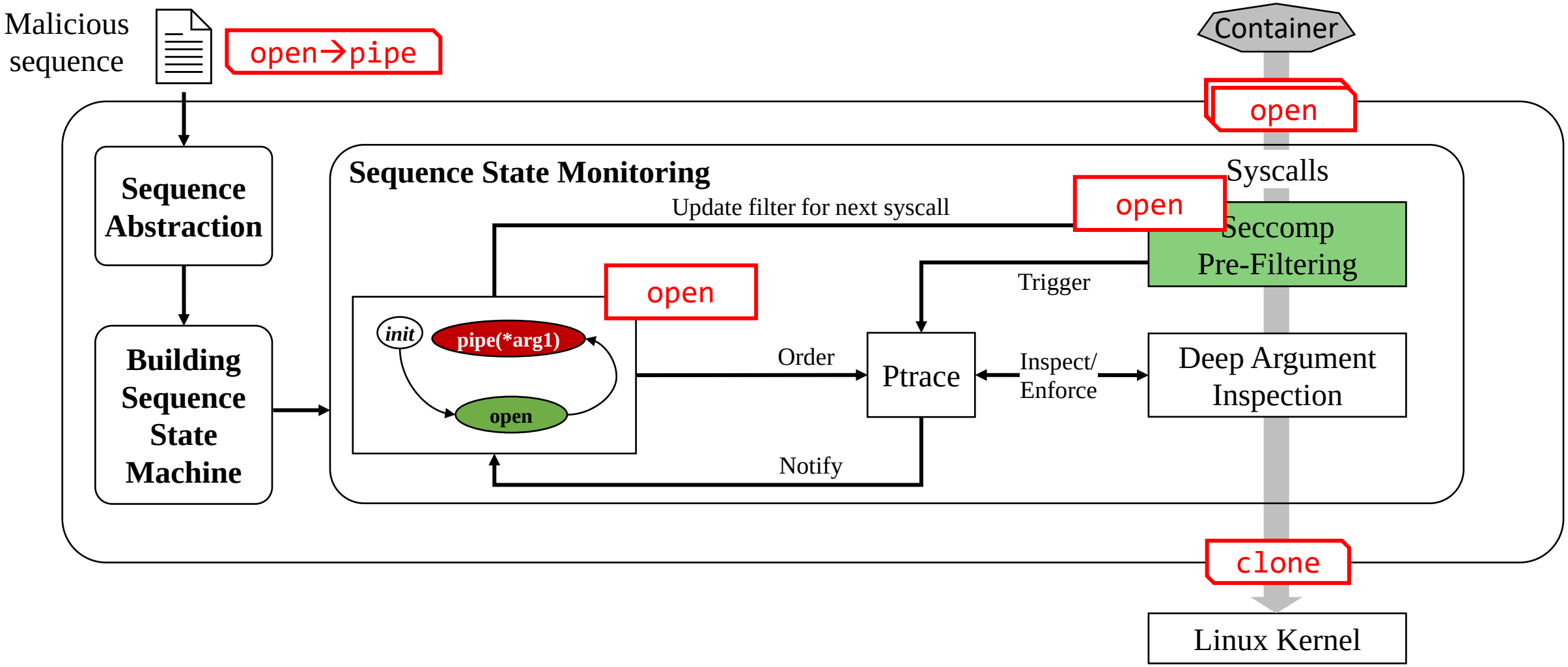
Dynamic Runtime Protection



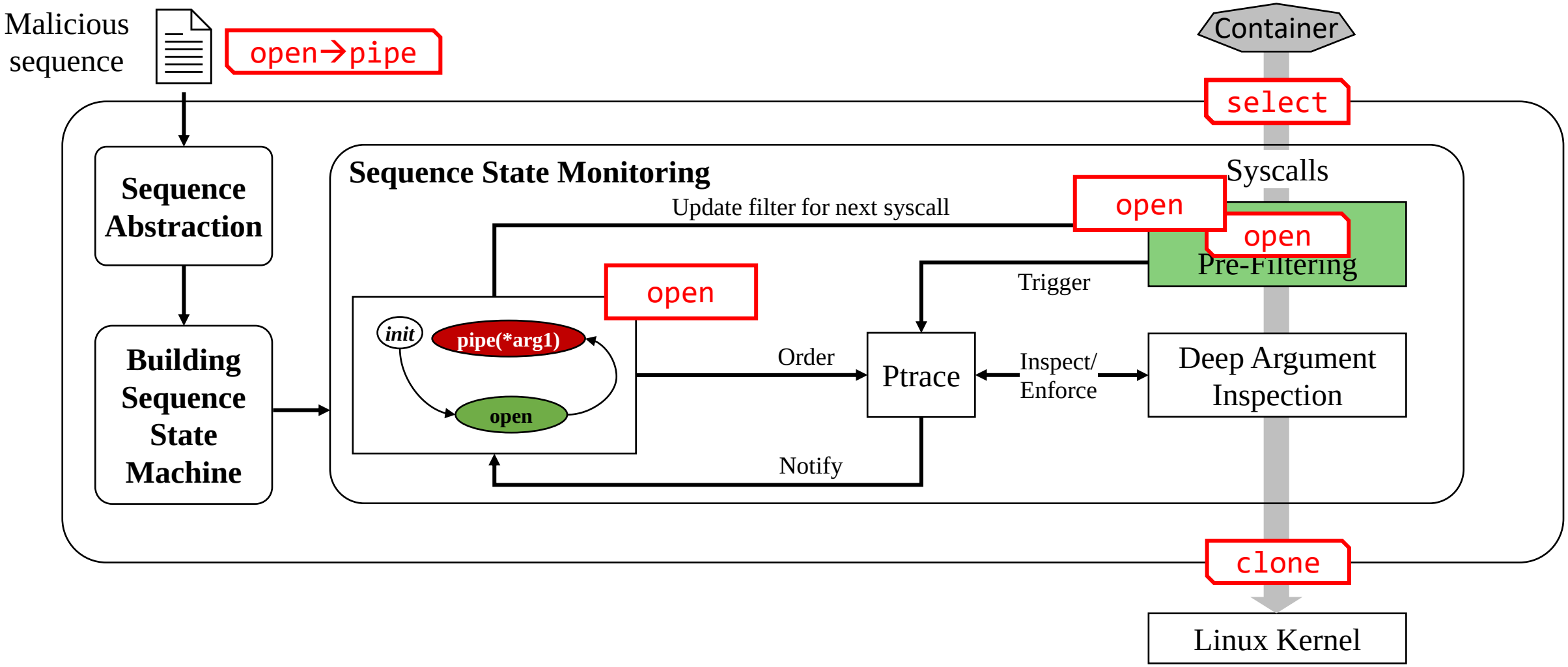
Dynamic Runtime Protection



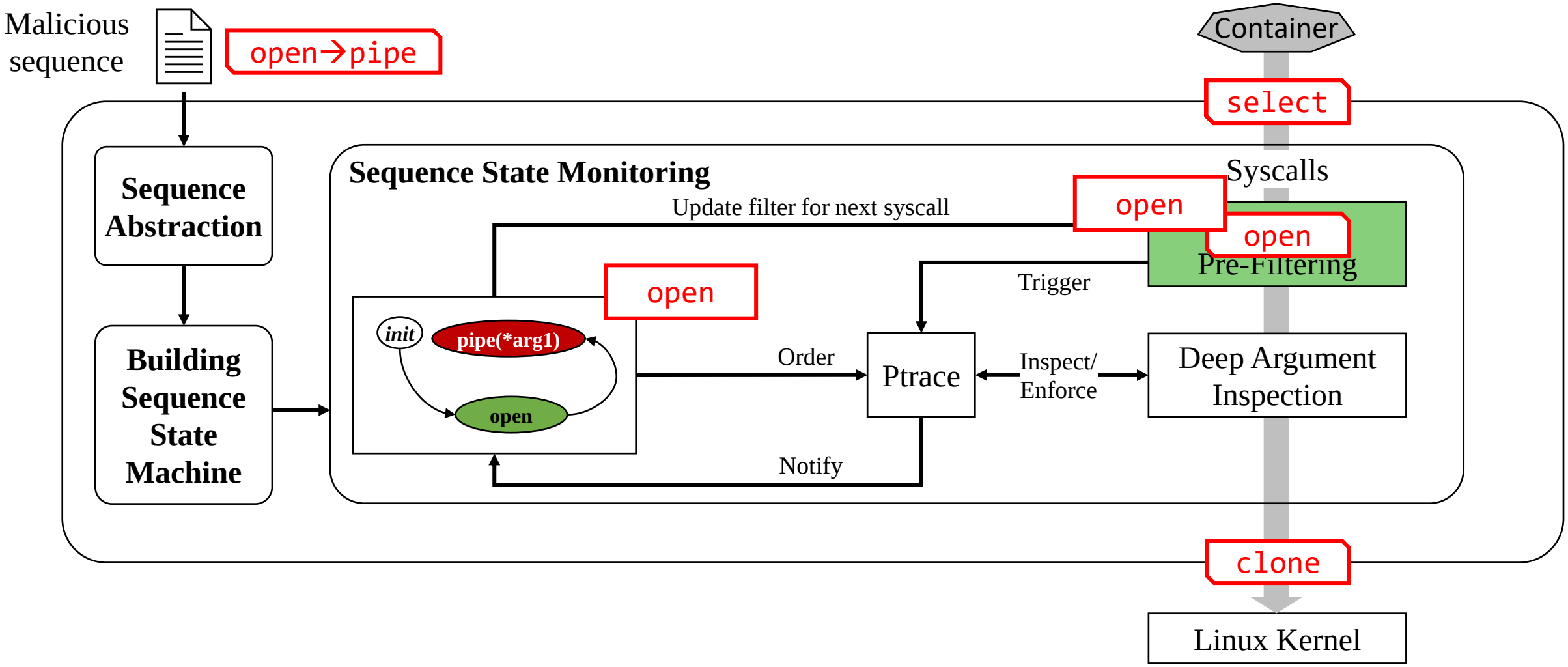
Dynamic Runtime Protection



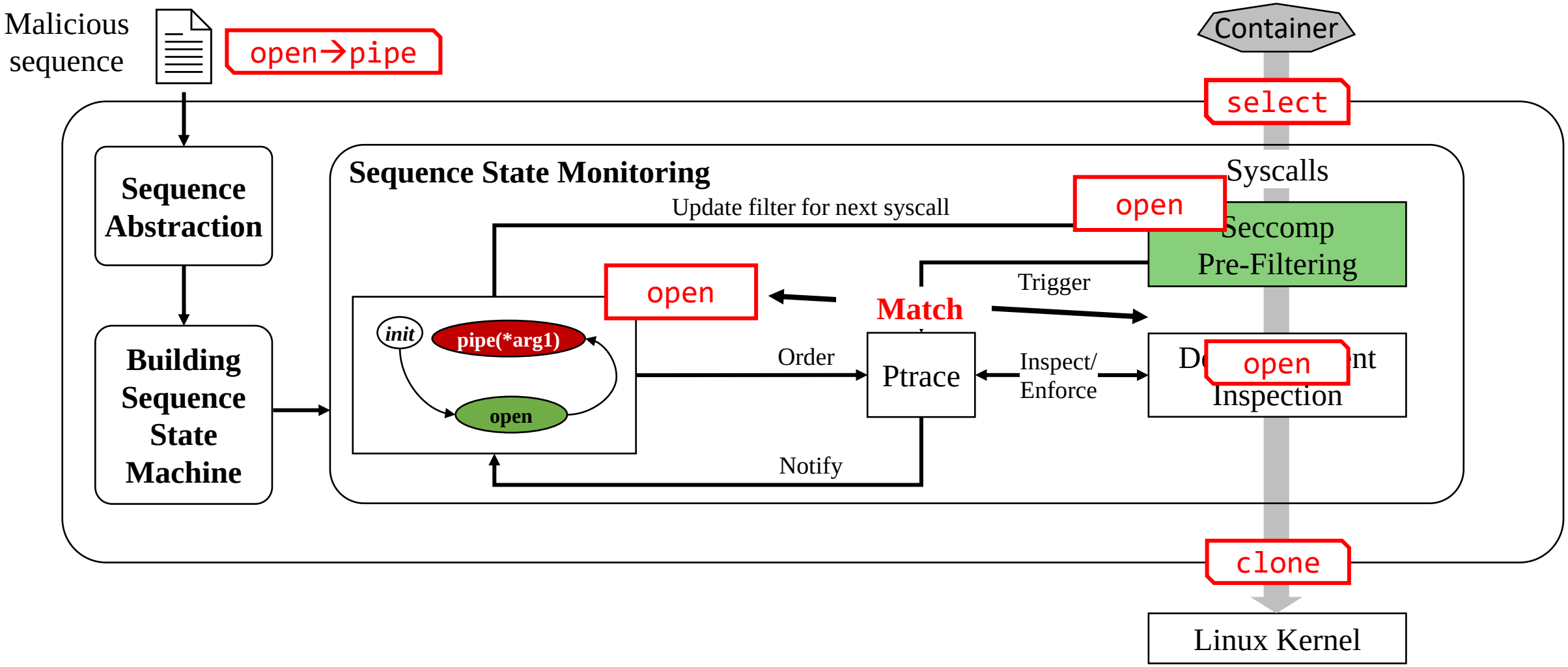
Dynamic Runtime Protection



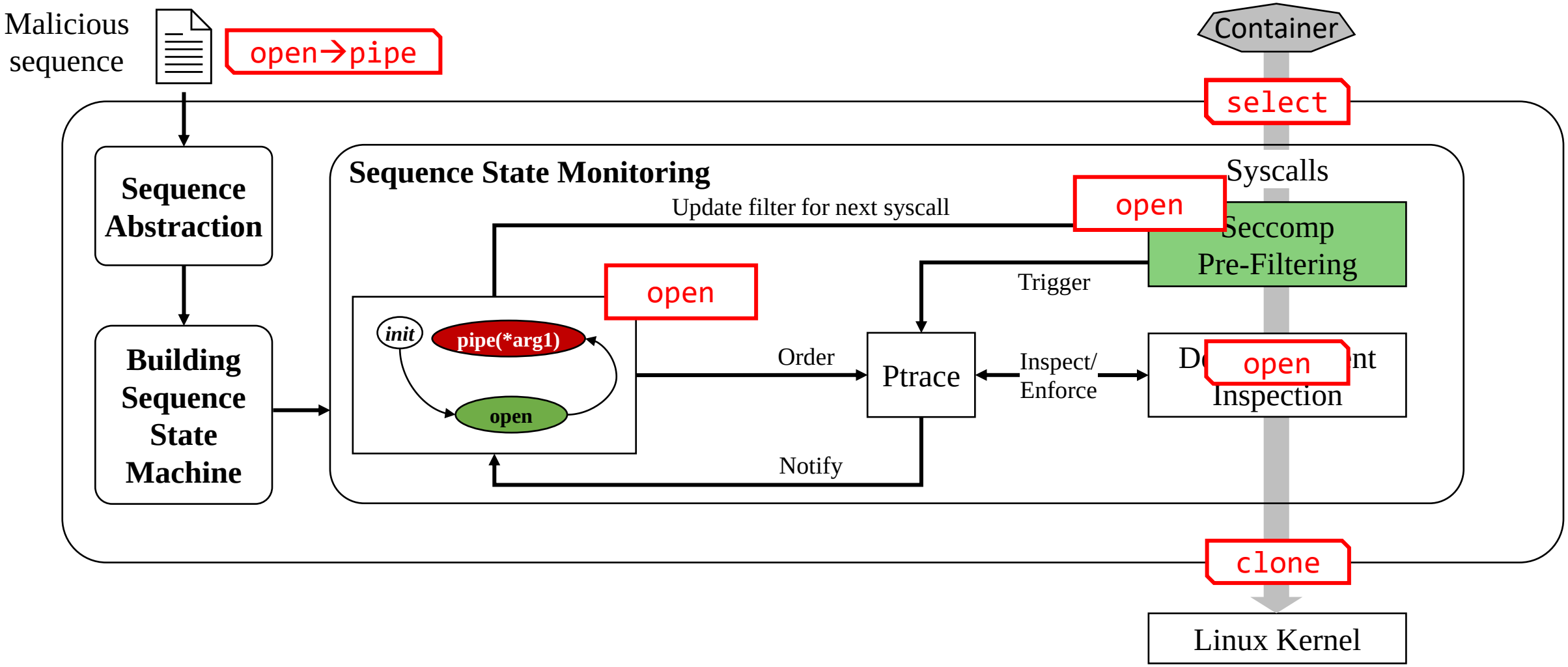
Dynamic Runtime Protection



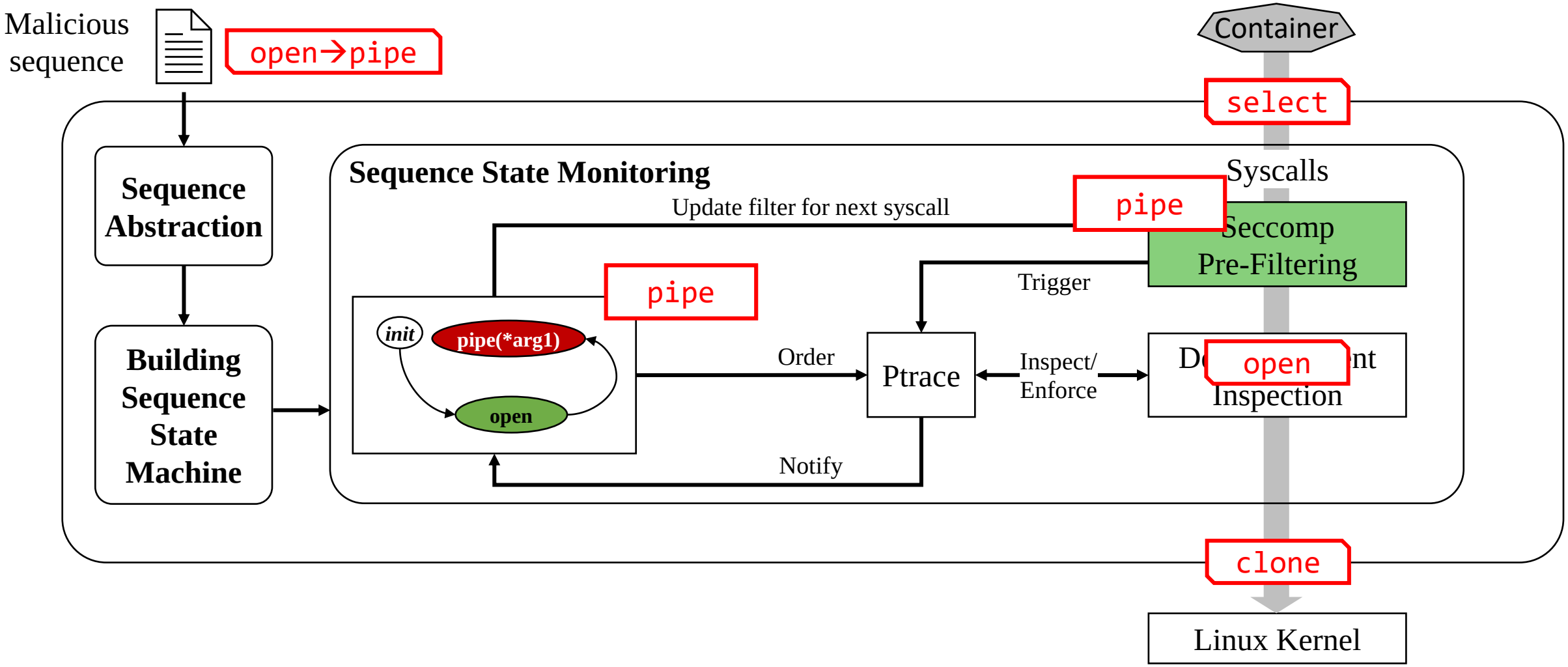
Dynamic Runtime Protection



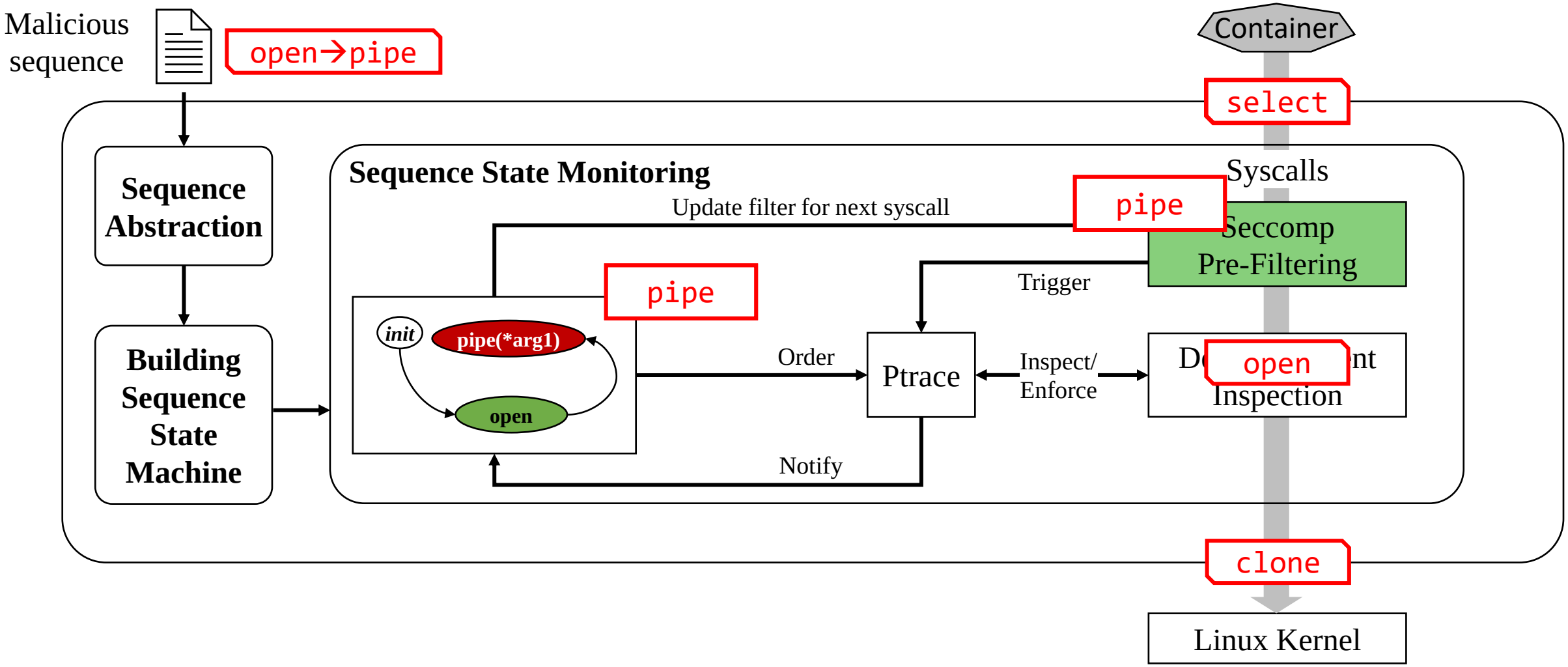
Dynamic Runtime Protection



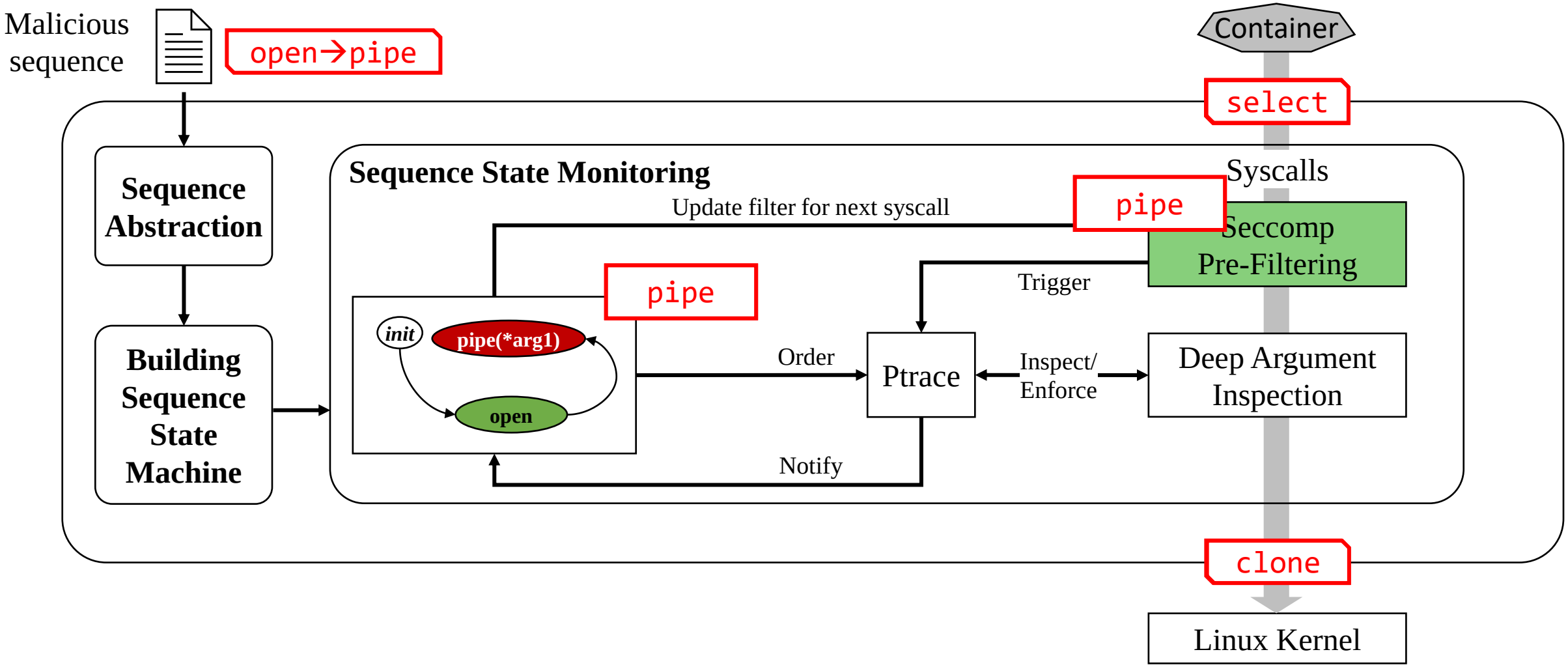
Dynamic Runtime Protection



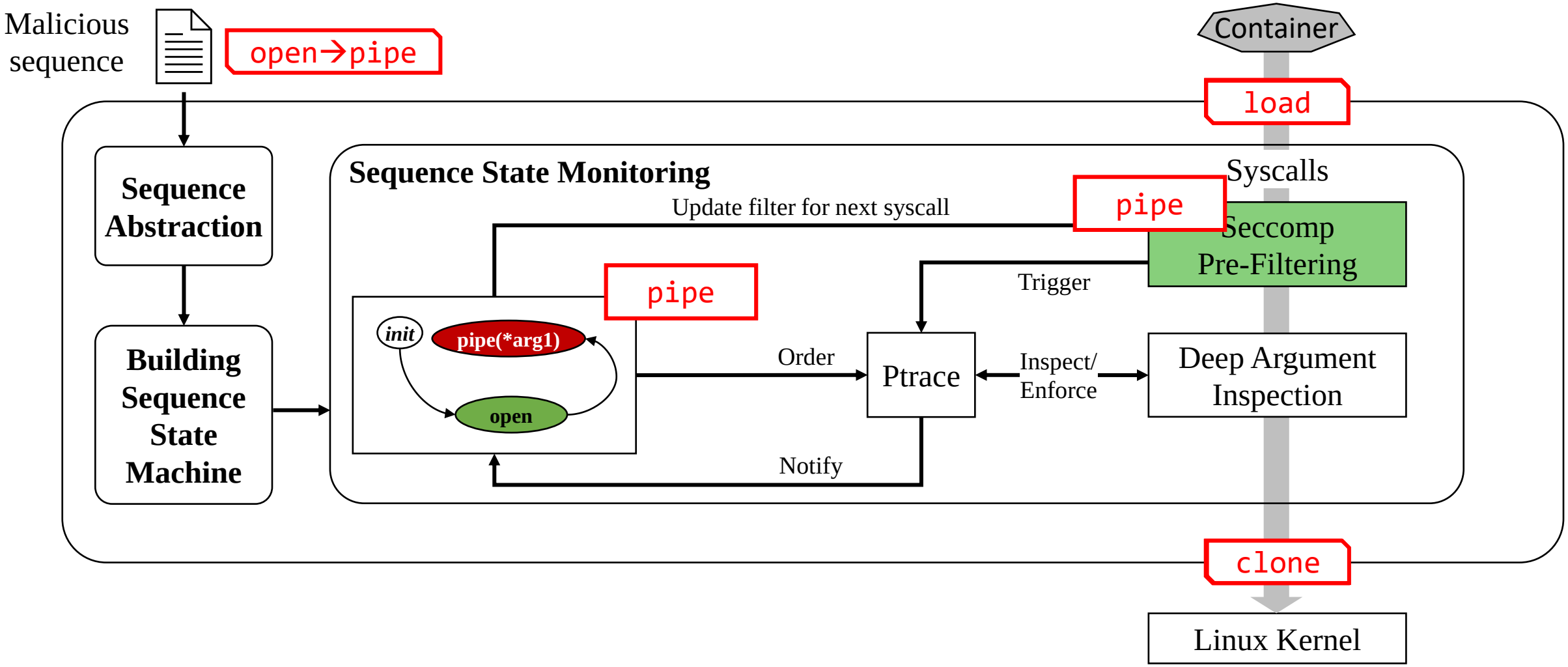
Dynamic Runtime Protection



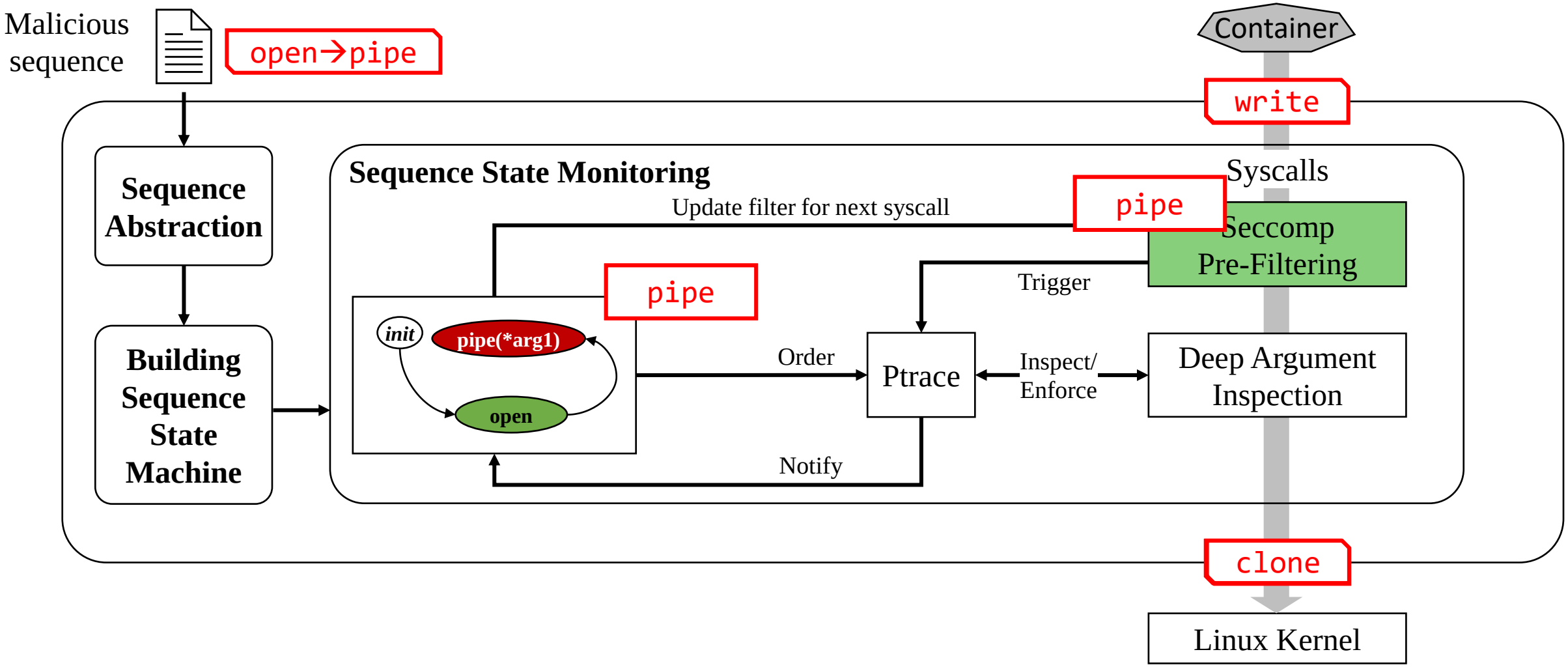
Dynamic Runtime Protection



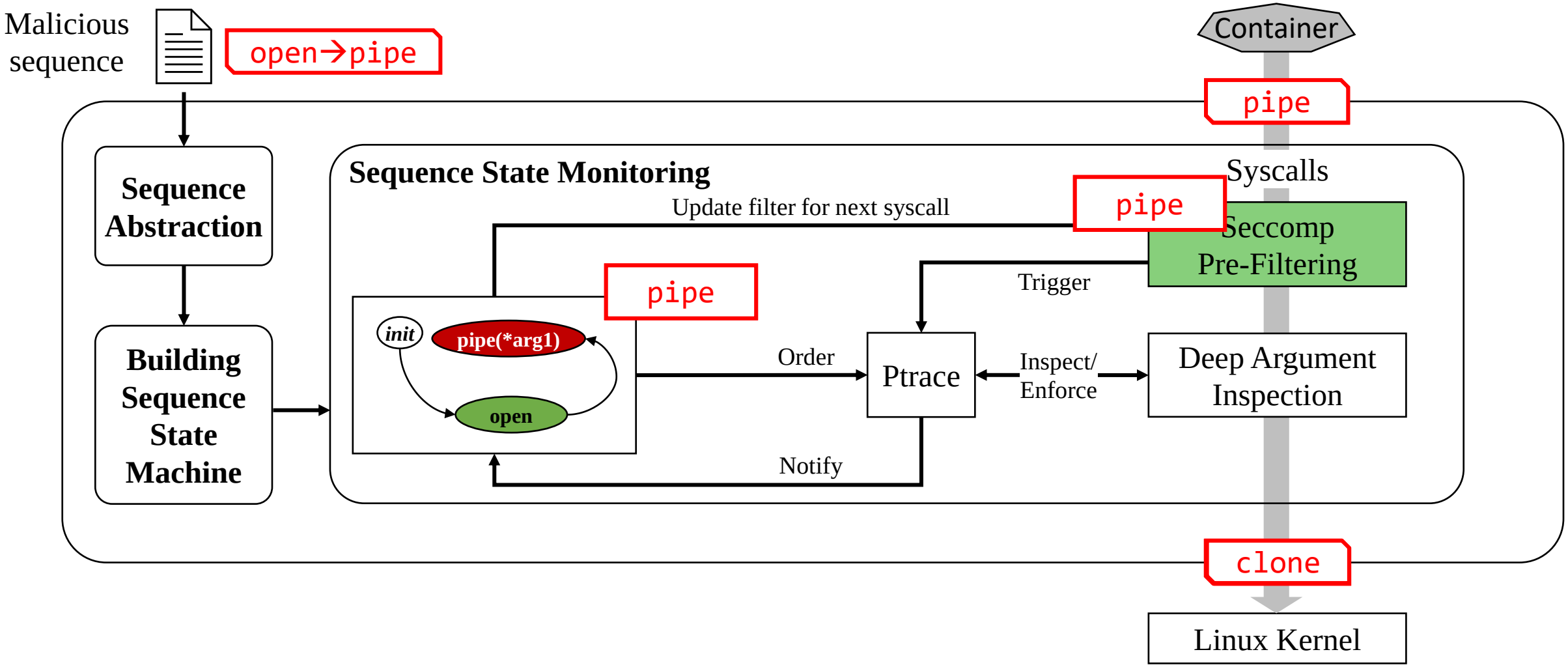
Dynamic Runtime Protection



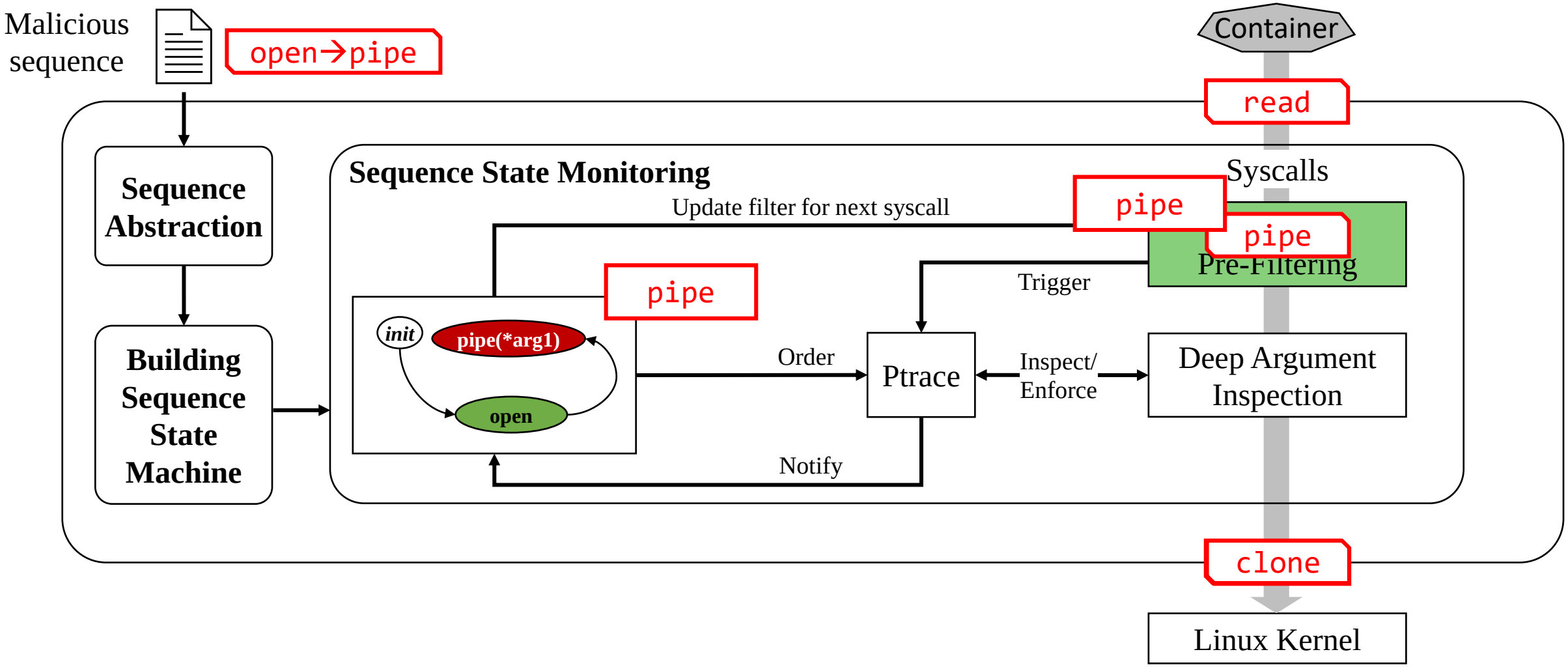
Dynamic Runtime Protection



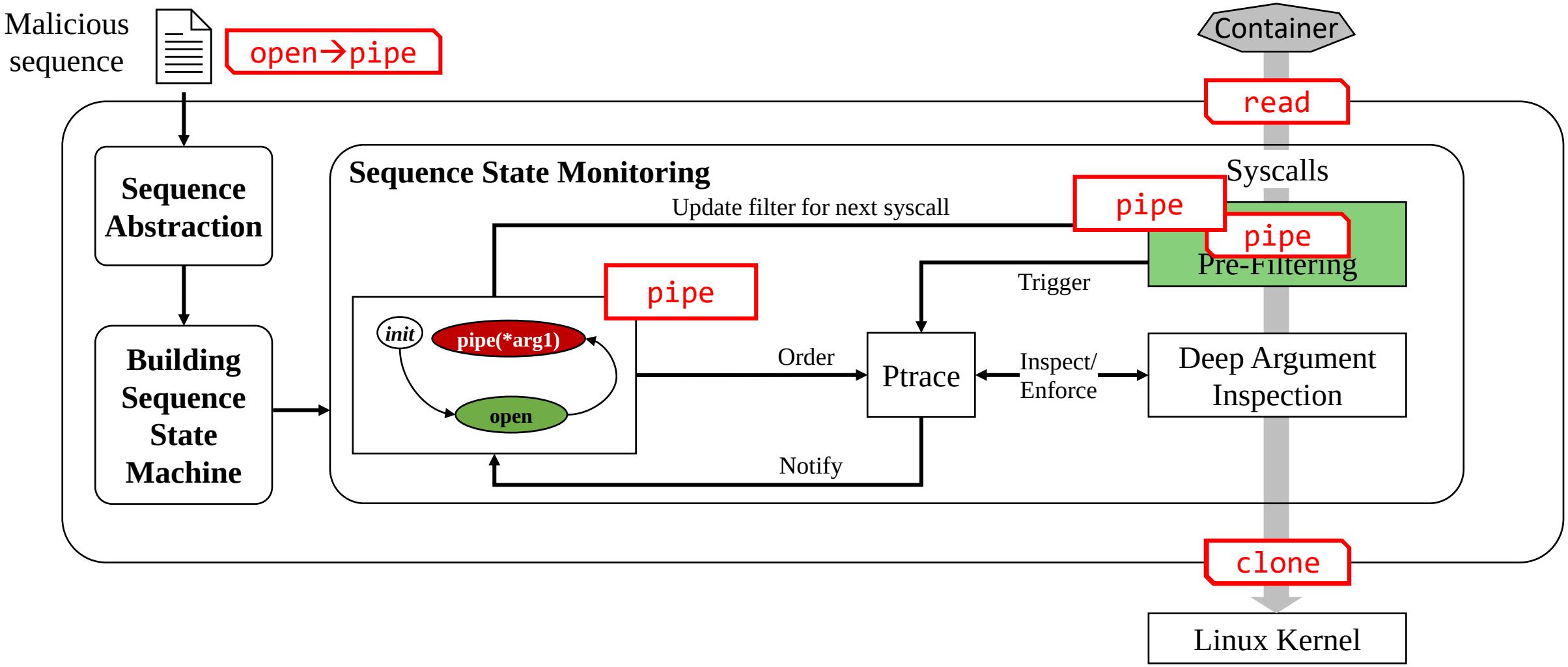
Dynamic Runtime Protection



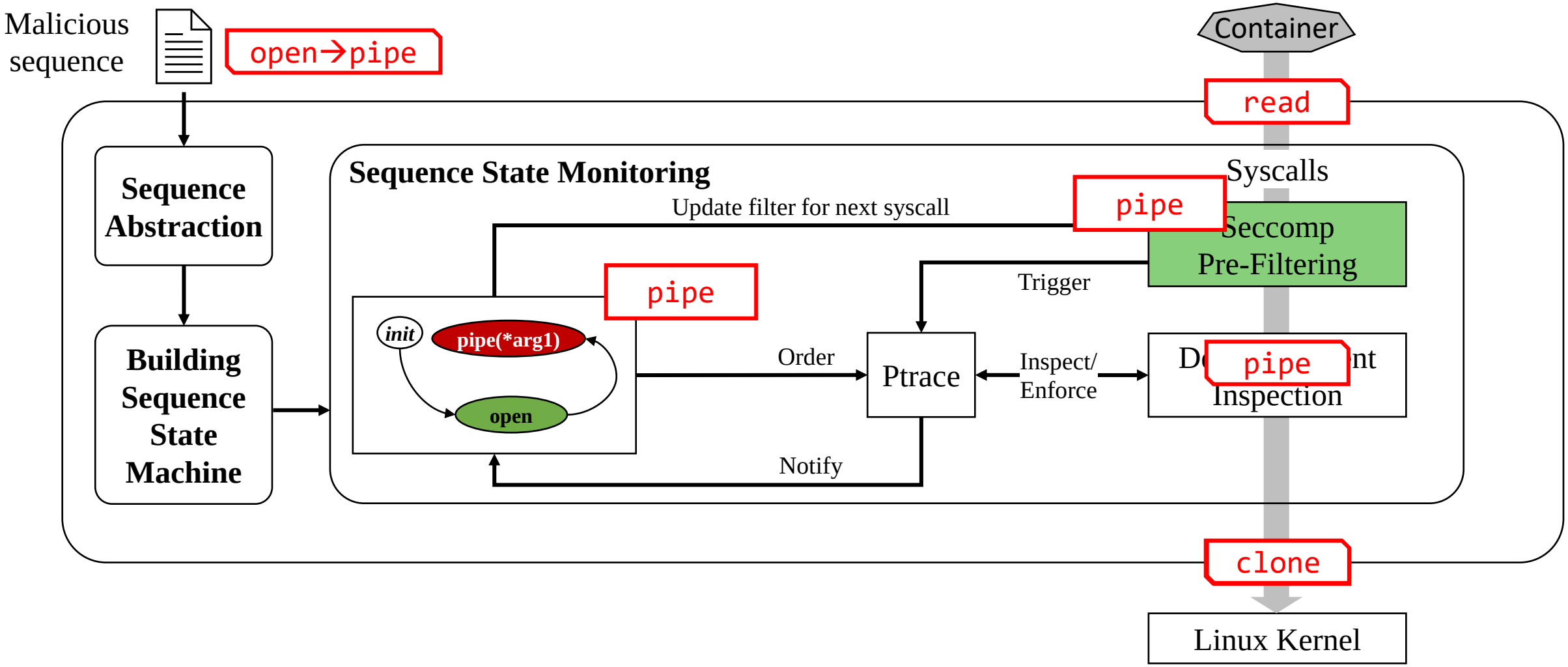
Dynamic Runtime Protection



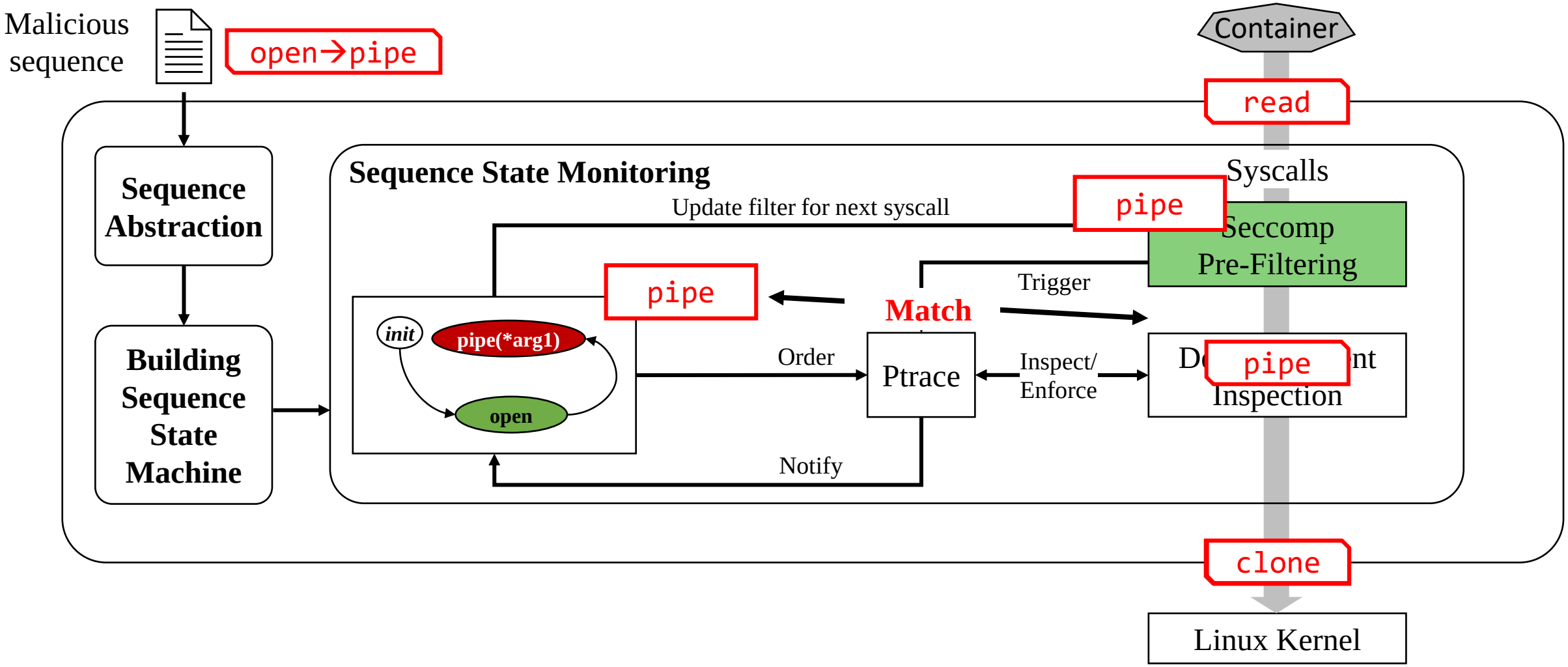
Dynamic Runtime Protection



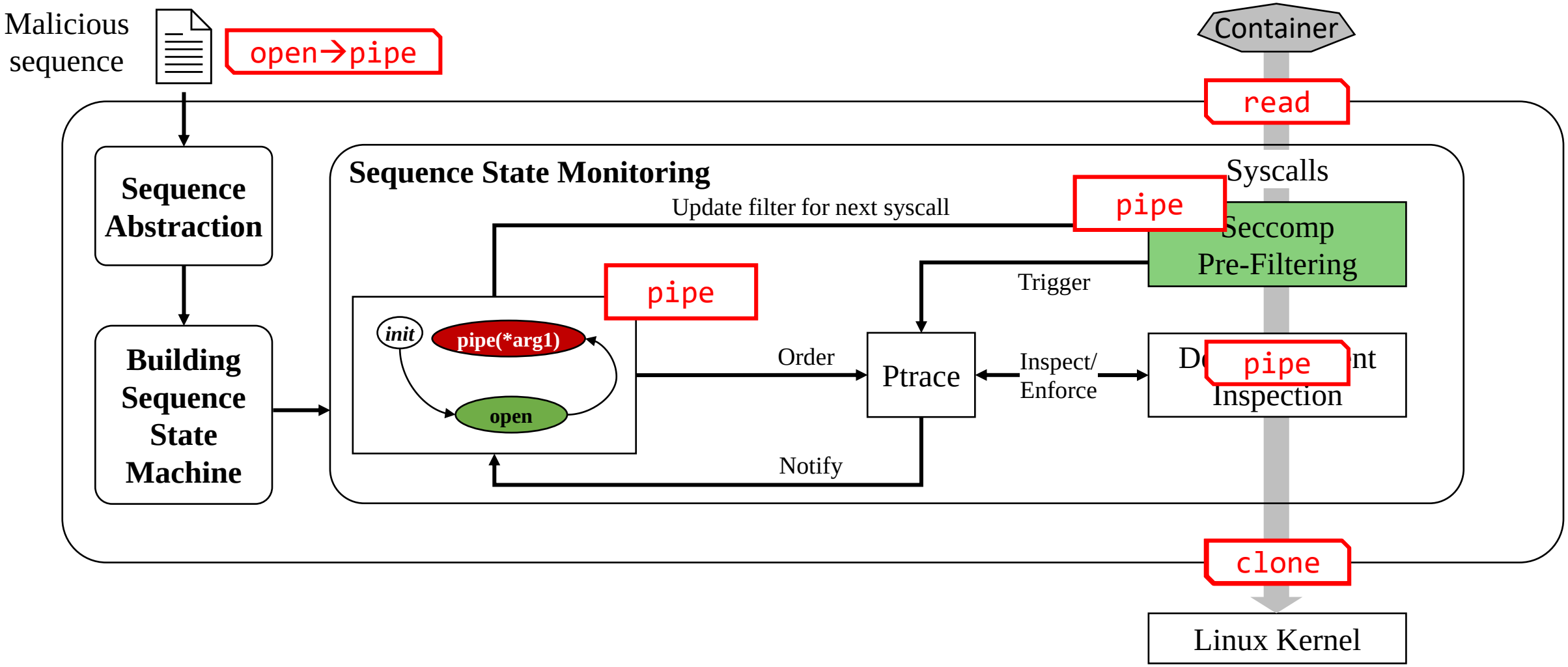
Dynamic Runtime Protection



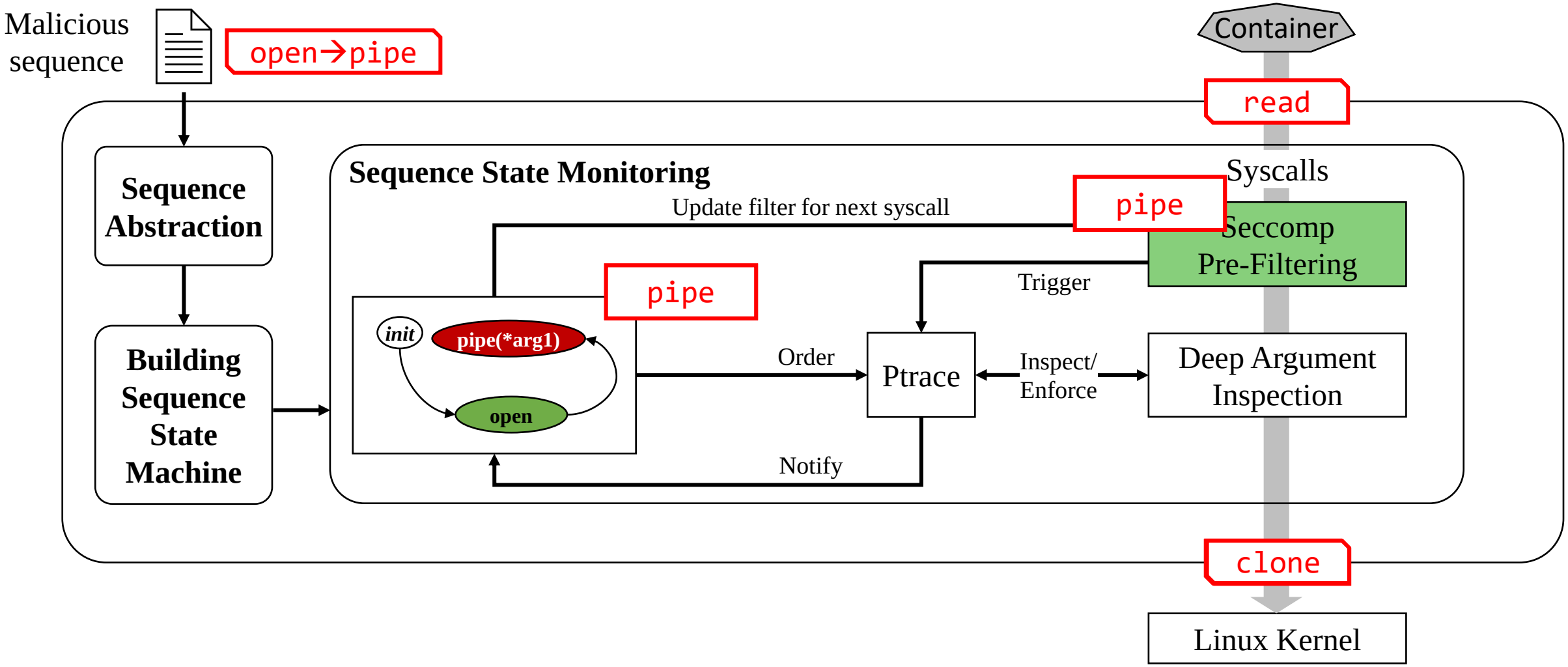
Dynamic Runtime Protection



Dynamic Runtime Protection



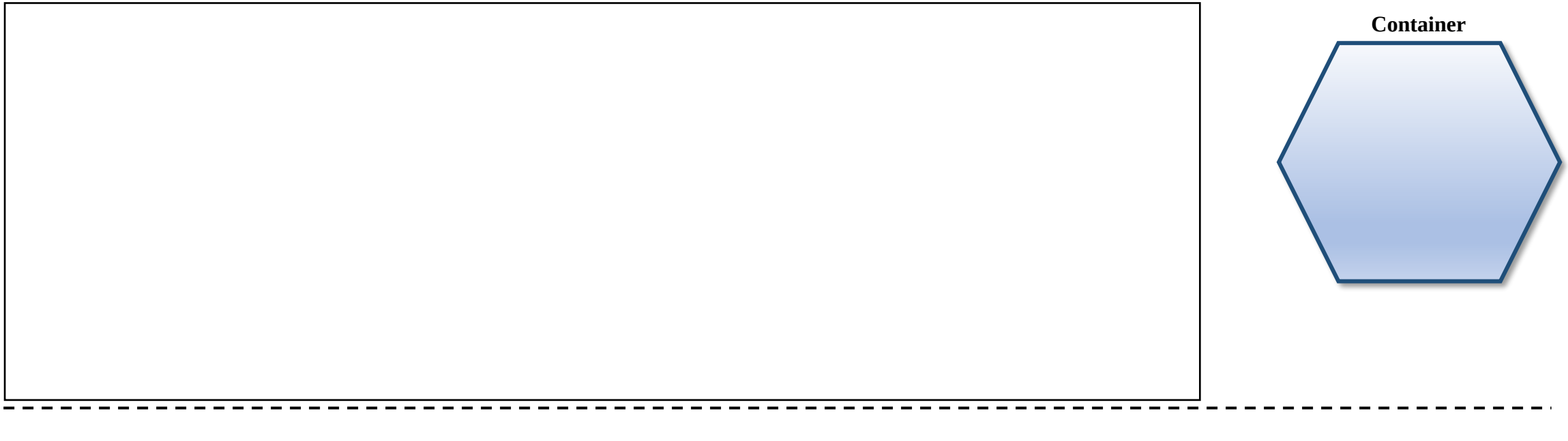
Dynamic Runtime Protection



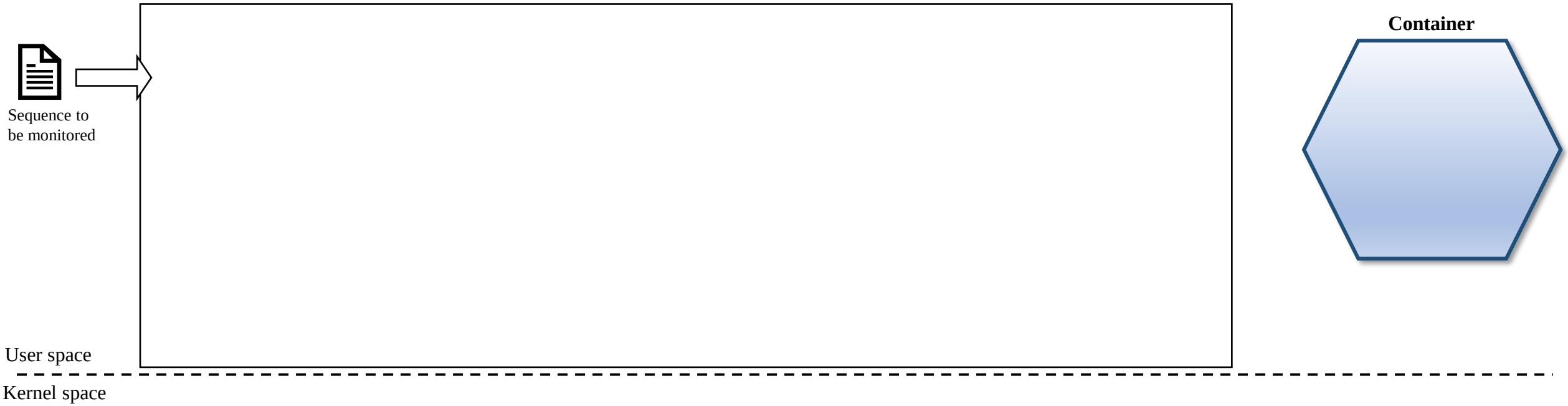
Outline

- Intro
  - Motivation
  - Related Work
- Methodology
  - Key Ideas & Overview
  - Malicious Sequence Identification
  - Dynamic Runtime Protection
- **Implementation**
- Experimental Results
  - Security
  - Performance
  - Provenance Analysis
- Conclusion

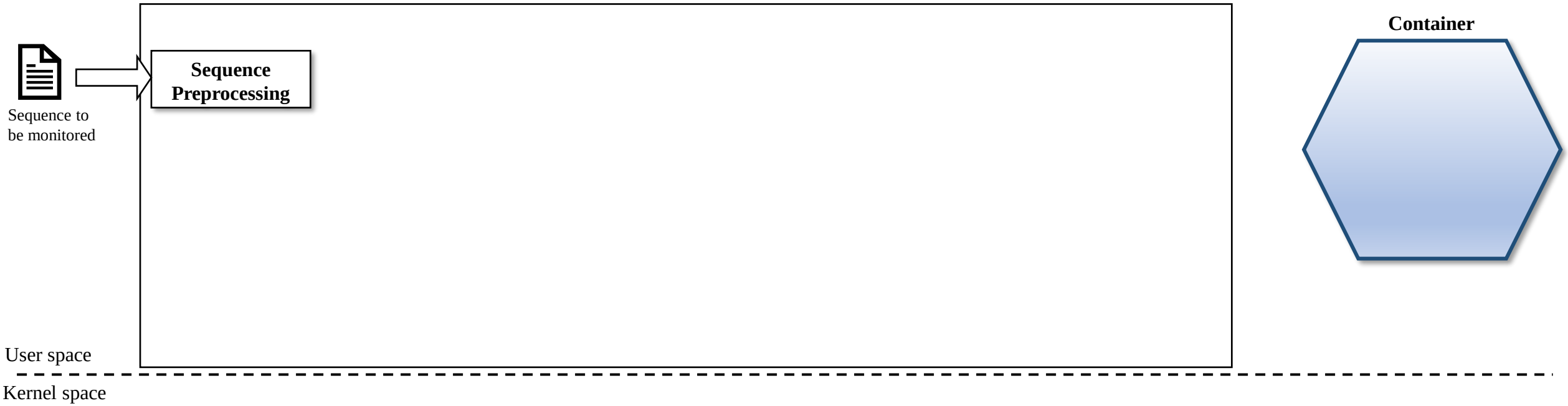
Implementation of Phoenix's Dynamic Runtime Protection



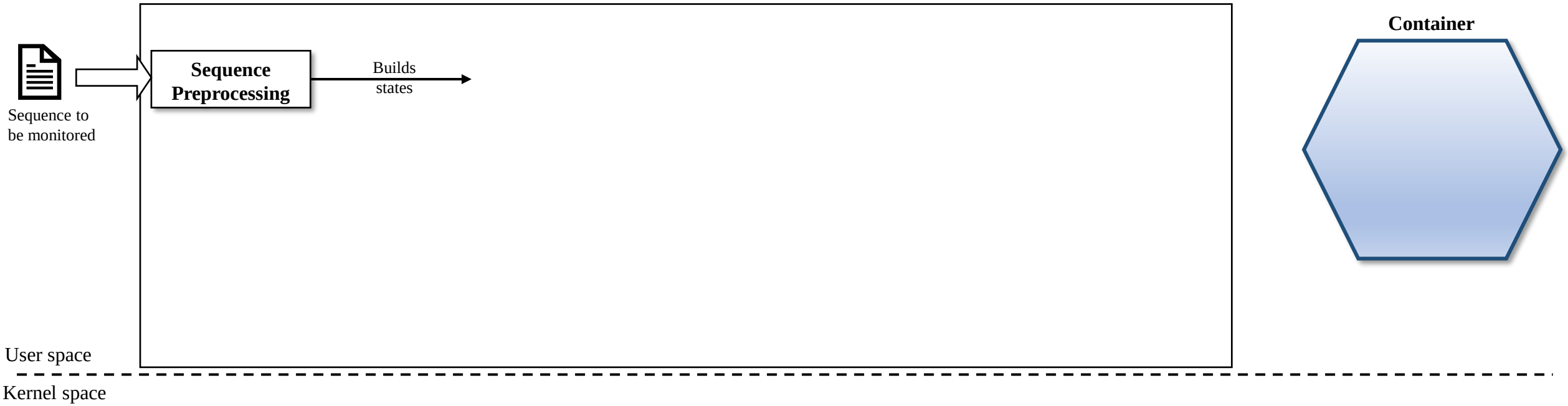
Implementation of Phoenix's Dynamic Runtime Protection



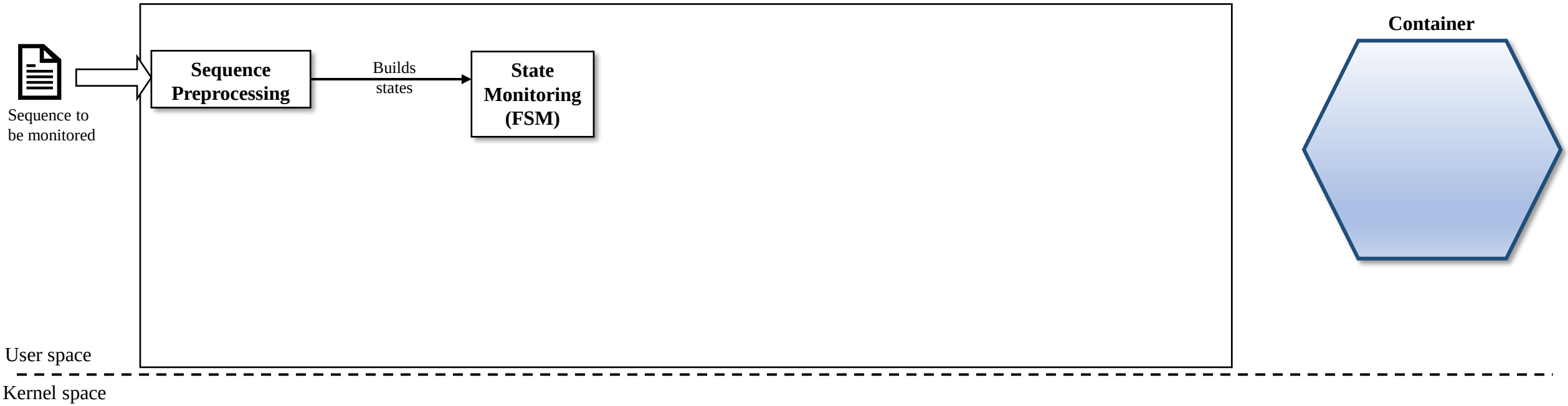
Implementation of Phoenix's Dynamic Runtime Protection



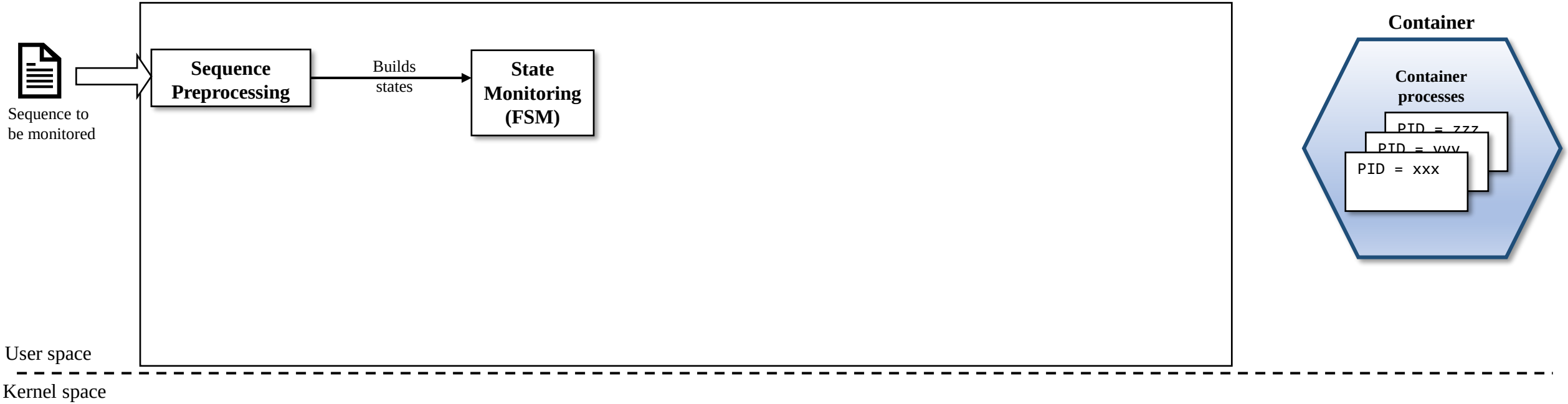
Implementation of Phoenix's Dynamic Runtime Protection



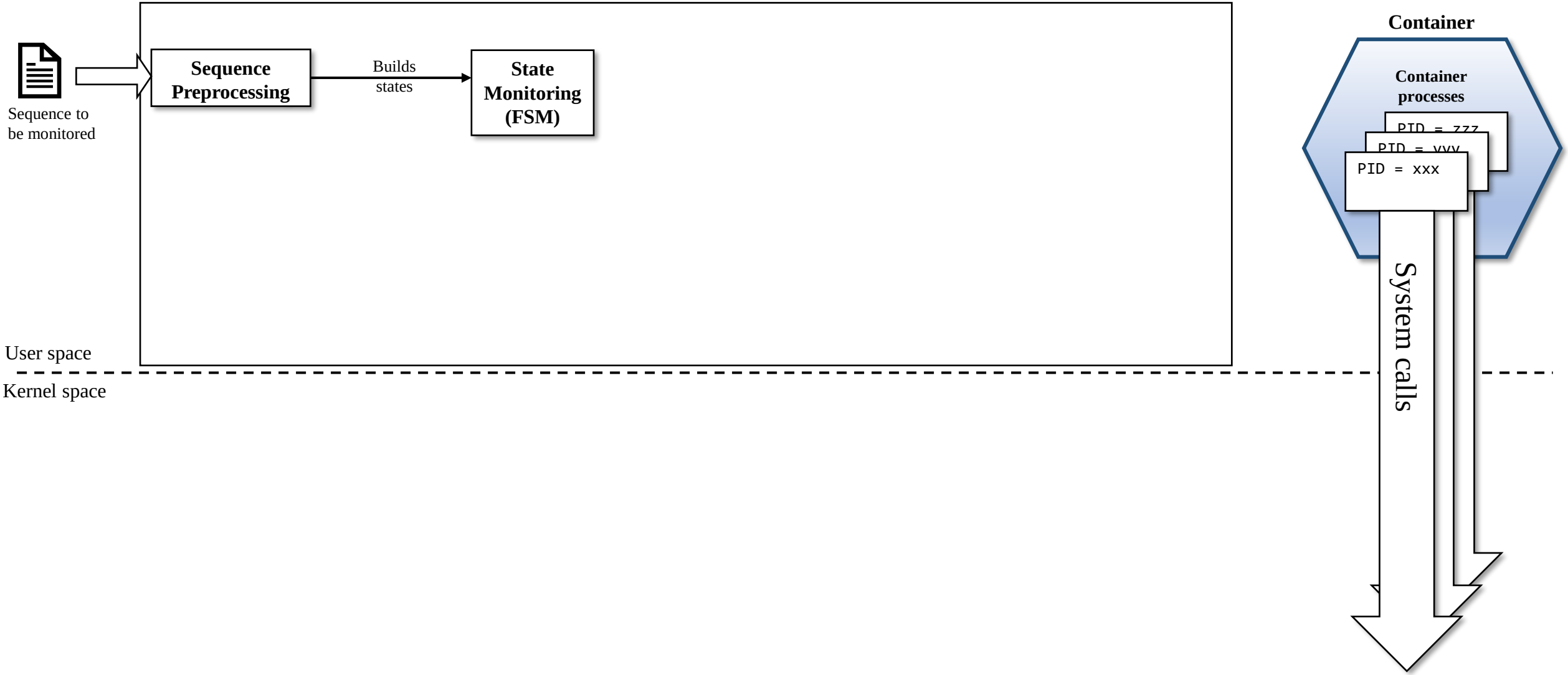
Implementation of Phoenix's Dynamic Runtime Protection



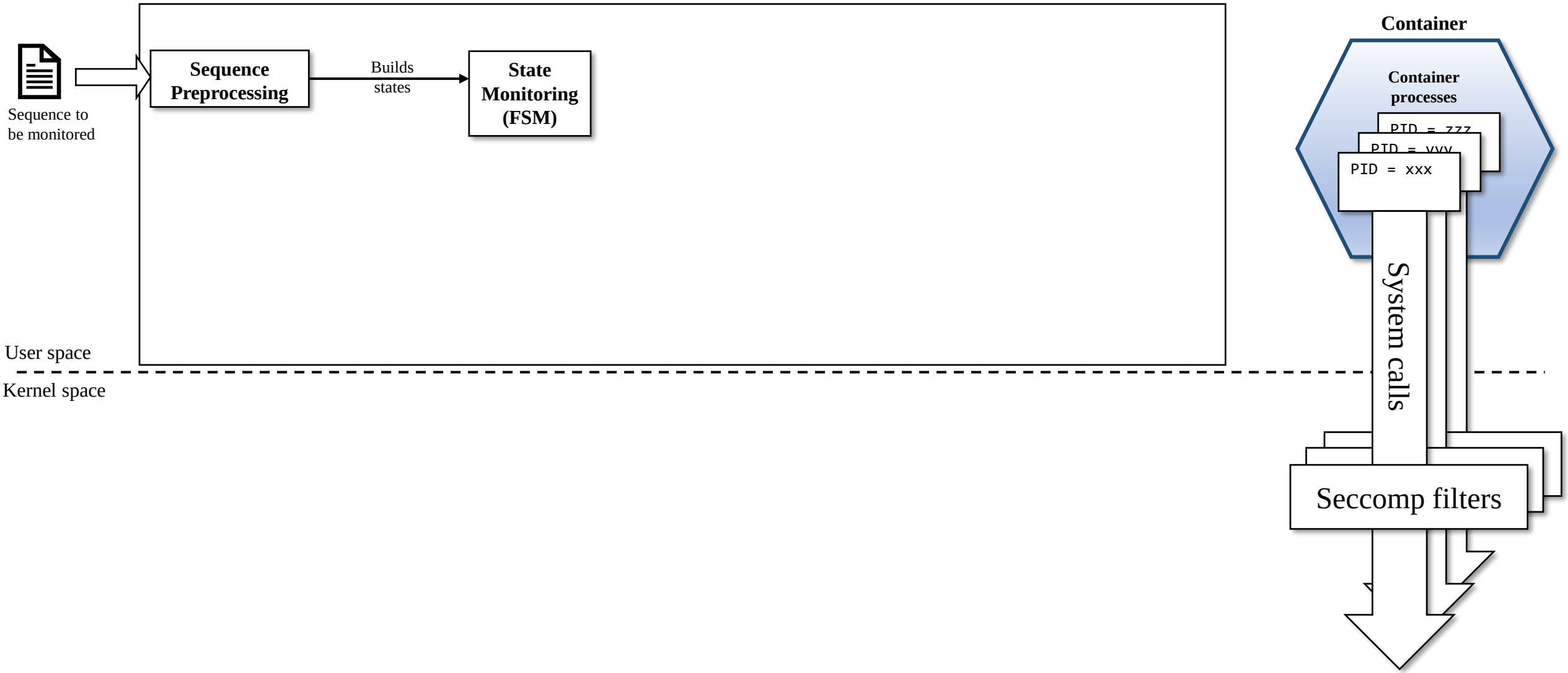
Implementation of Phoenix's Dynamic Runtime Protection



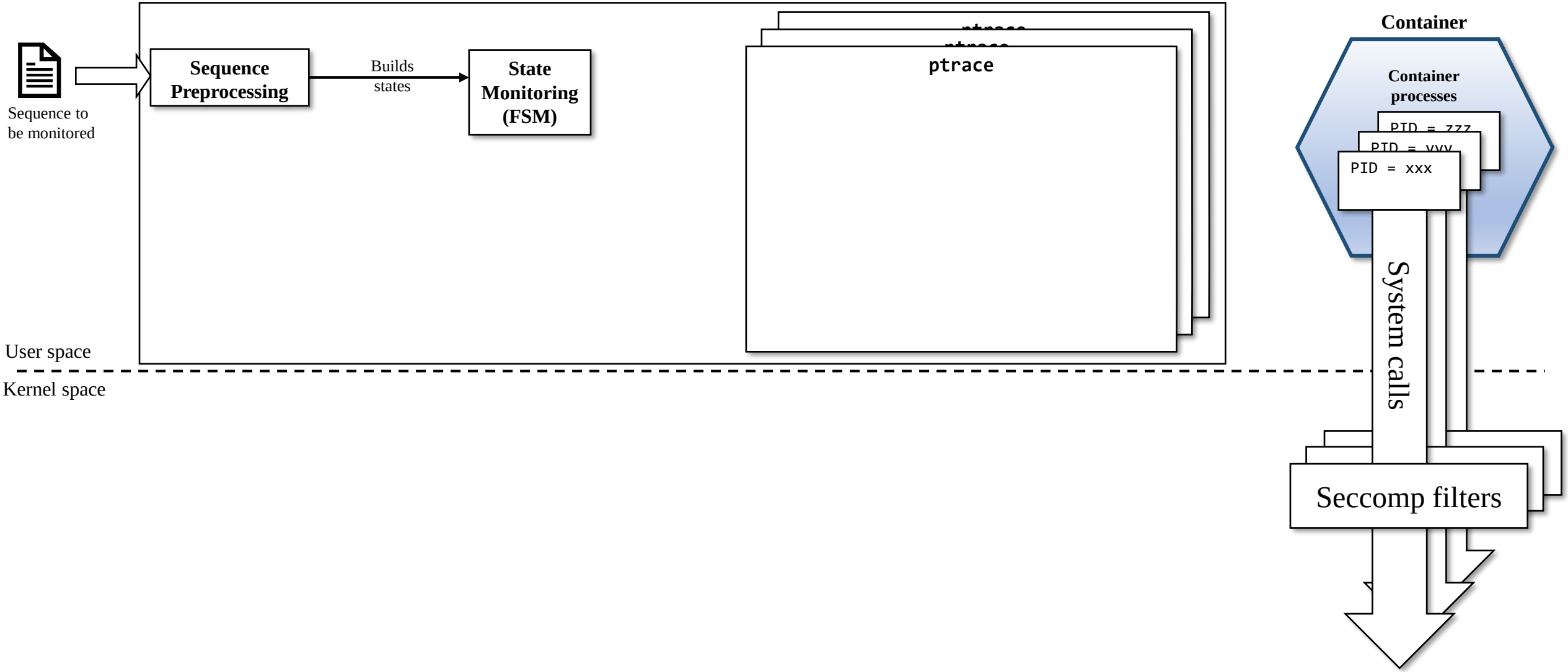
Implementation of Phoenix's Dynamic Runtime Protection



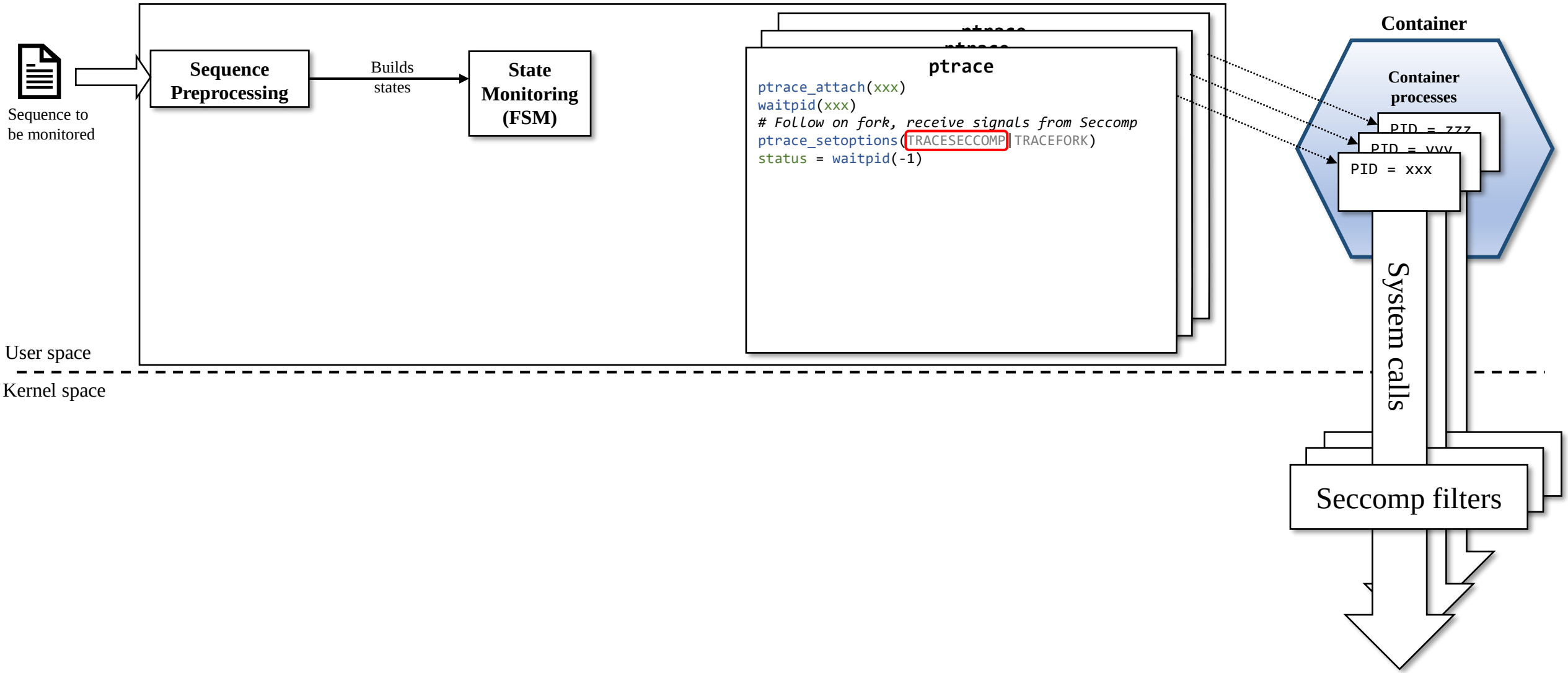
Implementation of Phoenix's Dynamic Runtime Protection



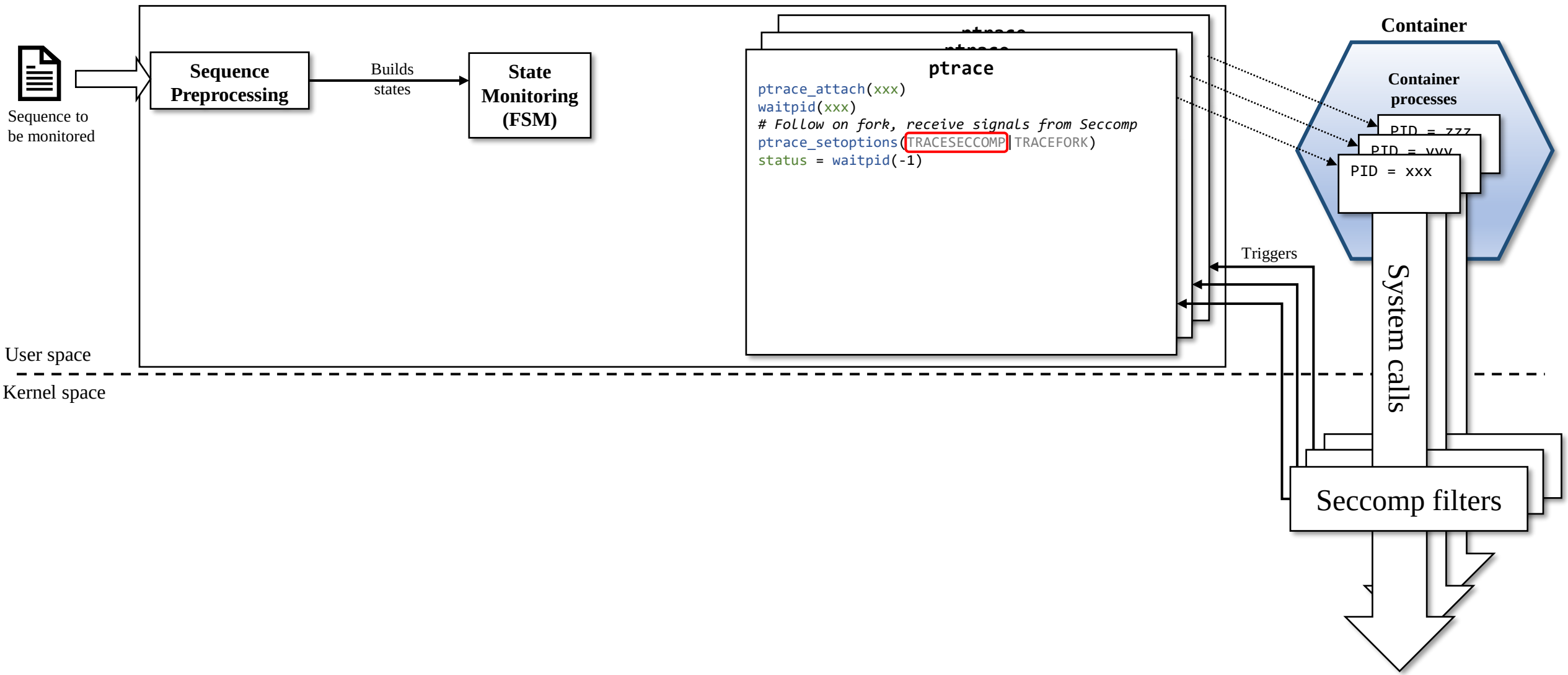
Implementation of Phoenix's Dynamic Runtime Protection



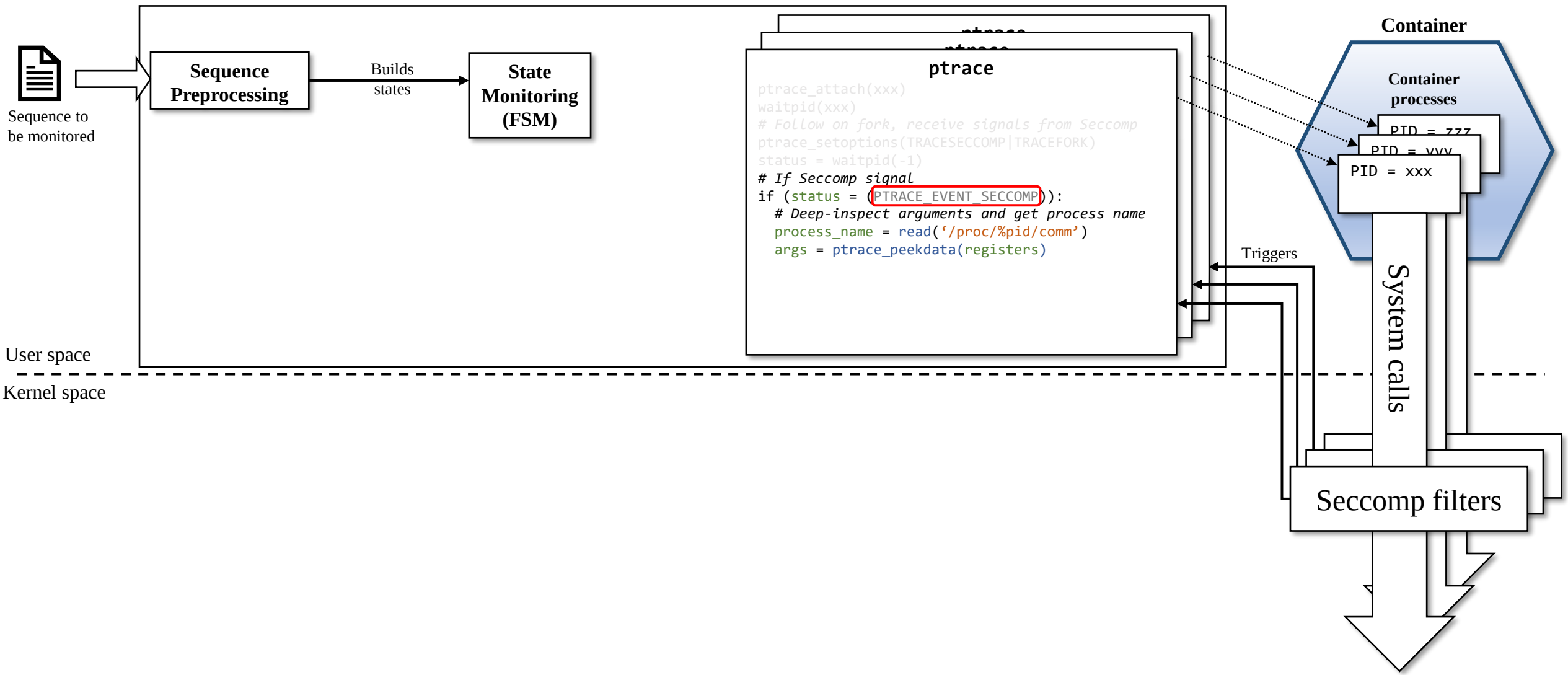
Implementation of Phoenix's Dynamic Runtime Protection



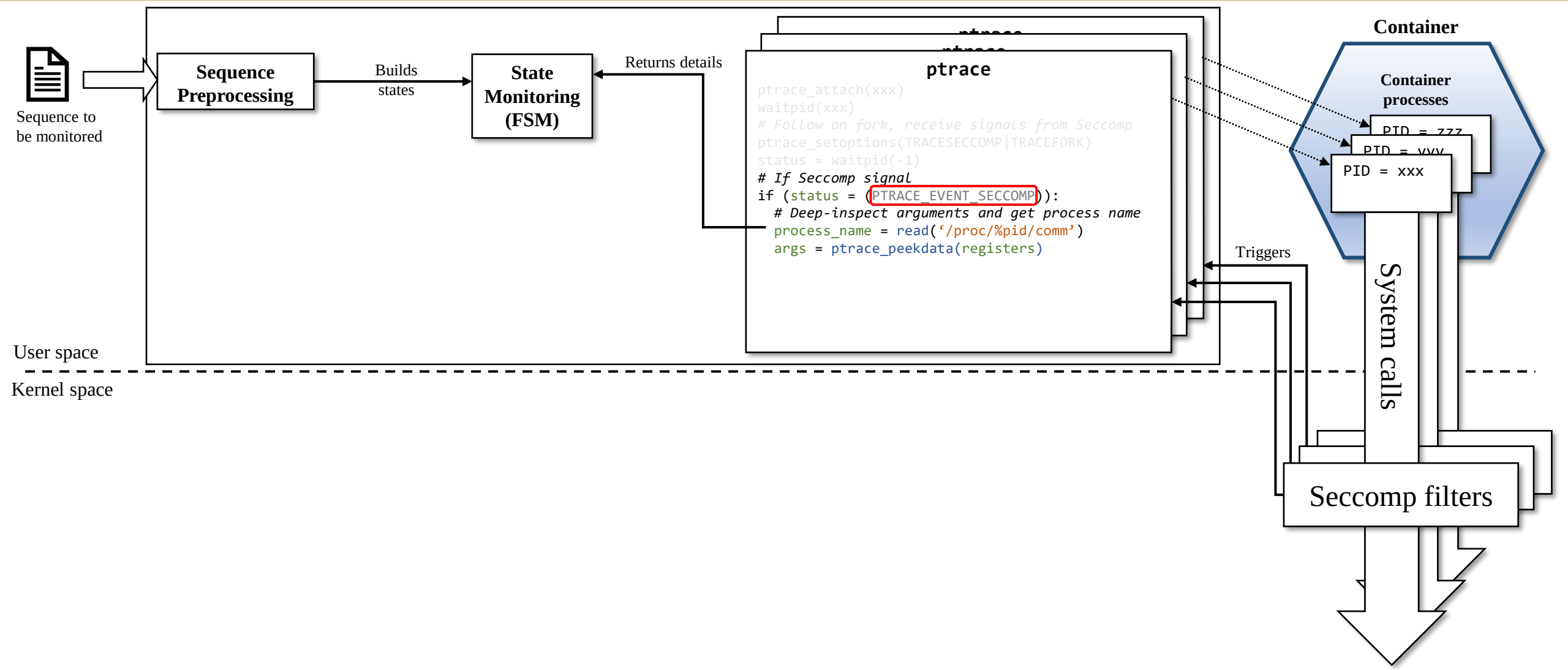
Implementation of Phoenix's Dynamic Runtime Protection



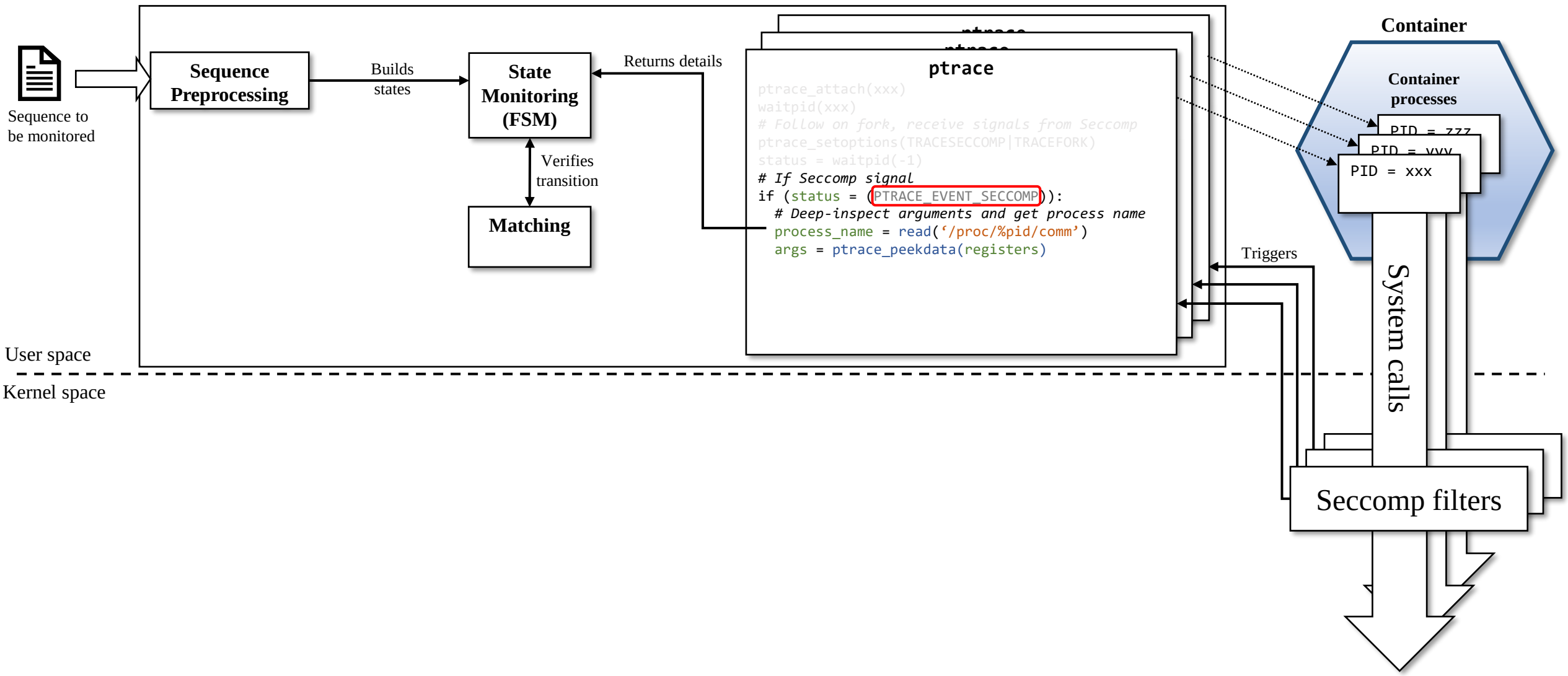
Implementation of Phoenix's Dynamic Runtime Protection



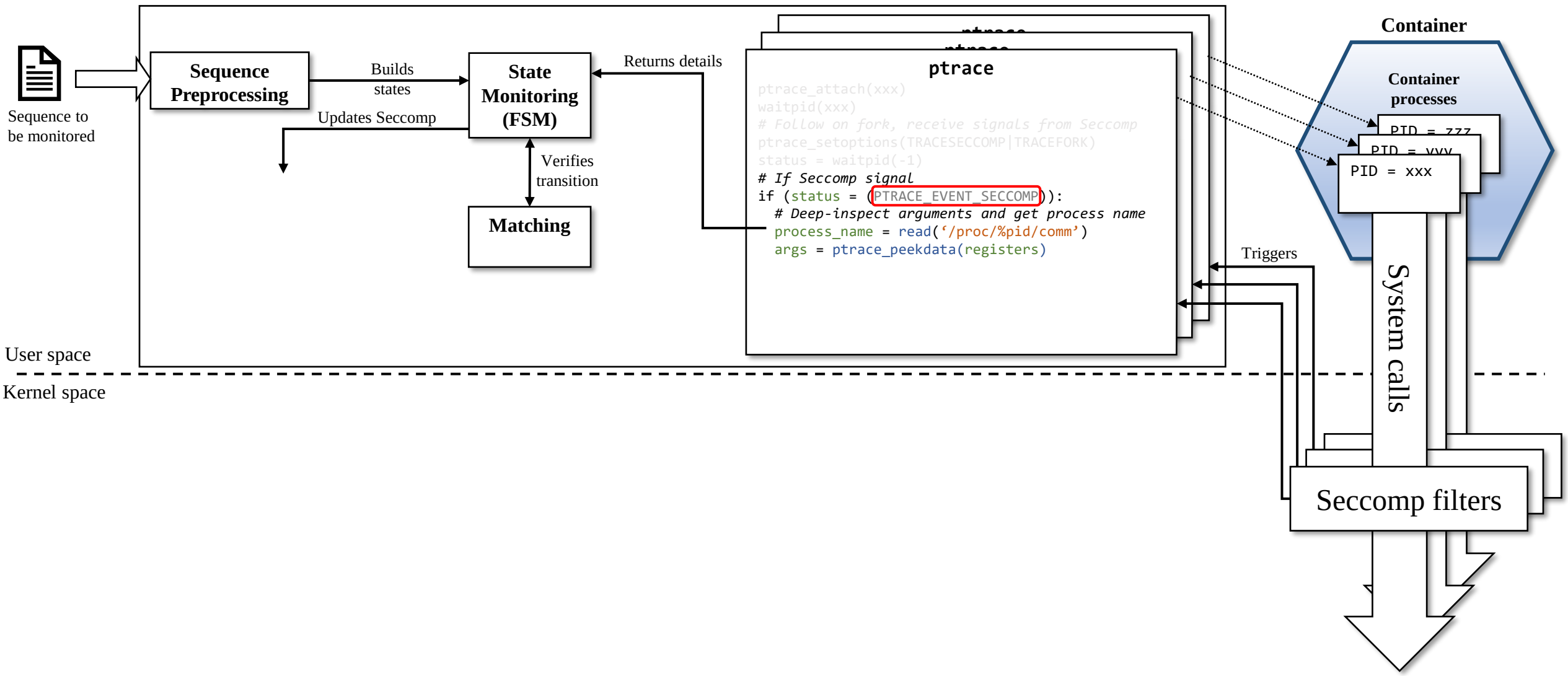
Implementation of Phoenix's Dynamic Runtime Protection



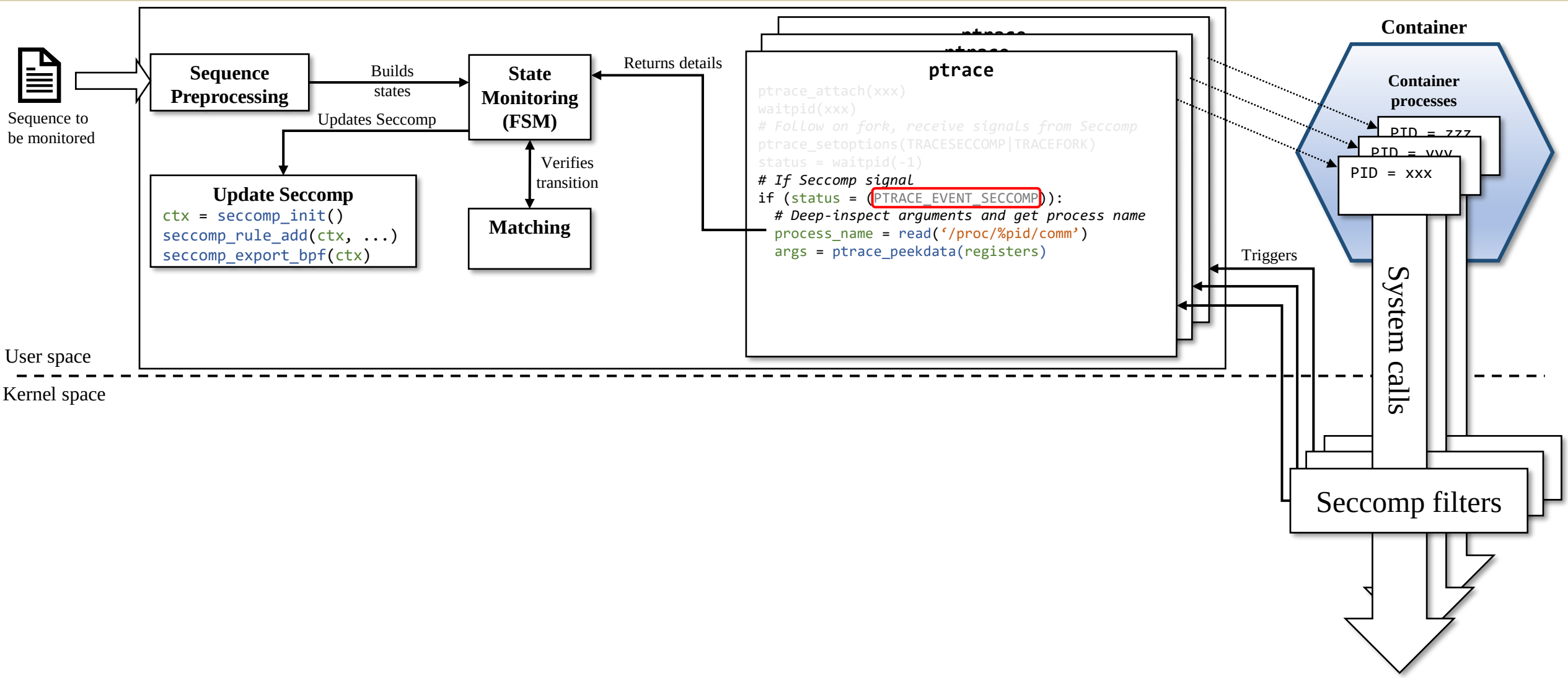
Implementation of Phoenix's Dynamic Runtime Protection



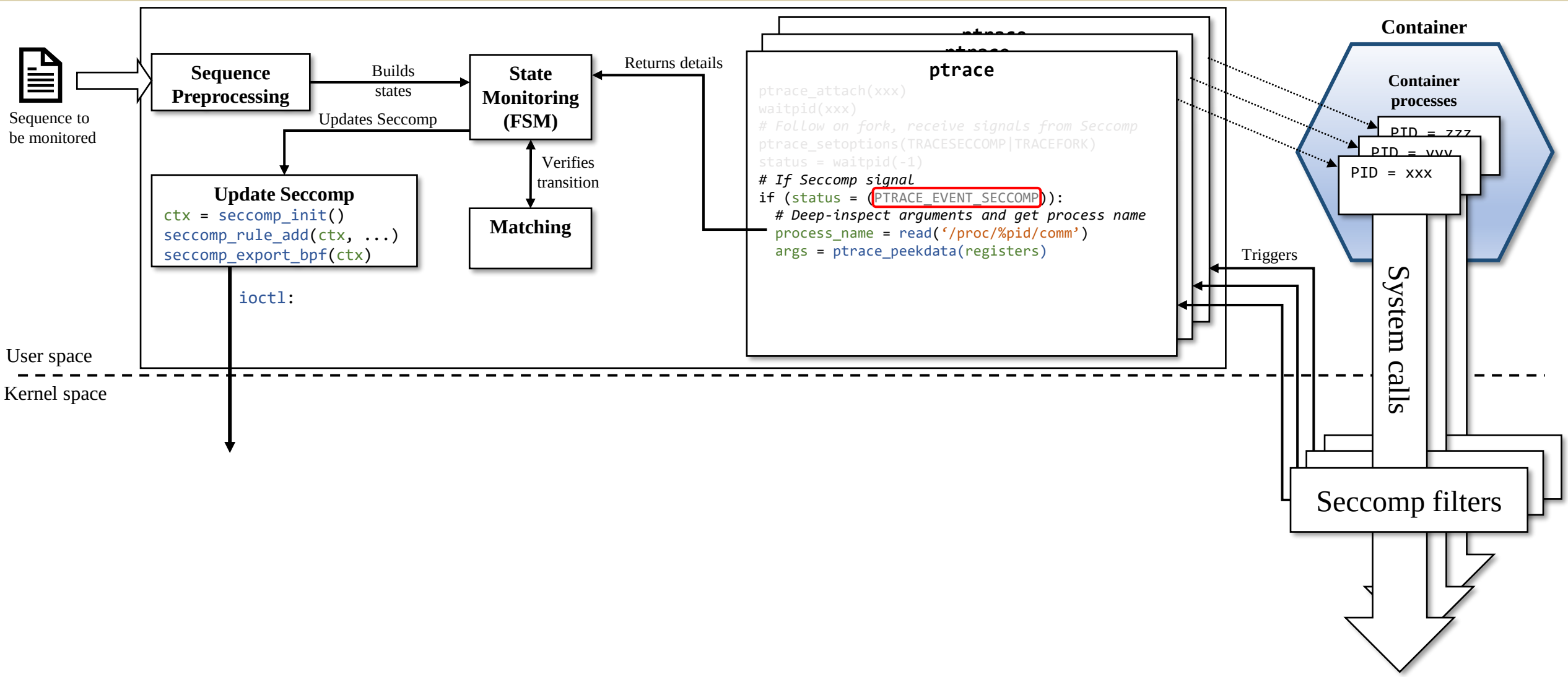
Implementation of Phoenix's Dynamic Runtime Protection



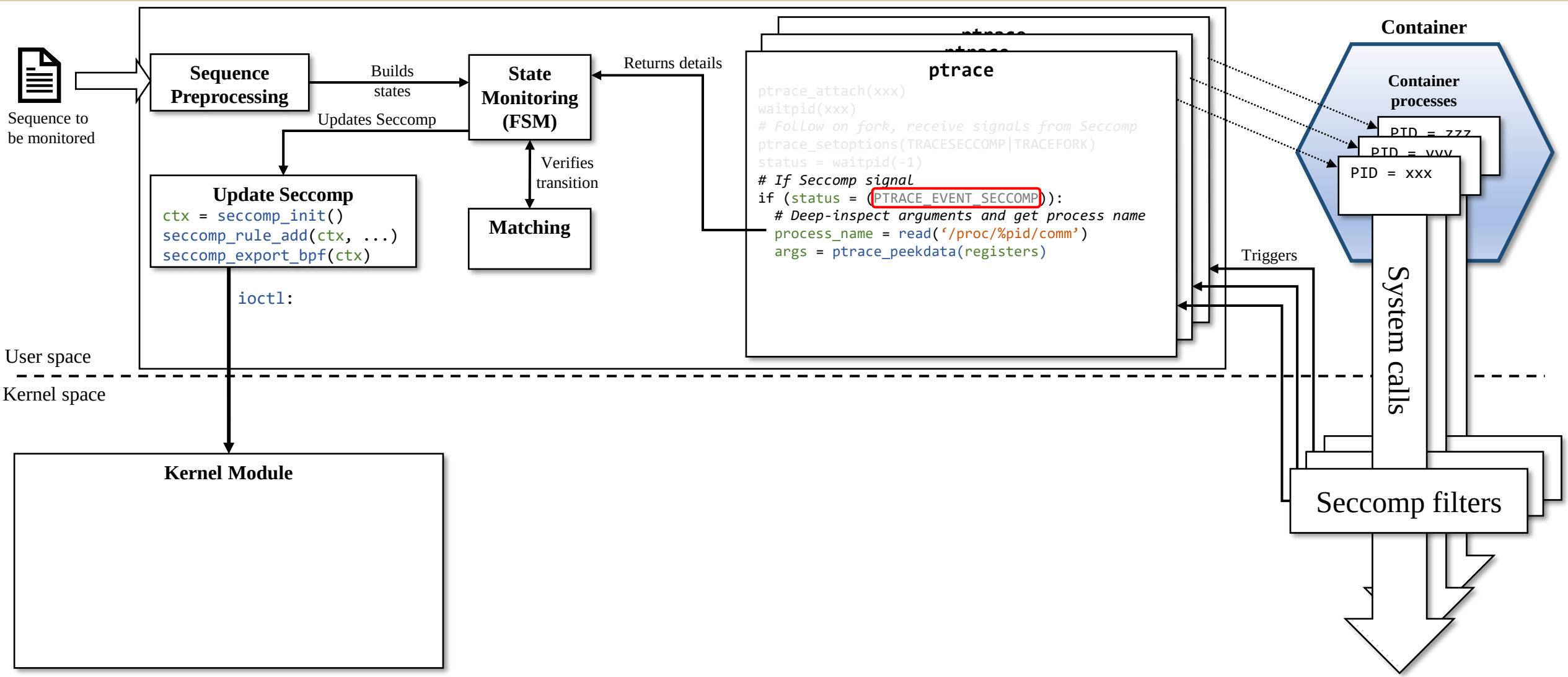
Implementation of Phoenix's Dynamic Runtime Protection



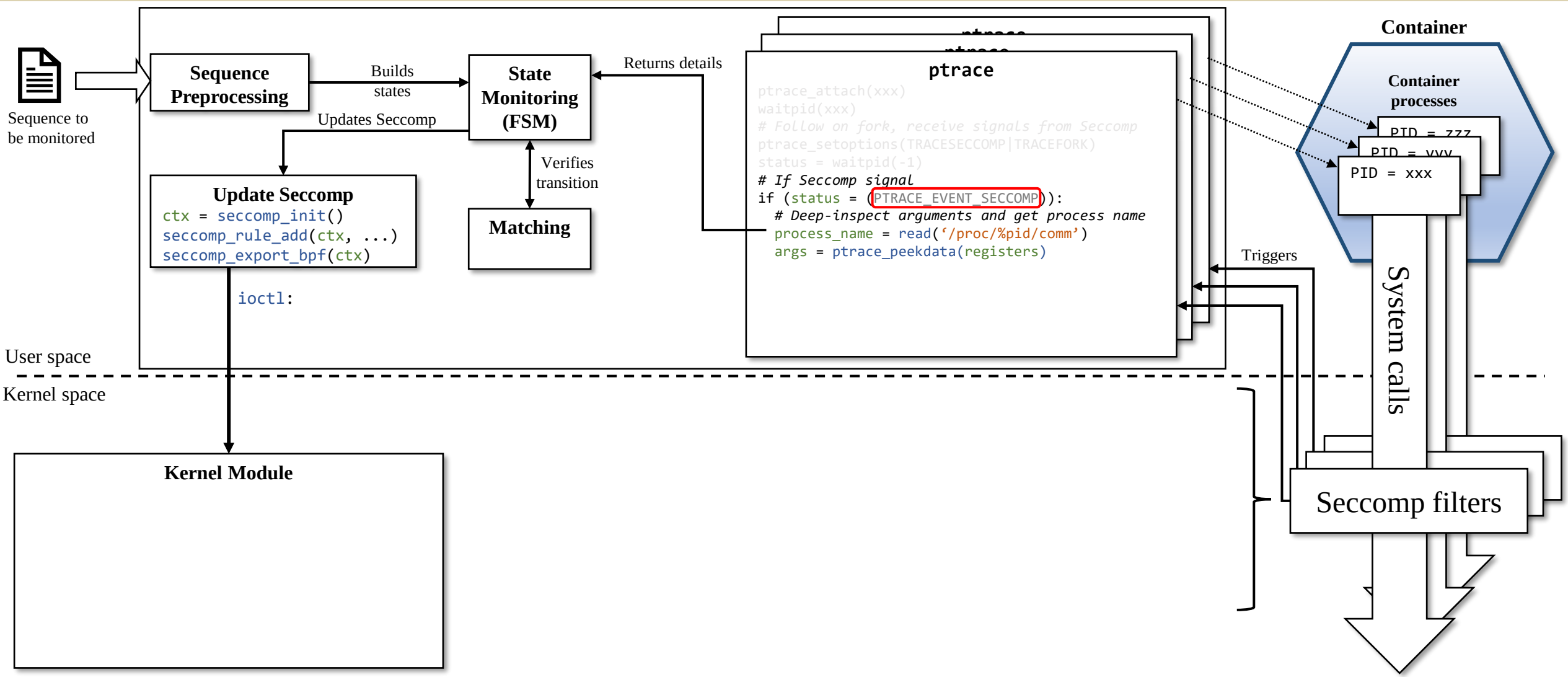
Implementation of Phoenix's Dynamic Runtime Protection



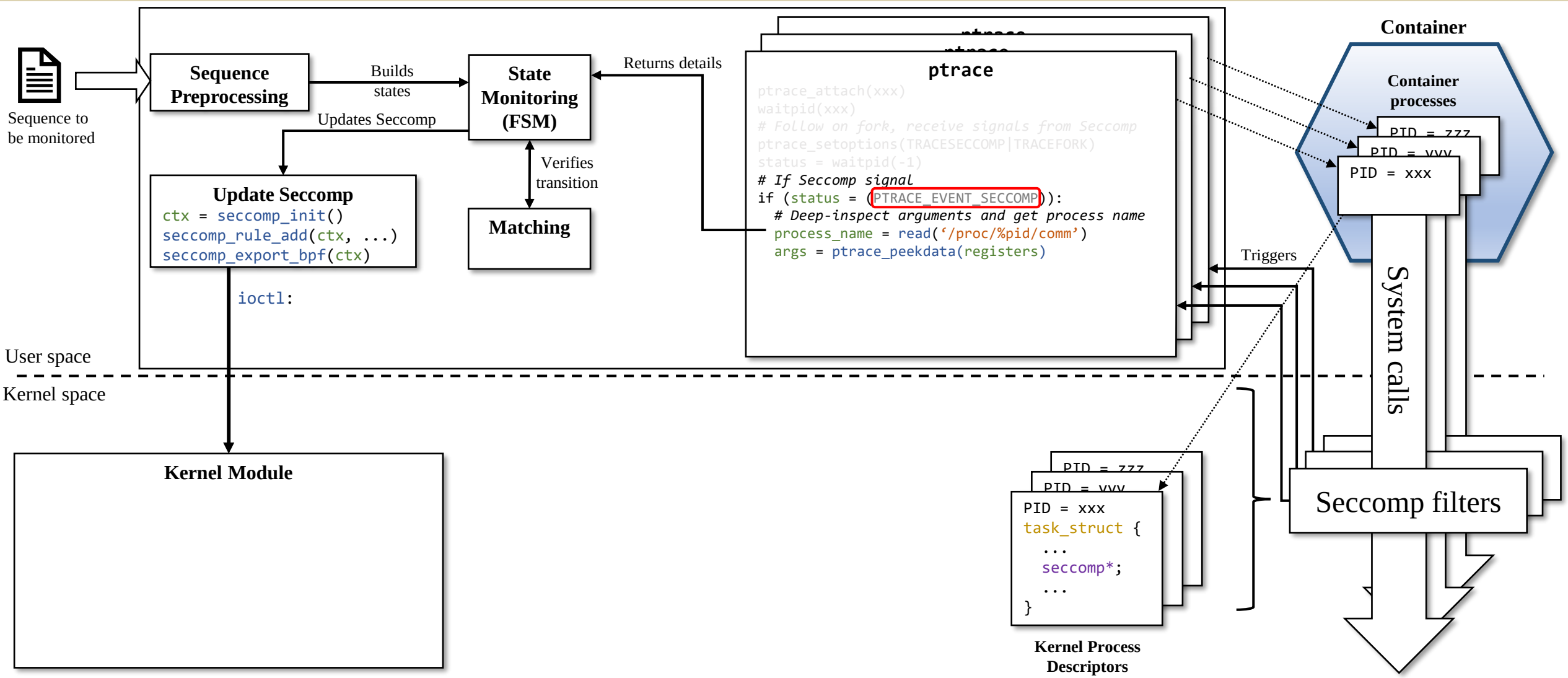
Implementation of Phoenix's Dynamic Runtime Protection



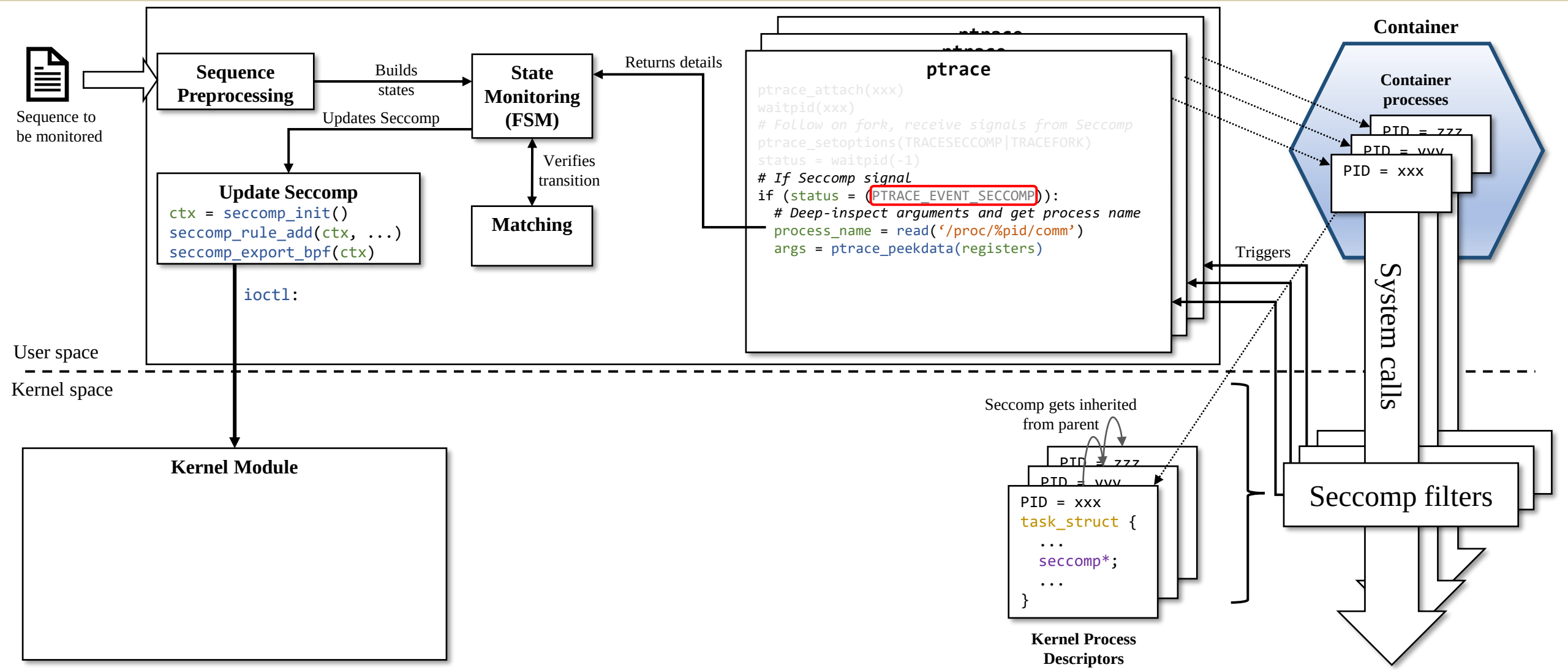
Implementation of Phoenix's Dynamic Runtime Protection



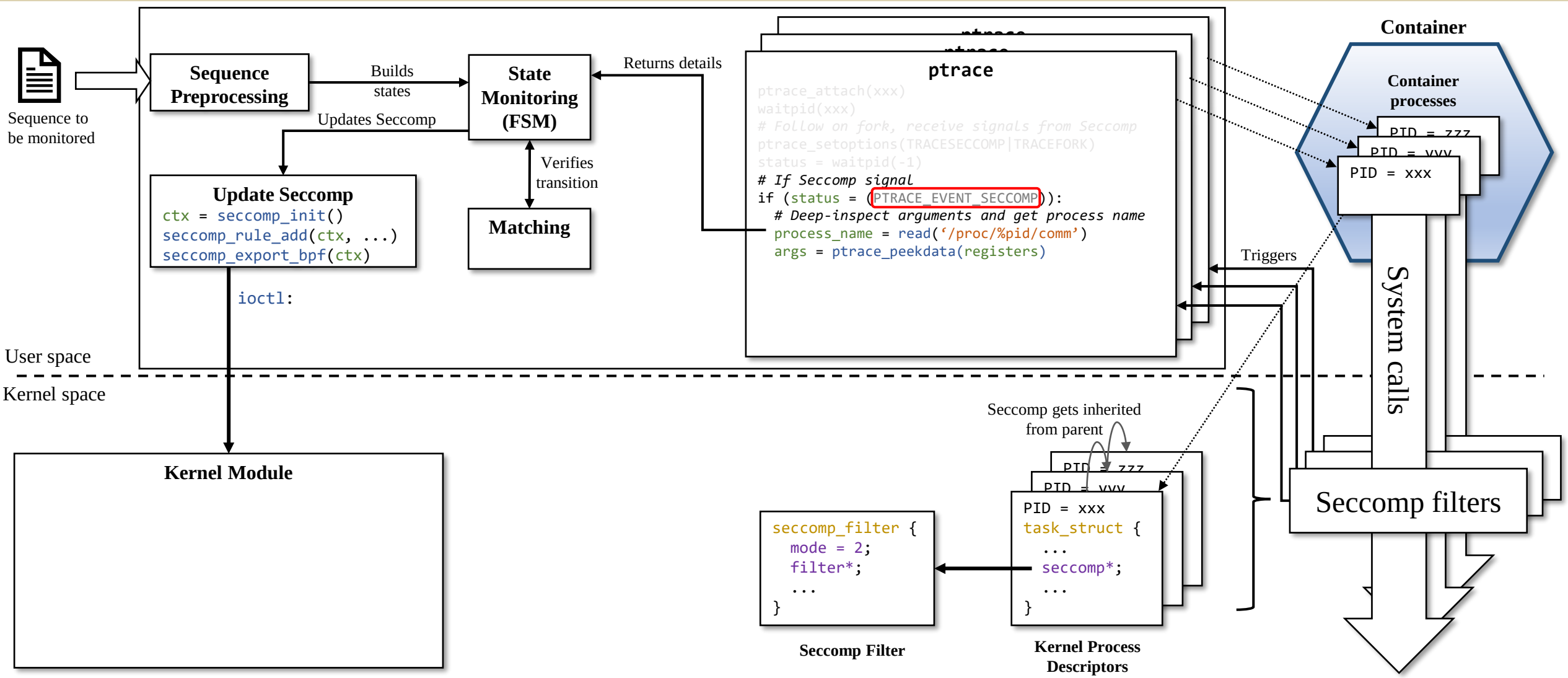
Implementation of Phoenix's Dynamic Runtime Protection



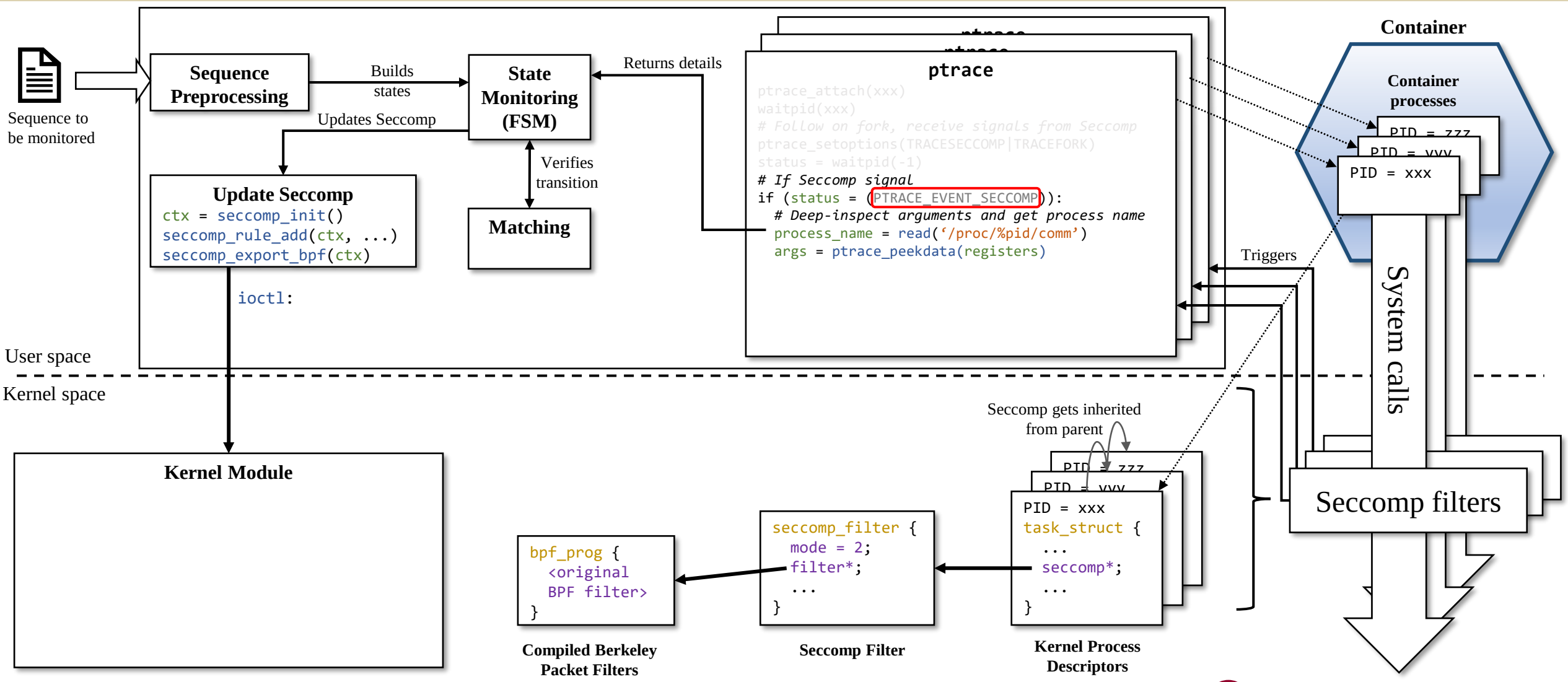
Implementation of Phoenix's Dynamic Runtime Protection



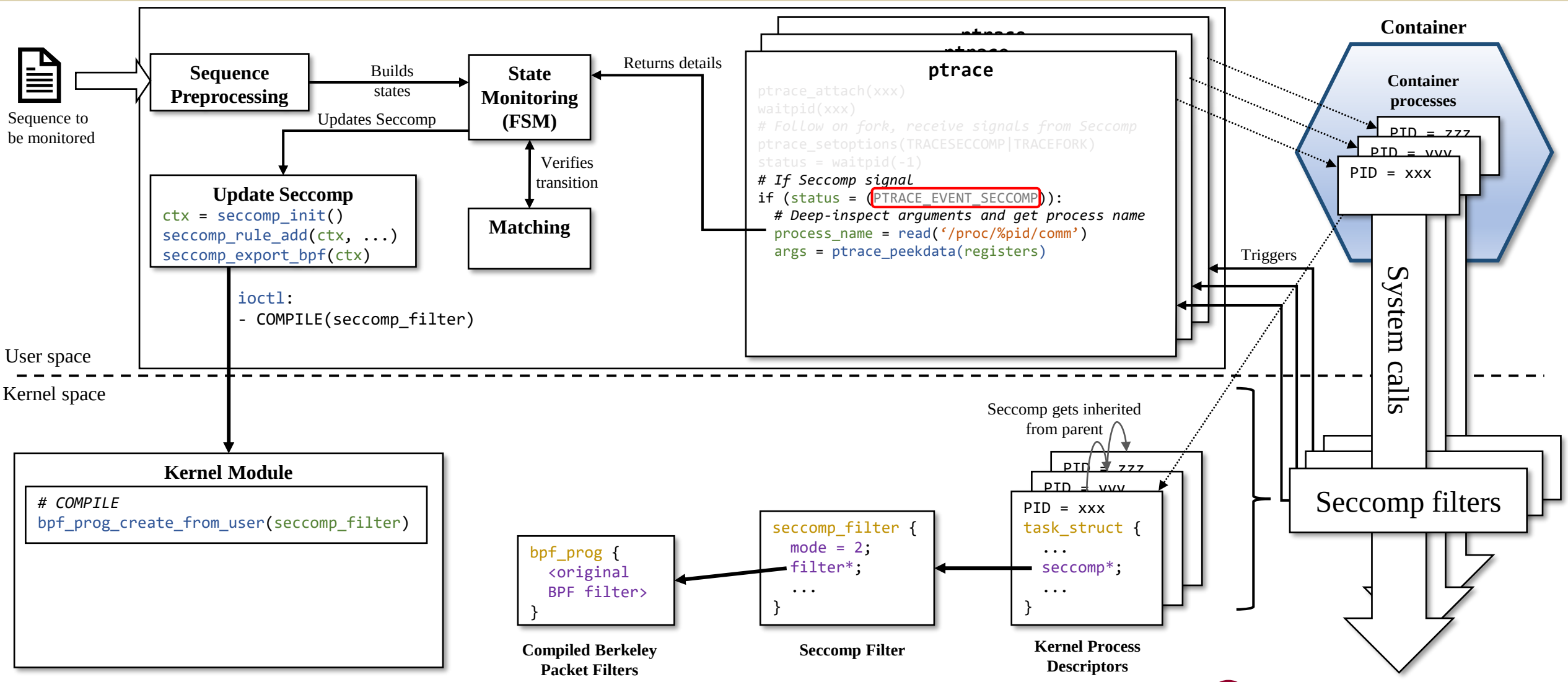
Implementation of Phoenix's Dynamic Runtime Protection



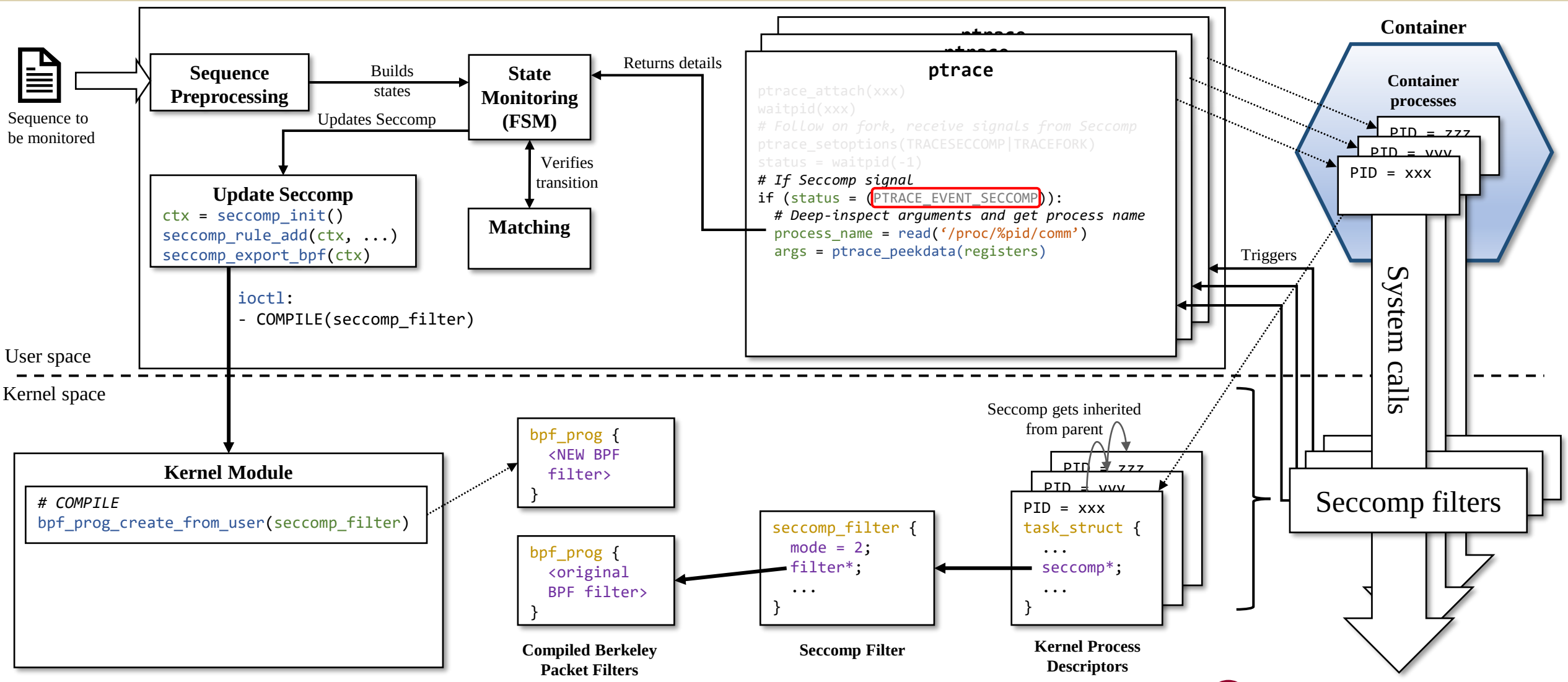
Implementation of Phoenix's Dynamic Runtime Protection



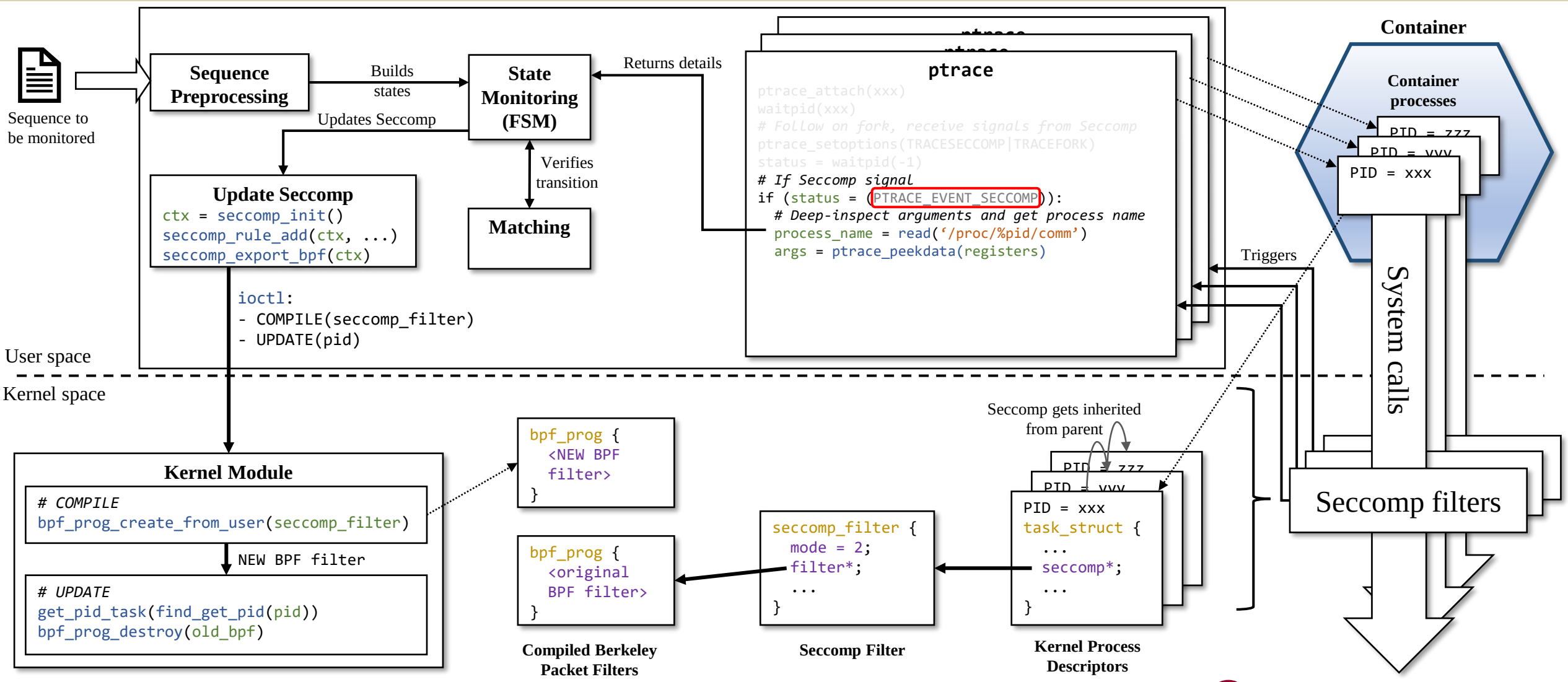
Implementation of Phoenix's Dynamic Runtime Protection



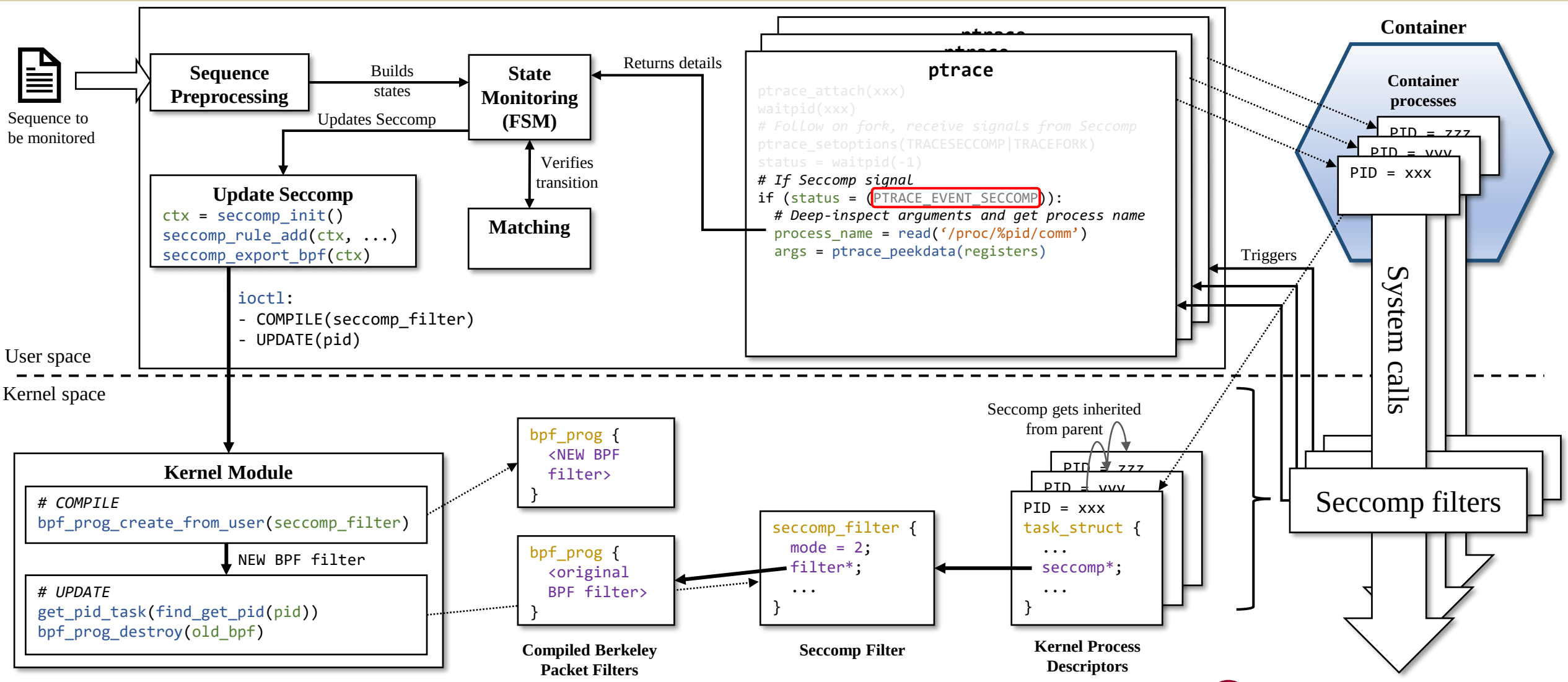
Implementation of Phoenix's Dynamic Runtime Protection



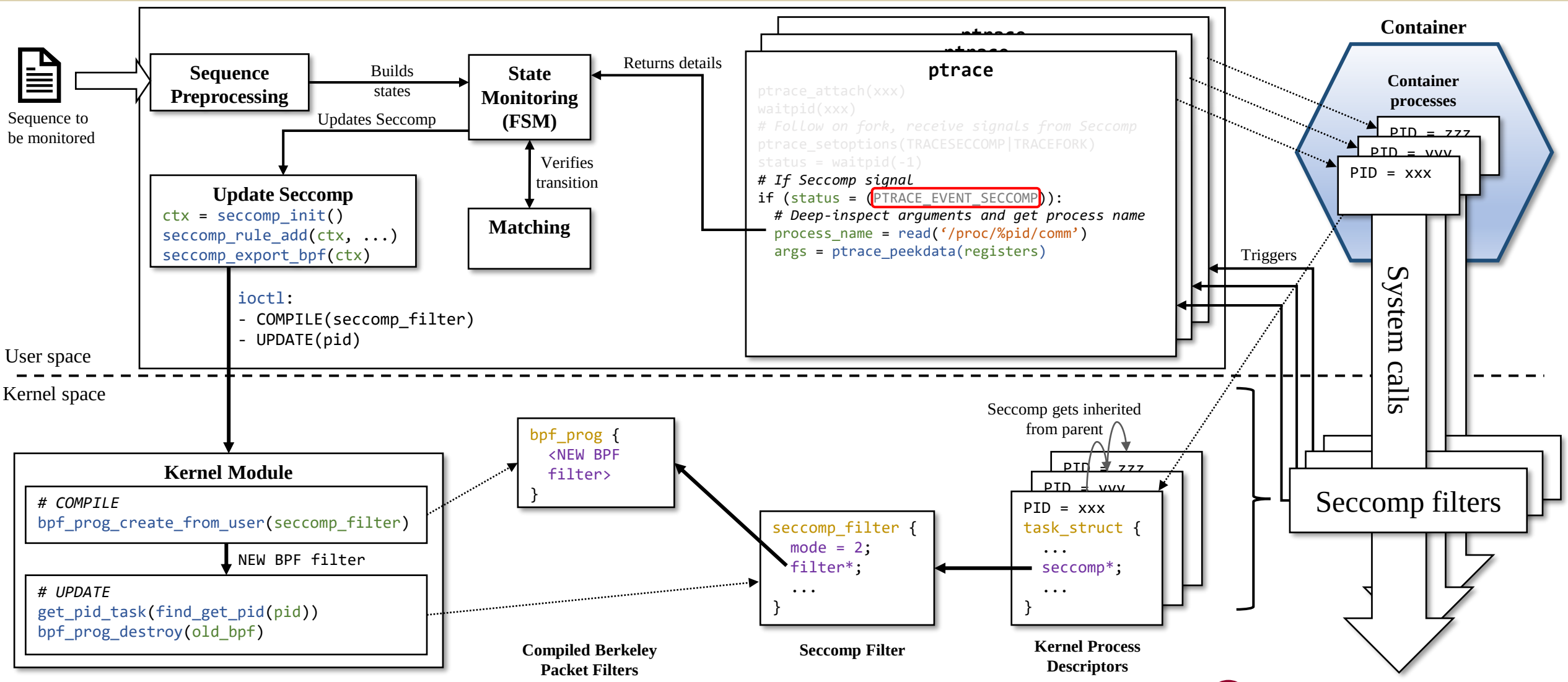
Implementation of Phoenix's Dynamic Runtime Protection



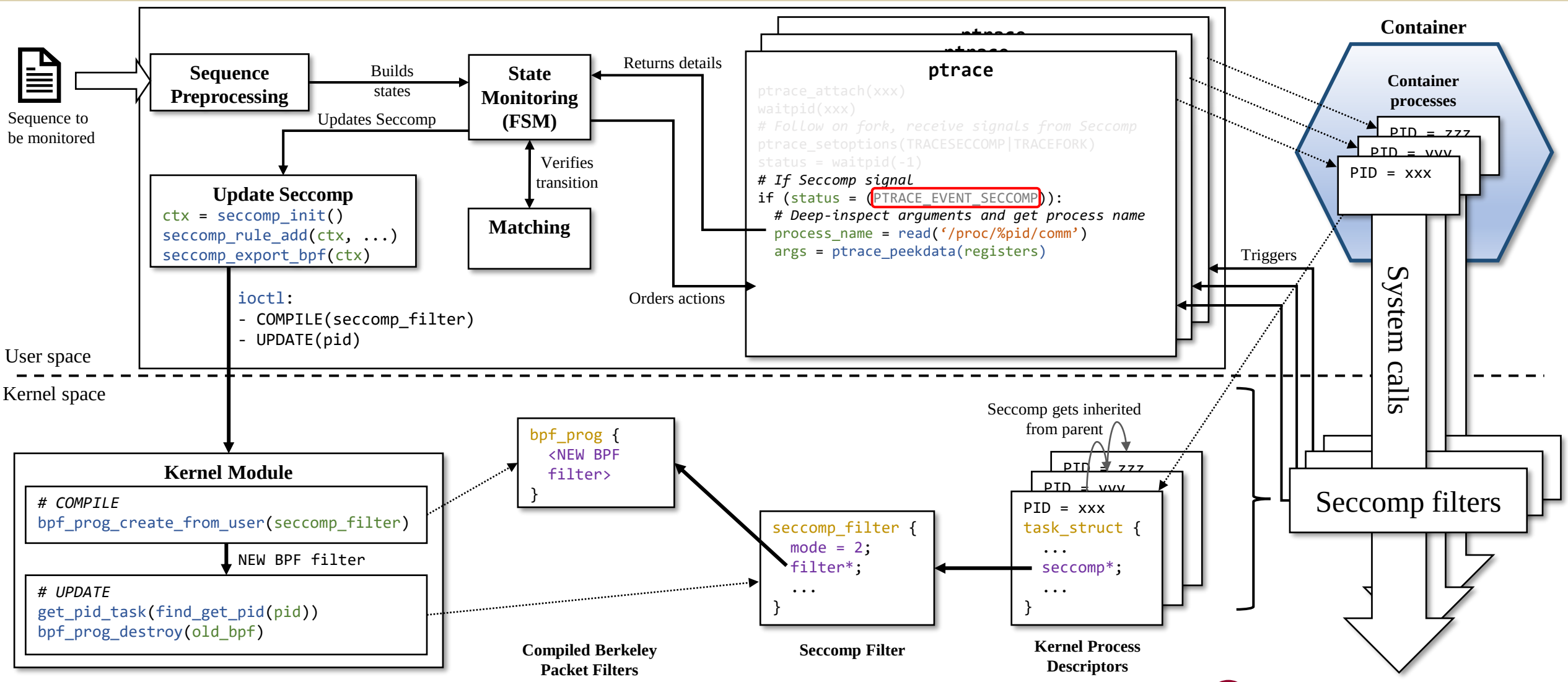
Implementation of Phoenix's Dynamic Runtime Protection



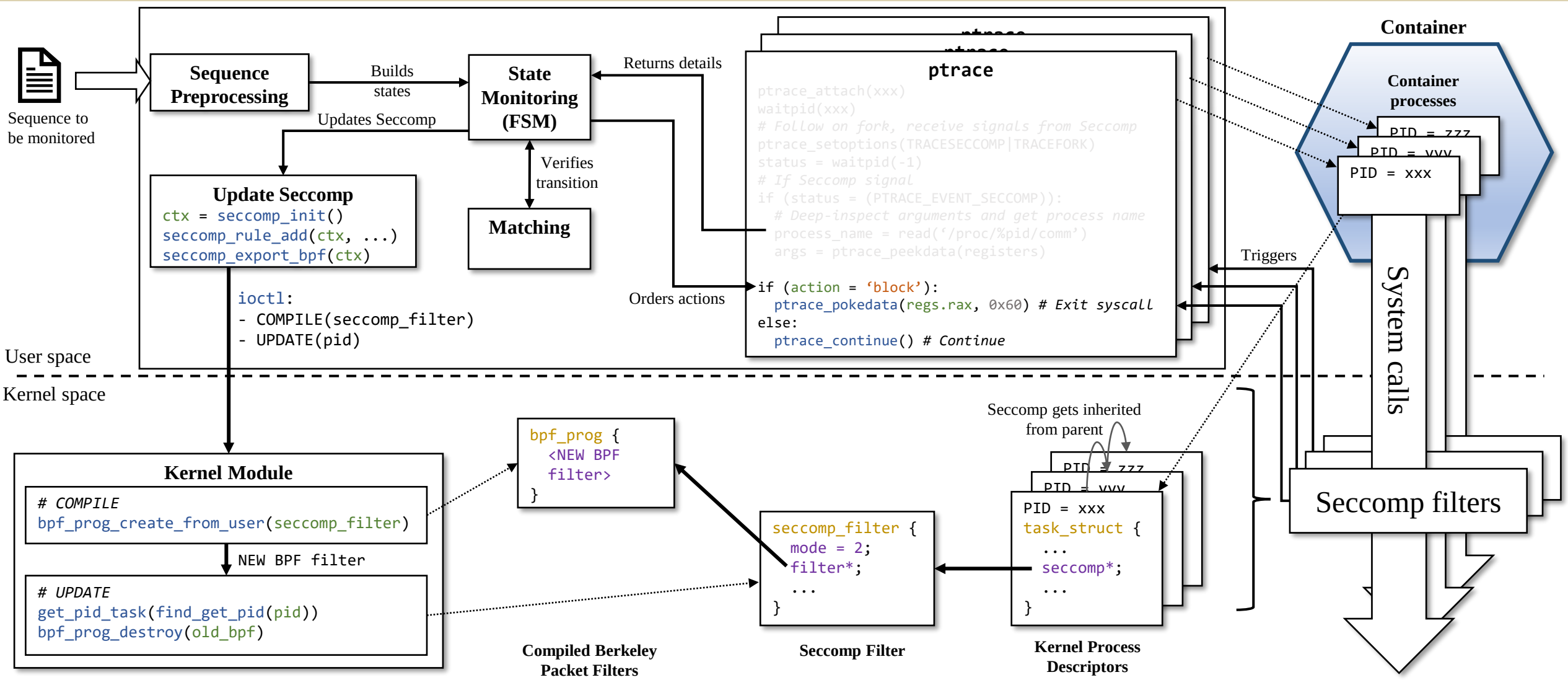
Implementation of Phoenix's Dynamic Runtime Protection



Implementation of Phoenix's Dynamic Runtime Protection



Implementation of Phoenix's Dynamic Runtime Protection



Outline

- Intro
  - Motivation
  - Related Work
- Methodology
  - Key Ideas & Overview
  - Malicious Sequence Identification
  - Dynamic Runtime Protection
- Implementation
- **Experimental Results**
  - **Security**
  - **Performance**
  - **Provenance Analysis**
- Conclusion

Experimental Results - Security

CVE	Severity	Application 1	2	3	4	5	6	7	8	9	10
2017-18344	2.1	--p	csp	csp	csp	csp	csp	csp	csp	-sp	csp
2017-5123	4.6	--p	--p	csp	--p	c-p	--p	--p	c-p	-sp	c-p
2019-5489	5.5	--p	csp	csp	csp	c-p	csp	c-p	-sp	--p	csp
2022-1015	6.6	--p	csp	csp	csp	csp	csp	csp	csp	--p	csp
2017-17053	6.9	-sp	csp	csp	csp	csp	csp	csp	csp	--p	csp
2022-0492*	6.9	-sp	-sp	csp	csp	-sp	-sp	csp	csp	-sp	-sp
2022-2602	7.0	--p	csp	csp	-sp	-sp	csp	csp	csp	-sp	-sp
2017-11176	7.2	--p	csp	csp	csp	csp	csp	csp	csp	-sp	csp
2018-14634	7.2	--p	--p	--p	--p	--p	--p	--p	--p	-sp	--p
2021-3347	7.2	-sp	--p	csp	--p	c-p	--p	csp	-sp	-sp	-sp
2021-4154	7.2	-sp	-sp	c-p	-sp	csp	-sp	-sp	csp	-sp	csp
2022-0847	7.2	--p	c-p	-sp	--p	-sp	--p	-sp	-sp	-sp	-sp
2016-9793	7.8	-sp	csp	csp	csp	--p	csp	csp	csp	-sp	csp
2017-6074	7.8	-sp	csp	csp	-sp	--p	csp	csp	-sp	-sp	-sp
2017-7308	7.8	-sp	-sp	-sp	-sp	-sp	-sp	-sp	--p	-sp	-sp
2022-0995	7.8	-sp	csp	csp	csp	c-p	csp	csp	csp	-sp	csp
2022-2588	7.8	--p	csp	csp	csp	csp	csp	csp	csp	-sp	csp
2022-2639	7.8	--p	csp	csp	csp	csp	csp	csp	csp	-sp	csp
2023-0386	7.8	--p	-sp	-sp	-sp	-sp	-sp	csp	--p	-sp	-sp
2023-32233	7.8	-sp	csp	csp	-sp	csp	csp	csp	csp	-sp	csp

1: CRIU†, 2: Django, 3: Httpd, 4: Nginx, 5: Postgres, 6: Python, 7: Redis, 8: Tomcat, 9: Wine†, 10: Wordpress.
c: blocked by Confine [24], s: blocked by Sysfilter [11], p: blocked by Phoenix, -: not blocked.
*blocked by default Seccomp filter [67], †Confine not tested (not a container).

TABLE I: Comparison of the effectiveness of Confine [24], Sysfilter [11], and Phoenix for blocking 20 CVEs without affecting the normal operation of 10 popular applications.

Solution	Vulnerability learned	Same vulnerability exploit (TP)	Modified vulnerability exploit (TP)	Normal behavior (FP)
VtPath	2023-32233	100%	15%	0%
	2017-6074	100%	17%	0%
	2022-0847	100%	63%	0%
	2021-4154	100%	64%	0%
	2023-0386	100%	94%	0.01%
Mutz et al.	2023-32233	0.57%	0.42%	0%
	2017-6074	1.83%	1.22%	0%
	2022-0847	7.59%	3.80%	0%
	2021-4154	7.89%	7.16%	0%
	2023-0386	1.34%	1.29%	0%
PoLPer	2023-32233	100% (2/2)	0% (0/2)	0% (0/2)
	2017-6074	0*%	0*%	0*%
	2022-0847	0*%	0*%	0*%
	2021-4154	0*%	0*%	0*%
	2023-0386	100% (3/3)	33% (1/3)	0% (0/3)
Phoenix	2023-32233	100%	100%	0%
	2017-6074	100%	100%	0%
	2022-0847	100%	100%	0%
	2021-4154	100%	100%	0%
	2023-0386	100%	100%	0%

*: exploit does not invoke *setuid* calls

TABLE III: Comparison of Phoenix with existing stateful solutions for blocking vulnerabilities (blacklisting)

Experimental Results - Security

CVE	Severity	Application 1	2	3	4	5	6	7	8	9	10
2017-18344	2.1	--p	csp	csp	csp	csp	csp	csp	csp	-sp	csp
2017-5123	4.6	--p	--p	csp	--p	c-p	--p	--p	c-p	-sp	c-p
2019-5489	5.5	--p	csp	csp	csp	c-p	csp	c-p	-sp	--p	csp
2022-1015	6.6	--p	csp	csp	csp	csp	csp	csp	csp	--p	csp
2017-17053	6.9	-sp	csp	csp	csp	csp	csp	csp	csp	--p	csp
2022-0492*	6.9	-sp	-sp	csp	csp	-sp	-sp	csp	csp	-sp	-sp
2022-2602	7.0	--p	csp	csp	-sp	-sp	csp	csp	csp	-sp	-sp
2022-2639	7.8	--p	csp	csp	csp	csp	csp	csp	csp	-sp	csp
2023-0386	7.8	--p	-sp	-sp	-sp	-sp	-sp	csp	--p	-sp	-sp
2023-32233	7.8	-sp	csp	csp	-sp	csp	csp	csp	csp	-sp	csp

1: CRIU†, 2: Django, 3: Httpd, 4: Nginx, 5: Postgres, 6: Python, 7: Redis, 8: Tomcat, 9: Wine†, 10: Wordpress.
c: blocked by Confine [24], s: blocked by Sysfilter [11], p: blocked by Phoenix, -: not blocked.
*blocked by default Seccomp filter [67], †Confine not tested (not a container).

TABLE I: Comparison of the effectiveness of Confine [24], Sysfilter [11], and Phoenix for blocking 20 CVEs without affecting the normal operation of 10 popular applications.

Phoenix shows superior effectiveness for blocking vulnerabilities thanks to its capability of considering sequences

Solution	Vulnerability learned	Same vulnerability exploit (TP)	Modified vulnerability exploit (TP)	Normal behavior (FP)
VtPath	2023-32233	100%	15%	0%
	2017-6074	100%	17%	0%
	2022-0847	100%	63%	0%
	2021-4154	100%	64%	0%
	2023-0386	100%	94%	0.01%
Mutz et al.	2023-32233	0.57%	0.42%	0%
	2017-6074	1.83%	1.22%	0%
	2022-0847	7.59%	3.80%	0%
	2021-4154	7.89%	7.16%	0%
	2023-0386	1.34%	1.29%	0%
PoLPer	2023-32233	100% (2/2)	0% (0/2)	0% (0/2)
	2017-6074	0*%	0*%	0*%
	2022-0847	0*%	0*%	0*%
	2021-4154	0*%	0*%	0*%
	2023-0386	100% (3/3)	33% (1/3)	0% (0/3)
Phoenix	2023-32233	100%	100%	0%
	2017-6074	100%	100%	0%
	2022-0847	100%	100%	0%
	2021-4154	100%	100%	0%
	2023-0386	100%	100%	0%

*: exploit does not invoke *setuid* calls

TABLE III: Comparison of Phoenix with existing stateful solutions for blocking vulnerabilities (blacklisting)

Experimental Results - Security

CVE	Severity	Application 1	2	3	4	5	6	7	8	9	10
2017-18344	2.1	--p	csp	csp	csp	csp	csp	csp	csp	-sp	csp
2017-5123	4.6	--p	--p	csp	--p	c-p	--p	--p	c-p	-sp	c-p
2019-5489	5.5	--p	csp	csp	csp	c-p	csp	c-p	-sp	--p	csp
2022-1015	6.6	--p	csp	csp	csp	csp	csp	csp	csp	--p	csp
2017-17053	6.9	-sp	csp	csp	csp	csp	csp	csp	csp	--p	csp
2022-0492*	6.9	-sp	-sp	csp	csp	-sp	-sp	csp	csp	-sp	-sp
2022-2602	7.0	--p	csp	csp	-sp	-sp	csp	csp	csp	-sp	-sp
2022-2639	7.8	--p	csp	csp	csp	csp	csp	csp	csp	-sp	csp
2023-0386	7.8	--p	-sp	-sp	-sp	-sp	-sp	csp	--p	-sp	-sp
2023-32233	7.8	-sp	csp	csp	-sp	csp	csp	csp	csp	-sp	csp

1: CRIU†, 2: Django, 3: Httpd, 4: Nginx, 5: Postgres, 6: Python, 7: Redis, 8: Tomcat, 9: Wine†, 10: Wordpress.
c: blocked by Confine [24], s: blocked by Sysfilter [11], p: blocked by Phoenix, -: not blocked.
*blocked by default Seccomp filter [67], †Confine not tested (not a container).

TABLE I: Comparison of the effectiveness of Confine [24], Sysfilter [11], and Phoenix for blocking 20 CVEs without affecting the normal operation of 10 popular applications.

Phoenix shows superior effectiveness for blocking vulnerabilities thanks to its capability of considering sequences

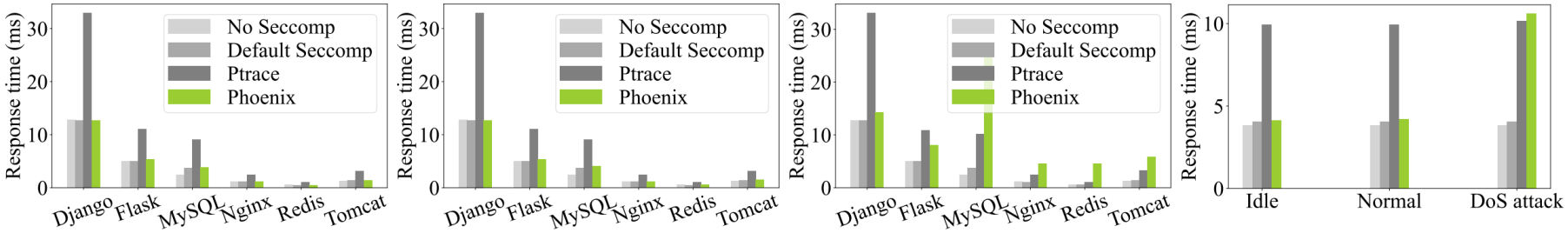
	Vulnerability learned	Same vulnerability exploit (TP)	Modified vulnerability exploit (TP)	Normal behavior (FP)
VtPath	2023-32233	100%	15%	0%
	2017-6074	100%	17%	0%
	2022-0847	100%	63%	0%
	2021-4154	100%	64%	0%
	2023-0386	100%	94%	0.01%
Phoenix	2023-32233	100%	100%	0%
	2017-6074	100%	100%	0%
	2022-0847	100%	100%	0%
	2021-4154	100%	100%	0%
	2023-0386	100%	100%	0%

*: exploit does not invoke *setuid* calls

TABLE III: Comparison of Phoenix with existing stateful solutions for blocking vulnerabilities (blacklisting)

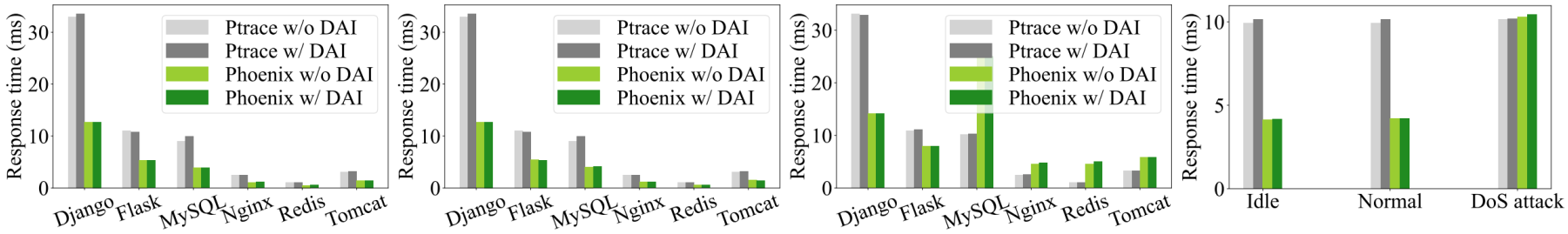
Existing solutions cannot detect slight variation of an attack, while Phoenix can accurately block vulnerabilities without false positives

Experimental Results - Performance



(a) Idle (b) Normal (c) DoS attack (d) Average on all applications

Fig. 12: Overhead of different solutions in terms of response time on various container applications



(a) Idle (b) Normal (c) DoS attack (d) Average on all applications

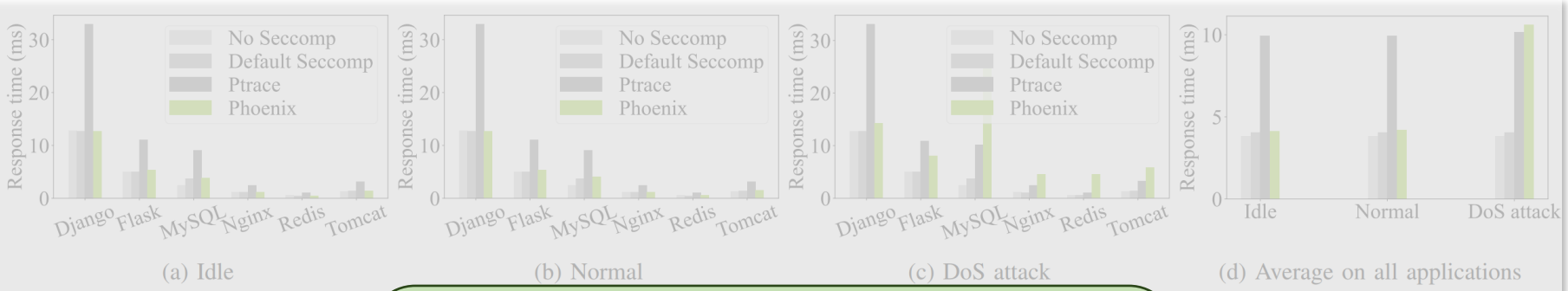
Fig. 14: Overhead of DAI on various container applications

		App. CPU (%)	App. Mem. (MB)	Solution CPU (%)	Solution Mem. (MB)
No Seccomp		5.01	58.01	N/A	N/A
Default Seccomp		5.03	58.01	N/A*	N/A*
Ptrace	(idle)	4.98	57.75	5.18	0.58
	(normal)	5.02	57.90	5.18	0.57
	(DoS)	5.01	57.82	5.24	0.59
Phoenix	(idle)	4.97	57.73	0.03	0.63
	(normal)	5.02	57.82	0.1	0.64
	(DoS)	5.01	57.71	6.82	0.66

*: not collected as Seccomp does not execute in a separate kernel thread

TABLE VI: Average CPU and memory consumption of the application and the solutions (Seccomp, Ptrace, and Phoenix)

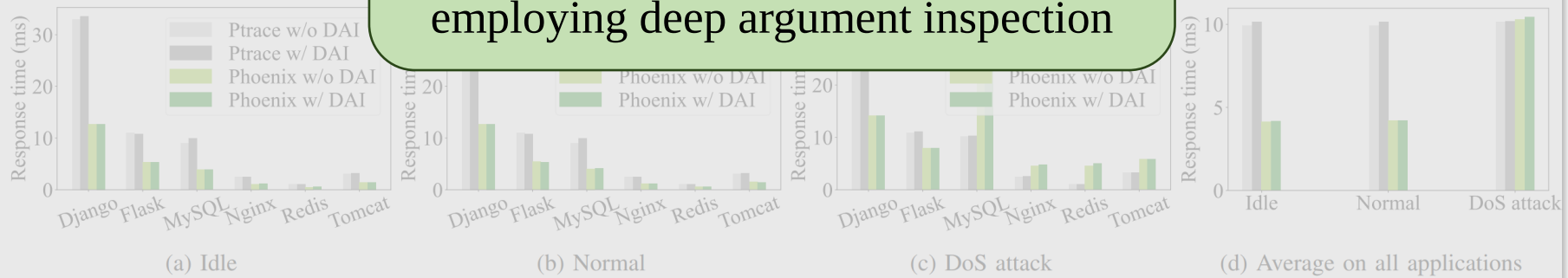
Experimental Results - Performance



(a) Idle (b) Normal (c) DoS attack (d) Average on all applications

Fig. 12: Overhea tainer applications

Phoenix incurs almost no overhead compared to the default security mechanism for containers, even when employing deep argument inspection



(a) Idle (b) Normal (c) DoS attack (d) Average on all applications

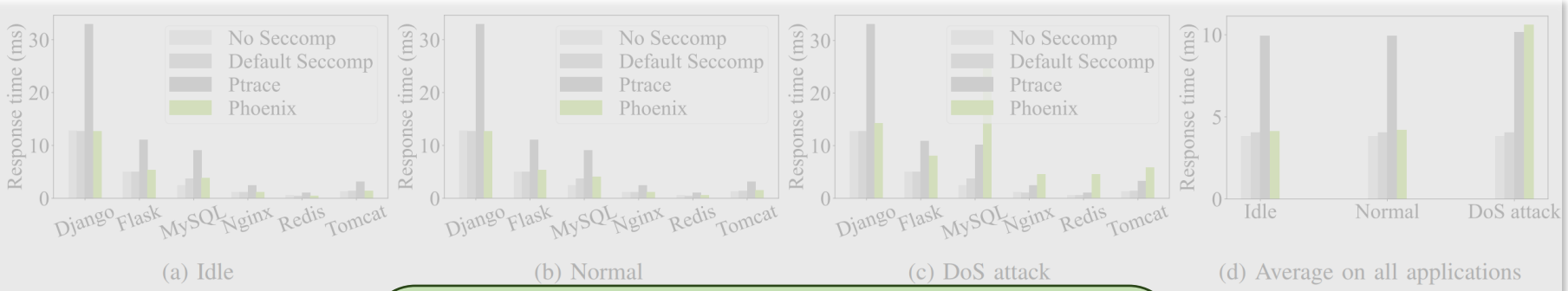
Fig. 14: Overhead of DAI on various container applications

		App. CPU (%)	App. Mem. (MB)	Solution CPU (%)	Solution Mem. (MB)
No Seccomp		5.01	58.01	N/A	N/A
Default Seccomp		5.03	58.01	N/A*	N/A*
Ptrace	(idle)	4.98	57.75	5.18	0.58
	(normal)	5.02	57.90	5.18	0.57
	(DoS)	5.01	57.82	5.24	0.59
Phoenix	(idle)	4.97	57.73	0.03	0.63
	(normal)	5.02	57.82	0.1	0.64
	(DoS)	5.01	57.71	6.82	0.66

*: not collected as Seccomp does not execute in a separate kernel thread

TABLE VI: Average CPU and memory consumption of the application and the solutions (Seccomp, Ptrace, and Phoenix)

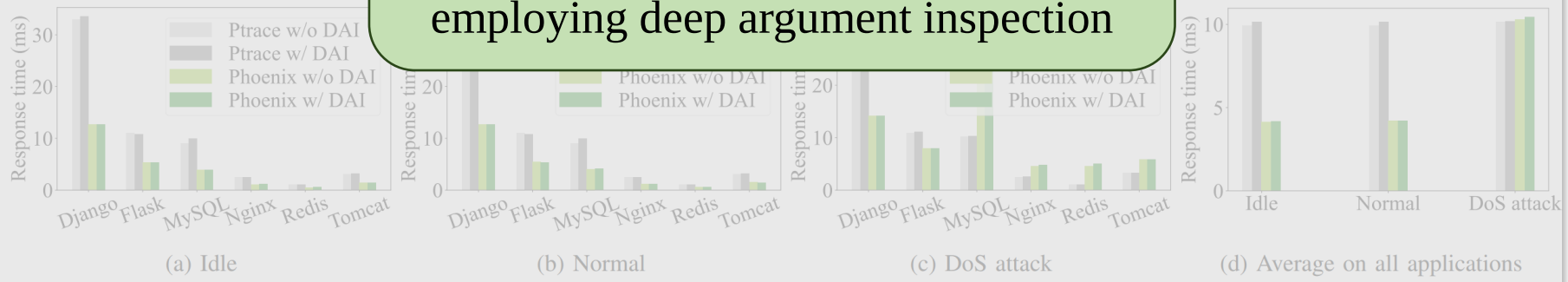
Experimental Results - Performance



(a) Idle (b) Normal (c) DoS attack (d) Average on all applications

Fig. 12: Overhea tainer applications

Phoenix incurs almost no overhead compared to the default security mechanism for containers, even when employing deep argument inspection



(a) Idle (b) Normal (c) DoS attack (d) Average on all applications

Fig. 14: Overhead of DAI on various container applications

	App. CPU (%)	App. Mem. (MB)	Solution CPU (%)	Solution Mem. (MB)
No Seccomp	5.01	58.01	N/A	N/A
Default Sec	5.03	58.01	N/A*	N/A*
Pt				
Ph				

*: in a

TA mem application and the solutions (Seccomp, Ptrace, and Phoenix)

Less than 0.1% CPU overhead against more than 5% for the same level of protection using Ptrace

Experimental Results - Malicious Sequence Investigation

	2017-6074			2021-4154			2022-0847			2023-0386			2023-32233		
	Size	*FP*	*FN*	*Size*	*FP*	*FN*	*Size*	*FP*	*FN*	*Size*	*FP*	*FN*	*Size*	*FP*	*FN*
S/A/S	329	323	0	457	453	0	80	72	0	>18k	>18k	0	>7k	>7k	0
Nimos	4	4	6	3	2	3	4	1	5	4	4	8	4	3	5
Madani	3	0	0	3	3	4	3	2	7	3	1	5	4	3	5
CLARION	310	304	0	>1k	>1k	0	777	769	0	>29k	>29k	0	>4k	>4k	0
DepImpact	25	16	0	N/A			44	33	0	149	117	0	N/A		
Phoenix	6	0	0	4	0	0	8	0	0	8	0	0	8	0	0

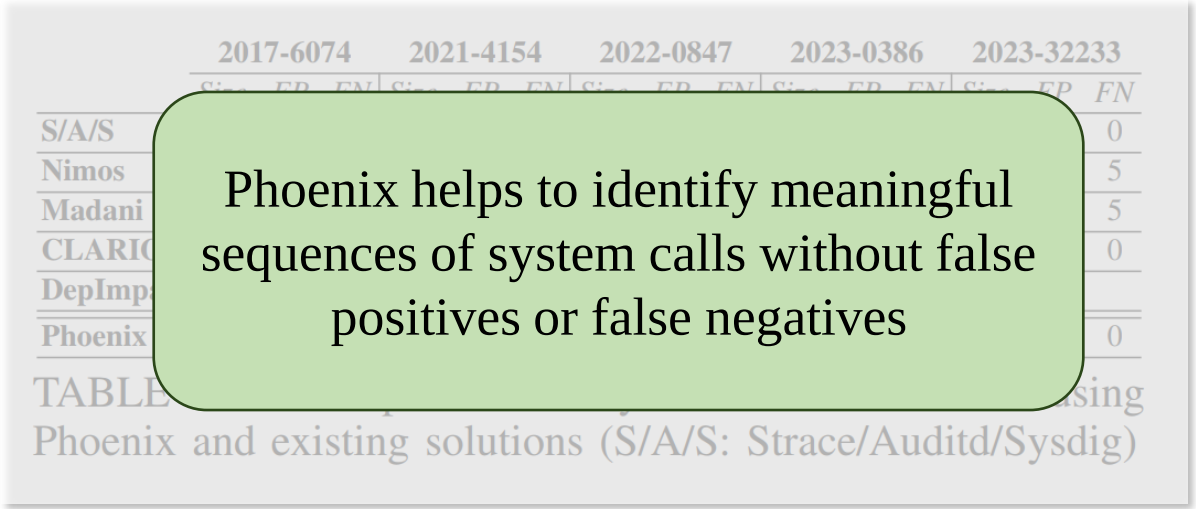
TABLE VII: Comparison of system calls identified using Phoenix and existing solutions (S/A/S: Strace/Auditd/Sysdig)

	2021-4154				2022-0847				2023-0386				2023-32233			
	T1	*Size*	*T2*	*TP*	*T1*	*Size*	*T2*	*TP*	*T1*	*Size*	*T2*	*TP*	*T1*	*Size*	*T2*	*TP*
#1	22′	249	2′	13/13	9′	62	1′	12/12	21′	233	4′	10/12	23′	128	3′	17/17
#2	24′	154	2′	11/13	34′	218	3′	11/12	8′	203	4′	10/12	10′	232	3′	15/17
#3	25′	250	4′	13/13	18′	63	1′	9/12	12′	495	6′	4/12	11′	182	2′	17/17
#4	15′	187	4′	12/13	25′	413	5′	5/12	14′	376	3′	2/12	17′	92	2′	11/17
#5	11′	169	3′	12/13	12′	498	6′	9/12	18′	162	2′	10/12	5′	62	1′	1/17
#6	14′	413	7′	11/13	24′	245	2′	8/12	7′	634	6′	10/12	9′	585	5′	17/17
#7	18′	278	5′	13/13	29′	376	5′	12/12	30′	491	6′	10/12	18′	258	3′	17/17
Avg.	18′	243	4′	12/13	22′	268	3′	9/12	16′	370	4′	8/12	13′	220	3′	14/17

T1: time taken by a user to identify candidate sequences (subgraph)
T2: time taken by an expert to extract sequence from the user's result
Original graph sizes: 1.7k, 227k, 123k, 4k

TABLE VIII: Results of a user study on the usability of Phoenix approach for identifying malicious sequences of system calls

Experimental Results - Malicious Sequence Investigation



	2021-4154				2022-0847				2023-0386				2023-32233			
	T1	*Size*	*T2*	*TP*	*T1*	*Size*	*T2*	*TP*	*T1*	*Size*	*T2*	*TP*	*T1*	*Size*	*T2*	*TP*
#1	22′	249	2′	13/13	9′	62	1′	12/12	21′	233	4′	10/12	23′	128	3′	17/17
#2	24′	154	2′	11/13	34′	218	3′	11/12	8′	203	4′	10/12	10′	232	3′	15/17
#3	25′	250	4′	13/13	18′	63	1′	9/12	12′	495	6′	4/12	11′	182	2′	17/17
#4	15′	187	4′	12/13	25′	413	5′	5/12	14′	376	3′	2/12	17′	92	2′	11/17
#5	11′	169	3′	12/13	12′	498	6′	9/12	18′	162	2′	10/12	5′	62	1′	1/17
#6	14′	413	7′	11/13	24′	245	2′	8/12	7′	634	6′	10/12	9′	585	5′	17/17
#7	18′	278	5′	13/13	29′	376	5′	12/12	30′	491	6′	10/12	18′	258	3′	17/17
Avg.	18′	243	4′	12/13	22′	268	3′	9/12	16′	370	4′	8/12	13′	220	3′	14/17

T1: time taken by a user to identify candidate sequences (subgraph)
T2: time taken by an expert to extract sequence from the user's result
Original graph sizes: 1.7k, 227k, 123k, 4k

TABLE VIII: Results of a user study on the usability of Phoenix approach for identifying malicious sequences of system calls

Experimental Results - Malicious Sequence Investigation

Phoenix helps to identify meaningful sequences of system calls without false positives or false negatives

Using Phoenix, users took less than 30 minutes to identify an attack in large provenance graph (up to 227k nodes)

Outline

- Intro
  - Motivation
  - Related Work
- Methodology
  - Key Ideas & Overview
  - Malicious Sequence Identification
  - Dynamic Runtime Protection
- Implementation
- Experimental Results
  - Security
  - Performance
  - Provenance Analysis
- **Conclusion**

Conclusion

- Phoenix for preventing exploit of unpatched vulnerabilities.
- Accurately and efficiently blocking a syscall sequence by combining **Seccomp** and **Ptrace**.
- Malicious sequence identification using provenance analysis.
- Evaluated on real-world CVEs, negligible delay, high efficiency.

Thank you!

Question Time

Hugo Kermabon-Bobinnec, Concordia University, Montreal

hugo.kermabonbobinnec@concordia.ca