Identifying potential attack scenarios in complex systems by means of provenance graphs

Loïc Robert (LAAS-CNRS / Airbus), Vincent Nicomette (LAAS-CNRS), Eric Lacombe (Airbus), Marie-Jose Huguet (LAAS-CNRS), Emmanuel Hebrard (LAAS-CNRS)

LAAS-CNRS
/ Laboratoire d'analyse et d'architecture des systèmes du CNRS

Laboratoire conventionné avec
Université de Toulouse

Outline

1. **Context & motivation**
   Vulnerability investigation in partially known systems
2. **State-Expanded Provenance Graphs formalism**
   Events, traces, state-versions, graph construction
3. **Mitigating over-linking**
   Inter-process and intra-process refinement
4. **The proposed toolchain**
   Capture software and analysis application
5. **Illustrative example**
   Application of the method with demo video
6. **Limitations & next steps**

Outline

1. **Context & motivation**
   Vulnerability investigation in partially known systems
2. **State-Expanded Provenance Graphs formalism**
   Events, traces, state-versions, graph construction
3. **Mitigating over-linking**
   Inter-process and intra-process refinement
4. **The proposed toolchain**
   Capture software and analysis application
5. **Illustrative example**
   Application of the method with demo video
6. **Limitations & next steps**

Auditing information flows is challenging

The setting

A complex system: many long-lived processes, shared files, IPC, background activity.

The evaluator's question

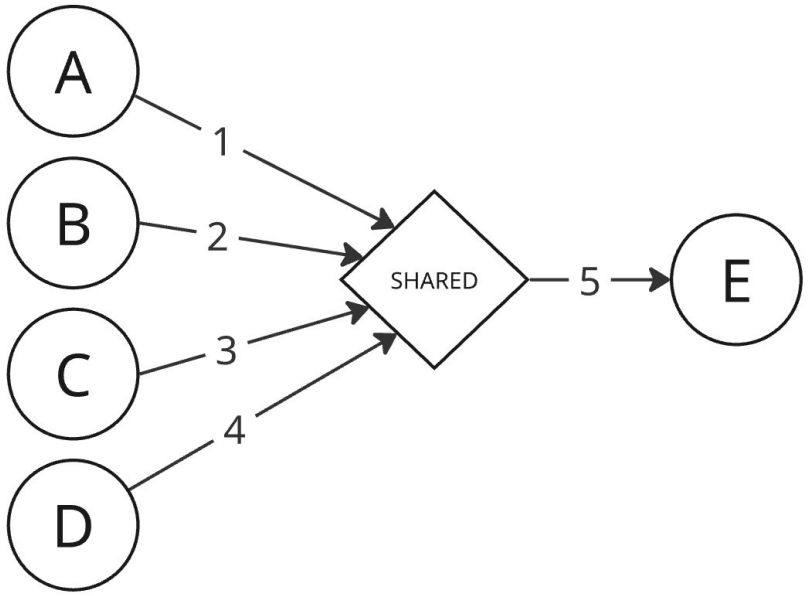
Which actions could trigger a desired effect on a target entity?

What the evaluator has

- Functional executions
- Partial implementation knowledge
- Maybe some programs binaries

but no confirmed attack scenario.

Over-linking, illustrated



Naive provenance: every prior writer becomes a candidate cause of every later read — paths explode, real causal chains hidden.

What our approach is for

IN SCOPE

Vulnerability investigation

- Single host, multiple processes
- Communication via files, FIFOs, pipes
- Monitored functional executions
- Iterative, evaluator-in-the-loop investigation
- Variable granularity (depends on monitoring and modeling effort)
- Discovery of design vulnerabilities

Three ideas, one workflow

01

Lightweight capture

eBPF probes on a small set of file-system centered system-calls.

No app instrumentation, no special hardware, no source-code changes.

Easily replaceable to monitor other systems

02

Stateful provenance

State-Expanded Provenance Graph: every information change creates a new node

No precedence confusion

03

Evaluator-driven pruning

Declarative properties mitigate intra-process over-linking.

Model-to-system fidelity is strenghtened where the evaluator judges it is necessary

Iterative. *Capture → explore → reverse-engineer → encode insight as properties → re-query.*

Outline

1. **Context & motivation**
   Vulnerability investigation in partially known systems
2. **State-Expanded Provenance Graphs formalism**
   Events, traces, state-versions, graph construction
3. **Mitigating over-linking**
   Inter-process and intra-process refinement
4. **The proposed toolchain**
   Capture software and analysis application
5. **Illustrative example**
   Application of the method with demo video
6. **Limitations & next steps**

Formalism · 1/4 — Entities

We model the system as two sets of entities.

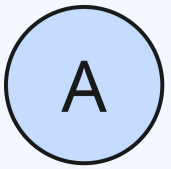
P the set of processes

$R = R_s \sqcup R_c$ the set of resources, partitioned into:

→ R_s storage resources: read operation do not alter its content

→ R_c consumable resources: read mutates the content

This split matters: the two resource classes produce different SEPG structures



Process
e.g., router, client, log collector

Storage resource
Regular file (read does not alter)

Consumable resource
FIFO, pipe (read mutates content)

Formalism · 2/4 — Events

An event is a tuple

$$e = (t_e,\ p_e,\ S_e,\ D_e,\ in_e,\ out_e)$$

t_e	type of event (read, write, open, close, lseek)
p_e	emitting process $p_e \in P$
S_e	set of source entities whose state influences the event
D_e	set of destination entities whose state is changed by the event
in_e	type-specific input values (e.g. fd, count, buf)
out_e	type-specific output values (e.g. fd, size_read, offset, return code)

A trace is $T = \langle e_1,\ e_2,\ \ldots,\ e_n \rangle$; a finite, time-ordered sequence of events.

Formalism · 3/4 — State-versions

Each time an entity's information is changed, a new **state-version** is created.

$$v_k(x) = \left| \{ e_{k'} \mid x \in D_{e_{k'}}, k' \leq k \} \right|$$

Counts how many times x was a destination of information up to event e_k.

Notation: x_0 is the initial version, $x_{v_k(x)}$ is the version after e_k.

State-Expanded Provenance Graph

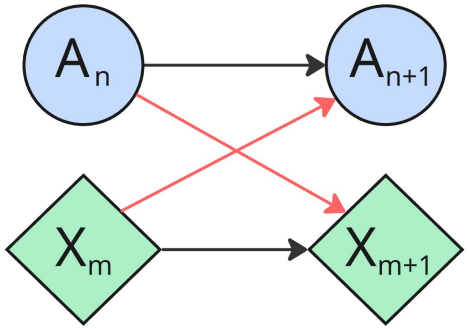
The SEPG of $T = \langle e_1, e_2, \ldots, e_n \rangle$ is

$$V = \{ x_i \mid x \in (P \cup R), \; 0 \leq i \leq v_n(x) \}$$

$$E = \{ (s_{v_k(s)-1}, d_{v_k(d)}) \mid s \in S_{e_k}, d \in D_{e_k}, e_k \in T \}$$

Each event creates a fresh destination state-versions, and connects current source state-versions to it. Time-ordering is built-in.

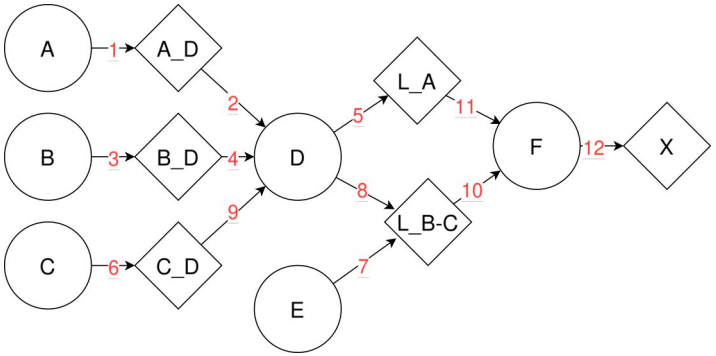
Example : read event structure



e = (read, A, {A, X}, {A, X}, buf="hello", ret=5)

black: state-version progression
red: information transfer

Example of a State-Expanded Provenance Graph



e_1 = (write, A, {A, A_D}, {A, A_D}, ·, ·)

e_2 = (read, D, {D, A_D}, {D, A_D}, ·, ·)

e_3 = (write, B, {B, B_D}, {B, B_D}, ·, ·)

e_4 = (read, D, {D, B_D}, {D, B_D}, ·, ·)

e_5 = (write, D, {D, L_A}, {D, L_A}, ·, ·)

e_6 = (write, C, {C, C_D}, {C, C_D}, ·, ·)

e_7 = (write, E, {E, L_BC}, {E, L_BC}, ·, ·)

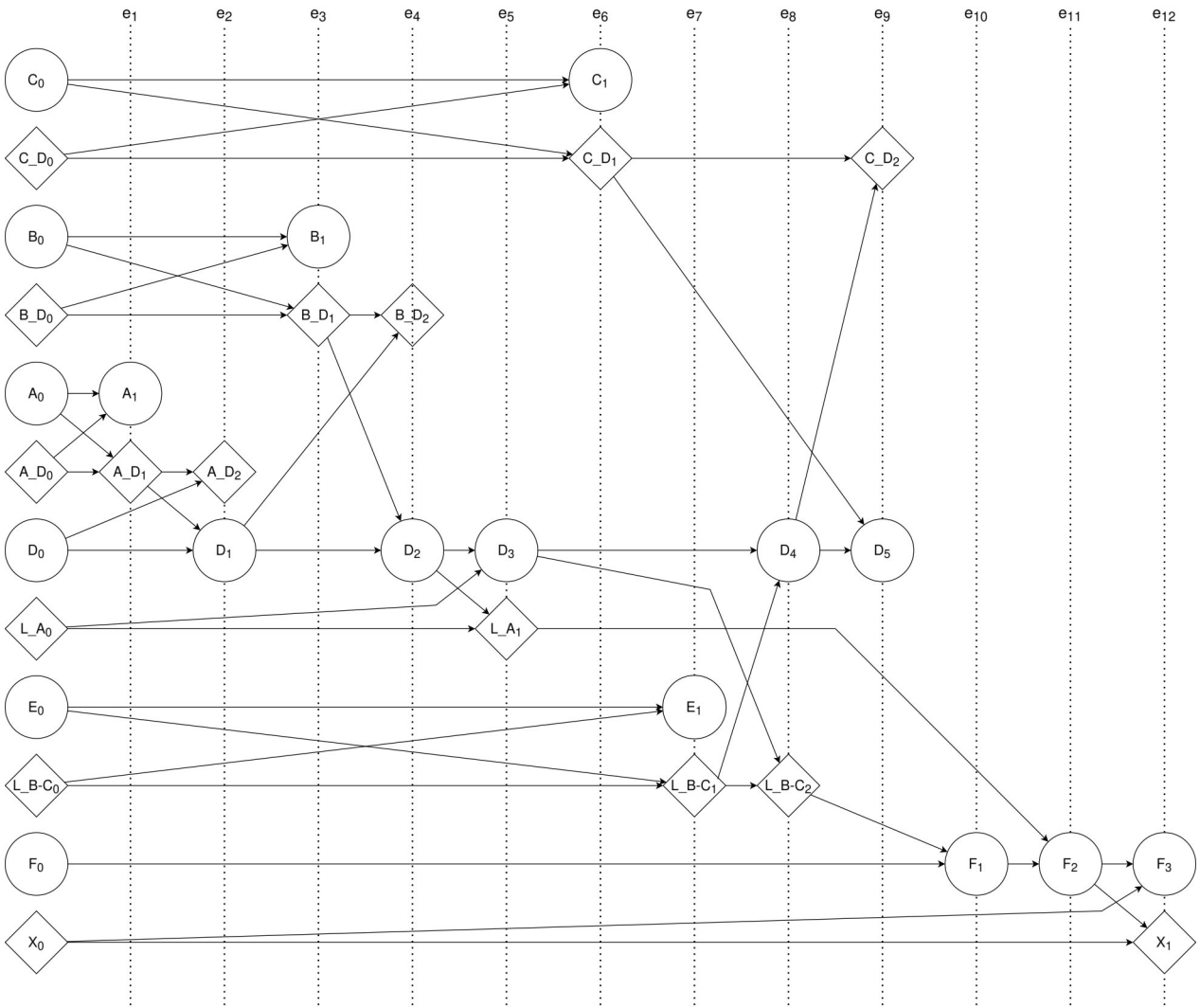
e_8 = (write, D, {D, L_BC}, {D, L_BC}, ·, ·)

e_9 = (read, D, {D, C_D}, {D, C_D}, ·, ·)

e_{10} = (read, F, {F, L_BC}, {F}, ·, ·)

e_{11} = (read, F, {F, L_A}, {F}, ·, ·)

e_{12} = (write, F, {F, X}, {F, X}, ·, ·)



Formalism · 4/4 — Reachability

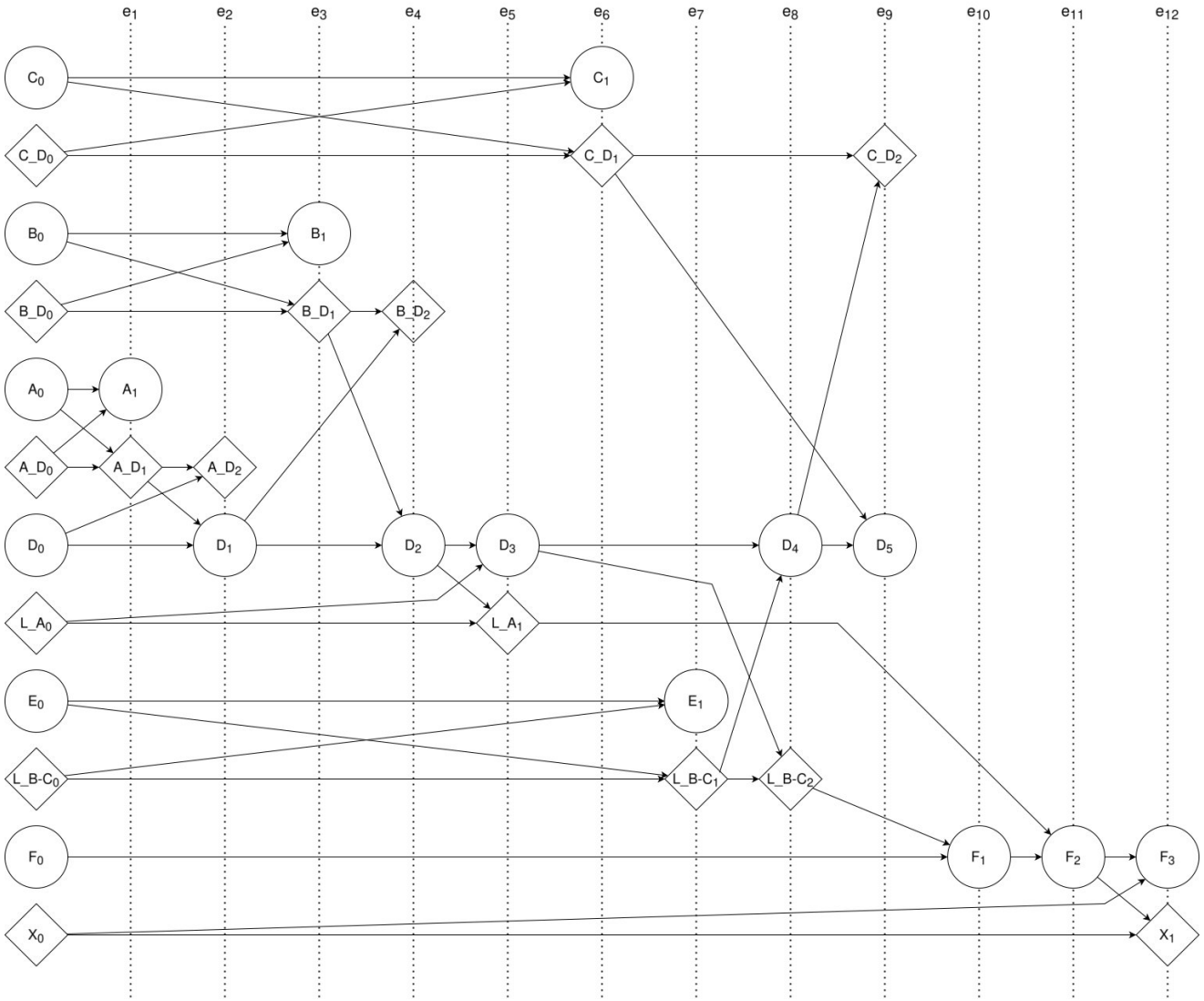
Proposition

If information flowed from x to y between events e_j and e_k in trace T,
then there is a path in $SEPG_T$ from $x_{v_j(x)}$ to $y_{v_k(y)}$.

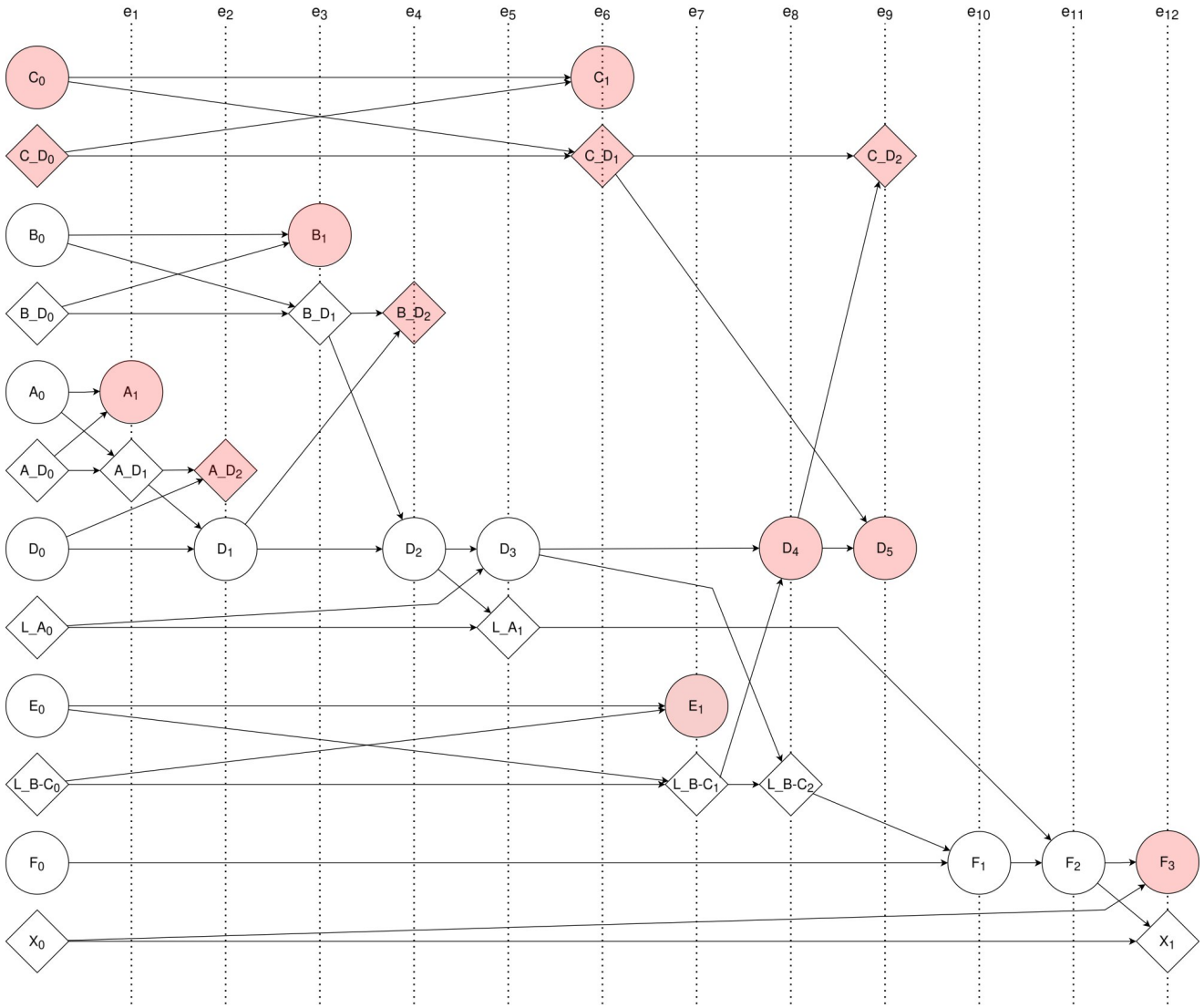
The converse is NOT true

A path in the SEPG does not guarantee a real information flow.
This is the over-linking problem — the gap between possible and actual information flow.

Formalism · 4/4 — Reachability



Formalism · 4/4 — Reachability



Formalism · 4/4 — Reachability

Proposition

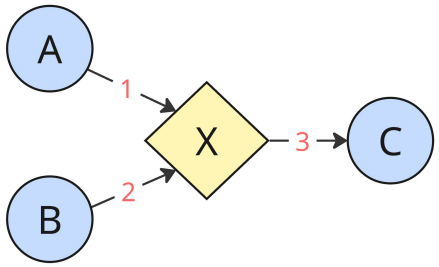
If information flowed from x to y between events e_j and e_k in trace T,
then there is a path in $SEPG_T$ from $x_{v_j(x)}$ to $y_{v_k(y)}$.

The converse is NOT true

A path in the SEPG does not guarantee a real information flow.
This is the over-linking problem — the gap between possible and actual information flow.

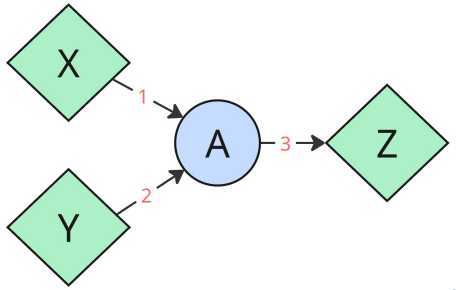
Two over-linking motifs we will mitigate:

Inter-process: through a shared resource



A → X → C ?
B → X → C ?
A + B → X → C ?

Intra-process: inside a single process



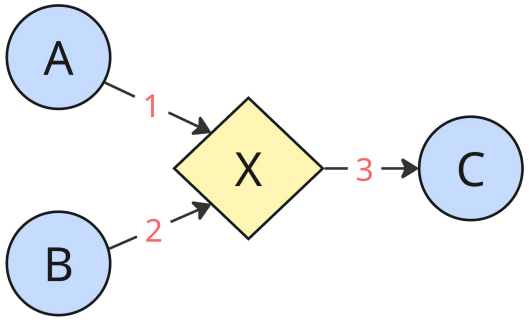
X → A→ Z ?
Y → A → Z ?
X + Y → A → Z ?

Outline

1. **Context & motivation**
   Vulnerability investigation in partially known systems
2. **State-Expanded Provenance Graphs formalism**
   Events, traces, state-versions, graph construction
3. **Mitigating over-linking**
   Inter-process and intra-process refinement
4. **The proposed toolchain**
   Capture software and analysis application
5. **Illustrative example**
   Application of the method with demo video
6. **Limitations & next steps**

Inter-process pruning 1/2

Objective: express how information flows through a resource



e_1 = (write, A, {A, X}, {A, X}, (buf="hello"), (size_written =5))

e_2 = (write, B, {B, X}, {B, X}, (buf="world"), (size_written =5))

e_3 = (read, C, {C, X}, {C, X}, ·, (buf="hello", size_read=5))

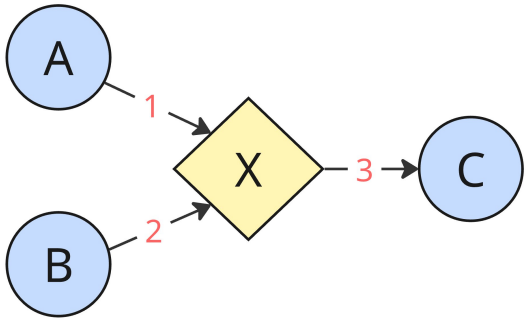
Idea: model the resource interactions to deduce its current state.

Example of a FIFO: modeling a simple queue behavior

→ Each write enqueues a chunk tagged with its writer state-version.

→ Each read dequeues from the head, splitting or merging chunks as needed to yield a chunk of the resquested size

Inter-process pruning 2/2



e_1 = (write, A, {A, X}, {A, X}, (buf="hello"), (size_written =5))

e_2 = (write, B, {B, X}, {B, X}, (buf="world"), (size_written =5))

e_3 = (read, C, {C, X}, {C, X}, ·, (buf="hello", size_read=5))

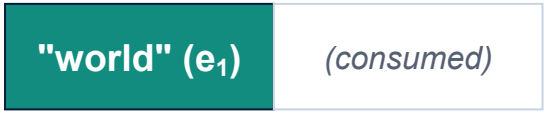
Representation of FIFO X :

Before C reads:

✓ asserted: $e_1 \leftarrow e_3$ ✗ discarded: $e_2 \leftarrow e_3$

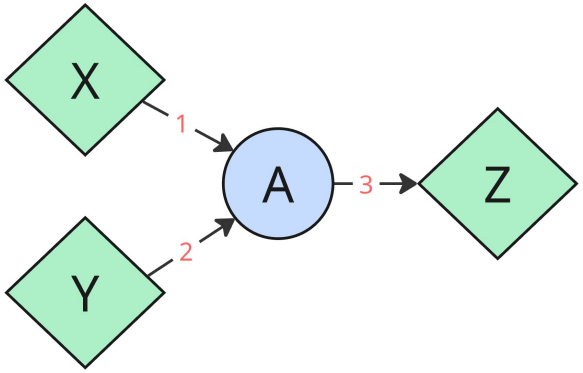
C: read(5) → consumes "hello" (A's chunk)

After:



Intra-process pruning 1/3 : objective

Objective: express how information flows through a process



e_1 = (read, A, {A, X}, {A}, · , (buf="hello", size_read=5))

e_2 = (read, A, {A, Y}, {A}, · , (buf="world", size_read=5))

e_3 = (write, A, {A, Z}, {A, Z}, (buf="HELLO") , (size_written=5))

→ **This is expressing the implementation details of the process**

→ Rather than complex logic tracking and process emulation,
we are simply interested in expressing relation between events produced

Intra-process pruning 2/3 : causal properties

A trace is too big. We need a way to describe general rules, not per-event ones

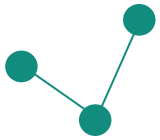
An Event pattern π is a predicate over event fields.

Example: $\pi = \{ p_e == A,\ \ t_e == read,\ \ X \in S_e \}$

A Causal property $(p, \pi_s, \pi_t, mode, \alpha)$

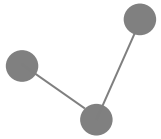
p — described process
π_s / π_t — source and target event patterns
mode — dependency mode (DEP or INDEP)
α — alignement constraint (see next slide)

Each candidate intra-process flow ends up in one of three buckets:



Asserted

DEP property matches: source–target link confirmed.



Discarded

INDEP property matches: link ruled out.

Uncertain

No property matches: residual ambiguity.

Intra-process pruning 3/3 : alignement constraint

Scenario: process A reads a filename from a request FIFO, then opens that file.

π_1 = { t_e == read, fifo_request ∈ S_e }

π_2 = { t_e == open }

CP = (A, π_1, π_2, DEP, α)

with α : (e_1, e_2) => {out_{e1}.buf == S_{e2}.resource.name}

Read it: a read on fifo_request is the source of a later open by A only if the buffer it returned matches the resource name it then opens.

Why it matters: alignment encodes data-dependent semantics — not just structural patterns. Event patterns define per-event details, causal properties define cross-event linkage.

Three layers, applied cumulatively

Full SEPG (raw, all reachability)

1. Backward reachability from target

Drops everything not on a path to the asset of interest.

2. Inter-process content models

Cuts edges through shared resources that don't carry the relevant bytes.

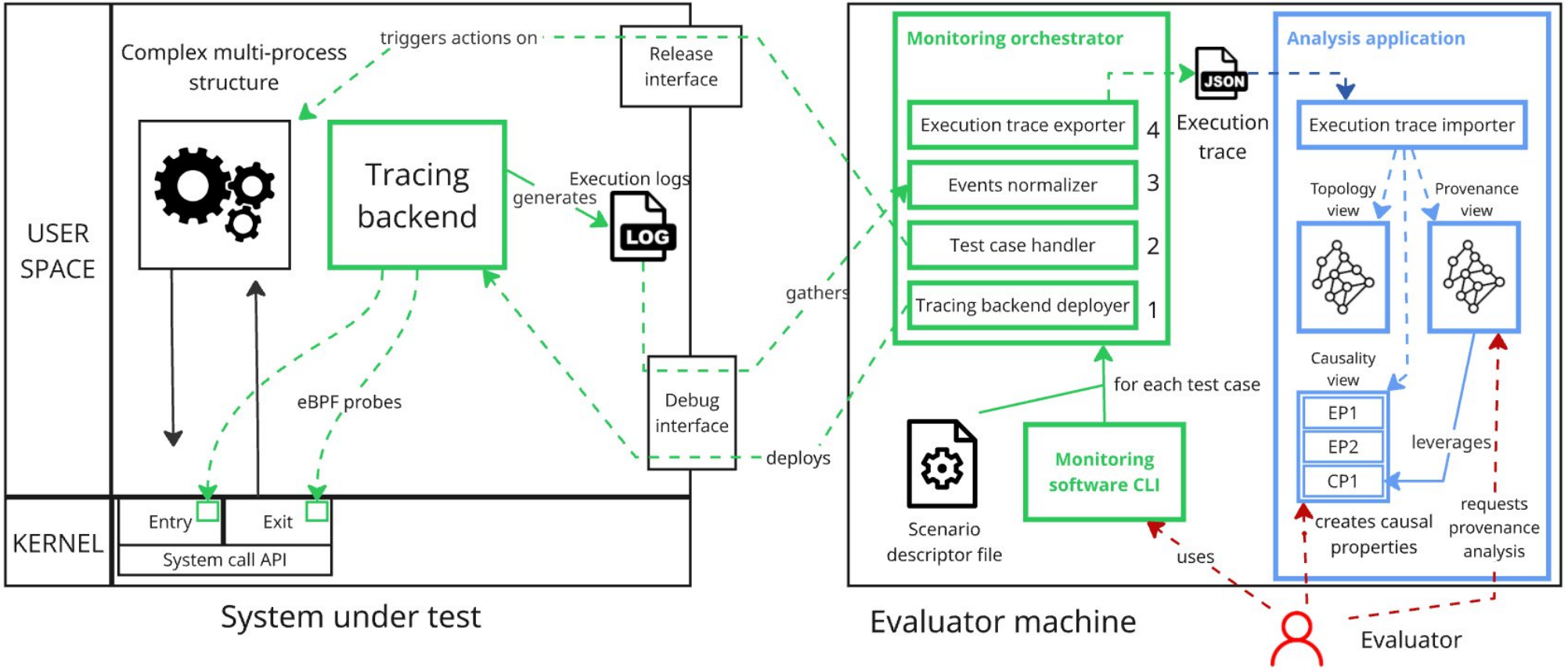
3. Causal properties (DEP / INDEP)

Carves intra-process flows into asserted, discarded, uncertain.

Outline

1. **Context & motivation**
   Vulnerability investigation in partially known systems
2. **State-Expanded Provenance Graphs formalism**
   Events, traces, state-versions, graph construction
3. **Mitigating over-linking**
   Inter-process and intra-process refinement
4. **The proposed toolchain**
   Capture software and analysis application
5. **Illustrative example**
   Application of the method with demo video
6. **Limitations & next steps**

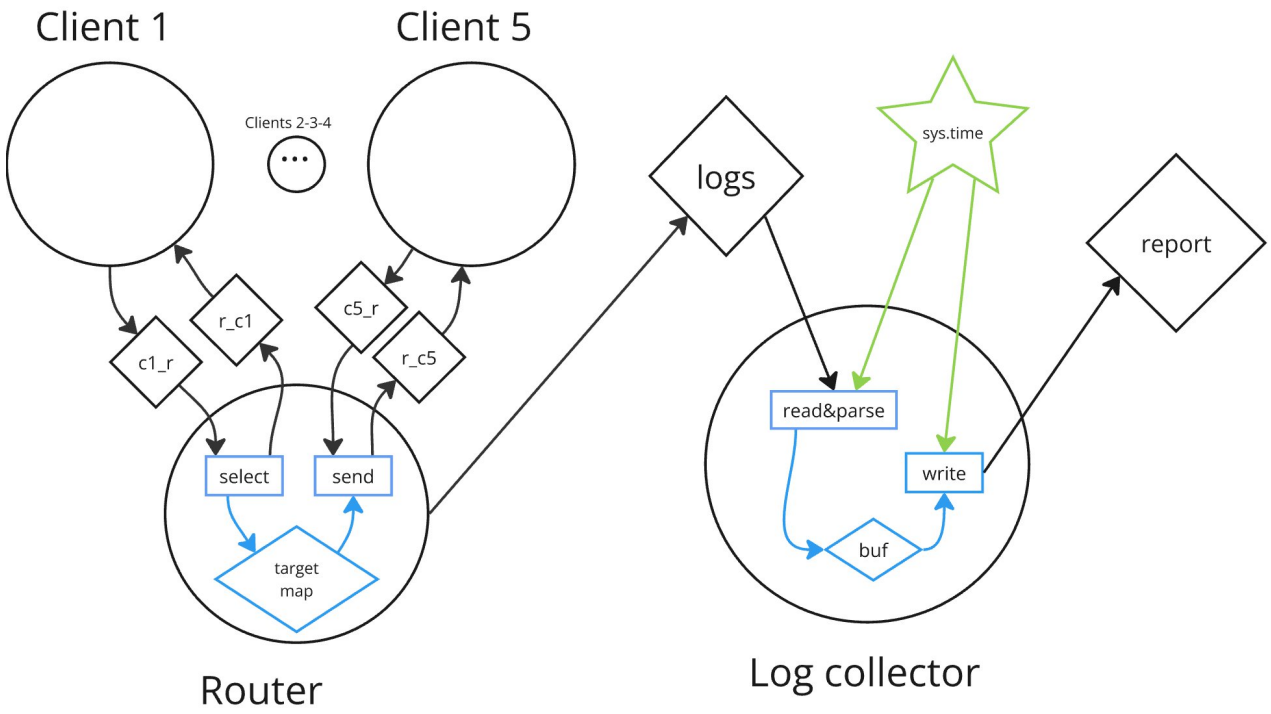
Overview of the toolchain



Outline

1. **Context & motivation**
   Vulnerability investigation in partially known systems
2. **State-Expanded Provenance Graphs formalism**
   Events, traces, state-versions, graph construction
3. **Mitigating over-linking**
   Inter-process and intra-process refinement
4. **The proposed toolchain**
   Capture software and analysis application
5. **Illustrative example**
   Application of the method with demo video
6. **Limitations & next steps**

Example 1/3: system overview



What the trace shows: 5 clients exchange messages through the router, which logs every transit. The log-collector periodically writes digests to the report file. Report is the entity of interest.

The vulnerability: the protocol lets a client supply its own identity → identity spoofing → a corrupted entry appears in the report.

Let's find which client is at fault of such spoofing, and expose the information flow to it from report !

Example 2/3: find over-linking causes

Log-collector

Intra-process · 1 property

Dependency property: read on logs ⇒ write on report

Alignement constraint can mimic parsing, or just discriminating substract of it

Logs file

Inter-process · automatic

The storage-resource model walks the read offsets and write extents. The log-collector's read is linked only to the specific router writes that produced those bytes; no property needed.

Router

Intra-process · 1 property

Dependency property: read on a client FIFO ⇒ write on logs.

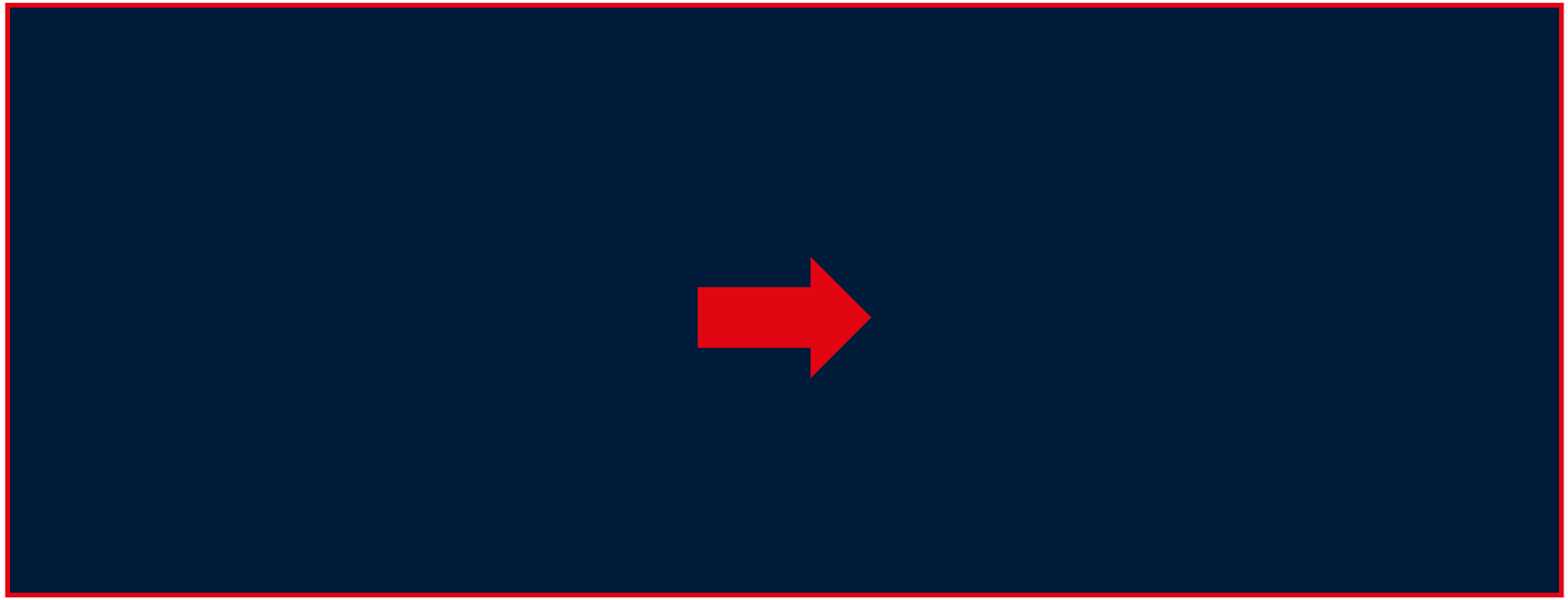
Alignement constraint has to express

Outcome: an asserted causal chain from the corrupted report fragment back to the client whose message triggered it.

Example 3/3 — live walkthrough

DEMO

Live walkthrough on the example system



Performances

Processes / FIFOs	5 / 6
Events captured	1,698
SEPG nodes / edges	3,285 / 6,476
SEPG build time (ms)	14
Uncertain (raw query)	**2,771**
Asserted (after pruning)	**20**
Provenance query time (ms)	90

Take-away · build/query stay sub-second; the asserted chain stays small enough to read end-to-end.

Outline

1. **Context & motivation**
   Vulnerability investigation in partially known systems
2. **State-Expanded Provenance Graphs formalism**
   Events, traces, state-versions, graph construction
3. **Mitigating over-linking**
   Inter-process and intra-process refinement
4. **The proposed toolchain**
   Capture software and analysis application
5. **Illustrative example**
   Application of the method with demo video
6. **Limitations & next steps**

Limitations and future works

LIMITATIONS

- Dynamically spawned processes leave silent gaps
- Property quality depends on the evaluator (but this allows “what-if” analysis)
- Single-trace analysis: real design vulnerabilities often need composition of interactions

FUTURE WORK

- Systematic evaluation on a real industrial system (in progress)
- Auto-discover process set (inotify-style)
- Multiple-trace analysis — combine effects of multiples traces to build attack scenarios
- Auto-generate properties from binaries (taint + symbolic exec)

In conclusion — *minimal capture + a stateful graph + evaluator-stated knowledge = an investigable model of a system the evaluator only partially understands.*

Questions

Thank you

Identifying potential attack scenarios in complex systems by means of provenance graphs

Contact: loic.robert@laas.fr

Tool are freely available on my Github !

Where this sits — related work

CAPTURE

Whole-system provenance

Hi-Fi, SPADE, CamFlow.
Rich capture but dense graphs still need pruning.

PARTITIONING

Execution units

BEEP, MPI, ProTracer
Split processes into smaller logical units.
Strong on main loop with handlers architecture

APP SEMANTICS

Log fusion

OmegaLog, ALCHEMIST
Align app logs with audit.
Concise graphs where logs are stable.

HARDWARE

Hardware-assisted

PalanTír
Instruction-level taint via Intel PT / ARM ETM.
High fidelity, restricted hardware base.

Our positioning: *minimal capture, application-agnostic, hardware-agnostic, evaluator-centered*
Can be applied to vulnerability research where the trace does not contain an attack.

What our approach is — and is not — for

IN SCOPE

Vulnerability investigation

- Single host, multiple processes
- Communication via files, FIFOs, pipes, sockets
- Monitored functional executions
- Iterative, evaluator-in-the-loop investigation
- Variable granularity (depends on monitoring and modeling effort)
- Discovery of design vulenrabilities

OUT OF SCOPE

What we don't claim to do

- Real-time intrusion detection or forensics
- Soundness / completeness: model is evaluator-driven
- Outperform specialized state-of-the-art methods
- Distributed systems (single-host clock assumed)
- Shared memory communication tracking
- Implementation vulnerability, bugs in code