Slides 6.8.23

Math

- Preorder is – Reflexive &Transitive
- If anti-symmetric then partial order
- Homomorphisms bestow a preorder
- Ensure we understand up/down set…

Definition 3 (Up-/down-sets). *Given a preorder* $(\mathcal{P}, \leqslant)$*, a set* $\mathcal{S} \subseteq \mathcal{P}$ *is an* up-set *(or* order filter*) iff for all structures* G *and* H*, whenever* $G \in \mathcal{S}$ *and* $G \leqslant H$*, then* $H \in \mathcal{S}$*.* $\mathcal{S}$ *is a* down-set *(or* order ideal*) iff for all structures* G *and* H*, whenever* $H \in \mathcal{S}$ *and* $G \leqslant H$*, then* $G \in \mathcal{S}$*.*

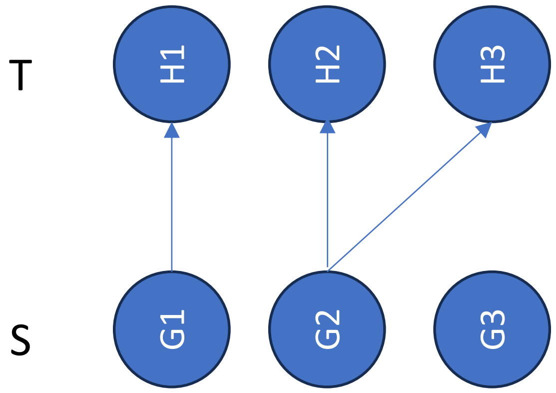
The upward closure *of a set* $\mathcal{S}$ *is* $\phi(\mathcal{S}) = \{H \in \mathcal{P} \mid \exists G \in \mathcal{S} \wedge G \leqslant H\}$*. Similarly the* downward closure *of a set* $\mathcal{S}$ *is* $\iota(\mathcal{S}) = \{G \in \mathcal{P} \mid \exists H \in \mathcal{S} \wedge G \leqslant H\}$*.*

The symbols ϕ and ι reflect the terminology of order filters and order ideals.

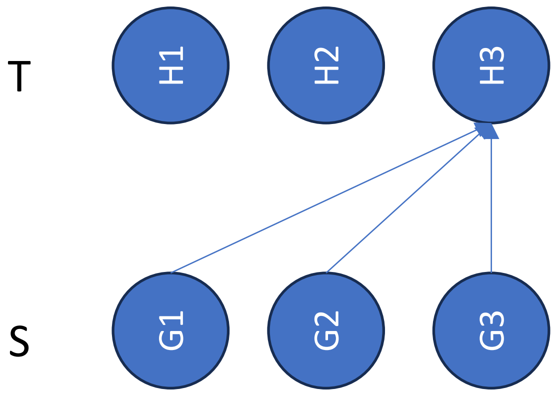
Covers / supports

Definition 10 (Supports, Covers). *Given two sets of graphs $\mathcal{S}$ and $\mathcal{T}$, we say that $\mathcal{S}$* supports *$\mathcal{T}$ iff for every $H \in \mathcal{T}$, there is some $G \in \mathcal{S}$, such that $G \leqslant H$. We say that $\mathcal{T}$* covers *$\mathcal{S}$ iff for every $G \in \mathcal{S}$ there is some $H \in \mathcal{T}$ such that $G \leqslant H$.*

S supports T



T covers S



Copland preorder

$$c_1 \preceq^C_{\mathcal{I}} c_2 \; \mathit{iff} \; \{[\![\, c_2 \,]\!]^e_p\} \; \mathit{covers} \; \{[\![\, c_1 \,]\!]^e_p\}$$
$$c_1 \preceq^C_{\mathcal{F}} c_2 \; \mathit{iff} \; \{[\![\, c_2 \,]\!]^e_p\} \; \mathit{supports} \; \{[\![\, c_1 \,]\!]^e_p\}$$

$$c_1 \preceq^C_{\mathcal{I}} c_2 \text{ iff } [\![\, c_1 \,]\!]^e_p \leqslant [\![\, c_2 \,]\!]^e_p$$
$$c_1 \preceq^C_{\mathcal{F}} c_2 \text{ iff } [\![\, c_2 \,]\!]^e_p \leqslant [\![\, c_1 \,]\!]^e_p$$

6.2 slides

Input to the problem

- Set of measurement operations
- Ranking of corruption events
- Dependency structure?

Principles

1. Increasing volume of measurement operations may not confine the adversary
2. Stronger measurements mimic the dependency chain
   - If a depends on b which depends on c (all at some place p1) then ms(p1,rtm, p1,c) -> ms (p1, c, p1, b) -> ms(p1, b, p1, a) will be the strongest measurement
   - Same idea as *well-supported* measurement from confining paper
   - Hypothesis: if measurement chain is not well-supported, then it is “easy” for an adversary to corrupt

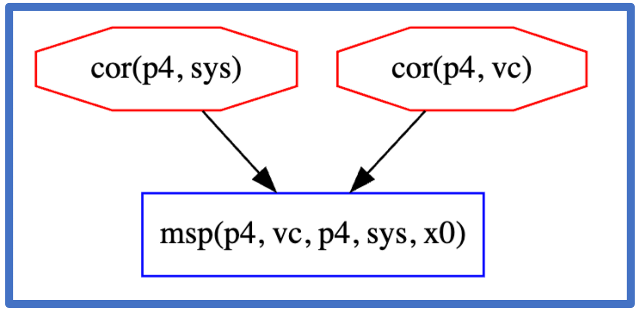
Definition 6. *A measurement event* $e = \mathsf{ms}(o_2, o_1)$ *in execution* E *is* well-supported *iff either*

i. $o_2 = \mathsf{rtm}$, *or*
ii. for every $o \in D^1(o_1)$, *there is a measurement event* $e' \prec_E e$ *such that* o *is the target of* e'.

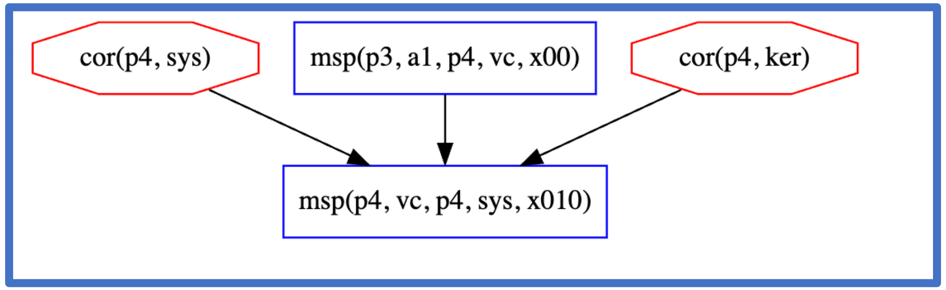
When e *is well-supported, we call the set of* e' *from Condition ii above the* support *of* e. *An execution* E measures bottom-up *iff each measurement event* $e \in E$ *is well-supported.*

a
b
c
rtm

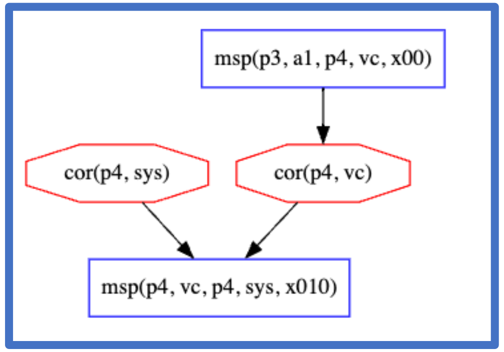
Try different ordering



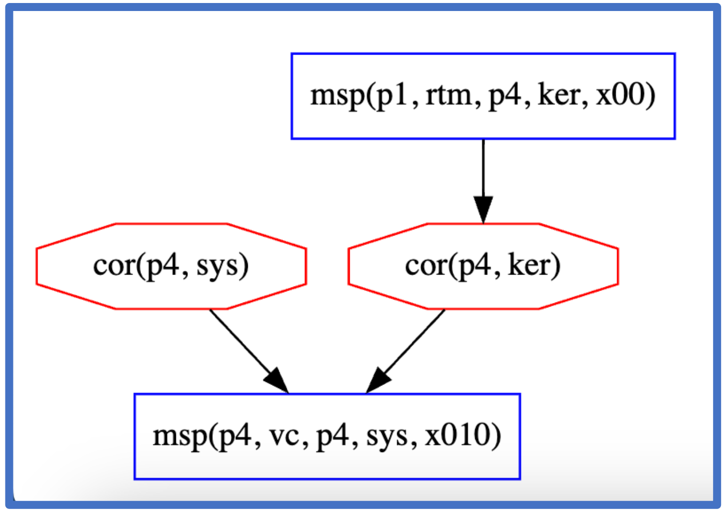
<



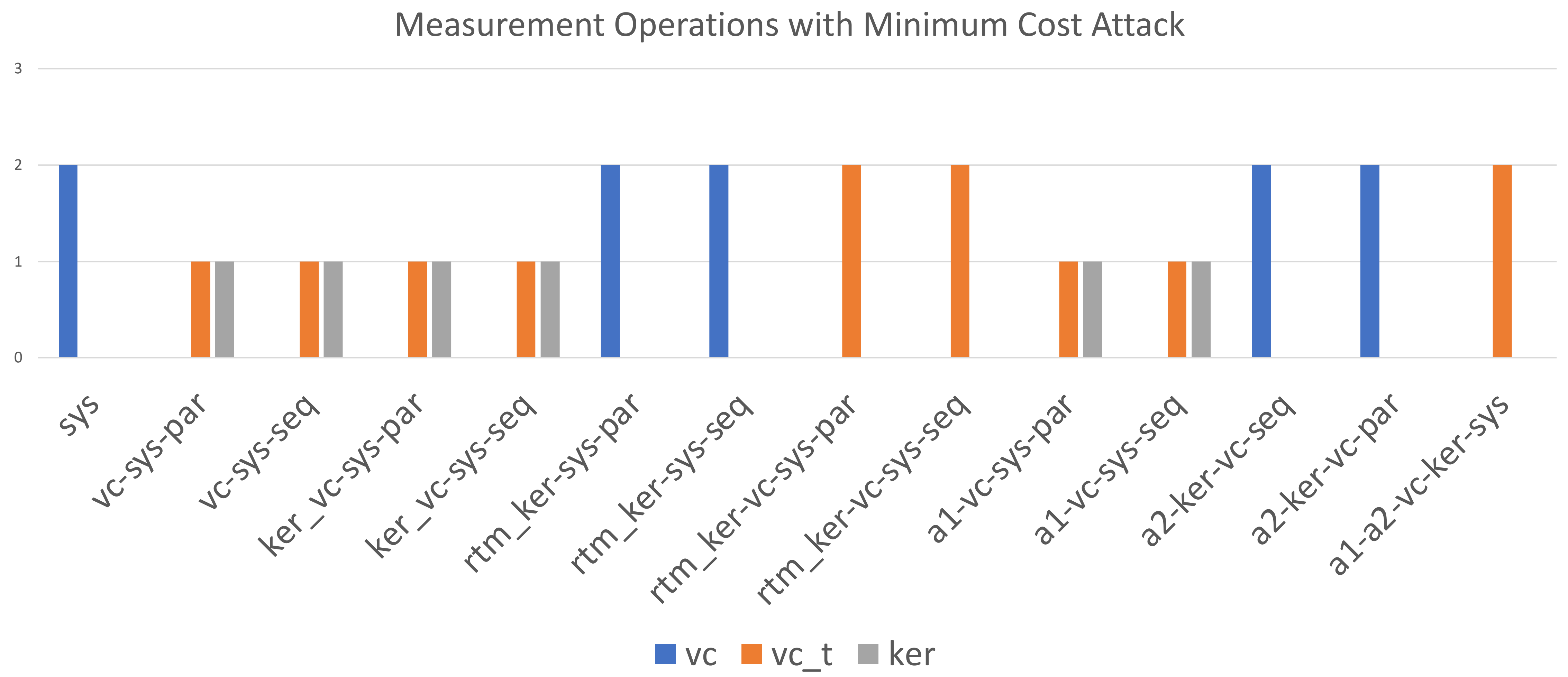
<



<



Protocol Name	Protocol
sys	*target: @p4 [vc p4 sys]
rtm_ker-sys-par	*target: @p1 [rtm p4 ker +~+ @p4 vc p4 sys]
rtm_ker-sys-seq	*target: @p1 [rtm p4 ker +<+ @p4 vc p4 sys]
ker_vc-sys-par	*target: @p4 [(ker p4 vc) +~+ @p4 (vc p4 sys)]
ker_vc-sys-seq	*target: @p4 [(ker p4 vc) +<+ @p4 (vc p4 sys)]
rtm_ker-vc-sys-par	*target: @p1 [(rtm p4 ker) +~+ @p4 [(ker p4 vc) +~+ @p4 (vc p4 sys)]]
rtm_ker-vc-sys-seq	*target: @p1 [(rtm p4 ker) +<+ @p4 [(ker p4 vc) +<+ @p4 (vc p4 sys)]]
vc-sys-par	*target: @p3 [(a1 p4 vc) +~+ @p4 (vc p4 sys)]
vc-sys-seq	*target: @p3 [(a1 p4 vc) +<+ @p4 (vc p4 sys)]
a1-vc-sys-seq	*target: @p1 [(rtm p3 a1) +<+ @p3 [(a1 p4 vc) +<+ @p4 (vc p4 sys)]]
a1-vc-sys-par	*target: @p1 [(rtm p3 a1) +~+ @p3 [(a1 p4 vc) +~+ @p4 (vc p4 sys)]]
a2-ker-vc-seq	*target: @p1 [rtm p3 a2 +<+ @p3 [a2 p4 ker +<+ @p4 (vc p4 sys)]]
a2-ker-vc-par	*target: @p1 [(rtm p3 a2) +~+ @p3 [(a2 p4 ker) +~+ @p4 (vc p4 sys)]]
a1-a2-vc-ker-sys	*target: @p1 [(rtm p3 a1 +~+ rtm p3 a2) +<+ @p3 [(a1 p4 vc +~+ a2 p4 ker) +<+ @p4 (vc p4 sys1)]]


Measurement Operations with Minimum Cost Attack
3
2
1
0
sys
vc-sys-par
vc-sys-seq
ker_vc-sys-par
ker_vc-sys-seq
rtm_ker-sys-par
rtm_ker-sys-seq
rtm_ker-vc-sys-par
rtm_ker-vc-sys-seq
a1-vc-sys-par
a1-vc-sys-seq
a2-ker-vc-seq
a2-ker-vc-par
a1-a2-vc-ker-sys
vc
vc_t
ker

5.26 slides

Add protocols

- *target: @p4 [ker p4 vc] +~+ @p4 [vc p4 sys]
- *target: @p4 [ker p4 vc] +<+ @p4 [vc p4 sys]

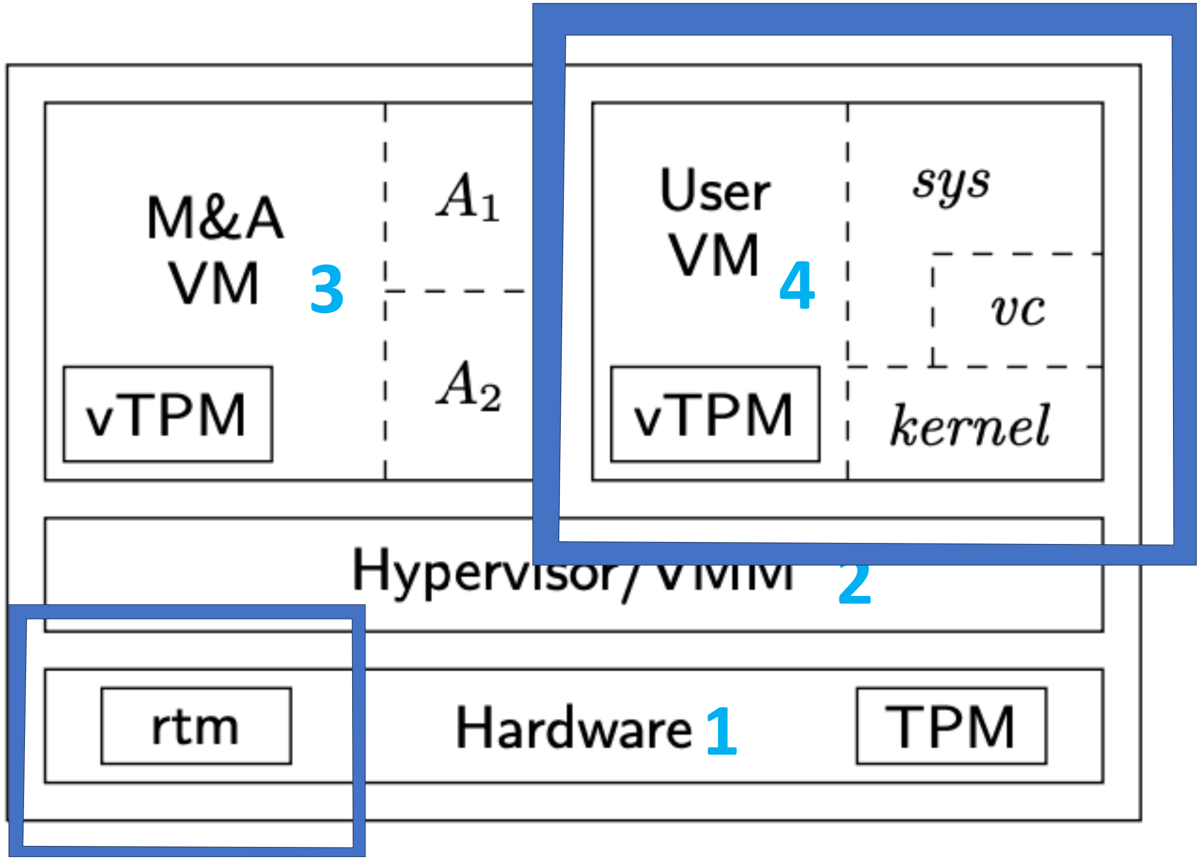
- *target: @p1 [(rtm p4 ker) +~+ @p4 [(ker p4 vc) +~+ @p4 (vc p4 sys)]]
- *target: @p1 [(rtm p4 ker) +<+ @p4 [(ker p4 vc) +<+ @p4 (vc p4 sys)]]

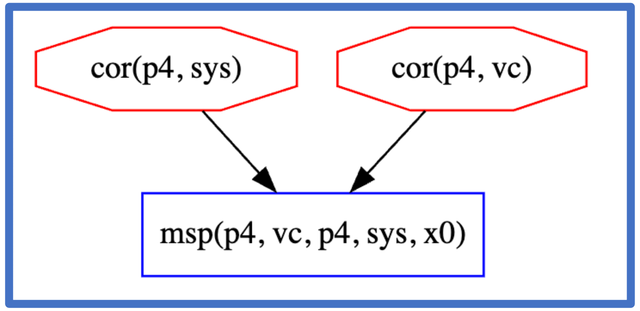
Protocols

Protocol Name	Protocol
sys	*target: @p4 [vc p4 sys]
rtm_ker-sys-par	*target: @p1 [rtm p4 ker +~+ @p4 vc p4 sys]
rtm_ker-sys-seq	*target: @p1 [rtm p4 ker +<+ @p4 vc p4 sys]
ker_vc-sys-par	*target: @p4 [(ker p4 vc) +~+ @p4 (vc p4 sys)]
ker_vc-sys-seq	*target: @p4 [(ker p4 vc) +<+ @p4 (vc p4 sys)]
rtm_ker-vc-sys-par	*target: @p1 [(rtm p4 ker) +~+ @p4 [(ker p4 vc) +~+ @p4 (vc p4 sys)]]
rtm_ker-vc-sys-seq	*target: @p1 [(rtm p4 ker) +<+ @p4 [(ker p4 vc) +<+ @p4 (vc p4 sys)]]
vc-sys-par	*target: @p3 [(a1 p4 vc) +~+ @p4 (vc p4 sys)]
vc-sys-seq	*target: @p3 [(a1 p4 vc) +<+ @p4 (vc p4 sys)]
a1-vc-sys-seq	*target: @p1 [(rtm p3 a1) +<+ @p3 [(a1 p4 vc) +<+ @p4 (vc p4 sys)]]
a1-vc-sys-par	*target: @p1 [(rtm p3 a1) +~+ @p3 [(a1 p4 vc) +~+ @p4 (vc p4 sys)]]
a2-ker-vc-seq	*target: @p1 [rtm p3 a2 +<+ @p3 [a2 p4 ker +<+ @p4 (vc p4 sys)]]
a2-ker-vc-par	*target: @p1 [(rtm p3 a2) +~+ @p3 [(a2 p4 ker) +~+ @p4 (vc p4 sys)]]
a1-a2-vc-ker-sys	*target: @p1 [(rtm p3 a1 +~+ rtm p3 a2) +<+ @p3 [(a1 p4 vc +~+ a2 p4 ker) +<+ @p4 (vc p4 sys1)]]

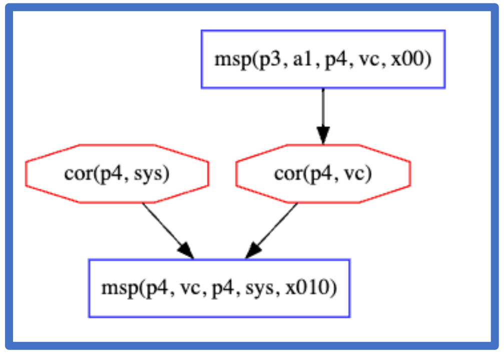
Say we have the architecture from “Confining the Adversary” Paper

- ms(rtm, A1)
- ms(rtm, A2)
- ms(A1, vc)
- ms(A2, ker)
- msker (vc, sys)

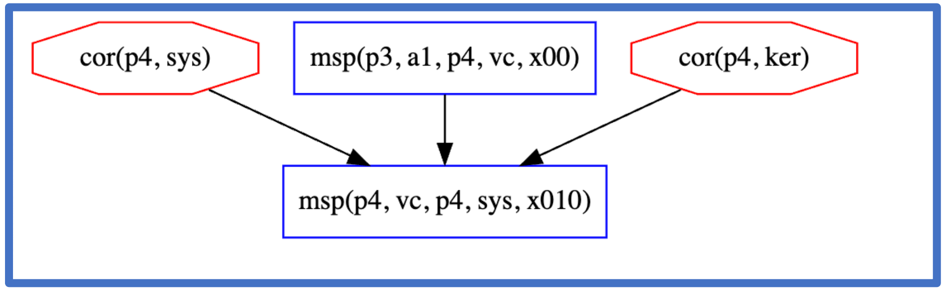



cor(p4, sys)
cor(p4, vc)
msp(p4, vc, p4, sys, x0)

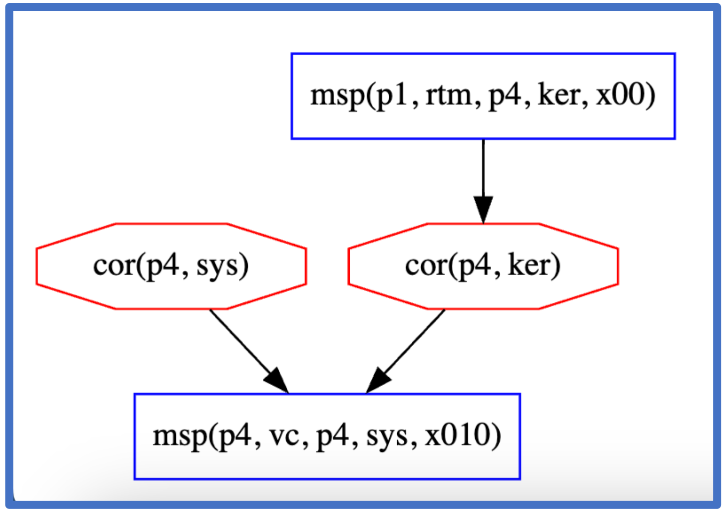
<


msp(p3, a1, p4, vc, x00)
cor(p4, sys)
cor(p4, vc)
msp(p4, vc, p4, sys, x010)

<


cor(p4, sys)
msp(p3, a1, p4, vc, x00)
cor(p4, ker)
msp(p4, vc, p4, sys, x010)

<


msp(p1, rtm, p4, ker, x00)
cor(p4, sys)
cor(p4, ker)
msp(p4, vc, p4, sys, x010)

Corruption event order

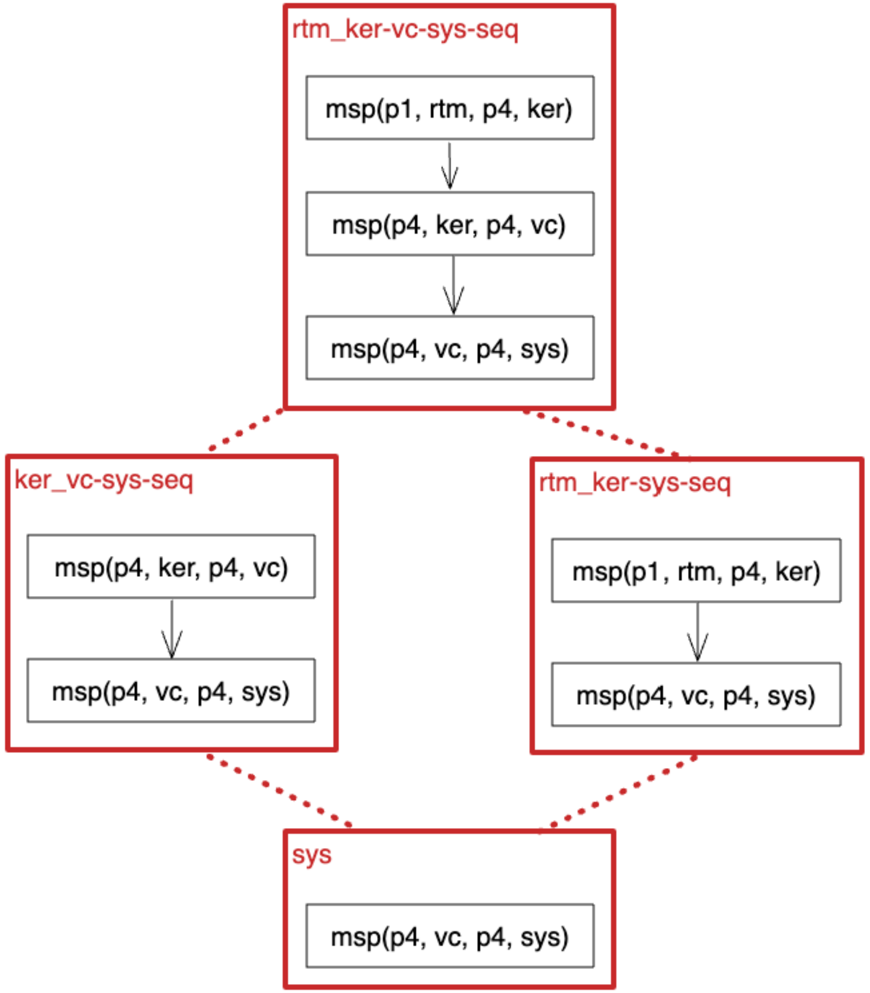
p4 < p1

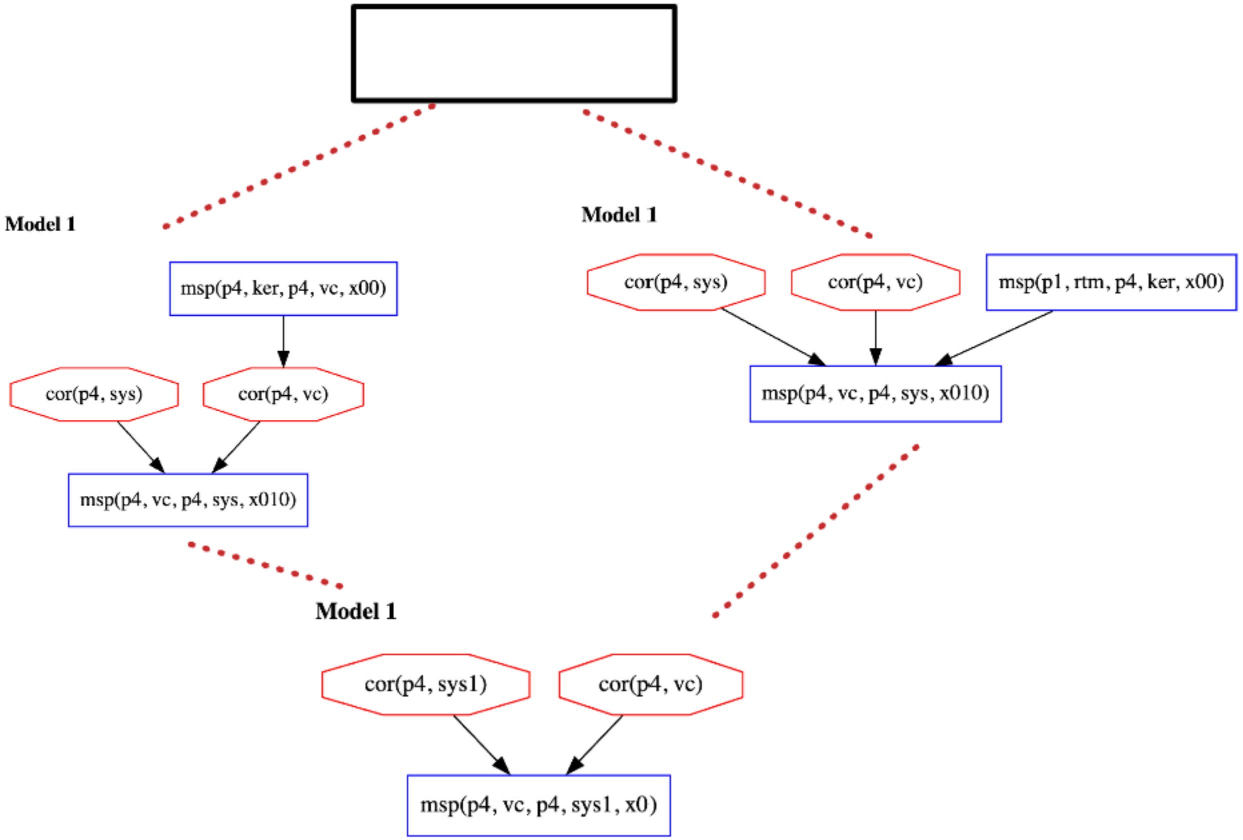
- To corrupt something at p4 is lowest cost to adversary
- Corrupting at p1 (rtm location) is the highest cost

sys < vc < vc_t < ker < ker_t

- Corrupting the system is easiest for the adversary
- Denote x_t as a corruption event of x in a timely manner

- Possible lattice of protocols





- Ordering relation = subset relation

Slides (5/16)

Goals of cost analysis

* Maximize the adversary’s minimum cost

- Cost considerations
  - Weight of cost depends on component
  - Cor-rep-cor should be more expensive then just corrupt
  - Cost of repair
    - Maybe repair is not costly to an adversary
    - Maybe need to model partial repair (need to decompose system further for this)

What I did for today (5/16)

1. Ran all protocols
2. Selected lowest cost/costs model
3. Took maximum of lowest costs to produce best (?) protocol

Principles

- Increase cost after start event
- Any corruption of a deeper component is higher cost
- Add weight to cost event
  - Have some base cost to the corruption event
  - Add to cost more if its in a protected place

Protocols

Protocol Name	Protocol
sys	*target: @p4 [vc p4 sys1]
vc-sys-par	*target: @p3 [(a1 p4 vc) +~+ @p4 (vc p4 sys)]
vc-sys-seq	*target: @p3 [(a1 p4 vc) +<+ @p4 (vc p4 sys)]
a1-vc-sys-seq	*target: @p1 [(rtm p3 a1) +<+ @p3 [(a1 p4 vc) +<+ @p4 (vc p4 sys)]]
a1-vc-sys-par	*target: @p1 [(rtm p3 a1) +~+ @p3 [(a1 p4 vc) +~+ @p4 (vc p4 sys)]]
a2-ker-vc-seq	*target: @p1 [rtm p3 a2 +<+ @p3 [a2 p4 ker +<+ @p4 (vc p4 sys)]]
a2-ker-vc-par	*target: @p1 [(rtm p3 a2) +~+ @p3 [(a2 p4 ker) +~+ @p4 (vc p4 sys)]]
a1-a2-vc-ker-sys	*target: @p1 [(rtm p3 a1 +~+ rtm p3 a2) +<+ @p3 [(a1 p4 vc +~+ a2 p4 ker) +<+ @p4 (vc p4 sys1)]]

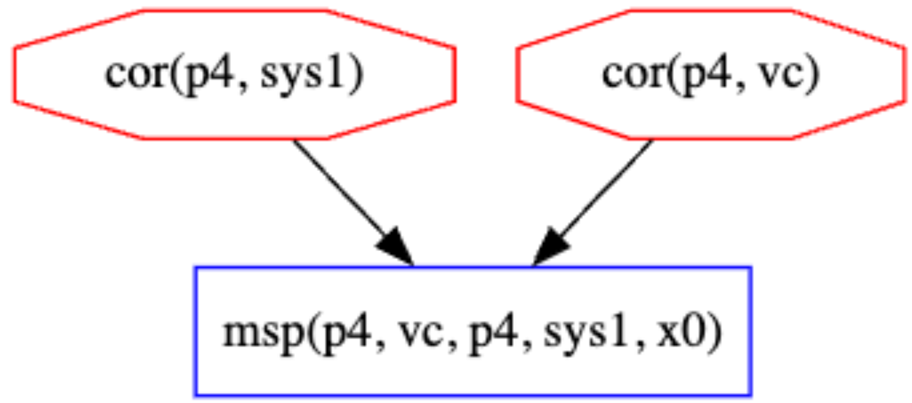
sys

Protocol: *target: @p4 [vc p4 sys1]

Problem Configuration

```
[ bound = 500, limit = 5000, input_order ]

% Assume adversary avoids detection at our main measurement
% event. This is a measurement of sys
l(V) = msp(p4, M, p4, sys1, X)
 => corrupt_at(p4, sys1, V).

% Assume no dependencies

% No recent assumptions
% prec(V, V1) & l(V1) = cor(P,C) & ms_evt(V)
%  => false.

% No deep assumptions
l(V) = cor(p1, M) => false.

m4_include(`sys.gli')m4_dnl

m4_include(`sys_dist.gli')m4_dnl

m4_include(`thy.gli')m4_dnl
```

Model 1



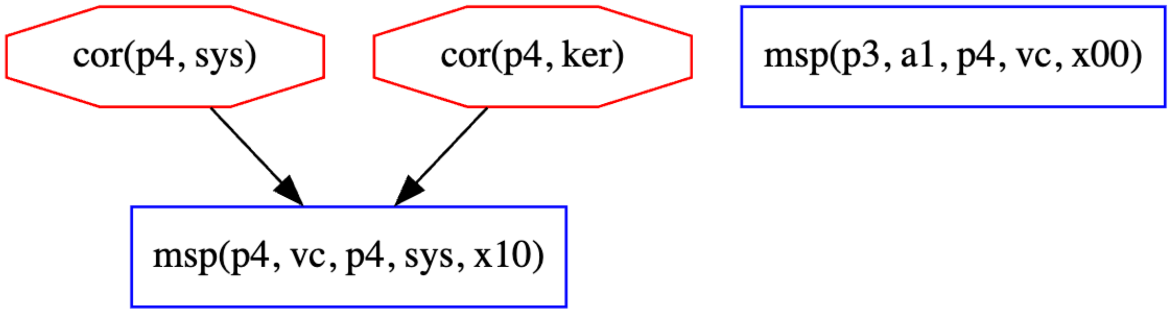
vc-sys-par

- *target: @p3 [(a1 p4 vc) +~+ @p4 (vc p4 sys)]

Problem Configuration

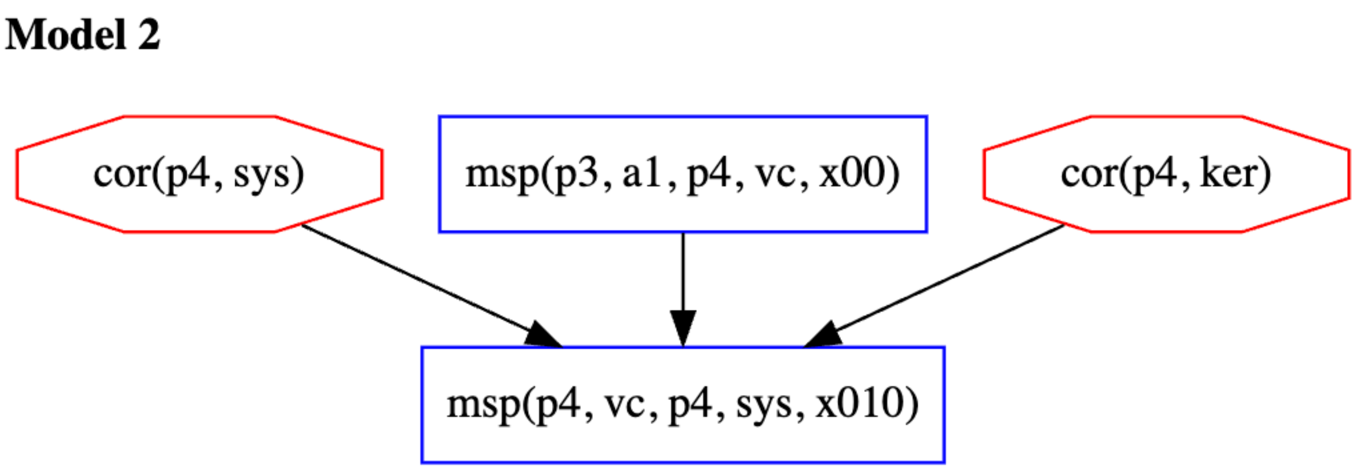
```
[ bound = 500, limit = 5000, input_order ]

% Assume adversary avoids detection at
% our main measurement event.
% This is a measurement of sys.
l(V) = msp(p4, M, p4, sys, X)
 => corrupt_at(p4, sys, V).

% Assume dependencies
depends(p4, C, p4, sys) => C = ker.
depends(p4, C, p4, vc) => C = ker.
% rtm has no dependencies
depends(p1, C, p1, rtm) => false.

% Assume no recent corruptions
%prec(V, V1) & l(V1) = cor(P,C) & ms_evt(V)
%=> false.

% Assume no deep corruptions
l(V) = cor(p1, M) => false.

m4_include(`vc-sys-par.gli')m4_dnl

m4_include(`vc-sys-par_dist.gli')m4_dnl

m4_include(`thy.gli')m4_dnl
```

Model 2



vc-sys-seq

- *target: @p3 [(a1 p4 vc) +<+ @p4 (vc p4 sys)]



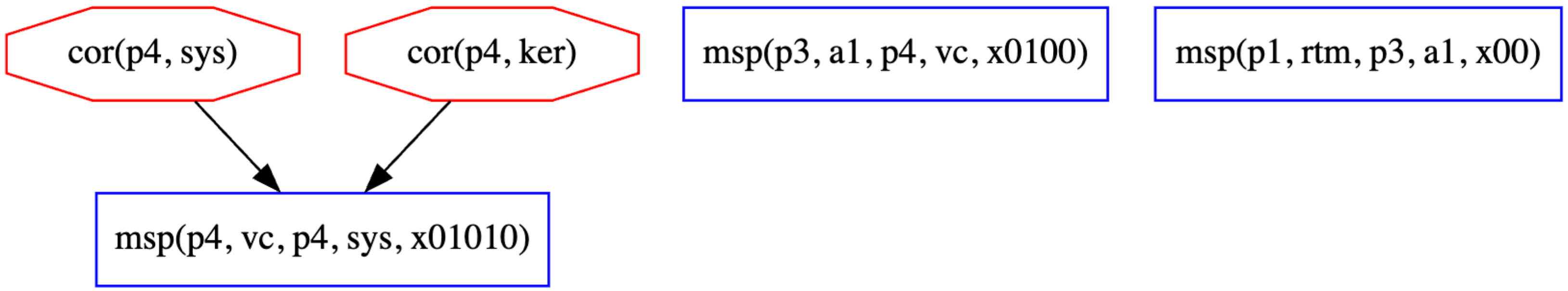
a1-vc-sys-par

Protocol:

*target: @p1 [(rtm p3 a1) +~+ @p3 [(a1 p4 vc) +~+ @p4 (vc p4 sys)]]

```
% Assume adversary avoids detection at
% our main measurement event.
% This is a measurement of sys.
l(V) = msp(p4, M, p4, sys, X)
 => corrupt_at(p4, sys, V).

% system dependencies
depends(p4, C, p4, sys) => C = ker.
depends(p4, C, p4, vc) => C = ker.

% rtm has no dependencies
depends(p1, C, p1, rtm) => false.


% Assume no recent corruptions
% prec(V, V1) & l(V1) = cor(P,C) & ms_evt(V)
% => false.

% Assume no deep corruptions
l(V) = cor(p1, M) => false.
```

Model 2

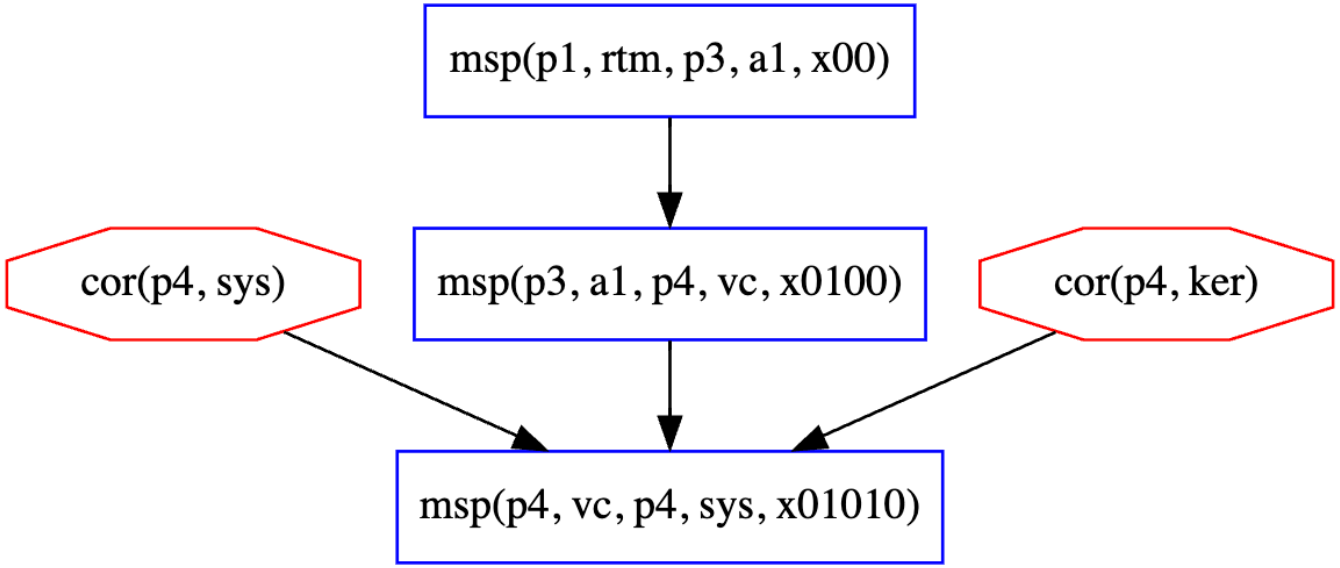


A1-vc-sys-seq

Protocol:

*target: @p1 [(rtm p3 a1) +<+ @p3 [(a1 p4 vc) +<+ @p4 (vc p4 sys)]]

Model 2



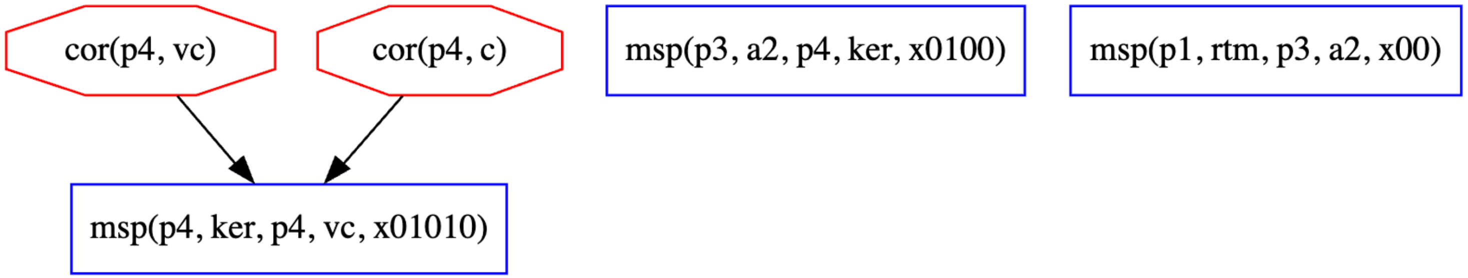
a2-ker-vc-par

- *target: @p1 [(rtm p3 a2) +~+ @p3 [(a2 p4 ker) +~+ @p4 (ker p4 vc)]]

Problem Configuration

```
[ bound = 500, limit = 5000, input_order ]

% Assume adversary avoids detection at
% our main measurement event.
% This is a measurement of ker.
l(V) = msp(p4, M, p4, vc, X)
 => corrupt_at(p4, vc, V).

% dependencies
depends(p4, C, p4, sys) => C = ker.
depends(p4, C, p4, vc) => C = ker.
% rtm has no dependencies
depends(p1, C, p1, rtm) => false.


% Assume no recent corruptions
%prec(V, V1) & l(V1) = cor(P,C) & ms_evt(V)
%  => false.

% Assume no deep corruptions
l(V) = cor(p1, M) => false.

m4_include(`a2-ker-vc-par.gli')m4_dnl

m4_include(`a2-ker-vc-par_dist.gli')m4_dnl

m4_include(`thy.gli')m4_dnl
```

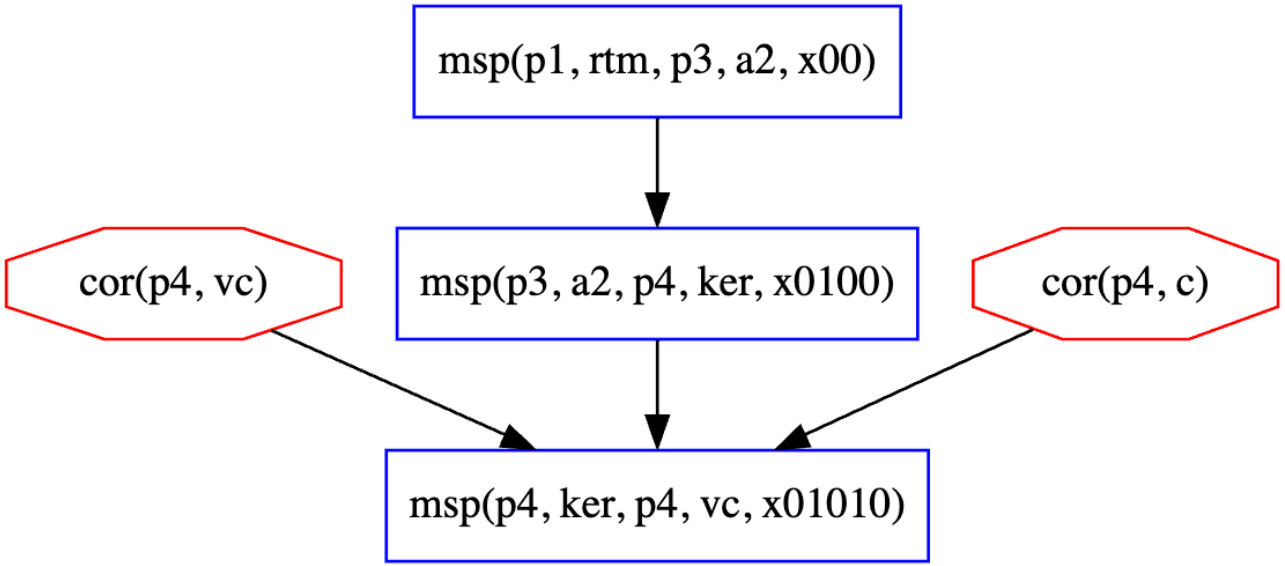
Model 1



a2-ker-vc-seq

- *target: @p1 [rtm p3 a2 +<+ @p3 [a2 p4 ker +<+ @p4 (ker p4 vc)]]

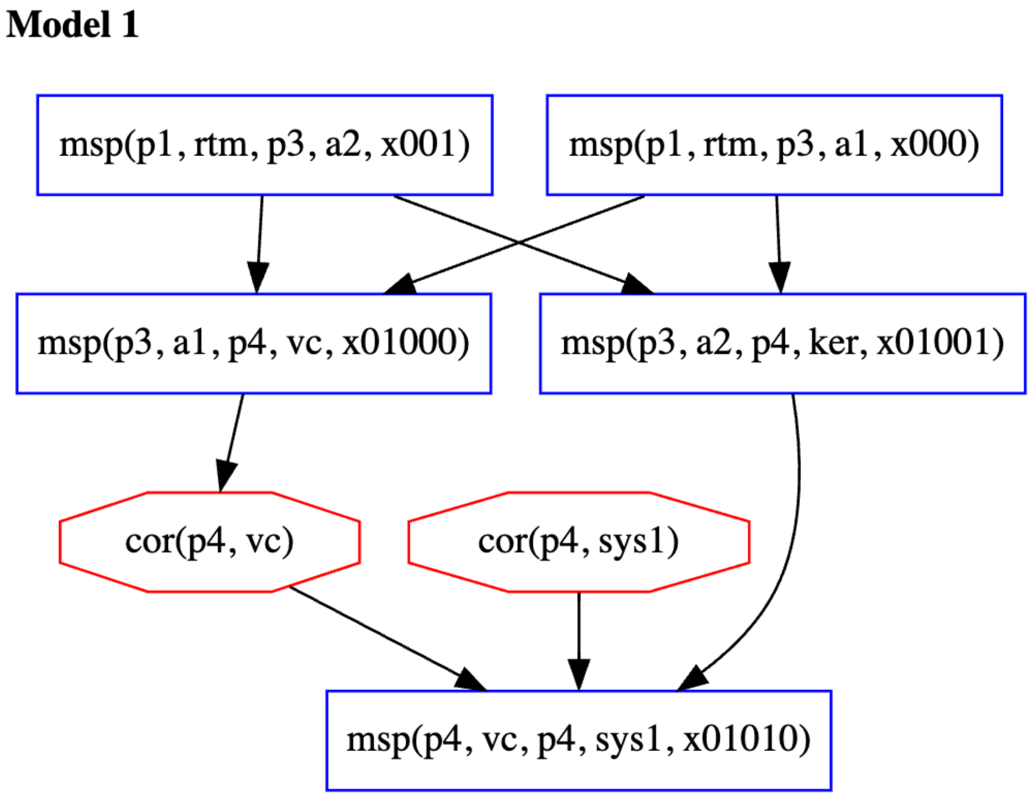
Model 1



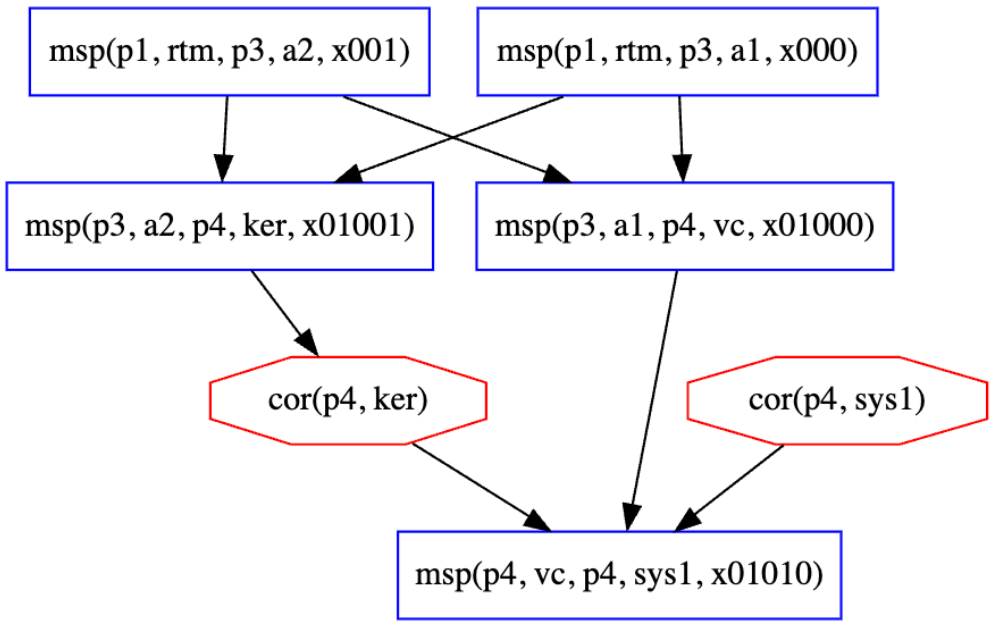
a1-a2-vc-ker-sys

- *target: @p1 [(rtm p3 a1 +~+ rtm p3 a2) +<+ @p3 [(a1 p4 vc +~+ a2 p4 ker) +<+ @p4 (vc p4 sys1)]]

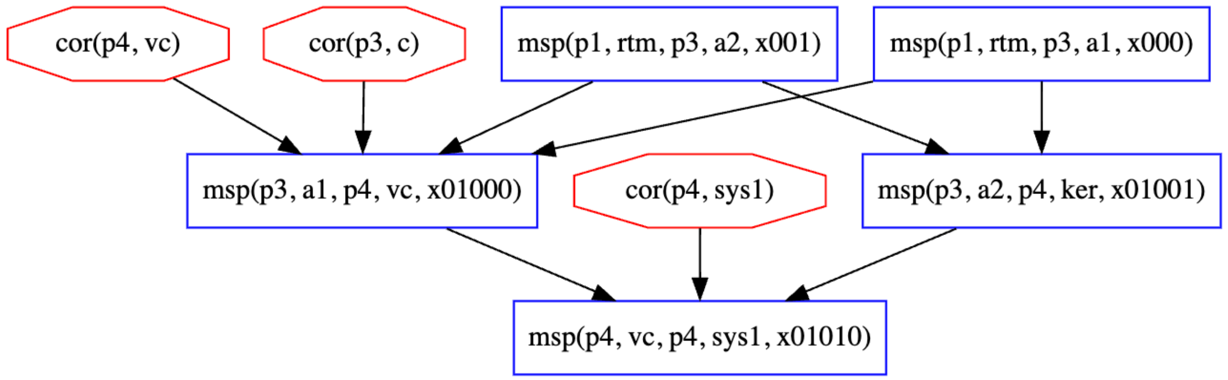
```
% dependencies
depends(p4, C, p4, sys1) => C = ker.
depends(p4, C, p4, vc) => C = ker.
% depends(p1, C, p4, ker) => C = rtm.
% depends(p1, C, p3, a1) => C = rtm.

% depends(P1, C, p3, a2) => P1 = p1 & C = rtm.
% rtm has no dependencies
depends(p1, C, p1, rtm) => false.


% Assume no recent corruptions
%prec(V, V1) & l(V1) = cor(P,C) & ms_evt(V)
%  => false.

% Assume no deep corruptions
l(V) = cor(p1, M) => false.
```



Model 2



Model 4



Same protocol… change theory

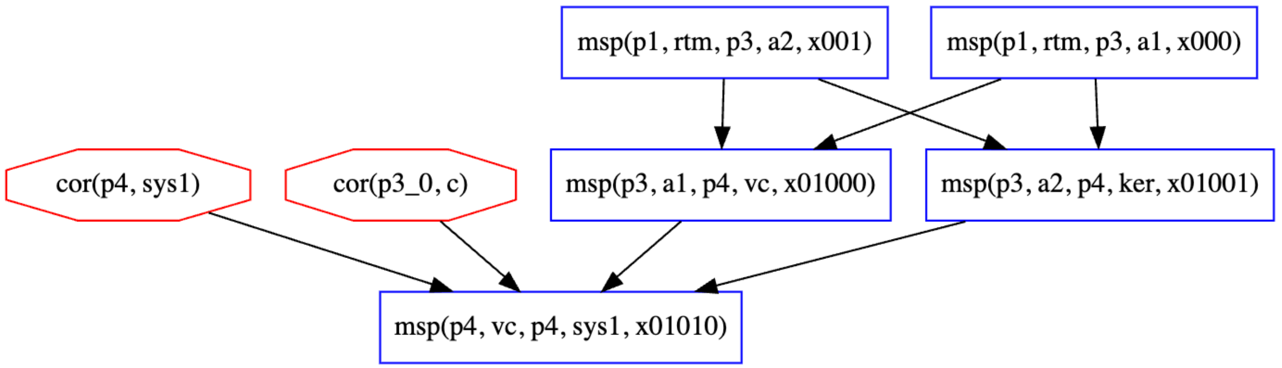
Problem Configuration

```
[ bound = 500, limit = 5000, input_order ]

% Assume adversary avoids detection at
% our main measurement event.
% This is a measurement of ker.
l(V) = msp(p4, M, p4, sys1, X)
 => corrupt_at(p4, sys1, V).

% dependencies
depends(p4, C, p4, sys1) => C = ker.
depends(p4, C, p4, vc) => C = ker.
depends(P1, C, p4, ker) => P1 = p1 & C = rtm.
depends(P1, C, p3, a1) => P1 = p1 & C = rtm.
depends(P1, C, p3, a2) => P1 = p1 & C = rtm.
% rtm has no dependencies
depends(p1, C, p1, rtm) => false.


% Assume no recent corruptions
%prec(V, V1) & l(V1) = cor(P,C) & ms_evt(V)
%  => false.

% Assume no deep corruptions
l(V) = cor(p1, M) => false.

m4_include(`a1-a2-vc-ker-sys.gli')m4_dnl

m4_include(`a1-a2-vc-ker-sys_dist.gli')m4_dnl

m4_include(`thy.gli')m4_dnl
```

```
% Rule 1
%l(V) = msp(P2, M, P1, T, X) & corrupt_at(P1, T, V)
%  => corrupt_at(P2, M, V) | depends(P2, C, P2, M) & corrupt_at(P2, C, V).
%corruption events can be at different places
l(V) = msp(P2, M, P1, T, X) & corrupt_at(P1, T, V)
  => corrupt_at(P2, M, V) | depends(P3, C, P2, M) & corrupt_at(P3, C, V).
```

Models

Model 1

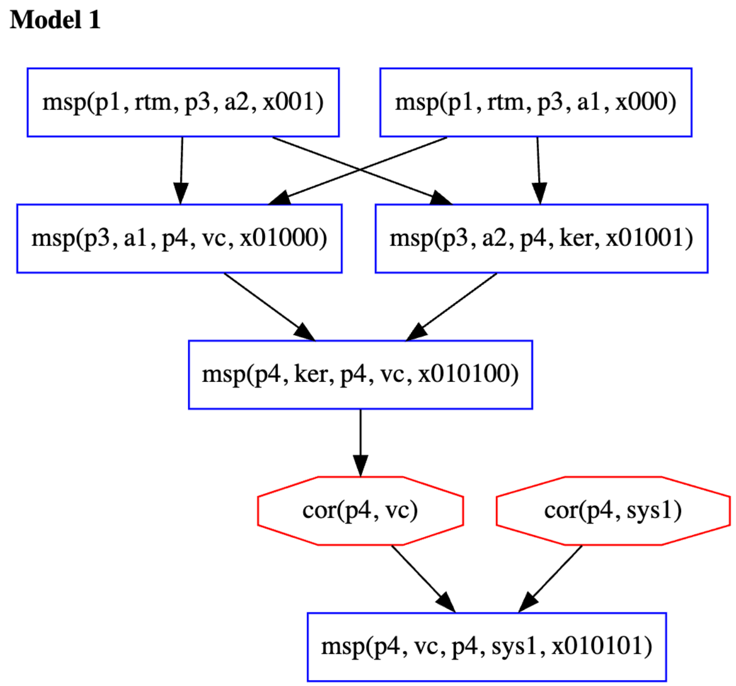


Back to original theory… change measurement

- *target: @p1 [(rtm p3 a1 +~+ rtm p3 a2) +<+

  @p3 [(a1 p4 vc +~+ a2 p4 ker) +<+

  @p4 [(ker p4 vc) +<+ (vc p4 sys1)]]]



Ways to manipulate CHASE outcome

Things I tried but didn’t have much impact…

1. Change measurement combination (parallel to sequence)
2. Add measurements
   - Not sure how to distinguish when something is a “useful” measurement
3. Changing dependencies

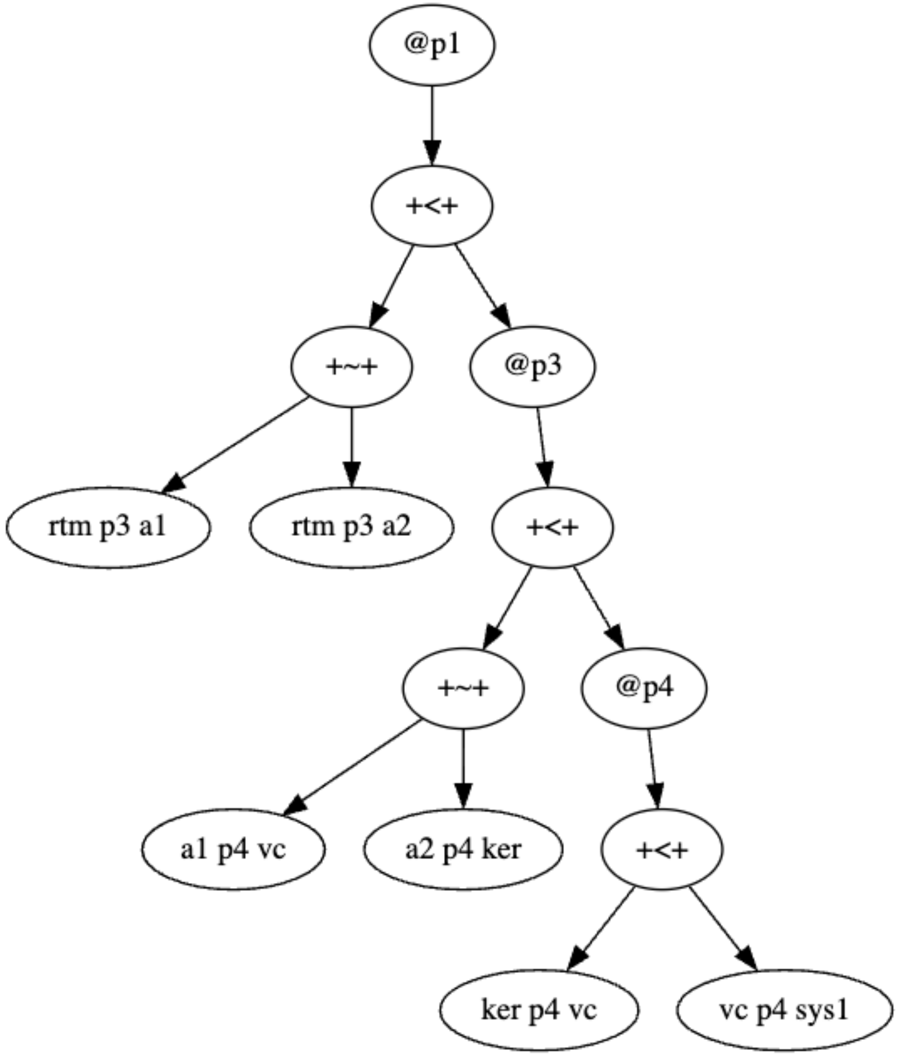
Slides (5/5)

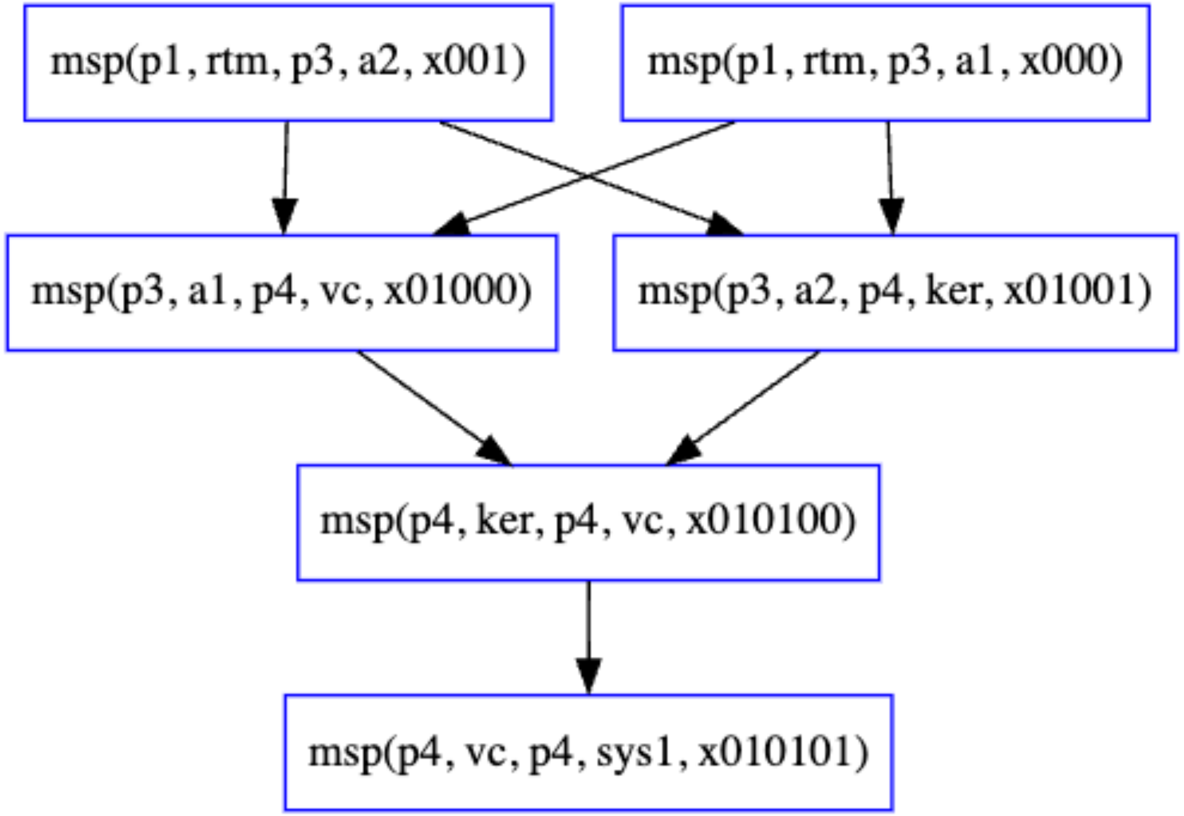
Assumptions

- Always assume deep theorem (remove recent theorem)
  - l(V) = cor(p1, M) => false.
- Assumptions about system dependencies
  - TPM is the root of trust… has no dependencies
  - Virus checker depends on kernel (p4,ker)
  - System depends on kernel (p4,ker)
  - Kernel depends on the hardware (p1,rtm)
  - A1 depends on the hardware (p1,rtm)
  - A2 depends on the hardware (p1,rtm)

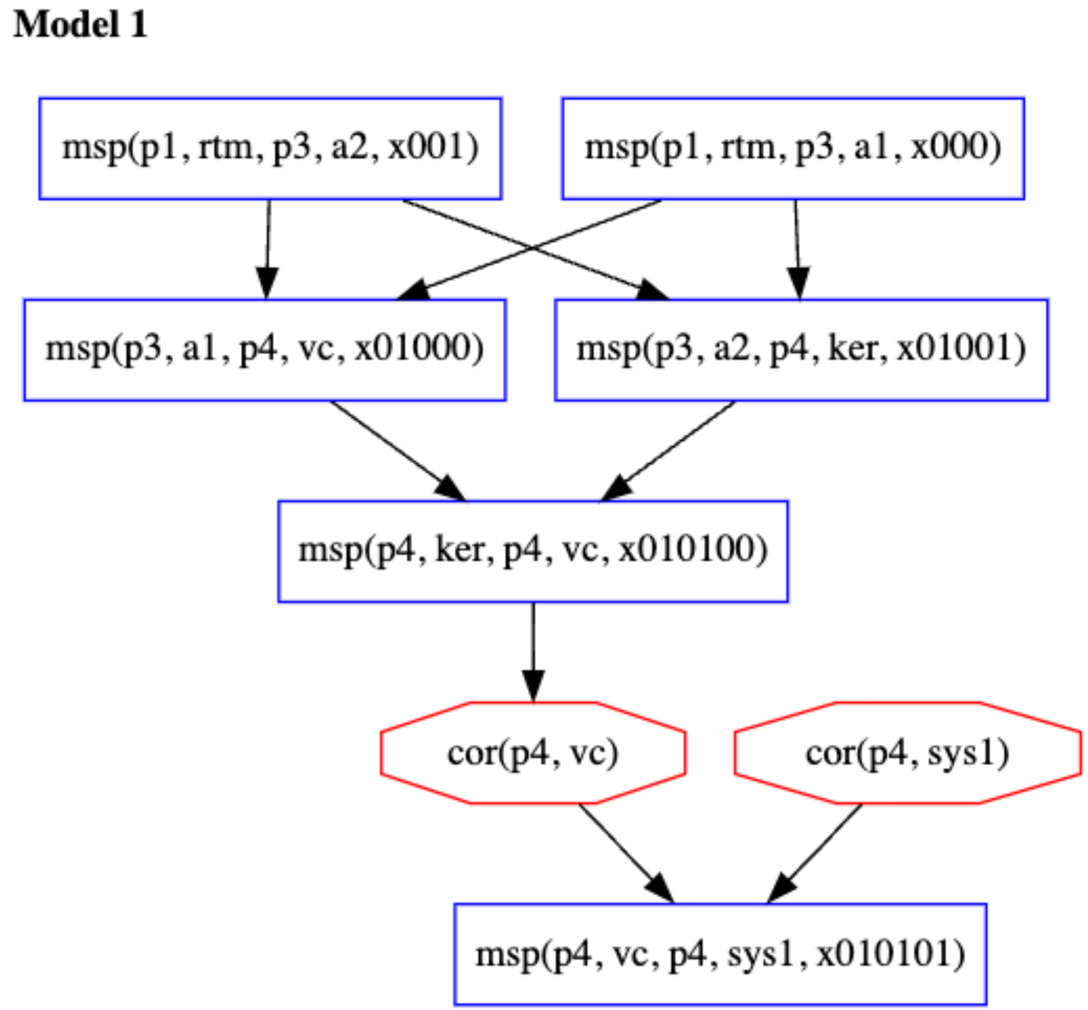
```
% dependencies
depends(p4, C, p4, sys1) => C = ker.
depends(p4, C, p4, vc) => C = ker.
depends(p1, C, p4, ker) => C = rtm.
depends(p1, C, p3, a1) => C = rtm.
depends(p1, C, p3, a2) => C = rtm.
% rtm has no dependencies
depends(p1, C, p1, rtm) => false.
```

Abstract Syntax Tree

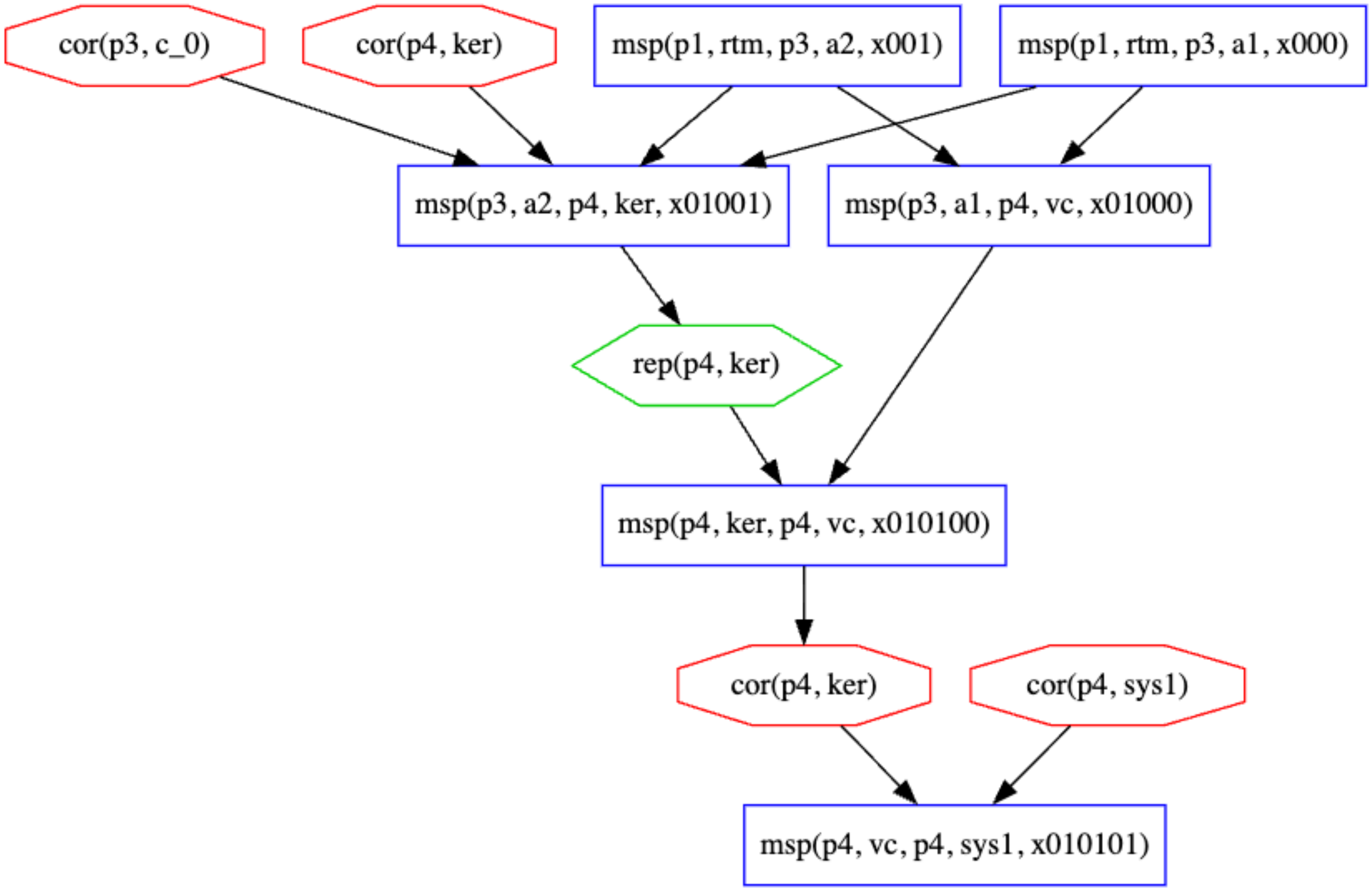




Assuming recent measurements may be corrupted there are 21 models…



Model 21


cor(p3, c_0)
cor(p4, ker)
msp(p1, rtm, p3, a2, x001)
msp(p1, rtm, p3, a1, x000)
msp(p3, a2, p4, ker, x01001)
msp(p3, a1, p4, vc, x01000)
rep(p4, ker)
msp(p4, ker, p4, vc, x010100)
cor(p4, ker)
cor(p4, sys1)
msp(p4, vc, p4, sys1, x010101)

Order	Event	Cost	Present In	Details
low	cor(p4,sys1)	c1	all models	Always before the last measurement event
	cor(p4,c)	c2	4	happens before ms(ker,vc)
	cor(p4,vc)	c3	1 4 5 8 9 10 11 12 13 14 15 16 17 18(2) 19(2)	occurs after some attestation start event (between measurements) or before a measurement, sometimes happens twice (once and then after a repair)
	cor(p4, ker)	c4	2 3 5 6 7 10 11 12 13 14 15 16 17 20(2) 21(2)	occurs various places.. Before/after ms(a2,ker), before ms(vc,sys1)
	cor(p4,c_1)	c5	8	before ms(ker,vc)
	cor(p3,a1)	c6	8 10 14 15 18	before ms(a1,vc), always after the attestation begins… maybe this is most difficult because you have to consider time window for adversary
	cor(p3,c_3)	c7	9 11 16 17 19	before ms(a1,vc), no attestation start event… could be easiest for an adversary
	cor(p4,c_2)	c8	9	before ms(ker,vc), no attestation start event… could be easiest for adversary
	cor(p3,a2)	c9	6 12 14 16 20	between ms(rtm, a2) – ms(a2, ker)… close to root of trust. Difficult for an adversary
	cor(p3,c_4)	c10	13	before ms(a2,ker), no attestation start event… could be easiest for adversary
	cor(p3,c_5)	c11	15	before ms(a2,ker) no attestation start event… could be easiest for adversary
	cor(p3,c_6)	c12	17	before ms(a2,ker) , no attestation start event… could be easiest for adversary
	rep(p4,vc)	c13	18 19	between ms(a1,vc) – ms(ker,vc),
	rep(p4,ker)	c14	20 21	between ms(a2,ker) -- ms(ker, vc)
	cor(p3,c_0)	c15	7 21	before ms(a2,ker)

Thoughts/Takeaways

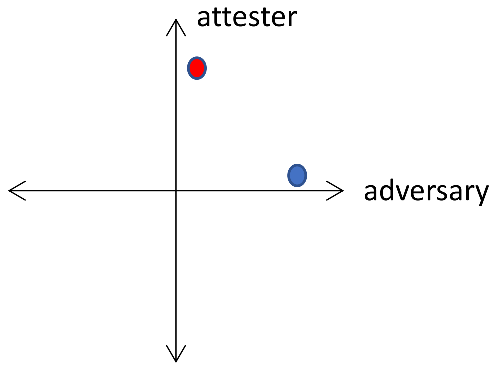
- Write some script to assign cost

Slides (3) 4.21.23

Goals of cost analysis

Ultimate goal: guide selection of a protocol

- How:
  - Systematic variation of assumption
  - Assign abstract cost to each component that's corrupted
  - Define function to create order between cost and value
- Consider:
  - Cost to adversary
  - Cost to attester



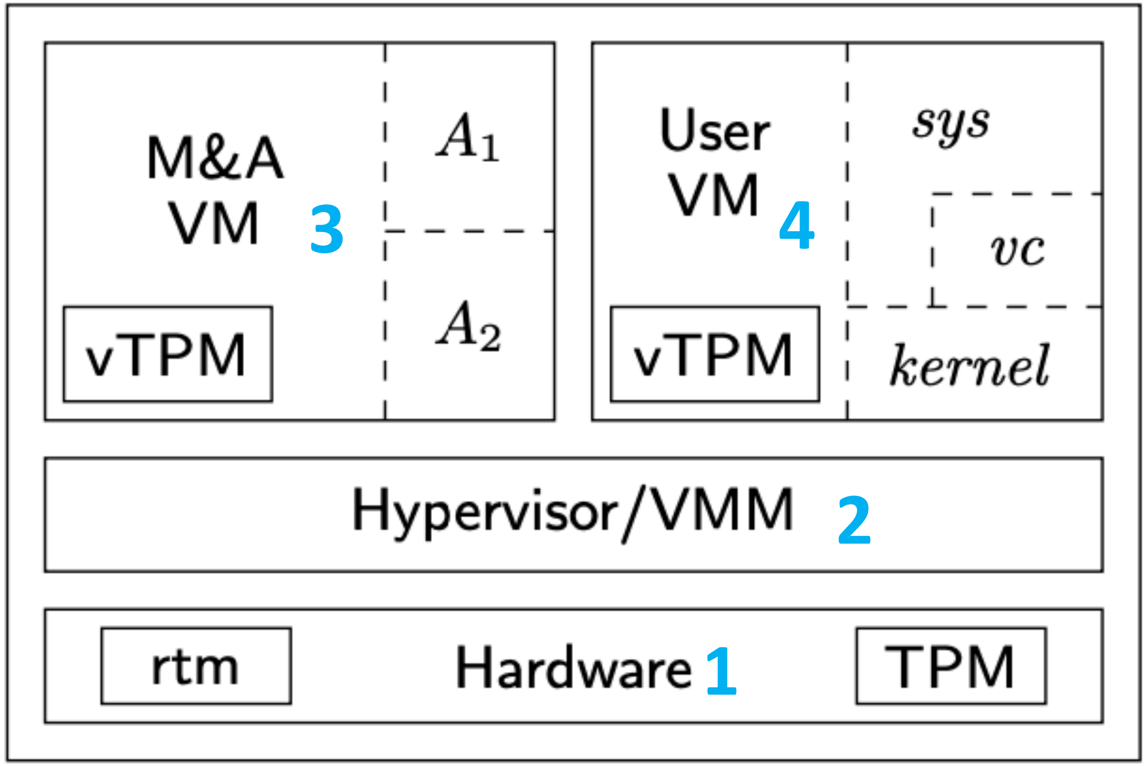
What I did for today (5/5)

- Ran one complex protocol which considered all measurement operations in “Confining” system

- What I did not do…
  - Consider different ordering of the protocol
  - I think we have enough material to discuss with just one protocol

Say we have the architecture from “Confining the Adversary” Paper

- ms(rtm, A1)
- ms(rtm, A2)
- ms(A1, vc)
- ms(A2, ker)
- msker (vc, sys)



I’m not really sure how to abstractly model msker… so I just assumed it was a measurement event and added dependency between the vc and kernel

Assumptions

- Always assume deep theorem (remove recent theorem)
- Assumptions about system dependencies
  - TPM is the root of trust… has no dependencies
  - Virus checker depends on kernel (p4,ker)
  - System depends on kernel (p4,ker)
  - Kernel depends on the hardware (p1,rtm)
  - A1 depends on the hardware (p1,rtm)
  - A2 depends on the hardware (p1,rtm)

```
% dependencies
depends(p4, C, p4, sys1) => C = ker.
depends(p4, C, p4, vc) => C = ker.
depends(p1, C, p4, ker) => C = rtm.
depends(p1, C, p3, a1) => C = rtm.
depends(p1, C, p3, a2) => C = rtm.
% rtm has no dependencies
depends(p1, C, p1, rtm) => false.
```

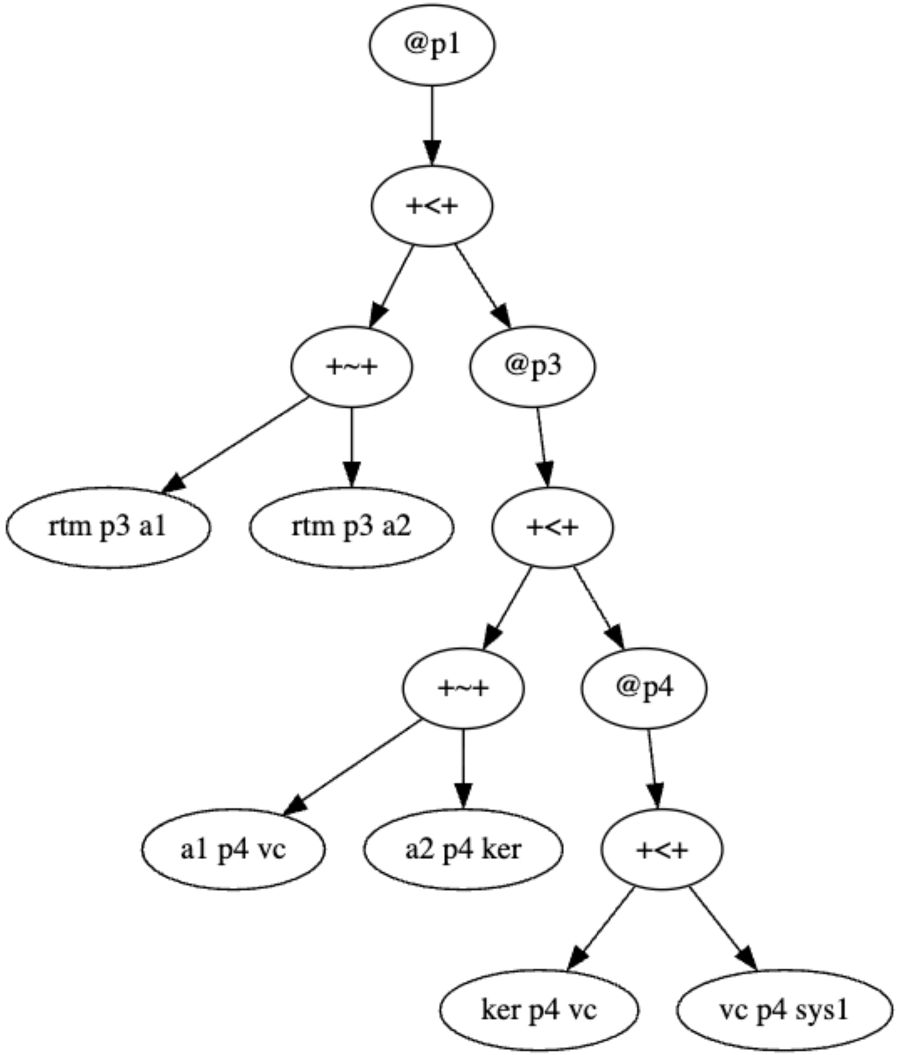
Principles

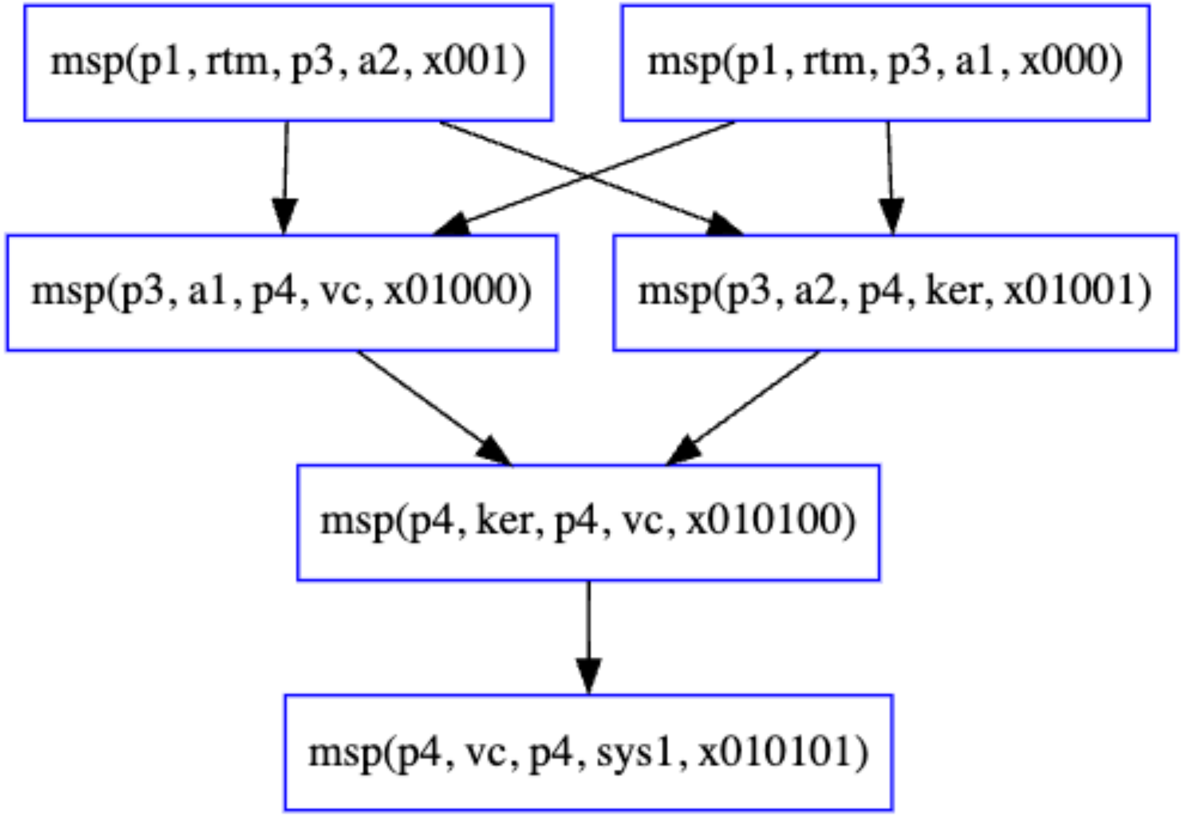
- Increase cost after start event
- Any corruption of a deeper component is higher cost
- Add weight to cost event
  - Have some base cost to the corruption event
  - Add to cost more if its in a protected place

Protocols

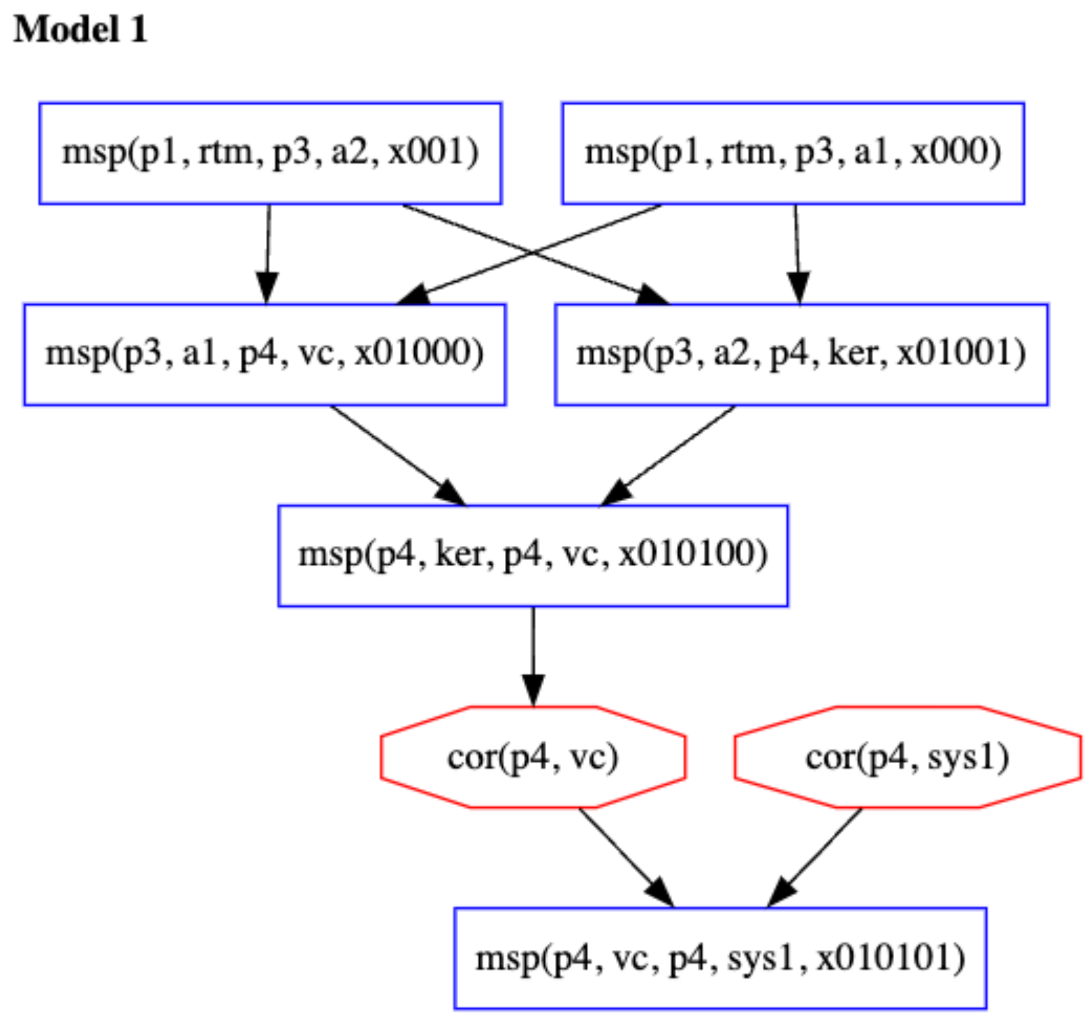
Protocol Name	Protocol
sys	*target: @p4 [vc p4 sys1]
vc-sys-seq	*target: @p3 [a p4 vc] +<+ @p4 [vc p4 sys]
vc-sys-par	*target: @p3 [a p4 vc] +~+ @p4 [vc p4 sys]
a-vc-sys-seq	*target: @p1 [rtm p3 a] +<+ @p3 [a p4 vc] +<+ @p4 [vc p4 sys]
a-vc-sys-par	*target: @p1 [rtm p3 a +~+ @p3 [a p4 vc +~+ @p4 [vc p4 sys]]]
a1-a2-vc-ker-sys	*target: @p1 (rtm p3 a1 +~+ rtm p3 a2) +<+ @p3 (a1 p4 vc +~+ a2 p4 ker) +<+ @p4 ((ker p4 vc) +<+ (vc p4 sys1))

Abstract Syntax Tree

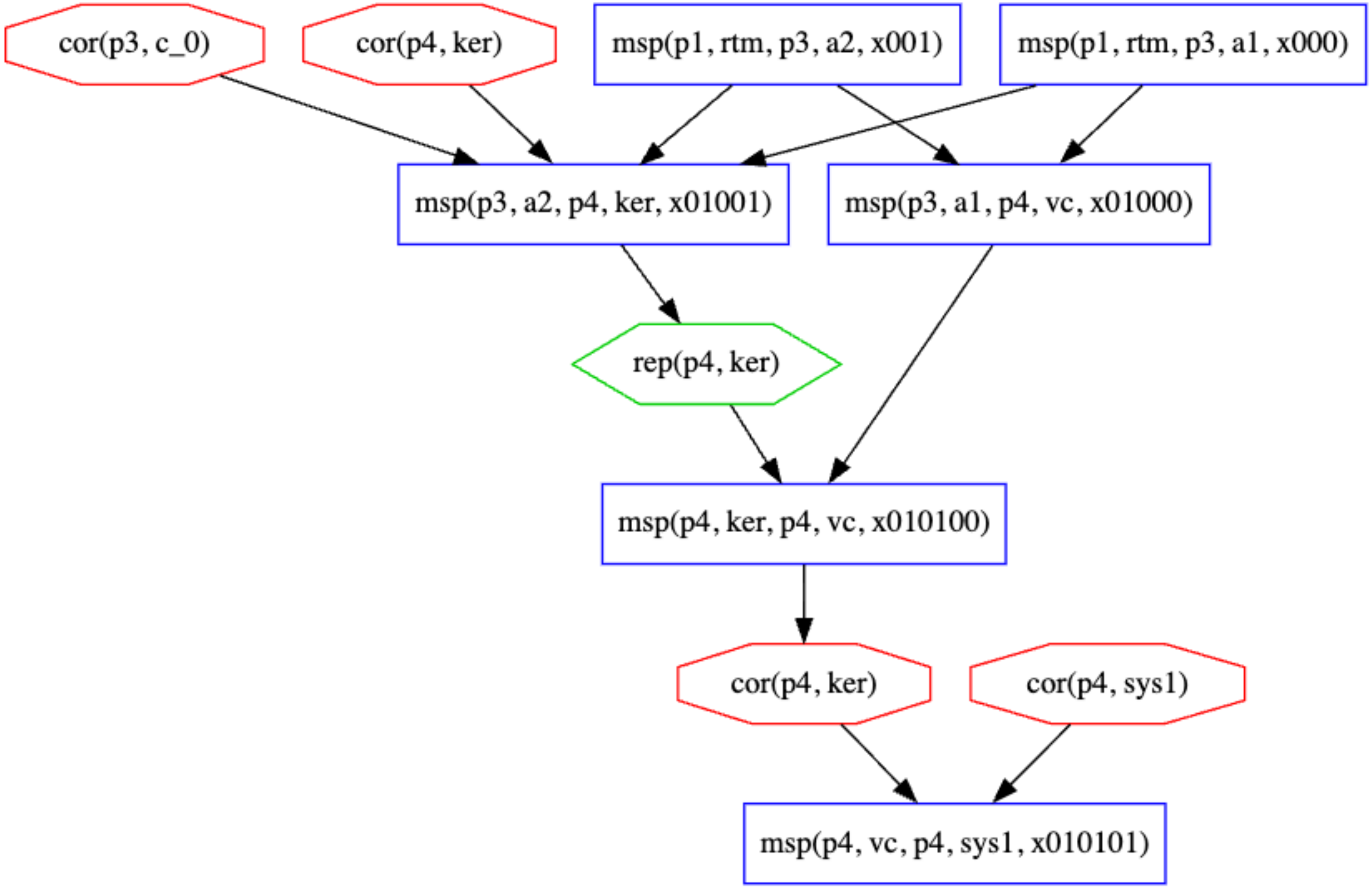




Assuming recent measurements may be corrupted there are 21 models…



Model 21


cor(p3, c_0)
cor(p4, ker)
msp(p1, rtm, p3, a2, x001)
msp(p1, rtm, p3, a1, x000)
msp(p3, a2, p4, ker, x01001)
msp(p3, a1, p4, vc, x01000)
rep(p4, ker)
msp(p4, ker, p4, vc, x010100)
cor(p4, ker)
cor(p4, sys1)
msp(p4, vc, p4, sys1, x010101)

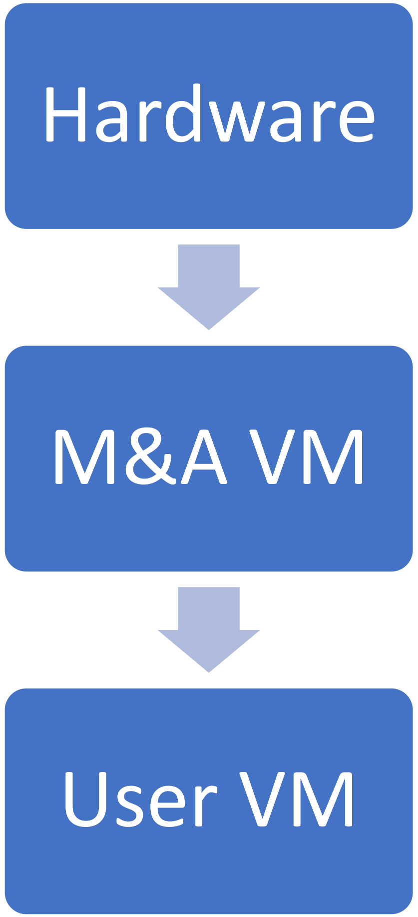
Order	Event	Cost	Present In	Details
low	cor(p4,sys1)	c1	all models	Always before the last measurement event
	cor(p4,c)	c2	4	happens before ms(ker,vc)
	cor(p4,vc)	c3	1 4 5 8 9 10 11 12 13 14 15 16 17 18(2) 19(2)	occurs after some attestation start event (between measurements) or before a measurement, sometimes happens twice (once and then after a repair)
	cor(p4, ker)	c4	2 3 5 6 7 10 11 12 13 14 15 16 17 20(2) 21(2)	occurs various places.. Before/after ms(a2,ker), before ms(vc,sys1)
	cor(p4,c_1)	c5	8	before ms(ker,vc)
	cor(p3,a1)	c6	8 10 14 15 18	before ms(a1,vc), always after the attestation begins… maybe this is most difficult because you have to consider time window for adversary
	cor(p3,c_3)	c7	9 11 16 17 19	before ms(a1,vc), no attestation start event… could be easiest for an adversary
	cor(p4,c_2)	c8	9	before ms(ker,vc), no attestation start event… could be easiest for adversary
	cor(p3,a2)	c9	6 12 14 16 20	between ms(rtm, a2) – ms(a2, ker)… close to root of trust. Difficult for an adversary
	cor(p3,c_4)	c10	13	before ms(a2,ker), no attestation start event… could be easiest for adversary
	cor(p3,c_5)	c11	15	before ms(a2,ker) no attestation start event… could be easiest for adversary
	cor(p3,c_6)	c12	17	before ms(a2,ker) , no attestation start event… could be easiest for adversary
	rep(p4,vc)	c13	18 19	between ms(a1,vc) – ms(ker,vc),
	rep(p4,ker)	c14	20 21	between ms(a2,ker) -- ms(ker, vc)
	cor(p3,c_0)	c15	7 21	before ms(a2,ker)

Considering Cost to an Adversary

- Hardware = highest cost to adversary
- M&A = middle cost
- User VM = lowest cost to an adversary

Considering Cost to an Attester

- Hardware = worst case for an adversary
- M&A = ??
- User VM =??



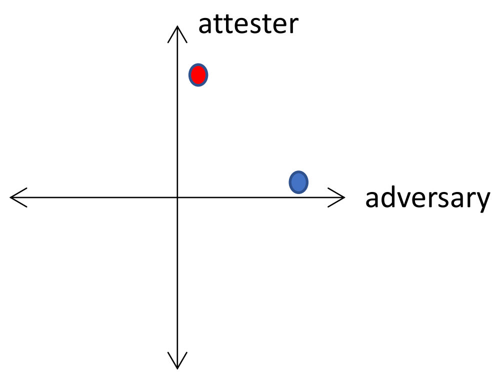
Cost

Cost	Reasoning
high	• corruption events that occur between measurement events are difficult… Thus, high cost • corruption events closer to root of trust are difficult. Thus, high cost. • corruption then repair then corruption requires a lot of work from adversary. This is a high cost.
medium	• corruption event at M&A domain is medium as it is in the middle of the architecture
low	• corruption before last measurement is probably the easiest thing for an adversary therefore the lowest cost.

Slides 4.14.23

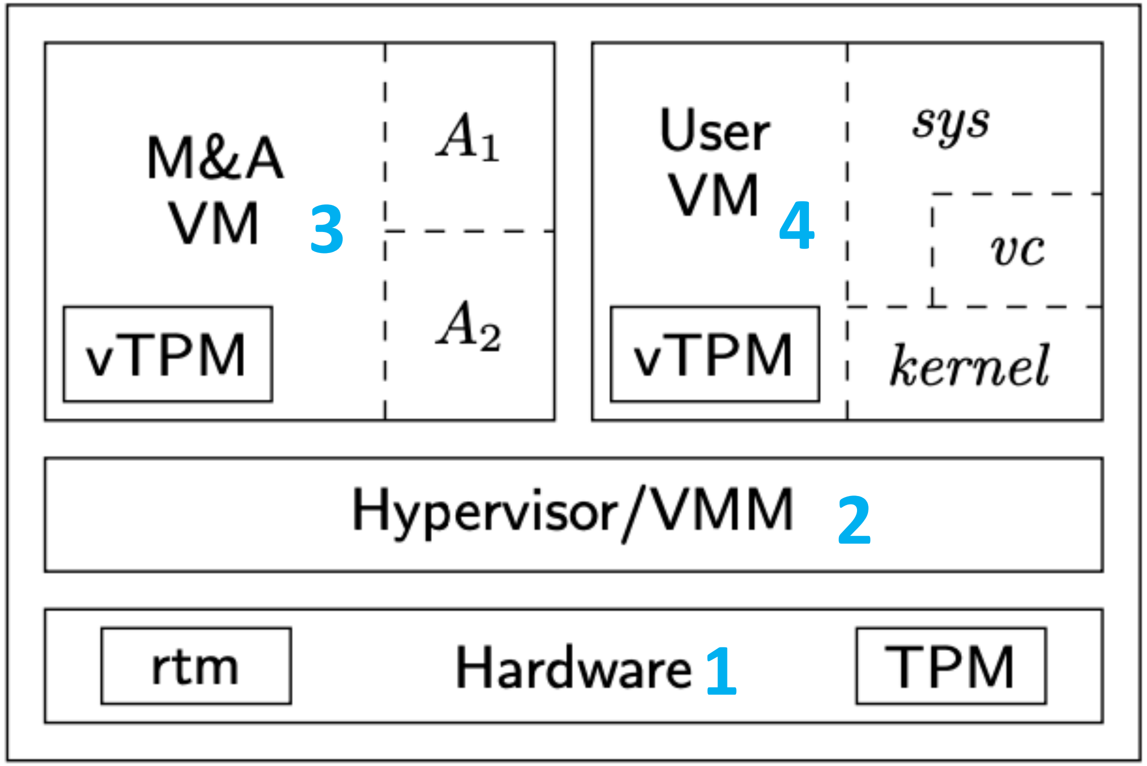
Goals of cost analysis

- Ultimate goal: guide selection of a protocol
- How:
  - Systematic variation of assumption
  - Assign abstract cost to each component that's corrupted
- Consider:
  - Cost to adversary
  - Cost to attester



Say we have the architecture from "Confining the Adversary" Paper

- ms(rtm, A1)
- ~~ms(rtm, A2)~~
- ms(A1, vc)
- ~~ms(A2, ker)~~
- msker (vc, sys)



Control Variables

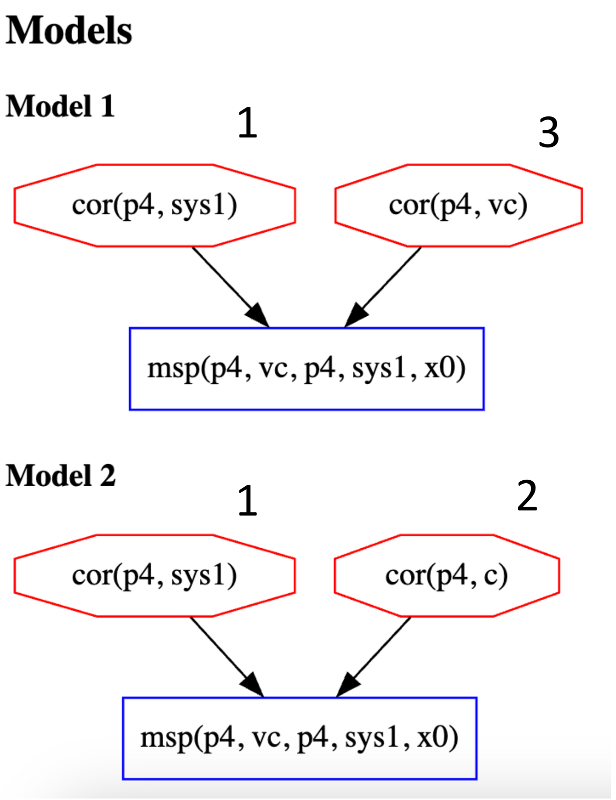
```
% Assume sys depends on kernel
% if sys1 or vc depend on anything, that thing is the root of trust
depends(p1, C, p4, sys) => C = rtm.
depends(p1, C, p4, vc) => C = rtm.
depends(p1, C, p3, a) => C = rtm.
% rtm has no dependencies
depends(p1, C, p1, rtm) => false.

% Assume no deep corruptions
l(V) = cor(p1, M) => false.
```

- Assumptions
  - Always assume recent/deep
  - Make no assumptions about system dependencies except…
    - TPM is the root of trust… has no dependencies
    - Virus checker and system depend on the hardware (p1,rtm)
    - A1 depends on the hardware (p1,rtm)

Side note: I changed all theory files to the original… allows for corruption only at the same place

- If I made it allow for corruption at different places… CHASE seemed to introduce corruption events with odd labels sys

First protocol…. Just measure *sys* using *vc*



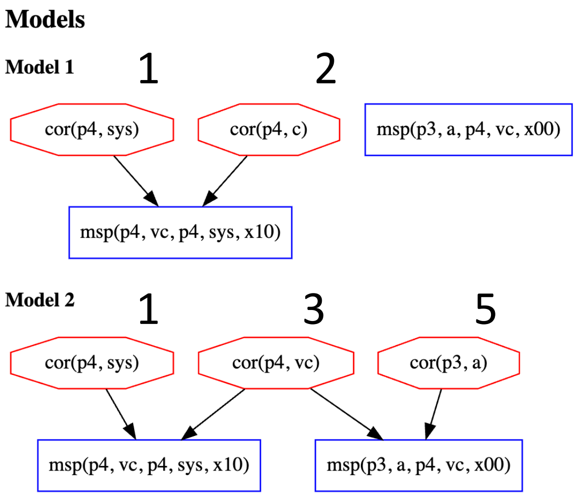
Event	Cost
cor(p4,sys1)	c1
cor(p4,vc)	c3
cor(p4,c)	c2
MODEL 1 COST	c1 + c3
MODEL 2 COST	c1 + c2

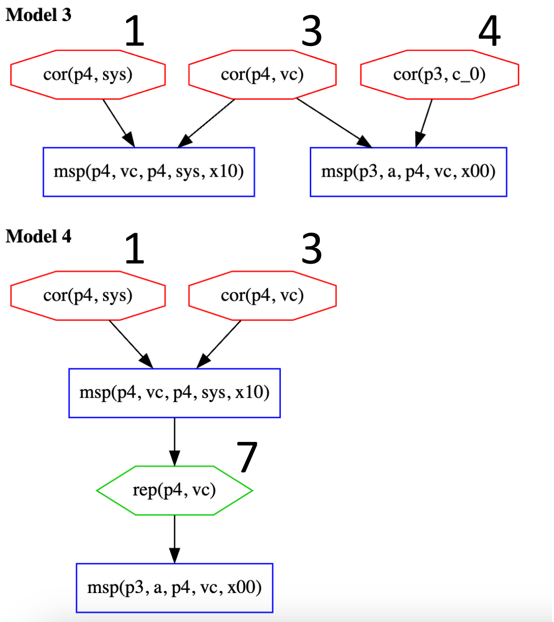
If you add in dependencies about a then labels to corruption events change

Measure *vc* and *sys* in parallel

- Protocol
  - *target: @p3 [a p4 vc]
    +~+ @p4 [vc p4 sys]

```
% Assume dependencies
% if sys1 or vc depend on anything, that thing is the root of trust
depends(p1, C, p4, sys) => C = rtm.
depends(p1, C, p4, vc) => C = rtm.
depends(p1, C, p3, a) => C = rtm.
% rtm has no dependencies
depends(p1, C, p1, rtm) => false.

% Assume no recent corruptions
prec(V, V1) & l(V1) = cor(P,C) & ms_evt(V)
=> false.

% Assume no deep corruptions
l(V) = cor(p1, M) => false.
```

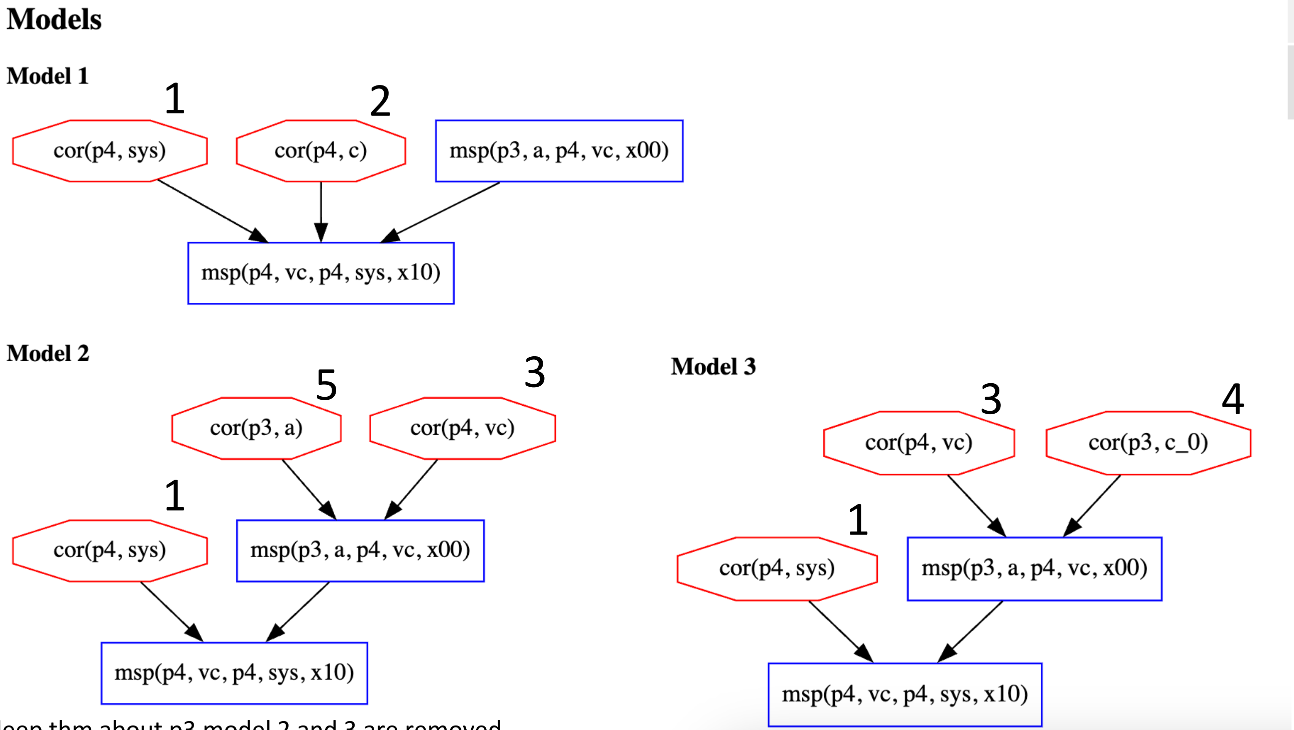




Model	Total cost
1	c1 + c2
2	c1 + c3 + c5
3	c1 + c3 + c4
4	c1 + c3 + c7

Measure *vc* and *sys* in sequence

- Protocol
  - *target: @p3 [a p4 vc]
    +<+ @p4 [vc p4 sys]



If you add deep thm about p3 model 2 and 3 are removed

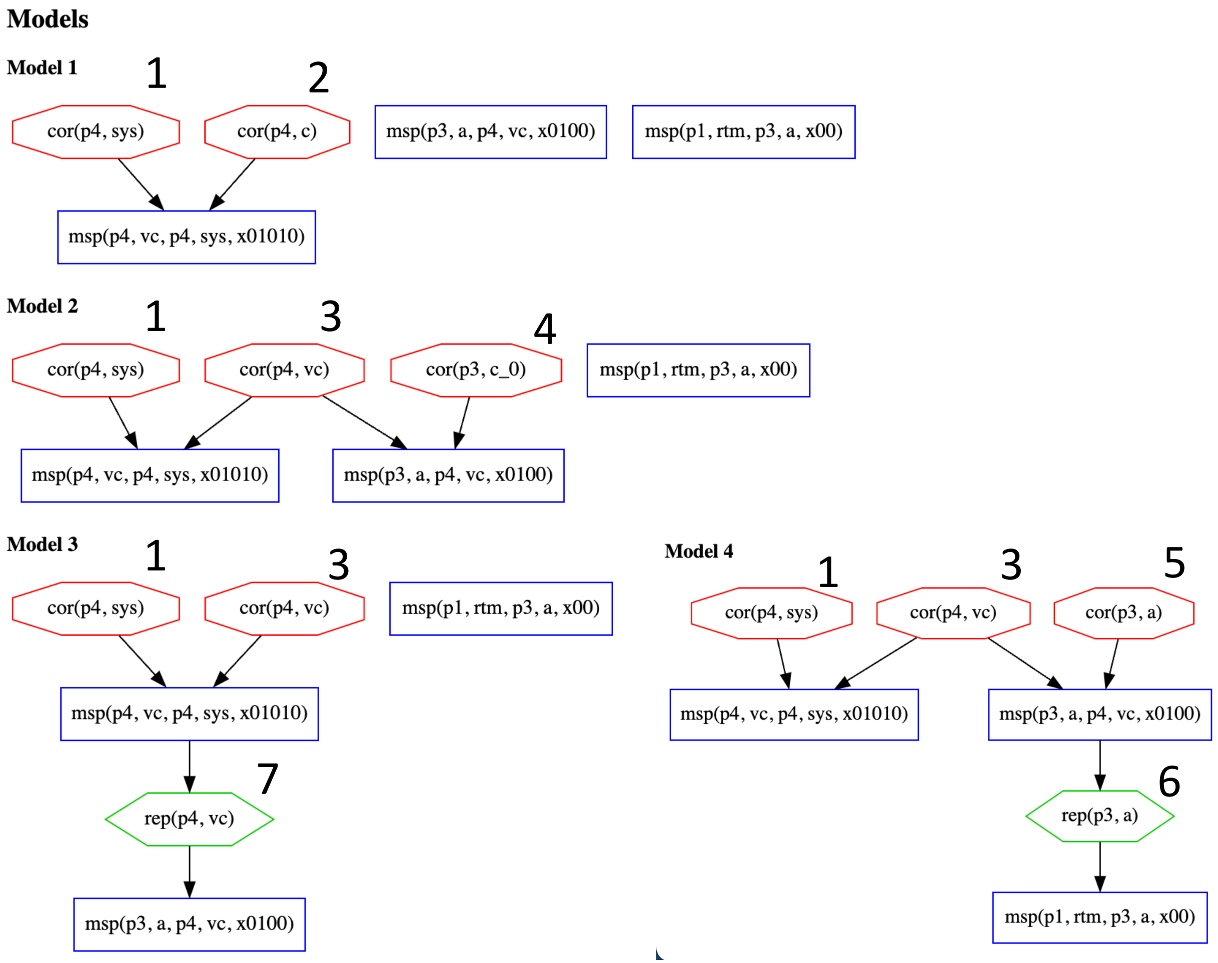
Model	Total cost
1	c1 + c2
2	c1 + c3 + c4
3	c1 + c5 + c3

a-vc-sys-par

Measure *a* then *vc* then *sys* in parallel

- Protocol
  - *target: @p1 [rtm p3 *a*
    +~+ @p3 [a p4 *vc*
    +~+ @p4 [vc p4 *sys*]]]]

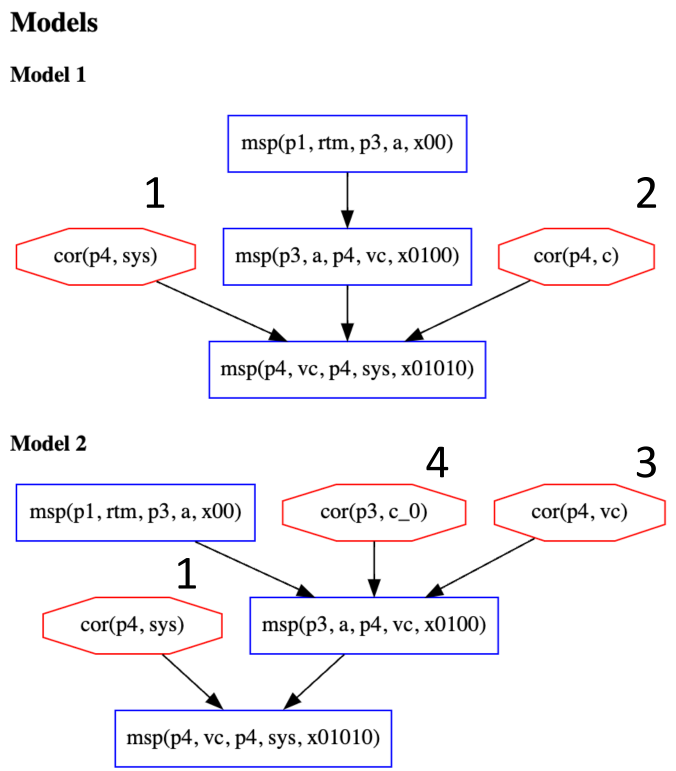
Model	Total cost
1	c1 + c2
2	c1 + c2 + c4
3	c1 + c3 + c7
4	c1 + c3 + c5 + c6



Measure *a* then *vc* then *sys* in sequence

- Protocol
  - *target: @p1 [rtm p3 *a*
    +<+ @p3 [a p4 *vc*
    +<+ @p4 [vc p4 *sys*]]]]

Model	Total cost
1	c1 + c2
2	c1 + c3 + c4



All together

label	protocol	total cost
sys	*target: @p4 [vc p4 sys1]	(c1 + c3) OR (c1 + c2)
vc-sys-par	*target: @p3 [a p4 vc] +~+ @p4 [vc p4 sys]	(c1 + c2) OR (c1 + c3 + c5) OR (c1 + c3 + c4) OR (c1 + c3 + c7)
vc-sys-seq	*target: @p3 [a p4 vc] +<+ @p4 [vc p4 sys]	(c1 + c2) OR (c1 + c3 + c4) OR (c1 + c5 + c3)
a-vc-sys-par	*target: @p1 [rtm p3 *a* +~+ @p3 [a p4 *vc* +~+ @p4 [vc p4 *sys*]]]]	(c1 + c2) OR (c1 + c2 + c4) OR (c1 + c3 + c7) OR (c1 + c3 + c5 + c6)
a-vc-sys-seq	*target: @p1 [rtm p3 *a* +<+ @p3 [a p4 *vc* +<+ @p4 [vc p4 *sys*]]]]	(c1 + c2) OR (c1 + c3 + c4)

Event with label and cost

Event	Label	Cost	Present In
cor(p4,sys)	1	c1	sys(1,2),vc-sys-par(1,2,3,4), vc-sys-seq(1,2,3), a-vc-sys-par(1,2,3,4), a-vc-sys-seq(1,2)
cor(p4,c)	2	c2	sys(2), vc-sys-par(2), vc-sys-seq(1), a-vc-sys-par(1,2), a-vc-sys-seq(1)
cor(p4,vc)	3	c3	sys(1), vc-sys-par(2,3,4), vc-sys-seq(2,3), a-vc-sys-par(3,4), a-vc-sys-seq(2)
cor(p3, c_0)	4	c4	vc-sys-par(3), vc-sys-seq(2), a-vc-sys-par(2), a-vc-sys-seq(2)
cor(p3,a)	5	c5	vc-sys-par(2), vc-sys-seq(3), a-vc-sys-par(4)
rep(p3,a)	6	c6	a-vc-sys-par(4)
rep(p4,vc)	7	c7	vc-sys-par(4), a-vc-sys-par(3)

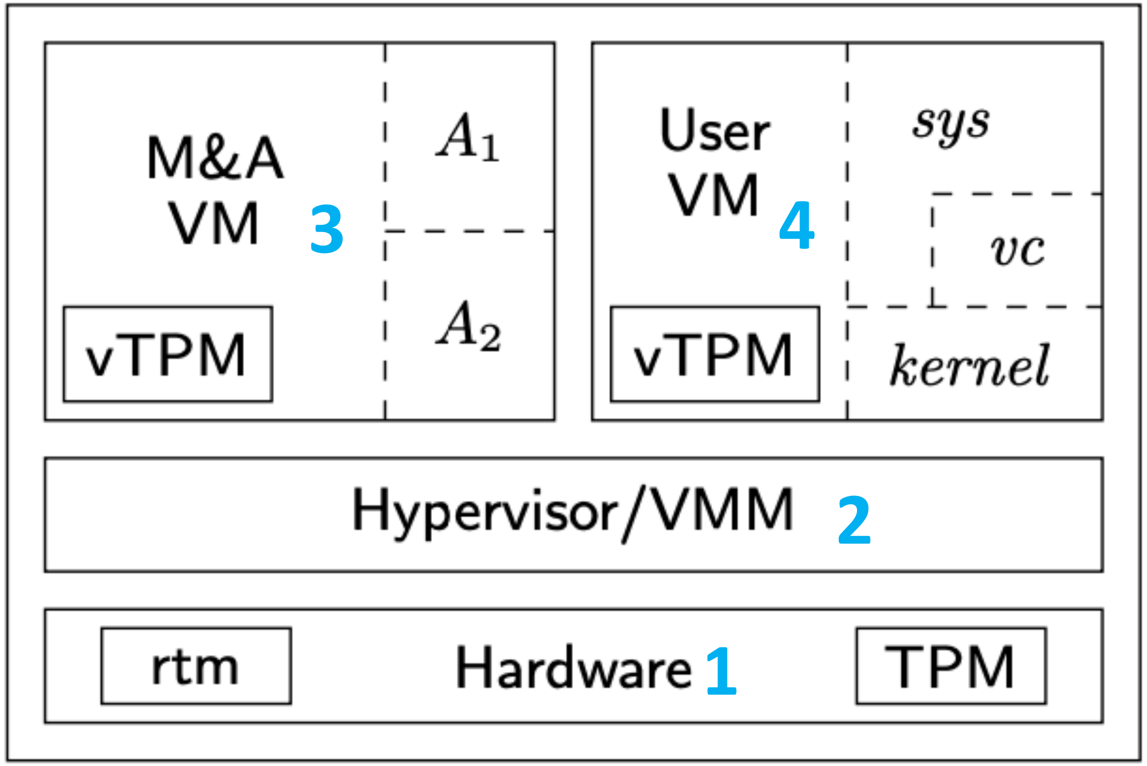
Slides 4.7.23

Goals of cost analysis

- Ultimate goal: guide selection of a protocol
- How:
  - systematic variation of assumption
  - assigning cost to each component that's corrupted
    - Assign low (or high?) values to difficult actions
    - Realize set of protocols, one with minimum (maximum) cost
    - Cost may reflect ordering

Say we have the architecture from "Confining the Adversary" Paper

- ms(rtm, A1)
- ms(rtm, A2)
- ms(A1, vc)
- ms(A2, ker)
- msker (vc, sys)



First protocol…. Just measure *sys* using *vc*

- Protocol:
  - @4 [vc 4 sys1]
- Cost?
  - Potentially 2?
    - 2 places where corruptions could occur

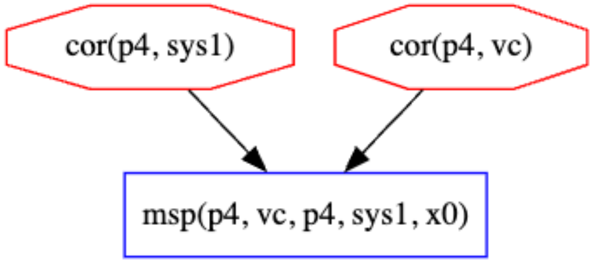
msp(p4, vc, p4, sys1, x0)

Problem Configuration

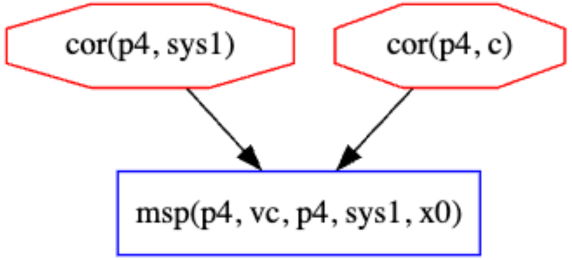
```
[ bound = 500, limit = 5000, input_order ]

% Assume adversary avoids detection at our main measurement
% event. This is a measurement of sys
l(V) = msp(p4, M, p4, sys1, X)
 => corrupt_at(p4, sys1, V).

% Assume no dependencies
depends(p4, C, p4, sys1) => false.

% No recent assumptions

% No deep assumptions

m4_include(`sys.gli')m4_dnl

m4_include(`sys_dist.gli')m4_dnl

m4_include(`thy.gli')m4_dnl
```

Models

Model 1



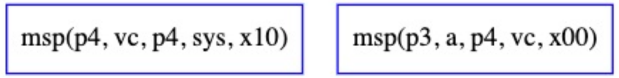
Model 2



Measure *vc* and *sys* in parallel

- Protocol
  - *target: @p3 [a p4 vc]
    +~+ @p4 [vc p4 sys]

- Cost?
  - Two corruption events and a repair event…
  - What should be the cost of a repair?

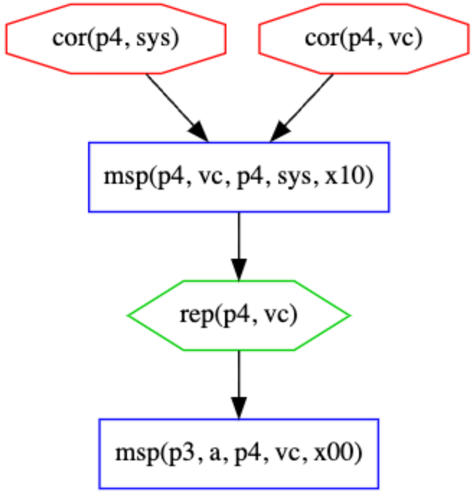


Problem Configuration

```
[ bound = 500, limit = 5000, input_order ]

% Assume adversary avoids detection at
% our main measurement event.
% This is a measurement of sys.
l(V) = msp(p4, M, p4, sys, X)
 => corrupt_at(p4, sys, V).

% Assume sys depends on kernel
depends(p4, C, p4, sys) => false.
depends(p4, C, p4, vc) => false.
depends(p3, C, p3, a) => false.

% Assume no recent corruptions
prec(V, V1) & l(V1) = cor(P,C) & ms_evt(V)
=> false.

% Assume no deep corruptions
l(V) = cor(p3, M) => false.

m4_include(`vc-sys.gli')m4_dnl

m4_include(`vc-sys_dist.gli')m4_dnl

m4_include(`thy.gli')m4_dnl
```

Models

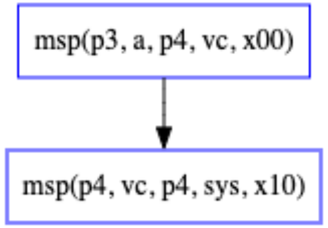
Model 1



Measure *vc* and *sys* in sequence

- Protocol
  - *target: @p3 [a p4 vc]
    +<+ @p4 [vc p4 sys]

- Analysis
  - No models if recent or deep assumption… this is expected

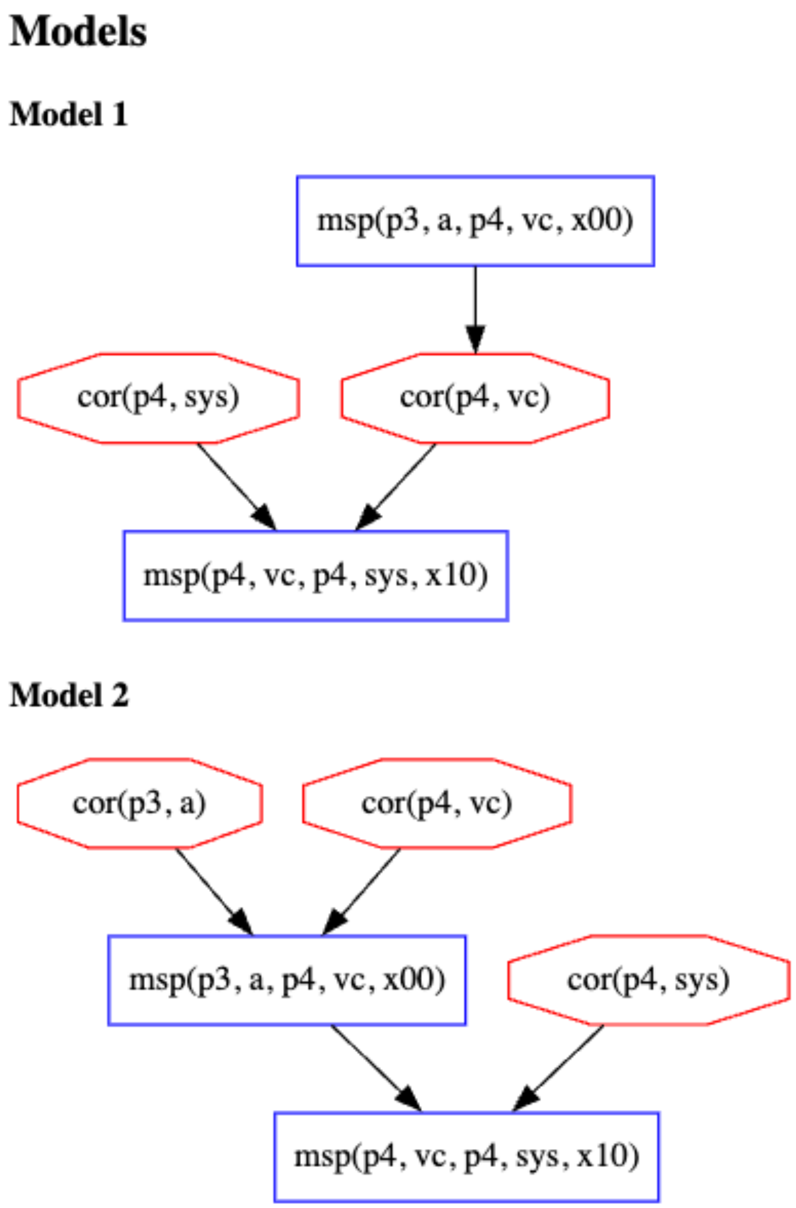


Problem Configuration

```
[ bound = 500, limit = 5000, input_order ]

% Assume adversary avoids detection at
% our main measurement event.
% This is a measurement of sys.
l(V) = msp(p4, M, p4, sys, X)
 => corrupt_at(p4, sys, V).

% Assume sys depends on kernel
% depends(p3, C, p3, a) => C = p1.
depends(p4, C, p4, sys) => false.
depends(p4, C, p4, vc) => false.
depends(p3, C, p3, a) => false.

% Assume no recent corruptions
prec(V, V1) & l(V1) = cor(P,C) & ms_evt(V)
  => false.

% Assume no deep corruptions
l(V) = cor(p3, M) => false.

m4_include(`vc-sys-seq.gli')m4_dnl

m4_include(`vc-sys-seq_dist.gli')m4_dnl

m4_include(`thy.gli')m4_dnl
```

Same protocol…. No recent or deep assumption

- This makes me think... What is the cost of including the recent/deep theorem?

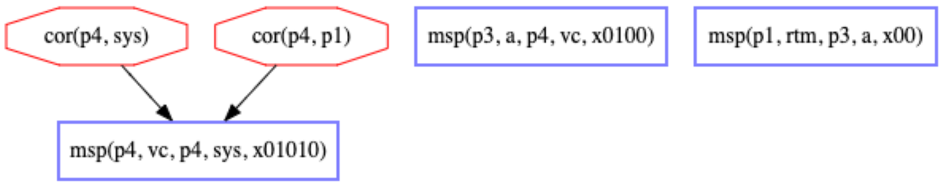


Measure *a* then *vc* then *sys* in parallel

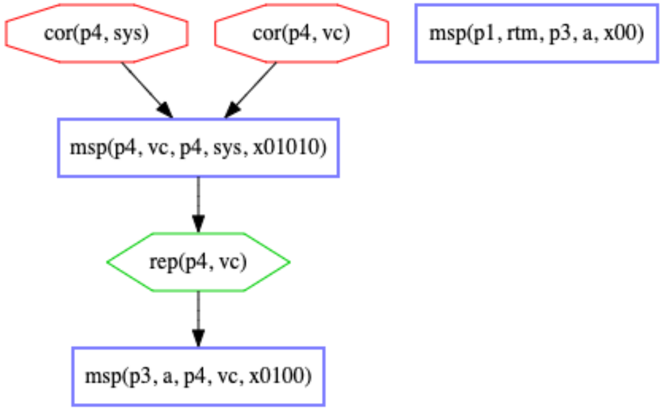
- Protocol
  - *target: @p1 [rtm p3 *a*
    +~+ @p3 [a p4 *vc*
    +~+ @p4 [vc p4 *sys*]]]]

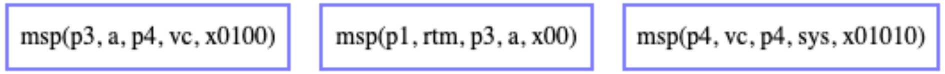
Models

Model 1



Model 2



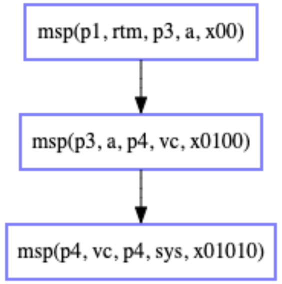


Problem Configuration

```
[ bound = 500, limit = 5000, input_order ]

% Assume adversary avoids detection at
% our main measurement event.
% This is a measurement of sys.
l(V) = msp(p4, M, p4, sys, X)
 => corrupt_at(p4, sys, V).

% system dependencies
depends(p3, C, p3, a) => C = p1.
depends(p1, C, p1, rtm) => false.
depends(p4, C, p4, sys) => C = p1.
depends(p4, C, p4, vc) => C = p1.


% Assume no recent corruptions
prec(V, V1) & l(V1) = cor(P,C) & ms_evt(V)
 => false.

% Assume no deep corruptions
l(V) = cor(p3, M) => false.

m4_include(`a-vc-sys-par.gli')m4_dnl

m4_include(`a-vc-sys-par_dist.gli')m4_dnl

m4_include(`thy.gli')m4_dnl
```

Measure *a* then *vc* then *sys* in sequence

- Protocol
  - *target: @p1 [rtm p3 *a*
    +<+ @p3 [a p4 *vc*
    +<+ @p4 [vc p4 *sys*]]]]

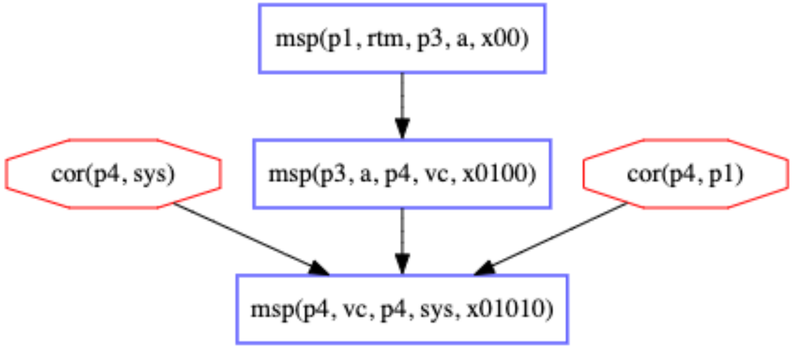


Problem Configuration

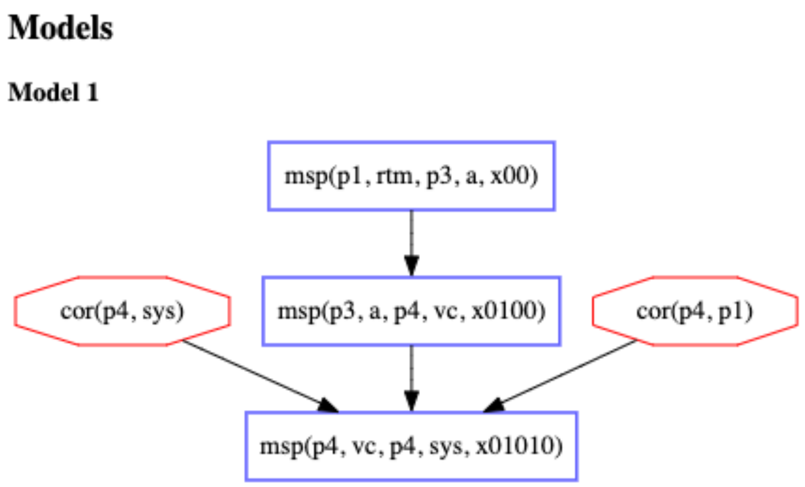
```
[ bound = 500, limit = 5000, input_order ]

% Assume adversary avoids detection at
% our main measurement event.
% This is a measurement of sys.
l(V) = msp(p4, M, p4, sys, X)
 => corrupt_at(p4, sys, V).

% Assume sys depends on kernel
depends(p3, C, p3, a) => C = p1.
depends(p1, C, p1, rtm) => false.
depends(p4, C, p4, sys) => C = p1.
depends(p4, C, p4, vc) => C = p1.

% Assume no recent corruptions
prec(V, V1) & l(V1) = cor(P,C) & ms_evt(V)
  => false.

% Assume no deep corruptions
l(V) = cor(p3, M) => false.

m4_include(`a-vc-sys-seq.gli')m4_dnl

m4_include(`a-vc-sys-seq_dist.gli')m4_dnl

m4_include(`thy.gli')m4_dnl
```

Models

Model 1



Thoughts/Takeaways



- Cost of adding an assumption?
- Cost of adding a dependencies?
- Cost of applying recent/deep theorem?
  - Should we consider this a standard assumption?

- Cost of a corruption/repair event?
  - Maybe turn protocol execution into a tree… then could look at depth of corruption/repair event and that could be the event's cost. Sum all costs together and that is the total cost.