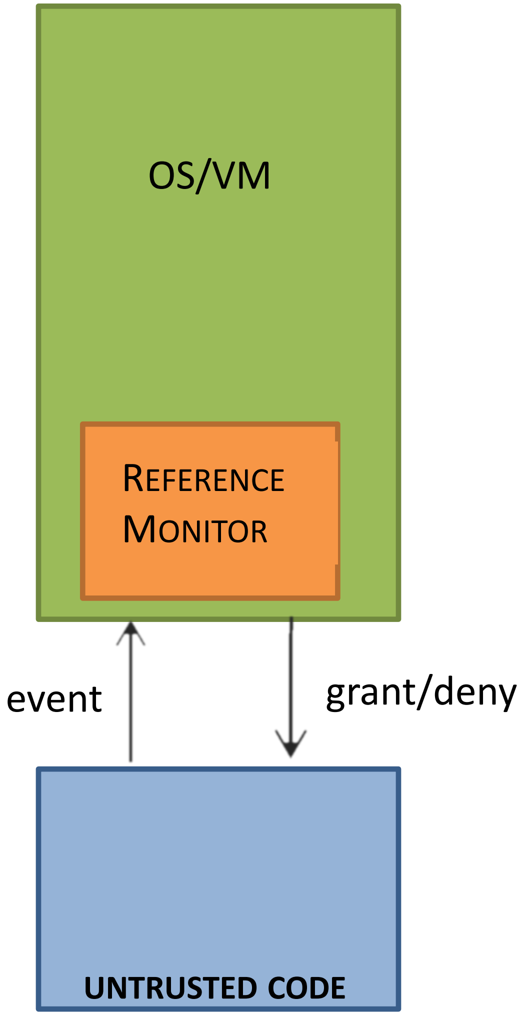
Model-Checking In-lined Reference Monitors

Language-based Security

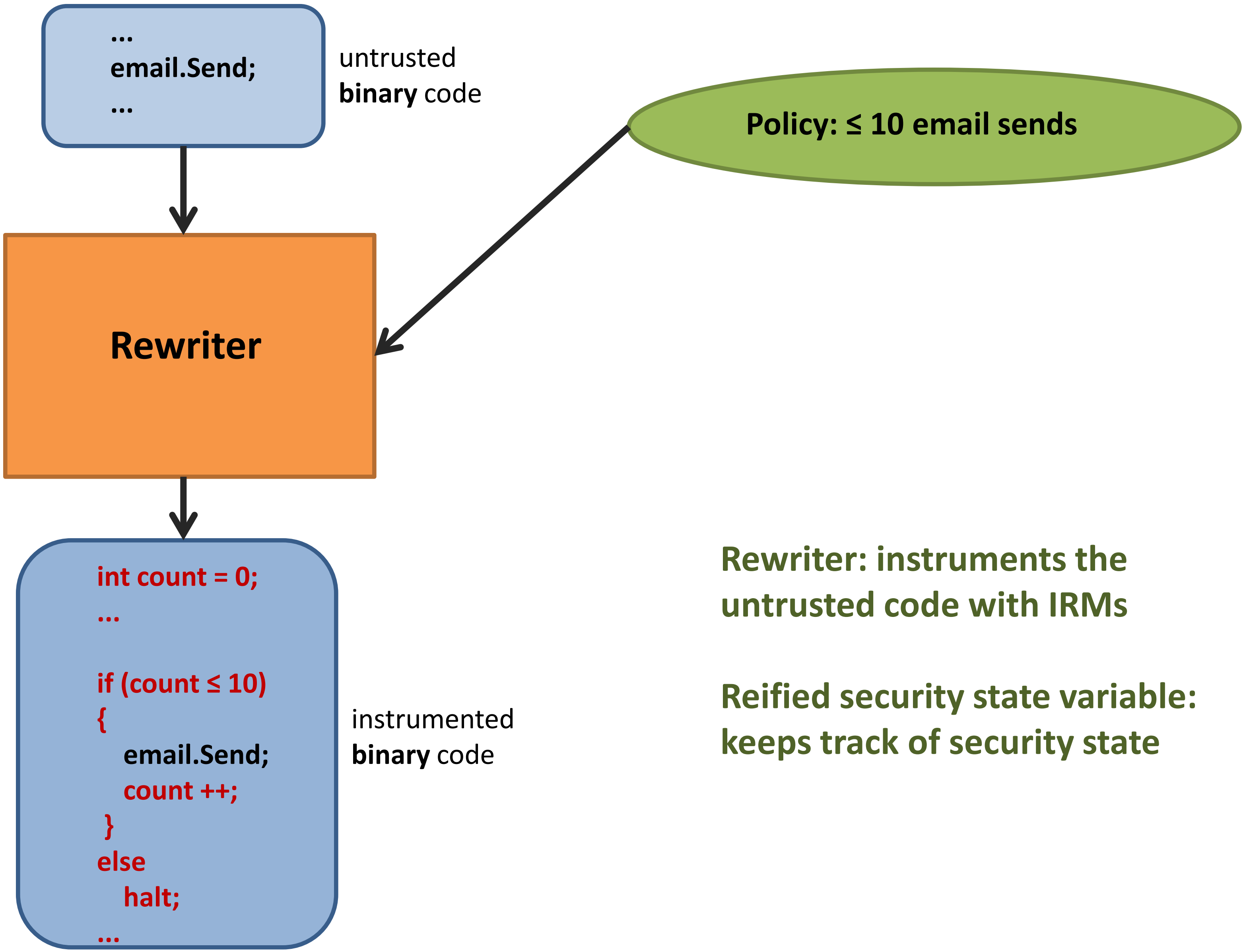
In-lined Reference Monitors (IRMs)

[Schneider, TISSEC, ‘00]



- enforce safety policies by injecting security guards directly into untrusted binaries
- maintain *history* of security-relevant events
- Advantages:
  - deployment flexibility (OS/VM remains unmodified)
  - enforce richer policies, sequence-sensitive policies
  - code recipient can specify security policy
  - application-specific policies

In-lined Reference Monitors

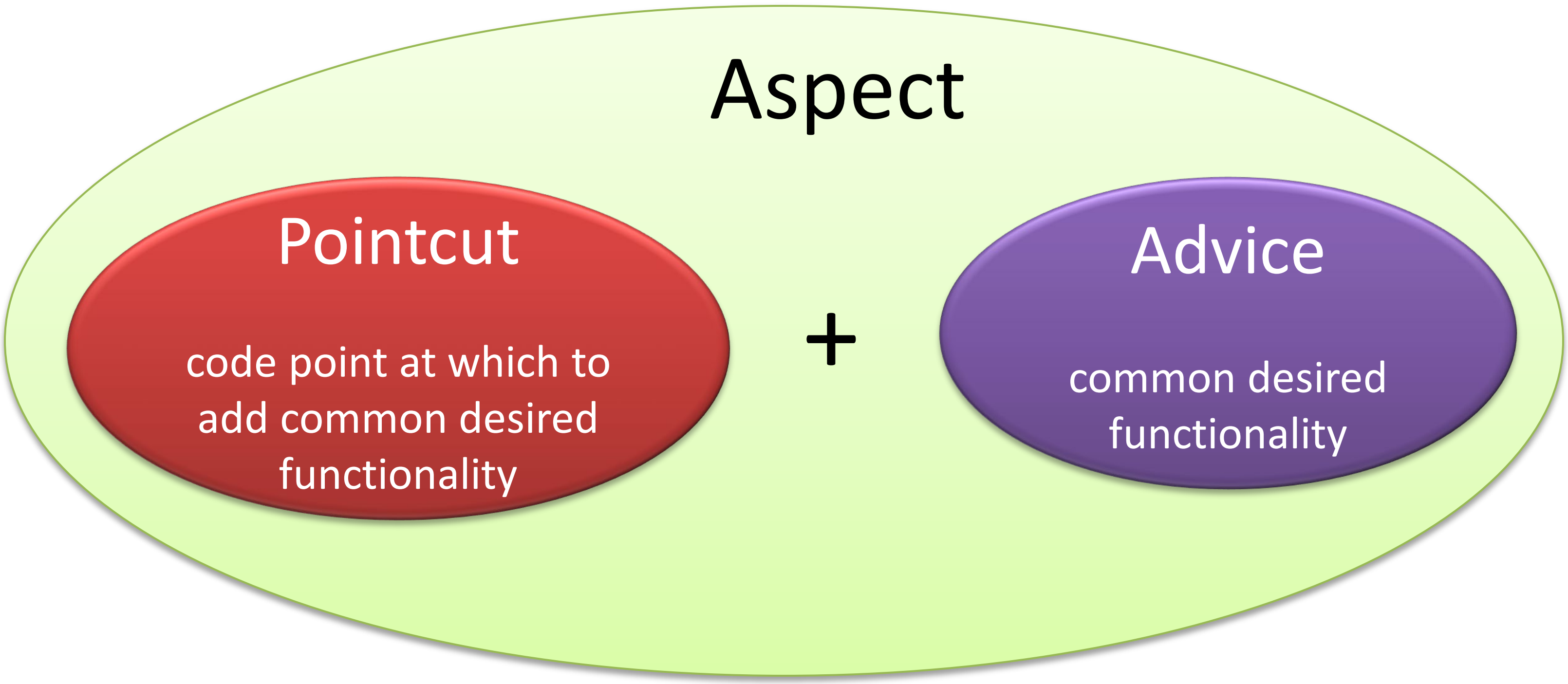


Rewriter: instruments the untrusted code with IRMs

Reified security state variable: keeps track of security state

Aspect-Oriented IRMs

Aspect-Oriented Programming [Kiczales et al, ECOOP, 1997] **has become a standard approach for implementing IRMs**



Aspect-Oriented IRMs

Aspect-Oriented Programming **[Kiczales et al, ECOOP, 1997]** **has become a standard approach for implementing IRMs**

EXAMPLE:

Policy: at most 10 calls to Mail.mail(Mail.Send,…)

AspectJ implementation:

```
aspect Monitor {
    private static int counter = 0;

    pointcut sendevent(x): call(Mail.mail(int,..)) &&
                            if(thisJoinPoint.getArgs()[0]==x);

    before() : sendevent(Mail.Send) {
        if (counter >= 10)
            throw new Exception("security violation");
        ++counter;
    }
}
```

reified security state

pointcuts: identify security-relevant operations (events)

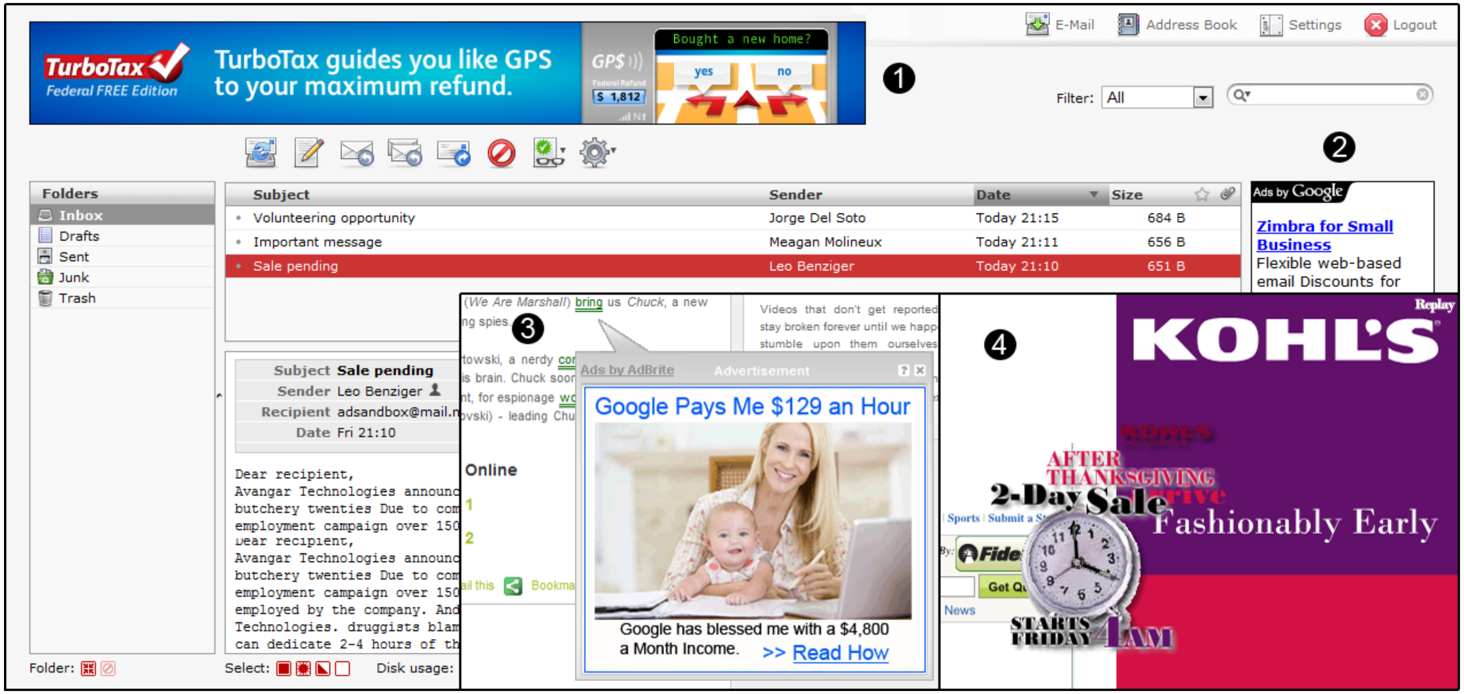
advice: implement guards and interventions

In-lined Reference Monitors

- Long history of IRM Implementations
  - SASI/PoET [Erlingsson & Schneider, NSPW 99]
  - MOBILE [Hamlen, Morrisett, & Schneider, PLAS 06]
  - Polymer [Ligatti, Bauer, & Walker, TISSEC 09]
  - Java-MOP [Chen & Roşu, TACAS 05]
  - ConSpec [Aktug & Naliuka, SCP 08]
  - FIRM [Li & Wang, ACSAC 10]
  - many others

IRM Example: Web Ad Security

[Louw, Ganesh, Venkatakrishnan, USENIX Security, 2010]

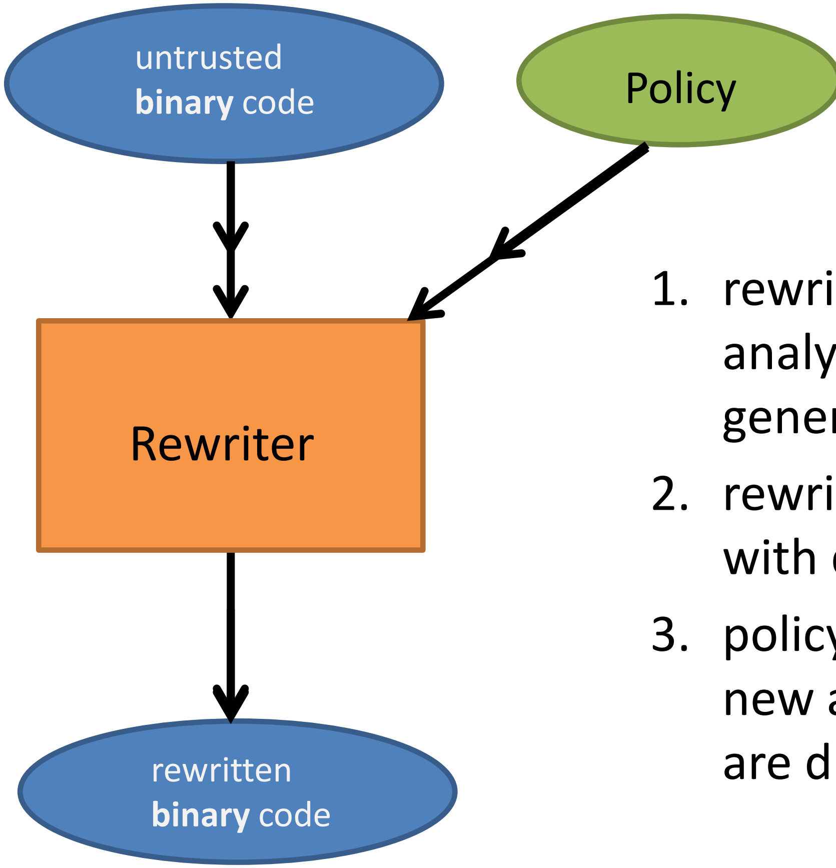


Third Party Ad content given full page access by default! – Confidentiality and Integrity issues

1. Banner ad
2. Skyscraper ad – needs to read page for *contextual targeting* – risk of exposing private content such as email ids
3. Inline text ad – *contextual targeting* – same risk
4. Floating ad – needs control of page real estate – may interfere with trusted components

Phu H. Phung, Maliheh Monshizadeh, Meera Sridhar, Kevin Hamlen and V.N. Venkatakrishnan. ***Between Worlds: Securing Mixed JavaScript/ActionScript Multi-party Web Content***. IEEE Transactions on Dependable and Secure Computing, November 2014.

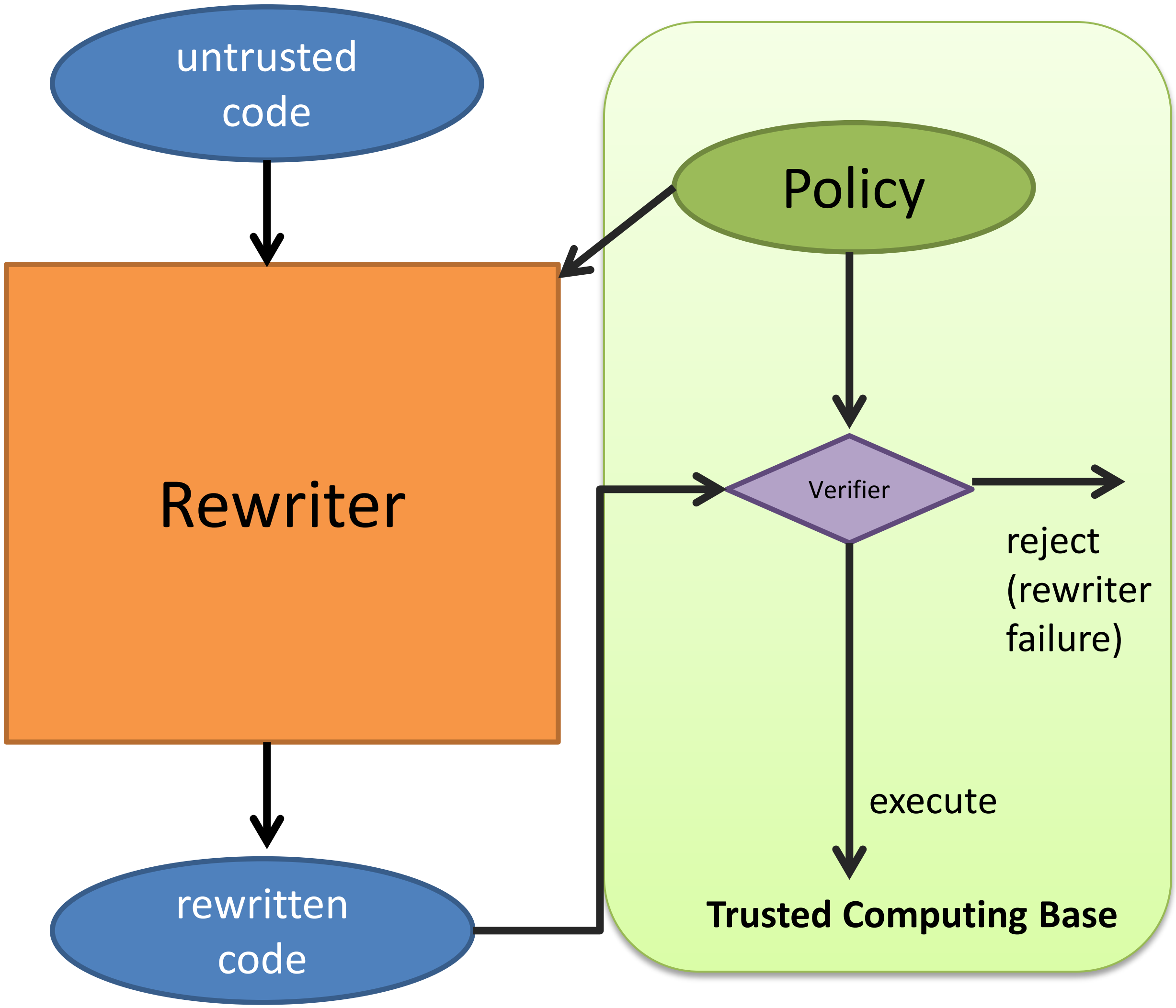
Certifying In-lined Reference Monitors



1. rewriters contain disassemblers, binary analysis tools, compilers, optimizers, code-generators
2. rewriters may be outsourced to third parties with different security interests
3. policy specifications can change rapidly as new attacks appear and new vulnerabilities are discovered

Without certification, TCB large & complex!

Certifying In-lined Reference Monitors



- certifying IRMs easier than verifying safety of arbitrary code!
- lighter weight
  - SPIN vs. our early work
- different from Proof-Carrying Code (PCC)
  - PCC rewriters (certifying compilers) leverage source level info typically unavailable to binary rewriters
- Related work:
  - ConSpec (certification via contracts)
  - MoBILe (certification via type-checking)

Certifying In-lined Reference Monitors

Bottom Line: Runtime monitoring is very powerful, but we want the high assurance of static analysis.

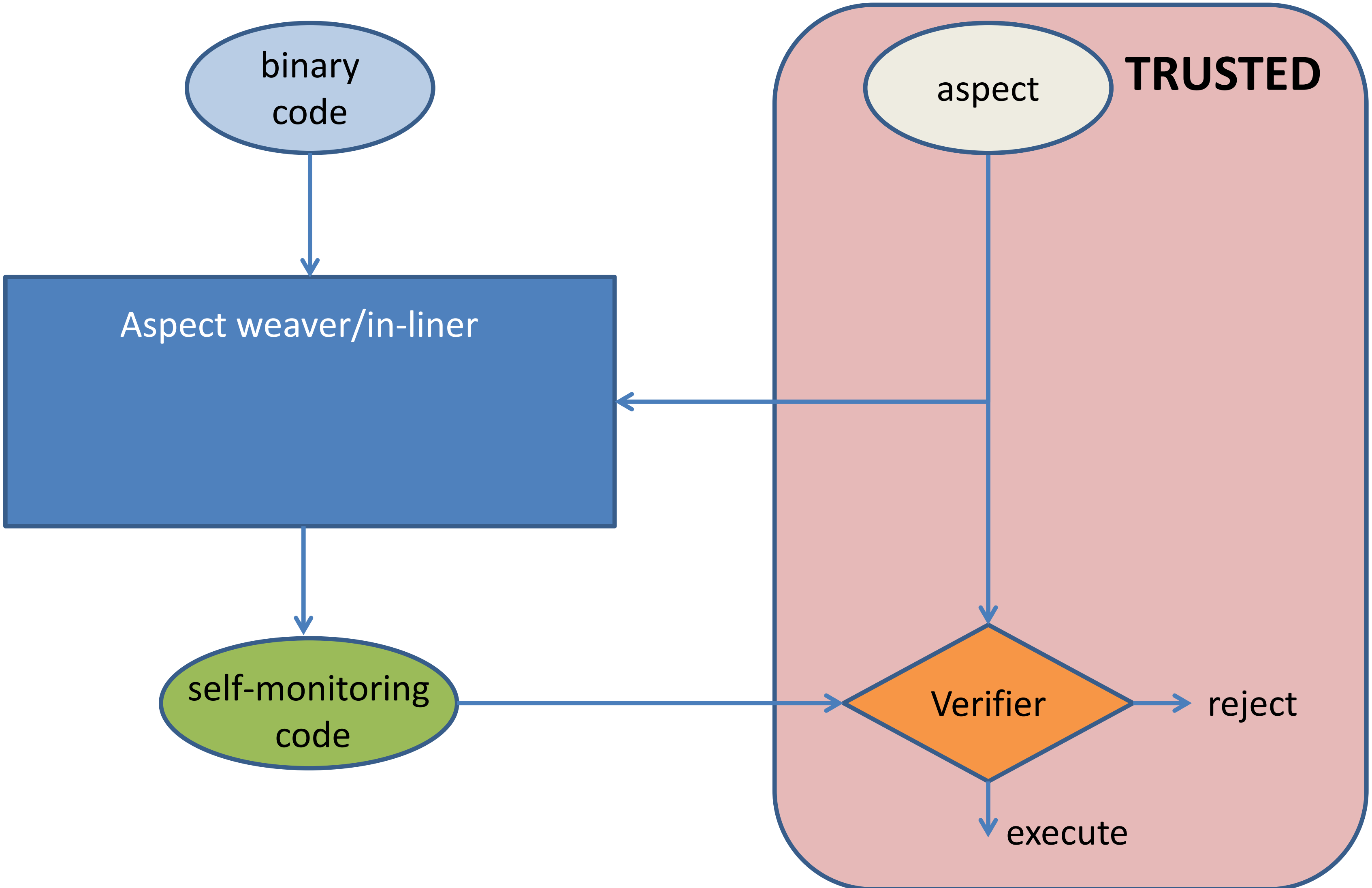
Solution: *Static verification of IRMs* yields best of both worlds! Combine the power & flexibility of runtime monitoring with strong formal guarantees of static analysis.

Certifying In-lined Reference Monitors

What do we want from the certifier?

- automatic, machine-certification of IRMs on-demand
- formal guarantees of
  - ✓ **soundness**
  - ✓ **transparency (behavior-preservation)**
- light-weight certifier (embedded systems)

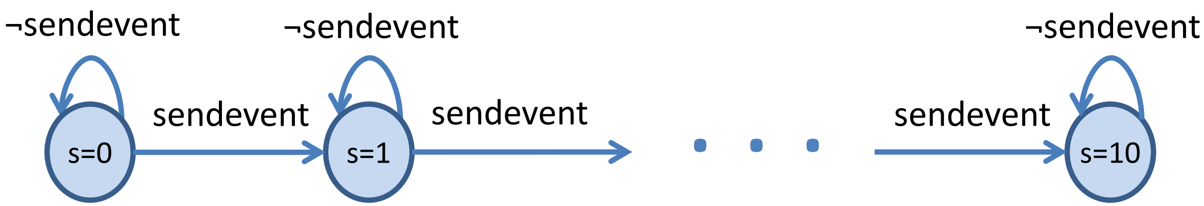
Aspect-Oriented IRM In-lining and Certification



SPoX Policy Example [Hamlen, Jones, PLAS, 2008]

Policy: at most 10 calls to Mail.mail(Mail.Send,…)

Security Automaton:



SPoX formalization:

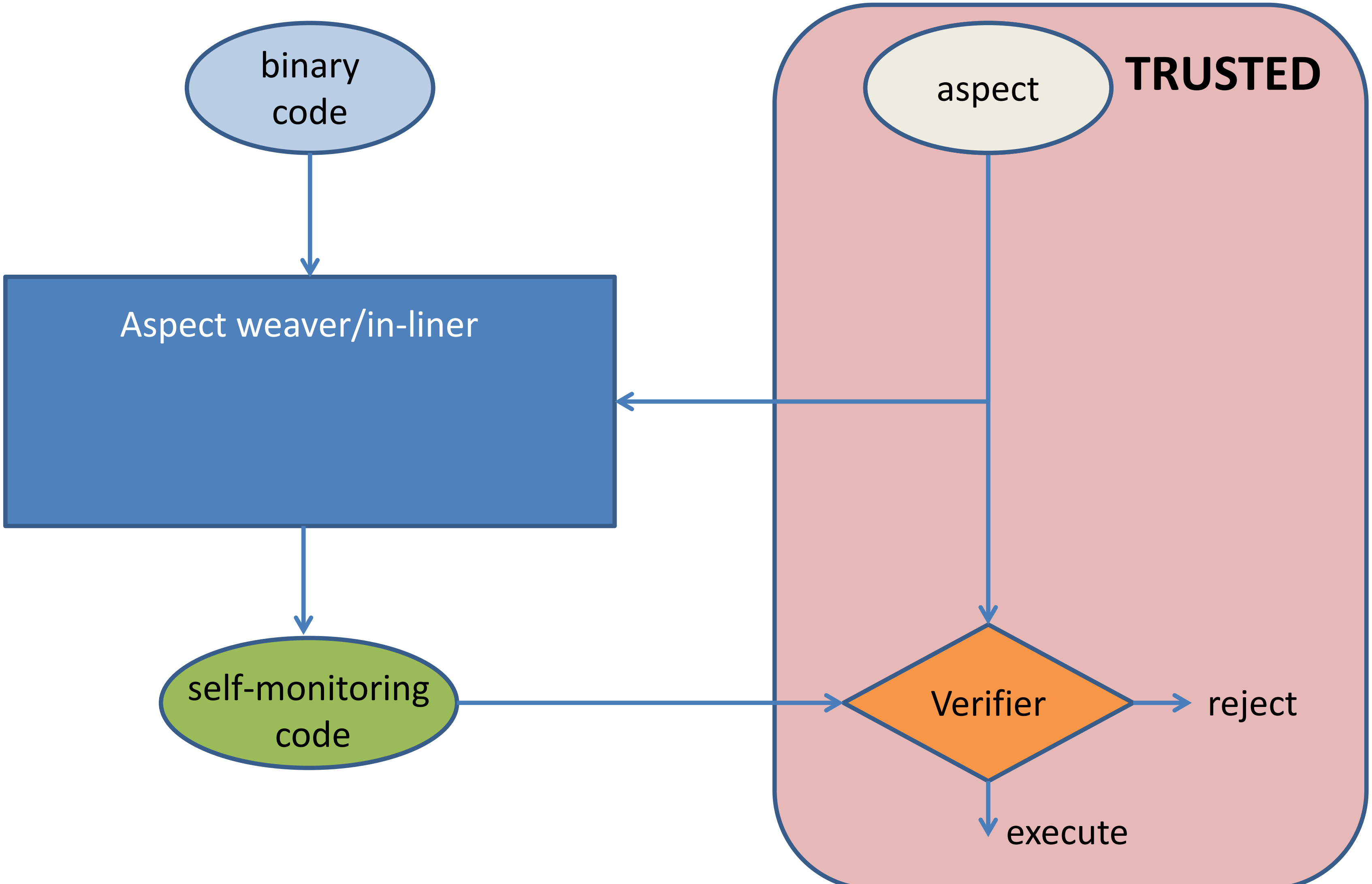
```
(state name="s")
(pointcut name="sendevent"
    (and (call Mail.mail) (argval 1 (inteq Mail.Send))))
(forall "i" from 0 to 9
    (edge name="increment"
        (pc name="sendevent")
        (nodes "s" i, i+1)))
(edge name="violation"
    (pc name="sendevent")
    (nodes "s" 10, #))
```

abstract security state

pointcuts: automaton edge labels (events)

edges: security state transitions

Aspect-Oriented IRM In-lining and Certification



Approach: Model-checking

- policy model + new binary code are the two inputs to model-checker
- model-checking process
  - abstract-interpret new binary code
  - interpreter bi-simulates code and automaton
  - model-checker proves that there are no automaton-rejected states in any reachable flows
- **Main Challenge: How to curb state-space explosion?**

Meera Sridhar and Kevin W. Hamlen. ***Model Checking In-Lined Reference Monitors***. In Proc. of the Eleventh International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI), Jan 2010.

Kevin W. Hamlen, Micah M. Jones, and Meera Sridhar. ***Aspect-oriented Runtime Monitor Certification***. In Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), March 2012.

In-lining Example

Policy: at most 10 calls to Mail.mail(Mail.Send,…)

```
if (x == Mail.Send) {
    if (counter >= 0 && counter <= 9)
        temp_counter = counter + 1;
    else
        throw new Exception("security violation");
    counter = temp_counter;
}
Mail.mail(x,…);
```

Abstract Interpretation Example

Policy: at most 10 calls to Mail.mail(Mail.Send,…)

```
if (x == Mail.Send) {
    if (counter >= 0 && counter <= 9)
        temp_counter = counter + 1;
    else
        throw new Exception("security violation");
    counter = temp_counter;
}
Mail.mail(x,…);
```

s≤10 ∧ s=c

s≤10 ∧ s=c ∧ x≠Mail.Send

Legend:
- s = abstract security state (from SPoX policy)
- c = counter (reified state)
- t = temp_counter (reified state)

Abstract Interpretation Example

Policy: at most 10 calls to Mail.mail(Mail.Send,…)

```
if (x == Mail.Send) {
    if (counter >= 0 && counter <= 9)
        temp_counter = counter + 1;
    else
        throw new Exception("security violation");
    counter = temp_counter;
}
Mail.mail(x,…);
```

$s \leq 10 \land s=c$

$s \leq 10 \land s=c \land x=\text{Mail.Send}$

$\ldots \land c \geq 0 \land c \leq 9$

$\ldots \land t=c+1$

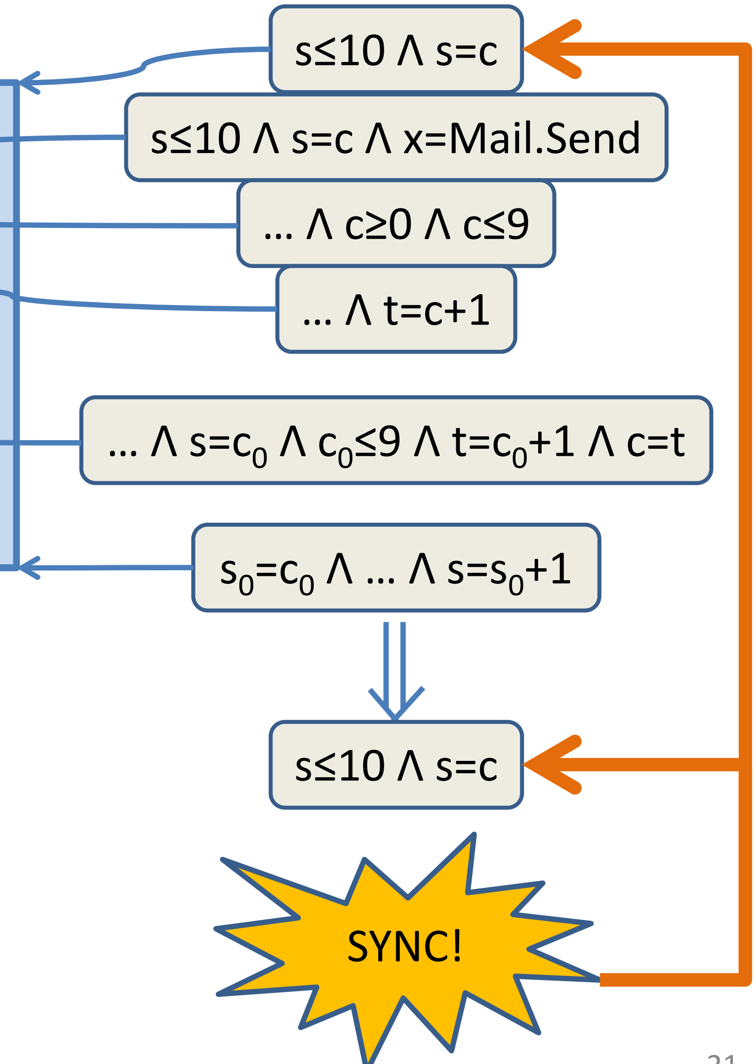
$\ldots \land s=c_0 \land c_0 \leq 9 \land t=c_0+1 \land c=t$

Legend:
- s = abstract security state (from SPoX policy)
- c = counter (reified state)
- t = temp_counter (reified state)

Abstract Interpretation Example

Policy: at most 10 calls to Mail.mail(Mail.Send,…)

```
if (x == Mail.Send) {
    if (counter >= 0 && counter <= 9)
        temp_counter = counter + 1;
    else
        throw new Exception("security violation");
    counter = temp_counter;
}
Mail.mail(x,…);    // s=s+1
```



Legend:

s = abstract security state (from SPoX policy)

c = counter (reified state)

t = temp_counter (reified state)

Synchronization States

- Definition
  - A state is *synchronized* when the abstract and reified security states "match"
  - different definition of "match" for each aspect implementation
  - each binary rewriter declares its definition of "match"
  - definition remains untrusted by verifier!
- Certification
  - verifies that initial symbolic state is synchronized
  - abstracts state to just "sync" whenever possible
  - uses "sync" as a loop invariant whenever possible
  - conservatively rejects if "sync" is insufficient to verify safety
- Controlling state-space explosion
  - vast majority of state-exploration reduces to linear-time sync-preservation checks
  - remaining exploration verifies that small blocks of in-lined code are sync-preserving, and that sync-preservation implies safety
  - "wrong" definition of sync just causes conservative rejection or slow convergence

Kevin W. Hamlen, Micah M. Jones, and Meera Sridhar. ***Aspect-oriented Runtime Monitor Certification***. In Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), March 2012.

Model-checking Certifier Implementation for SPoX IRM System

- IRM system for Java bytecode
- Prolog (about 5200 lines)
  - implements abstract interpreter
  - implements model-checker
    - decides boolean sentences over symbolic states
    - implemented with Constraint Logic Programming (CLP)
- Java code (about 9100 lines)
  - parses Java bytecode binaries using BCEL
  - outputs Prolog structures for certification
  - answers Prolog's questions(e.g., class inheritence)
- Capabilities and limitations
  - certifier fully inter-procedural and inter-modular
  - almost all loops verify easily using sync as loop invariant
    - monitor-introduced loops in non-sync regions (rare) are the only hard ones
  - supports most forms of reflection
    - certifier just verifies adequacy of guards of reflective operations
  - synchronization invariant must be expressible as linear constraints
  - multithreading not supported

Model-checking Certifier Implementation for SPoX IRM System

Program	Policy	File Sizes (KB)			# Classes		Rewrite Time (s)	# Events	Total Verif. Time (s)	Model-check Time (s)
		old	new	libs	old	libs				
EJE	NoExecSaves	439	439	0	147	0	6.1	1	202.8	16.3
RText		1264	1266	835	448	680	52.1	7	2797.5	54.5
JSesh		1923	1924	20878	863	1849	57.8	1	5488.1	196.0
vrenamer	NoExecRename	924	927	0	583	0	50.1	9	1956.8	41.0
jconsole	NoUnsafeDel	35	36	0	33	0	0.6	2	115.7	15.1
jWeather	NoSendsAfterReads	288	294	0	186	0	12.3	46	308.2	156.7
YTDownload		279	281	0	148	0	17.8	20	219.0	53.6
jfilecrypt	NoGui	303	303	0	164	0	9.7	1	642.2	2.8
jknight	OnlySSH	166	166	4753	146	2675	4.5	1	650.1	3.0
Multivalent	EncryptPDF	1115	1116	0	559	0	129.9	7	3567.0	26.9
tn5250j	PortRestrict	646	646	0	416	0	85.4	2	2598.2	23.6
jrdesktop	SafePort	343	343	0	163	0	8.3	5	483.0	17.8
JVMail	TenMails	24	25	0	21	0	1.6	2	35.1	8.0
JackMail		165	166	369	30	269	2.5	1	626.7	8.9
Jeti	CapLoginAttmpts	484	484	0	422	0	15.3	1	524.3	8.8
ChangeDB	CapMembers	82	83	404	63	286	4.3	2	995.3	12.0
projtimer	CapFileCreates	34	34	0	25	0	15.3	1	56.2	6.1
xnap	NoFreeRiding	1250	1251	0	878	0	24.8	4	1496.2	56.4
Phex		4586	4586	3799	1353	830	69.4	2	5947.0	172.7
Webgoat	NoSqlXss	429	431	6338	159	3579	16.7	2	10876.0	120.0
OpenMRS	NoSQLInject	1781	1783	24279	932	17185	78.7	6	2897.0	37.3
Averages		**747**	**748**	**2522**	**369**	**1120**	**32.4**	**5**	**1846.6**	**45.2**

IRM Implementation Challenges & Logic Programming Advantage

1. IRMs must be fairly light-weight because they run on the code-consumer side
2. binary code parsing, code generation: tedious and error-prone
   - DCG's facilitate binary parser implementation
   - Reversible predicates combine parser and code-generator into one piece of code!
3. IRM must elegantly implement many AST analyses and optimizations during rewriting
   - needed to preserve policy-compliant programs, generate efficient code
   - ASTs very elegantly represented and manipulated as Prolog structures
4. Instrumented code should be amenable to formal verification
   - Prolog implementation of binary rewriting isomorphic to a search for a correctness proof
   - excellent for integration with a certifying IRM system or a PCC system

Brian W. DeVries, Gopal Gupta, Kevin W. Hamlen, Scott Moore, and Meera Sridhar. ***ActionScript Bytecode Verification With Co-Logic Programming***. In Proc. of the ACM SIGPLAN Workshop on Prog. Languages and Analysis for Security (PLAS), June 2009.

Meera Sridhar and Kevin W. Hamlen. ***ActionScript In-Lined Reference Monitoring in Prolog***. In Proceedings of the Twelfth Symposium on Practical Aspects of Declarative Languages (PADL), Jan 2010.

A Simple LTL Model Checker written in Prolog for ActionScript Bytecode

```
% verify/2 takes a state and an existentially
% quantified LTL formula and checks
% whether the formula holds for that state.
%
% Atomic Propositions are labeled by 'ap'.
%
% holds/2 is true when the atomic proposition holds
% in the current state
%
% ftype/2 is a mapping from top-level temporal
% operators to their interpretation semantics
%
% The clause for 'a and b' should ensure that 'a' and
% 'b' hold on the same execution path. For simplicity
% of presentation, we omit this check here.

verify(State, F) :- ftype(F, inductive),
        verify_inductive(State, F).
verify(State, F) :- ftype(F, coinductive),
        verify_coinductive(State, F).

:- tabled verify_inductive/2.
verify_inductive(S, ap(AP)) :- holds(S,AP). % p
% Logical operators
verify_inductive(S, not(ap(AP))) :-      % not(p)
        \+ holds(S, AP).
verify_inductive(S, or(A,B)) :-          % a or b
        verify(S, A) ; verify(S, B).
verify_inductive(S, and(A,B)) :-         % a and b
        verify(S, A), verify(S, B).
% Inductive temporal operators
verify_inductive(S, x(A)) :-             % X(a)
        trans(S, S1), verify(S1, A).
verify_inductive(S, f(A)) :-             % F(a)
        verify(S, A); verify(S, x(f(A))).
verify_inductive(S, u(A,B)) :-           % a U b
        verify(S, B);
        verify_inductive(S, and(A, x(u(A,B)))).

:- coinductive verify_coinductive/2.
% Coinductive temporal operators
verify_coinductive(S, g(A)) :-           % G(a)
        verify(S, and(A, x(g(A)))).
verify_coinductive(S, r(A,B)) :-         % a R b
        verify(S, and(A,B)).
        % {a and b both occur, releasing b}
verify_coinductive(S, r(A,B)) :-
        verify(S, and(B, x(r(A,B)))).
        % {a does not hold, so b is not released}
```

FlashJaX: IRM technology for Web Ads

Proof of Certifier Correctness

certifier returns true ⟹ for all executions of the program there is no policy violation

Proof based on Cousot's abstract interpretation framework [Cousot & Cousot, POPL 77]

- bismulation of concrete and abstract machines
  - concrete operational semantics of Java bytecode based on ClassicJava [Flatt, Krishnamurthi, & Felleisen, POPL 98]
  - abstract operational semantics of our interpreter
  - *soundness* relation between abstract and concrete states
- denotational semantics of SPoX [Hamlen & Jones, PLAS 08]
- preservation: The abstract machine soundly abstracts the concrete machine step-wise (uses soundness relation).
- progress: If the abstract machine doesn't reject, the concrete machine doesn't violate the policy. Abstract machine covers all real executions.

Kevin W. Hamlen, Micah M. Jones, and Meera Sridhar. ***Aspect-oriented Runtime Monitor Certification***. In Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), March 2012.

Kevin W. Hamlen, Micah M. Jones, and Meera Sridhar. ***Chekov: Aspect-oriented Runtime Monitor Certification via Model-checking (Extended Version)***. Technical Report UTDCS-16-11, Computer Science Department, The University of Texas at Dallas, Richardson, Texas, May 2011.

Concrete Machine

LANGUAGE SYNTAX (SIMPLIFIED ACTIONSCRIPT)	$i ::= \textbf{ifle}\ L \mid \textbf{getlocal}\ n \mid \textbf{setlocal}\ n \mid \textbf{jmp}\ L \mid \textbf{event}\ e \mid \textbf{setstate}\ n \mid \textbf{ifstate}\ n\ L$

PROGRAMS AND LABELS

$P ::= (L, p, s)$	(programs)
$p : L \to i$	(instruction labels)
$s : L \to L$	(label successors)

CONCRETE STATES

$\chi ::= \langle L : i, \sigma, \nu, m, \tau \rangle$	(configurations)
$\sigma ::= \cdot \mid v :: \sigma$	(concrete stacks)
$v \in \mathbb{Z}$	(concrete values)
$\nu : \mathbb{Z} \to v$	(concrete stores)
$m \in \mathbb{Z}$	(concrete reified state)
$e \in \Sigma$	(events)
$\tau \in \Sigma^*$	(concrete traces)
$\chi_0 = \langle L_0 : p(L_0), \cdot, \nu_0, 0, \epsilon \rangle$	(initial configurations)
$\nu_0 = \mathbb{Z} \times \{0\}$	(initial stores)

Meera Sridhar and Kevin W. Hamlen. ***Model Checking In-Lined Reference Monitors***. In Proc. of the Eleventh International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI), Jan 2010.

Concrete Small-step Operational Semantics

$$\frac{n_1 \leq n_2}{\langle L_1 : \textbf{ifle } L_2, n_1{::}n_2{::}\sigma, \nu, m, \tau\rangle \mapsto \langle L_2 : p(L_2), \sigma, \nu, m, \tau\rangle}\ (\text{CIflePos})$$

$$\frac{n_1 > n_2}{\langle L_1 : \textbf{ifle } L_2, n_1{::}n_2{::}\sigma, \nu, m, \tau\rangle \mapsto \langle s(L_1) : p(s(L_1)), \sigma, \nu, m, \tau\rangle}\ (\text{CIfleNeg})$$

$$\frac{}{\langle L : \textbf{getlocal } n, \sigma, \nu, m, \tau\rangle \mapsto \langle s(L) : p(s(L)), \nu(n){::}\sigma, \nu, m, \tau\rangle}\ (\text{CGetlocal})$$

$$\frac{}{\langle L : \textbf{setlocal } n, n_1{::}\sigma, \nu, m, \tau\rangle \mapsto \langle s(L) : p(s(L)), \sigma, \nu[n := n_1], m, \tau\rangle}\ (\text{CSetlocal})$$

$$\frac{}{\langle L_1 : \textbf{jmp } L_2, \sigma, \nu, m, \tau\rangle \mapsto \langle L_2 : p(L_2), \sigma, \nu, m, \tau\rangle}\ (\text{CJmp})$$

$$\frac{\tau e \in \mathcal{P}}{\langle L : \textbf{event } e, \sigma, \nu, m, \tau\rangle \mapsto \langle s(L) : p(s(L)), \sigma, \nu, m, \tau e\rangle}\ (\text{CEvent})$$

$$\frac{}{\langle L : \textbf{setstate } n, \sigma, \nu, m, \tau\rangle \mapsto \langle s(L) : p(s(L)), \sigma, \nu, n, \tau\rangle}\ (\text{CSetstate})$$

$$\frac{}{\langle L_1 : \textbf{ifstate } n\ L_2, \sigma, \nu, n, \tau\rangle \mapsto \langle L_2 : p(L_2), \sigma, \nu, n, \tau\rangle}\ (\text{CIfstatePos})$$

$$\frac{m \neq n}{\langle L_1 : \textbf{ifstate } n\ L_2, \sigma, \nu, m, \tau\rangle \mapsto \langle s(L_1) : p(s(L_1)), \sigma, \nu, m, \tau\rangle}\ (\text{CIfstateNeg})$$

Abstract Machine

Abstract States

$$\hat{\chi} ::= \bot \mid \langle L : i, \hat{\sigma}, \hat{\nu}, m, (Res(q_m), \bar{\tau})\rangle \mid \langle L : i, \hat{\sigma}, \hat{\nu}, \top_{VS}, \hat{\tau}\rangle \qquad \text{(abstract configs)}$$

$$\hat{\sigma} ::= \cdot \mid \hat{v} :: \hat{\sigma} \qquad \text{(evaluation stacks)}$$

$$\hat{v} \in VS \qquad \text{(abstract values)}$$

$$\hat{\nu} : \mathbb{Z} \to \hat{v} \qquad \text{(abstract stores)}$$

$$\hat{m} \in \mathbb{Z} \cup \top_{VS} \qquad \text{(abstract reified state)}$$

$$\bar{\tau} \in \cup_{n \leq k} \Sigma^n \qquad \text{(bounded traces)}$$

$$\hat{\tau} \in SS \qquad \text{(abstract traces)}$$

Abstract Small-step Operational Semantics

$$\frac{n_1 \leq n_2}{\langle L_1 : \textbf{ifle } L_2, n_1{::}n_2{::}\hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle \rightsquigarrow \langle L_2 : p(L_2), \hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle}\text{(AIfLEPos)}$$

$$\frac{n_1 > n_2}{\langle L_1 : \textbf{ifle } L_2, n_1{::}n_2{::}\hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle \rightsquigarrow \langle s(L_1) : p(s(L_1)), \hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle}\text{(AIfLENeg)}$$

$$\frac{\top_{VS} \in \{va_1, va_2\} \qquad L' \in \{L_2, s(L_1)\}}{\langle L_1 : \textbf{ifle } L_2, va_1{::}va_2{::}\hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle \rightsquigarrow \langle L' : p(L'), \hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle}\text{(AIfLETop)}$$

$$\frac{}{\langle L : \textbf{getlocal } n, \hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle \rightsquigarrow \langle s(L) : p(s(L)), \hat{\nu}(n){::}\hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle}\text{(AGetLocal)}$$

$$\frac{}{\langle L : \textbf{setlocal } n, va_1{::}\hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle \rightsquigarrow \langle s(L) : p(s(L)), \hat{\sigma}, \hat{\nu}[n := va_1], \hat{m}, \hat{\tau}\rangle}\text{(ASetLocal)}$$

$$\frac{}{\langle L_1 : \textbf{jmp } L_2, \hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle \rightsquigarrow \langle L_2 : p(L_2), \hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle}\text{(AJmp)}$$

$$\frac{\hat{\tau}e \subseteq \hat{\tau}' \subseteq \mathcal{P}}{\langle L : \textbf{event } e, \hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle \rightsquigarrow \langle s(L) : p(s(L)), \hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}'\rangle}\text{(AEvent)}$$

$$\frac{\hat{\tau} \subseteq Res(q_n)}{\langle L : \textbf{setstate } n, \hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle \rightsquigarrow \langle s(L) : p(s(L)), \hat{\sigma}, \hat{\nu}, n, (Res(q_n), \epsilon)\rangle}\text{(ASetState)}$$

$$\frac{\hat{m} \in \{n, \top\}}{\langle L_1 : \textbf{ifstate } n\ L_2, \hat{\sigma}, \hat{\nu}, \hat{m}, (S, \tau)\rangle \rightsquigarrow \langle L_2 : p(L_2), \hat{\sigma}, \hat{\nu}, n, (Res(q_n), \tau)\rangle}\text{(AIfStatePos)}$$

$$\frac{\hat{m} \neq n \qquad (S - Res(q_n))\tau \subseteq \hat{\tau}}{\langle L_1 : \textbf{ifstate } n\ L_2, \hat{\sigma}, \hat{\nu}, \hat{m}, (S, \tau)\rangle \rightsquigarrow \langle s(L_1) : p(s(L_1)), \hat{\sigma}, \hat{\nu}, \hat{m}, \hat{\tau}\rangle}\text{(AIfStateNeg)}$$

Other Proofs of Correctness

- Proof of Convergence
  - proof bounds height of abstraction lattice
  - abstract machine reaches fixed point in $O(n^2)$, n = security automaton size

Meera Sridhar and Kevin W. Hamlen. ***Model Checking In-Lined Reference Monitors***. In Proc. of the Eleventh International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI), Jan 2010.

- Proof of Correctness of IRM *Transparency* Certifier
  - SCP paper presents the first automated transparency-verifier for IRMs
  - untrusted, external invariant-generator
    - safely leverages rewriter-specific instrumentation information during verification
  - correctness of IRM transparency certifier extends previous proof with trace equivalence

Meera Sridhar, Richard Wartell and Kevin W. Hamlen. ***Hippocratic Binary Instrumentation: First Do No Harm***. Science of Computer Programming: Special Issue on Invariant Generation, 93(B):110-124, Nov. 2014.

References

Irem Aktug, Mads Dam, and Dilian Gurov. **Provably Correct Runtime Monitoring**. *In Proc. of the International Symposium on Formal Methods (FM):* 262 – 277, 2008.

Irem Aktug and Katsiaryna Naliuka. **ConSpec – A Formal Language for Policy Specification**. *Science of Computer Programming*. 74: 2 – 12, 2008.

Lujo Bauer, Jay Ligatti, and David Walker. **Composing Security Policies with Polymer**. *In Proc. of the ACM Conference on Programming Languages Design and Implementation (PLDI):* 305 – 314, 2005.

Patrick Cousot and Radhia Cousot. **Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints**. *In Proc. of the ACM Symposium of Programming Languages (POPL)*: 234 – 25, 1977.

Patrick Cousot and Radhia Cousot. **Abstract Interpretation Frameworks***. Journal of Logic and Computation*:2(4): 511-547, 1992.

Brian W. DeVries, Gopal Gupta, Kevin W. Hamlen, Scott Moore, and Meera Sridhar. **ActionScript Bytecode Verification With Co-Logic Programming**. *In Proc. of the ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*, June 2009.

Ulfar Erlingsson and Fred B. Schneider. **SASI Enforcement of Security Policies: A Retrospective**. *In Proc. of the New Security Paradigms Workshop (NSPW):* 87 – 95, 1999.

Matthew Flatt, Shriram Krishnamurthi, and Matthias Fellesein. **Classes and Mixins**. *In Proc. of the ACM Symposium on Principles of Programming Languages (POPL):*171-183, 1998.

References

Kevin W. Hamlen and Micah Jones. **Aspect-oriented In-lined Reference Monitors.** In *Proc. ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*: 11-20, 2008.

Kevin W. Hamlen, Micah M. Jones, and Meera Sridhar. **Chekov: Aspect-oriented runtime monitor certification via model-checking (extended version).** Technical Report UTDCS-16-11, University of Texas at Dallas, 2011.

Kevin W. Hamlen, Micah M. Jones, and Meera Sridhar. **Aspect-oriented Runtime Monitor Certification**. *In Proc. of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2012.

Kevin W. Hamlen, Greg Morrisett, and Fred B. Schneider. **Certified In-lined Reference Monitoring on .NET**. *In Proc. of the ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS): 7-16, 2006.*

Kevin W. Hamlen, Greg Morrisett, and Fred B. Schneider. **Computability Classes for Enforcement Mechanisms.** *ACM Trans. Prog. Lang. and Systems (TOPLAS), 28(1):175-205, 2006.*

Gregor Kiczales, John Lamping, Anurag Medhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. **Aspect-Oriented Programming**. *In Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP)*, 1997.

Zhou Li, XiaoFeng Wang. **FIRM: Capability-based Inline Mediation of Flash Behaviors**. *The 26th Annual Computer Security Applications Conference (ACSAC), 2010.*

Mike Ter Louw, Karthik Thotta Ganesh, and V. N. Venkatakrishnan. **AdJail: Practical Enforcement of Confidentiality and Integrity Policies on Web Advertisements.** *USENIX Security Symposium,* 2010.

References

George C. Necula. **Proof Carrying Code**. *In Proc. of the ACM Symposium of Programming Languages (POPL):* 106 – 119, 1997.

Phu H. Phung, Maliheh Monshizadeh, Meera Sridhar, Kevin Hamlen and V.N. Venkatakrishnan. ***Between Worlds: Securing Mixed JavaScript/ActionScript Multi-party Web Content***. Submitted to IEEE Transactions on Dependable and Secure Computing.

Fred B. Schneider. **Enforceable Security Policies**. *ACM Transactions on Information and Systems Security (TISSEC)*. 3(1):30 – 50, 2000.

Meera Sridhar and Kevin W. Hamlen. **Model Checking In-Lined Reference Monitors**. *In Proc. of the Eleventh International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI),* 2010.

Meera Sridhar and Kevin W. Hamlen. **ActionScript In-Lined Reference Monitoring in Prolog**. *In Proc. of the Twelfth Symposium on Practical Aspects of Declarative Languages (PADL),* 2010.

Meera Sridhar and Kevin W. Hamlen. **Flexible In-lined Reference Monitor Certification: Challenges and Future Directions.** *In Proc. 5th ACM SIGPLAN Workshop on Programming Languages meets Program Verification (PLPV):* 55-60, 2011.

Meera Sridhar, Richard Wartell and Kevin W. Hamlen. ***Hippocratic Binary Instrumentation: First Do No Harm***. Submitted to Science of Computer Programming: Special Issue on Invariant Generation, in **second round review, minor revision.**