Generalizing permissive-upgrade in dynamic information flow analysis

Vineet Rajani, Deepak Garg
MPI-SWS

Joint work with
Abhishek Bichhawat, Christian Hammer
University of Saarland

Funded by the Deutscheforschungsgemeinschaft (DFG) priority program – Reliably Secure Software Systems (RS3)

Objective

Build **Information Flow Control (IFC)** for **web browsers**

- Performance and permissiveness

Plan:

- JavaScript, DOM, event handlers, local storage
- Declassification

Summary of results: POST'14

- Hybrid approach for IFC for Webkit's JS Bytecode:
  - Taint tracking
  - Immediate postdominator analysis
- Complete JavaScript: eval, exceptions, return in the middle and all unstructured control flow
- Performance: ~ 40% on micro benchmarks
- Deferred NSU: Permissive handling of implicit flows

Focus of this talk

Generalizing Permissive upgrade strategy to arbitrary lattices.

Implicit leak

Low:Visible, High: Secret

- Due to control structure

```
v := 0
if (s=1){
  v := 1
}
```

- No direct assignment

Secret gets leaked **via visible variable**

Program counter label

```
v := 0
if (s=1){
  v := 1
}
```

PC = ●

- **Lower bound** on the **taint** of the variables on which the current instruction is **control dependent**

No Sensitive Upgrade

```
v := 0, w := 0
if (s=1){
  v := 1
}
if (v=0)
v := 1
 w := 1
```

s = 1=> w = 0

s = 0=> w = 1

- No Sensitive Upgrade (NSU) [1]
- Does not allow assignment to low variables under high guard
- Ends up over-approximating the set of safe programs.
- Sound but gives some obvious false positives

1. Stephan A. Zdancewic, PhD thesis, 2002.

Permissive Upgrade Strategy

- Permissive Upgrade Strategy (PUS) [1]

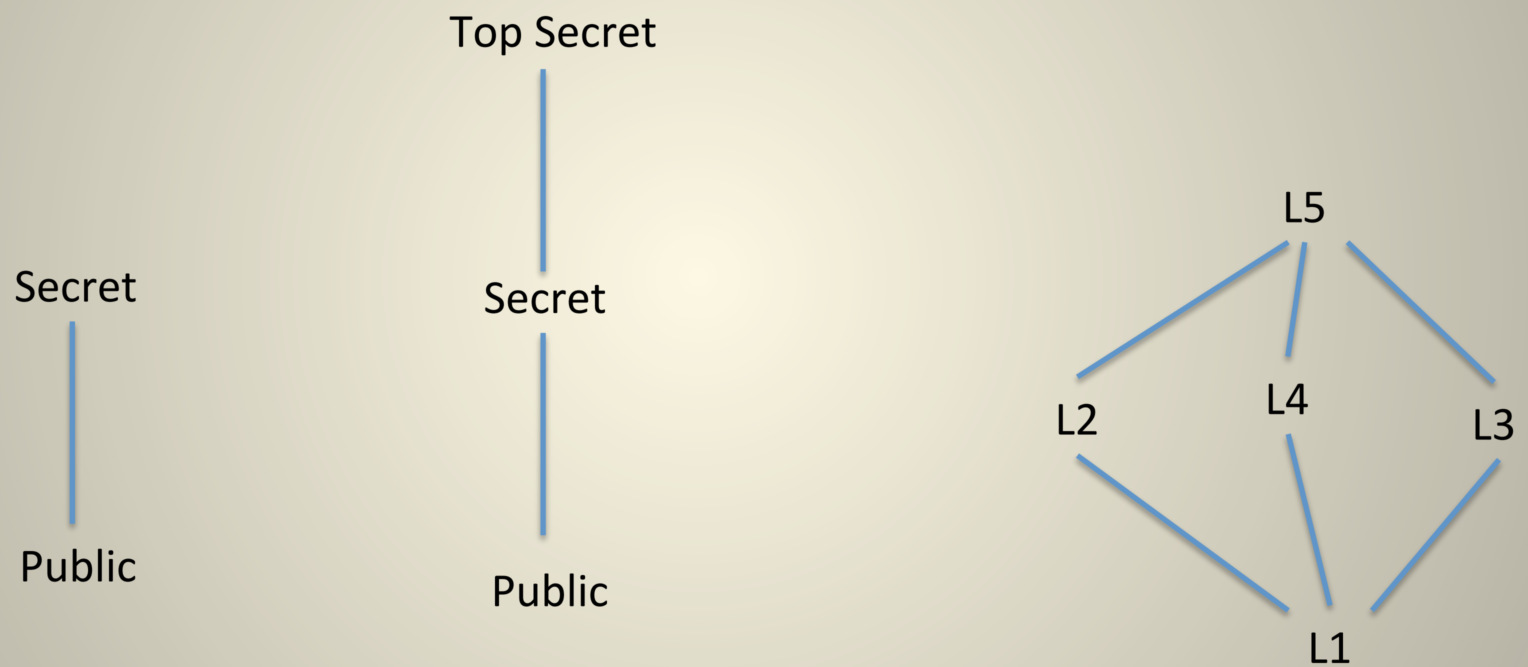
```
v := 0, w:=0
if (s=1)
  v := 1

→ if (v=0)
   w := 1
```

"P": label for Partially leaked data

- The non-leaky program is permitted by this approach

1. Austin and Flanagan, Permissive Dynamic Information Flow Analysis, PLAS 10

Security Lattice



There is a problem

- NSU generalizes to arbitrary lattice
- It is not clear if PUS generalizes to arbitrary lattices[1]

1. Buiras et al, On dynamic flow-sensitive floating label systems, CSF 14

Our contribution: PLAS'14

- It is indeed possible to generalize permissive upgrade to arbitrary lattices
- We present a provably sound approach

Outline

- New label for specifying partial leaks
- Assignment rules and examples
- Soundness
- Comparison

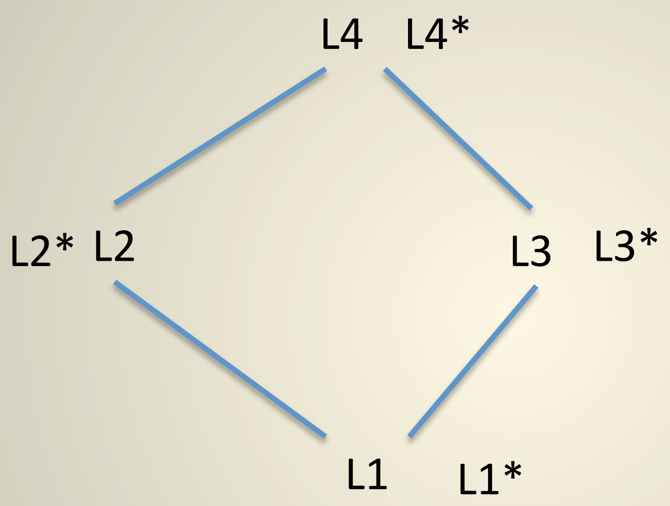
Standard while language

$$
\begin{aligned}
e &:= n \mid x \mid e_1 \odot e_2 \\
c &:= x := e \mid c_1; c_2 \mid \\
&\quad\ \texttt{if } e \texttt{ then } c_1 \texttt{ else } c_2 \mid \\
&\quad\ \texttt{while } e \texttt{ do } c_1
\end{aligned}
$$

New Label for Partial leak

- When we work with arbitrary lattice a single partially leaked label “P” is too coarse grained
- Every label ***A*** in the lattice has a corresponding ***A**** label

New Label for Partial leak



- Intuition of ***A****: ***A*** is a lower bound on the label in all alternate executions

Assignment rules

Case: No variable upgrade

$$\text{assn-1: } \frac{l := \Gamma(\sigma(x)) \qquad \langle e, \sigma \rangle \Downarrow n^m \qquad l = \mathcal{A}_x \vee l = \mathcal{A}_x{}^{\star} \qquad pc \sqsubseteq \mathcal{A}_x \qquad k := pc \sqcup m}{\langle x := e, \sigma \rangle \Downarrow_{pc} \sigma[x \mapsto n^k]}$$

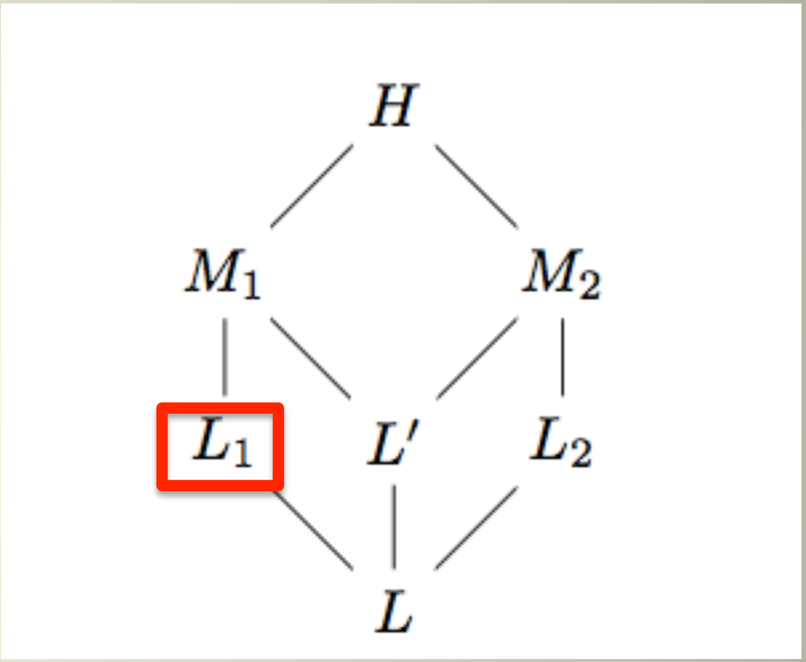
Case: Variable upgrade

$$\text{assn-2: } \frac{l := \Gamma(\sigma(x)) \qquad \langle e, \sigma \rangle \Downarrow n^m \qquad l = \mathcal{A}_x \vee l = \mathcal{A}_x{}^{\star} \qquad pc \not\sqsubseteq \mathcal{A}_x \qquad k := (\mathcal{A}_x)^{\star}}{\langle x := e, \sigma \rangle \Downarrow_{pc} \sigma[x \mapsto n^k]}$$

$$\text{assn-2: } \frac{l := \Gamma(\sigma(x)) \qquad \langle e, \sigma \rangle \Downarrow n^m \qquad l = \mathcal{A}_x \vee l = \mathcal{A}_x{}^{\star} \qquad pc \not\sqsubseteq \mathcal{A}_x \qquad k := (pc \sqcap \mathcal{A}_x)^{\star}}{\langle x := e, \sigma \rangle \Downarrow_{pc} \sigma[x \mapsto n^k]}$$

Example

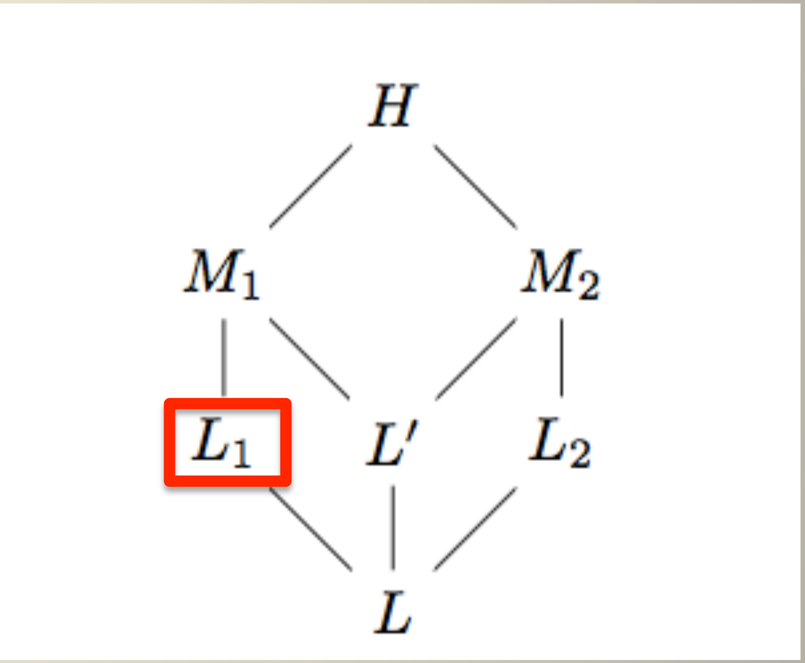
- Attacker at level L1



Execution

w=false^{L1}, x1 = true^{L1}, y1= false^{M1}, y2=true^{M2}

x'=true$^{L'}$
x2=true^{L2}




```
   if(x')
→   z := y1          pc=L', z=false^M1
   else
    z := y2
   if(x1)
→   z := x1          pc=L1, z=true^L1
   if(not(x2))
    z := x2          branch not taken
   if (z)
→   w := z           pc=L1, w=true^L1
```

Execution with A_x*

w=false^{L1}, x1 = true^{L1}, y1= false^{M1}, y2=true^{M2}

x'=false$^{L'}$
x2=false^{L2}

```
if(x')
  z := y1
else
  z := y2
if(x1)
  z := x1
if(not(x2))
  z := x2
if (z)
  w := z
```

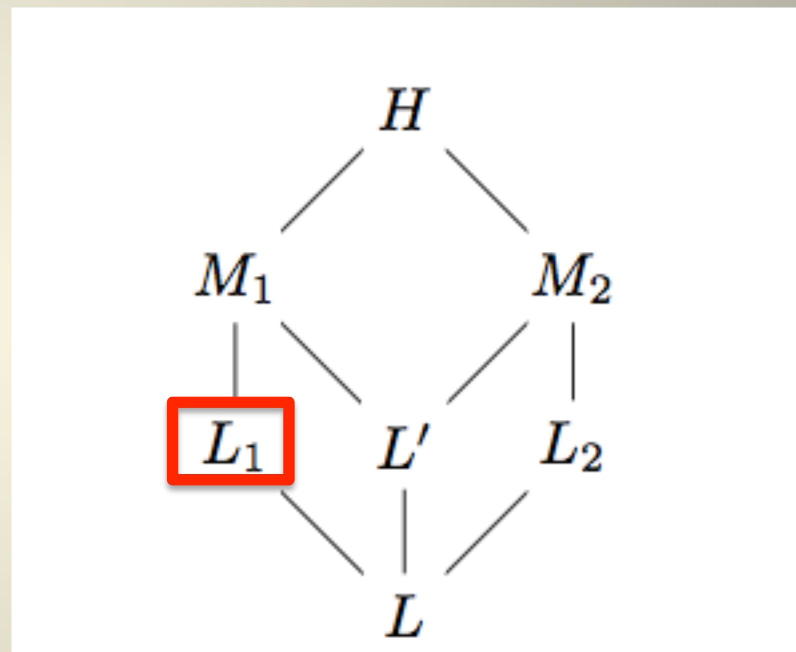
→ z := y2: pc=L', z=true^{M2}

→ z := x1: pc=L1, z=true^{M2*}

→ z := x2: pc=L2, z=false^{L2}

w := z: Branch not taken

w=false^{L1}



'w' differs in 2 runs

Information leaked

Execution with (pc Π A_x)*

w=$false^{L1}$, x1 = $true^{L1}$, y1= $false^{M1}$, y2=$true^{M2}$

x'=$false^{L'}$
x2=$false^{L2}$

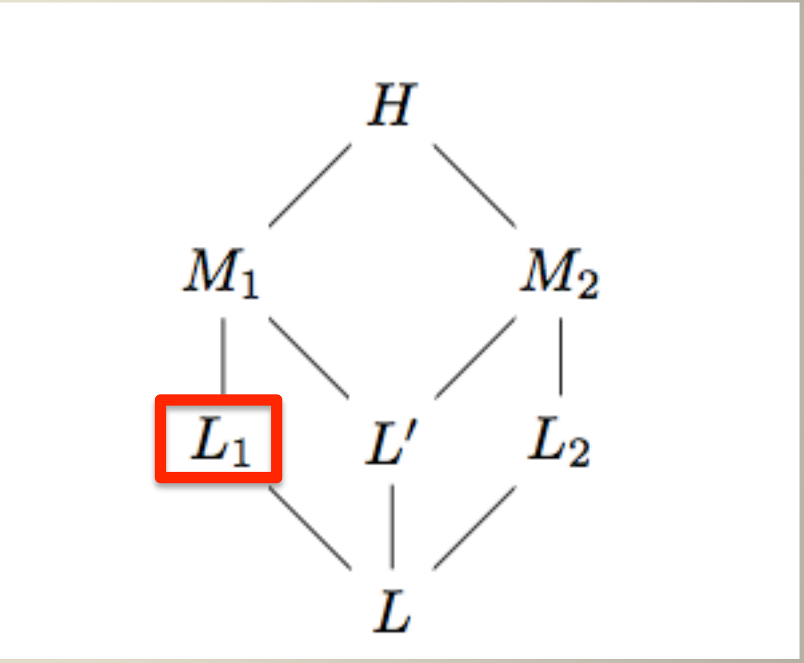
```
if(x')
  z := y1
else
→ z := y2
if(x1)
→ z := x1
if(not(x2))
→ z := x2
if (z)
  w := z
```

pc=L', z=$true^{M2}$

pc=L1, z=$true^{L*}$

pc=L2, z=$false^{L*}$

Execution halted



Safe (Termination insensitively)

Memory equivalence

Definition 5. *Two values* n_1^k *and* n_2^m *are* $\mathcal{A}$*-equivalent, written* $n_1^k \sim_{\mathcal{A}} n_2^m$*, iff either*

1. $k = m = \mathcal{A}' \sqsubseteq \mathcal{A}$ *and* $n_1 = n_2$*, or*
2. $k = \mathcal{A}' \not\sqsubseteq \mathcal{A}$ *and* $m = \mathcal{A}'' \not\sqsubseteq \mathcal{A}$*, or*
3. $k = \mathcal{A}_1{}^\star$ *and* $m = \mathcal{A}_2{}^\star$*, or*
4. $k = \mathcal{A}_1{}^\star$ *and* $m = \mathcal{A}_2$ *and* $(\mathcal{A}_2 \not\sqsubseteq \mathcal{A}$ *or* $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2)$*, or*
5. $k = \mathcal{A}_1$ *and* $m = \mathcal{A}_2{}^\star$ *and* $(\mathcal{A}_1 \not\sqsubseteq \mathcal{A}$ *or* $\mathcal{A}_2 \sqsubseteq \mathcal{A}_1)$

Memory equivalence

- We obtain this by constructing examples of all possible transitions of pairs of labels

- Necessary and sufficient
  - Necessary: because we can construct example programs which use these states.
  - Sufficient: because it suffices to prove sondness

Soundness

Theorem 4 (TINI for generalized permissive-upgrade). *If* $\sigma_1 \sim_{\mathcal{A}} \sigma_2$ *and* $\langle c, \sigma_1 \rangle \Downarrow_{pc} \sigma_1'$ *and* $\langle c, \sigma_2 \rangle \Downarrow_{pc} \sigma_2'$, *then* $\sigma_1' \sim_{\mathcal{A}} \sigma_2'$.

- $\sim_A$ is not transitive
- some additional lemmas

Comparison

- Generalization from 2 element lattice to pointwise product lattice[1]
- Both approaches are sound
- Since both of them apply to powerset lattice
  - Which one is more permissive ?
- Neither is more permissive than the other in all cases

1. Austin and Flanagan, Permissive Dynamic Information Flow Analysis, PLAS 10

Conclusion

- It is indeed possible to generalize permissive upgrade to arbitrary lattices.
- Design choices are quite non-trivial
  - Assignment rules are really non-obvious
  - Equivalence definition is quite involved
- Proved the soundness of permissive upgrade strategy for generalized lattice

Thank You!!