Australian Government
Department of Defence
Science and Technology

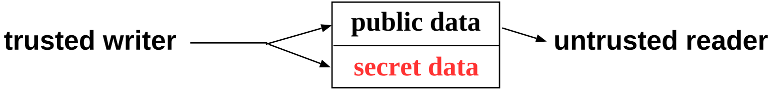
Backwards-directed information flow analysis for concurrent programs

Kirsten Winter*, Nicholas Coughlin, Graeme Smith
DST Group and The University of Queensland
Australia

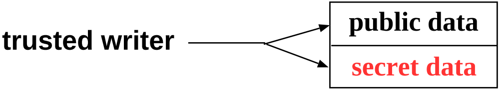
Cyber Security Foundations Symposium
21 - 24 June 2021

Information flow security for cross-domain components

Information flow security for cross-domain components



Value-dependent information flow security (IFS)



sync_write(dataType secret) :

```
while (¬ CAS(z, 0, 1)) {
   while (z ≠ 0) {}
}
x := secret;
...
x := 0;
z := 0;
```

z=1

Value-dependent information flow security (IFS)

trusted writer → **public data** | **secret data**

```
sync_write(dataType secret) :
while (¬ CAS(z, 0, 1)) {
  while (z ≠ 0) {}
}
x := secret;
...
x := 0;
z := 0;
```

control variable z
(to alter the security classification of x)

Value-dependent information flow security (IFS)

trusted writer

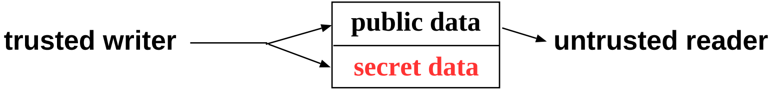
public data
secret data

sync_write(dataType secret) :
while ($\neg$ CAS(z, 0, 1)) {
  while ($z \neq 0$) {}
}
x := secret;
. . .
x := 0;
z := 0;

control variable z
(to alter the security classification of x)

PO
proof obligation to guarantee IFS

Information flow in concurrent programs



Information flow in concurrent programs


```
sync_write(dataType secret) :
while (¬ CAS(z, 0, 1)) {
   while (z ≠ 0) {}
}
x := secret;
...
x := 0;
z := 0;
```

```
sync_read : dataType
dataType y;
if (CAS(z, 0, 2)) {
   y := x;
   z := 0;
   return y;
}
```

Information flow in concurrent programs


```
sync_write(dataType secret) :
while (¬ CAS(z, 0, 1)) {
   while (z ≠ 0) {}
}
x := secret;
...
x := 0;
z := 0;
```

```
sync_read : dataType
dataType y;
if (CAS(z, 0, 2)) {
   y := x;
   z := 0;
   return y;
}
```

Information flow in concurrent programs

sync_write(dataType secret) :

```
while (¬ CAS(z, 0, 1)) {
   while (z ≠ 0) {}
}
x := secret;
...
x := 0;
z := 0;
```

$\mathcal{R}_{read}$ rely condition

$(x{=}x') \wedge (z{=}1 \Rightarrow z{=}z')$

Information flow in concurrent programs

trusted writer → **public data** / **secret data** → **untrusted reader**

sync_write(dataType secret) :
while ($\neg$ CAS(z, 0, 1)) {
  while ($z \neq 0$) {}
}
x := secret;
. . .
x := 0;
z := 0;

$\mathcal{R}_{read}$ rely condition

$(x{=}x') \wedge (z{=}1 \Rightarrow z{=}z')$

guarantee condition

$$\mathcal{G}_{write} \Rightarrow \mathcal{R}_{write}$$

Information flow in concurrent programs

trusted writer → **public data** / **secret data** → **untrusted reader**

sync_write(dataType secret) :
while ($\neg$ CAS(z, 0, 1)) {
  while (z $\neq$ 0) {}
}
x := secret;
...
x := 0;
z := 0;

stable$_{\mathcal{R}}$(PO)

$\mathcal{R}_{read}$ rely condition

$(x{=}x') \wedge (z{=}1 \Rightarrow z{=}z')$

$$stable_{\mathcal{R}}(p) \mathrel{\widehat{=}} p \wedge \mathcal{R} \Rightarrow p'$$

Proof Obligations to guarantee IFS

Sec: lattice of security values (e.g., *true* as *low* and *false* as *high*)

$\mathcal{L} : Var \rightarrow \mathcal{P}red$ security classification (value dependent)

$\Gamma : Var \rightarrow Sec$ security level of data held in a variable ($\rightsquigarrow \Gamma_E$)

e.g., $\mathcal{L}(x) = (z \neq 1)$ and $\mathcal{L}(z) = true$

Proof Obligations to guarantee IFS

Sec: lattice of security values (e.g., *true* as *low* and *false* as *high*)

$\mathcal{L} : Var \rightarrow \mathcal{P}red$ security classification (value dependent)

$\Gamma : Var \rightarrow Sec$ security level of data held in a variable ($\leadsto \Gamma_E$)

e.g., $\mathcal{L}(x) = (z \neq 1)$ and $\mathcal{L}(z) = true$

$$PO(z := 0) \mathrel{\widehat{=}} (z \in \mathcal{G} \Rightarrow (\mathcal{L}(z) \Rightarrow \Gamma_E(0))) \wedge (z \in \mathcal{C} \Rightarrow secureUpd(z := 0))$$

$$secureUpd(z := 0) \mathrel{\widehat{=}} \forall x \in ctrled(z).\, \mathcal{L}(x)[z \leftarrow 0] \Rightarrow \Gamma_x \vee \mathcal{L}(x)$$

Proof Obligations to guarantee IFS

Sec: lattice of security values (e.g., *true* as *low* and *false* as *high*)
$\mathcal{L} : Var \to \mathcal{P}red$ security classification (value dependent)
$\Gamma : Var \to Sec$ security level of data held in a variable ($\leadsto \Gamma_E$)

e.g., $\mathcal{L}(x) = (z \neq 1)$ and $\mathcal{L}(z) = true$

$$\begin{aligned} PO(z := 0) \mathrel{\widehat{=}} & (z \in \mathcal{G} \Rightarrow (\mathcal{L}(z) \Rightarrow \Gamma_E(0))) \wedge \\ & (z \in \mathcal{C} \Rightarrow secureUpd(z := 0)) \end{aligned}$$

$$\begin{aligned} & secureUpd(z := 0) \mathrel{\widehat{=}} \\ & \quad \forall x \in ctrled(z).\, \mathcal{L}(x)[z \leftarrow 0] \Rightarrow \Gamma_x \vee \mathcal{L}(x) \\ & \quad \Gamma_x \vee (z \neq 1) \end{aligned}$$

Backwards analysis via weakest precondition reasoning

$wp(c, Q)$:finds all states for which instruction c results in states Q
$wpif(c, Q) \mathrel{\widehat{=}} PO(c) \wedge wp(c, Q)$

Backwards analysis via weakest precondition reasoning

$wp(c, Q)$:finds all states for which instruction c results in states Q
$wpif(c, Q) \mathrel{\widehat{=}} PO(c) \wedge wp(c, Q)$

wpif

sync_write(dataType secret) :
while (¬ CAS(z, 0, 1)) {
  while (z ≠ 0) {}
}

wpif x := secret;

wpif x := 0;

wpif z := 0;

Backwards analysis via weakest precondition reasoning

$wp(c, Q)$:finds all states for which instruction c results in states Q

$wpif(c, Q) \mathrel{\widehat{=}} PO(c) \wedge wp(c, Q)$

wpif

sync_write(dataType secret) :
while ($\neg$ CAS(z, 0, 1)) {
  while (z $\neq$ 0) {}
}

wpif

x := secret;

wpif

x := 0;

wpif

z := 0;

true

Backwards analysis via weakest precondition reasoning

$wp(c, Q)$:finds all states for which instruction c results in states Q
$wpif(c, Q) \mathrel{\widehat{=}} PO(c) \wedge wp(c, Q)$

wpif

sync_write(dataType secret) :

```
while (¬ CAS(z, 0, 1)) {
  while (z ≠ 0) {}
}
```

wpif

x := secret;

wpif

x := 0;

PO(z:=0)

wpif

z := 0;

true

Backwards analysis via weakest precondition reasoning

$wp(c, Q)$:finds all states for which instruction c results in states Q
$wpif(c, Q) \mathrel{\widehat{=}} PO(c) \wedge wp(c, Q)$

```
sync_write(dataType secret) :
while (¬ CAS(z, 0, 1)) {
  while (z ≠ 0) {}
}
```

wpif

x := secret;

wpif

$PO(x{:=}0) \wedge wp(x{:=}0, \_\_)$

x := 0;

wpif

$PO(z{:=}0)$

z := 0;

wpif

true

Backwards analysis via weakest precondition reasoning

$wp(c, Q)$:finds all states for which instruction c results in states Q
$wpif(c, Q) \mathrel{\widehat{=}} PO(c) \wedge wp(c, Q)$

sync_write(dataType secret) :

```
while (¬ CAS(z, 0, 1)) {
  while (z ≠ 0) {}
}
```

wpif

PO(x:=secret) ∧ wp(x:=secret, __)

wpif x := secret;

PO(x:=0) ∧ wp(x:=0, __)

wpif x := 0;

PO(z:=0)

wpif z := 0;

true

Backwards analysis via weakest precondition reasoning

$wp(c, Q)$:finds all states for which instruction c results in states Q
$wpif(c, Q) \mathrel{\widehat{=}} PO(c) \wedge wp(c, Q)$

P_0 ==> $PO(while...) \wedge wp(while, \_\_)$

wpif

sync_write(dataType secret) :
while ($\neg$ CAS(z, 0, 1)) {
  while (z $\neq$ 0) {}
}

$PO(x:=secret) \wedge wp(x:=secret, \_\_)$

wpif x := secret;

$PO(x:=0) \wedge wp(x:=0, \_\_)$

wpif x := 0;

$PO(z:=0)$

wpif z := 0;

true

Backwards analysis for concurrent programs

$$
\begin{aligned}
\mathit{wpif}_{\mathcal{R}\mathcal{G}}(c, Q) \mathrel{\widehat{=}}\ & \mathit{PO}(c) \wedge \mathit{wp}(c, Q) \wedge \\
& \color{magenta}{\mathit{guar}(c, \mathcal{G})} \wedge \\
& \color{green}{\mathit{stable}_{\mathcal{R}}}(\mathit{PO}(c) \wedge \mathit{guar}(c, \mathcal{G}) \wedge \mathit{wp}(c, Q))
\end{aligned}
$$

Backwards analysis for concurrent programs

$$
\begin{aligned}
wpif_{\mathcal{RG}}(c, Q) \mathrel{\widehat{=}}\ & PO(c) \wedge wp(c, Q) \wedge \qquad \text{---} \quad \color{blue}{pre(c)} \\
& \color{magenta}{guar(c, \mathcal{G})} \wedge \\
& \color{lime}{stable_{\mathcal{R}}}(PO(c) \wedge guar(c, \mathcal{G}) \wedge wp(c, Q))
\end{aligned}
$$

Backwards analysis for concurrent programs

$$wpif_{\mathcal{RG}}(c, Q) \mathrel{\widehat{=}} PO(c) \wedge wp(c, Q) \wedge \quad \text{—— } pre(c)$$
$$guar(c, \mathcal{G}) \wedge$$
$$stable_{\mathcal{R}}(PO(c) \wedge guar(c, \mathcal{G}) \wedge wp(c, Q))$$

P_0 ==> $pre(while) \wedge guar(\_) \wedge stable_{RG}(\_)$

$wpif_{RG}$

sync_write(dataType secret) :
while ($\neg$ CAS(z, 0, 1)) {
  while (z $\neq$ 0) {}
}

$pre(x{:=}secret) \wedge guar(\_) \wedge stable_{RG}(\_)$

$wpif_{RG}$ x := secret;

$pre(x{:=}0) \wedge guar(\_) \wedge stable_{RG}(\_)$

$wpif_{RG}$ x := 0;

$pre(z{:=}0) \wedge guar(\_) \wedge stable_{RG}(\_)$

$wpif_{RG}$ z := 0;

true

R

Conclusion

- general (rely/guarantee) approach to concurrency
- backwards analysis leads to simpler predicates (for IFS)
- based on standard verification techniques
- soundness proof in Isabelle/HOL
- automation:
  - theorem prover via tactics
  - custom-made analyser with interface to SMT solver
- analysis of assembly code