Gradual Security Types and Gradual Guarantees

Abhishek Bichhawat, **McKenna McCall,** Limin Jia

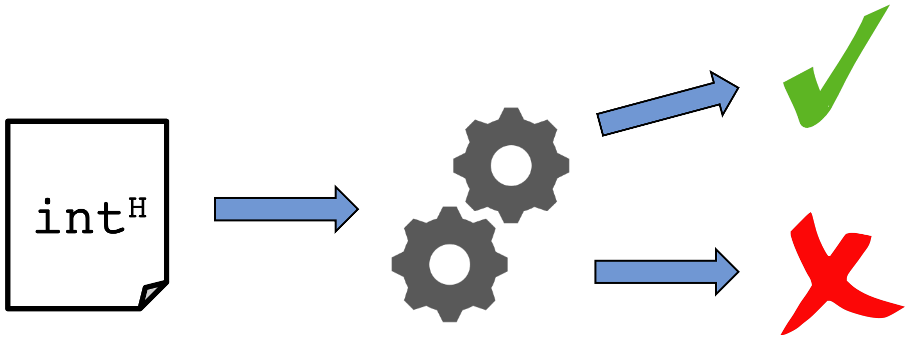
Carnegie Mellon University CyLab Security and Privacy Institute

Carnegie Mellon University

Electrical & Computer Engineering

Static Techniques

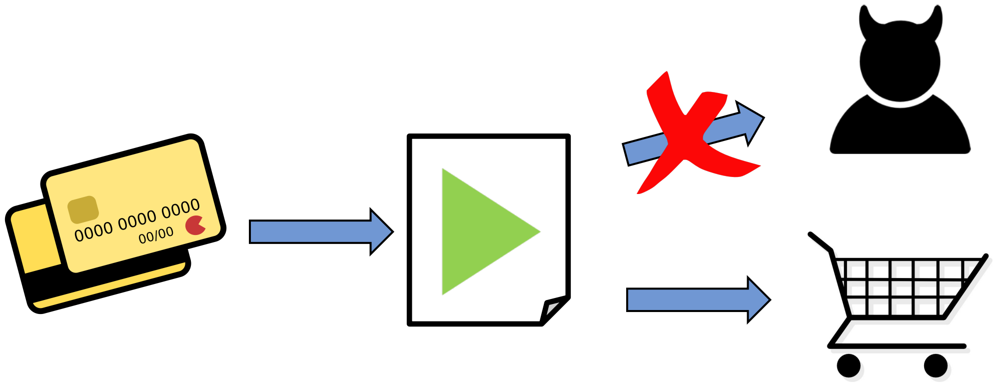
Well-typed code is secure



- ✓ Run at compile-time
- ✗ Require annotations, unsuitable for some languages

Dynamic Techniques

Suppress insecure behaviors



- ✗ Produce runtime errors
- ✓ Suitable for untyped/dynamically-typed languages

Do both with gradual typing!

What is Gradual Typing?

Dynamic label for untyped code or unknown labels

`location`

`x` `date`

Secret (H)

Public (L)

Dynamic (?)

What is Gradual Typing?

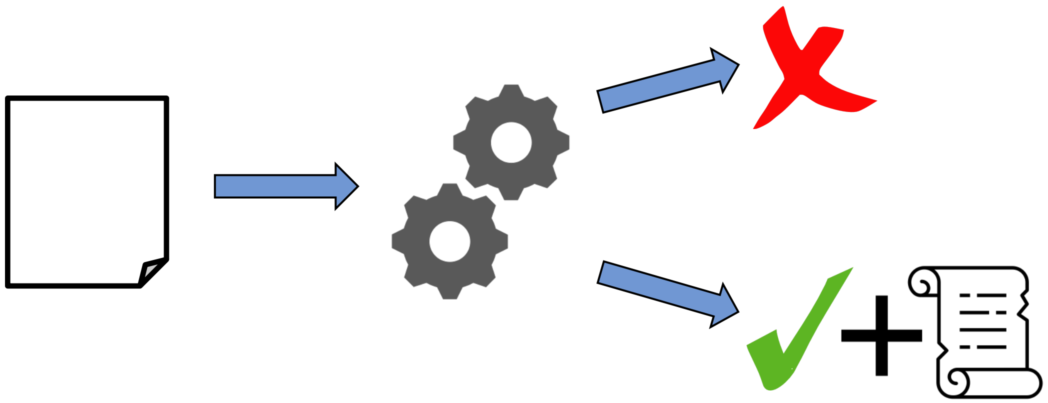
Static Typing
Check known labels

Dynamic Typing
Refine unknown labels

What is Gradual Typing?

Static Typing

Check known labels



$\texttt{true}^{?} \rightarrow (\texttt{[L,H] true})^{?}$

Security Lattice ($\mathcal{L}$)

H
|
L

Dynamic Label

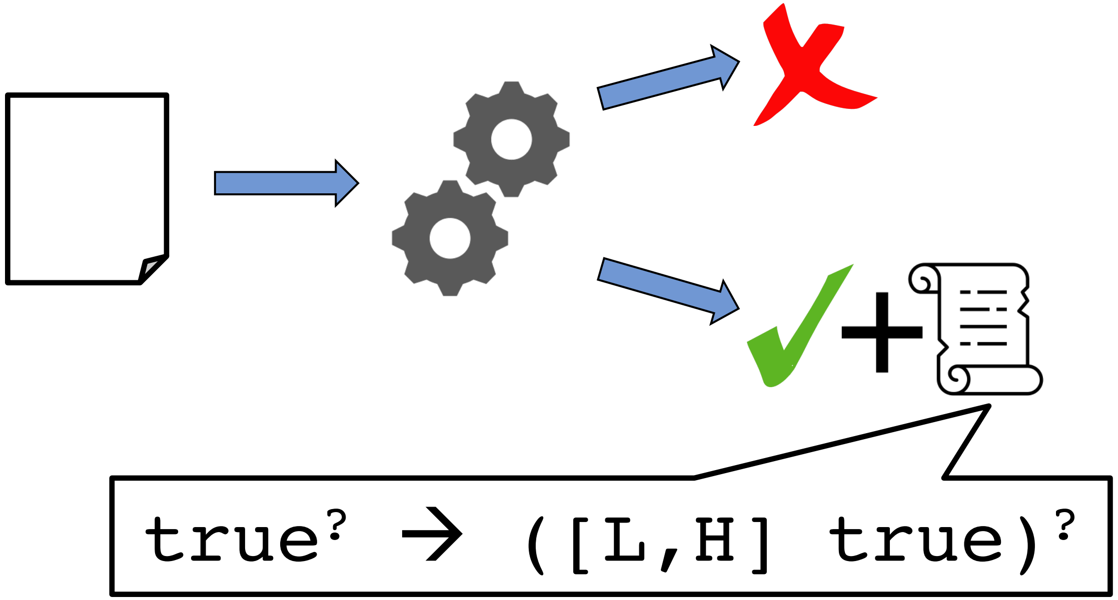
?

$$\forall l \in \mathcal{L},\ l \preccurlyeq_c ? \text{ and } ? \preccurlyeq_c l$$

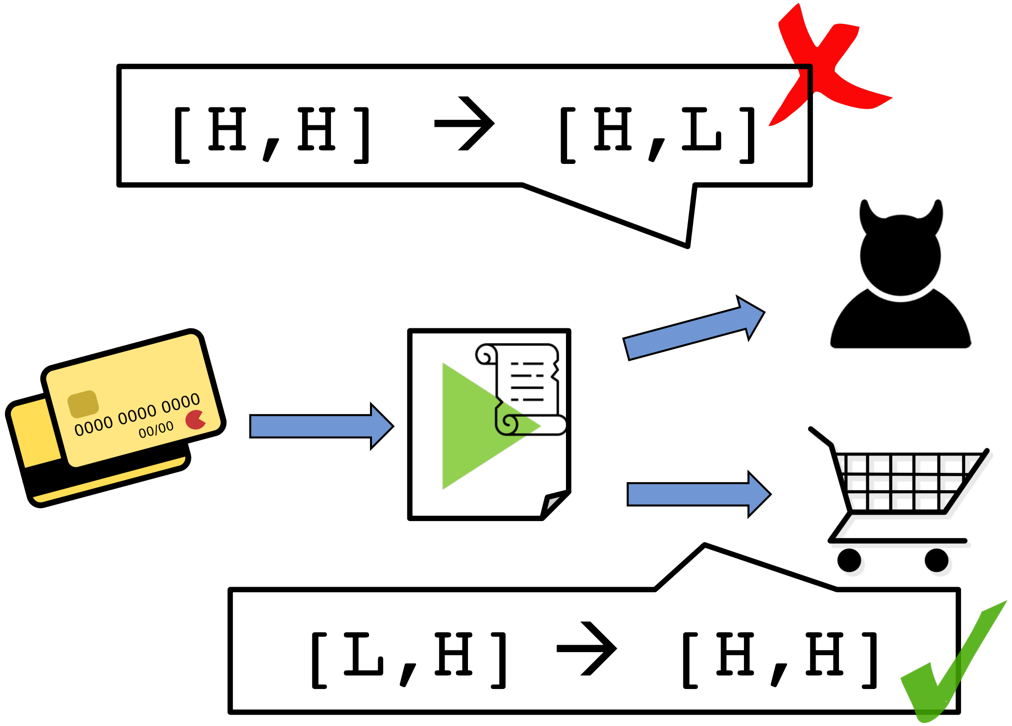
R. Garcia et al., “Abstracting gradual typing” POPL ‘16

What is Gradual Typing?

Static Typing
Check known labels



Dynamic Typing
Refine unknown labels



R. Garcia et al., "Abstracting gradual typing" POPL '16

Standard IFC type system

Secret

`x,y`

Public

`z`

```
y := trueᴴ
z := trueᴸ
if (x) then y := falseᴴ
output(L,z)
```

Always output `true` independent of `x` and `y`

Gradual IFC type system: Example

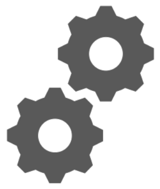
Secret	Public	?
`x`	`z`	`y`

```
y := true?
z := trueL
if (x) then y := false?
output(L,z)
```

Always output `true` independent of `x` and `y`

Gradual IFC type system: Example

Secret	Public	?
x	z	y

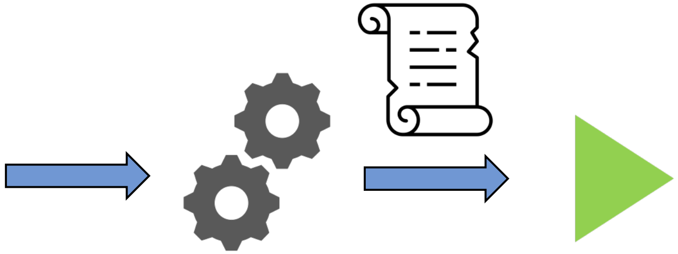


$\texttt{y := true}^{?}$
$\texttt{z := true}^{L}$
$\texttt{if (x) then y := false}^{?}$
$\texttt{output(L,z)}$

Assign $\texttt{bool}^{?}$ to $\texttt{y:bool}^{?}$ and $\texttt{bool}^{L}$ to $\texttt{z:bool}^{L}$

Flow from $\texttt{x:bool}^{H}$ to $\texttt{y:bool}^{?}$
$\texttt{H} \preccurlyeq_c ?$

Gradual IFC type system: Example



Secret	Public	?
`x`	`z`	`y`

```
y := true?
z := trueL
if (x) then y := false?
output(L,z)
```

Assign $\texttt{true}^{?}$ to $\texttt{y:bool}^{?}$

$[L,H] \rightarrow [L,H]$

Flow from $\texttt{x:bool}^{H}$ to $\texttt{y:bool}^{?}$

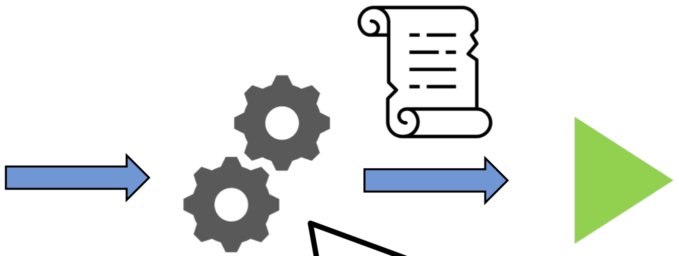
$\texttt{x} = [H,H]\ \texttt{true}^{H}$

$\texttt{y} = [L,H]\ \texttt{true}^{?}$

$[L,H] \rightarrow [H \curlyvee L, H] = [H,H]$

Gradual IFC type system: Example

Secret	Public	?
`x`	`z`	`y`



Assign $\texttt{y:bool}^{?}$ to $\texttt{z:bool}^{L}$

```
y := true?
z := trueL
if (x) then y := false?
z := y
output(L,z)
```

Flow from $\texttt{x:bool}^{H}$ to $\texttt{y:bool}^{?}$

`[L,H]` → `[H,H]`

Flow from $\texttt{y:bool}^{?}$ to $\texttt{z:bool}^{L}$

`[L,L]` → `[H,L]`

Gradual IFC type system: Gradual Guarantees

Secret	Public	?
x, y	z	

```
y := true^H
z := true^L
if (x) then y := false^H
output(L,z)
```

✓

Secret	Public	?
x	z	y

```
y := true^?
z := true^L
if (x) then y := false^?
output(L,z)
```

✓

Gradual guarantees:
? should not cause type-checking or runtime errors in secure programs

J. G. Siek et al., "Refined Criteria for Gradual Typing" SNAPL '15

Implicit flows are tricky!

Secret	Public	?
`x`	`z`	`y`

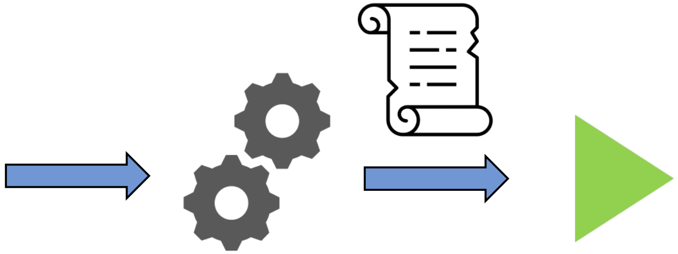
`y := true`$^{?}$
`z := true`^{L}
`if (x) then y := false`$^{?}$
`if (y) then z := false`^{L}
`output(L, z)`

When `x` is `true`:
`y := false`$^{?}$ and `z` remains `true`

When `x` is `false`:
`y` remains `true` and `z:=false`

Implicit flows are tricky!

Secret	Public	?
`x`	`z`	`y`



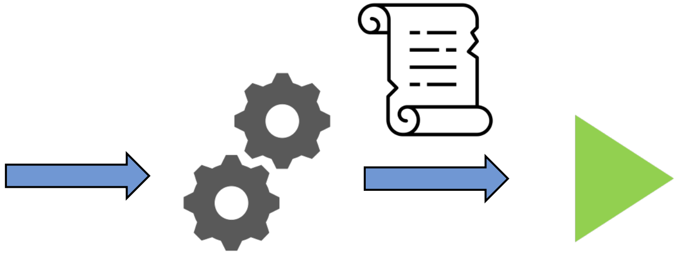
$\texttt{y := true}^{?}$
$\texttt{z := true}^{L}$
$\texttt{if (x) then y := false}^{?}$
$\texttt{if (y) then z := false}^{L}$
$\texttt{output(L, z)}$

Prior work: When `x` is `true`
Flow from $\texttt{x:bool}^{H}$ to $\texttt{y:bool}^{?}$
`[L,H]` → `[H,H]`

M. Toro et al., "Type-driven gradual security with references" TOPLAS '18

Implicit flows are tricky!

Secret	Public	?
`x`	`z`	`y`



```
y := true?
z := trueL
if (x) then y := false?
output(L,z)
```

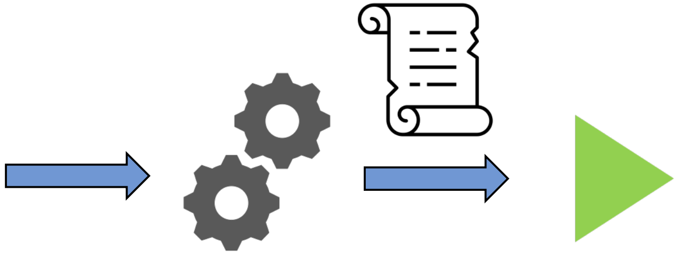
Prior work: When `x` is `true`
Flow from `x:bool`^{H} to `y:bool`$^{?}$
`[L,H]` → `[H,H]`

```
y := trueH
z := trueL
if (x) then y := falseH
output(L,z)
```

M. Toro et al., "Type-driven gradual security with references" TOPLAS '18

Implicit flows are tricky!

Secret	Public	?
x	z	y

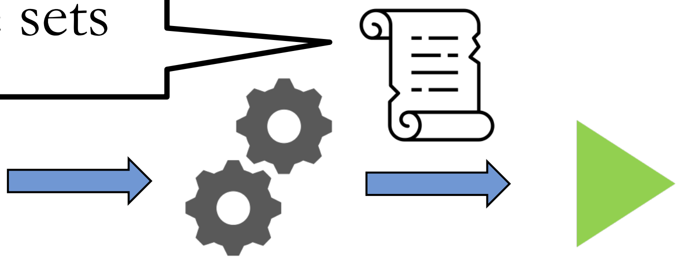


```
y := true^?
z := true^L
if (x) then y := false^?
if (y) then z := false^L
output(L, z)
```

Dynamic checker doesn't know what happens in the untaken branch, but static checker does!

Our approach

Evidence **and** write sets



Secret	Public	?
x	z	y

$\texttt{y := true}^{?}$
$\texttt{z := true}^{L}$
$\texttt{if}^{\{y\}}\ \texttt{(x) then y := false}^{?}$
$\texttt{if}^{\{z\}}\ \texttt{(y) then z := false}^{L}$
$\texttt{output(L, z)}$

Possible flow from $\texttt{x:bool}^{H}$ to $\texttt{y:bool}^{?}$
$\texttt{[L,H]} \rightarrow \texttt{[H,H]}$

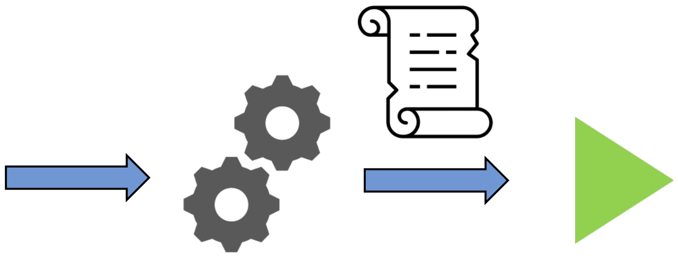
Possible flow from $\texttt{y:bool}^{?}$ to $\texttt{z:bool}^{L}$
$\texttt{pc:[H,H]}$

Our approach

Secret
x

Public
z

?
y



`y := true`$^{?}$
`z := true`^{L}
`if`$^{\{y\}}$ `(x) then y := false`$^{?}$
`output(L,z)`

Possible flow from `x:bool`^{H} to `y:bool`$^{?}$
`[L,H]` → `[H,H]`

Security

- **Satisfies termination-insensitive noninterference**
  - Terminating runs with the same public inputs produce the same public outputs

Theorem 1 (Noninterference) *Given a program,* c *and two stores* δ_1, δ_2 *s.t.* $\delta_1 \approx_L \delta_1$, $\Gamma \vdash c$, *and* $\forall i \in \{1, 2\}, \delta_i / c \xrightarrow{\tau_i}{}^{*} \delta'_i, \mathsf{skip}$ *then* $\tau_1 \approx_L \tau_2$

- **Proof uses paired execution technique**

F. Pottier and V. Simonet, "Information flow inference for ML" POPL '02

Paired execution

- **Pairs simulate multiple runs with different secrets**

Secret
```
x↦⟨true|false⟩
```

Public
```
z↦true
```

?
```
y↦false
```

- **Semantics are similar to faceted execution**

```
if{…} ⟨true|false⟩
  then c1
  else c2
…
```

$\Rightarrow$

```
⟨ c1 |c2 ⟩
…
```

F. Pottier and V. Simonet, "Information flow inference for ML" POPL '02

Gradual Guarantees

- **Satisfies gradual guarantees**
  - Adding ? does not cause type-checking or runtime errors in secure programs

Theorem 2 (Static Guarantee) *If* $\Gamma_1 \vdash c_1$, $\Gamma_1 \sqsubseteq \Gamma_2$, *and* $c_1 \sqsubseteq c_2$, *then* $\Gamma_2 \vdash c_2$.

Theorem 3 (Dynamic Guarantee) *If* $\delta_1 \ / \ c_1 \xrightarrow{\alpha_1} \delta_1' \ / \ c_1'$ *and* $\delta_1 \ / \ c_1 \sqsubseteq \delta_2 \ / \ c_2$, *then* $\delta_2 \ / \ c_2 \xrightarrow{\alpha_2} \delta_2' \ / \ c_2'$

Conclusion

- **Write set prevents implicit leaks without sacrificing gradual guarantees**
  - Satisfies termination-insensitive noninterference
  - Satisfies gradual guarantees

- **More details in paper including…**
  - Complete typing rules, monitor semantics
  - Label interval operations
  - More general security lattice
  - Language with references left to future work