Gradual Security Types and Gradual Guarantees

Abhishek Bichhawat, McKenna McCall, Limin Jia
**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

true? \Rightarrow ([L,H] \text{ true})?

Security Lattice ($\mathcal{L}$)

\[ H \]
\[ L \]

Dynamic Label

? ?

\[ \forall l \in \mathcal{L}, l \preceq_c ? \text{ and } ? \preceq_c l \]

R. Garcia et al., “Abstracting gradual typing” POPL ‘16
What is Gradual Typing?

Static Typing
Check known labels

Dynamic Typing
Refine unknown labels

true? \to ([L,H] \text{ true})?

R. Garcia et al., “Abstracting gradual typing” POPL ‘16
Standard IFC type system

Secret
\( x, y \)

Public
\( z \)

\[ y := \text{true}^H \]
\[ z := \text{true}^L \]
\[ \text{if (x) then } y := \text{false}^H \]
\[ \text{output}(L,z) \]

Always output \text{true} independent of \( x \) and \( y \)
Gradual IFC type system: Example

\[
\begin{align*}
y & := \text{true}^? \\
z & := \text{true}^L \\
\text{if (x) then } y & := \text{false}^? \\
\text{output(L,z)}
\end{align*}
\]

Always output \text{true} independent of \text{x} and \text{y}
Gradual IFC type system: Example

Secret $x$

Public $z$

? $y$

$y := \text{true}^?$

$z := \text{true}^L$

if (x) then $y := \text{false}^?$

output($L,z$)

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

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

$H \leq_c ?$
Gradual IFC type system: Example

Secret \( x \)  
Public \( z \)  
? \( y \)

\[ y := \text{true}^? \]
\[ z := \text{true}^L \]
if (x) then \( y := \text{false}^? \)
output(L,z)

Assign \( \text{true}^? \) to \( y: \text{bool}^? \)
\[ [L,H] \rightarrow [L,H] \]

Flow from \( x: \text{bool}^H \) to \( y: \text{bool}^? \)
\[ x = [H,H] \text{ true}^H \]
\[ y = [L,H] \text{ true}^? \]
\[ [L,H] \rightarrow [H \vee L,H] = [H,H] \]
Gradual IFC type system: Example

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

- **Flow from** $x: \text{bool}^H$ to $y: \text{bool}^?$
  - $[L,H] \rightarrow [H,H]$

- **Flow from** $y: \text{bool}^?$ to $z: \text{bool}^L$
  - $[L,L] \rightarrow [H,L]$

- **Assign** $y: \text{bool}^?$ to $z: \text{bool}^L$
Gradual IFC type system: Gradual Guarantees

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
\[ x \]
Public
\[ z \]
\[ ? \]
\[ y \]

When \( x \) is true:
\[ y := \text{false}? \] and \( z \) remains true

When \( x \) is false:
\[ y \] remains true and \( z := \text{false} \)

\[ y := \text{true}? \]
\[ z := \text{true}^L \]
\[ \text{if (x) then } y := \text{false}? \]
\[ \text{if (y) then } z := \text{false}^L \]
\[ \text{output}(L, z) \]
Implicit flows are tricky!

Secret x
Public z
?

Prior work: When \( x \) is \texttt{true}
Flow from \( x:\texttt{bool}^H \) to \( y:\texttt{bool}? \)
\[ [L,H] \rightarrow [H,H] \]

\[ y := \texttt{true}? \]
\[ z := \texttt{true}^L \]
if \((x)\) then \( y := \texttt{false}? \)
if \((y)\) then \( z := \texttt{false}^L \)
output\((L, z)\)
Implicit flows are tricky!

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

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

M. Toro et al., “Type-driven gradual security with references” TOPLAS ‘18
Implicit flows are tricky!

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

\[ y := \text{true} \]
\[ z := \text{true} \]

\[
\begin{align*}
\text{if } \{y\}(x) \text{ then } y & := \text{false} \\
\text{if } \{z\}(y) \text{ then } z & := \text{false}
\end{align*}
\]

output(L, z)

Possible flow from \(x: \text{bool}^H\) to \(y: \text{bool}\)
\([L, H] \rightarrow [H, H]\)

Possible flow from \(y: \text{bool}\) to \(z: \text{bool}^L\)
\(\text{pc: [H, H]}\)

Evidence and write sets
Our approach

\[
y := \text{true} \\
z := \text{true} \\
\text{if } \{y\} (x) \text{ then } y := \text{false} \\
\text{output}(L, z)
\]

Possible flow from \(x: \text{bool}^H\) to \(y: \text{bool}\)\quad[L,H] \rightarrow [H,H]\
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 $\delta_1, \delta_2$ s.t. $\delta_1 \approx_L \delta_1$, $\Gamma \vdash c$, and $\forall i \in \{1, 2\}$, $\delta_i/c \xrightarrow{\tau_i}$\*$\delta'_i$, 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

\[
\begin{align*}
\text{Secret} & \quad x \mapsto \langle \text{true} \mid \text{false} \rangle \\
\text{Public} & \quad z \mapsto \text{true} \\
? & \quad y \mapsto \text{false}
\end{align*}
\]

- Semantics are similar to faceted execution

\[
\text{if}\{\ldots\} \langle \text{true} \mid \text{false} \rangle \\
\text{then } c_1 \\
\text{else } c_2 \\
\ldots \\
\Rightarrow \\
\langle c_1 \mid c_2 \rangle \\
\ldots
\]

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 $\Gamma_1 \vdash c_1$, $\Gamma_1 \subseteq \Gamma_2$, and $c_1 \subseteq c_2$, then $\Gamma_2 \vdash c_2$.

**Theorem 3 (Dynamic Guarantee)** If $\delta_1 / c_1 \xrightarrow{\alpha_1} \delta'_1 / c'_1$ and $\delta_1 / c_1 \subseteq \delta_2 / c_2$, 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