Securing the SSA transform

C. Deng, S. Namjoshi, 2016, 2017

```
void foo ()
{
    int x;
    x = read_password();
    use (x);
    x = 0; // clear password
    other();
    return ;
}
```

```
void foo ()
{
    int x1, x2;
    x1 = read_password();
    use (x1);
    x2 = 0;
    other();
    return ;
}
```

Forma SSA

Trasformazione *corretta*, ma non **sicura**

Trasformazione unSSA

$$P \overset{\text{SSA}}{\rightarrow} Q_0 \rightarrow Q_1 \rightarrow \cdots \rightarrow Q_n \overset{\text{unSSA}}{\rightarrow} R$$

R è sicuro tanto quanto P

Leak

- Due tipi di variabili: H, L
- Dati due valori H-var a, b e una L-var c, leak se le computazioni
  - Producono output diverso
  - Entrambi terminano differiscono per un L-var nello stato finale
- (a, b, c) è una *leaky triple*

Taint proof

Taint environment E: Variabili $\longrightarrow$ Bool

Una coppia di stati $(s = (m, p), t = (n, q))$ soddisfano E ($(s, t) \models E$) se:

1. $m = n$
2. Per ogni variabile x, se $E(x)$ è falsa, allora $s(x) = t(x)$

Taint proof

Ordinamento: $E \sqsubseteq F$ definito da
$(\forall x : E(x) \implies F(x))$

Se $(s, t) \models E$ e $E \sqsubseteq F$, allora $(s, t) \models F$

Taint proof

Stati $s = (m, p)$, $s' = (n, q)$, statement S

Successione: $s \xrightarrow{S} s'$

$$\{E\}\ S\ \{F\}$$

Proof system

- skip: $\{E\}$ skip $\{F\}$
- out(e): $\{E\}$ out(e) $\{F\}$
- $x := e$

$$\frac{F(x) = E(e) \quad \forall y \neq x : F(y) = E(y)}{\{E\}\ x := e\ \{F\}}$$

- Sequenza

$$\frac{\{E\}\ S_1\ \{G\} \qquad \{G\}\ S_2\ \{F\}}{\{E\}\ S_1; S_2\ \{F\}}$$

Condizione

$$\frac{E(c) = \text{false} \quad \{E\}\ S_1\ \{F\} \qquad \{E\}\ S_2\ \{F\}}{\{E\}\ \text{if}\ c\ \text{then}\ S_1\ \text{else}\ S_2\ \{F\}}$$

$$\frac{E(c) = \text{true} \quad \{E\}\ S_1\ \{F\} \qquad \{E\}\ S_2\ \{F\} \qquad \forall x \in \text{Assign}(S_1) \cup \text{Assign}(S_2) : F(x)}{\{E\}\ \text{if}\ c\ \text{then}\ S_1\ \text{else}\ S_2\ \text{fi}\ \{F\}}$$

Ciclo while

$$\frac{E \sqsubseteq I \quad \{I\}\ \text{if } c \text{ then } S \text{ else skip fi}\ \{I\} \quad I \sqsubseteq F}{\{E\}\ \text{while } c \text{ do } S \text{ od}\ \{F\}}$$

Proprietà trasformazione

- Corretta: Q ha lo stesso output di P
- Sicura: Leaky triples $Q \subseteq$ Leaky triples P

unSSA

Il valore di x in P nel punto p corrisponde al valore di un x_k in Q_0 nel punto p.

Si raggruppano le versioni di x in un gruppo G e si rinominano a z_G

Vogliamo minimizzare i gruppi, ma è indecidibile.

Core set

Le variabili nel core set C_{i+1} di Q_{i+1} rispettano

$$\xi_i(t, s) \land \mathbf{final}_{i+1}(t) \land \xi_i(t', s') \land \mathbf{final}_{i+1}(t') \land s =_{C_i} s'$$
$$\implies t =_{C_{i+1}} t'$$

Ogni leak di C_n deriva da P

Partizionamento

Ogni programma Q_i ha un partizione delle variabili che rispetta:

1. Ogni gruppo ha un rappresentante core o untainted non-core
2. Le variabili nel gruppo sono mutualmente disgiunte (*interference-free*)
3. La definzione del rappresentante post-domina le altre variabili nel gruppo

unSSA

1. Costruire il core set C_n
2. Taint analysis su Q_n
3. Partizione delle variabili di Q_n su P1-P3
4. Rinominare le variabili dei gruppi individuati

Preservare P1-P3

ν_i relazione di simulazione tra Q_i e Q_{i+1}

μ associazione tra variabili di Q_i e Q_{i+1}

1. Se y è core o untainted non core, allora esiste x core o untainted non-core in Q_{i+1} tale che $\mu(x) = y$
2. La relazione ν_i tra Q_i e Q_{i+1} preserva definizioni e usi di variabili

Constant propagation

P

```
void foo ()
{
    int x;
    x = password();
    use (x);
    x = 0;
    other();
    return ;
}
```

Q_0

```
void foo ()
{
    int x;
    x = password();
    use (x);
    x = 0;
    other();
    return ;
}
```

Q_1

```
void foo ()
{
    int x;
    x = password();
    use (x);
    x = 0;
    other();
    return ;
}
```

R

```
void foo ()
{
    int x;
    x = password();
    use (x);
    x = 0;
    other();
    return ;
}
```

Constant propagation e folding

La trasformazione non agisce sul CFG

La taint proof rimane invariata

$x := e$ diventa $x := \bar{e}$ costante

$$\{E\}\ x := e\ \{F\} \implies$$
$$E(e) \sqsubseteq F(x) \wedge \forall y \neq x : E(y) \sqsubseteq F(y)$$

$$E(\bar{e}) \sqsubseteq E(e) \implies E(\bar{e}) \sqsubseteq F(x)$$

Loop unrolling

P

```
void foo()
{
  int i, x;
  x = 0;
  for(i = 0; i < 2*K ; i++)
  {
    x = f (x, i);
  }
  return;
}
```

Q_0

```
void foo ()
{
  int i1, i2, i3,
      x1, x2, x3;
  x1 = 0;
  i1 = 0;
  loop :
    x2 = phi (x1, x3) ||
    i2 = phi (i1, i3);
    if(i2 >= 2*K) goto end;
    x3 = f(x2, i2);
    i3 = i2 + 1;
  goto loop;
  end:
    return;
}
```

Loop unrolling

Q_1

```
void foo ()
{
  int i1, i2, i3, i4, i5,
      x1, x2, x3, x4, x5;
  x1 = 0;
  i1 = 0;
  loop:
    x2 = phi(x1, x5) ||
    i2 = phi(i1, i5);
    if(i2 >= 2*K) goto end;
    x3 = f(x2, i2);
    i3 = i2 + 1;
    x4 = x3 || // phi
    i4 = i3;
    skip; // test
    x5 = f(x4, i4);
    i5 = i4 + 1;
    goto loop ;
  end:
    return;
}
```

R

```
void foo ()
{
  int i1, i2, i3,
      x1, x2, x3;
  x1 = 0;
  i1 = 0;
  loop :
    x2 = phi (x1, x3) ||
    i2 = phi (i1, i3);
    if(i2 >= 2*K) goto end;
    x3 = f(x2, i2);
    i3 = i2 + 1;
  goto loop;
  end:
    return;
}
```

Loop unrolling

La trasformazione aggiunge variabili nel corpo del ciclo

L'invariante I è estesa:

- $I'(x) = I(x)$ per x originale
- $I'(x') = I(x)$ per x' copia di x

μ si comporta nello stesso modo

Secure DSE

DSE conforme a P1 - P3

Elimina $x := e$ ($x \in Y_k$) se:

- $Y_k = \{x\}$
- x non è rappresentante del gruppo

x è dead in Q_i e untainted in Q_{i+1}

x non contamina altre variabili e rimane untainted

Se $Y_k ! = \{x\}$ e x non rappresentante definiamo

$$Y_k' = Y_k \setminus \{x\}, \quad Y_k'' = \{x\}$$

Secure DSE

P

```
void foo()
{
    int x;
    x = read_password();
    use(x);
    x = 0;
    x = read_password2();
    return;
}
```

R

```
void foo()
{
  int x;
  x1 = read_password();
  use (x1);
  x2 = 0;
  x3 = read_password2();
  return;
}
```

Riferimenti

Deng, C., Namjoshi, K.S. (2017).
Securing the SSA Transform

Deng, C., Namjoshi, K.S. (2016).
Securing a compiler transformation