Techniki zabezpieczania kodu – analiza przepływu informacji

Temat V

Bezpieczeństwo w systemie operacyjnym

1. system operacyjny kontroluje dostęp do zasobów,
2. zmieniają się potrzeby,
3. ewolucja API jądra powolna,
4. ewolucja API powolna – potrzeba standaryzacji,
5. nieodpowiednie pojęcie aktora

Potrzeby bezpieczeństwa zależne od aplikacji

- GUI – dostęp do okien,
- kod mobilny – nie wysyłamy przez sieć po przeczytaniu pliku z numerem konta,
- e-zakupy – towaru nie wysyłamy/pokazujemy przed zapłaceniem,
- zarządzanie prawami intelektualnymi.

Problemy z zasadą minimalnych przywilejów

- przywileje w systemie operacyjnym na poziomie użytkownika-grupy,
- różne wątki tej samej aplikacji mogą wymagać różnych praw,

Wielkość Zaufanej Bazy Kodowej (TCB)

- system operacyjny – duża baza kodowa,
  trudno testować, debugować,
- kompilator – mała baza kodowa
  łatwiejsze testowanie, debugowanie,

Techniki oparte na językach programowania

- analiza programów na etapie kompilacji/ładowania
- sprawdzanie analizy na etapie ładowania redukuje TCB
- transformacja programów na etapie kompilacji/ładowania/działania tak, żeby nie mogły naruszyć bezpieczeństwa; albo żeby logowały akcje

Założenia dotyczące bezpieczeństwa

- atakujący nie ma fizycznego dostępu do sprzętu,
- kod programu nie może być (semantycznie) zmodyfikowany w trakcie wykonania,
- nikt nie monitoruje aktywności radiowej komputera
- itp.

Problemy z założeniami

- założenia to zagrożenia (czasami znane, a czasami nie),
- analiza bezpieczeństwa możliwa tylko po abstrakcji,
- abstrakcja ukrywa szczegóły (być może nieistotne)

Analiza przepływu informacji

- sprawdzamy, jak informacje wędrują poprzez program

Model La Paduli–Bella

- informacje są poklasyfikowane np.
  - ściśle tajne,
  - tajne specjalnego przeznaczenia,
  - tajne,
  - poufne,
  - publiczne
- w programie zachowanie na niskim poziomie tajności nie powinno zależeć od danych na wysokim poziomie tajności

Typy związane z modelem LP-B

- dodajemy etykiety do języka jako adnotacje typowe,
- najprostszy język: H – tajne, L – publiczne;
  - $L \rightarrow H$ – OK
  - $H \rightarrow L$ – źle

```
int{H} x; // tajna liczba całkowita
int{L} y; // publiczna liczba całkowita
String{L} z; // publiczny napis
x = y; // OK
y = x; // źle
x = z.size(); // OK
z = Integer.toString(x); // źle
```

Własności bezpieczeństwa jako kraty

- zasady przepływu informacji dobrze reprezentują się jako kraty,
  - nie deptać trawników A,
  - nie palić w miejscach publicznych B,
  - kombinacje: $A \sqcup B$, $A \sqcap B$

Własności bezpieczeństwa – jak zapewniać?

- łączenie wartości z różnych zasad przepływu informacji
- podejście konserwatywne:
  etykieta wyniku powinna być co najmniej tak silna jak etykiety wszystkich danych wejściowych
- jeśli $\underline{x}$ to etykieta x, a $\underline{y}$ to etykieta y, to etykieta $x + y$ to $\underline{x} \sqcup \underline{y}$.

Przepływy niejawne

```
x = 0;
if (b) {
  x = a;
}
```

- wartość końcowa x wskazuje na wartości a i b
- przyjmujemy: $a \leq x \;\&\&\; b \leq x$

Przypisanie i licznik instrukcji

- pomysł: niejawne przepływy ujmowane w liczniku instrukcji ($\underline{pc}$)
- instrukcje `if`, `while`, `switch` podnoszą $\underline{pc}$

```
x = 0;
if (b) {
  x = a;        // _pc_ = _b_
}
```

- statyczne sprawdzanie to już nie tylko optymalizacja (co się stanie, gdy powyższe jest sprawdzone w czasie wykonania)?

Jif: Java + Information Flow

- programy w Javie są anotowane etykietami
- zmienne mają typ oraz etykietę przepływu
- etykiety to zasady wyrażone w terminach zleceniodawców (aktorów):

```
int {Ala->Bartek} x;
```

- etykiety wyznaczają jak przepływa informacja

Jif – etykiety

- zleceniodawcy pozwalają na symulowanie grup (gromadzących wielu aktorów) i roli (w ramach jednego aktora),
- podstawowa relacja: q *acts-for* p, $q \geq p$,
  reprezentuje zaufanie
- *acts-for* jest zwrotna i przechodnia
- zleceniodawca $\top$
- zleceniodawca $\bot$
- zleceniodawca $p \& q$ oraz p, q

Jif – zasady dotyczące odczytów

- Zapis $o \rightarrow r$ (o - właściciel, r - czytacz)
- o pozwala q na odczyt, gdy q *acts-for* o lub r, a także
- p uważa, że zasada c powinna ograniczać jego czytaczy, gdy właściciel c może *act-for* p:
- $\mathsf{readers}(p, o \rightarrow r) \triangleq \{q \mid$ jeśli $o \geq p$ then $(q \geq o$ lub $q \geq r)\}$
- złożone zasady: $c \sqcup d$, $c \sqcap d$
- $\sqcup$ zdefiniowane jako przecięcie zbiorów readers
- $\sqcap$ zdefiniowane jako suma zbiorów readers

Jif – zasady dotyczące zapisów

- Zapis $o \leftarrow w$ (o - właściciel, w - pisarz)
- o pozwala q może wpłynąć na wartość informacji, gdy q *acts-for* o lub w, a także
- p uważa, że zasada c powinna ograniczać jego zapisy, gdy właściciel c może *act-for* p:
- $\mathsf{writers}(p, o \rightarrow w) \triangleq \{q \mid$ jeśli $o \geq p$ then $(q \geq o$ lub $q \geq w)\}$
- złożone zasady: $c \sqcup d$, $c \sqcap d$
- $\sqcup$ zdefiniowane jako suma zbiorów writers
- $\sqcap$ zdefiniowane jako przecięcie zbiorów writers

Jif – etykiety

- $\{c; d\}$
  - c to zasada dotycząca odczytów
  - d to zasada dotycząca zapisów

Jif – przykład

- $\{\mathsf{Ala} \rightarrow \mathsf{Bartek}, \mathsf{Czesiek}; \mathsf{Ala} \leftarrow \mathsf{Czesiek} \sqcup \mathsf{Bartek} \leftarrow \mathsf{Czesiek}, \mathsf{Darek}\}$
- Czytanie (I): zleceniodawcy, których roli Ala nie może grać pozwalają wszystkim czytać,
- Czytanie (II): zleceniodawcy, których rolę może grać Ala są ograniczeni i odczyty są dozwolone tylko dla zleceniodawców, którzy mogą grać rolę Ali, Bartka lub Cześka

Jif – przykład c.d.

- $\{\mathsf{Ala} \rightarrow \mathsf{Bartek}, \mathsf{Czesiek}; \mathsf{Ala} \leftarrow \mathsf{Czesiek} \sqcup \mathsf{Bartek} \leftarrow \mathsf{Czesiek}, \mathsf{Darek}\}$
- Zapis (I): Ala wierzy, że tylko zleceniodawcy, którzy mogą grać rolę Czarka lub grać rolę Ali mogli wpływać na informację
- Zapis (II): Bartek wierzy, że tylko zleceniodawcy, którzy mogą grać rolę Czarka, Darka lub grać rolę Bartka mogli wpływać na informację
- Zapis (III): ktoś, kto gra Alę i Bartka wierzy, że tylko zleceniodawcy, którzy mogą grać rolę Ali, Bartka, Czarka lub Darka mogli wpływać na informację
- Zapis (IV): dla pozostałych na informację mógł wpływać ktokolwiek, więc nie jest ona godna zaufania

Jif – dodatkowe możliwości

- parametry etykiet,
- etykiety dynamiczne,
- polimorficzne etykiety argumentów

Jif – adnotacje w kodzie

```
public class Vector[label L] extends AbstractList[L] {
    private int{L} length;
    private Object{L}[]{L} elements;

    public Vector() ...
    public Object elementAt(int i):{L; i}
                    throws IndexOutOfBoundsException {
        ...
        return elements[i];
    }
    public void setElementAt{L}(Object{L} o, int{L} i) ...
    public int{L} size() { return length; }
    public void clear{L}() ...
    ...
}
```

Jif – adnotacje w kodzie, klasy

```
public class Vector[principal P, label L]
                    extends AbstractList[L] {
  ...
}
```

- klasy mogą być parametryzowane zleceniodawcami i etykietami
- parametryzacja jest polimorficzna
- deklaracje obiektów:

```
final Vector[Alicja, lb] vec =
                    new Vector[Alicja, lb]();!
```

- inne operacje...

Jif – adnotacje w kodzie, pola obiektów

```
private int{L} length;
private Object{L}[]{M} elements;
```

- można opatrzeć zmienne etykietami
- można określić etykietę elementów tablicy (L)
- można określić etykietę tablicy (M)

Jif – adnotacje w kodzie, metody

```
public void setElementAt{L}(Object{L} o,
                            int{L} i) ...
```

- etykieta początkowa: w `setElementAt` – L
  - wywołanie w miejscu nie bardziej restrykcyjnym niż L
  - modyfikacje co najmniej tak restrykcyjne jak L
- L jest też górnym ograniczeniem na argumenty faktyczne (o, i)

Jif – adnotacje w kodzie, metody c.d.

```
public Object elementAt(int i):{L; i}
                throws IndexOutOfBoundsException {
    ...
    return elements[i];
}
```

- po wyjściu znajdujemy się w miejscu co najmniej tak restrykcyjnym jak `{L; i}`
- etykieta `i` oznacza etykietę zmiennej i
- o etykiecie wyjściowej decyduje informacja podawana w return oraz wyjątkach

Jif – adnotacje w kodzie, autorytet

```
class C authority(Alicja) {...
void m() where authority(P);
void n() where caller(Q, Alicja);
```

- klasa może deklarować, że wykonuje się w imieniu jakiegoś zleceniodawcy, np. Alicja
- także pojedyncza metoda może deklarować, w imieniu kogo się wykonuje
- można też deklarować, że można ją wykonywać tylko z kodu o określonych pełnomocnictwach

Jif – adnotacje w kodzie, zmiana poziomu

```
declassify(e, L1 to L2);
endorse(e, L1 to L2);
declassify(L1 to L2) s;
endorse(L1 to L2) s;
```

- declassify – można odczytywać bardziej tajne rzeczy,
- endorse – można zapisywać bardziej jawne rzeczy

Jif – adnotacje w kodzie, różne

- można stosować dynamiczne etykiety – cały system deklaracji,
- można specyfikować biblioteki zewnętrzne