Security-Typed Langages

Hannu-Daniel Goiss - 0302800
Seminar aus Programmiersprachen – WS 2007/2008
Technische Universität Wien
Institut für Computersprachen – Programmiersprachen und Übersetzer

Kurzfassung

Jif und Flow Caml sind die einzigen Security-Typed Languages, die zur Zeit zur Verfügung stehen. Sie bieten die Möglichkeit mittels Annotationen festzulegen, wer auf gewisse Daten zugreifen darf und stellen sicher, dass auch nur die Zugriffsberechtigten zugreifen können.

1 Einleitung

Es wird immer wichtiger sicherzustellen, dass unbefugte Personen keinen Zugriff auf sensible Daten bekommen. In allen Wirtschaftszweigen werden heutzutage IT-Systeme für alle möglichen Aufgaben verwendet, zum Beispiel werden unter anderem auch persönliche Daten von Bankkunden gespeichert. Dadurch resultiert, das es ein enormes Interesse daran gibt, das die Systeme sicherer werden und die Möglichkeiten Datenmissbrauch zu begehen eingeschränkt werden. Wünschenswert wären Systeme, mit denen man Sicherheitsvorschriften formulieren kann und die dann auch sicherstellen, das diese Sicherheitsvorschriften immer umgesetzt werden. Programmiersprachen, mit denen so etwas möglich ist sind Jif [3] und Flow Caml[4]. Jif und Flow Caml sind die einzigen Security-Typed Languages.[1][2] Bei Security-Typed Languages gibt man mittels Annotationen an, wer auf welche Variablen Schreib- bzw. Lesezugriff hat. Leider sind Security-Typed Languges nicht weit verbreitet. Es wurden bisher keine nennenswerten Applikationen mit ihnen entwickelt. Die einzigen Möglichkeiten, die es sonst gibt um sicherzustellen, das Sicherheitsvorschriften in Programmen eingehalten werden, sind „gutes Design“ und „Implementationspraktiken“.[1] Dies ist jedoch nicht ideal, weil man diese Eigenschaften nicht automatisch überprüfen kann, Security-Typed Languages stellen jedoch sicher, dass die Sicherheitsvorschriften eingehalten werden.

2 Security-Typed Languages

Security-Typed Languges sind Programmiersprachen, bei denen Annotationen verwendet werden um Variablen zu markieren.[1] Diese Markierungen sagen aus, wer Zugriff auf die annotierten Variablen hat. In den folgenden Kapiteln werden wir einen Überblick die Features von Jif und Flow Caml erhalten und versuchen einen Vergleich zwischen den beiden Programmiersprachen zu erstellen.

2.1 Jif

Jif ist eine Erweiterung von Java. Die von [1] beschriebene Version ist Jif 3.0, erschienen im Juni 2006. Die aktuellste Version ist Jif 3.1.1, erschienen am 1.11.2007.

2.1.1 Principals

Als Principals werden die „Namen“ bezeichnet, die man als Annotation zu den Variablen dazuschreibt. Es ist hierbei zu beachten, dass es in der in [1] beschriebenen Version keine Hierarchie von Prinzipals gibt. Wenn man also den Wert einer Variablen, die den Principal bob hat in eine Variable schreiben will, die den Principal alice hat, dann ergibt das einen Error. In der aktuellsten Version gibt es jedoch bereits eine Principalhierarchie und auch Principal Polymorphismus.[5]

2.1.2 Annotationen

Annotationen werden nach dem DLM (decentralized label model) angeführt. Ein Bespiel hierfür ist:

```
String{alice:} password = "1fth2;zg";
```

Password bekommt hier den Principal alice. Wenn man nun versucht diese Variable einer anderen Variable zuzuweisen, die einen anderen Principal hat, ergibt das einen Error.

```
String{bob:} leak = password;
```

Ein weiteres Beispiel, wie man Annotationen angeben kann ist:

```
public class Email {
   String{} toAddress;
   String{} fromAddress;
   String{this} body;
   public Email(String{} to, String{} from, String{this}
body) { ... }
}
```

{} bedeutet, das es keine Zugriffsbeschränkung für diese Variable gibt. Jeder kann also darauf zugreifen. {this} bedeutet, dass der Principal der Klasse auch für diese Variable gibt. Außerdem kann man Principals auch bei den Parametern von Methoden angeben.
Der folgende Methodenaufruf wäre ein gültiger Methodenaufruf:

```
Email{bob:} msgToBob = new
Email("bob@psu.edu","alice@psu.edu","Hi Bob!");
```

Der folgende Methodenaufruf ware nicht gültig, weil der Principal von msgToBob und password nicht übereinstimmen:

```
Email{bob:} msgToBob = new
Email("bob@psu.edu","alice@psu.edu",password);
```

In Jif 3.0 kann man nur einen Principal angeben. Es ist nicht möglich beispielsweise eine Readers- und eine Writerslist anzugeben, obwohl dies eigentlich im DLM vorgesehen ist.[1] In der aktuellsten Version von Jif

wird dies bereits unterstützt. Ein Beispiel, wie man so etwas angeben könnte ist:

```
String {alice:bob} text;
```

Wobei der Principal bob nur Lesezugriff auf text hätte.

2.1.3 De-classify Mechanismen

Es ist wichtig zu ermöglichen, dass man einen Principal einer Variable auch irgendwie ändern kann. Dafür wird die Methode declassify zur Verfügung gestellt.

```
declassify(AES.encrypt(key,msgToBob), {});
```

Hier wird msgToBob deklassifiziert und hat danach den Principal {}. Bei der Deklassifizierung wird auch ein kryptographisches Verfahren verwendet. Die Stellen im Code, an denen deklassifiziert wird, sind die einzigen kritischen Stellen. Bei einer Überprüfung des Codes muss somit nur auf diese Stellen geachtet werden. Bei Jif kann jeder Principal in jeden anderen umklassifiziert werden. In [1] wird ein Mechanismus vorgestellt, mit dem sichergestellt wird, das nur die Deklassifizierungen vorgenommen werden können, die auch zuvor in einem Policy-File definiert worden sind.

2.1.4 Standardeingabe

Die Standardeingabe muss natürlich auch mit einem Principal versehen werden können. Dafür wird bei der main-Methode ein eigener Parameter angegeben, der angibt welchen Principal die Standardeingabe hat.

```
static void main(Principal user, String[] args)
```

2.2 Flow Caml

Flow Caml ist eine Erweiterung von Objective Caml.[2] Es bietet genauso wie Jif die Möglichkeit den Code mit Annotationen zu versehen. Flow Caml ist jedoch kein gültiger Objective Caml Code und kann somit nicht von einem Objective Caml Compiler kompiliert werden.
Der Flow Caml Compiler kompiliert Flow Caml Code und erzeugt daraus gültigen Objective Caml Code, der von jedem Objective Caml Compiler kompiliert werden kann. Somit kann Flow Caml auf jedem System ausgeführt werden, für das es auch einen Objective Caml Compiler gibt.

2.2.1 Principals

Genauso wie bei Jif gibt es auch in Flow Caml Principals.
Das folgende Beispiel illustriert, wie man Principals definiert:

```
let x1 : !alice int = 42;;
val x1 : !alice int
let x2 : !bob int = 53;;
val x2 : !bob int
let x3 : !charlie int = 11;;
val x3 : !charlie int
```

Wenn man nun Operationen auf Variablen durchführt, die verschiedene Principals haben, muss das Ergebnis beide Principals „erfüllen“.

```
x1 + x2;;
x1 * x2 * x3;;
```

Man kann auch Listen und alle anderen Datentypen mit Principals versehen. Ein Beispiel hierfür:

```
let l1 = [1; 2; 3; 4];;
val l1 : ('a int, 'b) list
let l2 = [x1; x2];;
val l2 : ([> !alice; !bob] int, 'b) list
```

2.2.2 Imperative Ausführung

Flow Caml bietet auch die Möglichkeit imperativen Code auszuführen.
Ein Beispiel hierfür ist:

```
r := not y
r := if y then false else true
if y then r := false else r := true
r := true; if y then r := false
```

Außerdem gibt es natürlich auch die Möglichkeit Schleifen auszuführen, wie in dem folgenden Beispiel:

```
let length' list =
  let counter = ref 0 in
  let rec loop = function
       [] -> ()
    | _ :: tl ->
      incr counter;
      loop tl
  in
  loop list;
  !counter
;;
val length' : ('a, 'b) list -{'b ||}-> 'b int
```

2.2.3 De-classify Mechanismen

Bei Jif gibt die Methode declassify, mit der man den Principal einer Variable ändern kann. Flow Caml bietet hier andere Möglichkeiten. Und zwar kann man mit dem Befehl flow eine Principalhierarchie definieren, die es dann auch ermöglicht von Variablen mit einem Principal einen Wert einer anderen Variable mit einem anderen Principal zuzuweisen. Ein Beispiel hierfür wäre:

```
flow !alice < !stdout;;
```

Hier wird definiert, das !alice in der Hierarchie unter !stdout steht. Das bewirt, das man von einer Variablen mit dem Principal !alice einen Wert an die Variable mit dem Principal !stdout zuweisen kann.

2.2.4 Standardeingabe

Bei Flow Caml gibt es für die Standardeingabe und die Standardausgabe vordefinierte Principals. Diese sind !stdin und !stdout. Um nun Werte ein oder ausgeben zu können müssen sie in der Hierarchie der Principals entsprechend zu diesen vordefinierten Werten gestellt sein. Ein Beispiel für eine Standardausgabe wäre:

```
print_int x1;;
```

2.3 Zusammenfassung - Vergleich Jif/Flow Caml

Meiner Meinung nach liegt der Hauptunterschied zwischen Jif und Flow Caml darin, das die beiden Programmiersprachen auf vollkommen unterschiedlichen Programmiersprachen basieren. Jif basiert auf Java und Flow Caml auf Objective Caml, was eine eher funktionale Programmiersprache ist. Die Entscheidung, ob nun Jif oder Flow Caml verwendet wird liegt nun möglicherweise daran, welche Programmiersprache einem selber sympathischer ist.

Außerdem ist erwähnenswert, dass Jif nicht mit einem normalen Java-Compiler ausführbar ist. Jif muss von einem Jif Compiler aus ausgeführt werden und kann nicht direkt mit normalen Java-Programmen kommunizieren.
Flow Caml hingegen wird vom Flow Caml Compiler in Objective Caml übersetzt und ist von jedem Objective Caml Compiler ausführbar.

3 zukünftige Entwicklungen

Folgende Entwicklungen sind für die Zukunft erwünschenswert:

- Jif
  - Debugging: Auch in der aktuellsten Version von Jif gibt es noch immer keine Möglichkeit zu debuggen.
  - Readers List wie in der DLM: Wurde in [1] vorgeschlagen und ist mittlerweile in der aktuellsten Version von Jif verfügbar.
  - Implementationen für Jif von Sockets, Network Stacks, File Systems, usw.
- Flow Caml
  - Laut [2] ist es vorgesehen Flow Caml in Zukunft so zu erweitern, dass man als Principals angeben kann, sondern auch Variablen und Datentypen angeben kann und Sicherheitsstufen (security levels) zu ermöglichen.

4 Quellen

[1] Boniface Hicks, Kiyan Ahmadizadeh, and Patrick McDaniel.
Understanding practical application development in security-typed languages. In 22nd Annual Computer Security Conference. ACM, Dezember 2006

[2] Vincent Simonet. Flow Caml in a Nutshell. In Graham Hutton, editor, Proceedings of the first APPSEM-II workshop, Seiten 152--165, März 2003.

[3] http://www.cs.cornell.edu/jif/, zugegriffen am 6.12.2007

[4] http://cristal.inria.fr/~simonet/soft/flowcaml/, zugegriffen am 6.12.2007

[5]http://www.cs.cornell.edu/jif/doc/jif-3.1.1/manual.html, zugegriffen am 6.12.2007