渐进信息流定型

摘要

我们提出了一种方法，通过形式化方法来对简单带类型λ演算的扩展，从而支持安全代码的渐进演化，该扩展提供了信息流构造。这些构造允许最初不安全的程序进行有针对性的重构演化，并通过强制转换提供动态信息流保证，同时通过标记类型提供静态信息流安全。

关键词：类型系统；渐进类型；信息流

1 引言

中国在2000年以来，信息技术和互联网行业得到了快速发展，这既带来了大量的程序开发和使用，也带来了严重的安全威胁，如网络攻击、数据泄露等问题。随着网络犯罪的不断升级和演变，程序安全变得尤为重要。黑客、病毒、勒索软件等威胁层出不穷，这促使了更加安全的编程实践和工具的需求。随着安全问题的不断曝光，企业和个人对程序安全的重视程度不断增加。越来越多的人开始关注如何编写安全的代码，以保护他们的应用和数据。

各个大学和研究机构积极参与安全领域的研究，与产业界合作开展安全相关的项目。这种合作有助于将研究成果转化为实际的安全编程工具和实践。

本次课程的论文复现工作拟通过实现一门带渐进式信息流安全类型系统的函数式编程语言来保证程序的信息流安全。

2 相关工作

许多先前的工作都解决了信息流安全性的问题。其中大部分工作集中在静态类型系统上，如 JFlow [10]、Jif [1] 和其他一些系统 [13] [11]，这些系统需要显著的前期投入。更近期的研究探讨了动态信息流 [3] [4] [5]，它需要较少的前期投资，但不能将安全属性以类型的形式记录下来。

在原论文中，作者探讨了朝着将无类型脚本逐渐演化为安全类型应用的愿景迈出的一步。由于先前的工作已经解决了将动态脚本演变为类型化代码的问题 [2] [12] [9] [8]，原论文的开发起点是一种具有类型的语言。作者研究了如何逐渐在这种语言中扩展程序，以提供安全保证和安全类型。

3 本文方法

3.1 本文方法概述

本文是一篇纯理论的论文，采用形式化方法，严谨的描述了一门编程语言的语法定义（图1）、操作语义（图2）和类型系统（图3）。

$$
\begin{array}{lr}
\iota ::= \texttt{Int} \mid \texttt{Bool} \mid \texttt{Str} & \textit{Base Types} \\
a, b ::= \iota \mid A \to B & \textit{Raw Types} \\
A, B ::= a^k & \textit{Labeled Types} \\
t, s ::= v \mid x \mid t\ s \mid op\ \bar{t} \mid t \colon A \Rrightarrow B \mid t \colon A \Rightarrow^p B & \textit{Terms} \\
r ::= c \mid \lambda x \colon A.\ t & \textit{Raw Values} \\
v, w ::= r^k & \textit{Labeled Values} \\
k, l, m & \textit{Labels} \\
\Gamma ::= \emptyset \mid \Gamma, x \colon A & \textit{Typing Environment}
\end{array}
$$

图 1. 语法

$$\boxed{t \Downarrow v}$$

$$\frac{}{v \Downarrow v}\ [\text{E-VALUE}] \qquad \frac{t \Downarrow (\lambda x \colon A.\ t_1)^k \quad s \Downarrow v \quad t_1[x := v] \Downarrow r^m}{t\ s \Downarrow r^{m \sqcup k}}\ [\text{E-APP}]$$

$$\frac{r = \delta_{op}(r_1, \cdots, r_n) \quad t_i \Downarrow r_i^{k_i} \quad k = \sqcup k_i}{op\ \bar{t} \Downarrow r^k}\ [\text{E-PRIM}] \qquad \frac{t \Downarrow r^m \quad m \sqsubseteq l}{(t \colon \iota^k \Rightarrow^p \iota^l) \Downarrow r^m}\ [\text{E-CAST-BASE}]$$

$$\frac{t \Downarrow r^m \quad m \sqsubseteq l \quad v = (\lambda x' \colon A'.\ (r^m\ (x' \colon A' \Rightarrow^{\overline{p}} A)) \colon B \Rightarrow^p B')^{\perp}}{(t \colon (A \to B)^k \Rightarrow^p (A' \to B')^l) \Downarrow v}\ [\text{E-CAST-FN}]$$

$$\frac{t \Downarrow r^m}{(t \colon \iota^k \Rrightarrow \iota^l) \Downarrow r^{m \sqcup l}}\ [\text{E-CLASSIFY-BASE}]$$

$$\frac{t \Downarrow r^m \quad v = (\lambda x' \colon A'.\ (r^m\ (x' \colon A' \Rrightarrow A)) \colon B \Rrightarrow B')^{m \sqcup l}}{(t \colon (A \to B)^k \Rrightarrow (A' \to B')^l) \Downarrow v}\ [\text{E-CLASSIFY-FN}]$$

$$\boxed{t \Downarrow blame\ p}$$

$$\frac{t \Downarrow r^m \quad m \not\sqsubseteq l}{(t \colon a^k \Rightarrow^p b^l) \Downarrow blame\ p}\ [\text{B-CAST-BAD}] \qquad \frac{t_i \Downarrow v_i \quad \forall i \in 1..j-1 \quad t_j \Downarrow blame\ p}{op\ \bar{t} \Downarrow blame\ p}\ [\text{B-PRIM}]$$

$$\frac{t \Downarrow blame\ p}{t\ s \Downarrow blame\ p}\ [\text{B-APP-L}] \qquad \frac{t \Downarrow v \quad s \Downarrow blame\ p}{t\ s \Downarrow blame\ p}\ [\text{B-APP-R}]$$

$$\frac{t \Downarrow blame\ p}{(t \colon a^k \Rightarrow^p b^l) \Downarrow blame\ p}\ [\text{B-CAST}] \qquad \frac{t \Downarrow blame\ p}{(t \colon A \Rrightarrow B) \Downarrow blame\ p}\ [\text{B-CLASSIFY}]$$

图 2. 操作语义

$$\boxed{\Gamma \vdash t : A}$$

$$\frac{}{\Gamma \vdash c^l : type(c)^l}\ [\text{T-CONST}] \qquad \frac{x : A \in \Gamma}{\Gamma \vdash x : A}\ [\text{T-VAR}]$$

$$\frac{\Gamma, x\colon A \vdash t : B}{\Gamma \vdash (\lambda x\colon A.\ t)^l : (A \to B)^l}\ [\text{T-ABST}] \qquad \frac{\Gamma \vdash t : A \quad A <:^+ B}{\Gamma \vdash (t\colon A \Rrightarrow B) : B}\ [\text{T-CLASSIFY}]$$

$$\frac{\Gamma \vdash t : (A \to b^k)^l \quad \Gamma \vdash s : A' \quad A' <: A}{\Gamma \vdash t\ s : b^{k \sqcup l}}\ [\text{T-APP}] \qquad \frac{\Gamma \vdash t : A \quad \lfloor A \rfloor = \lfloor B \rfloor}{\Gamma \vdash (t\colon A \Rightarrow^p B) : B}\ [\text{T-CAST}]$$

$$\frac{\Gamma \vdash t_i : a_i^{l_i} \quad i \in 1..n \qquad type(op) : a_1 \times \ldots \times a_n \to b \qquad l = \sqcup l_i}{\Gamma \vdash op\ \overline{t} : b^l}\ [\text{T-PRIM}]$$

$$\boxed{A <: B}$$

$$\frac{l \sqsubseteq k}{\imath^l <: \imath^k}\ [\text{SUB-BASE}] \qquad \frac{l \sqsubseteq k \quad A' <: A \quad B <: B'}{(A \to B)^l <: (A' \to B')^k}\ [\text{SUB-APP}]$$

$$\frac{l \sqsubseteq k}{\imath^l <:^+ \imath^k}\ [\text{SUB-P-BASE}] \qquad \frac{l \sqsubseteq k \quad A' <:^- A \quad B <:^+ B'}{(A \to B)^l <:^+ (A' \to B')^k}\ [\text{SUB-P-APP}]$$

$$\frac{k \sqsubseteq l}{\imath^l <:^- \imath^k}\ [\text{SUB-N-BASE}] \qquad \frac{k \sqsubseteq l \quad A' <:^+ A \quad B <:^- B'}{(A \to B)^l <:^- (A' \to B')^k}\ [\text{SUB-N-APP}]$$

$$\boxed{\lfloor A \rfloor : \lambda_{gif}\ types \to \lambda_{stlc}\ types}$$

$$\lfloor (A \to B)^k \rfloor = \lfloor A \rfloor \to \lfloor B \rfloor \qquad \lfloor a^k \rfloor = a$$

图 3. 类型系统

4 复现细节

4.1 与原论文对比

原论文只给出了形式化表达，并没有给出实际的编程实现，因此复现没有参考任何源代码。为了方便实现，复现中扩展了SecurityLevel的定义，在L和H的基础上增加了G，表示类型中没有标注标签的部分。此外，复现也对语法进行修改，将$t : A \Rrightarrow B$改为$t \Rrightarrow B$，$t : A \Rightarrow^p B$改为$t \Rightarrow^p B$。此举是为了方便程序员编写代码，去掉的A可由typing过程补充。另外，添加了两个方便编程的运算符#print和#string，前者将信息输出到控制台，后者将Int和$Bool$类型的值转换为字符串。

修改语法后变动的typing规则如下：

$$\frac{\Gamma \vdash t : A \quad A <:^+ B}{\Gamma \vdash t \Rrightarrow B : B}\ [\text{T-CAST}] \qquad \frac{\Gamma \vdash t : A \quad \lfloor A \rfloor = \lfloor B \rfloor}{\Gamma \vdash t \Rightarrow^p B : B}\ [\text{T-CLASSIFY}]$$

复现所实现的编译流程分为四个阶段：(1)词法分析；(2)语法分析，(3)语义分析（类型检查）；(4)解释执行。

4.2 词法分析和语法分析

由于所复现的内容是一门编程语言，因此需要对输入的源代码进行词法分析和语法分析。所用的工具是著名的编译器前端工具yacc。下面展示的是词法分析的部分代码。

```
program:
| term EOF { $1 }

term:
| simple_term       { $1 }
| simple_term args { mkApply $1 $2 }

simple_term:
| LPAREN term RPAREN { $2 }
| labeled_value       { TermValue $1 }
| IDENT               { TermVariable $1 }
| term EXCL labeled_type { TermClassification ($1, $3) }
| term DOUBLE_ARROW labeled_type { TermCast ($1, $2, $3) }
| term PLUS term { TermOperation (Add, [$1; $3]) }
| LET IDENT SEMI_COLON labeled_type EQUAL term IN term
        { TermApply (TermValue (LabeledValue
                (RawVLambda ($2, $4, $8), G)), $6) }
| HASH_STRING term { TermOperation (ToStr, [$2]) }
| HASH_PRINT term { TermOperation (Print, [$2]) }


raw_value:
| constant { RawVConstant $1 }
| BACK_DASH IDENT SEMI_COLON labeled_type DOT term
        { RawVLambda ($2, $4, $6) }
| LPAREN raw_value RPAREN { $2 }

labeled_value:
| raw_value              { LabeledValue ($1, L) }
| raw_value ANGLE LEVEL { LabeledValue ($1, $3) }

constant:
| BOOL          { ConstantBool $1 }
| INT           { ConstantInt $1 }
| STRING        { ConstantString $1 }
| LPAREN RPAREN { ConstantUnit }
```

```
37
38 raw_type:
39 | TYPE_IDENT { RawTBase (stringToBaseType $1) }
40 | labeled_type ARROW labeled_type { RawTArrow ($1, $3) }
41
42 labeled_type:
43 | LPAREN labeled_type RPAREN { $2 }
44 | raw_type                   { LabeledType ($1, G) }
45 | raw_type ANGLE LEVEL       { LabeledType ($1, $3) }
46
47 args:
48 | simple_term args   { $1 :: $2 }
49 | simple_term        { [$1] }
```

4.3 创新点

原论文是将渐进类型用于安全类型系统的开山之作，从此衍生出了许多这方面的研究，如在原论文的基础上加上可变变量的类型系统 [6]，将渐进安全类型系统扩展到面向对象语言 [7]等。

复现则对原论文的系统做了部分修改，使得这门编程语言更加人性化，更加符合实际编写程序的需要。

5 实验结果分析

实验设置如下，选用若干个会泄露高机密信息的程序源代码进行编译并执行，对于安全的程序将能正常运行；对于泄露高机密信息的程序会有两种情况：静态检查错误、动态检查错误。无论是哪种检查，都能保证高机密信息不泄露。

let $age : Int = 42$ in

let $salary : Int = 58000 \Rightarrow Int^H$ in

let $intToString : Int \to String = \lambda x : Int.\#\text{string}\ x$ in

let $print : String \to Unit = \lambda s : String.\#\text{print}\ (s \Rightarrow String^L)$ in

$print\ (intToString\ salary)$

图 4. 实验一（动态检查错误）

图 4所示程序有一个敏感信息salary，而#print函数会将任何东西泄露到公共信道，该程序定义了一个包装过的print函数，它可以检查即将打印的值是否为非高机密信息，如果不是，则抛出一个异常从而终止程序，防止信息泄露。这种通过运行时检查来防止泄露的方式称为动态检查错误。

let $age : Int^L = 42$ in

let $salary : Int^H = 58000 \Rrightarrow Int^H$ in

let $intToString : Int \to String = \lambda x : Int.\#\text{string } x$ in

let $intToStringL : Int^L \to String^L = intToString \Rightarrow (Int^L \to String^L)$ in

let $print : String^L \to Unit^L = \lambda s : String^L.\#\text{print } s$ in

$print\ (intToStringL\ salary)$

图 5. 实验二（静态检查错误）

图 5所示程序是以类型系统的方式静态检查发现泄露，intToStringL函数只允许将一个非机密的整形转换为一个非机密的字符串。当尝试转换机密信息salary时，类型系统检查发现salary的类型不匹配，从而拒绝继续编译程序。

let $fun : (Unit^L \to Int^L) \to Int^L = \lambda f : (Unit^L \to Int^L).f\ ()$ in

$\#\text{print } ((fun \Rightarrow (Unit^L \to Unit^H) \to Int^L)(\lambda u : Unit^L.1^H))$

图 6. 实验三（动态检查错误）

图 6所示程序主要是用于说明本编程语言的系统能够应对复杂的函数类型转换。

let $salary : Int^H = 12445^H$ in

let $leakFunction : Int^H \to String^L = \lambda x : Int^H.(\#\text{string } x) + "leaking"$ in

$\#\text{print } (leakFunction\ salary)$

图 7. 实验四（静态检查错误）

图 7当一个高机密变量x和一个非机密的字符串进行符号运算时，其运算结果也将是高机密的，因此此处检查不通过，拒绝编译。

let $salary : Int = 12445^H$ in

let $leakFunction : Int \to String = \lambda x : Int.(\#\text{string } x) + "leaking"$ in

$\#\text{print } ((leakFunction\ salary) \Rightarrow String^L)$

图 8. 实验五（动态检查错误）

图 8leakFunction属于一个不进行信息流泄露检查的普通函数，其返回值的机密性不确定，为确保安全，可以在#print之前使用强制转换进行运行时检查，此处程序就通过这一方式防止了一次信息流泄露。

6 总结与展望

本文介绍了软件安全对于我国信息技术以及互联网行业的重要意义。选用了一门带渐进安全类型系统的语言，介绍了其相关工作。复现修复了原论文的部分缺陷，但仍然存在不足：

（1）原论文所选用的类型系统过于简单，缺乏编写复杂程序的抽象能力，如递归、代数数据类型、泛型、可变变量。

（2）文中的系统不能很智能地插入动态类型检查，有些时候仍然需要手动添加。

（3）理论方面只证明了non-interference性质，没有做gradual type相关性质的证明。

未来可通过解决以上缺点来改进整门编程语言。

参考文献

[1] Jif homepage, 2010.

[2] Amal Ahmed, Robert Bruce Findler, Jeremy G Siek, and Philip Wadler. Blame for all. In *Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 201–214, 2011.

[3] Thomas H Austin and Cormac Flanagan. Efficient purely-dynamic information flow analysis. In *Proceedings of the ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security*, pages 113–124, 2009.

[4] Thomas H Austin and Cormac Flanagan. Permissive dynamic information flow analysis. In *Proceedings of the 5th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, pages 1–12, 2010.

[5] Dominique Devriese and Frank Piessens. Noninterference through secure multi-execution. In *2010 IEEE Symposium on Security and Privacy*, pages 109–124. IEEE, 2010.

[6] Luminous Fennell and Peter Thiemann. Gradual security typing with references. In *2013 IEEE 26th Computer Security Foundations Symposium*, pages 224–239. IEEE, 2013.

[7] Luminous Fennell and Peter Thiemann. Ljgs: Gradual security types for object-oriented languages. In *30th European Conference on Object-Oriented Programming (ECOOP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[8] Robert Bruce Findler and Matthias Felleisen. Contracts for higher-order functions. In *Proceedings of the seventh ACM SIGPLAN international conference on Functional programming*, pages 48–59, 2002.

[9] Kenneth Knowles, Aaron Tomb, Jessica Gronski, S Freund, and Cormac Flanagan. Sage: Unified hybrid checking for first-class types, general refinement types and dynamic. Technical report, Technical report, UCSC, 2007.

[10] Andrew C. Myers. Jflow: Practical mostly-static information flow control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, page 228–241, New York, NY, USA, 1999. Association for Computing Machinery.

[11] Andrew C Myers and Barbara Liskov. A decentralized model for information flow control. *ACM SIGOPS Operating Systems Review*, 31(5):129–142, 1997.

[12] Philip Wadler and Robert Bruce Findler. Well-typed programs can' t be blamed. In *European Symposium on Programming*, pages 1–16. Springer, 2009.

[13] Stephan Arthur Zdancewic. *Programming languages for information security*. Cornell University, 2002.