ISSN 1746-7659, England, UK
Journal of Information and Computing Science
Vol. 2, No. 4, 2007, *pp*. 252-260

Secure Information Flow Based on Data Flow Analysis

Jianbo Yao

Center of Information and computer, Zunyi Normal College, Zunyi, Guizhou, 563002, China

(*Received December 21, 2006, accepted 9 April 2007*)

Abstract. The static analysis of secure information flow has been studied for many years. The existing methods tend to be overly conservative or to be overly attention to location information leak. This paper uses data flow analysis to deal with secure information flow. Two variables of dynamic update security levels were introduced. The program is secure without any variable of downgrade security level at exit of a program. The analysis can deal with more secure programs. The soundness of the analysis is proved.

Keywords: secure information flow, data flow analysis, static analysis, formal semantics

1. Introduction

The static analysis of secure information flow has been studied for many years[1]. The analysis of secure information flow is that check whether the information flow of a given program is secure. Secure information flow ensures the noninterference, that is, observations of the initial and final values of low-security variables do not provide any information about the initial value of high-security variables.

Consider a program whose variables are partitioned into two disjoint tuples H (secret) and L (public). The program is secure if examinations of the initial and final values of any L variable do not give any information about the initial values of any H variable.

For example, the program

$$H := L$$

is secure since the value of L is independent of initial value of H .

Similarly, the program

$$L := 8$$

is secure, because the final value of L is always 8, regardless of the initial value of H .

However, the program

$$L := H$$

is not insecure since the value of H can be observed as the final value of L . the flow of information from H to L is called explicit. We call this explicit leak from H to L .

The program

$$\text{if } H \text{ then } L := 0 \text{ else } L := 1$$

is insecure, despite each branch of the conditional is secure, H is indirectly copied into L . the flow of information from H to L is called implicit. We call this implicit leak from H to L .

A secure program may have some location information leak. For example, in each of four programs

$$L := H;\ L := 8 \qquad ①$$

$$H := L;\ L := H \qquad ②$$

$$L := H;\ L := L - H \qquad ③$$

$$\text{if false then } L := H \text{ end} \qquad ④$$

although existing location information leak $L := H$, the four program are all secure.

The existing methods tend to be overly conservative, giving “insecure” answers to many “secure” programs, or to be overly attention to location information leak, existing location information leak does not

imply there is information leak in a program.

In this paper, the data flow analysis is used to deal with secure information flow. The analysis proposed in this paper is more precise than the existing syntactic approaches. However, since the analysis is syntactic in nature, it cannot be as precise as the Joshi-Leino's and Sabelfeld-Sands' semantic approaches [6,7] or Darvas-Hähnle-Sands' theorem proving approach. [8]

The rest of the paper is organized as follows: Section 2 informally describes the problem of secure information flow using some simple examples. Section 3 presents the syntax and semantics of While language. Section 4 explains how to construct the flow graph and then shows data flow equations for detecting information leaks. Section 5 proves the soundness of the analysis. Section 6 concludes.

The Denning-Denning's original method[9], the Mizuno-Schmidt's data flow analysis[2] and the Volpano-Smith's type-system all assert that above four programs are insecure. Doh-Shin's data flow analysis claim that it can solve the problem of program① and program②, but in fact, it only can solve the secure problem of program②. Our analysis in the paper certifies that above four programs are all secure[6].

2. Syntax and Semantics

In this paper, we shall consider an imperative language core[10], While. In order to identify statements and tests in a While program, we give a unique label to each of the assignments, skip statements, and tests.

Syntax Domain:

$a \in AExp$ arithmetic expressions

$b \in BExp$ Boolean expressions

$S \in Stmt$ statements

$x \in Var$ variables

$l \in Lab$ labels

Abstract Syntax:

$$S ::= [x:=a]^l \mid [skip]^l \mid S_1;S_2 \mid \text{if } [b]^l \text{ then } S_1 \text{ else } S_2 \mid \text{while } [b]^l \text{ do } S$$

Configurations and Transitions:

A state is defined as a mapping from variables to integers:

$$\sigma \in State = Var \to Z$$

A configuration of the semantics is either a pair $\langle S, \sigma \rangle$ or σ, where $S \in Stmt$ and $\sigma \in State$. A terminal configuration consists only of a state. The transition of the semantics shows how the configuration is changed by one step of computation and is represented as one of the followings:

$$\langle S, \sigma \rangle \to \sigma' \text{ and } \langle S, \sigma \rangle \to \langle S', \sigma' \rangle$$

For arithmetic and Boolean expressions, a and b, we assume that the semantic functions are defined as follows:

$$A : AExp \to (State \to Z)$$

$$B : BExp \to (State \to T)$$

where Z is the set of integers and T is the set of truth values.

Structural Operational Semantics:

$$[ass] \qquad \langle [x := a]^l, \sigma \rangle \to \sigma[x \mapsto A[\![a]\!]\sigma]$$

$$[skip] \qquad \langle [skip]^l, \sigma \rangle \to \sigma$$

$$[seq_1] \qquad \frac{\langle S_1, \sigma \rangle \to \langle S_1', \sigma' \rangle}{\langle S_1; S_2, \sigma \rangle \to \langle S_1'; S_2, \sigma' \rangle}$$

$$[seq_2] \quad \frac{\langle S_1, \sigma \rangle \to \sigma'}{\langle S_1; S_2, \sigma \rangle \to \langle S_2, \sigma' \rangle}$$

$[if_1]$ 〈 if [b]^l then S_1 else S_2, σ 〉 → 〈S_1, σ 〉 , if $B[\![b]\!]\sigma = true$

$[if_2]$ 〈 if [b]^l then S_1 else S_2, σ 〉 → 〈S_2, σ 〉 , if $B[\![b]\!]\sigma = false$

$[wh_1]$ 〈 while [b]^l do S, σ 〉 → 〈S; while [b]^l do S , σ 〉 , if $B[\![b]\!]\sigma = true$

$[wh_2]$ 〈 while [b]^l do S, σ 〉 → σ , if $B[\![b]\!]\sigma = false$

3. Secure Information Flow Analysis

In this section, we use the data flow analysis to deal with secure information flow[3,11]. We first define the suitable flow graph of While programs, and then formulate data flow equations for the analysis.

3.1. The Flow Graph

The flow graph is defined in the style of Nielson-Nielson-Hankin's book[8]. In order to analysis the secure information flow, we need explicitly add to the flow graph an implicit flow from a test block to each statement block in the conditional branch or in the while-loop body, in addition to the normal control flow.

A flow graph consists of the set of elementary blocks and the set of (control and implicit)flows between blocks. More formally, the flow graph for a While statement S is defined to be a quintuple:

flowgraph(S) = (block(S), flow(S), flow $_I$(S), init(S), final(S))

where each of the functions are defined below.

Let Blocks be the set of elementary blocks of form $[x := a]^l$, $[skip]^l$ or $[b]^l$ where $l \in Lab$. Then the function blocks finds the set of elementary blocks in a given statement:

$$blocks : Stmt \to P(Blocks)$$

$$blocks([x := a]^l) = \{[x := a]^l\}$$

$$blocks([skip]^l) = \{[skip]^l\}$$

$$blocks(S_1, S_2) = blocks(S_1) \cup blocks(S_2)$$

$$blocks(\text{ if } [b]^l \text{ then } S_1 \text{ else } S_2) = \{[b]^l\} \cup blocks(S_1) \cup blocks(S_2)$$

$$blocks(\text{ while } [b]^l \text{ do S}) = \{[b]^l\} \cup blocks(S)$$

A flow graph always has a single entry, but it may have multiple exits due to conditional statements. Thus the function init returns the initial label of a give statement:

$$init : Stmt \to Lab$$

$$init([x := a]^l) = l$$

$$init([skip]^l) = l$$

$$init(S_1; S_2) = init(S_1)$$

$$init(\text{ if } [b]^l \text{ then } S_1 \text{ else } S_2) = l$$

$$init(\text{ while } [b]^l \text{ do S}) = l$$

The function final returns the set of final labels of a given statement:

$$final : Stmt \to P(Lab)$$

$$final\left(\left[x := a\right]^l\right) = \{l\}$$

$$final\left(\left[skip\right]^l\right) = \{l\}$$

$$final\left(S_1; S_2\right) = final(S_2)$$

$$final\ (\text{ if } \left[b\right]^l \text{ then } S_1 \text{ else } S_2) = final\left(S_1\right) \cup final(S_2)$$

$$final\ (\text{ while } \left[b\right]^l \text{ do } S) = \{l\}$$

The function flow returns control flows between blocks in a given statement:

$$flow : Stmt \to P(Lab \times Lab)$$

$$flow\left(\left[x := a\right]^l\right) = \phi$$

$$flow\left(\left[skip\right]^l\right) = \phi$$

$$flow\left(S_1; S_2\right) = flow(S_1) \cup flow(S_2) \cup \left\{\left(l, init\left(S_2\right)\right) \middle| l \in final\left(S_1\right)\right\}$$

$$flow\ (\text{ if } \left[b\right]^l \text{ then } S_1 \text{ else } S_2) = flow(S_1) \cup flow(S_2) \cup \left\{\left(l, init\left(S_1\right)\right), \left(l, init\left(S_2\right)\right)\right\}$$

$$flow\ (\text{ while } \left[b\right]^l \text{ do } S) = flow(S) \cup \left\{\left(l, init\left(S\right)\right)\right\} \cup \left\{\left(l', l\right) \middle| l' \in final\left(S\right)\right\}$$

The function $flow_I$ defines the implicit flows in a given statement:

$$flow_I : Stmt \to P(Lab \times Lab)$$

$$flow_I\left(\left[x := a\right]^l\right) = \phi$$

$$flow_I\left(\left[skip\right]^l\right) = \phi$$

$$flow_I\left(S_1; S_2\right) = flow_I(S_1) \cup flow_I(S_2)$$

$$flow_I\ (\text{ if } \left[b\right]^l \text{ then } S_1 \text{ else } S_2) =$$

$$flow_I(S_1) \cup flow_I(S_2) \cup \left\{\left(l, l'\right) \middle| B^{l'} \in blocks(S_1)\right\} \cup \left\{\left(l, l'\right) \middle| B^{l'} \in blocks(S_2)\right\}$$

$$flow_I\ (\text{ while } \left[b\right]^l \text{ do } S) = flow_I(S) \cup \left\{\left(l, l'\right) \middle| B^{l'} \in blocks(S)\right\}$$

For example, consider the Power program:

$$\left[z := 1\right]^1; while\left[x > 0\right]^2 do\left(\left[z := z * y\right]^3; \left[x := x - 1\right]^4\right)$$

we have

$$flowgraph\left(Power\right) = \left(blocks(S), flow(S), flow_I(S), init(S), final(S)\right)$$

$$= \left(\left\{\left[z := 1\right]^1, \left[x > 0\right]^2, \left[z := z * y\right]^3, \left[x := x - 1\right]^4\right\}, \left\{(1,2), (2,3), (3,4), (4,2)\right\},$$

$$\left\{(2,3), (2,4)\right\}, 1, \{2\}\ \{2\}\right)$$

3.2. The Analysis

Assume information only have two security levels H and L. Each variable x in a program is initially bound to a security level, which is denoted by underline, $\underline{x}$. $x\uparrow$ denotes a L variable x which security level is upgrade after coping s H variable to a L variable x; $x\downarrow$ denotes a H variable x which security level is

downgrade after coping L variable to a H variable x. x_I denotes a implicit H variable when x is a H variable in test blocks.

The analysis is defined as follow:

$$gen_{IL} : Blocks \to P\left(\{x\uparrow\},\{x\downarrow\},\{x_I\}\right):$$

$$gen_{IL}\left(\left[x := a\right]^l\right) = \left(\{x\uparrow\},\{x\downarrow\},\phi\right)$$

where

$$\{x\uparrow\} = \left\{x\uparrow \middle| y \in FV(a), y \notin \{x\downarrow\}, \underline{x} < \underline{y}\right\} \cup \left\{x\uparrow \middle| z \in FV(a), z := y, z \in \{x\uparrow\}, \underline{x} < \underline{y}\right\}$$

$$\cup \left\{x\uparrow \middle| y \in \{x_I\}, x := y, \underline{x} < \underline{y}\right\}$$

$$\{x\downarrow\} = \left\{x\downarrow \middle| \underline{x} = H; \forall y \in FV(a), \underline{y} = L, y \notin \{x\uparrow\}\right\}$$

$$gen_{IL}\left(\left[x := a\right]^l\right) = (\phi,\phi,\phi) \text{ if never execute } \left[x := a\right]^l$$

$$gen_{IL}\left(\left[skip\right]^l\right) = (\phi,\phi,\phi)$$

$$gen_{IL}\left(\left[b\right]^l\right) = (\phi,\phi,\{x_I\})$$

where

$$\{x_I\} = \left\{x_I \middle| x \in FV(b), \underline{x} = H, x \notin \{x\downarrow\}\right\} \cup \left\{x_I \middle| y \in FV(a), y := x, \underline{x} = H\right\}$$

$$kill_{IL} : Blocks \to P\left(\{x\uparrow\},\{x\downarrow\},\{x_I\}\right):$$

$$kill_{IL}\left(\left[x := a\right]^l\right) = \left(\{x\uparrow\},\{x\downarrow\},\phi\right)$$

where

$$\{x\uparrow\} = \left\{x\uparrow \middle| \forall y \in FV(a), \underline{y} = L \cup \left(y\uparrow -H\right)\right\}$$

$$\{x\downarrow\} = \left\{x\downarrow \middle| \forall y \in FV(a), \underline{y} = H \cup y \in \{x\uparrow\}\right\}$$

$$kill_{IL}\left(\left[x := a\right]^l\right) = (\phi,\phi,\phi) \text{ if never execute } \left[x := a\right]^l$$

$$kill_{IL}\left(\left[skip\right]^l\right) = (\phi,\phi,\phi)$$

$$kill_{IL}\left(\left[b\right]^l\right) = (\phi,\phi,\phi)$$

Information Low Equations: $IL^{=}$:

$$IL_{entry}(l) = \begin{cases} (\phi,\phi,\phi) & \text{if } l = init(S) \\ \left(\cup\{x\uparrow\}(l'),\cup\{x\downarrow\}(l'),\cup\{x_I\}(l')\right) & \\ & (l',l) \in flow(S) \cup flow_I(S) \text{ otherwise} \end{cases}$$

$$IL_{exit}(l) = \left(\left(\{x\uparrow\} \setminus \{x\uparrow\}_{kill}\right) \cup \{x\uparrow\}_{gen}, \left(\{x\downarrow\} \setminus \{x\downarrow\}_{kill}\right) \cup \{x\downarrow\}_{gen},\right.$$

$$\left.\left(\{x_I\} \setminus \{x_I\}_{kill}\right) \cup \{x_I\}_{gen}\right)$$

Example 1.Consider the following program:

$$[x := y]^1 ; [x := z]^2 \text{, where } \underline{y} = H \text{ and } \underline{x}=\underline{z}=\text{L}.$$

$$IL_{entry}(1) = (\phi, \phi, \phi)$$

$$IL_{exit}(1) = (\{x\uparrow\}, \phi, \phi)$$

$$IL_{entry}(2) = (\{x\uparrow\}, \phi, \phi)$$

$$IL_{exit}(2) = (\phi, \phi, \phi)$$

$x\uparrow$ is killed at the exit of block 2, at the termination of the program, $\underline{x}=\underline{z}=\text{L}$ is not relation with $\underline{y} = H$. Thus the program is secure.

Example 2.Consider the following program:

$$[y := x]^1 ; [x := y]^2 \text{, where } \underline{y} = H \text{ and } \underline{x}=\text{L}.$$

$$IL_{entry}(1) = (\phi, \phi, \phi)$$

$$IL_{exit}(1) = (\phi, \{x\downarrow\}, \phi)$$

$$IL_{entry}(2) = (\phi, \{x\downarrow\}, \phi)$$

$$IL_{exit}(2) = (\phi, \{x\downarrow\}, \phi)$$

at the termination of the program, $\underline{x}=\text{L}$ is not relation with $\underline{y} = H$. Thus the program is secure.

Example 3.Consider the following program:

$$[x := y]^1 ; [x := x - y]^2 \text{, } where \ \underline{x} = L \text{ and } y := H.$$

$$IL_{entry}(1) = (\phi, \phi, \phi)$$

$$IL_{exit}(1) = (\{x\uparrow\}, \phi, \phi)$$

$$IL_{entry}(2) = (\{x\uparrow\}, \phi, \phi)$$

$$IL_{exit}(2) = (\phi, \phi, \phi)$$

at the termination of the program, $\underline{x}=\text{L}$ is not relation with $\underline{y} = H$. Thus the program is secure.

Example 4.Consider the following program:

$$\text{if } [false]^1 \text{ then } [x := y]^2 \text{ end, where } \underline{x}=\text{L} \text{ and } \underline{y} = H$$

$$IL_{entry}(1) = (\phi, \phi, \phi)$$

$$IL_{exit}(1) = (\phi, \phi, \phi)$$

$$IL_{entry}(2) = (\phi, \phi, \phi)$$

$$IL_{exit}(2) = (\phi, \phi, \phi)$$

at the termination of the program, $\underline{x}=\text{L}$ is not relation with $\underline{y} = H$. Thus the program is secure.

4. The Soundness as Noninterference

In this section, we prove that the analysis is sound by proving our analysis' noninterference property[11].

Theorem 1. Given a While program S, for each block $B^l \in blocks(S)$, we let

$$IL_{entry}(l)=\left(\{x\uparrow\}_{entry},\{x\downarrow\}_{entry},\{x_I\}_{entry}\right)$$

$$IL_{exit}(l)=\left(\{x\uparrow\}_{exit},\{x\downarrow\}_{exit},\{x_I\}_{exit}\right)$$

$N(l)$ is the set of all variables having L values at the entry of block B^l, $X(l)$ is the set of all variables having L values at the entry of block B^l.

$$N(l)=\{x\,|\,x\in Var,\underline{x}=L\}\setminus\{x\uparrow\}\setminus\{x_I\}\cup\{x\downarrow\}$$

$$X(l)=\{x\,|\,x\in Var,\underline{x}=L\}\setminus\{x\uparrow\}\cup\{x\downarrow\}$$

(1) if $\langle S,\sigma_1\rangle\to\langle S',\sigma_1'\rangle$ and $\sigma_1\sim_{N(init(S))}\sigma_2$ then there exists σ_2' such that $\langle S,\sigma_2\rangle\to\langle S',\sigma_2'\rangle$ and $\sigma_1'\sim_{N(init(S))}\sigma_2'$, and

(2) if $\langle S,\sigma_1\rangle\to\sigma_1'$ and $\sigma_1\sim_{N(init(S))}\sigma_2$ then there exists σ_2' such that $\langle S,\sigma_2\rangle\to\sigma_2'$ and $\sigma_1'\sim_{N(init(S))}\sigma_2'$

Proof: The proof is by induction on the shape of the inference tree used to establish $\langle S,\sigma_1\rangle\to\langle S',\sigma_1'\rangle$ and $\langle S,\sigma_1\rangle\to\sigma_1'$, respectively.

The case $[ass]$. Then $\langle[x:=a]^l,\sigma_1\rangle\to\sigma_1[x\mapsto A[\![a]\!]\sigma_1]$, and we have

$$\{x\uparrow\}_{exit}=\{x\uparrow\}_{entry}\setminus\{x\uparrow\}_{kill}\cup\{x\uparrow\}_{gen}$$

$$=\{x\uparrow\}_{entry}\setminus\{x\uparrow\}_{kill}\cup\{x\uparrow\,|\,y\in FV(a),y\notin\{x\downarrow\},\underline{x}<\underline{y}\}$$

$$\cup\{x\uparrow\,|\,z\in FV(a),z:=y,z\in\{x\uparrow\},\underline{x}<\underline{y}\}\cup\{x\uparrow\,|\,y\in\{x_I\},x:=y,\underline{x}<\underline{y}\}$$

$$\{x\downarrow\}_{exit}=\{x\downarrow\}_{entry}\setminus\{x\downarrow\}_{kill}\cup\{x\downarrow\}_{gen}$$

$$=\{x\downarrow\}_{entry}\setminus\{x\downarrow\}_{kill}\cup\{x\downarrow\,|\,\underline{x}=H;\forall y\in FV(a),\underline{y}=L,y\notin\{x\uparrow\}\}$$

Since information flow is secure, this is $\{x\uparrow\}_{exit}=\phi$, then $\{x\uparrow\}_{gen}=\phi$. Thus we get

$$\forall y\in FV(a),y\in\{x\downarrow\}\vee\underline{y}=L$$

$$\forall z\in FV(a),y\notin\{x\uparrow\}\vee\underline{y}=L$$

$$\{x_I\}_{entry}=\phi$$

Therefore, we have

$$X(l)=N(l)\cup\{x\downarrow\}_{gen}$$

and thus

$$\sigma_1\sim_{N(l)}\sigma_2\ \text{implies}\ A[\![a]\!]\sigma_1=A[\![a]\!]\sigma_2$$

because the value of a is only affected by the L variables occurring in it. Taking

$$\sigma_2'=\sigma_2[x\mapsto A[\![a]\!]\sigma_2]$$

we have that $\sigma_1'(x)=\sigma_2'(x)$ and thus $\sigma_1'(x)\sim_{X(l)}\sigma_2'(x)$ as required.

The case $[skip]$. Then $\langle [skip]^l, \sigma_1 \rangle \to \sigma_1$, we have

$$X(l) = N(l)$$

and we take $\sigma_2^{'}$ to be σ_2.

The case $[seq1]$. Then by the induction hypothesis, because $\sigma_1(x) \sim_{N(init(S_1))} \sigma_2(x)$ implies $\sigma_1^{'}(x) \sim_{N(init(S_1^{'}))} \sigma_2^{'}(x)$, Since $init(S_1; S_2) = init(S_1)$ and $init(S_1^{'}; S_2) = init(S_1^{'})$, we can immediately conclude: $\sigma_1(x) \sim_{N(init(S_1;S_2))} \sigma_2(x)$ implies $\sigma_1^{'}(x) \sim_{N(init(S_1^{'};S_2))} \sigma_2^{'}(x)$.

The case $[seq2]$. Similar to The cass $[seq1]$.

The case $[if_1]$. Then $\langle \text{if } [\text{b}]^l \text{ then } \text{S}_1 \text{ else } \text{S}_2, \sigma_1 \rangle \to \langle S_1, \sigma_1 \rangle$ because

$$N_{init(\text{if } [\text{b}]^l \text{ then } \text{S}_1 \text{ and } \text{S}_2)} = \{x | x \in Var, \underline{x} = L\} \setminus \{x\uparrow\} \setminus \{x_I\} \cup \{x\downarrow\}$$

and

$$N_{init(\text{S}_1)} = \{x | x \in Var, \underline{x} = L\} \setminus \{x\uparrow\} \setminus \{x_I\} \setminus \{x_I(b)\} \cup \{x\downarrow\}$$

where

$$\{x_I(b)\} = \{x_I | y \in FV(b), y := x, \underline{x} = H\}$$

Hence, we have

$$N_{init(S_1)} \subseteq N_{init(\text{if } [\text{b}]^l \text{ then } \text{S}_1 \text{ else } \text{S}_2)}$$

Hence, we have

$$\sigma_1(x) \sim_{N(init(S_1))} \sigma_2(x)$$

The case $[if_2]$. Similar to The cass $[if_1]$.

The case $[wh]$. Similar to The cass $[if]$.

This completes the proof.

Finally, we have an important corollary which states that noninterference is preserved throughout the execution of the entire program.

Corollary 1: Under the same assumption as Theorem 1. then

(1) (not yet terminated programs)

$$\text{if } \langle \text{S}, \sigma_1 \rangle \to^* \langle S^{'}, \sigma_1^{'} \rangle \text{ and } \sigma_1 \sim_{N(init(S))} \sigma_2 \text{ then there exists}$$

$$\sigma_2^{'} \text{ such that } \langle \text{S}, \sigma_2 \rangle \to^* \langle S^{'}, \sigma_2^{'} \rangle \text{ and } \sigma_1^{'} \sim_{N(init(S))} \sigma_2^{'}, \text{ and}$$

(2) (terminated programs)

$$\text{if } \langle \text{S}, \sigma_1 \rangle \to * \sigma_1^{'} \text{ and } \sigma_1 \sim_{N(init(S))} \sigma_2 \text{ then there exists}$$

$$\sigma_2^{'} \text{ such that } \langle \text{S}, \sigma_2 \rangle \to * \sigma_2^{'} \text{ and } \sigma_1^{'} \sim_{N(l)} \sigma_2^{'} \text{ for some } l \in final(S).$$

5. Conclusion

This paper uses data flow analysis to deal with secure information flow. The analysis proposed in this paper is more precise than the existing syntactic approaches. The analysis is proved to be sound by proving

our analysis' noninterference property.

6. References

[1] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communication*, 2003, **21**(1).

[2] Dennis Volpano, Geoffrey Smith. A Type-Based Approach to Program Security. *Proceeding of TAPSOFT'97, colloquium on Formal Approaches in Software Engineering*, Lille France, 14-18 April, 1997.

[3] Dennis Volpano, Geoffrey Smith, Cynthia Irvine. A Sound Type System for Secure Flow Analysis. *Journal of computer security*. 1996, **29**.

[4] M. Mizuno and D. A. Schmidt. A security flow control algorithm and its denotational semantics correctness proof. *Formal Aspects of Computing*, 1992, **4**: 722-754.

[5] Kyung-Goo Doh and Seung Cheol Shin. Data Flow Analysis of Secure Information-Flow. *ACM SIGPLAN Notices*. 2002, **37**(8).

[6] R. Joshi and K. R. M. Leino. A semantic approach to secure information flow. *Science of Computer Programming*, 2000, **37**: 113-138.

[7] A.Sabelfeld and D. Sands. A per model of secure information flow in sequential programs. *Higher-Order and Symbolic Computations*, 2001, **14**: 59-91.

[8] A. Darvas, R.Hahnle and D. Sands. A Theorem Proving Approach Analysis of Secure Information Flow. *Technical Report*, no. 2004-01.

[9] D. Denning, P. Denning. Certification of Programs for Secure Information Flow. *Communications of the ACM*, 1977, **20**(7): 504-513.

[10] Hanne Riis Nielson and Flemming Nielson. *Semantics with Applications a Formal Introduction*. July, 1999.

[11] F. Nielson, H. R. Nielson and C. Hankin. *Principles if Program Analysis. Springer*.1999.