CS4450 Lecture 32 Notes, Page 1

Andrew Krall

11-02-2018

Today, we'll be talking about security.

<u>Language-Based Security (LBS)</u>

The goal was to build security into a program.

Basically, you build security into the tool you use to build software.

Will define a notion of information flow within a language, then will define a system for proving that there is no insecure information flow in a program.

<u>Formal Security Specification</u>

<u>Non-interference:</u> Famous security model.

<u>Interference:</u> A thread in a program that performs differently than intended, can be exploited.

<u>Non-interference security:</u> Thread behavior is always the same.

Security variables are added to the program to ensure program security. →

Andrew Krall
11-02-2018

The main goal is that you don't want any flows from "high" variables to "low" variables.

↳ Effectively, there should be no impact between high and low variables.

Information Policy Flow as Lattice

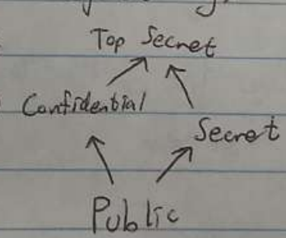
Smallest reasonable lattice will be assumed in our examples.

Lattice: Ordered structure.

$\underline{x}$ ("x underbar") is the security level of x.

"$\underline{x} \rightarrow \underline{y}$" means that information flow is permitted by policy from object x to object y.

Confidential $\oplus$ Secret = Top Secret



Single arrow means "Flow is permitted".

Double arrow means "Flow may occur".

The paper regarding this security basically discusses how to find a way to calculate double arrow. →

CS4450 Lecture 32 Notes,

Page 3

Andrew Krall

11-02-2018

$y := e[x]$
means
$x \Rightarrow y$
(x is flowing to y)

$\underbrace{y := x}+y$

x still flows to y. A flow here is obvious when one variable is being assigned to another.

Implicit Flows: "$y := 1$; if $x = 0$ then $y := 0$"

Notice x is not being assigned to y, but the contents of y must rely on x. Therefore, assuming x is $\emptyset$ or 1, then $x = y$ after completion and $x \Rightarrow y$.

Also, if $x \Rightarrow y$ and $y \Rightarrow z$, then $x \Rightarrow z$ (x is transitive).

A program statement specifies a flow if its execution could result in a flow.
↳ Note that this is weaker than "does result in a flow".

$y := 1$
while $x = 0$ do $y := 0$

If $x \Rightarrow y$, we know that there's some conditional that occurs as long as $x = 0$. Since this is not guaranteed, we must approximate and therefore say "could". →

CS4450 Lecture 32 Notes,

Andrew Krall

11-02-2018

Security Requirements

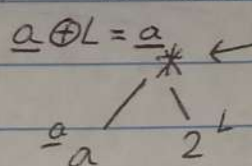
Program p is <u>secure</u> iff flow $x \Rightarrow y$ results from executing p only when $\underline{x} \rightarrow \underline{y}$.

<u>Security Definition:</u> Flow $x \Rightarrow y$ is specified by p only when $\underline{x} \rightarrow \underline{y}$.

Certification Mechanism

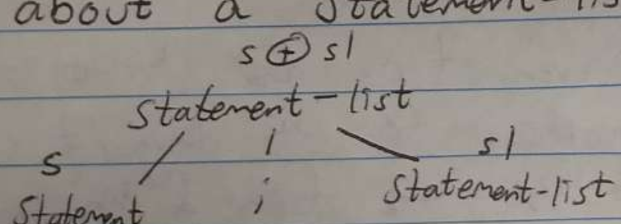
Abstract syntax can be written for these flows.

The security definitions of the variables in the tree are contained inside the variables. The security levels are denoted by the underbar (e.g $\underline{c}$ is the security level of variable c).



← The security level of multiplication is the "least upper bound" of the two child nodes. Therefore, flows are calculated "upwards".

How about a Statement-list?



→

CS4450 Lecture 32 Notes,
Page 5

Andrew Krall
11-02-2018

Statement

if Exp then Statement else Statement 2

<u>Theorem:</u> A program is certified only if it is secure.

<u>Summary</u>

Compile-time security certification is a big plus.
↳ Check the program once and no run-time checks necessary.