Turvalised infovood ja nende kontroll

Jaak Ristioja
j077@ut.ee

MTAT.03.204 Programmeerimiskeelte semantika uurimisseminar
Arvutiteaduse instituut, Tartu Ülikool
November 2009.

Abstrakt

Informatsiooni turvamiseks kasutatakse enim ligipääsu piiramist failidele, programmidele või muudele ressurssidele. Selleks kasutatavad meetodid aga ei suuda tagada, et informatsioon ei leki programmide siseselt. Käesoleva töö eesmärgiks on anda ülevaade programmianalüüsi kasutamisest infovoogude turvalisuse kindlakstegemisel. Esmalt anname ülevaate infovoogudest, nende liikidest ja turvamise vajalikkusest; ning turvalise infovoo võremudelist. Ühe lihtsa programmeerimiskeele näidete varal kirjeldame informatsiooni lekkeid, sertifitseerimismehhanismide ja tüübisüsteemide kasutamist infovoogude turvalisuse analüüsiks. Töö keskendub ühelõimeliste, determineeritud, staatiliselt tüübitud programmide infovoogude semantilisele analüüsile turvalisuse aspektist.

Sissejuhatus

Traditsionaalselt on tarkvara turvalisust jõustatud operatsioonisüsteemi tasemel erinevatele ressurssidele, nagu näiteks failidele või võrguühendustele, ligipääsu piiramisega; andmete turvalisust on jõustatud ka krüpteerimisega. Andmete konfidentsiaalsuse tagamiseks kasutatavad pääsumehhanismid ei suuda aga piisaval määral piirata informatsiooni tegelikku levikut, määrates vaid, kas protsessidel on informatsioonile ligipääs või mitte. Pääsumehhanismid ei kontrolli, kuidas ligipääsuõigusi omavad protsessid informatsiooni töötlevad ja edastavad. Seega ei suuda pääsumehhanismid garanteerida, et informatsiooni tegelik levik vastaks täielikult soovitud turvapoliitikale. Näiteks kui kasutajaga suhtleval rakendusel on pääsuõigus lugemaks salajast informatsiooni, ning rakendusega suhtleval kasutajal vastavad pääsuõigused puuduvad, ei suuda ainuüksi pääsuõigused garanteerida, et antud rakendus kasutajale konfidentsiaalseid andmeid ei lekita.

Andmete turvalisuse tagamiseks ehk infolekete vältimiseks peavad informatsiooni töötlevad programmid järgima teatud reegleid informatsiooni käitlemisel. Selleks on võetud kasutusele mudeleid, mis on tüüpiliselt määravad erinevatele andmetele ja süsteemi kasutajatele turvaklassid ning määravad lubatud informatsiooni liikumise suunad nende turvaklasside vahel. Näiteks kahe turvaklassi, „salajane“ ja „avalik“, korral võib olla lubatud „avalikku“ turvaklassi kuuluva informatsiooni levik „salajasse“ turvaklassi, kuid mitte vastupidi. Andmete liikumine turvaklasside siseselt on tüüpiliselt lubatud. Taoliste mudelite rakendamise eesmärkideks on informatsiooni lekked täielikult elimineerida või neid piirata miinimumini.

Nimetatud mudeleid kasutatakse arvutiprogrammide turvalisuse kontrollimiseks informatsiooni leviku kontekstis. Kõige mugavam on seda teha programmi arendamise etappidel, kasutades staatilist programmianalüüsi kontrollimaks arvutiprogrammide sisemise ehk algoritmilise andmetöötluse korrektsust mitte lekitada infot. Sarnaselt teistele programmianalüüsi valdkondadele, kasutatakse seega ka siin eelkõige programmeerimiskeelte semantikaga seotud vahendeid.

Käesoleva töö esimeses peatükis selgitame informatsiooni voo ja informatsiooni kanalite mõisteid. Teises peatükis kirjeldame informatsiooni voogude turvalisust ja infolekete võimalusi programmeerimiskeele tasemel. Kolmandas peatükis esitleme võremudeli kasutamist informatsiooni voogude määratlemiseks [Den76]. Neljandas peatükis tutvustame lühidalt mehhanismi infovoogude turvalisuse kontrollimiseks [DD77] ning viiendas peatükis demonstreerime kuidas on samal eesmärgil võimalik kasutada tüübisüsteemi [VSI96].

1 Informatsiooni voog ja lekked

Informatsiooniteoreetilises mõttes nimetatakse informatsiooni vooks ehk infovooks objektist x objekti y nähtust kui objektis x sisalduv informatsioon kandub objekti y või seda kasutatakse objekti y kantava informatsiooni tuletamiseks. Tähistame seda informatsiooni voogu $x \Rightarrow y$. Ütleme, et programmi lause määrab infovoo $x \Rightarrow y$, kui selle lause käivitamisel mingis kontekstis võib toimuda informatsiooni voog $x \Rightarrow y$.

Mehhanisme, mille abil informatsioon arvutussüsteemides levib, nimetatakse kanaliteks. Üheks informatsiooni voo kanaliks on näiteks omistamisoperatsioon. Infovoo kanaleid, mis kasutavad mehhanisme, mille põhiline eesmärk pole informatsiooni kandmine, nimetatakse peitkanaliteks (*covert channels*) [PE08]. Näiteks võib programm olla ebaturvaline ka juhul kui selle salajastest muutujatest sõltub programmi termineeruvus või mittetermineeruvus, programmi täitmiseks kulunud mõõdetav aeg, riistvaralised kõrvalmõjud (voolutarve, müratase), ressursside (mälu, kettaruum) ammendumine jne. Lisaks võib eksisteerida tõenäosuslikke peitkanaleid, mille korral sõltub avalike andmete tõenäosuslik jaotus salajastest andmetest.

Turvapoliitikat, mis näeb ette, et avalikud andmed ei sõltu salajastest andmetest, kirjeldab mittemõjutatavuse (*noninterference*) omadus [Kas03, PE08], mille kohaselt ei tohi salajane informatsioon kuidagi mõjutada avalikku informatsiooni. Tavaliselt näidatakse mittemõjutatavuse omaduse kehtivust tõestades, et pole võimalik eristada kahte programmi käivitust, mis erinevad üksteisest ainult salajaste sisendite poolest [PE08].

Olgu programmi sisendid ja väljundid jagatud kahte erinevasse turvaklassi – avalikku ja salajasse. Ütleme, et programmil on turvaline infovoog, kui avalikud väljundid ei anna informatsiooni salajaste sisendite kohta [Kas04]. Vastasel juhul esineb programmis infolekkeid, ehk informatsioon salajastest sisenditest levib avalikesse väljunditesse.

Vaatleme järgnevalt mõningaid põhilisi info lekkimise võimalusi. Piirdume juhuga, kus programmil on ainult kaks muutujat – avalik muutuja $l \in L$ ja salajane muutuja tähisega $h \in H$ – mis on mõeldud vastavasse turvaklassi kuuluva info hoidmiseks. Eeldame, et need muutujad kannavad programmi vastavaid sisendeid ja väljundeid.

Programmis

$$l := h \tag{1}$$

omistatakse salajase muutuja h algväärtus avaliku muutuja l lõppväärtuseks, mistõttu see programm on ebaturvaline, sest avalikku turvaklassi kuuluvasse väljundisse satub informatsioon salajasse turvaklassi kuuluva informatsiooni kohta. Programmi 1 informatsiooni voogu $h \Rightarrow l$ nimetame ilmutatud infovooks [Kas03], sest informatsiooni voog on ilmne omistamisoperatsiooni semantika tõttu.

Erinevalt programmist 1 on programm

$$h := l$$

turvaline, sest selles ei ole keelatud infovoogu.

Turvaline programm võib sisaldada ka mitteturvalist alamprogrammi, nagu on programmi

$$l := h$$
$$l := 0$$

korral, mille avalikud väljundid ei sisalda informatsiooni salajase informatsiooni kohta, ent avalikke muutujaid kasutatakse vastava informatsiooni ajutiseks hoidmiseks. Taoliste programmide puhul sõltub tegelik infolekke võimalus sellest, kas avalikke muutujaid on programmi töö käigus või programmi töö katkestamisel võimalik vaadelda – kui avalike muutujate väärtusi on võimalik vaadelda ainult programmi töö lõppemisel, on programm turvaline, vastasel juhul mitte.

Infovoog võib esineda ka ilmutamata kujul [Den76, Kas03], näiteks programmis

$$\textbf{if } h = l \textbf{ then } l := 0 \textbf{ else } l := 1, \tag{2}$$

mis on ebaturvaline hoolimata sellest, et mõlemad tingimuslause harud on turvalised alamprogrammid. Ilmutamata infovood $x \Rightarrow y$ toimuvad, kui programmi lauses on mingi infovoog $z \Rightarrow y$ kuid selle lause täitmine/mittetäitmine sõltub x väärtusest. Üldiselt võivad kõik tingimuslaused ilmutamata infovooge tekitada.

Juhul kui muutujatel h ja l on ainult kaks võimalikku väärtust (0 ja 1), siis programm 2 lekitab kogu info salajase muutuja h kohta – programmi töö lõppedes on võimalik avaliku muutuja l väärtusest tuletada salajase muutuja h väärtus. Kui muutujate h ja l võimalikke väärtuste hulgad T_1 ja T_2 on suuremad, võib informatsioon h kohta lekkida ainult osaliselt. Näiteks kui $T_1 = T_2 = \{0, 1, 2\}$, ning programmi käivitamisel $l = 0$ ja $h \neq l$, siis ei leki salajase muutuja h täpne väärtus, vaid ainult informatsioon, et h ei ole 0, ehk informatsioon, et h võib olla kas 1 või 2.

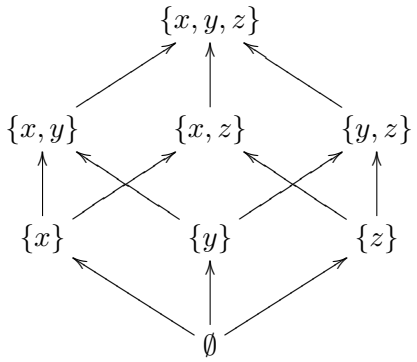
Osaliselt lekkinud informatsiooni võib olla võimalik akumuleerida täielikku lekke saavutamiseks, kui programmi saab erinevate avalike sisenditega korduvalt jooksutada. Näiteks kui programmi 2 korral $|T_1 \cap T_2| \geq |T_1| - 1$, siis salajase muutuja h täpse väärtuse tuvastamiseks piisab programmi jooksutamisest maksimaalselt $|T_1| - 1$ korral, varieerides sisendmuutujat l hulgal $T_1 \cap T_2$. Teades muutuja h väärtuste esinemise tõenäosusjaotusi hulgal T_1, on võimalik oluliselt vähendada h täpse väärtuse leidmiseks tehtavate katsete keskmist arvu. Viimast lähenemist võidakse kasutada näiteks jõumeetodil paroole ära arvates (nn *brute-force* rünnete puhul).

2 Turvaklasside võremudel

Mitmetel juhtudel on tarvilik informatsiooni jaotada enamatesse turvaklassidesse kui avalikku ja salajasse. Näiteks võib informatsiooni vastavalt selle sisule jaotada klassidesse x, y ja z. Näiteks võib leiduda osapool, kellel on õigus käidelda salastatud informatsiooni turvaklassist x, ning osapool, kellel on õigus käidelda nii turvaklassi x kui ka turvaklassi z kuuluvat salastatud informatsiooni.

Informatsiooni voogude turvaliseks määratlemiseks pakkus Dorothy Denning informatsiooni klassifitseerimiseks välja lõplike võrede [Den76] kasutamise, mille struktuur modelleerib hästi infovoogude semantikat. Võre klassifitseerib turvatasemed kõrgemast madalamani, ning määrab ka info liikumise lubatud suunad ($\rightarrow$) turvatasemete vahel. Võremudeli infovood on refleksiivsed (st $\forall \underline{x} \in SC \quad \underline{x} \rightarrow \underline{x}$) ja transitiivsed (st kui $\forall \underline{x}, \underline{y}, \underline{z} \in SC$ puhul $\underline{x} \rightarrow \underline{y}$ ja $\underline{y} \rightarrow \underline{z}$ siis ka $\underline{x} \rightarrow \underline{z}$). Võremudelit järgides on võimalik programmide turvalisi infovooge sertifitseerida staatilise analüüsi abil.

Joonisel 1 on toodud üks võimalik turvaklasside võremudel informatsiooni klasside x, y ja z korral, kus turvaklasside hulgaks on $SC = \mathcal{P}(\{x, z, y\})$, kõrgeimaks turvaklassiks $\{x, y, z\}$, madalaimaks turvaklassiks $\emptyset$ ning lubatud infovooge määravaks osaliseks järjestusseoseks $\subseteq$.



Joonis 1: Lubatud infovoogude võremudel turvaklasside $\mathcal{P}(\{x, y, z\})$ korral. Joonisel pole näidatud refleksiivseid infovooge, st infovooge turvaklassist iseendasse.

Joonis 2: Lubatud infovoogude võremudel kahe turvaklassi korral: salajane turvaklass H ja avalik turvaklass L. Võremudeleid, mille kõik turvaklassid on ühes ahelas, nimetatakse ka lineaarseteks.

Mudeli kohaselt on infovoog turvaklassist A turvaklassi B ehk $A \to B$ lubatud parajasti siis kui $A \subseteq B$. Näiteks võib turvaklassi $\{x, z\}$ kuuluv osapool käidelda turvaklassi $\{x\}$ kuuluvat informatsiooni, kuid mitte vastupidi. Joonistel 2 on kujutatud turvaklasside võremudel kahe turvaklassiga. Võremudelite korral tähistatakse madalaimat turvaklassi tüüpiliselt tähega L (*low*) ning kõrgeimat turvaklassi tähega H (*high*).

Kui turvaklasside hulk SC ei ole piisav, et $\langle SC, \to \rangle$ oleks võre, siis võib võrestruktuuri saavutamiseks lisada uusi turvaklasse ja infovooge, sealjuures olemasolevaid infovooge muutmata.

Tähistagu muutujate $A = \{a_1, \ldots, a_n\}$ turvaklasse $\underline{A} = \{\underline{a_1}, \ldots, \underline{a_n}\}$. Olgu f n-aarne funktsioon ning binaarne, assotsiatiivne ja kommutatiivne operaator $\oplus$ nii, et

$$\forall f \quad \underline{a_1} \oplus \ldots \oplus \underline{a_n} = \underline{f(a_1, \ldots, a_n)},$$

ehk operaator $\oplus$ väljendaks suvaliste funktsioonide rakendamisel tekkiva tulemuse turvaklassi. Suvalise funktsiooni tulemus võib sisaldada informatsiooni funktsioonile ette antud parameetrite kohta, seetõttu peab tulemuse turvaklass võremudeli kohaselt olema vähemalt sama kõrge kui tema parameetrite turvaklassid. Defineerime ka unaarse $\oplus$ operaatori hulkadel:

$$\oplus X = \begin{cases} L, & \text{kui } X = \emptyset \\ x_1 \oplus \ldots \oplus x_n, & \text{kui } X = \{x_1, \ldots, x_n\} \end{cases}$$

Programmi loeme turvaliseks, kui kõik programmi infovood on turvalised, ehk programmi iga infovoo $x \Rightarrow y$ korral kehtib $\underline{x} \to \underline{y}$. Programmi

$$b := f(a_1, \ldots, a_n) \tag{3}$$

loeme turvaliseks parajasti siis kui $\forall i \in \{1, \ldots, n\}$ $\underline{a_i} \to \underline{b}$. Infovoogude võremudeli kasutamisel piisab programmi 3 puhul näidata, et kehtib $\underline{a_1} \oplus \ldots \oplus \underline{a_n} \to \underline{b}$.

On näidatud [Den76], et $\oplus$ on väljapakutud turvaklasside võre SC alamraja operaatoriks. Sarnaselt operaatorile $\oplus$ on tähistatud operaatoriga $\otimes$ ülemraja operaatorit ning seda laiendatud hulkadele:

$$\otimes X = \begin{cases} H, & \text{kui } X = \emptyset \\ x_1 \otimes \ldots \otimes x_n, & \text{kui } X = \{x_1, \ldots, x_n\}, \end{cases}$$

Lisaks on näidatud, et $X = \{B_1, \ldots, B_n\}$ korral $\forall i \in \{1, \ldots, n\}$ $A \to B_i$ parajasti siis, kui $A \to \otimes X$, ehk informatsioon muutujast a võib voolata muutujatesse $b_1, \ldots, b_n$ parajasti siis, kui $\underline{a} \to \underline{b_1} \otimes \ldots \otimes \underline{b_n}$. Kõrgeimat ja madalaimat turvaklassi on vastavalt tähistatud $H = \oplus SC$ ja $L = \otimes SC$

3 Sertifitseerimine

Kui muutujate turvaklassid on programmis deklareeritud ning staatilised, st ei muutu programmi töö käigus, siis on lihtne sertifitseerida programmi turvalisust staatilise analüüsi abil, näiteks programmi kompileerimisel. Sertifitseerimiseks võib kasutada Peter ja Dorothy Denningu välja pakutud sertifitseerimissemantikat [DD77], mida kompilaator võib üheskoos teiste semantiliste toimingutega (näiteks tüübikontrolli või koodi genereerimisega) süntaktilisi konstruktsioone läbides teostada.

Piirdume sertifitseerimismehhanismide selgitamisel lihtsa programmeerimiskeelega, mille kõik muutujad on täisarvud ning ainukesteks avaldise tüüpideks on liitmine ja võrdsuse kontroll. Joonisel 3 on toodud selle programmeerimiskeele süntaktilised- ja sertifitseerimise reeglid. Sertifitseerimist alustatakse eeldades, et programm on turvaline ning sertifitseerimist teostatakse vastava programmi abstraktsel süntaksipuul. Programmi loetakse turvaliseks kui sertifitseerimine ei tagasta vigu. Toodud programmeerimiskeele semantika kohaselt deklareeritakse muutujaid ja neile vastavat turvaklassi (turvatüüpi) konstruktsiooni **let** ⟨sectype⟩ ⟨ident⟩ **in** ⟨stmt′⟩ abil, mis sätestab ka deklareeritava muutuja skoobiks lause ⟨stmt′⟩.

Kirjeldatud programmeerimiskeele programmide iga infovoo korral, mis toimub muutujatest (või sisenditest) $a_1, \ldots, a_n$ muutujatesse (või väljunditesse) $b_1, \ldots, b_n$, võib näha, et joonisel 3 toodud sertifitseerimismehhanism kontrollib lõpuks seda, kas kehtib tingimus

$$\underline{a_1} \oplus \ldots \oplus \underline{a_n} \rightarrow \underline{b_1} \otimes \ldots \otimes \underline{b_n}. \tag{4}$$

Tüüpiliselt leidub imperatiivsetes programmeerimiskeeltes omistamislause mingite avaldiste väärtuste omistamiseks muutujatele. Kui muutujale a omistatakse muutujaid $b_1, \ldots, b_n$ sisalduva avaldise väärtus, siis peame sertifitseerimisel kontrollima, et iga $i \in \{1, \ldots, n\}$ korral kehtib $\underline{b_i} \rightarrow \underline{a}$. Kasutades võremudelit piisab kui kontrollida, et $\underline{b_1} \oplus \ldots \oplus \underline{b_n} \rightarrow \underline{a}$. Sama kehtiks ka juhul kui muutuja a asemel oleks mingi väljundkanal, näiteks fail.

Kui programmeerimiskeeles leiduks ka vahend sisendiks saamiseks mingist sisendkanalist, näiteks lause **input** $a_1, \ldots, a_n$ **from** b, peaks sertifitseerimismehhanism selle korral kontrollima, et $\underline{b} \rightarrow \underline{a_i}$ iga $i \in \{1, \ldots, n\}$ korral. Võremudelit kasutades oleks seegi ülesanne lihtsustatud, sest peaksime kontrollima vaid, et kehtib $\underline{b} \rightarrow \underline{a_1} \otimes \ldots \otimes \underline{a_n}$.

Tüüpilised lihtsad tsüklid programmeerimiskeeltes sisaldavad avaldist, mis kontrollib tsükli täitmist, ning sisu, mida täidetakse selle avaldise kehtimisel. Lihtsa sertifitseerimise puhul peab tuvastama, millised selle avaldise muutujad $a_1, \ldots, a_n$ mõjutavad tsükli täitmist, ja millistesse muutujatesse $b_1, \ldots, b_n$ liigub informatsioon tsükli sisus, ning kontrollima avaldise 4 kehtivust.

Massiivide puhul kasutatakse programmeerimiskeeltes tihti viiteid (*references*) massiivide elementidele, nagu näiteks *list*[12] viitamaks massiivi „list“ 12-ndale elemendile. Sertifitseerimisel nende viidete puhul eristama kahte juhtu. Esiteks, kui viidet kasutatakse massiivi elemendi muutmiseks, näiteks süntaksireegli ⟨ident⟩[⟨expr⟩] ::= ⟨expr′⟩ korral, siis tuleb veenduda, et massiivi ⟨ident⟩ oleks lubatud infovoog nii avaldisest ⟨expr′⟩ kui ka avaldisest ⟨expr⟩, et oleks täidetud mittemõjutatavuse omadus. Turvaklasside võremudeli korral piisab, kui kontrollida, et kehtiks ⟨expr⟩ ⊕ ⟨expr′⟩ → ⟨ident⟩.

Teisel juhul kasutatakse viidet massiivi elemendile selle elemendi lugemiseks mingi avaldise osana. Kui lugemisel vastab viitele süntaksireegel ⟨arrayref⟩ ::= ⟨ident⟩[⟨expr⟩], siis selle turvaklassiks oleks ⟨arrayref⟩ := ⟨ident⟩ ⊕ ⟨expr⟩, sest siingi sõltub loetav väärtus nii massiivist kui ka kasutatavast indeksist.

Käsitleme järgnevalt ka protseduuride väljakutseid. Olgu meil protseduur q formaalsete sisendparameetritega $x_1, \ldots, x_n$, formaalsete väljundparameetritega $y_1, \ldots, y_m$ ning mis kõrvalefektina kannab informatsiooni muutujatesse $c_1, \ldots, c_p$, mis sisaldavad ka väljundparameetreid. Protseduuri q väljakutset loetakse sertifitseerimisel turvaliseks kui

Süntaksireegel	**Sertifitseerimise reegel**
$\langle \text{sectype} \rangle ::= \mathbf{L} \mid \mathbf{H}$	
$\langle \text{var} \rangle ::= \langle \text{ident} \rangle$	$\langle \underline{\text{var}} \rangle ::= \langle \underline{\text{ident}} \rangle$
$\langle \text{term} \rangle ::= \langle \text{const} \rangle$	$\langle \underline{\text{term}} \rangle := L$ (alati vähim turvaklass)
$\langle \text{term} \rangle ::= \langle \text{var} \rangle$	$\langle \underline{\text{term}} \rangle := \langle \underline{\text{var}} \rangle$
$\langle \text{term} \rangle ::= (\langle \text{expr} \rangle)$	$\langle \underline{\text{term}} \rangle := \langle \underline{\text{expr}} \rangle$
$\langle \text{addexpr} \rangle ::= \langle \text{term} \rangle$	$\langle \underline{\text{addexpr}} \rangle := \langle \underline{\text{term}} \rangle$
$\langle \text{addexpr} \rangle ::= \langle \text{addexpr} \rangle + \langle \text{term} \rangle$	$\langle \underline{\text{addexpr}} \rangle := \langle \underline{\text{addexpr}} \rangle \oplus \langle \underline{\text{term}} \rangle$
$\langle \text{expr} \rangle ::= \langle \text{addexpr} \rangle$	$\langle \underline{\text{expr}} \rangle := \langle \underline{\text{addexpr}} \rangle$
$\langle \text{expr} \rangle ::= \langle \text{addexpr} \rangle = \langle \text{addexpr}' \rangle$	$\langle \underline{\text{expr}} \rangle := \langle \underline{\text{addexpr}} \rangle \oplus \langle \underline{\text{addexpr}'} \rangle$
$\langle \text{stmt} \rangle ::= \langle \text{var} \rangle := \langle \text{expr} \rangle$	Kui $\langle \underline{\text{expr}} \rangle \rightarrow \langle \underline{\text{var}} \rangle$ siis $\langle \underline{\text{stmt}} \rangle := \langle \underline{\text{var}} \rangle$, vastasel juhul viga
$\langle \text{stmt} \rangle ::= \{\langle \text{stmts} \rangle\}$	$\langle \underline{\text{stmt}} \rangle := \langle \underline{\text{stmts}} \rangle$
$\langle \text{stmts} \rangle ::= \langle \text{stmt} \rangle$	$\langle \underline{\text{stmts}} \rangle := \langle \underline{\text{stmt}} \rangle$
$\langle \text{stmts} \rangle ::= \langle \text{stmts}' \rangle ; \langle \text{stmt} \rangle$	$\langle \underline{\text{stmts}} \rangle := \langle \underline{\text{stmts}'} \rangle \otimes \langle \underline{\text{stmt}} \rangle$
$\langle \text{stmt} \rangle ::= \mathbf{if} \langle \text{expr} \rangle \mathbf{then} \langle \text{stmt}' \rangle \mathbf{else} \langle \text{stmt}'' \rangle$	Kui $\langle \underline{\text{expr}} \rangle \rightarrow \langle \underline{\text{stmt}'} \rangle \otimes \langle \underline{\text{stmt}''} \rangle$ siis $\langle \underline{\text{stmt}} \rangle := \langle \underline{\text{stmt}'} \rangle \otimes \langle \underline{\text{stmt}''} \rangle$, vastasel juhul viga
$\langle \text{stmt} \rangle ::= \mathbf{while} \langle \text{expr} \rangle \mathbf{do} \langle \text{stmt}' \rangle$	Kui $\langle \underline{\text{expr}} \rangle \rightarrow \langle \underline{\text{stmt}'} \rangle$ siis $\langle \underline{\text{stmt}} \rangle := \langle \underline{\text{stmt}'} \rangle$, vastasel juhul viga
$\langle \text{stmt} \rangle ::= \mathbf{let}\ \langle \text{sectype} \rangle\ \langle \text{ident} \rangle\ \mathbf{in}\ \langle \text{stmt}' \rangle$	Seostab $\langle \text{stmt}' \rangle$-s $\langle \text{ident} \rangle$-ga turvaklassi $\langle \text{sectype} \rangle$; $\langle \underline{\text{stmt}} \rangle := \langle \underline{\text{stmt}'} \rangle$
$\langle \text{prog} \rangle ::= \langle \text{stmt} \rangle$	$\langle \underline{\text{prog}} \rangle := \langle \underline{\text{stmt}} \rangle$

Joonis 3: Sertifitseerimissemantika näide. Süntaktilise tüübi $\langle \text{x} \rangle$ turvaklassi oleme tähistanud $\langle \underline{\text{x}} \rangle$.

1. q on turvaline,
2. $\underline{a_i} \rightarrow \underline{x_i}$ iga $i \in \{1, \ldots, n\}$ korral, kus a_i on tegelik sisendparameeter,
3. $\underline{y_i} \rightarrow \underline{b_i}$ iga $i \in \{1, \ldots, m\}$ korral, kus b_i on tegelik väljundparameeter, ning
4. $\underline{e_1} \oplus \ldots \oplus \underline{e_r} \rightarrow \underline{c_1} \otimes \ldots \otimes \underline{c_p}$, kus $\underline{e_1}, \ldots, \underline{e_r}$ on avaldised, mis mille väärtusest sõltuvad tingimusdirektiivid või tsüklid, mille skoobis protseduuri väljakutse asub.

Protseduuride erijuhuks oleks funktsioonid, millel on ainult üks väljundparameeter, ning mis sisemiselt ei muuda ühtegi mittelokaalset muutujat. Funktsioonide väljakutsete korral piisaks ainult tingimuste 1-3, kontrollist.

Taoline lähenemine protseduuride kasutamisel tekitab aga probleemi juhtudel kui on tarvis protseduure välja kutsuda eri turvaklassidesse kuuluvate sisenditega. Seega peaksid ka protseduuri kõik formaalsed parameetrid tingimuse 2 tõttu kuuluma kõrgeimasse turvaklassi H. Üldjuhul sõltuvad protseduuri väljundparameetrite väärtused sisendparameetrite väärtustest, seega peaksid sel juhul ka protseduuri formaalsed väljundparameetrid kuuluma turvaklassi H, mis pole enamasti soovitud tulemus.

Abikaasad Denningud on sellele probleemile pakkunud välja kaks lahendust. Esiteks oleks võimalik valmistada protseduurist eri versioonid kõikide võimalike sisendväärtuste turvaklasside kombinatsioonide jaoks. Teiseks oleks võimalik protseduure piirata nii, et väljundparameetrid sõltuvad ainult sisendparameetritest ning protseduur ei oma kõrvalefekte, st ei kirjuta mittelokaalsetesse muutujatesse. Nende piirangute puhul piisaks sertifitseerimisel kontrollida, et kehtib

1. $\underline{a_1} \oplus \ldots \oplus \underline{a_n} \rightarrow \underline{b_1} \otimes \ldots \otimes \underline{b_m}$, ja
2. $\underline{e_1} \oplus \ldots \oplus \underline{e_r} \rightarrow \underline{b_1} \otimes \ldots \otimes \underline{b_m}$.

4 Tüübisüsteemid

Volpano, Smith ja Irvine [VSI96] leidsid, et Peter ja Dorothy Denning poolt esitatud sertifitseerimist turvaklasside võremudeli jaoks on võimalik abstraheerida ka tüübisüsteemina, mille kasutamine programmianalüüsis on ilmselt mugavam tänu olemasolevatele vahenditele. Seetõttu võib kaduda vajadus eraldiseisvaks analüüsiks – sertifitseerimisega seotud turvalisuse info võib integreerida tüübisüsteemi nii, et sertifitseerimine toimuks kui tüübikontroll. Nende esitatud tüübisüsteem jagunes kahekihiliseks: andmete turvatüübid τ ning fraaside turvatüübid ρ. Andmete turvatüübid kuuluvad turvaklasside SC hulka. Fraasitüübid sisaldavad nii andmete turvatüüpe τ (avaldiste tüüpimiseks), muutujate turvatüüpe $\tau\ var$ kui ka käskude turvatüüpe $\tau\ cmd$. Muutuja turvatüübiga $\tau\ var$ võib sisaldada ainult turvaklassi τ informatsiooni või mõnda sellest madalamasse turvaklassi kuuluvat informatsiooni. Programmi lause turvatüüp on $\tau\ cmd$ kui kõik muutujad, millesse selles lauses väärtusi omistatakse, kuuluvad turvaklassi $\tau\ var$ või mõnda kõrgemasse turvaklassi.

Turvaklassi võremudeli osalist järjestust $\rightarrow$, mis määrab informatsiooni voo lubatud suuna, on laiendatud fraaside turvatüüpidele ρ seosena $\subseteq$ vastavalt joonisele 4. Reegel (BASE) tähistab, et võremudeli osaline järjestus $\rightarrow$ kandub antud tüübisüsteemis otseselt üle andmete turvatüüpidele. Reegel (CMD) defineerib aga vastupidise seose lausete turvatüüpidele. Reeglid (REFLEX) ja (TRANS) määravad vastavalt fraaside turvatüüpide seose refleksiivsuse ja transitiivsuse. Kui turvatüüp ρ on turvatüübi ρ' alamtüüp, ehk $\rho \subseteq \rho'$, lubab reegel (SUBTYPE) tüüpimise γ korral fraasile p turvatüübiga ρ omistada ka turvatüübi ρ'.

$$\frac{\tau \rightarrow \tau'}{\vdash \tau \subseteq \tau'}\ (\text{BASE}) \qquad \frac{\vdash \rho \subseteq \rho' \qquad \vdash \rho' \subseteq \rho''}{\vdash \rho \subseteq \rho''}\ (\text{TRANS})$$

$$\frac{\vdash \tau \subseteq \tau'}{\vdash \tau'\ cmd \subseteq \tau\ cmd}\ (\text{CMD}) \qquad \frac{\gamma \vdash p : \rho \qquad \vdash \rho \subseteq \rho'}{\gamma \vdash p : \rho'}\ (\text{SUBTYPE})$$

$$\vdash \rho \subseteq \rho \quad (\text{REFLEX})$$

Joonis 4: Fraasitüüpide reeglid.

Joonisel 5 oleme toonud eelpool kirjeldatud lihtsale programmeerimiskeelele (joonis 3) vastavad turvalised tüüpimisreeglid. Vastavalt reeglile (CONST) anname kõikidele konstantidele madalaima turvaklassi, sest iseeneses nad ei sisalda turvalist informatsiooni ning neile ei saa omistada väärtusi. Reegli (SUBTYPE) abil jooniselt 4 võib konstante käsitleda ka kõrgemate turvaklasside objektidena. Reegel (VAR) seob igale muutuja nime esinemisega selle muutuja tüübi vastavalt tüüpimisele vastavas kontekstis.

Avaldiste tüüpimise kohta käiv reegel (ADD) koos fraasitüüpide reeglitega (joonis 4) sätestab, et kõikide avaldiste turvatüüp oleks kõikide avaldises loetavate informatsiooni kandvate osade (muutujad, konstandid jne) turvaklasside vähim ülemraja nagu joonisel 3 toodud ⟨addexpr⟩ reeglite puhul.

Reegliga (R-VAL) määratakse muutujanimede tüüpimine avaldise osana, milles loetakse nende väärtust. Reegel (ASSIGN) lubab mõnda muutujasse x omistada ainult $\gamma(x)$ või madalama turvaklassi informatsiooni. Siinkohal tuleb panna tähele, et antud tüüpimisreeglite korral pole võimalik muuta muutujate endi turvatüüpe, sest tegu pole dünaamiliselt tüübitud (turvatüübitud) programmeerimiskeelega.

Olgu meil sarnaselt peatükile 1 kaks muutujat l ja h, mis kuuluvad vastavalt avalikku ning salajasse turvaklassi: $\gamma(l) = L\ var$, $\gamma(h) = H\ var$. Lause $h := l + h$ turvatüübiks τ oleks sel juhul $H\ cmd$:

$$\gamma \vdash n : L \quad \text{(CONST)} \qquad \frac{\gamma \vdash x : \tau\ var \qquad \gamma \vdash e : \tau}{\gamma \vdash x := e : \tau\ cmd}\ \text{(ASSIGN)}$$

$$\frac{\gamma\,(x) = \tau\ var}{\gamma \vdash x : \tau\ var}\ \text{(VAR)} \qquad \frac{\gamma \vdash c : \tau\ cmd \qquad \gamma \vdash c' : \tau\ cmd}{\gamma \vdash c;\ c' : \tau\ cmd}\ \text{(COMPOSE)}$$

$$\frac{\gamma \vdash e : \tau \qquad \gamma \vdash e' : \tau}{\gamma \vdash e + e' : \tau}\ \text{(ADD)} \qquad \frac{\gamma \vdash e : \tau \qquad \gamma \vdash c : \tau\ cmd \qquad \gamma \vdash c' : \tau\ cmd}{\gamma \vdash \textbf{if}\ e\ \textbf{then}\ c\ \textbf{else}\ c' : \tau\ cmd}\ \text{(IF)}$$

$$\frac{\gamma \vdash e : \tau \qquad \gamma \vdash e' : \tau}{\gamma \vdash e = e' : \tau}\ \text{(COMPARE)} \qquad \frac{\gamma \vdash e : \tau \qquad \gamma \vdash c : \tau\ cmd}{\gamma \vdash \textbf{while}\ e\ \textbf{do}\ c : \tau\ cmd}\ \text{(WHILE)}$$

$$\frac{\gamma \vdash e : \tau\ var}{\gamma \vdash e : \tau}\ \text{(R-VAL)} \qquad \frac{\gamma[x : \tau\ var] \vdash c : \tau'\ cmd}{\gamma \vdash \textbf{let}\ \tau\ x\ \textbf{in}\ c : \tau'\ cmd}\ \text{(LETVAR)}$$

Joonis 5: Turvaliste tüüpimisreeglite näide.

$$\dfrac{\dfrac{\gamma(h) = H\ var}{\gamma \vdash h : H\ var}\ \text{(VAR)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\gamma(l) = L\ var}{\gamma \vdash l : L\ var}\ \text{(VAR)}}{\gamma \vdash l : L}\ \text{(R-VAL)} \qquad \vdash L \subseteq H}{\gamma \vdash l : H}\ \text{(SUBTYPE)} \qquad \dfrac{\dfrac{\gamma(h) = H\ var}{\gamma \vdash h : H\ var}\ \text{(VAR)}}{\gamma \vdash h : H}\ \text{(R-VAL)}}{\gamma \vdash l + h : H}\ \text{(ADD)}}{\gamma \vdash h := l + h : H\ cmd}\ \text{(ASSIGN)}$$

Sealjuures tuleb tähele panna, et antud lause turvatüüp oli sõltumatu tüübituletuspuu juure parempoolsest harust, ent kui oleks osutunud võimatuks lausele vastavat tüübituletuspuud konstrueerida, tähendanuks see, et lause $h := l + h$ ei oleks tüübitav (ehk ebaturvaline käesoleva töö raames).

Arvestades fraasitüüpide reegleid – eriti (SUBTYPE) ja (CMD) – määrab reegel (COMPOSE) lausete järjest täitmise tüübi vastavalt madalamale turvaklassile, mis tüüpi muutujasse toimub mõni kirjutamine neis lausetes.

Reeglid (IF) ja (WHILE) määravad, et alamprogrammides c ja c' ei omistata informatsiooni madalama turvaklassiga muutujatesse kui tingimusavaldises e loetud informatsioon. Seega on tagatud, et ei toimu mitte-lubatud informatsiooni voogu ilmutamata kujul. Näiteks programm

$$\textbf{if}\ x = 0\ \textbf{then}\ y := 0\ \textbf{else}\ y := 1$$

on tüübitav parajasti siis kui $\gamma(x) \subseteq \gamma(y)$. Kui antud programm on tüübitav ning $\gamma(x) \neq \gamma(y)$, siis peab tüübituletuspuu konstrueerimiseks kasutama reeglit (SUBTYPE) kas tingimuse $x = 0$ tüüpimiseks kõrgemas turvaklassis või tingimusdirektiivi harude tüüpimiseks madalamas turvaklassis. Viimane lähenemine on võimalik, sest reegli (CMD) tõttu on käskude turvatüübid vastupidi järjestatud. Tüüpides tingimusdirektiivi harud madalamas turvaklassis, on ka tingimusdirektiivi tüüp madalas turvaklassis. Näide:

$$\dfrac{\dfrac{\ldots}{\gamma \vdash l = 0 : L} \qquad \dfrac{\dfrac{\ldots}{\gamma \vdash h := 0 : H\ cmd}}{\gamma \vdash h := 0 : L\ cmd}\ \text{(SUBTYPE)} \qquad \dfrac{\dfrac{\ldots}{\gamma \vdash h := 1 : H\ cmd}}{\gamma \vdash h := 1 : L\ cmd}\ \text{(SUBTYPE)}}{\gamma \vdash \textbf{if}\ l = 0\ \textbf{then}\ h := 0\ \textbf{else}\ h := 1 : L\ cmd}\ \text{(IF)}$$

Tüüpides tingimuse $l = 0$ kõrgemas turvaklassis saame:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\gamma(l) = L\ var}{\gamma \vdash l : L\ var}\ \text{(VAR)}}{\gamma \vdash l : L}\ \text{(R-VAL)} \qquad \gamma \vdash 0 : L}{\gamma \vdash l = 0 : L}\ \text{(COMPARE)}}{\gamma \vdash l = 0 : H}\ \text{(SUBTYPE)} \qquad \dfrac{\ldots}{\gamma \vdash h := 0 : H\ cmd} \qquad \dfrac{\ldots}{\gamma \vdash h := 1 : H\ cmd}}{\gamma \vdash \textbf{if}\ l = 0\ \textbf{then}\ h := 0\ \textbf{else}\ h := 1 : H\ cmd}\ \text{(IF)}$$

Kokkuvõte

Käesolevas töös tõime põgusa ülevaate infovoogude turvalisusest ning tutvustasime ka infolekete tekkimise võimalusi programmeerimiskeelte semantika seisukohast. Samas kontekstis kirjeldasime ühe lihtsa programmeerimiskeele näitel infovoogude turvalisuse kontrollimiseks kasutatavat sertifitseerimise mehhanismi ning sama eesmärki täitvat tüübisüsteemi. Seeläbi demonstreerisime kahe suures osas erineva metoodika sarnasust.

Viited

[Den76] D. E. Denning. *A Lattice Model of Secure Information Flow.* Communications of the ACM, köide 19, väljaanne 5, lk 236-243. Mai, 1976.

[DD77] D. E. Denning, P. J. Denning. *Certification of Programs for Secure Information Flow.* Communications of the ACM, köide 20, väljaanne 7, lk 504-513. Juuli, 1977.

[Kas03] K. Kasak. *Turvaline infovoog.* Semestritöö, Tartu Ülikool, Arvutiteaduse instituut. Mai, 2003.

[Kas04] K. Kasak. *Relatiivselt turvaline infovoog.* Bakalaureusetöö, Tartu Ülikool, Arvutiteaduse instituut. Mai, 2004.

[PE08] M. Pistoia, Ú. Erlingsson. *Programming Languages and Program Analysis for Security: A Three-Year Retrospective.* ACM SIGPLAN Notices, köide 43, väljaanne 12, lk 32-39. Detsember, 2008.

[VSI96] D. Volpano, G. Smith, C. Irvine. *A Sound Type System for Secure Flow Analysis.* Journal of Computer Security, köide 4, väljaanne 3, lk 167-187. Detsember, 1996.