Security Through Safety

Benedikt Nordhoff

Westfälische Wilhelms-Universität
Münster, Germany

In the first section, we will define the program model and security property we are aiming at. The second section contains a characterization of critical executions which must exist for the security property to be violated and has been formalized in the interactive theorem prover Isabelle/HOL. This is then used in the last section to construct a safety analysis that asserts the security property.

1 Security Property

For the definition of our program model let Var be a finite set of variables and Val be an arbitrary set of values. The states on which our programs operate are maps from variables to values $\Sigma = \mathrm{Var} \to \mathrm{Val}$. We assume the confidential input is part of the initial state as values of a set $H \subseteq \mathrm{Var}$ of initially *high* variables. We call the remainder $L = \mathrm{Var} \setminus H$ initially *low* variables. For an arbitrary set $X \subseteq \mathrm{Var}$ of variables and two states $\sigma, \sigma' \in \Sigma$ we denote by $\sigma =_X \sigma'$ the pointwise equality of σ and σ' for all $x \in X$ and in the special case that $X = L$ we call the states σ and σ' *low equal*. We use control flow graphs for our program model.

Definition 1 (Program model). *A control flow graph is a tuple* $(N, E, \mathfrak{e}, \mathfrak{r}, [\![.]\!])$ *where* N *is a finite set program locations,* $[\![.]\!] \colon N \times \Sigma \to N \times \Sigma$ *is the deterministic semantics function,* $E \subseteq N \times N$, *is a safe control flow abstraction i.e.* $\forall n, n'.\ \exists \sigma, \sigma'.\ [\![n, \sigma]\!] = (n', \sigma') \Rightarrow (n, n') \in E$. $\mathfrak{e}, \mathfrak{r} \in N$ *are the initial respectively final location and we require, that* $N \times \{\mathfrak{r}\} \subseteq E^*$, $\forall n.\ (\mathfrak{r}, n) \in E \Rightarrow n = \mathfrak{r}$ *and* $\forall \sigma. [\![\mathfrak{r}, \sigma]\!] = (\mathfrak{r}, \sigma)$.

For a location state tuple $x \in N \times \Sigma$, we denote by x_l (respectively x_s) the projection to the program location (respectively the state). We assume that we have two auxiliary functions $\mathrm{use} \colon N \to 2^{\mathrm{Var}}$, which models the sets of variables used, and $\mathrm{def} \colon N \to 2^{\mathrm{Var}}$, which model the sets of variables defined by an instruction in the canonical way. That is $\mathrm{def}(n) \supseteq \{v \in \mathrm{Var} \mid \exists \sigma.\ \sigma(v) \neq [\![n, \sigma]\!]_s(v)\}$ and $\mathrm{use}(n) \subseteq \mathrm{Var}$, such that for all σ, σ' that satisfy $\sigma =_{\mathrm{use}(n)} \sigma'$ it holds $[\![n, \sigma]\!]_l = [\![n, \sigma']\!]_l$ and $[\![n, \sigma]\!]_s =_{\mathrm{def}(n)} [\![n, \sigma']\!]_s$.

Definition 2 (Execution). *A formal execution is a path* $\pi = (\pi_i)_{0 \leq i} \in N^\omega$ *where* $\pi_0 = \mathfrak{e}$ *and for all* $0 < i$ *it holds that* $(\pi_{i-1}, \pi_i) \in E$. *It is called feasible from some state* σ_0 *if* $\forall i.\ \pi_i = [\![\pi_0, \sigma_0]\!]_l^i$ *and in this case* $([\![\pi_0, \sigma_0]\!]_s^i)_{0 \leq i}$ *is called the corresponding state sequence. An execution is called terminating if there exists an index* k *such that* $\pi_k = \mathfrak{r}$.

Definition 3 (Attacker). *An attacker is modeled as a uniquely defined relation (partial function) of observables* $\mathrm{obs} \subseteq N \times (\Sigma \to \mathrm{Val})$ *such that for all* $(n, f) \in \mathrm{obs}$ *the value of* f *only depends upon the values of the variables in* $\mathrm{use}(n)$*, that is:* $\forall \sigma, \sigma' \in \Sigma.\ \sigma =_{\mathrm{use}(n)} \sigma' \Rightarrow f(\sigma) = f(\sigma')$*. The observation made by the attacker from an initial state* σ_0 *is defined as follows. Let* $\pi = \pi_0\pi_1 \dots$ *be the execution corresponding to* σ_0*,* $(\sigma_i)_{1\leq i}$ *be the corresponding states and* $(i_j)_{1\leq j<l}$ *for* $l \in \mathbb{N} \cup \{\infty\}$ *be the maximal increasing family of indices such that* $\pi_{i_j} \in \mathrm{dom}(\mathrm{obs})$ *for all* j*. Then the sequence* $\mathrm{obs}(\sigma_0) := (\mathrm{obs}(\pi_{i_j})(\sigma_{i_j}))_{1\leq j<l}$ *is the observation made from* σ_0 *by the attacker* obs*.*

Definition 4 (Security). *A program is called* α^L_{obs}*-secure if for all pairs of low equivalent states* $\sigma =_L \sigma'$*, for which the corresponding maximal executions terminate, it holds that* $\mathrm{obs}(\sigma) = \mathrm{obs}(\sigma')$*.*

A program is called β^L_{obs}*-secure if for all pairs of low equivalent states* $\sigma =_L \sigma'$*, it holds that* $\mathrm{obs}(\sigma) \leq \mathrm{obs}(\sigma')$ *or* $\mathrm{obs}(\sigma') \leq \mathrm{obs}(\sigma)$*. Here* $\leq$ *is the prefix order on sequences.*

Note that $\alpha + \beta$ security corresponds to indistinguishable security [2]. We don't require that non-terminating executions produce the same observation, as this would require the analysis to prove termination of all loops influenced by high data.

2 Characterization

Let Π denote the set of infinite paths in E. That is $\Pi = \{(\pi_i)_{0\leq i} \mid \forall 0 < i.\ (\pi_{i-1}, \pi_i) \in E\}$. Let $\xrightarrow{pd} \subseteq N \times N$ be the post dominance relation, that is $n \xrightarrow{pd} m$ iff $\forall (\pi_i)_{0\leq i} \in \Pi, k.\ \pi_0 = m \wedge \pi_k = \mathfrak{r} \Rightarrow \exists i \leq k.\ \pi_i = n$. As we assumed, that $\mathfrak{r}$ can be reached from all locations and doesn't reach any other locations, the post dominance relation $\xrightarrow{pd}$ is the transitive reflexive closure of the post dominance tree rooted in $\mathfrak{r}$ and the immediate post dominator $\mathrm{ipd}(m)$ is well defined for all $m \in N \setminus \{\mathfrak{r}\}$ as the unique location $n \neq m$ such that $n \xrightarrow{pd} m$ and $\forall n'.\ n' \neq m \wedge n' \xrightarrow{pd} m \Rightarrow n' \xrightarrow{pd} n$.

We define control dependencies in the style of Xin & Zhang [3] based on regions which correspond to a path from a branching point up to the corresponding immediate post dominator and use the standard definition of data dependency. For a path $\pi \in \Pi$ the control dependency relation $\xrightarrow{cd_\pi} \subseteq \mathbb{N} \times \mathbb{N}$ is defined as follows $i \xrightarrow{cd_\pi} k$ iff $i < k \wedge \pi_k \neq \mathfrak{r} \wedge \forall j.\ i \leq j \leq k \Rightarrow \pi_j \neq ipd(\pi_i)$. With this we can define the control slice of index k in $\pi \in \Pi$ as $cs^\pi_k = (\pi_{i_j})_{0\leq j\leq l}$ where $(i_j)_{0\leq j\leq l}$ is the maximal increasing sequence of indices such that $i_l = k$ and for all $0 \leq j < j' \leq l$ $i_j \xrightarrow{cd_\pi} i_{j'}$. The control slice is useful to identify occurrences of the same location in different executions as it has the useful property: $\forall i, i', j, j' \in \mathbb{N}, \pi, \pi' \in \Pi.\ \mathrm{cs}^\pi_i = \mathrm{cs}^{\pi'}_{i'} \neq \mathfrak{r} \wedge \mathrm{cs}^\pi_j = \mathrm{cs}^{\pi'}_{j'} \wedge i < j \Rightarrow i' < j'$.

For a path $\pi \in \Pi$ and variable $v \in \mathrm{Var}$ the data dependency relation $\xrightarrow{dd_{\pi,v}} \subseteq \mathbb{N} \times \mathbb{N}$ is defined by $i \xrightarrow{dd_{\pi,v}} k$ iff $i < k \wedge v \in \mathrm{def}(\pi_i) \cap \mathrm{use}(\pi_k) \wedge \forall j \in [i+1, k-1].\ v \notin \mathrm{def}(\pi_j)$.

We use the following characterization to identify critical executions. For a state σ let $\pi^\sigma = (\pi^\sigma_i)_{0 \leq i}$ denote the corresponding path with state sequence $(\sigma_i)_{0 \leq i}$. We define the the set of conflict pairs $cp \subseteq (\Sigma \times \mathbb{N})^2$ as the smallest relation closed under rules defined in equations (1) – (4).

$$\frac{\begin{array}{c} \sigma =_L \sigma' \wedge \mathrm{cs}^{\pi^\sigma}_i = \mathrm{cs}^{\pi^{\sigma'}}_{i'} \wedge \\ h \in \mathrm{use}(\pi^\sigma_i) \wedge \sigma_i(h) \neq \sigma'_{i'}(h) \wedge \\ \forall j < i.\ h \notin \mathrm{def}(\pi^\sigma_j) \wedge \\ \forall j' < i'.\ h \notin \mathrm{def}(\pi^{\sigma'}_{j'}) \end{array}}{(\sigma, i)\ cp\ (\sigma', i')} \tag{1}$$

$$\frac{\begin{array}{c} (\sigma, j)\ cp\ (\sigma', j') \wedge \mathrm{cs}^{\pi^\sigma}_i = \mathrm{cs}^{\pi^{\sigma'}}_{i'} \wedge \\ j \xrightarrow{dd_{\pi,v}} i \wedge j' \xrightarrow{dd_{\pi',v}} i' \wedge \\ \sigma_i(v) \neq \sigma'_{i'}(v) \end{array}}{(\sigma, i)\ cp\ (\sigma', i')} \tag{2}$$

$$\frac{\begin{array}{c} (\sigma, j)\ cp\ (\sigma', j') \wedge \mathrm{cs}^{\pi^\sigma}_i = \mathrm{cs}^{\pi^{\sigma'}}_{i'} \wedge \\ j \xrightarrow{cd_\pi} k \wedge k \xrightarrow{dd_{\pi,v}} i \wedge \\ \pi^\sigma_{j+1} \neq \pi^{\sigma'}_{j'+1} \wedge \sigma_i(v) \neq \sigma'_{i'}(v) \wedge \\ \forall l' \in [\inf\{\iota' | j' < \iota' \wedge \exists \iota.\ \mathrm{cs}^{\pi^\sigma}_\iota = \mathrm{cs}^{\pi^{\sigma'}}_{\iota'}\}, i' - 1].\ v \notin \mathrm{def}(\pi^{\sigma'}_{l'}) \end{array}}{(\sigma, i)\ cp\ (\sigma', i')} \tag{3}$$

$$\frac{(\sigma', i')\ cp\ (\sigma, i)}{(\sigma, i)\ cp\ (\sigma', i')} \tag{4}$$

We obtain the property, that, if the executions for two low equal states reach the same control slice but read different values for some variable, then they create a conflicting pair.

Lemma 1. $\sigma =_L \sigma' \wedge \mathrm{cs}^{\pi^\sigma}_i = \mathrm{cs}^{\pi^{\sigma'}}_{i'} \wedge \sigma_i \neq_{\mathrm{use}(\pi^\sigma_i)} \sigma'_{i'} \Rightarrow (\sigma, i)\ cp\ (\sigma', i')$

Based on this we define observable conflict pairs as those, that either both reached an observable node or where one has but the other can not reach an observable node. We define the the set of observable conflict pairs $cop \subseteq (\Sigma \times \mathbb{N})^2$ as the smallest set closed under the following rules:

$$\frac{(\sigma, i)\ cp\ (\sigma', i') \wedge \pi^\sigma_i \in \mathrm{dom}(\mathrm{obs})}{(\sigma, i)\ cop\ (\sigma', i')} \tag{5}$$

$$\frac{(\sigma, j)\ cp\ (\sigma', j') \wedge j \xrightarrow{cd_\pi} i \wedge \pi^\sigma_{j+1} \neq \pi^{\sigma'}_{j'+1} \wedge \pi^\sigma_i \in \mathrm{dom}(\mathrm{obs})}{(\sigma, i)\ cop\ (\sigma', j')} \tag{6}$$

Theorem 1. *If there do not exist any states σ, σ' and indices i, i' such that $\sigma =_L \sigma'$ and $(\sigma, i)\ cop\ (\sigma', i')$ then the program is α^L_{obs} and β^L_{obs} secure.*

3 Analysis

Inspecting equations (1) – (6) we can obtain a sound approximation for individual potentially critical executions. Equation (1) is handled by the initialization $I_\pi = \{i \in \mathbb{N} | \exists h \in H \cap \mathrm{use}(\pi_i).\ \forall j < i.\ h \notin \mathrm{def}(\pi_j)\}$. To handle the executions from equation (2) we can use the previously defined data dependencies. In equation (3) the execution π^σ is captured by control dependencies plus data dependencies and to capture execution $\pi^{\sigma'}$ we use data control dependencies defined in equation (7) below.

$$i \xrightarrow{dcd_{\pi,v}} k \Leftrightarrow \exists \pi' i' j' k'.\ \mathrm{cs}^{\pi}_{i} = \mathrm{cs}^{\pi'}_{i'} \wedge \mathrm{cs}^{\pi}_{k} = \mathrm{cs}^{\pi'}_{k'} \wedge i' \xrightarrow{cd_{\pi'}} j' \xrightarrow{dd_{\pi',v}} k' \qquad (7)$$

Theorem 2. *If there exists no feasible execution π and index i such that $\pi_i \in \mathrm{dom}(\mathrm{obs})$ and $I_\pi \times \{i\} \cap (\bigcup\{\xrightarrow{cd_\pi}, \xrightarrow{dd_{\pi,v}}, \xrightarrow{dcd_{\pi,v}} | v \in \mathrm{Var}\})^* \neq \emptyset$ then the program is α^L_{obs} and β^L_{obs} secure.*

Our prototypical implementation within the CPAchecker framework [1] corresponds to a finite automaton that uses five types of states to track control, data and data control dependencies within an execution and marks potentially critical states as target states who's reachability is then tried to be refuted by different analyses in the CPAchecker framework.

The analysis starts with the special state **Init**. The state **Target** marks a potentially critical state. States of the form $\mathbf{DD(v)}$ are used to track a data dependency on the variable v, the state $\mathbf{CD(m)}$ tracks a control dependency that stretches up to node m which is the immediate post dominator of some branching node n that introduced this control dependency, and the state $\mathbf{DCD(m, v)}$ denotes a data control dependency where m is the immediate post dominator of the branching node that introduced this data control dependency and v is the variable that could have be written on an alternative branch as identified by a static preanalysis.

The transfer relation on these states is defined by $s \xrightarrow{n} t \Leftrightarrow Crit(s, n, t) \wedge Intro(t, n, s) \vee s = t \wedge P(s, n)$ where $Crit$, $Intro$ and P are defined as in Table 1.

Table 1. Rules for the transfer relation of the prototypical implementation.

$\mathbf{s}$	$\mathbf{Intro(s, n, f)}$	$\mathbf{Crit(s, n, t)}$	$\mathbf{P(s, n)}$
Init	$false$	$H \cap \mathrm{use}(n) \neq \emptyset$	$true$
$\mathbf{DD(v)}$	$v \in \mathrm{def}(n) \vee isDCD(v, f)$	$v \in \mathrm{use}(n)$	$v \notin \mathrm{def}(n)$
$\mathbf{CD(m)}$	$m = \mathrm{ipd}(n)$	$m \neq n$	$m \neq n$
$\mathbf{DCD(m, v)}$	$m = \mathrm{ipd}(n) \wedge \exists \xrightarrow{cd_v}$ before m	$m = n \wedge isDD(v, t)$	$m \neq n$
Target	n is observable	$false$	$false$

References

1. Beyer, D., Keremoglu, M.E.: Cpachecker: A tool for configurable software verification. In: Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings. pp. 184–190 (2011), https://doi.org/10.1007/978-3-642-22110-1_16
2. Bohannon, A., Pierce, B.C., Sjöberg, V., Weirich, S., Zdancewic, S.: Reactive noninterference. In: Proceedings of the 16th ACM Conference on Computer and Communications Security. pp. 79–90. CCS '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1653662.1653673
3. Xin, B., Zhang, X.: Efficient online detection of dynamic control dependence. In: Proceedings of the 2007 international symposium on Software testing and analysis. pp. 185–195. ISSTA '07, ACM, New York, NY, USA (2007), http://doi.acm.org/10.1145/1273463.1273489