

Secret-Directed Unwinding

Brijesh Dongol Matt Griffin Andrei Popescu
 Jamie Wright

March 19, 2025

Abstract

This entry formalizes the secret-directed unwinding disproof method for relative security. The method was presented in the CSF 2023 paper “Relative Security: Formally Modeling and (Dis)Proving Resilience Against Semantic Optimization Vulnerabilities” [1]. Secret-directed unwinding can be used to prove the existence of transient execution vulnerabilities.

The main characteristics of secret-directed unwinding are that (1) it is used to disprove rather than prove security and (2) it proceeds in a manner that is “directed” by given sequences of secrets. The second characteristic is shared with the unwinding method for bounded-deducibility security [2].

Contents

1	Finitary Secret-Directed Unwinding	1
2	Secret-Directed Unwinding	3

1 Finitary Secret-Directed Unwinding

This theory formalizes the finitary version of secret-directed unwinding, which allows one to disprove finitary relative security.

theory *SD-Unwinding-fin*

imports *Relative-Security.Relative-Security*

begin

context *Rel-Sec*

begin

fun *validEtransO* **where** *validEtransO* (*s,secl*) (*s',secl'*) \longleftrightarrow
validTransV (*s,s'*) \wedge
(\neg *isSecV* *s* \wedge *secl* = *secl'* \vee
isSecV *s* \wedge *secl* = *getSecV* *s* # *secl'*)

definition $move1 \Gamma sv1 secl1 sv2 secl2 \equiv$
 $\forall sv1' secl1'. validEtransO (sv1, secl1) (sv1', secl1') \longrightarrow \Gamma sv1' secl1' sv2 secl2$

definition $move2 \Gamma sv1 secl1 sv2 secl2 \equiv$
 $\forall sv2' secl2'. validEtransO (sv2, secl2) (sv2', secl2') \longrightarrow \Gamma sv1 secl1 sv2' secl2'$

definition $move12 \Gamma sv1 secl1 sv2 secl2 \equiv$
 $\forall sv1' secl1' sv2' secl2'.$
 $validEtransO (sv1, secl1) (sv1', secl1') \wedge validEtransO (sv2, secl2) (sv2', secl2')$
 $\longrightarrow \Gamma sv1' secl1' sv2' secl2'$

definition $unwindSDCond ::$
 $('stateV \Rightarrow 'secret list \Rightarrow 'stateV \Rightarrow 'secret list \Rightarrow bool) \Rightarrow bool$

where

$unwindSDCond \Gamma \equiv \forall sv1 secl1 sv2 secl2.$

$reachV sv1 \wedge reachV sv2 \wedge$

$\Gamma sv1 secl1 sv2 secl2$

\longrightarrow

$(isIntV sv1 \longleftrightarrow isIntV sv2) \wedge$

$(\neg isIntV sv1 \longrightarrow move1 \Gamma sv1 secl1 sv2 secl2 \wedge move2 \Gamma sv1 secl1 sv2 secl2) \wedge$

$(isIntV sv1 \longrightarrow getActV sv1 = getActV sv2 \longrightarrow getObsV sv1 = getObsV sv2 \wedge$
 $move12 \Gamma sv1 secl1 sv2 secl2)$

proposition $unwindSDCond-aux:$

assumes $unw: unwindSDCond \Gamma$

and $1: \Gamma sv1 secl1 sv2 secl2$

$reachV sv1 \text{ Van. validFromS } sv1 (trv1 \text{ completedFromV } sv1 trv1$

$reachV sv2 \text{ Van. validFromS } sv2 (trv2 \text{ completedFromV } sv2 trv2$

$\text{Van. S } trv1 = secl1 \text{ Van. S } trv2 = secl2$

$\text{Van. A } trv1 = \text{Van. A } trv2$

shows $\text{Van. O } trv1 = \text{Van. O } trv2$

$\langle proof \rangle$

proposition $unwindSDCond-aux-strong:$

assumes $unw: unwindSDCond \Gamma$

and $1: \Gamma sv1 secl1 sv2 secl2$

$reachV sv1 \text{ Van. validFromS } sv1 (trv1 @ [trn1]) \text{ never isIntV } trv1$ **and** $11: isIntV$
 $trn1$ **and**

$2: reachV sv2 \text{ Van. validFromS } sv2 (trv2 @ [trn2]) \text{ never isIntV } trv2$ **and** $22:$
 $isIntV trn2$ **and**

$3: \text{Van. S } trv1 @ ssecl1 = secl1 \text{ Van. S } trv2 @ ssecl2 = secl2$

shows $\Gamma (lastt sv1 trv1) ssecl1 (lastt sv2 trv2) ssecl2$

<proof>

lemma *S-eq-empty-ConsE*:

assumes $\langle \text{Van.S } (x \# xs) = \text{Opt.S } (y \# ys) \rangle$ **and** $\langle xs \neq [] \rangle$ **and** $\langle ys \neq [] \rangle$
shows $\langle (\text{isSecO } y \wedge \text{isSecV } x \longrightarrow \text{getSecV } x = \text{getSecO } y \wedge \text{Van.S } xs = \text{Opt.S } ys) \rangle$

$\wedge (\text{isSecO } y \wedge \neg \text{isSecV } x \longrightarrow \text{Van.S } xs = (\text{getSecO } y \# \text{Opt.S } ys))$
 $\wedge (\neg \text{isSecO } y \wedge \text{isSecV } x \longrightarrow (\text{getSecV } x \# \text{Van.S } xs) = \text{Opt.S } ys)$
 $\wedge (\neg \text{isSecO } y \wedge \neg \text{isSecV } x \longrightarrow \text{Van.S } xs = \text{Opt.S } ys) \rangle$

<proof>

theorem *unwindSD-rsecure*:

assumes $tr14: \text{istateO } s1 \text{ Opt.validFromS } s1 \text{ } tr1 \text{ completedFromO } s1 \text{ } tr1$
 $\text{istateO } s4 \text{ Opt.validFromS } s4 \text{ } tr2 \text{ completedFromO } s4 \text{ } tr2$
 $\text{Opt.A } tr1 = \text{Opt.A } tr2 \text{ Opt.O } tr1 \neq \text{Opt.O } tr2$
and $\text{init}: \bigwedge sv1 \ sv2. \llbracket \text{istateV } sv1; \text{corrState } sv1 \ s1; \text{istateV } sv2; \text{corrState } sv2 \ s4 \rrbracket \Longrightarrow$

$\Gamma \ sv1 \ (\text{Opt.S } tr1) \ sv2 \ (\text{Opt.S } tr2)$

and $\text{unw}: \text{unwindSDCond } \Gamma$

shows $\neg \text{rsecure}$

<proof>

end

end

2 Secret-Directed Unwinding

This theory formalizes the secret-directed unwinding disproof method for relative security.

theory *SD-Unwinding*

imports *Relative-Security.Relative-Security*

begin

context *Rel-Sec*

begin

fun lvalidEtransO **where** $\text{lvalidEtransO } (s, \text{secl}) \ (s', \text{secl}') \longleftrightarrow$

$\text{validTransV } (s, s') \wedge$

$(\neg \text{isSecV } s \wedge \text{secl} = \text{secl}' \vee$

$\text{isSecV } s \wedge \text{secl} = \text{LCons } (\text{getSecV } s) \ \text{secl}')$

definition $\text{lmove1 } \Gamma \ sv1 \ \text{secl1} \ sv2 \ \text{secl2} \equiv$

$\forall \ sv1' \ \text{secl1}'. \ \text{lvalidEtransO } (sv1, \text{secl1}) \ (sv1', \text{secl1}') \longrightarrow \Gamma \ sv1' \ \text{secl1}' \ sv2 \ \text{secl2}$

definition $lmove2 \Gamma sv1 secl1 sv2 secl2 \equiv$
 $\forall sv2' secl2'. lvalidEtransO (sv2, secl2) (sv2', secl2') \longrightarrow \Gamma sv1 secl1 sv2' secl2'$

definition $lmove12 \Gamma sv1 secl1 sv2 secl2 \equiv$
 $\forall sv1' secl1' sv2' secl2'.$
 $lvalidEtransO (sv1, secl1) (sv1', secl1') \wedge lvalidEtransO (sv2, secl2) (sv2', secl2')$
 $\longrightarrow \Gamma sv1' secl1' sv2' secl2'$

definition $lunwindSDCond ::$
 $('stateV \Rightarrow 'secret\ llist \Rightarrow 'stateV \Rightarrow 'secret\ llist \Rightarrow bool) \Rightarrow bool$
where
 $lunwindSDCond \Gamma \equiv \forall sv1 secl1 sv2 secl2.$
 $reachV sv1 \wedge reachV sv2 \wedge$
 $\Gamma sv1 secl1 sv2 secl2$
 \longrightarrow
 $(isIntV sv1 \longleftrightarrow isIntV sv2) \wedge$
 $(\neg isIntV sv1 \longrightarrow lmove1 \Gamma sv1 secl1 sv2 secl2 \wedge lmove2 \Gamma sv1 secl1 sv2 secl2)$
 \wedge
 $(isIntV sv1 \wedge getActV sv1 = getActV sv2 \longrightarrow getObsV sv1 = getObsV sv2 \wedge$
 $lmove12 \Gamma sv1 secl1 sv2 secl2)$

lemma $lunwindSDCond\text{-}imp:$
assumes $lunwindSDCond \Gamma reachV sv1 reachV sv2 \Gamma sv1 secl1 sv2 secl2$
shows
 $(isIntV sv1 \longleftrightarrow isIntV sv2) \wedge$
 $(\neg isIntV sv1 \longrightarrow lmove1 \Gamma sv1 secl1 sv2 secl2 \wedge lmove2 \Gamma sv1 secl1 sv2 secl2)$
 \wedge
 $(isIntV sv1 \wedge getActV sv1 = getActV sv2 \longrightarrow getObsV sv1 = getObsV sv2 \wedge$
 $lmove12 \Gamma sv1 secl1 sv2 secl2)$
 $\langle proof \rangle$

lemma $lunwindSDCond\text{-}lmove12:$
assumes $unw: lunwindSDCond \Gamma$ **and** $gam: reachV sv1 reachV sv2 \Gamma sv1 secl1$
 $sv2 secl2$
and $i: isIntV sv1 \longrightarrow getActV sv1 = getActV sv2$
shows $lmove12 \Gamma sv1 secl1 sv2 secl2$
 $\langle proof \rangle$

proposition $unwindSDCond\text{-}aux\text{-}inductive:$
assumes $unw: lunwindSDCond \Gamma$
and $1: \Gamma sv1 secl1 sv2 secl2$
 $reachV sv1 \text{ Van. validFromS } sv1 (trv1 @ [ssv1]) \text{ never } isIntV trv1$ **and** $11: isIntV$
 $ssv1$ **and**
 $2: reachV sv2 \text{ Van. validFromS } sv2 (trv2 @ [ssv2]) \text{ never } isIntV trv2$ **and** $22: isIntV$
 $ssv2$ **and**
 $3: lappend (l\text{list-of } (Van.S\ trv1))\ ssecl1 = secl1\ lappend (l\text{list-of } (Van.S\ trv2))$

$ssecl2 = secl2$
shows Γ ($lastt\ sv1\ trv1$) $ssecl1$ ($lastt\ sv2\ trv2$) $ssecl2$
 $\langle proof \rangle$

proposition *unwindSDCond-inductive*:

assumes unw : $lunwindSDCond\ \Gamma$

and gam : $\Gamma\ sv1\ secl1\ sv2\ secl2$ **and**

$trv1$: $reachV\ sv1\ Van.validFromS\ sv1\ (trv1\ @\ [ssv1])\ never\ isIntV\ trv1\ isIntV\ ssv1$

and

$trv2$: $reachV\ sv2\ Van.validFromS\ sv2\ (trv2\ @\ [ssv2])\ never\ isIntV\ trv2\ isIntV\ ssv2$

and

s : $lappend\ (l\ list\ of\ (map\ getSecV\ (filter\ isSecV\ trv1)))\ ssecl1 = secl1$

$lappend\ (l\ list\ of\ (map\ getSecV\ (filter\ isSecV\ trv2)))\ ssecl2 = secl2$

shows ($getActV\ ssv1 = getActV\ ssv2 \longrightarrow \Gamma\ ssv1\ ssecl1\ ssv2\ ssecl2$) \wedge

($getActV\ ssv1 = getActV\ ssv2 \longrightarrow getObsV\ ssv1 = getObsV\ ssv2$)

$\langle proof \rangle$

proposition *lunwindSDCond-aux*:

assumes unw : $lunwindSDCond\ \Gamma$

and 1 : $\Gamma\ sv1\ secl1\ sv2\ secl2$

$reachV\ sv1\ Van.lvalidFromS\ sv1\ trv1\ lcompletedFromV\ sv1\ trv1$

$reachV\ sv2\ Van.lvalidFromS\ sv2\ trv2\ lcompletedFromV\ sv2\ trv2$

$Van.lS\ trv1 = secl1\ Van.lS\ trv2 = secl2$

$Van.lA\ trv1 = Van.lA\ trv2$

shows $Van.lO\ trv1 = Van.lO\ trv2$

$\langle proof \rangle$

theorem *unwindSD-lrsecure*:

assumes $tr14$: $istateO\ s1\ Opt.lvalidFromS\ s1\ tr1\ lcompletedFromO\ s1\ tr1$

$istateO\ s2\ Opt.lvalidFromS\ s2\ tr2\ lcompletedFromO\ s2\ tr2$

$Opt.lA\ tr1 = Opt.lA\ tr2\ Opt.lO\ tr1 \neq Opt.lO\ tr2$

and $init$: $\bigwedge sv1\ sv2. istateV\ sv1 \implies corrState\ sv1\ s1 \implies istateV\ sv2 \implies corrState\ sv2\ s2 \implies$

$\Gamma\ sv1\ (Opt.lS\ tr1)\ sv2\ (Opt.lS\ tr2)$

and unw : $lunwindSDCond\ \Gamma$

shows $\neg lrsecure$

$\langle proof \rangle$

end

end

References

- [1] A. P. Brijesh Dongol, Matt Griffin and J. Wright. Relative security: Formally modeling and (dis)proving resilience against semantic optimization vulnerabilities. In *37th IEEE Computer Security Foundations Symposium, CSF 2024*. To appear.

- [2] A. Popescu, T. Bauereiss, and P. Lammich. Bounded-Deducibility security (invited paper). In L. Cohen and C. Kaliszyk, editors, *12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference)*, volume 193 of *LIPICs*, pages 3:1–3:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.