Secret-Directed Unwinding

Brijesh Dongol Matt Griffin Andrei Popescu
Jamie Wright

June 5, 2024

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

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'*) ⟷
validTransV (*s*,*s'*) ∧
(¬ *isSecV s* ∧ *secl* = *secl'* ∨
isSecV s ∧ *secl* = *getSecV s* # *secl'*)

definition *move1* Γ *sv1 secl1 sv2 secl2* $\equiv$
$\forall$ *sv1′ secl1′*. *validEtransO* (*sv1*,*secl1*) (*sv1′*,*secl1′*) $\longrightarrow$ Γ *sv1′ secl1′ sv2 secl2*

definition *move2* Γ *sv1 secl1 sv2 secl2* $\equiv$
$\forall$ *sv2′ secl2′*. *validEtransO* (*sv2*,*secl2*) (*sv2′*,*secl2′*) $\longrightarrow$ Γ *sv1 secl1 sv2′ secl2′*

definition *move12* Γ *sv1 secl1 sv2 secl2* $\equiv$
$\forall$ *sv1′ secl1′ sv2′ secl2′*.
validEtransO (*sv1*,*secl1*) (*sv1′*,*secl1′*) $\wedge$ *validEtransO* (*sv2*,*secl2*) (*sv2′*,*secl2′*)
$\longrightarrow$ Γ *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* $\wedge$
Γ *sv1 secl1 sv2 secl2*
$\longrightarrow$
(*isIntV sv1* $\longleftrightarrow$ *isIntV sv2*) $\wedge$
($\neg$ *isIntV sv1* $\longrightarrow$ *move1* Γ *sv1 secl1 sv2 secl2* $\wedge$ *move2* Γ *sv1 secl1 sv2 secl2*) $\wedge$
(*isIntV sv1* $\longrightarrow$ *getActV sv1* = *getActV sv2* $\longrightarrow$ *getObsV sv1* = *getObsV sv2* $\wedge$
move12 Γ *sv1 secl1 sv2 secl2*)

proposition *unwindSDCond-aux*:
assumes *unw*: *unwindSDCond* Γ
and *1*: Γ *sv1 secl1 sv2 secl2*
reachV sv1 Van.validFromS sv1 trv1 completedFromV sv1 trv1
reachV sv2 Van.validFromS sv2 trv2 completedFromV sv2 trv2
Van.S trv1 = *secl1 Van.S trv2* = *secl2*
Van.A trv1 = *Van.A trv2*
shows *Van.O trv1* = *Van.O trv2*
⟨*proof*⟩

proposition *unwindSDCond-aux-strong*:
assumes *unw*: *unwindSDCond* Γ
and *1*: Γ *sv1 secl1 sv2 secl2*
reachV sv1 Van.validFromS sv1 (*trv1* @ [*trn1*]) *never isIntV trv1* **and** *11*: *isIntV trn1* **and**
2: *reachV sv2 Van.validFromS sv2* (*trv2* @ [*trn2*]) *never isIntV trv2* **and** *22*: *isIntV trn2* **and**
3: *Van.S trv1* @ *ssecl1* = *secl1 Van.S trv2* @ *ssecl2* = *secl2*
shows Γ (*lastt sv1 trv1*) *ssecl1* (*lastt sv2 trv2*) *ssecl2*

⟨*proof*⟩

lemma *S-eq-empty-ConsE*:
assumes ‹*Van.S* (*x* # *xs*) = *Opt.S* (*y* # *ys*)› **and** ‹*xs* ≠ []› **and** ‹*ys* ≠ []›
shows ‹(*isSecO y* ∧ *isSecV x* ⟶ *getSecV x* = *getSecO y* ∧ *Van.S xs* = *Opt.S ys*)
∧ (*isSecO y* ∧ ¬*isSecV x* ⟶ *Van.S xs* = (*getSecO y* # *Opt.S ys*))
∧ (¬*isSecO y* ∧ *isSecV x* ⟶ (*getSecV x* # *Van.S xs*) = *Opt.S ys*)
∧ (¬*isSecO y* ∧ ¬*isSecV x* ⟶ *Van.S xs* = *Opt.S ys*)›
⟨*proof*⟩

theorem *unwindSD-rsecure*:
assumes *tr14*: *istateO s1 Opt.validFromS s1 tr1 completedFromO s1 tr1*
istateO s4 Opt.validFromS s4 tr2 completedFromO s4 tr2
Opt.A tr1 = *Opt.A tr2 Opt.O tr1* ≠ *Opt.O tr2*
and *init*: ⋀*sv1 sv2*. ⟦*istateV sv1*; *corrState sv1 s1*; *istateV sv2*; *corrState sv2 s4*⟧ ⟹
Γ *sv1* (*Opt.S tr1*) *sv2* (*Opt.S tr2*)
and *unw*: *unwindSDCond* Γ
shows ¬ *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** *lvalidEtransO* (*s*,*secl*) (*s′*,*secl′*) ⟷
validTransV (*s*,*s′*) ∧
(¬ *isSecV s* ∧ *secl* = *secl′* ∨
isSecV s ∧ *secl* = *LCons* (*getSecV s*) *secl′*)

definition *lmove1* Γ *sv1 secl1 sv2 secl2* ≡
∀ *sv1′ secl1′*. *lvalidEtransO* (*sv1*,*secl1*) (*sv1′*,*secl1′*) ⟶ Γ *sv1′ secl1′ sv2 secl2*

definition *lmove2* Γ *sv1 secl1 sv2 secl2* $\equiv$
$\forall$ *sv2′ secl2′. lvalidEtransO* (*sv2*,*secl2*) (*sv2′*,*secl2′*) $\longrightarrow$ Γ *sv1 secl1 sv2′ secl2′*

definition *lmove12* Γ *sv1 secl1 sv2 secl2* $\equiv$
$\forall$ *sv1′ secl1′ sv2′ secl2′.*
lvalidEtransO (*sv1*,*secl1*) (*sv1′*,*secl1′*) $\wedge$ *lvalidEtransO* (*sv2*,*secl2*) (*sv2′*,*secl2′*)
$\longrightarrow$ Γ *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* $\wedge$
Γ *sv1 secl1 sv2 secl2*
$\longrightarrow$
(*isIntV sv1* $\longleftrightarrow$ *isIntV sv2*) $\wedge$
($\neg$ *isIntV sv1* $\longrightarrow$ *lmove1* Γ *sv1 secl1 sv2 secl2* $\wedge$ *lmove2* Γ *sv1 secl1 sv2 secl2*)
$\wedge$
(*isIntV sv1* $\wedge$ *getActV sv1* $=$ *getActV sv2* $\longrightarrow$ *getObsV sv1* $=$ *getObsV sv2* $\wedge$
lmove12 Γ *sv1 secl1 sv2 secl2*)

lemma *lunwindSDCond-imp*:
assumes *lunwindSDCond* Γ *reachV sv1 reachV sv2* Γ *sv1 secl1 sv2 secl2*
shows
(*isIntV sv1* $\longleftrightarrow$ *isIntV sv2*) $\wedge$
($\neg$ *isIntV sv1* $\longrightarrow$ *lmove1* Γ *sv1 secl1 sv2 secl2* $\wedge$ *lmove2* Γ *sv1 secl1 sv2 secl2*)
$\wedge$
(*isIntV sv1* $\wedge$ *getActV sv1* $=$ *getActV sv2* $\longrightarrow$ *getObsV sv1* $=$ *getObsV sv2* $\wedge$
lmove12 Γ *sv1 secl1 sv2 secl2*)
⟨*proof*⟩

lemma *lunwindSDCond-lmove12*:
assumes *unw*: *lunwindSDCond* Γ **and** *gam*: *reachV sv1 reachV sv2* Γ *sv1 secl1 sv2 secl2*
and *i*: *isIntV sv1* $\longrightarrow$ *getActV sv1* $=$ *getActV sv2*
shows *lmove12* Γ *sv1 secl1 sv2 secl2*
⟨*proof*⟩

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

ssecl2 = *secl2*
shows Γ (*lastt sv1 trv1*) *ssecl1* (*lastt sv2 trv2*) *ssecl2*
⟨*proof*⟩

proposition *unwindSDCond-inductive*:
assumes *unw*: *lunwindSDCond* Γ
and *gam*: Γ *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* (*llist-of* (*map getSecV* (*filter isSecV trv1*))) *ssecl1* = *secl1*
lappend (*llist-of* (*map getSecV* (*filter isSecV trv2*))) *ssecl2* = *secl2*
shows (*getActV ssv1* = *getActV ssv2* ⟶ Γ *ssv1 ssecl1 ssv2 ssecl2*) ∧
(*getActV ssv1* = *getActV ssv2* ⟶ *getObsV ssv1* = *getObsV ssv2*)
⟨*proof*⟩

proposition *lunwindSDCond-aux*:
assumes *unw*: *lunwindSDCond* Γ
and *1*: Γ *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*
⟨*proof*⟩

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* ≠ *Opt.lO tr2*
and *init*: ⋀*sv1 sv2*. *istateV sv1* ⟹ *corrState sv1 s1* ⟹ *istateV sv2* ⟹ *corrState sv2 s2* ⟹
Γ *sv1* (*Opt.lS tr1*) *sv2* (*Opt.lS tr2*)
and *unw*: *lunwindSDCond* Γ
shows ¬ *lrsecure*
⟨*proof*⟩

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.