

Minimizzazione di attacchi per acquisizione transittiva di fiducia in sistemi di installazione software

Giuseppe Primiero
(in collaborazione con Jaap Boender e Franco Raimondi)

Department of Computer Science
Middlesex University
London, UK



**Middlesex
University**

ZEI - Lecce

- 1 Su Fiducia e Computazione
- 2 Il problema dell'installazione minima
- 3 Il calcolo formale SecureND
- 4 La Libreria SecureNDC
- 5 Algoritmo per la soluzione

- 1 Su Fiducia e Computazione
- 2 Il problema dell'installazione minima
- 3 Il calcolo formale SecureND
- 4 La Libreria SecureNDC
- 5 Algoritmo per la soluzione

- Interazioni uomo-macchina basate sulla “fiducia” si trovano alla base di varie applicazioni: dalla verifica di programmi ai protocolli crittografici;
- Non e' inusuale che il comportamento computazionale resti nascosto agli utenti umani;
- Questo aumenta i rischi per gli utenti, per il valore dei risultati e per la stabilità e sicurezza dei sistemi.
- Protocolli computazionali per la fiducia hanno dunque bisogno di essere validati matematicamente per essere considerati sicuri

Modelli formali di sicurezza e reputazione con fiducia

- propagazione di fiducia, interferenza e bloccaggio, see e.g. [Goguen, Mesegur (1982), Carbone et al. (2003), Guha et al (2004), Ziegler, Lausen (2005), Marsh, Dibbem (2005), Josang, Pope (2005), Clarke et al. (2009), Chang et al. (2011)]
- fiducia in servizi internet, see e.g. [Grandison, Sloman (2000)]
- applicazioni: sistemi modulari a molteplici componenti [Yan, Prehofer (2011)], installazione software con reputazione [Bugiel (2011)], accuratezza [Ali et al. (2013)], affidabilita' epistemica [Zeller (2013)]
- larga letteratura in psicologia

Transitivá and Trasparenza

Le interazioni restano affidabili solo finché é possibile evitare relazioni di fiducia non intenzionali ([Christianson, Harbison (1996)]):

“Agente A ha fiducia nell’agente B, agente B ha fiducia nell’agente C, quindi agente A ha fiducia nell’agente C”.

Permettere tali relazioni puo' condurre evidentemente a risultati indesiderati.

- 1 Su Fiducia e Computazione
- 2 Il problema dell'installazione minima**
- 3 Il calcolo formale SecureND
- 4 La Libreria SecureNDC
- 5 Algoritmo per la soluzione

Lo scenario

Un utente interagisce con un sistema di gestione del software, e.g. `apt-get` su Debian; ogni operazione di (dis)installazione richiede di preservare la validità dell'attuale profilo di installazione:

Definition (Profilo di installazione valido)

L'insieme di pacchetti software su una macchina é chiamato il suo profilo di installazione. Un profilo valido soddisfa tutte le dipendenze ed evita tutti i conflitti per ognuno dei pacchetti installati (logicamente = soddisfacibilit ).

Il problema (versione originale [Tucker et al. (2007)])

Definition (Problema dell'installazione minima)

Determinare il modo di installare un nuovo pacchetto p tale che il minimo numero di dipendenze sia soddisfatto e conflitti siano evitati per garantire un profilo valido di installazione.

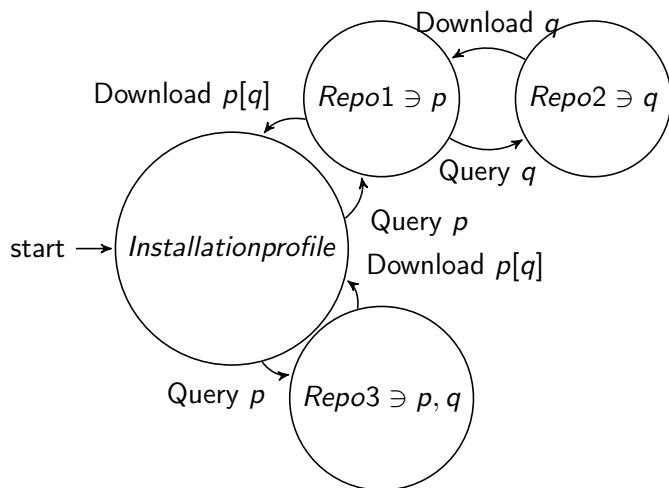
Il nuovo scenario

Consideriamo i due tipi di standard repositories software:

- *ufficiale*: per il sistema base e gli upgrades autorizzati;
- *trusted*: per pacchetti non inclusi nelle releases ufficiali.

Entrambi potrebbero usare software di terze parti. Sistemi sicuri richiedono in ogni caso privilegi da super-utente, ma raramente c'è un controllo rigido di tutte le relazioni transitive di fiducia.

Installazione di p , dipendente da libreria q



Definition (Problema dell'installazione minima con fiducia)

Determinare il modo di installare un nuovo pacchetto p tale che richieda il minimo numero di dipendenze transitive dalle repositories.

- 1 Su Fiducia e Computazione
- 2 Il problema dell'installazione minima
- 3 Il calcolo formale SecureND**
- 4 La Libreria SecureNDC
- 5 Algoritmo per la soluzione

- [Primiero, Raimondi (2014)] offre un calcolo di dimostrazioni che definisce un modello di controllo di accessi (*read/write*) con una funzione di fiducia (*trust*);
- SecureND risolve il problema della fiducia transitiva richiedendo che per ogni operazione di scrittura un atto di fiducia *esplicito* sulle risorse sia inserito nel protocollo di ogni agente;
- Per formalizzare il problema della installazione minima con fiducia, abbiamo tradotto il calcolo nel dimostratore interattivo di teoremi Coq e prodotto la libreria SecureNDC.

Definition

$$\mathcal{S} := \{A, B, \dots\}$$

$$\phi^S := a^A \mid \phi_1^A \rightarrow \phi_2^A \mid \phi_1^A \wedge \phi_2^A \mid \phi_1^A \vee \phi_2^A \mid \\ \text{Read}(\phi^A) \mid \text{Write}(\phi^A) \mid \text{Trust}(\phi^A)$$

$$\Gamma^A := \{\phi_1^A, \dots, \phi_n^A\}$$

Definition

Una formula $\Gamma^A \vdash \phi^B$ dice che in un profilo di installazione Γ contenente software da una repository A , é possibile (valida) un'operazione di accesso su un pacchetto ϕ di una repository B .

- pacchetti software sono accessibili tramite *Read* (query) and *Write* (installazione)
- Un'operazione *Import* aggiunge i pacchetti al profilo di installazione corrente, se soddisfano ad un criterio di consistenza (validità del profilo)
- *Trust* funziona da ponte tra privilegi di lettura e scrittura.

Definition (Trust)

Dato un profilo di installazione Γ^A , si assuma l'accesso sia valido ad un pacchetto ϕ da una repository B . Diciamo che la funzione di second'ordine *trust* caratterizza tale accesso se e solo se ϕ^B puo' essere importato nel profilo Γ^A preservandone la consistenza. Se tale accesso e' valido sotto fiducia, allora Γ^A e' autorizzato ad installare ϕ^B .

Non consideriamo per il momento dipendenze interne ad una repository, né inter-dipendenze tra due repositories. Le nostre repositories sono parzialmente ordinate:

Definition (Dependent Repositories)

Una relazione \leq su $\mathcal{S} \times \mathcal{S}$ è una relazione d'ordine parziale tale che $S \leq S'$ se e solo se esiste almeno un pacchetto $\phi^{S'}$ che abbia una relazione di dipendenza da un pacchetto ψ^S , mentre nessuna dipendenza esiste nella direzione inversa.

Esempio: Debian

- $\Gamma^{main} \vdash \phi^{main}$ esprime un'operazione su un pacchetto in *main*, eseguita da un profilo di installazione che contiene software della stessa repository;
- $\Gamma^{main} \vdash \phi^{non-free}$ esprime un'operazione su un pacchetto in *non-free*, eseguita da un profilo di installazione che contiene software da *main*;
- $\Gamma^{non-free} \vdash \phi^{main}$ esprime un'operazione su un pacchetto in *main*, eseguita da un profilo di installazione che contiene software da *non-free*.

- 1 Su Fiducia e Computazione
- 2 Il problema dell'installazione minima
- 3 Il calcolo formale SecureND
- 4 La Libreria SecureNDC**
- 5 Algoritmo per la soluzione

Coq in due slides I

- Proof-assistant per verificare che i programmi soddisfino le loro specifiche.
- Linguaggio di specifica: chiamato Gallina, i cui termini possono rappresentare programmi, loro proprietà o prove di queste.
- Isomorfismo di Curry-Howard: un programma con specifica e' equivalente ad una prova per una proposizione, formalizzati nello stesso linguaggio (Calcolo delle Costruzioni (Co)Induttive (CIC)),
- usato come un logical environment per la codifica di teorie matematiche.
- giudizi logici sono tipati: il type-checker verifica la correttezza delle prove, ovvero che i programmi soddisfino le rispettive specifiche.
- se CIC é consistente e il type-checker non contiene bug, allora ogni prova costruita in Coq sarà un termine che dimostra la tesi (type-inhabitation)

- ogni possibile estensione del sistema che non interessi il type-checker é sicura.
- il program extractor permette di sintetizzare programmi funzionali che obbediscano alle loro specifiche formali, scritte come asserzioni logiche in CIC.

Pacchetti Software (atomici)

```
(* Atom *)
Module Type ATOM (R: REPOSITORY) <: DecidableType.
  Definition t := R.t -> Type.

  Parameter eq: t -> t -> Prop.

  Axiom eq_refl: forall x: t, eq x x.
  Axiom eq_sym: forall x y: t, eq x y -> eq y x.
  Axiom eq_trans: forall x y z: t, eq x y -> eq y z -> eq x z.
  Instance eq_equiv : Equivalence eq :=
    Build_Equivalence t eq eq_refl eq_sym eq_trans.

  Parameter eq_dec: forall x y, { eq x y } + { ~ eq x y }.
End ATOM.
```


Pacchetti Software (composti)

(* The package type, defining the different connectives *)

```
Inductive Package {A: Type} {S: Type}: Type :=
```

```
| nd_atom: A -> Package
```

```
| nd_impl: Package -> Package -> Package
```

```
| nd_and: Package -> Package -> Package
```

```
| nd_or: Package -> Package -> Package
```

```
| nd_read: Package -> Package
```

```
| nd_write: Package -> Package
```

```
| nd_trust: Package -> Package.
```

Profili di Installazione

```
(* An installation profile is a set of resources typed by repository *)
```

```
Module Profile (R: REPOSITORY) (A: ATOM R).
```

```
  Module E := Package(R)(A).
```

```
  Include WSetsOn E.
```

```
End Profile.
```

```
(* a profile is typable if all its packages are typable *)
```

```
Definition typable_profile (R: Repository.t) (P: P.t): Prop :=
```

```
  forall f, In f P -> typable R f.
```

```
Add Morphism typable_profile with signature Repository.eq ==> P.eq  
  ==> Logic.iff as typable_profile
```

(* wellformedness *)

```
Definition well_formed (R: Repository.t) (P: P.t): Prop :=  
  ~Empty P /\ typable_profile R P.
```

Importare pacchetti

```
Axiom nd_import: forall Pa f,  
  NDProof (Pa::nil) (nd_read f) ->  
  typable Rb f ->  
  typable_profile Ra (P.add f Pa) ->  
  In f Pa.
```

- 1 Su Fiducia e Computazione
- 2 Il problema dell'installazione minima
- 3 Il calcolo formale SecureND
- 4 La Libreria SecureNDC
- 5** Algoritmo per la soluzione

Due obiettivi per la fiducia

- 1 provare che qualsiasi operazione di installazione può considerarsi sicura se fiducia e' garantita
- 2 provare che e' sempre possibile selezionare l'operazione di installazione piu' sicura, i.e. con il minimo numero di atti di fiducia.

Theorem (Eliminazione del taglio)

Un processo di installazione puo' essere considerato sicuro se

- 1 *o il profilo di installazione corrente appartiene ad una repository **dominante** su tutti pacchetti richiesti dall'installazione;*
- 2 *o un'operazione di fiducia **esplicita** e' garantita dal profilo di installazione corrente sulla relazione d'ordine per ogni pacchetto dominante richiesto dall'installaizone.*

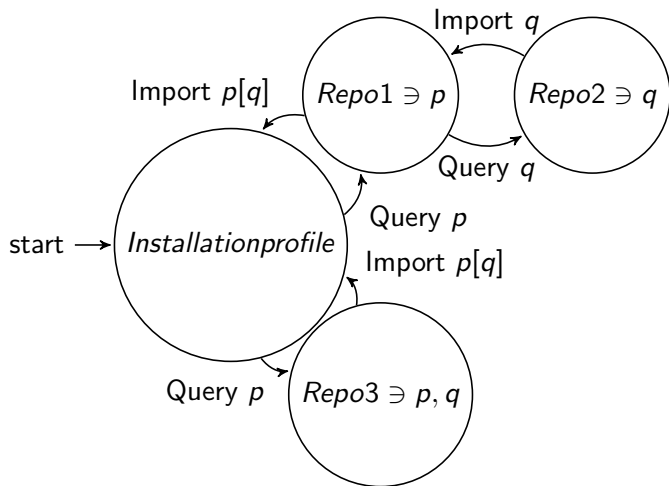
Secondo obiettivo: un algoritmo

- 1 associa ad ogni processo di installazione un numero naturale Proof
-> Nat
- 2 il numero indica quante volte un `import` con fiducia da una repository e' effettuato
- 3 installazioni locali hanno un valore 0
- 4 installazioni con piu' processi prendono la somma dei processi
- 5 query non aggiunge nulla
- 6 operazioni di fiducia aumentano il valore +1
- 7 installazione di un pacchetto p somma il valore della sorgente con quello del profilo locale

In altre parole...

- Possiamo formulare tutti i processi di installazione per un pacchetto p che usano repositories diverse
- per ciascuno computiamo il numero di volte che il processo richiede di importare un pacchetto da una nuova repository meno sicura (e quindi di controllarne la validita' con il profilo corrente);
- ordiniamoli secondo il valore computato;
- ovviamente meno volte questa operazione é eseguita, meno atti di fiducia sono richiesti, quindi meno rischiosa l'intera operazione.

Installazione di p , dipendente da libreria q



Definition




Dato un profilo di installazione Pa con pacchetti originati da una repository Ra e dato un pacchetto fb in un profilo Pb , formulare un processo di installazione ($NDUCProof(Pa :: Pb :: nil) \text{ nd_write } fb$) tale che il numero di istanze della regola nd_import su Pb da eliminare per ottenere una corrispondente derivazione $NDProof$ sia minimo.

Conclusioni

- Sistemi computazionali complessi richiedono un trattamento e l'implementazione di protocolli per la fiducia
- L'aspetto formale puo' garantire la validita' logica dell'algoritmo astratto
- La natura funzionale di Coq puo' essere usata per una traduzione in un linguaggio di programmazione come Haskell
- Esiste una controparte negativa da investigare (problema della disinstallazione)

Grazie! (Altre) Domande?

References I

-  N. Ali, Y.-G. Guéhéneuc, and G. Antoniol.
Trusttrace: Mining software repositories to improve the accuracy of requirement traceability links.
IEEE Trans. Software Eng., 39(5):725–741, 2013.
-  S. Bugiel, L. V. Davi, and S. Schulz.
Scalable trust establishment with software reputation.
In *Proceedings of the Sixth ACM Workshop on Scalable Trusted Computing, STC '11*, pages 15–24, New York, NY, USA, 2011. ACM.
-  J. Boender, G. Primiero, and F. Raimondi.
SecureNDC - Coq implementation of the SecureND calculus, October 2014.
<http://www.rmnd.net/MT/SecureND.v>.



S. Clarke, B. Christianson, and H. Xiao.

Trust*: Using local guarantees to extend the reach of trust.

In Bruce Christianson, James A. Malcolm, Vashek Matyas, and Michael Roe, editors, *Security Protocols Workshop*, volume 7028 of *Lecture Notes in Computer Science*, pages 171–178. Springer, 2009.



B. Christianson and W. S. Harbison.

Why isn't trust transitive?

In T. Mark A. Lomas, editor, *Security Protocols Workshop*, volume 1189 of *Lecture Notes in Computer Science*, pages 171–176. Springer, 1996.

References III



M. Carbone, M. Nielsen, and V. Sassone.

A formal model for trust in dynamic networks.

In A. Cerone and P. Lindsay, editors, *Int. Conference on Software Engineering and Formal Methods, SEFM 2003.*, pages 54–61. IEEE Computer Society, 2003.

A preliminary version appears as Technical Report BRICS RS-03-4, Aarhus University.







J. Chang, K. Venkatasubramanian, A. West, S. Kannan, B. Loo, O. Sokolsky, and I. & Lee.

As-trust: A trust quantification scheme for autonomous systems in bgp.

In *Trust and Trustworthy Computing: 4th International Conference, TRUST 2011*, volume 6740 of *Lecture Notes in Computer Science*, pages 262–276. Springer Berlin / Heidelberg, 2011.

References IV

-  R. Guha, R. Kumar, P. Raghavan, and A. Tomkins.
Propagation of trust and distrust.
In *Proceedings of the 13th International Conference on World Wide Web, WWW '04*, pages 403–412, New York, NY, USA, 2004. ACM.
-  J. A. Goguen and J. Meseguer.
Security policies and security models.
In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.
-  T. Grandison and M. Sloman.
A survey of trust in internet applications.
Communications Surveys Tutorials, IEEE, 3(4):2–16, Fourth 2000.
-  A. Jøsang and S. Pope.
Semantic constraints for trust transitivity.
In Sven Hartmann and Markus Stumptner, editors, *APCCM*, volume 43 of *CRPIT*, pages 59–68. Australian Computer Society, 2005.



D. H. Mcknight and N. L. Chervany.

The meanings of trust.

Technical report, MISRC Working Paper Series 96-04, University of Minnesota, Management Information Systems Research Center, 1996.



F. Mancinelli, J. Boender, R. Di Cosmo, J. Vouillon, B. Durak, X. Leroy, R. Treinen.

Managing the Complexity of Large Free and Open Source Package-Based Software Distributions.

ASE '06. 21st IEEE/ACM International Conference on Automated Software Engineering, 2006, pp.199–208.

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4019575&isnumber=4019544>.

References VI



S. Marsh and M.R. Dibben.

Trust, untrust, distrust and mistrust ? an exploration of the dark(er) side.

In Peter Herrmann, Valérie Issarny, and Simon Shiu, editors, *Trust Management*, volume 3477 of *Lecture Notes in Computer Science*, pages 17–33. Springer Berlin Heidelberg, 2005.



G. Primiero and F. Raimondi.

A typed natural deduction calculus to reason about secure messaging with trust.

IEEE Privacy Security and Trust 2014, pp.379-382.



C. Tucker, D. Shuffelton, R. Jhala, and S. Lerner.

Opium: Optimal package install/uninstall manager.

In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 178–188, 2007.

References VII



Z. Yan and C. Prehofer.

Autonomic trust management for a component-based software system.
IEEE Trans. Dependable Sec. Comput., 8(6):810–823, 2011.



A. Zeller.

Can we trust software repositories?

In Jürgen Münch and Klaus Schmid, editors, *Perspectives on the Future of Software Engineering*, pages 209–215. Springer Berlin Heidelberg, 2013.



C.-N. Ziegler and G. Lausen.

Propagation models for trust and distrust in social networks.

Information Systems Frontiers, 7(4-5):337–358, December 2005.