

Formal Methods: Techniques and tools based on mathematics and logic that support the specification, construction and analysis of hardware and software systems

Difficile definire cosa sia la sicurezza, è più semplice definire “da chi”

-> seguiamo meccanismi per prevenire un attacco che violerà policies tramite threat model

Possono fallire molte cose: policy sbagliate, assunzioni sull'utente o attaccante o meccanismi (“find my phone”).

Security Model: formato da un System Model, Threat Model e Security Properties.

➔ Security Analysis: processo per valutare se un sistema è designato in modo tale da poter raggiungere gli obiettivi di sicurezza preimpostati

I metodi formali possono aiutare ad analizzare sistemi complessi

I dati e le risorse computazionali hanno un valore (dipende dall'osservatore) -> Information Security

Può essere vista come policy compliant (rispetto delle policy) oppure risk minimization.

“Security is a pain because it stops you from doing things, and you have to do work to authenticate yourself and to set it up – Butler W. Lampson”

Attacker: find one weakness is enough

Defender: all weaknesses must be founded and eliminated

In SW Engineering specifici cosa devi fare

In sicurezza bisogna specificare quali comportamenti non sono ammessi

	Systems	Security
What is supposed to do?	Specifications	Policy
How does it do it?	Implementation	Mechanism
Does it really do it?	Correctness	Compliance

Formal Methods	Security
Specification φ	Security property φ
Program P	System P employing mechanisms
Correct: $P \models \varphi$	Secure: $P \models \varphi$

Moreover, φ should hold for P in all **malicious** environment E,

i.e. $P \parallel E \models \varphi$

The main difference: **active interference** from the environment

IS is CIA

Confidentiality: le informazioni non impropriamente divulgate

No accessi/letture non autorizzati

Integrity: le informazioni non impropriamente modificate

No modifiche non autorizzate

Availability: l'accesso alle informazioni non è impropriamente impedito ad utenti legittimi

No impedimenti d'accesso non autorizzati

Altre proprietà di sicurezza (casi speciali di CIA):

Authentication: identificare accuratamente i dati

Accountability: responsabilità, il tracciamento delle azioni -> *non-repudation*

Proprietà: alto livello, astratte

Policy: congiunzione di proprietà

Meccanismi: implementazioni, concreto per far rispettare le policy. Chiamati anche countermeasures.

Countermeasures:

Prevenzione:

Rilevazione: se fallisci la prevenzione

Response: restore di backup, etc. Attuare roba che ti fa tornare ad uno stato sicuro

Sicurezza come minimizzazione del rischio

Rischio = possibilità di successo * impatto

È una funzione dell'ambiente che cambia nel tempo.

Threat

Una vulnerabilità è una debolezza che può essere sfruttata da un attacco per causare un danno.

- 1) Identifica ciò che vuoi proteggere e rischi a riguardo, capendo chi sono i threat e threat agent.
- 2) Analizza soluzioni possibili, tradeoffs etc

Limiti della formalizzazione

La sicurezza non è una proprietà binaria, ha sfumature. È una combinazione non banale di proprietà che sono specifiche dell'applicazione che stiamo analizzando ed è difficile stabilire quali siano le proprietà più importanti e farle convivere tra loro (spesso confliggono). Un assioma è che un sistema non sarà mai sicuro al 100%: i metodi formali aiutano e caratterizzano più precisamente il comportamento del sistema e possono essere usati per dimostrare che il sistema soddisfa delle specifiche.

Come si applica un metodo formale?

Prima si costruisce un modello formale del comportamento del sistema. Poi si formalizzano i requisiti come proprietà di sicurezza e si verifica che il sistema soddisfa "queste cose qua"

Non necessariamente viene applicato a tutto il sistema, anche solo a parti di esso.

Cosa si vuole raggiungere vs Come si vuole raggiungere

Linguaggi formali per le specifiche e sfruttare la semantica di questi linguaggi per permettere la verifica. La validazione si fa con metodi matematici rigorosi.

.....

Language Based Security: features + mechanism in a PL to create secure apps

Trova giusto approccio tra flessibilità, velocità, controllo, sicurezza

Il SW è la maggiore sorgente di problemi di sicurezza, secondo solo al fattore umano

➔ Software security

Osservare le proprietà emergenti del sistema per intero.

Functionality: what SW should do. Security: what can't do

Vulnerability = è un flaw che viene sfruttato dall'attaccante se Accessibile e Sfruttabile

Language Based Security

Features garantite dal PL: memory / type / thread safety a vari livelli, Access Control: visibilità (public, private, ..) e sandboxing

In un programma safe abbiamo una semantica ben definita e quindi un comportamento ben definito e perciò possiamo vederlo in un modo modulare

Safety: protezione da incidenti

Security: protezione da cattivoni

Qualsiasi PL può essere usato per fare programmi non sicuri, ma il PL fa grossa differenza: senza safety è più difficile fare programmi sicuri.

Undefinedness

Molti costrutti dei linguaggi possono essere undefined in alcune circostanze e creare problemi. Un caso standard è il trovare loopholes, comportamenti dove la specifica non dice come dovrebbe andare: tipo fuzzing (testare software inviando input non validi, inattesi o random).

Ci possiamo fidare delle astrazioni fornite dal PL (es. boolean è o vero o falso e mai null), i programmi hanno sempre una semantica ben definita

Meccanismi per ottenere la safety:

Check in compilazione (type checking)

Runtime checks (array bounds, non null, runtime type checks)

Garbage collector (no free())

Usare un execution engine (sull'hw) tipo la JVM che fa questi 3

Memory Safety

Un PL è m.s. se garantisce che i programmi non accederanno mai a zone non/de-allocate di memoria (no segmentation faults) né accederanno a memoria non inizializzata. [extra: fat pointers – hanno la size]

Features che rompono la m.s.: buffer overflow (array bound), aritmetica dei puntatori, free(), non inizializzazione (calloc vs malloc), casting non vincolati (unconstrained)

Type Safety

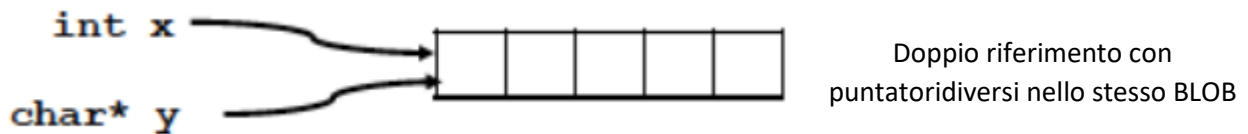
I tipi assicurano proprietà invarianti degli elementi del programma: x sarà sempre un intero, array length = 10, const var sarà sempre la stessa, ...

In questo modo il comportamento indefinito è sicuramente rimosso in quanto il controllo del type checker garantisce per esempio che non moltiplichiamo booleani o altro.

Esiste anche una strong type safety, quando un linguaggio garantisce queste asserzioni anche a runtime.

No memory safety -> no type safety

Come rompere la type safety? Basta un piccolo flaw e rip, per esempio sfruttando la type confusion ciao.



La semantica operativa **tipata** registra ogni informazione del type check a runtime (non tipata, no). La loro equivalenza è provata per programmi ben-tipati.

Arithmetic Safety

$i = i + 1$, overflow, vari approach

- Approccio non safe: undefined behavior (C, C++)
- Un po' meglio: specifica come si comporta (tipo $2^{**32} \rightarrow -2^{**32}$)
- Meglio ancora: eccezione (checked mode in C#)
- Meglio di tutte: avere precisione infinita (vedi alcuni linguaggi funzionali)

Thread Safety

Data race problem, operazioni non atomiche (read + assign). Un PL è thread-safe se la sua semantica è ben definita in presenza di multi-threading.

Aliasing

La causa principale di molti problemi: A e B puntano alla stessa cosa, in multithread data race, in single thread posso cambiare oggetti e rompere A o B. In MT usare strutture dati immutabili è safer.

.....

Memory Corruption

Char buffer[4] e poi buffer[4] = 'a', undefined, può accadere di tutto (segmentation fault o Remote Code Execution). Il più comune è il buffer overflow, ma anche non controllare fallimenti durante allocazioni o dimenticare deallocazioni (memory leaks).

Code Injection

L'attaccante inserisce il suo codice in un buffer e corrompe il return address per puntare al proprio codice. Altrimenti code reuse quanti corrompi il return address per puntare a codice già esistente.

Difese a livello di piattaforma

Stack canary, address space layout randomization, non-executable memory, ..

Information Flow

Un security requirement riguardo il fatto che informazioni confidenziali non dovrebbero avere leak sulla rete e nessun input non fidato dalla rete dovrebbe leakare nel database. Qui integrità e confidenzialità sono duali, flip una proprietà ed ottieni un esempio corrispondente dell'altro.

Gli inputs sono pericolosi per l'integrità, gli outputs sono pericolosi per la confidenzialità.

Ci sono almeno due tipi di information flow:

- **diretto** (*esplicito*): `low = high` o `print(h)`
- **indiretto** (*implicito*): `if(h>0) {l=99;}` possono essere parziali, leak solo di una parte dell'info

```
int h; // security label secret
int l; // security label public
1. while (h>99) do {...};
// timing or termination may reveal if h > 99
2. while (l>99) do {...}; //no problem
3. priv[h] = 23;
// exception may reveal if h is negative
4. publ[h] = 23;
// contents of publ may reveal value of h and,
// exception may reveal if h is negative
5. priv[l] = 23;
// exception may reveal the length of priv, which
may be secret
6. publ[l] = 23;
```

Indirect flows si possono presentare tramite altri “canali nascosti”: **termination channel** (`while (h > 99) do {...}`), **timing channels** (dipendono da quanto tempo ce metti) e **eccezioni** (`a[i] = 23` può rivelare se `a` è null, la lunghezza forse nascosta di `a`, etc.), **power consumption, resource exhaustion, ...**

Approccio: statico, niente costi a runtime, puoi capire se un programma è sicuro prima di eseguirlo e puoi fare un'analisi dell'implicit flow più precisa.

Type system: associa tipi ad oggetti che rispettano determinati requirements, tramite type checking (se un programma può essere accettato da un determinato type system) e type inference (inferisci il tipo di ogni oggetto)

In questo approccio type-based, consideriamo un “lattice” (reticolo) dove definiamo diversi livelli di sicurezza: per semplicità ne definiamo solo 2, **high** e **low**.

Legato a questi livelli di sicurezza ci mettiamo anche la **semantica** dei programmi: operazionale (in base al numero di step di computazione), assiomatica (in base a qualche proprietà logica del programma definiamo assiomi e “blablabla”) e denotazionale (definita in modo astratto tramite dei costrutti matematici).

Denotational semantics*

$[[P]]$

- Every linguistic construct (part of the program) P is given a **denotation** $[[P]]$
- $[[P]]$ is a mathematical object which represents P 's **contribution** to the meaning of any program that includes P
- The denotation of a program is determined just by the denotations of its components, i.e. the **semantics is compositional**

Type systems: ingredients*

- **Types** and the relation **has-type** of the form $e:t$ meaning e has type t
- Static typing **environment**, if any: Γ
- **Type judgement**: $\Gamma \vdash C$ meaning that : the program C is typable in Γ
- **Type rules** that assert the validity of certain judgments on the basis of other judgments that are already known to be valid: each type rule is written as a number of **premise** judgments above a horizontal line, with a single **conclusion** judgment below the line: TJs above imply TJs below

L'unica cosa che cambia è vedere se un programma C è tipabile in un contesto di sicurezza PC
 $[pc] \vdash C$

EXPRESSIONS

$\frac{e: \text{low}}{e: \text{high}}$ e can always be **high**

$\frac{h \text{ not in } \text{Vars}(e)}{e: \text{low}}$

e can be **low** only if it does not include **h** components

ATOMIC COMMANDS

$[pc] \vdash \text{skip}$

$\frac{e: \text{low}}{[low] \vdash l := e}$

$[pc] \vdash h := e$ This assignment is always ok

This assignment is ok, provided e is **low**
This avoids explicit flows

Assegnamento: se sei in un contesto basso, puoi assegnare un valore basso (evita explicit flow)

Sequenza di comandi: tutti i comandi devono essere tipabili nello stesso contesto

Se è tipabile in high, è tipabile in low

COMPOSITIONAL RULES

$$\frac{[pc] \mid - C1 \quad [pc] \mid - C2}{[pc] \mid - C1;C2}$$

Sequential commands must be typable in the same context

$$\frac{e: pc \quad [pc] \mid - C1 \quad [pc] \mid - C2}{[pc] \mid - \text{if } e \text{ then } C1 \text{ else } C2}$$

Implicit flows: branches of a **high** IF must be typable in a **high** context

$$\frac{e: pc \quad [pc] \mid - C}{[pc] \mid - \text{while } e \text{ do } C}$$

Implicit flows: the body of a WHILE with a **high** condition must be typable in a **high** context

no assignments to low variables in C

$$\frac{[high] \mid - C}{[low] \mid - C}$$

If a program is typable in a **high** context, it is typable also in a **low** one

Come siamo sicuri che un type system sia corretto?

Per il teorema di Soundness: un type system è sound quando programmi che sono ben tipati non fanno leaks.

Completeness di un TS: programmi che non leakano possono essere tipati. (*ar contrario*)

Teorema: $[pc] \mid - C$ implies C secure

programmi sicuri SSE input alto non interferisce con la parte a basso livello del sistema

Come definire se un sistema è sound? **Nozione di non-interferenza**, ci fa capire cosa può essere osservato da un TS.

In alcuni contesti è inevitabile il fare leak d'informazioni (*tipo il login*), quindi serve un modo per **declassificare** le informazioni: una nozione più "libera" rispetto a quella precedente della non-interferenza.

Va bene avere qualche leak. Se hai informazioni che dipendono da dati sensibili, ma mostrarle non è un problema, allora le declassifichi.

In Confidentiality: quando fai encryption di una pwd e stampi l'output dell'encryption, questo dato dipende da qualcosa di segreto ma è pubblico (declassificazione)

In Integrity: output di un controllo di un dato/firma digitale, l'output potrebbe dipendere anche da dati non fidati, ma lo fai comunque

$\text{declassify}(\text{expr}, \text{security-lvl})$

$\text{sec-lvl} \leq \text{attuale lvl di sicurezza dell'espressione}$

.....

Crittografia simmetrica

Lo schema a chiave simmetrica funziona con una coppia di E D, dove E è computazionalmente facile determinare D conoscendo solo E e viceversa. Quindi sia il sender che il receiver condividono una stessa chiave

Cifrario di Cesare, cifrari monoalfabetici, etc.

Facilmente fregabili: brute force, cyphertext attack (basta il msg cifrato), oppure statistical analysis (conoscere la frequenza delle lettere in una lingua) -> POLIalfabetica, non mono (una lettera viene cambiata in modi diversi in base alla sua posizione del testo, tipo in Cifrario Alberti, Augusto, Trasposizione (la chiave è una permutazione di una frase che decidi tu))

Cifrari perfetti: One Time Pad

Abbi una chiave almeno lunga quanto il messaggio che vuoi inviare, non riusarla. Cm invii la chiave xd

Criteri di Shannon: un cifrario sicuro si basa su

Confusione: la relazione tra il ciphertext e la chiave simmetrica è complessa. Ogni simbolo dovrebbe dipendere da varie parti della chiave

Diffusione: piccoli cambi in input, grossi in output

Un tempo si usavano sostituzione e trasposizione, oggi usiamo cifrari a blocchi e/o a stream.

Blocchi: dividono il testo in blocchi di lunghezza fissa e criptano un blocco a volta.

Stream: cripta un bit del testo a volta.

DES: 16 rounds, sostituzione (confusion) e permutazione (diffusion)

Block cipher, encrypting 64-bit blocks. Uses 56 bit keys.

Expressed as 64 bit numbers (8 bits parity checking* for correctness)

K

Nessuna dimostrazione di sicurezza, oltretutto è rotto, usa almeno il Triple-DES (il double Meet-In-The-Middle attack, nel triple diventano 2^{112} operazioni)

AES: cifrario a blocchi di 128 bits (con differenti modalità)

chiavi da 128/192/256 e può avere 10/12/14 rounds rispettivamente

S-box (*confusione*) è basato sull'aritmetica modulare coi polinomi

"In ogni round si fanno delle operazioni" – Jacopo 10/07/2020 16:14

E queste operazioni sono sostituzione *subBytes*, *shiftRow*, *mixColumn*, *addRoundKey*

Crittografia simmetrica

La crittografia a chiave pubblica si basa sull'idea di avere due chiavi, una pubblica per crittare i dati, ed una privata per decriptarli.

La chiave pubblica è una funzione trap-door one-way (facile da computare ma difficile da invertire a meno di conoscere il meccanismo segreto che rende facile il tutto [trap-door, tipo la fattorizzazione numeri primi]).

Dev'essere facile: generare una coppia di chiavi, mandare la chiave pubblica al receiver, decrittare usando la chiave privata,

Dev'essere infattibile: per un attaccante determinare la chiave privata dalla chiave privata e decrittare il messaggio usando la chiave pubblica

Gli attacchi di brute force, chosen plaintext atk (ti scegli un certo testo in chiaro, lo codifichi ed in base a quello che ti esce provi a capire quali sono le chiavi. Soluzione: fai un append di qualche bit random al messaggio in chiaro e ciao)

Algoritmi a chiave pubblica per la distribuzione delle chiavi, poi i dati vengono scambiati sulle chiavi simmetriche

Vedremo per lo scambio di chiavi: [ASIMMETRICO]

RSA (distribuzione chiavi, 1 a molti)

Difficile fattorizzare numeri molto grandi

Diffie-Hellman (scambio, 1 a 1)

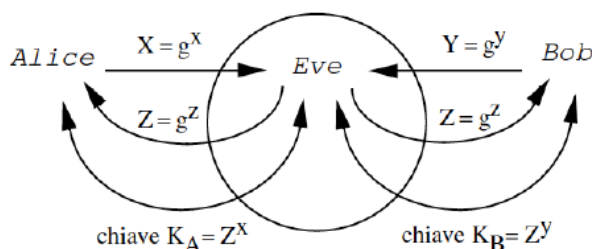
A – msg --> B chiudendolo con “un lucchetto”. B lo riceve ed aggiunge il suo lucchetto e lo rimanda indietro. A riceve, rimuove il suo lucchetto e rimanda a B. B ora può aprire e leggere

Matematicamente: A e B si mettono d'accordo su due numeri primi, **N** (Primitive Root: un numero le cui potenze modulo P generano 1, ..., p-1, cioè i numeri in Z^*p) e **G** (numero molto grande, generatore di Z^*n). Entrambi scelgono un numero random tra 0 e G e calcolano entrambi $G^{**}numero \bmod N$, generando x e y: poi se li scambiano. Così otteniamo:

$$K_A = Y^a \bmod N = X^b \bmod N = G^{ab} = K_B$$

Attacco:

Man In the Middle perché manca l'autenticazione delle chiavi



Soluzione: firmare, e ciò richiede delle chiavi condivise tra i due

Data Integrity: è la proprietà di un dato che non è mai stato alterato dal momento in cui è stato creato, trasmesso o immagazzinato da una sorgente autorizzata.

Hash Function: mappa dati di lunghezza arbitraria in dati di lunghezza fissa

Crypto HF: creata per essere one-way. Usate per rappresentare “unicamente” un msg e garantire integrità ed autenticazione, confrontando pre e post modifiche

Inoltre, si parla di:

- One-way [pre-image resistant]: dato y è difficile trovare x t.c. $h(x) = y$
- Weak collision resistant [2nd pre-image resistant]: dato x , difficile trovare $h(x) = h(x')$
- Strong collision resistant: difficile trovare x e x' t.c. $h(x) = h(x')$

Note: Hash Value si chiama *DIGEST*

Message authentication:

un tipo di autenticazione in cui confermi che qualcuno è la sorgente reale di un dato (*chiamata anche data-origin authentication*). Si fa con i MACs (Message Authentication Codes) e le digital signatures.

MACs:

famiglia di funzioni di hash parametriche rispetto al parametro k condiviso tra le due parti (segreto).

Dig Sig:

risolvono il problema della non-repudiation [sender non può dire di non aver inviato qualcosa]

Attacchi alle signatures

Forgery attack

L'attaccante B seleziona a random una signature da quelle possibili e computa $m = E(s)$. Siccome $S = M$, manda (m, s) come messaggio e la verifica ritorna true anche senza che A segni m . Allora scegli un set M' molto più piccolo di M ed accetta $V_a(s)$ se $E(s)$ è in M' .

Key Management and Distribution

Distribution of crypto keys, bind an identity to a key, generation/revoking/maintain for keys

Come?

- Link encryption (*fisicamente*)
Encrypt avviene indipendentemente ad ogni link, decripta e recripta ogni volta
- End-to-end encryption (3rd party, usare una chiave precedente per una nuova, ...)
Encrypt tra source e dest

Un Key Distribution Center (**KDC**) può essere usato per distribuire chiavi a coppie d'utenti. Deve essere "trusted", ma è un single point of failure. Le chiavi sono tipicamente usate in una gerarchia:

- **Session keys:** servono a cryptare i dati per una sola sessione e poi scartate
- **Master keys:** usate per criptare le session keys, mandate dal KDC

Essendoci molte chiavi, separale per tag (per PIN, per files, ...)

Note: KDC decentralizzati hanno troppe Master keys $n*(n-1)/2$

Trusted Authority: le chiavi pubbliche sono prese una una directory dove ci sono varie entry per ogni utente {ID, Public Key}. Tutti hanno un accesso sicuro a questa directory.

Central Authority: che gestisce la directory delle public key (*bottleneck, ...*)

Public-key certificates: contatti una Certification Authority che ti restituisce un certificato (chiave pubblica e un identificatore del proprietario, tutto firmato dal CA). *Un certificato quindi associa criptograficamente un'identità ad una chiave.*

Come viene certificata una signature? (*quindi certificare una signature chain?*)

- **Gerarchico:** scambia certificati fino al primo antenato in comune
- **Web of trust:** non hai una CA e ti basi sul giudizio degli utenti (*usato in PGP*)

KDC vs CA

Symmetric Keys

- KDC on-line, used at every session
- KDC knows secret key
- If KDC compromised, past and future messages are exposed
- Fast

Asymmetric Keys

- CA off-line, except for key generation
- CA only knows public key
- If CA compromised, only future messages are exposed
- Slow

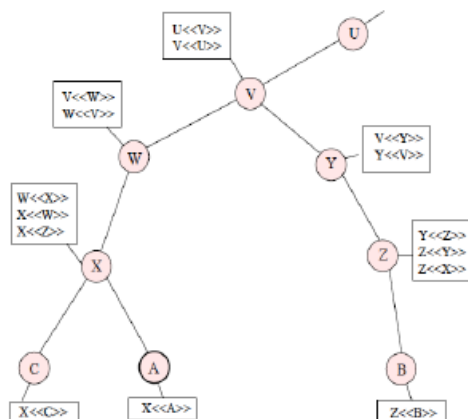
X.509

È lo standard e definisce come deve essere strutturato un certificato. Il cuore di questo schema è il certificato a chiave pubblica associato ad ogni utente, creato dalla CA e gestiti dall'utente.

Numero di serie, identifier dell'algoritmo di signature, nome della CA che ha emesso il certificato (issuer), periodo di validità, nome dell'utente, ...

Per certificare la validità:

- **Direct trust:** tutti sottoscrivono la CA e quindi tutti si fidano di quella
- **Hierarchical trust:** quando ci sono tanti utenti, diverse CA ed ognuna copre un subset di users.



A -> B, X si fida di W, W di V, ... fino al target

- **Web of trust:** niente CA, come prima

Pretty Good Privacy (PGP)

Utilizza un'infrastruttura della gestione delle chiavi basata sui certificati, diversi da quelli di X.509, per esempio:

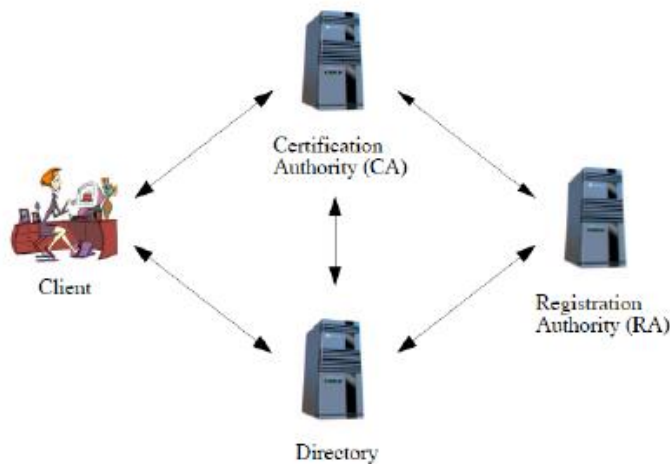
- una chiave può avere diverse firme, dove la signature è la nozione di trust del PGP;
- possono esserci diversi livelli di trust;
- ogni utente può essere visto come una CA. Una volta certificato un utente diventi un **introducer**

I certificati possono essere revocati prima che scadano perché possono venire compromessi, o un utente potrebbe non essere più certificato da una CA, etc. **Come?** Si inserisce il certificato nella Certification Revocation List (CRL). In questo modo è possibile controllare attivamente o usare un servizio di validazione.

Recupero della chiave quando una chiave viene persa? Con un **key escrow system** che è un sistema che permette a 3rd parties di recuperare una chiave crittografica.

Public Key Infrastructure (PKI)

Qualcosa che associa le chiavi pubbliche alle entità, quindi nuovamente certificati, revoca/recupero chiavi...



CA: crea certificati nella dir e gestisce CRL

Dir: rende i certificati (e CRL) disponibili

RA: gestisce il processo di registrazione degli utenti e certificati

La PKI può essere **aperta** (per collegare compagnie) o **chiusa** (per limitarsi ad un gruppo di persone)

Come assegnate il nome ad un'identità in modo da rappresentarlo univocamente? **Naming**

In X.509 si usano i Distinguished Names (devono contenere informazioni che non possono cambiare ed univoci nel tempo). La **authentication policy** stabilisce il livello minimo necessario affinché la CA accetti l'identità dell'utente. La **issuance policy** descrive a chi potrà essere dato un certificato.

Security Protocol

Un SP assicura che una comunicazione venga protetta tramite uno scambio di messaggi. Necessario perché gli utenti comunicano su una rete untrusted.

Un **protocollo** è un insieme di convenzioni/regole che determinano lo scambio di messaggi tra due o più parti. *“In breve, un algoritmo distribuito con enfasi sulla comunicazione.”*

Noi ci interesseremo ai protocolli di comunicazione. Per crearne uno (*build a key establishment protocol*) prendiamo un set di utenti dove qualunque coppia può creare una nuova chiave di sessione da usare in comunicazioni seguenti interagendo con un server trusted.

Nulla dovrebbe essere mandato in chiaro!

Usiamo il **nonce** (number **once**) per ovviare al problema che un attaccante possa usare una chiave di una vecchia sessione (*Replay attack*).

Needham-Schroeder secret-key

1. $A \rightarrow S : A, B, N_A$
 2. $S \rightarrow A : \{K_{AB}, B, N_A, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
 3. $A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$
 4. $B \rightarrow A : \{N_B\}_{K_{AB}}$
 5. $A \rightarrow B : \{N_B - 1\}_{K_{AB}}$
- $K_{AB} N_A$ bound and encrypted

Ma il messaggio 3 in Needham-Schroeder non ha nonce, quindi B non può sapere se K_{AB} è nuova o meno (C potrebbe spacciarsi per A usando una vecchia chiave). Questa cosa si risolve facendo mandare a B un suo nonce, diverso da quello di A

1. $B \rightarrow A : B, N_B$
2. $A \rightarrow S : A, B, N_A, N_B$
3. $S \rightarrow A : \{K_{AB}, B, N_A\}_{K_{AS}} \{K_{AB}, A, N_B\}_{K_{BS}}$
4. $A \rightarrow B : \{K_{AB}, A, N_B\}_{K_{BS}}$






In questo protocollo non otteniamo key confirmation (né A né B possono sapere se l'altro ha ricevuto K_{AB}), inoltre A non ha modo di collegare N_B a B (**Lowe's fix**: B manda anche il suo identificatore)

1. $A \rightarrow B : \{N_A, A\}_{K_B}$
2. $B \rightarrow A : \{N_A, N_B, B\}_{K_A}$
3. $A \rightarrow B : \{N_B\}_{K_B}$

Ma allora li tratti come coppie (N_B, B) da mandare ad A e verranno trattati come inizio di una nuova esecuzione dove la coppia è l'identità dell'agente (**Meadows**) e siamo di nuovo al punto di partenza.

➔ Formal Methods

Kinds of attacks

- **Man-in-the-middle** (or **parallel sessions**) attack: $A \leftrightarrow M \leftrightarrow B$.
- **Replay** (or **freshness**) attack: reuse parts of previous messages 
- **Masquerading** attack: pretend to be another principal, e.g.
 - M forges source address (e.g. present in network protocols), or
 - M convinces other principals that A's public key is K_M . 
- **Reflection** (or **mirror**) attack send transmitted information back to originator 
- **Oracle** attack: take advantage of normal protocol responses as encryption and decryption "services" 
- **Type flaw** attack: substitute a different type of message field (e.g. a key vs. a name) 

Otway-Rees

1. $A \rightarrow B: I, A, B, \{N_A, I, A, B\}_{K_{AS}}$
2. $B \rightarrow S: I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}}$
3. $S \rightarrow B: I, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
4. $B \rightarrow A: I, \{N_A, K_{AB}\}_{K_{AS}}$

Supponendo $|I, A, B| = |K_{AB}|$ abbiamo un type-flaw chiamato reflection, dove C può usare una tripla di informazioni conosciute come chiave di sessione, saltando step 2 e 3.

1. $A \rightarrow C(B): I, A, B, \{N_A, I, A, B\}_{K_{AS}}$
4. $C(B) \rightarrow A: I, \{N_A, I, A, B\}_{K_{AS}}$

C può anche spacciarsi per S rimandando indietro a B le stesse informazioni mandategli, similmente all'attacco precedente otteniamo una chiave di sessione formato da informazioni note.

Andrew Secure RPC protocol

Scambia una chiave segreta, autenticata e nuova tra due utenti tramite una chiave simmetrica

1. $A \rightarrow B: A, \{N_A\}_{K_{AB}}$
2. $B \rightarrow A: \{N_A + 1, N_B\}_{K_{AB}}$
3. $A \rightarrow B: \{N_B + 1\}_{K_{AB}}$
4. $B \rightarrow A: \{K'_{AB}, N'_B\}_{K_{AB}}$

K'_{AB} session key, N'_B nonce per sessioni future

Anche qui type flaw attack dove C può far accettare ad A " $N_A + 1$ " come chiave di sessione poiché il nonce è predicibile. **Note:** non è un problema di secrecy

Denning & Sacco (key exchange with..)

1. $A \rightarrow S: A, B$
2. $S \rightarrow A: C_A, C_B$
3. $A \rightarrow B: C_A, C_B, \{\{T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B}$

K_{AB} è la chiave di sessione segreta, K_B è la chiave pubblica di B, K_A^{-1} la segnatura della chiave privata di A, C_A e C_B i certificati di A e B rispettivamente ed infine T_A il timestamp generato da A

Quindi B è sicuro che è stato mandato da A perché decripta con la chiave pubblica di K_A , legata ad A per via di C_A ed oltretutto che il messaggio sia proprio per lui perché criptato con K_B

Indovina? MaN IN The MiDdLe AtTaCkKkKk

3. $A \rightarrow C: C_A, C_C, \{\{T_A, K_{AC}\}_{K_A^{-1}}\}_{K_C}$
- 3'. $C \rightarrow B: C_A, C_B, \{\{T_A, K_{AC}\}_{K_A^{-1}}\}_{K_B}$

Soluzione: basta inserire i nomi all'interno della segnatura $\{A, B, T_A, K_{AB}\}_{K_A^{-1}}$

Kerberos

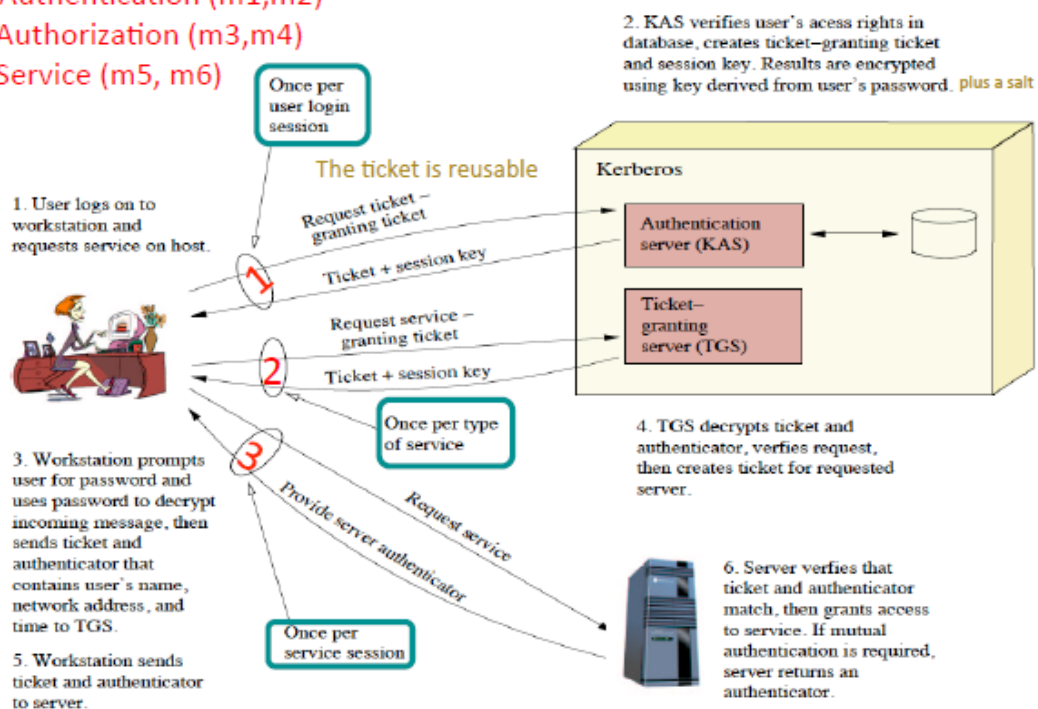
Autenticare in un sistema distribuito, basato su un Key Distribution Center e si basa sulla crittografia simmetrica. Usa i ticket per fare autenticazione per non reinserire le credenziali ogni volta ("Proof of Identity", una prova di un'identità)

- 1) Crea il canale tra A e B (utilizzando il KDC che darà la chiave di sessione)
- 2) La freshness è data dai timestamps (non nonces). L'architettura di Kerberos è basata su Kerberos Authentication Server (KAS) ed un Ticket Granting Server (TGS)

1. Authentication (m1,m2)

2. Authorization (m3,m4)

3. Service (m5, m6)



La fase di autenticazione: bisogna presentarsi all'Authentication Server.

Simplified version

1. $A \rightarrow KAS: A, TGS$

2. $KAS \rightarrow A: \{K_{A,TGS}, TGS, T_1, \{A, TGS, K_{A,TGS}, T_1\}_{KAS,TGS}\}_{KAS}$

Annotations for step 2:
 - $K_{A,TGS}$: TGS Session key
 - T_1 : timestamp
 - $\{A, TGS, K_{A,TGS}, T_1\}_{KAS,TGS}$: AuthTicket for TGS
 - The entire package: credentials to TGS

La fase di autorizzazione:

3. $A \rightarrow TGS: \{A, TGS, K_{A,TGS}, T_1\}_{KAS,TGS}, \{A, T_2\}_{KAS,TGS}, B$

Annotations for step 3:
 - $\{A, TGS, K_{A,TGS}, T_1\}_{KAS,TGS}$: AuthTicket
 - $\{A, T_2\}_{KAS,TGS}$: Authenticator

4. $TGS \rightarrow A: \{K_{AB}, B, T_3, \{A, B, K_{AB}, T_3\}_{KB,TGS}\}_{KA,TGS}$

Annotations for step 4:
 - K_{AB} : Session key
 - $\{A, B, K_{AB}, T_3\}_{KB,TGS}$: ServTicket for B
 - The entire package: credentials to B

La fase di servizio:

5. $A \rightarrow B: \{A, B, K_{AB}, T_3\}_{KB,TGS}, \{A, T_4\}_{KAB}$

Annotations for step 5:
 - $\{A, B, K_{AB}, T_3\}_{KB,TGS}$: ServTicket
 - $\{A, T_4\}_{KAB}$: Authenticator

6. $B \rightarrow A: \{T_4+1\}_{KAB}$

Kerberos lavora con i **realms**, definito come un server di Kerberos: immagazzina le informazioni di un utente necessarie per il realm a cui appartiene. Ogni server si registra sugli altri servers per permettere una connessione inter-realms

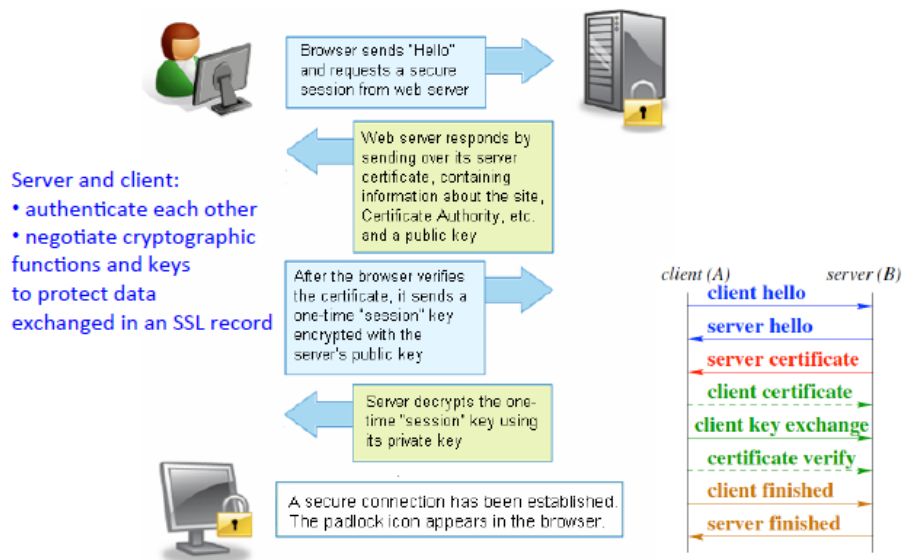
Internet security

SSL (Handshake)

Secure Socket Layer

Setta uno o due canali per comunicazioni segrete utilizzando i certificati. Consiste in più sottoprotocolli che lavorano su TCP (Handshake [(re)inizia le connessioni], Record [manda i dati], Alert Protocol, Change Cipher Spec)

SSL Handshake schema



IPsec

Crea un canale sicuro per tutte le applicazioni. È installato su tutti i sistemi operativi e su firewalls e routers. Sui routers viene usato per implementare i servizi di VPN.

Lo standard prevede che ci sia uno header che protegge l'integrità ed autenticità degli IP-datagram. Poi un Encapsulating Security Payload per la confidentiality. Poi il Key Management fatto con l'IKE

Internet Key Exchange

Non stabilisce soltanto le chiavi, ma setappa anche i parametri di sicurezza (Security Association SA), tra cui il format dei protocolli usati, gli algoritmi crittografici e di hashing utilizzati ed, ovviamente, le chiavi. È basato su Diffie-Hellman.

- 1) Due parti negoziano gli SA da usare nella fase 2
- 2) Gli SA sono usati per creare i "child SAs" per usarle nelle comunicazioni successive

Modello formale

Analizza il sistema partendo dalle sue specifiche. Vantaggio, come tutta l'analisi statica, si basa su teoria rigorosa ed automatizzabile, però dispendioso in termini di tempo.

Lo scopo è quello di modellare i protocolli e le loro proprietà utilizzando regole matematiche

Ci sono diversi approcci per fare una verifica formale dei protocolli

- **Belief logic:** permette di capire che cosa i partecipanti del protocollo possono inferire dai messaggi che vedono.
- **Model checking:** vede il protocollo come un automa a stati finiti e controlla che tutti gli stati raggiungibili siano sicuri
- **Theorem proving:** usa l'induzione su alcune esecuzioni del protocollo
- **Analisi statica:** inferire proprietà dalle specifiche

BAN logic

Logica basata sul belief (e sulla loro evoluzione come conseguenza di comunicazione), basata su una serie di regole di deduzione ed è usata sia per formalizzare protocolli sia per autenticazione.

La usiamo perché spesso non si ha una descrizione formale (spesso ambigua) e quindi necessitiamo di un modo per chiarire le assunzioni e provare che il protocollo raggiunge i suoi security goals.

Associa ad una sequenza di scambi di messaggi una collezione di belief statements (con alcune assunzioni iniziali): facilmente adattabile a vari contesti.

Per analizzare un protocollo con la BAN logic:

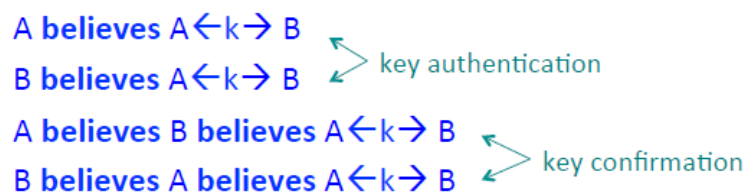
- trasforma il protocollo con logical statements
- dire quali sono le assunzioni iniziali
- applicare le regole BAN per dedurre nuovi statements
- alla fine prendi i belief finali ed interpreta il loro significato per vedere se è lo stesso dei goal del protocollo

Costrutti

- **P believes X:** P agisce come se X sia vero
- **P sees X:** P riceve un messaggio contenente X
- **P once said X:** in passato, P ha mandato un messaggio contenente X
- **P controls X:** P è proprietario di X

Deduction Rules

Un insieme di premesse ed una conclusione. Le regole permettono di creare dimostrazioni come alberi dove le foglie sono le assunzioni del protocollo.



Protocol

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{KBS}\}_{KAS}$
3. $A \rightarrow B : \{K_{AB}, A\}_{KBS}$
4. $B \rightarrow A : \{N_B\}_{KAB}$
5. $A \rightarrow B : \{N_B - 1\}_{KAB}$

Each msg is now a formula

Idealization

1. $A \rightarrow S$: omitted [because in clear, thus irrelevant]
2. $S \rightarrow A : \{N_A, A \leftarrow K_{AB} \rightarrow B, \text{fresh}(A \leftarrow K_{AB} \rightarrow B), \{A \leftarrow K_{AB} \rightarrow B\}_{KBS}\}_{KAS}$
3. $A \rightarrow B : \{A \leftarrow K_{AB} \rightarrow B\}_{KBS}$
4. $B \rightarrow A : \{N_B, A \leftarrow K_{AB} \rightarrow B\}_{KAB}$ from B
5. $A \rightarrow B : \{N_B, A \leftarrow K_{AB} \rightarrow B\}_{KAB}$ from A

Origin distinguishes the two messages

Interpretation

1. $A \rightarrow S$: -
2. $S \rightarrow A$: S said K_{AB} is ok and fresh for talking with B
3. $A \rightarrow B$: S said K_{AB} is ok for talking with A
4. $B \rightarrow A$: B said K_{AB} is ok for talking with A
5. $A \rightarrow B$: A said K_{AB} is ok for talking with B

Initial assumptions

- A and S believe $A \leftarrow K_{AS} \rightarrow S$
- B and S believe $A \leftarrow K_{BS} \rightarrow S$
- S believes $A \leftarrow K_{AB} \rightarrow B$
- A and B believe (S controls $A \leftarrow K \rightarrow B$)
- A believes (S controls $\text{fresh}(A \leftarrow K \rightarrow B)$) [A trust S to generate a fresh key]
- A believes $\text{fresh}(N_A)$ and B believes $\text{fresh}(N_B)$
- S believes $\text{fresh}(A \leftarrow K_{AB} \rightarrow B)$

The following assumption is pretty strong and is needed for authentication:

- B believes $\text{fresh}(A \leftarrow K \rightarrow B)$

Needham and Schroeder didn't realise they were making it!

Però BAN ha delle limitazioni, per esempio

Neset protocol (1990)

1. $A \rightarrow B : \{N_A, K_{AB}\}_{KA}^{-1}$
2. $B \rightarrow A : \{N_B\}_{KAB}$

A e B considerano K_{AB} una buona chiave quando è stata inviata in chiaro. **Ma la confidentiality?**

BAN non cerca di controllare chi non ha accesso a quella chiave (*non riguarda il rilascio di segreti non autorizzati*).

Inductive Method

È tipo un miscuglio delle altre, poiché prende dal metodo di esplorazione degli stati la nozione di eventi e dalla belief logic l'idea di derivazione. *"Prove correctness instead of looking for bugs"*

La presenza di stati finiti permette di trovare attacchi velocemente, ma può causare una semplificazione troppo stringente. Per quanto riguarda la belief logic, permette di creare prove astratte, anche se in alcuni casi potrebbe risultare complicato. *Usa un theorem prover chiamato Isabelle*

Il protocollo viene formalizzato come l'insieme delle possibili sequenze d'eventi: su queste verrà fatta induzione per provare proprietà. Esse includono anche azioni di un possibile attaccante generico. Usi poi l'induzione per dimostrare la correttezza di queste tracce. Se una proof fallisce, allora hai scoperto un difetto del protocollo -> Per ogni stato, in ogni trace, vai a dimostrare che nessuna condizione di sicurezza fallisce.

Scegli sempre la sequenza più corta per arrivare ad un "bad state"

L'insieme H contiene tutti i messaggi mandati all'interno di una traccia. Gli operatori aggiungono a questo insieme altri elementi derivabili da H stesso.

Operatori

- **Parts:** riguarda le componenti dei messaggi
Aggiunge ad H le componenti singole di ogni messaggio ed il corpo dei messaggi criptati
- **Analz:** riguarda la decriptazione dei messaggi
Come partz, ma le chiavi con cui si criptano i messaggi sono conosciute. $\text{Analz} \subseteq \text{Partz}$
- **Synth:** riguarda la falsificazione dei messaggi
Modella i messaggi che la "Spia" può costruire dagli elementi di H

Traces

Liste di eventi di send (*Says $A B X$ – "A sends X to B "*) e di store (*Notes $A X$ – "A records X for future use"*)

initState() definisce la conoscenza iniziale

spies() definisce l'insieme dei messaggi che una spia può vedere in una traccia. *Solitamente si possono vedere solo i propri messaggi inviati/ricevuti, una spia può vedere tutto.*

L'insieme *used* viene usato per formalizzare il concetto di freshness, utile per i segreti dinamici (*nonces, SK*)

Pros

Puoi parlare di numeri arbitrariamente grandi in termini di agenti, spazio di messaggi, etc [unbounded protocol model]

Cons

Non sempre ottieni una risposta e puoi dimostrare solo proprietà basate su trace di eventi

Possibility Lemmata: il formato dei messaggi è uguale tra steps successivi

Regularity Lemmata: una chiave può essere vista dalla spia solo se l'agente di quella chiave è BAD

Unicity Lemmata: Nonce e chiavi identificano univocamente il messaggio

Secrecy Theorem (basati sull'insieme *analz H*)

Una chiave non è mai criptata usando una chiave di sessione [*session key compromise theorem*]

Le chiavi che vengono distribuite rimangono confidenziali [*session key secrecy theorem*]

Pi Calculus

Lambda calculus = tutto viene visto come una funzione, quindi fare una computazione significa applicarla
Usiamo il Pi perché è tutto visto come un processo e le computazioni sono comunicazioni su un canale

Modello di Milner

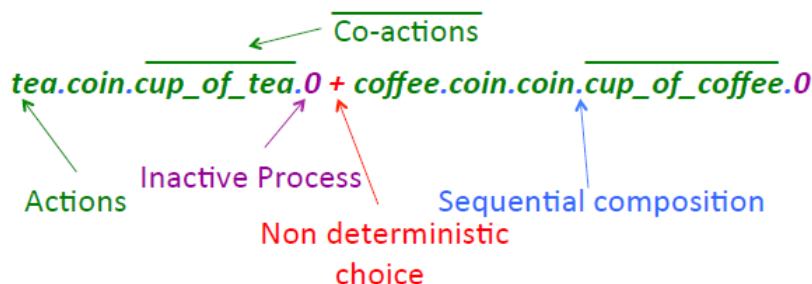
Un sistema concorrente è una collezione di processi, dove ogni processo è un agente indipendente che può fare attività interne ed in isolamento o può interagire con attività condivise dell'ambiente

Nel Pi Calculus è importante la nozione di mobility perché abbiamo che:

- un processo si può muovere nello spazio di computazione
- un processo si può muovere nello spazio virtuale dei processi linkati
- i collegamenti si muovono nello spazio virtuale dei processi linkati

Si evolve dal CCS (*Calculus of Communicating System*). Un processo CCS ha varie interfacce con cui comunica con gli altri [interfaccia = insieme di canali per ricevere input / dare output]. Il canale è un'astrazione che mostra il link di comunicazione fra due processi.

- **Azioni:** (notazione: lettera) alla m
- **Co-Azioni:** (notazione: lettera barrata)



Sintassi

Insieme di nomi e co-nomi (N) che insieme formano le labels (L) che, quando unite all'azione τ (null, "silent action") formano le azioni (Act). Per descrivere formalmente un processo CSS si usa:

- Labelled Transition System
È una tripla (Processi, Azioni, relazioni binarie di processi chiamate "relazioni di transizione").
Quindi è un grafo
- Structural Operational Semantic
Definisce un insieme di regole d'inferenza: sopra le premises, sotto le conclusioni, dx condizioni

Value-Passing CCS

Estendere il CCS permettendo lo scambio di valori, intuitivamente, permette di caratterizzare il processo come un insieme di canali che espone al mondo. Quindi mandando un valore X su un certo canale si sta dando la possibilità al processo ricevente di accedere a questo dato, catturando più accuratamente il concetto di mobilità per accedere al dato.

$$(\bar{a}5.P' \mid a(x).R') \setminus a$$

P sul canale A scrive 5, R aspetta X sul canale A

Name extrusion

Una volta passato il canale A da $P \leftrightarrow R$ a $Q \leftrightarrow R$ (P “non ha” più il canale) è name extrusion

Arriviamo finalmente al Pi Calculus, il linguaggio principale per la programmazione concorrente.

Syntax

$P ::= 0$	inactive process
$\tau.P$	silent action
$\bar{x}y.P$	output
$x(y).P$	input
$[x = y] P$	match
$(x) P$ [or $(u x)$]	channel creation or restriction
$P \mid P$	parallel composition
$P + P$	non deterministic choice
$! P$	replication

We take processes up to
alpha-conversion:
a bound names can be
renamed with a fresh name

- Prefix “.” imposes an order upon actions

$[x = y]P$ è “name matching”, equivalente a $\text{if } x = y \text{ then } P$

Due tipi di bindings:

- **Input:** $x(y).P$
 y è un placeholder per qualsiasi nome che può essere ricevuto sul canale x
- **Restriction:** $P \setminus y$ OR $(y)P$ OR $(u y) P y$
Crea un nuovo nome (privato a P) e lo lega a y in P univocamente. I nomi definiti così sono bound, altrimenti free

Sostituzione

Una funzione $\sigma: N \rightarrow N$ sui nomi che è l'identità eccetto su un insieme finito di nomi.

Puoi rinominare solo i nomi free!

Semantica delle azioni

Una transizione è nella forma $P \xrightarrow{\alpha} Q$, significante che P può evolvere in Q facendo α .

Le possibili forme sono:

- τ “silent action” che non richiede interazione con l'ambiente
- $\bar{x}y$ “free output” dove P emette il nome free y sul canale x
- $x(y)$ “free input” P riceve un qualsiasi nome w su x , diventando $Q\{w/y\}$
- $\bar{x}(y)$ “bound output” emette un nome privato y su x , portando nomi fuori dal loro scope

Scope intrusion

P ha un canale X verso R e vuole passarlo a Q. Q vuole riceverlo, ma possiede già un canale privato X verso S: rinominalo per evitare confusione

$$\overline{y}x.P' \mid R \mid (x)(y(z).Q' \mid S) \xrightarrow{\tau} P' \mid R \mid (x')(Q'\{x'/x\}\{x/z\} \mid S\{x'/x\})$$

Scope extrusion

P ha un link X verso R ma è privato, P vuole passarlo tramite Y a Q. Q non possiede X link ed è pronto a ricevere. Quando il link è esportato, lo scope della restrizione è esteso, poiché X ora è privato tra i 3

Esempio:

$$(x)(\overline{y}x.P' \mid R) \mid y(z).Q' \xrightarrow{\tau} (x)(P' \mid R \mid Q'\{x/z\})$$

“(x) è il canale privato tra P ed R, ha il suo scope esteso anche a Q tramite Y”

Note: Se P' non ha il canale privato X, non è un'estensione ma una migrazione dello scope

Come si modellano i protocolli con Pi Calculus?

A livello astratto, i protocolli sono processi. Tramite la scope extrusion [modellare lo scambio di risorse private] e la restriction [creare canali], possiamo modellare il possesso e la modellare il possesso e la comunicazione di segreti (come le chiavi).

Gli **step** per modellare il protocollo sono:

- 1) definire l'algebra
- 2) definire i partecipanti del protocollo
- 3) permettere un numero illimitato di sessioni parallele [tramite replication]
- 4) definire l'avversario
- 5) mettere tutto in parallelo

Usare un canale sicuro:

$$\begin{aligned} P &= (\nu c_{AS})(\nu c_{BS})(A|B) \mid S \\ A &= (\nu c_{AB})(\overline{c_{AS}}(c_{AB}).\overline{c_{AB}}(M)) \\ S &= c_{AS}(x).\overline{c_{BS}}(x) \\ B &= c_{BS}(w).w(y) \end{aligned}$$

creo il canale AS e BS

sul canale AS scrivo il nome AB, scopro poi che è un canale perché ci scrivo sopra M

S si aspetta X su AS che poi scriverà su BS

B aspetta W su BS e poi prende in input il canale

Spi-Calculus

Pi Calculus esteso con le primitive crittografiche. Le chiavi possono essere dinamicamente create e comunicate.

.....

Perché usare l'analisi statica? Ci sono molte domande che ci possiamo fare, ma sfortunatamente quelle interessanti sono indecidibili, ma possiamo dare delle risposte approssimate!

Tecniche di analisi statica: *[analizza il testo -> termina sempre, bassa complessità, tools leggeri]*

- Interpretazione astratta
- **Control Flow Analysis**
- Type Systems

Predici approssimazioni del comportamento dinamico del programma, analizzando proprietà che valgono in ogni esecuzione del programma.

A confronto l'analisi dinamica (che analizza il comportamento invece del testo) è più precisa, ma potrebbe non terminare ed è decisamente più complessa e costosa.

Interpretazione astratta (Abstract Interpretation)

Approssima una semantica concreta dando una semantica astratta, trattando la correttezza della semantica. Verrà poi usata come base per produrre tools automatici.

Type Systems

Divide i valori dei programmi in tipi e stabilisce regole di tipo. Per controllare questa "disciplina" si fa type checking.

Anche qui si tratta di correttezza della semantica.

Control Flow Analysis (CFA)

Serve a stimare i valori che gli oggetti di un programma possono assumere a runtime: cioè rappresenta l'astrazione di un'esecuzione.

I type systems sono prescrittivi (*inferiscono il tipo ed impongono condizioni di ben formatezza allo stesso tempo*), mentre la CFA è descrittiva perché inferisce le informazioni ed in uno step separato si imporranno le condizioni di ben formatezza. Questo è buono perché per ogni proprietà va riscritto da zero il type system, mentre nel CFA basta un solo nuovo test.

Come si descrive il comportamento astratto?

Come una coppia di domini astratti (ρ, k)

Dove ρ è una funzione che prende un nome e restituisce un insieme di nomi a cui può essere associato

k prende un binder/nome e restituisce un insieme di nomi che possono essere mandati su quell'input

(ρ, k) è una stima valida quando $(\rho, k) \models P$, ovvero soddisfa un insieme di clausole logiche (*una per ogni caso sintattico*)

Note: La CFA con il Pi Calculus non considera restriction e replication e da qui deriva l'approssimazione precedentemente citata.

La CFA è context-insensitive (0-CFA) ed ha le seguenti proprietà:

- **Soundness:** l'analisi è semanticamente corretta. Se ρ e k soddisfano P e P va in Q , allora ρ e k soddisfano anche Q
- **Existence:** le soluzioni proposte possono avere un ordine parziale e quindi esiste sempre una "least solution"
- **Construction:** è una procedura costruttiva per ottenere la "least solution" con complessità polinomialmente bassa

Si dividono i nomi in nomi pubblici e nomi segreti. Si controlla che ro e k soddisfino P e si controlla che k di un qualsiasi nome pubblico sia un sottoinsieme dei nomi pubblici.

NRU/NWD

Dynamic Property: **no read-up/no write-down**

A high level process cannot write any value to a process at low level; symmetrically a process at low level cannot read data from one of a high level.

Runtime Verification

Esegui il programma da analizzare osservando l'execution trace e costruendo da quello un modello da analizzare.

Le limitazioni sono che

- Hai meno copertura del codice: *devi fare tanto testing (esecuzioni con inputs diversi, ..)*
- Il codice deve essere eseguibile: *usa esecuzioni simboliche o tramite analisi statica*

Definizione

La RV è la disciplina di CS dedicata all'analisi delle esecuzioni di un sistema studiandone le specifiche di un linguaggio con algoritmi di analisi dinamica, tecnologie basate su linguaggi di programmazione, etc

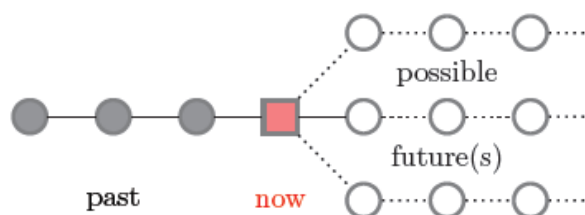
Eventi: quelli che registrano uno snapshot del comportamento a runtime

Tracce (τ): una sequenza finita di eventi

τ soddisfa ϕ iff $\tau \in L(\phi)$

Una traccia è una vista formale di un'esecuzione discreta: sequenza di stati ed eventi

Proprietà (ϕ): denota un linguaggio L (insieme di tracce)

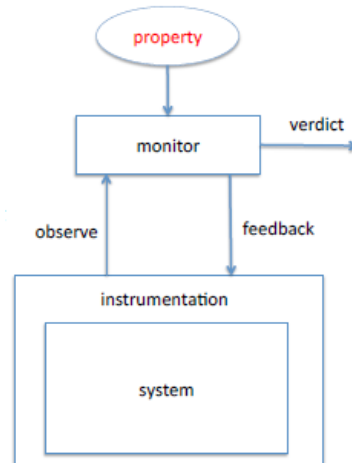


Verdetti

Dovrebbero rilevare successi/fallimenti asap. Un approccio standard è usare 4 domini di verdetti per considerare tutte le possibili estensioni di una traccia

	current trace τ	all suffixes σ	Action
1	$\tau \in \mathcal{L}(\varphi)$	$\tau\sigma \in \mathcal{L}(\varphi)$	stop with Success T
2	$\tau \in \mathcal{L}(\varphi)$	unknown	carry on monitoring T_{still}
3	$\tau \notin \mathcal{L}(\varphi)$	$\tau\sigma \notin \mathcal{L}(\varphi)$	stop with Failure F
4	$\tau \notin \mathcal{L}(\varphi)$	unknown	carry on monitoring F_{still}

Quindi il ciclo è che “hai una proprietà, devi monitorarla (farai un verdetto), il monitor manda dei feedback al sistema che rileverà eventi importanti e manda quelli più importanti indietro al monitor”.



Il monitor può essere:

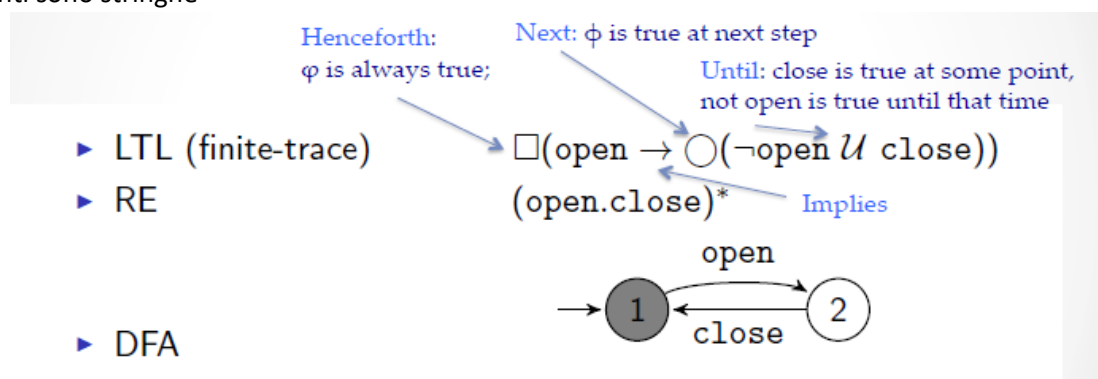
- **Offline:** la traccia è analizzata a posteriori (es. file di log, trace dump, ..)
- **Online:** la traccia è analizzata step by step e si divide in:
 - o **Esterno:** monitor va in più run in parallelo col sistema in modo sincrono o asincrono
 - o **Interno:** il codice del monitor è all'interno dell'applicazione

Reactions

Varie forme: mostrare un messaggio errore, lanciare un'eccezione, usare codice di recovery, ..

Propositional

Gli eventi sono stringhe



Verification: Check if each trace prefix is in the language of the property

Projection: prendi la traccia e filtri gli eventi irrilevanti

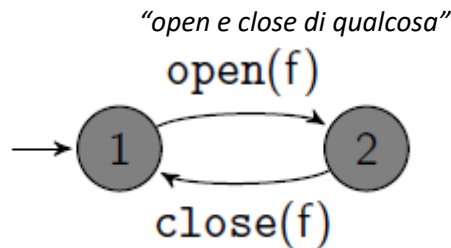
Open.read.write.close.open.read.close -> open.close.open.close

Parametric

Gli eventi contengono valori

open(f1).open(f2)

close(f2).open(f1)



CFA for Security Protocols

Hanno creato un sistema che traduce i protocolli in un process calculus chiamato *Lysa* ispirato allo Spi-calculus dove però i processi comunicano su una rete globale ed usano crittografia a chiavi simmetriche.

$$(\nu K) \langle A, S, A, B, \{K\}_{K_A}^A \rangle.$$

|

$$(A, S, A; x_B, x). \text{decrypt } x \text{ as } \{; x^K\}_{K_A}^S$$

Si possono aggiungere annotazioni ogni volta che (de)cripti qualcosa

La semantica standard ignora le annotazioni, ma possiamo creare una “reference monitor semantics” che le considera e blocca l’esecuzione di un processo quando una di queste viene violata.

Analisi

La CFA calcola: una approssimazione dei messaggi sulla rete

$$\kappa \in P(V^*)$$

ed i valori delle variabili

$$\rho : X \rightarrow P(V)$$

dove V è l’insieme dei valori (termini di variabili libere)

L’error component ψ colleziona coppie di cripto-punti dove le asserzioni nelle notazioni possono essere violate. $(A,B) \in \psi ==$ “Qualcosa criptato in A può essere inaspettatamente decriptato in B”

Oltre al solito

$$(\rho, \kappa) \models P : \psi$$

(ρ, κ) è una estimate per messaggi e valori, deve soddisfare P e ErrComp

Abbiamo anche decisioni ausiliarie per i termini

$$\rho \models E : \vartheta$$

perché a differenza del PiCalc (avevi che l’expr era di un solo tipo), la E può essere una variabile, un’encryption, un’encrypt di un’encrypt etc e devi poter identificare tutto ciò

dove $\vartheta \in P(V)$ approssima il set di valori in cui E può essere valutato

Judgement for Decryption

(of binary terms)

$\rho \models E : \vartheta \wedge$	evaluate terms
$\rho \models E_0 : \vartheta_0 \wedge \rho \models E_1 : \vartheta_1 \wedge$	and subterms
$\forall \{V_1, V_2\}_{V_0}^{\ell} [\text{dest } \mathcal{L}] \in \vartheta :$	for all encrypted term
$V_0 \in \vartheta_0 \wedge V_1 \in \vartheta_1 \Rightarrow$	if values match
$V_2 \in \rho(x) \wedge$	x has the value V_2
$\ell' \notin \mathcal{L} \vee \ell \notin \mathcal{L}' \Rightarrow (\ell, \ell') \in \psi \wedge$	check annotations
$(\rho, \kappa) \models P : \psi$	analyse the rest
<hr/>	
$(\rho, \kappa) \models \text{decrypt } E \text{ as } \{E_1; x\}_{E_0}^{\ell'} [\text{orig } \mathcal{L}'] \text{ in } P : \psi$	

Correttezza dell'analisi

Teorema 1 (subject reduction)

If $(\rho, \kappa) \models P : \psi$ and $P \rightarrow Q$ then also $(\rho, \kappa) \models Q : \psi$

“Se una stima soddisfa un certo stato e da quello stato posso passare ad un altro stato, allora anche quello stato soddisfa”

Teorema 2 ($\psi = \emptyset$ means we're happy)

If $(\rho, \kappa) \models P : \emptyset$ then the reference monitor cannot abort the execution of P .

“Se non c'è nessun Error Component, il reference monitor non può abortire l'esecuzione di P ”

Validating Authentication

Definition

P guarantees dynamic authentication if $P \mid Q$ cannot abort regardless of the choice of the attacker Q .

Definition

P guarantees static authentication if $(\rho, \kappa) \models P : \emptyset$ and $(\rho, \kappa, \emptyset)$ satisfies the formula describing the attacker.

Theorem

If P guarantees static authentication then P guarantees dynamic authentication.

Implementation

Transform the analysis into (an extension of) Horn clauses.

Calculate the solution using the Succinct Solver.

Main challenge:

The analysis is specified using the infinite universe of terms.

Use an encoding of terms in tree grammars where terms are represented as a finite number of production rules.

Runs in polynomial time in the size of the process P .

L'analisi trova i classici problemi già riscontrati nei protocolli principali, solitamente perché distinzioni cruciali non sono sufficientemente chiare (es. *identità e ruoli*) e che molti protocolli diventano non sicuri quando vecchie chiavi di sessione sono compromesse. A vantaggio di ciò, si trovano pochi falsi positivi e una procedura di validazione in tempo polinomiale ed è estendibile anche con la crittografia asimmetrica.