

APPLICATION LEVEL SANDBOXING

Compartmentalisation

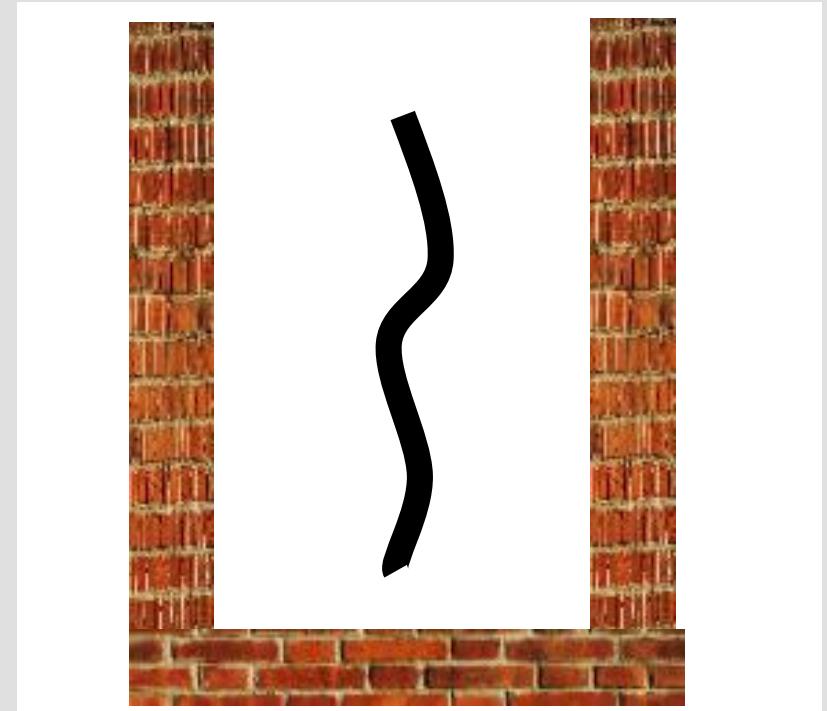
Compartmentalisation can be applied on many levels

1. In an IT systems
 - different machines for different tasks
2. On a single computer
 - different processes for different tasks
 - different user accounts for different tasks
 - use virtual machines to isolate tasks
 - partition your hard disk & install two OSs
3. **Inside a program / application / app**
 - **different 'modules' with different tasks**



Isolation

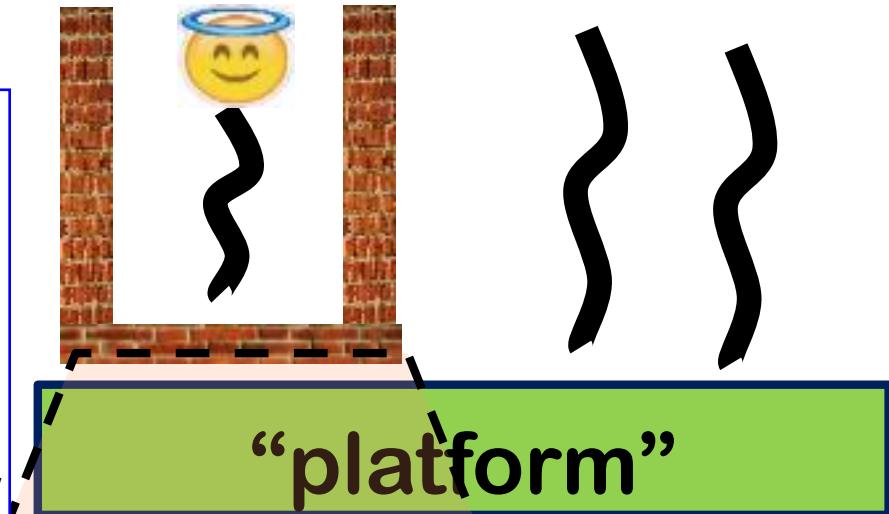
- Isolation is a useful security property for programs and processes
- ‘isolation’ can be broken down into a combination of **confidentiality & integrity of data & code,**

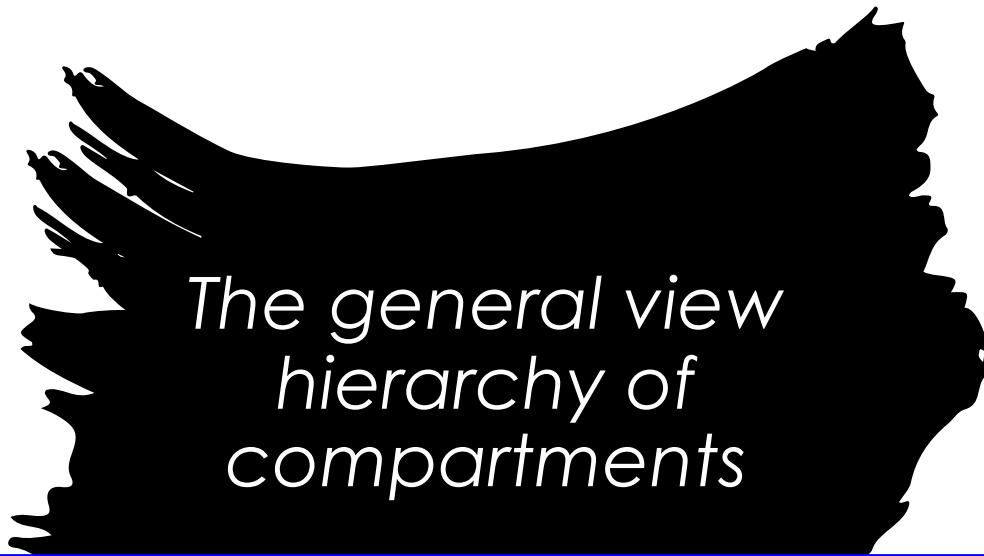


SANDBOX

protect a **trusted process**
from **outside attacks**

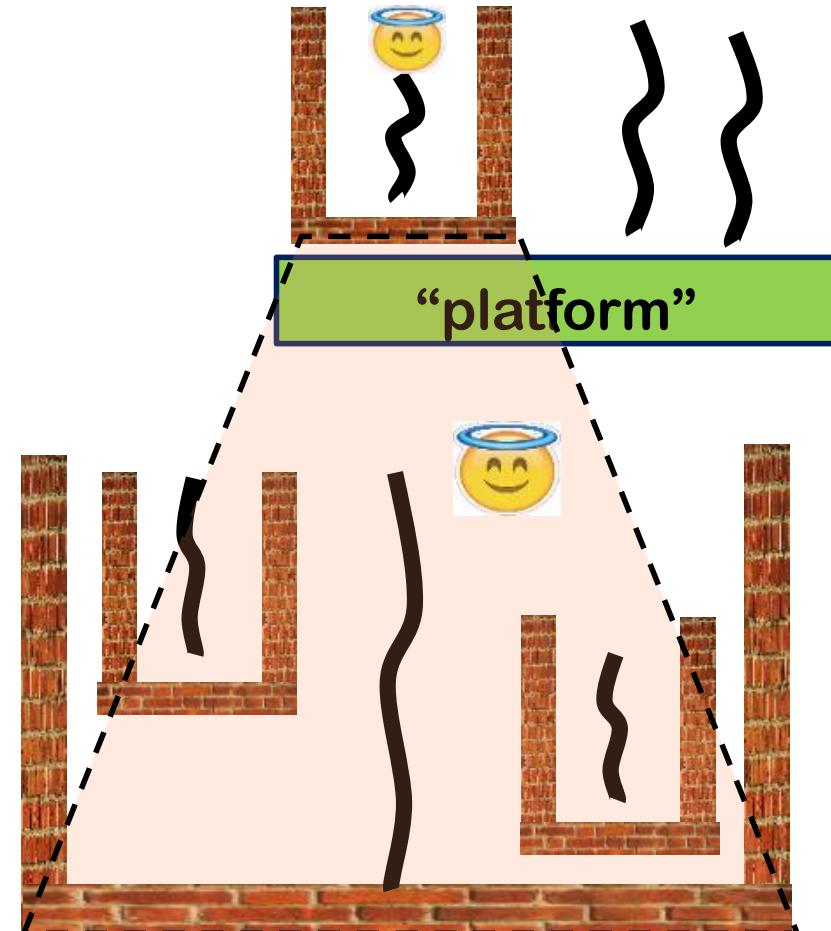
Dal punto di vista della sicurezza l'idea è quella che io ho una struttura di compartimento molto più forte che ci permette di isolare un flusso di esecuzione o un comportamento lì è semplicemente in questa figura, è semplicemente rappresentata da un flusso potete immaginarlo come un comportamento di un applicazione di un programma tenendo conto di diverse combinazioni di fattori di sicurezza, in modo particolare della confidenzialità quindi, vuol dire che l'informazione presente all'interno del flusso di esecuzione non devo uscire fuori, integrità, vuol dire che non si possano dall'esterno con le interfacce aperte poter modificare dei dati sensibili dell'applicazione, ma l'integrità non è soltanto dei dati ma anche del codice, cioè non si può modificare il codice in esecuzione. Quindi sostanzialmente uno potrebbe usare questo meccanismo che normalmente è la nozione di compartimento modulo in un qualcosa, in un entità che vuole isolare un contesto trusted da eventuali attacchi dall'esterno, normalmente questi si chiamano sandbox .





The general view hierarchy of compartments

in generale questa cosa qui può essere vista gerarchica, cioè noi potremmo avere una sandbox che al suo interno è costituita da altre sandbox che isolano dei comportamenti con delle interfacce ben definite. Questi comportamenti possono essere trusted/untrusted o composti assieme.



Gli aspetti interessanti di questo meccanismo di definire i compartimenti stagni e di definire un meccanismo di interazione tra questi compartimenti stagni sono: uno è il solito, la TCB. Il punto è che noi stiamo mettendo delle funzionalità all'interno di una sandbox e quindi stiamo riducendo lo spazio d'attacco perché a questo punto stiamo mettendo la trust computer base per quella funzionalità all'interno della sandbox, vuol dire che la TCB deve contenere anche i meccanismi di enforcement del isolamento e del modo in cui si stanno isolando dei comportamenti, quindi ha un effetto sulla TCB del linguaggio che è in esecuzione o in più in generale dell'applicazione che è in esecuzione. La seconda cosa è che uno vorrebbe esprimere e definire delle politiche di sicurezza che hanno a che vedere con l'isolamento e quindi ci sono delle domande di espressività. Quali sono le politiche che permettono di essere utili e espansive per isolare dei comportamenti di programmi? come le descriviamo e soprattutto qual è il meccanismo di enforcement di queste politiche? L'altra parte ha un aspetto più di natura ingegneristica, dato che noi siamo in un meccanismo dove vogliamo essere gerarchici, questo vuol dire che possiamo comporre politiche, possiamo comporre dei compartimenti, ma se ogni compartimento ha una sua caratterizzazione di una politica, dobbiamo anche comporre politiche, quindi potrebbe essere complesso avere un meccanismo di composizione delle politiche complessivo e sicuramente sono cose che devono essere considerate. Infine, dato che i vari compartimenti comunicano, hanno un'interfaccia ben definita ci potrebbero essere dei canali di comunicazione, e uno deve stare attento a come questi canali di comunicazione trasferiscono le informazioni.

Compartments: Issues!!

1. **What is the Trusted Computing Base (TCB) ?**

- **Compartmentalising critical functionality inside a trusted process reduces the TCB for that functionality, but increases the TCB with the TCB of the enforcement mechanism**

2. **Can the compartmentalisation be controlled by policies?**

- **How expressive & complex are these policies?**
- **Expressivity can be good, but resulting complexity can be bad...**

3. **What are input & output channels?**

- **We want exposed interfaces to be as simple, small, and just powerful enough**

Compartments: Access Control

Compartments may provide access control mechanisms that can be configured.

This involves:

- 1. Rights/permissions**
- 2. Parties** (eg. users, processes, components)
- 3. Policies** that give rights to parties
 - specifying who is allowed to do what
- 4. Runtime monitoring** to enforce policies,
 - which becomes part of the TCB

un esempio tipico di politica che viene fatto è il controllo degli accessi, in modo particolare controllo degli accessi come voi sapete da altri corsi ha che vedere con i diritti, quindi quali sono i permessi che uno dà alle entità, le parti che sono l'entità a cui vengono dati i diritti, che può essere a seconda del livello dell'astrazione, possono essere utenti, processi, componenti, moduli, eccetera, eccetera... le politiche, cioè quali sono i meccanismi che ci permettono di descrivere chi fa cosa e come quindi uno permette di dare un diritto a un'entità rispetto all'altra e poi il meccanismo di enforcement a tempo di esecuzione che è quello che garantisce che la politica in vigore è effettivamente eseguita e ovviamente il meccanismo di enforcement fa parte della TCB.

Se noi per il momento lasciamo perdere la sicurezza e affrontiamo il problema dal punto di vista dell'ingegneria del software, vedete che questo che noi stiamo dicendo è una tipica caratteristica dei sistemi software, cioè l'idea è che il sistema è diviso in parti, le chiamiamo componenti, ogni componente ha un compito che deve eseguire e la caratteristica dei componenti è che ogni componente deve avere un sistema minimale di diritto degli accessi perché?

Intermezzo: Secure Software Engineering

- 1. Divide systems into chunks – aka compartments**
 - Different compartments for different tasks
- 2. Give minimal access rights to each compartment**
 - principle of least privilege
- 3. Have strong encapsulation between compartments**
 - flaw in one compartment cannot corrupt others
- 4. Have clear and simple interfaces between compartments**
 - expose minimal functionality

Perché uno vuole utilizzare quel meccanismo standard dell' ingegneria del software che si chiama il principio del minimo privilegio, cioè io associo a un entità software di esecuzione il minimo insieme dei diritti che gli permette di effettuare il task, non gliene do molti di più. Sono incapsulati, quindi vuol dire che ci sono dei compartimenti che evitano di vedere delle proprietà all'interno e quindi che siano visibili dall'esterno delle proprietà significative. In termini di sicurezza i vari moduli e i vari comportamenti isolati non si possono corrompere uno non può corrompere un altro e hanno delle interfacce chiare di comportamento, dove si permette di interagire, quelle che prima chiamavamo i canali di comunicazione.

Standard concepts in SE!!!

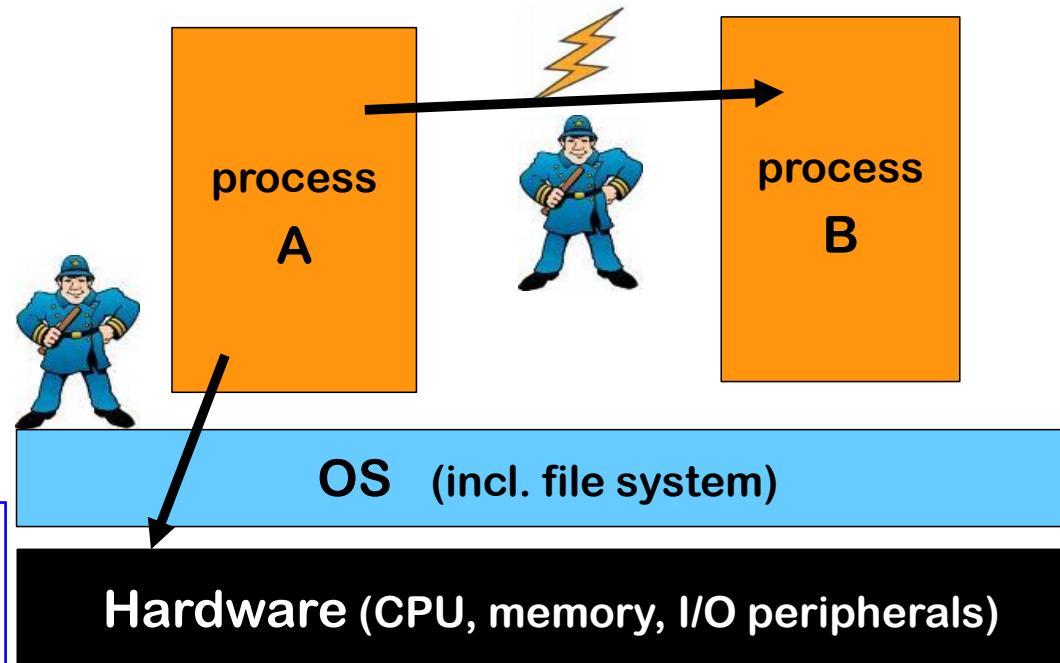
Benefits

1. Reduces TCB for certain security-sensitive functionality
2. Reduces the impact of any security flaws.

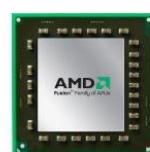
Example (monolithic design)

- Requirements
 - Receive and send email over external network
 - Place incoming email into local user inbox files
- Sendmail
 - Traditional Unix
 - Monolithic design
 - Historical source of many vulnerabilities

OS Access Control



Continuando a parlare del controllo degli accessi, se noi lo vediamo a livello del sistema operativo, il controllo degli accessi è tra i processi del sistema operativo, ci sono dei meccanismi che definiscono come i processi hanno i diritti



Issues

1. The size of the TCB

- The TCB for OS access control is **HUGE** so there will be security flaws in the code.

2. Too much complexity

- The languages to express access control policy are very complex, so people will make mistakes

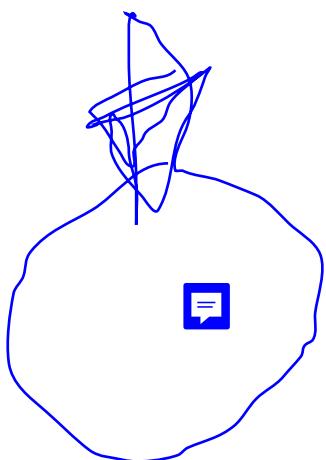
3. Not enough expressivity / granularity

- OS cannot do access control within process, as processes as the 'atomic' units

Note (personal): fundamental conflict between the **need for expressivity** and the **desire to keep things simple**

L'aspetto critico del controllo degli accessi a livello del sistema operativo è la dimensione della TCB che è molto grande proprio per il fatto che il kernel del sistema operativo è molto grande e questo è il motivo per cui ci sono attacchi a livello del sistema operativo. E' molto complicata perché i linguaggi per definire le politiche sono molto complicati e spesso gli amministratori del sistema fanno degli errori di accesso e della politica e poi ha un livello di granularità troppo grossa, cioè ad esempio, la politica la definisco dal processo io invece vorrei poter andare dentro al processo e poter definire delle politiche di controllo direttamente di porzioni di codice del processo.

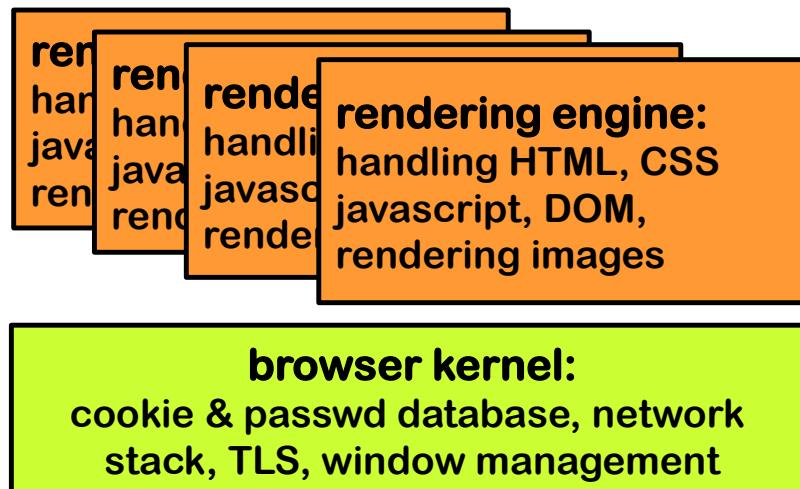
OS vs Browser

- 
- Operating system
 - Subject: Processes
 - Has User ID (UID, SID)
 - Discretionary access control
 - Objects
 - File
 - Network
 - ...
 - Vulnerabilities
 - Untrusted programs
 - Buffer overflow
 - ...
 - Web browser
 - Subject: web content (JavaScript)
 - Has “Origin”
 - Mandatory access control
 - Objects
 - Document object model
 - Frames
 - Cookies / localStorage
 - Vulnerabilities
 - Cross-site scripting
 - Implementation bugs
 - ...

The web browser enforces its own internal policy. If the browser implementation is corrupted, this mechanism becomes unreliable.



The Chrome browser process is split into multiple OS processes



One rendering engine per tab,
plus one for trusted content
(eg HTTPS certificate warnings)

*No access to local file system
and to each other*

One browser kernel
with *full user privileges*

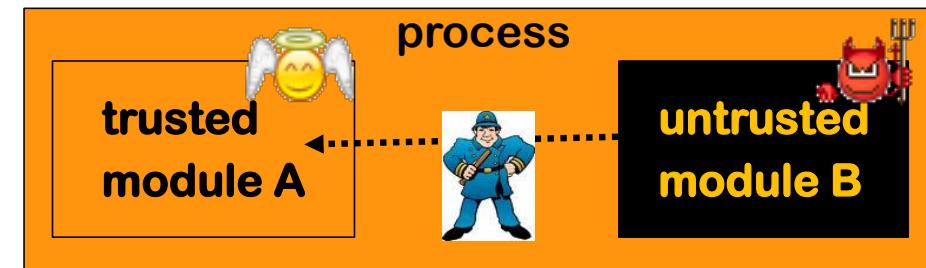
- (Complex!) rendering engine is black box for browser kernel
- Plugins also run as different processes
- Running a new process per domain can enforce the restrictions of the SOP (Same Origin Policy)
- *Advantage: TCB for certain operations drastically reduced*

access control at the language
level



Safe programming languages and access control

In a safe programming language, access control can be provided at language-level, because interactions between components can be restricted & controlled by language abstractions



Vogliamo fare in modo di avere un meccanismo di granularità differente, vogliamo poter vedere il fatto che un piccola componente del programma ha dei controlli e vogliamo avere dei meccanismi di enforcement delle politiche di controllo degli accessi a livello del linguaggio di programmazione quindi vogliamo avere un meccanismo più articolato e di una granularità più bassa per affrontare questo stesso aspetto

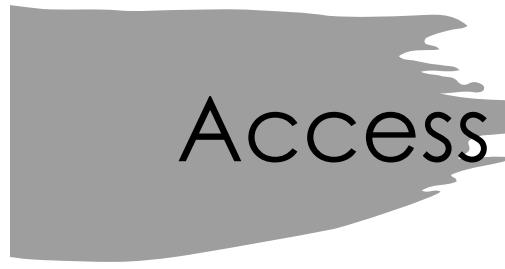
Access Control at the language level

AC at the language level makes it possible to have security guarantees in the presence of untrusted code (which could be malicious or just buggy)

1. Without **memory-safety**, this is impossible. Why?
2. Without **type-safety**, it is hard. Why?

allora adesso ci domandiamo rispetto a quello che abbiamo visto in questi giorni, quindi rispetto agli aspetti di memory safety e gli aspetti di type safety come si comporta e come opera il controllo degli accessi. Quindi immaginando di avere un linguaggio di programmazione che non è memory safe, è possibile avere il controllo degli accessi? Ovviamente avere il controllo degli accessi a livello del linguaggio di programmazione, quindi avere un linguaggio di programmazione che permette di definire delle politiche di controllo accessi a grana molto fine quindi vuol dire che ci sono delle parti di codice che hanno delle proprietà di controllo accessi che sono diverse da altre parti del codice. Vogliamo che l' enforcement della politica di controllo accessi sia fatta dall'engine del linguaggio di programmazione e vogliamo anche che il programmatore sia in grado di programmare la propria politica di controllo degli accessi e quindi la prima cosa che ci domandiamo è: come la politica di controllo degli accessi impatta sulla memory safe?

Immaginiamo di avere una componente B del programma scritto, se può accedere alla parte della memoria di un'altra componente del programma allora a questo punto accede indirettamente perché non siamo memory safe, quindi facendo delle operazioni di buffer overflow o quello che sia, chiaramente rompiamo le proprietà di controllo degli accessi, quindi una caratteristica che deve avere è la memory safety. Altra cosa che ci domandiamo è: come impatta e quali sono le relazioni del controllo degli accessi al livello del linguaggio con la type safety? Anche in questo caso, se noi non siamo type safe, potremmo avere dei grossi problemi ad avere l'enforcement della politica di sicurezza a livello del linguaggio perché? Perché se io sono una sotto parte del programma B che passa un dato a un'altra sotto parte del programma A con cui ho la possibilità di passargli dei dati e questi dati non sono type safety, non sono tipati correttamente quindi se non sono dei dati volutamente errati allora a questo punto mi si rompe anche in questo caso il controllo degli accessi a livello del linguaggio



Access Control at the language level

AC at the language level makes it possible to have security guarantees in the presence of untrusted code (which could be malicious or just buggy)

1. Without **memory-safety**, this is impossible. Why?
 - Because B can access any memory used by A
2. Without **type-safety**, it is hard. Why?
 - Because B can pass ill-typed arguments to A's interface

Code-based Access Control

allora quello che succede nei linguaggi di programmazione moderni è che il controllo degli accessi è basato su una nozione di code-based vuol dire su una nozione che dipende da dov'è il codice che è stato originato. Infatti molto spesso, invece di trovare semplicemente la parola access control viene trovata la parola codebase access control e questa qui è una cosa che hanno linguaggi di programmazione o piattaforme per i linguaggi di programmazione, come java nel caso del linguaggio, dotnet nel caso delle piattaforme dove diversi linguaggi possono essere compilati e la caratteristica che il controllo degli accessi sul codice si basa sul controllo degli accessi del sistema operativo e ha un livello di granularità fine perché permette di vedere e di trattare in modo differente parti del programma e di programmarsi le politiche di controllo degli accessi.

Code-based Access Control

Languages & platforms such as Java & .NET provide code-based access control

- **Code-based access control treats different parts of a program differently**
- **Code-based access control works on top of the user-based access control of the OS**

Code-based access control: ingredients

Ingredients for code-based access control, as for any form of access control

- 1. permissions**
- 2. protection domains**
- 3. policies**

gli ingredienti tipici del controllo degli accessi, quindi ci sono i permessi, ci sono le politiche e poi ci sono i soggetti dell'azione che si chiamano nel caso del code based i domini di protezione.

Code-based Access Control in Java

```
grant  
codebase "http://www.di.unipi.it/lbt", signedBy "Informatica",  
{ permission  
    java.io.FilePermission "/home/lbt/ferrari","read";  
};
```

```
grant  
codebase "file:/*"  
{ permission  
    java.io.FilePermission "/home/lbt/ferrari","write";  
}
```



Code-based Access Control in Java

```
grant  
codebase "http://www.di.unipi.it/lbt", signedBy "Informatica",  
{ permission  
    java.io.FilePermission "/home/lbt/ferrari", "read";  
};
```

```
grant  
codebase "file:/*"  
{ permission  
    java.io.FilePermission "/home/lbt/ferrari", "write";  
}
```

PROTECTION DOMAIN

Protection domains

1. Where did it come from?

- where on the local file system (hard disk) or where on the internet

2. Was it digitally signed and if so by who?

- using a standard PKI

allora, nel gergo di java, il codebase, quindi l'indirizzo dell'entità che mette dei file a disposizione o del codice a disposizione, si chiama dominio di protezione. Quindi sta definendo sul dominio di protezione quali sono i permessi. Il dominio di protezione dice sostanzialmente dove sono le entità: se sono su sistema locale o se sono in rete, dice da chi sono firmate, si possono usare dei meccanismi standard di cifratura. Sono esattamente delle cose abbastanza statiche.

JVM and Protection Domains

When loading a component (e.g. an object), the Java Virtual Machine (JVM) consults the security policy and stores the permissions

A questo punto quello che dobbiamo capire, quello che succede nel run time, cioè quando viene caricato una componente, ad esempio un oggetto che è una istanza di una classe, noi sappiamo che cosa succede che il class loader va a vedere se questa classe è stata già caricata o meno, dato che il class loader il meccanismo della JVM ha un meccanismo lazy cioè le classi vengono caricate nel run time la prima volta, quando carico una classe carico tutto quello che sta sopra la gerarchia di quella classe.

Permissions

1. Permissions are Java Objects

1. permissions are an instance of the **AllPermission class**,

2. Permissions represent a right to perform some actions.

Examples:

- FilePermission(name, mode)
- NetworkPermission
- WindowPermission

A questo punto quello che succede immaginiamo che non l'abbiamo ancora trovata prendiamo la classe associata all'oggetto, carichiamo la classe tutta sua, la sua gerarchia verso l'alto e a questo punto andiamo a vedere dove sta questa classe, andiamo a vedere il suo codebase e andiamo a vedere la politica di sicurezza e memorizziamo nelle informazioni associate alla classe, i permessi associati alla politica di sicurezza. Questo è una componente del run time che quindi, oltre all'aspetto di caricamento, va a ispezionare le informazioni associate al .class, vede qual è la politica e memorizza i permessi associati a quella classe. I permessi sono degli oggetti java, quindi che cosa vuol dire? vuol dire che sono degli opportuni oggetti che definiscono come fare delle azioni. Le azioni possono essere ad esempio: accesso un file, quindi si dice il tipo del file modo di accesso, lettura scrittura, possono essere anche dei permessi di rete, quindi possa leggere o scrivere delle informazioni su un socket oppure possono essere dei permessi chiamati windows description che hanno a che vedere con il toolkit per la gestione delle finestre che non hanno dei diritti d'accesso a poter eseguire delle funzionalità associate a un particolare bottone dell'interfaccia grafica.

Permissions

- Permissions have a **set semantics**, so one permission can be a superset of another one.
 - `FilePermission("*", "read")` includes `FilePermission("some_file.txt", "read")`
- Developers can program and define new custom permissions.

dal punto di vista concettuale i permessi sono visti come un insieme. Vuol dire che i permessi sono un insieme di permessi di base, lettura, scrittura e poi, dato che ho la normale nozione di inclusione tra insiemi, mi permettono di dire che un permesso è contenuto in un altro permesso. Nell esempio che stiamo vedendo qui io stavo scrivendo che ho su una certa directory un file permission che mi permette di leggere tutti i file all'interno di questa directory ovviamente questo mi dice che se ho un file all'interno di questa directory con il nome `some_file.txt` il permesso di leggere è sicuramente incluso nel permesso più generale proprio per il fatto che hanno una semantica a insiemi. La cosa importante è che uno si può programmare i permessi, quindi, utilizzando il fatto che i permessi sono esprimibili a livello del linguaggio, io posso programmare i miei meccanismi di definizione dei permessi.

abbiamo la classe trusted, un package che è affidabile, che con il metodo m1 chiama le API di system, chiama le primitive che permettono di cancellare un file. Essendo la componente trusted abbiamo il monitor, la java virtual machine che va a controllare i permessi. Il monitor che li è rappresentato da un poliziotto arcigno controlla se nel class file di trusted abbiamo il permesso per cancellare sotto quella opportuna directory, se abbiamo il permesso, lo cancelliamo. Se abbiamo un entità che è il diavolo che non è associato il permesso di cancellare il solito meccanismo dell'enforcement che è il solito poliziotto arcigno vieta il permesso.

P
I
C
T
O
R
I
A
L
L
Y

```
package trusted;  
class Trusted {  
    void m1 ()  
    { ....  
        System.delete file;  
    }  
}
```



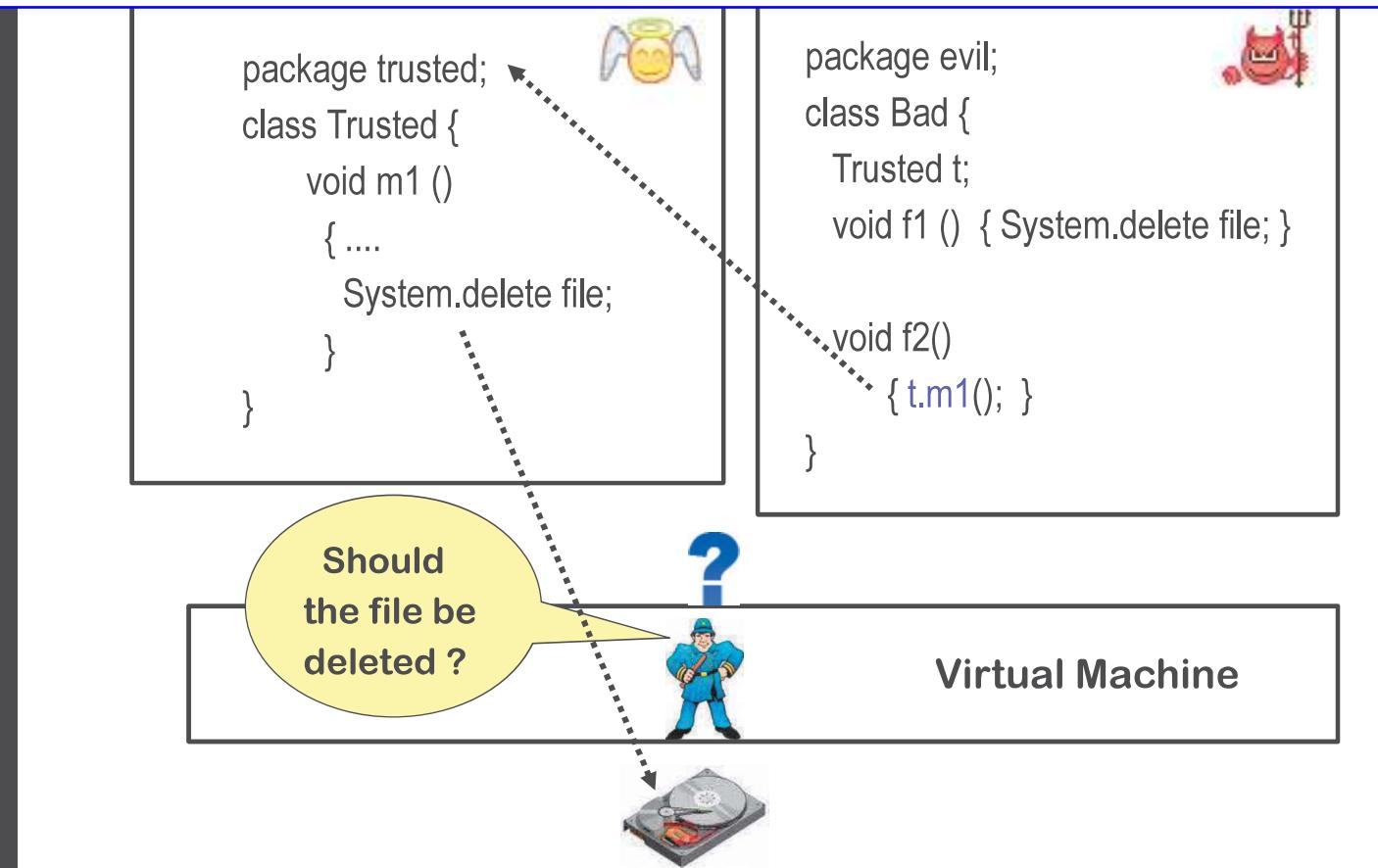
```
package evil;  
class Bad {  
    void f1 () { System.delete file; }  
}
```



Virtual Machine



a immaginiamo che il diavolo è un po più sottile, come strutture di programmazione e allora il nostro meccanismo trusted è sempre quello di prima, l'unica cosa è che il diavolo è riuscito a capire che il metodo m1 è pubblico allora cosa fa? Il diavolo crea la possibilità di vedersi localmente la componente trusted e poi chiama una volta che ha attivato la componente trusted chiama il metodo m1 che fa l'operazione di cancellare il file. A questo punto uno si domanda: ma il file lo devo cancellare o meno o in modo particolare, il meccanismo dell'enforcement si pone la domanda, devo cancellare il file o non lo devo cancellare?



Method Invocation: management

There are different possibilities to manage at run-time method invocation.

- Take #1: allow action if top frame (aka activation record) on the stack has permission for that action
- Take #2: only allow action if all frames (activation records) on the stack have permission
- Take #3: ...

Nell esempio della slide di prima abbiamo un accesso indiretto ad una risorsa critica, in modo particolare al file e abbiamo che l'ultimo passo di questo accesso indiretto a questa risorsa critica è fatto da una componente trusted, quindi se io vado a vedere la componente trusted e isolo e vedo solo il comportamento della componente trusted, so che potrei in modo particolare accedere, fare delle operazioni sulle strutture critica però dovrei anche vedere come sono arrivato sino a questo punto e infatti il punto è come la gestione dell' invocazione di metodi.



Method Invocation: management

There are different possibilities to manage at run-time method invocation.

- Take #1: allow action if top frame (aka activation record) on the stack has permission for that action
- **Our example (evil -> angel): file is deleted**
- Take #2: only allow action if all frames (activation records) on the stack have permission
- **Our example (evil -> angel): permission denied since the attacker has non permission to delete files.**

Discussion

- Take #1: The solution is very dangerous: a class may accidentally expose dangerous functionality
- Take #2: The solution is very restrictive: a class may want to, and need to, expose some dangerous functionality, but in a controlled way
- We need a ore flexible solution: **stack walking** also known as **stack inspection**

la strategia 1 non la prendiamo in considerazione perchè è una strategia perdente, la strategia 2 invece richiede una modifica non banale del run time e richiede una soluzione altamente complicata e poi capirete tra un attimo perchè, che si chiama stack walking o stack inspection, cioè richiede la possibilità di ispezionare a tempo di esecuzione la struttura del run time stesso, andare a esaminare tutto quello che c'è in quel momento nello stack a tempo di esecuzione, quindi è una operazione che produce un overhead.

EXPOSING FUNCTIONALITIES (insecurely)

```
Class Trusted{

    public void unsafeMethod(File f) {
        delete f; } // Could be abused by evil caller

    public void safeMethod(File f) {
        .... // lots of checks on f;
        if all checks are passed, then delete f; }

        // Cannot be abused, assuming checks are bullet-proof

    public void anotherSafeMethod() {
        delete "/tmp/bla"; }

        // Cannot be abused, as filename is fixed.

        // Assuming this file is not important..
```

Andiamo per il momento a vedere invece dei pattern di programmazione, quindi supponiamo di avere il solito meccanismo dei permessi, il solito meccanismo di controllo degli accessi a livello del linguaggio di programmazione e definiamo ad esempio in modo particolare questa classe trusted che ha un metodo unsafe che fa la operazione di eliminazione di un file. Ovviamente questo potrebbe essere utilizzato da un evil e questa è la generalizzazione dell'esempio che abbiamo fatto in precedenza, però uno potrebbe scrivere un metodo un po più safe, prima faccio un po di check, vado a vedere i permessi, se tutti i permessi sono superati, allora faccio l'operazione di delete. Il terzo metodo che sempre può essere messo e che io vado a cancellare un file e scrivo qual è il file che devo cancellare, quindi chiaramente il file è fissato e assumo ovviamente che il file che voglio cancellare non contenga dell'informazione critica.

allora come posso operare a livello del linguaggio? beh, posso mettere i modificatori, quindi posso avere già un primo meccanismo che mi dice: il primo metodo unsafe è privato quindi che cosa vuol dire? vuol dire che l'entità untrusted non può accedere , però quello che potrebbe accadere è che io progetto in modo tale che l'entità che sono untrusted a causa di una sequenza di metodi e di chiamate indirette accedono direttamente alla fine di una lunga sequenza di invocazione di metodi. Allora, proprio in quel caso lì, non mi basta il meccanismo dei modificatori, ma devo proprio esattamente fare la stack inspection.

Using visibility to control access?

```
Class Trusted{

    private void unsafeMethod(File f) {
        delete f; } // Could be abused by

    public void safeMethod(File f) {
        .... // lots of checks on f;
        if all checks are passed, then delete
            // Cannot be abused, assuming checks are bullet-proof

        public void anotherSafeMethod() {
            delete "/tmp/bla"; }

            // Cannot be abused, as filename is fixed.

            // Assuming this file is not important..
    }
}
```

Making the unsafe method private & hence *invisible* to untrusted code helps, but is error-prone. Some public method may call this private method and indirectly expose access to it
Hence: stackwalking

Ogni volta che io ho un accesso a una risorsa critica, ci metto una guardia e questa guardia si chiama demandPermission, cioè vado a vedere se quel particolare permesso in questo caso il parametro P della guardia del demandPermission è dato a quella particolare invocazione, cioè se quella particolare invocazione ha il diritto di eseguire e ha il permesso di eseguire l'operazione descritta da P. Se ha il diritto di l'accesso, se non ha il diritto devo stoppare l'esecuzione. Qual'è l'algoritmo dunque il meccanismo di enforcement che ogni volta che vedo la demandPermission mi risolve questo aspetto? Si chiama stack inspection ed è stata implementata per la prima volta da netscape la versione se non ricordo male la versione 4.0 e poi è stata adottata ad esempio nei browser internet explorer, ma poi è stata adottata nei linguaggi e nelle piattaforme.

Stack inspection

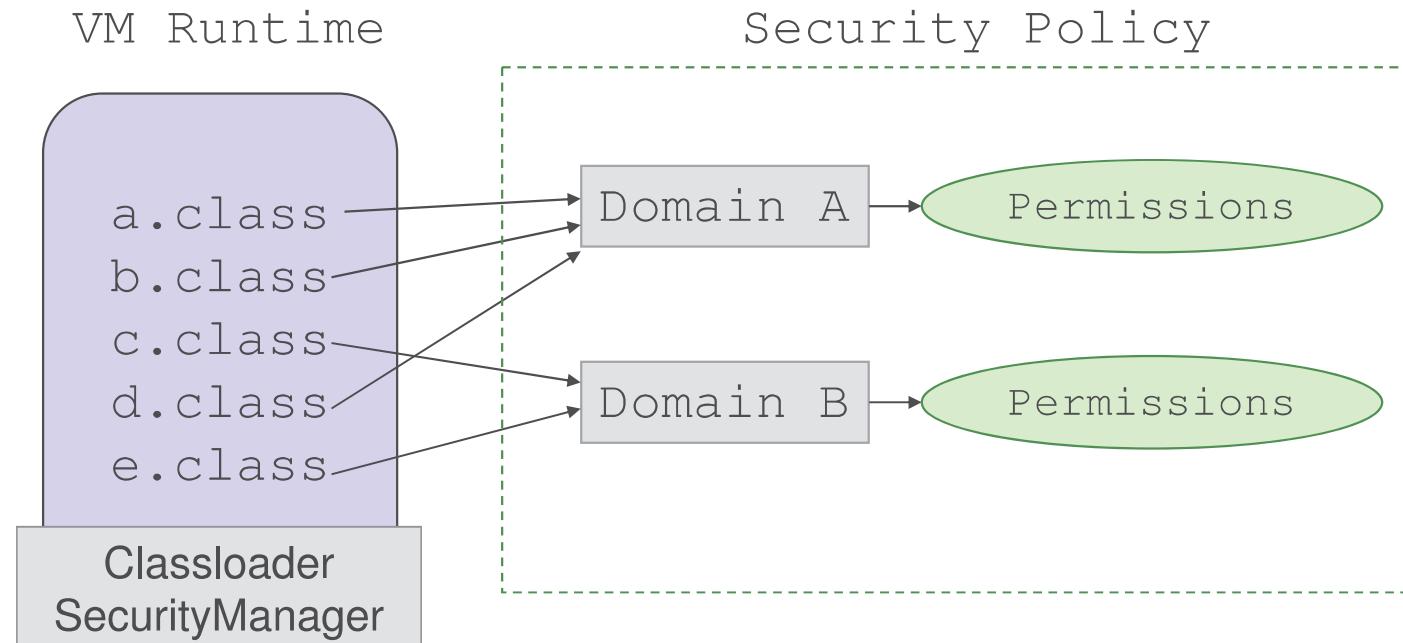
Every resource access or sensitive operation is protected by a guard call (**demandPermission(P)**) for an appropriate permission P

Constraint: No access without permission!

The algorithm for granting permission is based on stack inspection (also called stack walking)

Stack inspection first implemented in Netscape 4.0, then adopted by Internet Explorer, Java, .NET

Allora supponiamo che nel run time di java hanno caricato un certo numero di file .class in questo caso sono a b c d e, quindi all'interno di ogni file .class, oltre alle solite informazioni che sono presenti nel file .class, quindi la cosa in più ci sono tutte le informazioni dei nomi che servono per poter risolvere i nomi a tempo di esecuzione ,ci sono le informazioni relative alle tabelle dei metodi dove stanno i metodi e ci sono anche l'informazione sul dominio di protezione associato al protection domain, associato alla classe e associato al dominio di protezione ci sono i permessi, sono esattamente i grant che abbiamo visto prima, quindi ad esempio nel nel caso particolare di questa trasparenza che stiamo vedendo, abbiamo le classi a b e d che appartengono allo stesso dominio di protezione A che è caratterizzato da quell insieme di permessi, mentre le classi c ed e hanno il dominio di protezione B caratterizzato da altri permessi.



JAVA SECURITY MODEL

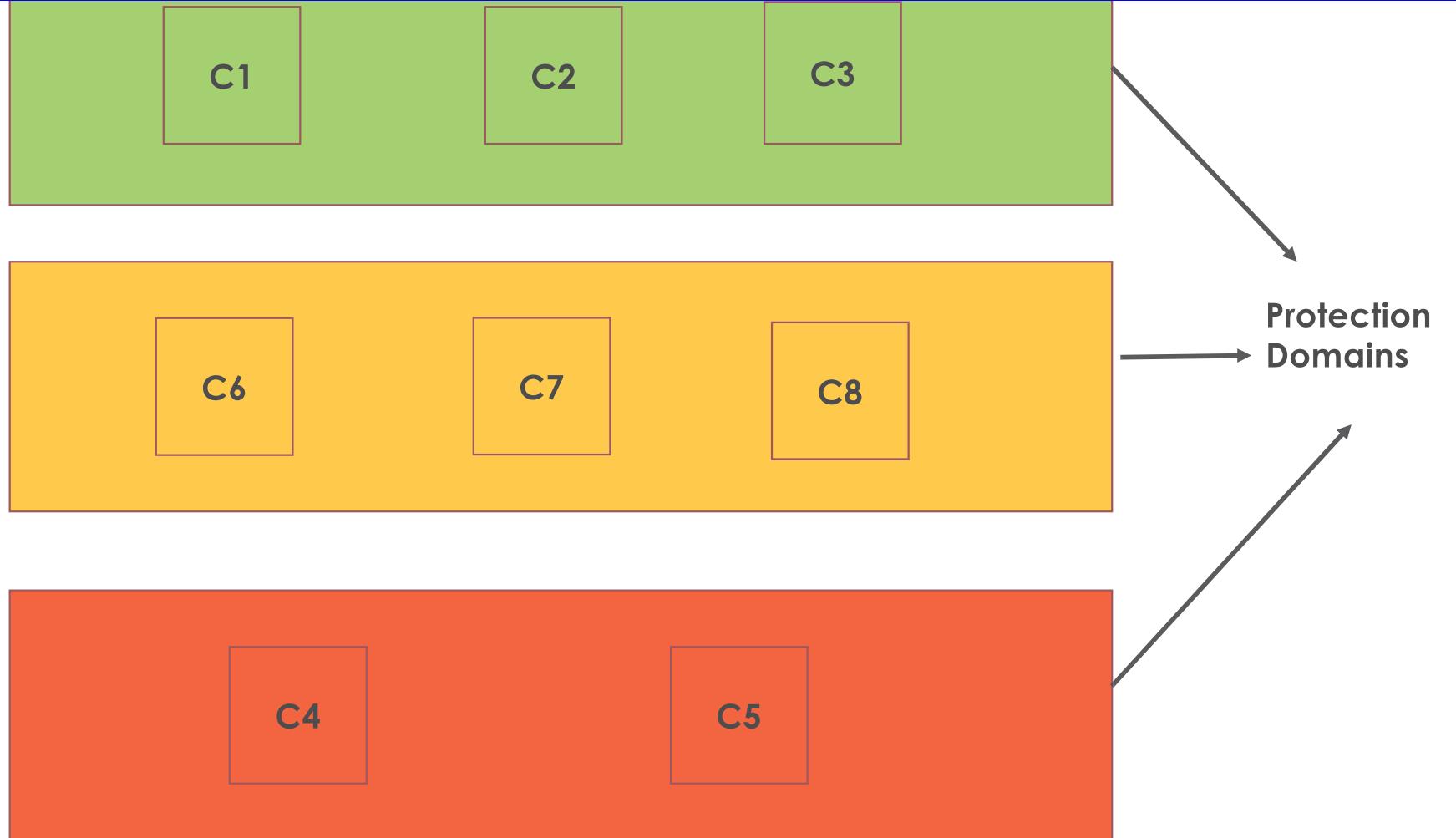
Quello che dicevamo pocanzi è che quando la classe viene caricata, il class loader, oltre a mettere questa informazione del .class nel run time fa l'ispezione del dominio di protezione associa quella classe al dominio di protezione ed associa a quel dominio di protezione i permessi definiti dai grant. Chiaramente il meccanismo di caricamento del classe al solito lazy, cioè nel senso una classe viene caricata ogni volta che viene trovata e non cambia rispetto al normale trattamento del caricamento delle classi, così che la modifica della JVM è compatibile con le versioni precedenti della JVM.

Stack Inspection

- Stack frames are annotated with their protection domains and any enabled privileges.
- During inspection, stack frames are searched from most to least recent:
 - **fail** if a frame belonging to someone not authorized for privilege is encountered
 - **succeed** if activated privilege is found in frame

A questo punto, quello che cambia è la gestione dello stack, la gestione dello stack è che ogni frame in java contiene tutte le informazioni sui parametri locali, sul parametro del metodo, sulle variabili locali e poi al suo interno ha altre informazioni, utilizzate per l'esecuzione delle operazioni del metodo. La JVM è una macchina a stack, quindi cosa vuol dire? vuol dire che, ad esempio, un'operazione di addizione prende due argomenti questi due argomenti li trovo sullo stack nel record di attivazione del metodo corrente, prende i due argomenti, fa la somma, fa un'operazione di pop dello stack locale e poi fa un'operazione di push del risultato dell'operazione. Tra le diverse informazioni che sono presenti nel record di attivazione del metodo sullo stack c'è anche quali sono i privilegi. Quindi come è il dominio di protezione e quali sono i permessi associati a quel dominio di protezione. Quello che fa l'operazione di stack inspection è di andare a vedere se quel p, è abilitato dallo stack corrente, quindi quello che fa una visita dello stack, quindi da quello in testa lo stack fino all'inizio dello stack e va a vedere se ha il permesso e abilitato dal dominio di protezione di tutti i metodi che stanno sulla stack. Se trova facendo questa visita in un'entità che non abilita il permesso fallisce, invece da successo se tutti gli elementi sono tali da permettere quel permesso. Qual è il dominio di protezione vero? Non quello associata staticamente, quello associato staticamente è quello che noi vediamo andando a vedere il dominio di protezione e i permessi che sono ottenuti caricando, questo è il dominio di protezione statico. Il dominio di protezione dinamico invece ha a che vedere sulla sequenza delle invocazioni dei metodi: allora qual è il dominio di protezione associato al metodo che è sulla testa dello stack? E' sostanzialmente l'intersezione dei domini di protezione di tutte le invocazioni che sono attualmente attive sullo stack perché ha il permesso di eseguire una certa operazione se tutti i record di attivazione sullo stack hanno quel permesso.

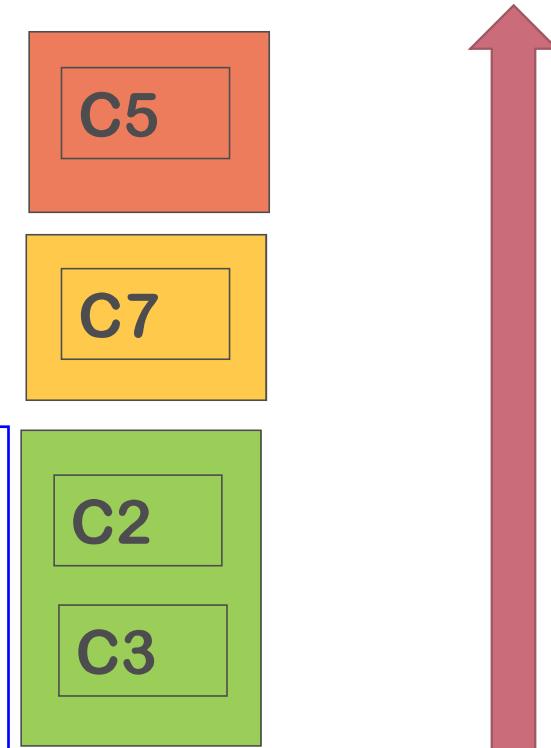
Allora andiamo a vedere un po' di esempi per capire come sono fatti: immaginiamo di avere le classi verdi, arancioni, rosse e ogni classe ha un suo dominio di protezione, quindi non diciamo come sono fatti i permessi stiamo dicendo che abbiamo il verde, il rosso e l'arancione questi sono i tre domini di protezione.



CALL SEQUENCE FOR THREAD T

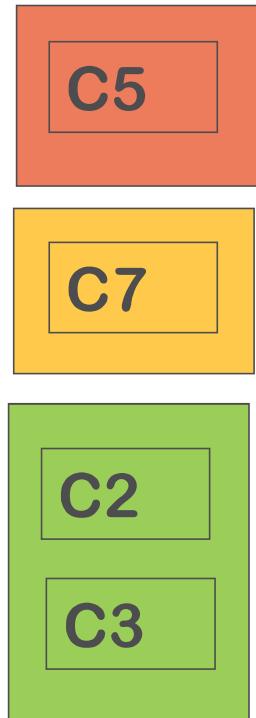
**C5 called by
C7 called by
C2 called by
C3**

Adesso supponiamo di avere sulla stack un certo thread che ha questa sequenza di esecuzione: quando io scrivo c5 è chiamato da significa che un metodo di c5 è chiamato da un metodo c7 che a sua volta è chiamato da un metodo di c2 che a sua volta è chiamato da un metodo di c3, se io vado a vedere com'è fatto lo stack, inizialmente ho un metodo di c3 che è verde poi ho un invocazione di metodo di c2 che è verde, poi un metodo di c7 arancione e infine chiamo un metodo di c5 che è rosso, quindi lo stack cresce verso l'alto in questo nostro esempio. Non faccio vedere com'è fatto il metodo non faccio vedere come è fatta la struttura del record di attivazione annoto soltanto il dominio di protezione associato alla classe e questo è il motivo per cui viene messo il nome della classe perché è associato alla classe.



RUN TIME STACK

Suppose thread T tries to access a resource



Basic algorithm:

access is allowed iff

all components on the call stack have
the right to access the resource

ie

- rights of a thread is the *intersection* of rights of all outstanding method calls

allora a questo punto com'è fatto l'algoritmo? Supponiamo che il thread T a partire dall'ultimo metodo che è quello associato alla classe 5 cerchi di accedere a una determinata risorsa critica, quindi vuol dire che deve andare a vedere se quei permessi per accedere alla risorsa sono stati garantiti da quella particolare istanza dello stack. Quindi l'accesso è permesso se tutti i componenti sullo stack, quindi, le classi c5 c7 c2 e c3 hanno il diritto di accedere, quindi qual è il dominio di protezione del metodo attualmente in esecuzione, perché poi in realtà non è tanto la classe perché dipende dalla sequenza di invocazione di metodi, è dato dall' intersezione di tutti i diritti d'accesso dei metodi sullo stack questo è esattamente lo stack inspection.

STACK INSPECTION ALGORITHM

```
checkPermission(T) {  
    // loop newest to oldest stack frame  
    foreach stackFrame {  
        if (local policy forbids access to T by class executing in  
            stack frame) throw ForbiddenException;  
  
        if (stackFrame has enabled privilege for T)  
            return; // allow access  
  
        if (stackFrame has disabled privilege for T)  
            throw ForbiddenException;
```

questo è un modo di scriverlo un po' ambiguo non è un vero e proprio algoritmo, ma da l'idea intuitiva di quello che succede devo andare a vedere il permesso di T allora quello che faccio vado a vedere tutti i record di attivazione che ci sono sullo stack e per ogni record di attivazione quello che faccio è controllare se il dominio di protezione associato al metodo su quel particolare frame non mi garantisce il permesso, allora sollevo un'eccezione, se ha diritto di accesso quel particolare frame sto mettendo un flag e devono essere tutti questi flag messi a true me ne basta una che non sia true per fallire, se invece ha disabilitato i privilegi per T allora a questo punto, anche in questo caso tolgo l'esecuzione, poi capiremo che cosa vuol dire disabilitare privilegi, però sostanzialmente questo è un modo di dire che per avere successo, faccio un controllo su tutti i metodi dello stack e devono avere il permesso per accedere alla risorsa.

Discussion

The Basic algorithm is too restrictive in some cases.

Example

One may give an app the right to phone certain phone numbers

l'algoritmo è ragionevole perché mi dice io vado a vedere il diritto di accesso di una entità e lo vado a vedere in base alla storia delle chiamate. Vuol dire un meccanismo di diritti di accesso statici, ma dipende dal flusso di esecuzione, quindi dalla sequenza di attivazione dei metodi. Nella letteratura si chiama history dependance access control è una cosa sana ed è una cosa che permette di caratterizzare bene alcuni aspetti della programmazione. Ha ovviamente un overhead, non è tanto il caricamento delle classi, quello è banale, cioè il class loader quando va a caricare una classe deve fare una valanga di cose per cui se anche va a ispezionare il file .class e metto quali sono i suoi diritti in una opportuna parte della jvm è una array che non si tiene nemmeno conto, il vero overhead è l'esecuzione a tempo dell'ispection dello stack ma quello però è il prezzo da pagare per avere il controllo degli accessi, però è restrittivo, nel senso che uno potrebbe voler scrivere delle applicazioni, ad esempio dell'applicazione che stanno sul telefonino dove dal telefonino io voglio dare accesso per una certa app non ad esempio a tutta l'agenda del dell' utente ma soltanto a degli opportuni telefoni che l'utente ha indicato di avere dato l'accesso. Quindi se io questa operazione la facessi ad un livello di granularità un po più fine, non riuscirei a scrivere delle belle applicazioni che, tipicamente nel caso della mobilità e della telefonia mobile sono significative.

vuol dire che uno non solo deve programmare la politica degli accessi, cioè deve definire a livello di linguaggio di programmazione, quali sono i permessi e che caratteristiche hanno i permessi, ma creare una politica ancora a livello più fino dove il programmatore è responsabile in prima persona dell'accesso alle risorse, in modo particolare il modello di esecuzione di java mette a disposizione di chi programma e ancora una volta questo cosa vuol dire? vuol dire che quelli sono dei particolari metodi e classi che sono accessibili a livello del programma mette a disposizione due primitive, una che si chiama `Enable_permission(P)` e l'altra si chiama `Disable_permission(P)` qual è l'idea? l'idea che è in `Enable_permission` questa proprietà perché il flusso di controllo del programma è tale che io come programmatore certifico che tutto quello che sta succedendo è sotto la mia responsabilità, quindi do il permesso di eseguire questa operazione, quindi questo cosa vuol dire?

Programming stack inspection

Enable_permission(P)

means: don't check my callers for this permission, I take full responsibility

This is essential to allow controlled access to resources for less trusted code

Disable_permission(P)

means: don't grant me this permission, I don't need it

vuol dire che io do la possibilità a codice che non necessariamente è completamente trusted la possibilità di accedere alcune risorse sotto il controllo di chi programma perché abilito il permesso. L'altra primitiva `Disable_permission` invece dice: non darmi questo permesso non la voglio a questo punto voglio dare al programmatore la possibilità di programmare delle applicazioni che soddisfano il principio del minimo privilegio, cioè questo vuol dire: è la mia responsabilità scrivere un'applicazione e permettendogli di programmando i diritti di accesso alle risorse solo che quelli che io che ho scritto l'applicazione ritengo indispensabili per poter operare, gli altri diritti non li voglio e quindi vedete, ho la possibilità quindi di chiudere, di allargare a livello però del programma, quindi lasciando il programmatore la definizione della strategia di accesso.

l'algoritmo diventa più o meno in questo modo: supponiamo di creare un nuovo thread allora a questo punto il nuovo thread viene creato e viene creato a partire dal contesto di esecuzione, quindi eredita tutti i controlli del contesto di esecuzione padre, notate che questa se vi ricordate è esattamente la strategia utilizzata nella creazione dei tab di chrome a partire da un altro tab, quindi c'è un meccanismo di ereditarietà, non nel senso di gerarchia di classi, che dipende da come vengono creati i thread, a questo punto abbiamo il solito la guardia, la permission P e come è fatto l'algoritmo?

THE REVISED ALGORITHM

On creating new thread:

new thread inherit access control context of creating thread

DemandPermission(P) algorithm:

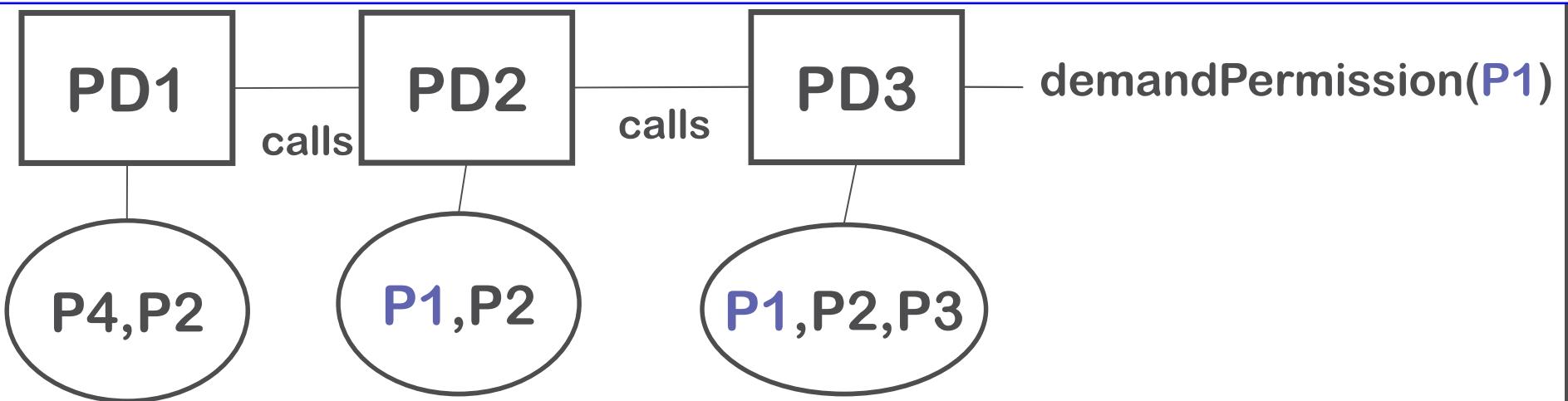
1. **for each caller on the stack, from top to bottom:**
if the caller
 - a) **lacks Permission P:** **throw exception**
 - b) **has disabled Permission P:** **throw exception**
 - c) **has enabled Permission P:** **return**
2. **check inherited access control context**

anche in questo caso l'algoritmo va a vedere sullo stack, allora se manca il permesso funziona esattamente come prima, il solito discorso, questo algoritmo che facciamo vedere è ambiguo nel senso che vado a vedere tutti e faccio la AND di tutte le operazioni che stanno sullo stack, allora appena ne trovo uno che gli manca il permesso finisco, se a questo punto il programmatore ha disabilitato il permesso lancio un exception perché sta cercando di accedere a una risorsa con un permesso che il programmatore ha disabilitato quindi cosa vuol dire? vuol dire che il minimo insieme di operazioni che mi serviva era quello che aveva definito il programmatore io sto cercando di averne uno in più, allora lancio un eccezione. A questo punto, se è abilitata l'accesso alla risorsa per tutti gli stack, do l'accesso, adesso ho finito di guardare lo stack, però non ho ancora concluso, devo andare a vedere se in caso il permesso è ereditato e a questo punto devo andare a vedere il contesto.



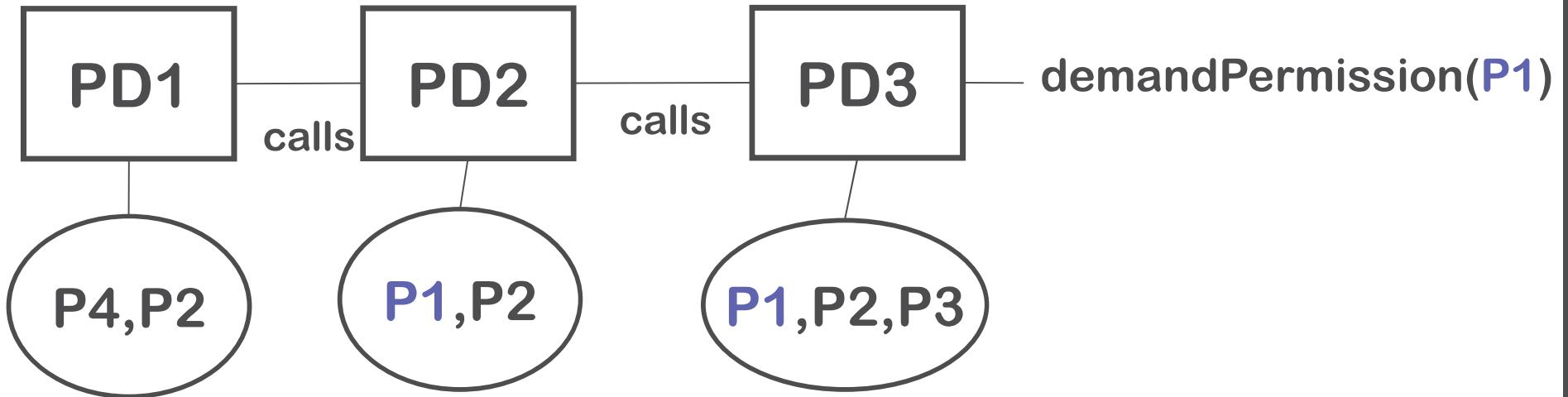
EXAMPLES

supponiamo di avere il record di attivazione del metodo PD1 che chiama il metodo PD2 che a sua volta chiama il metodo PD3 e a questo punto, il metodo PD3 all'interno del suo codice ha la guardia che mi fa la demand permission del permesso P1 e quindi va a vedere se effettivamente se il permesso PD1 è permesso e se io vado a vedere qual è il protection domain associato ai vari processi, sui vari record di attivazione sulla stack vedo che nel caso del metodo PD3 appartiene a una classe che ha un protection domain che contiene P1,P2,P3 lo stack del record di attivazione del metodo PD2 invece, appartiene a una classe con protection domain P1 e P2 mentre quello del metodo PD1 ha P4 e P2.



Will DemandPermission(P1) succeed ?

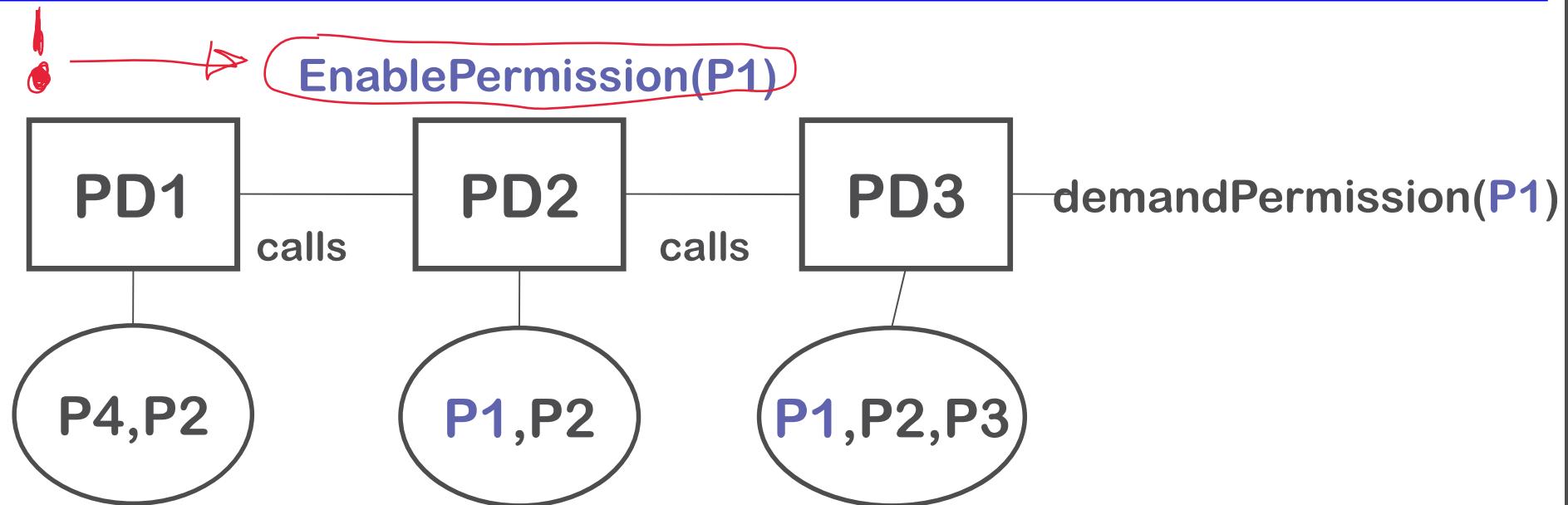
Ha successo la richiesta del PD1? Se vado a controllare tutto lo stack dei record di attivazione, vado a fare un intersezione di tutti i permessi, scopro che P1 non appartiene ai permessi e quindi cosa vuol dire? vuol dire che in questa sequenza di esecuzione non ho il diritto di accedere con P1.



Will DemandPermission(P1) succeed ?

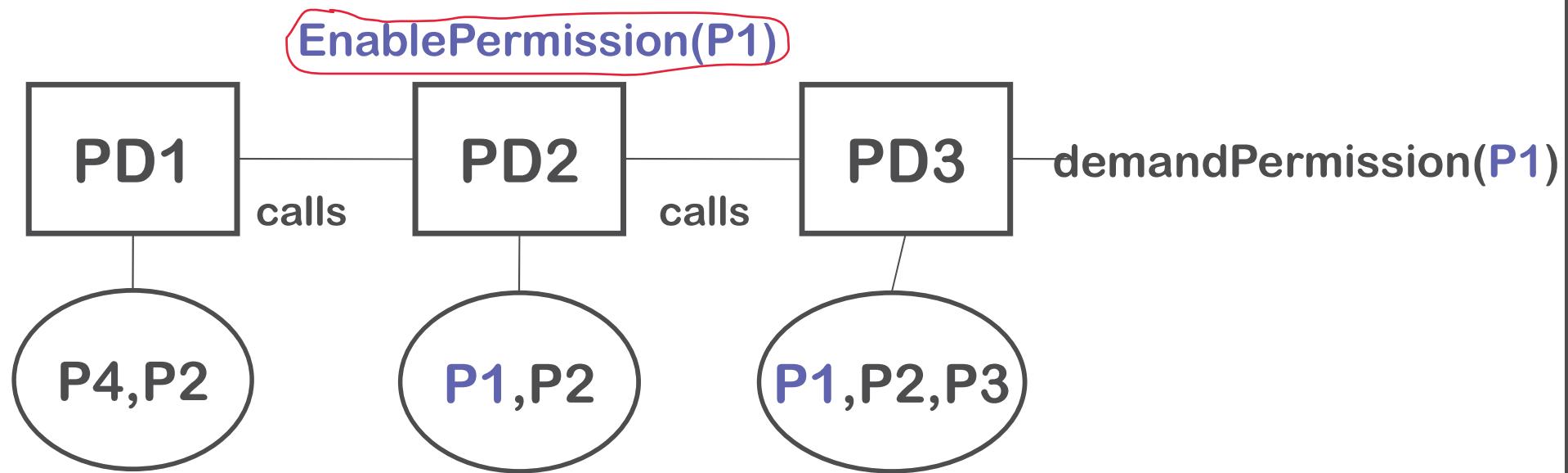
DemandPermission(P1) fails because PD1 does not have Permission P1

adesso immaginiamo di essere nella stessa situazione, ma supponiamo che il metodo nella seconda posizione dello stack PD2 ha eseguito l'enable permission di P1, allora la domanda è la solita, è garantita la P1? Notare che PD1 continua a non avere P1.



Will DemandPermission(P1) succeed ?

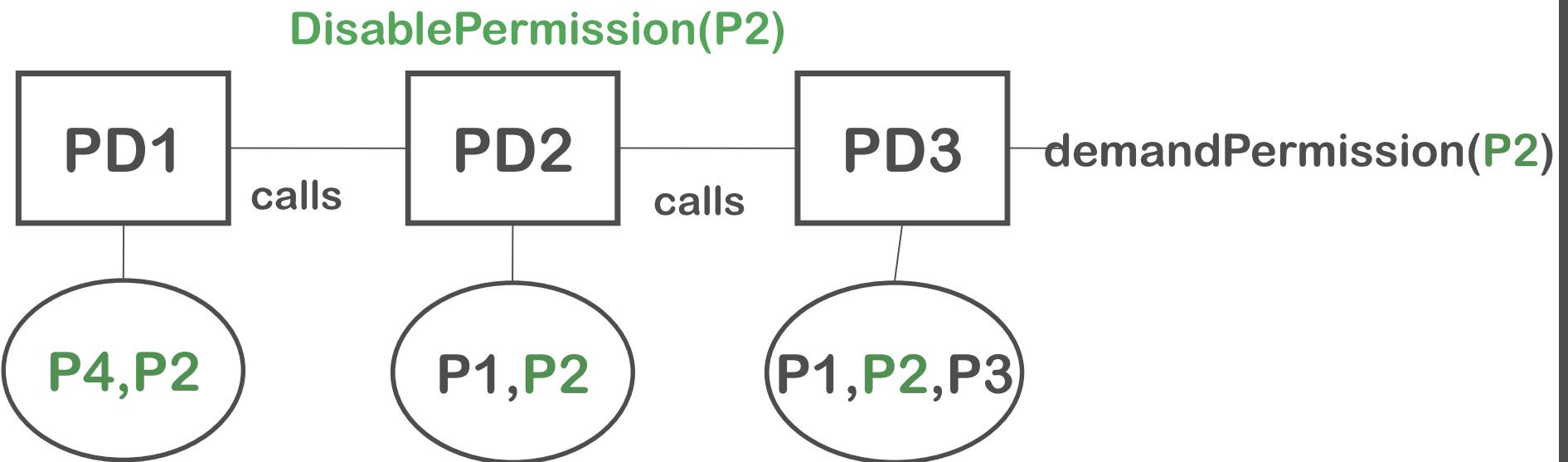
La risposta è sì, perché a questo punto, è successa la stack inspection che blocca perché sono arrivato ad un certo punto che ho eseguito quello che nel modello di esecuzione di java si chiama un'operazione privilegiata e stavo garantendo che da questo punto in poi la gestione della proprietà di accesso del permesso P1 è sotto la mia responsabilità e sto facendo un grant di questo permesso in modo dinamico, quindi questo vuol dire?



Will DemandPermission(P1) succeed ?

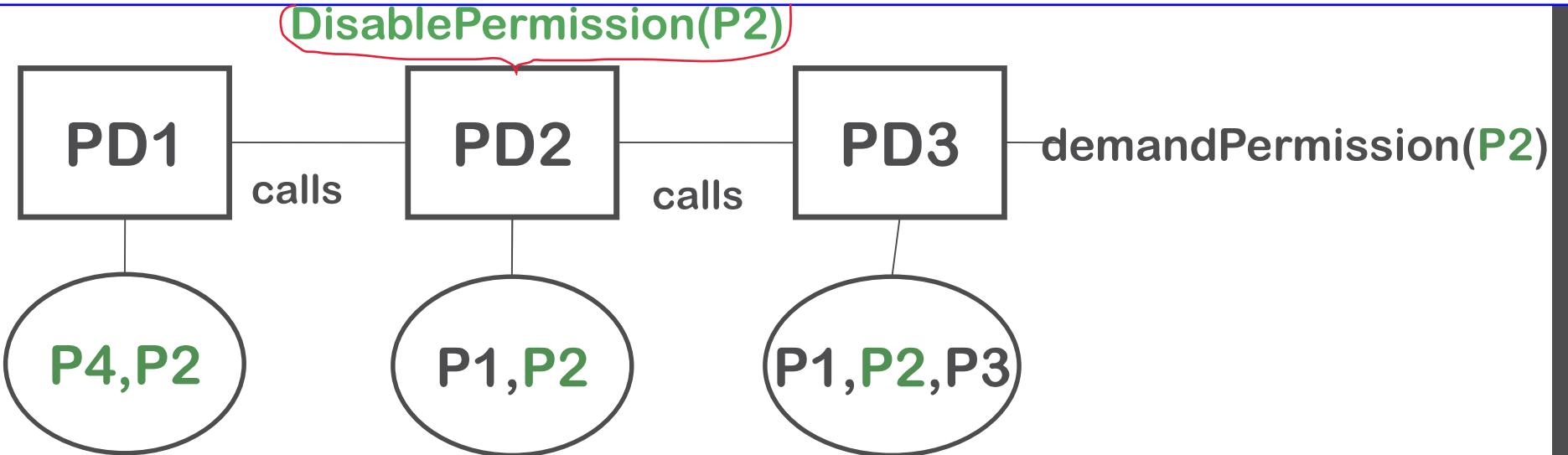
DemandPermission(P1) succeeds

vuol dire che mi prendo la responsabilità di fare eseguire le operazioni con la proprietà P1. Infatti la stack inspection ha successo proprio per il fatto che il metodo PD2 ha eseguito un'operazione privilegiata.



Will DemandPermission(P2) succeed ?

a questo punto dobbiamo far vedere il caso delle operazioni privilegiate che disabilitano il permesso P2 notate che il permesso P2 sarebbe stato un permesso che appartiene all'intersezione dei domini dei tre metodi, quindi se vado a vedere l'intersezione dei tre domini ho esattamente P2 nell'intersezione quindi se io avessi eseguito la stack inspection con l'algoritmo standard senza aver eseguito la parte del secondo metodo PD2 e l'operazione di disable permission, a questo punto avrei avuto l'accesso mentre con l'esecuzione dell'operazione privilegiata di disable P2 sto dicendo che non voglio più P2 e quindi vuol dire che fallisce perché ho esattamente un punto dove non permetto più di avere il permesso P2 quindi vedete un meccanismo molto fine di controllo degli accessi.



Will DemandPermission(P2) succeed ?

DemandPermission(P2) fails

allora a questo punto andiamo a rivedere come scriviamo il pattern di esecuzione e utilizzando esattamente le primitive che mi permette di utilizzare java: vi ricordate l'esempio che avevamo fatto prima del metodo pubblico che permetteva di cancellare il file f?

```
Class Trusted{  
    public void unsafeMethod(File f) {  
        delete f; }  
  
    public void safeMethod(File f) {  
        ... // lots of checks on f;  
        enablePermission (FileDeletionPermission);  
        delete f; }  
  
    public void anotherSafeMethod() {  
        enablePermission (FileDeletionPermission);  
        delete "/tmp/bla"; }  
}
```

"I take full responsibility for my callers"

avevamo un mucchio di controlli su f questo perché al solito siamo in un contesto di programmazione difensiva a questo punto quello che volevamo fare per potere operare con stack inspection e poter operare con tutti i meccanismi di programmazione si abilitava il permesso di cancellazione del file e cosa voleva dire? voleva dire che chi programma, si sta prendendo la responsabilità per questa operazione e per tutti quelli che mi invocano, quindi questo vuol dire che bisogna tenere particolarmente conto del flusso di esecuzione perché il flusso di esecuzione dipende dall'invocazione dei metodi.

allora a questo punto il tipico pattern di programmazione all'interno di java quando si sta operando in una situazione dove ho un metodo pubblico che può essere potenzialmente accessibile da un metodo potenzialmente untrusted e quindi che viene ad esempio dalla rete, quello che succede è che faccio un po di controlli per vedere quali sono le proprietà degli argomenti A e B qualunque cosa sia, abilito i privilegi, in modo particolare eseguo l'operazione speciale di abilitare i privilegi,

The typical programming pattern in privileged components, esp. in public methods accessible by untrusted code:

```
public methodExposingScaryFunctionality (A a, B b) {  
    ....; do security checks on arguments a and b  
    enable privileges (P1,P2);  
    do the dangerous stuff that needs these privileges;  
    disable privileges;  
    .... }
```

in keeping with the principle of least privilege

in modo particolare i privilegi P1 e P2 ,faccio le operazioni in modo controllato con questi privilegi e poi li disabilo nuovamente, quindi a questo punto sto cercando di encapsulare la possibilità di interagire con codice untrusted tenendo conto di informazioni che ho dalla rete, ad esempio se ho un entità che già conosco e cercando di avere il minimo insieme di privilegi che mi permettono di eseguire l'operazione. Quindi sto cercando di avere a livello del programma politiche di controllo degli accessi che rispettano il principio del minimo privilegio.

Spot the security flaw?

```
Class Good{

    public void m1 (String filename) {
        lot of checks on filename;
        enablePermission (FileDeletionPermission);
        delete filename; }

    public void m2 (byte[] filename) {
        lot of checks on filename;
        enablePermission (FileDeletionPermission);
        delete filename; }

}
```

abbiamo questa classe Good abbiamo il metodo pubblico m1 che prende il nome di un file, ho un po' di controlli sul file name, abilito i permessi perché a questo punto, alla fine dei controlli, abilito i permessi, do il permesso di modificare il file e faccio il delete del file. Il metodo m2 fa la stessa cosa e l'unica differenza del metodo m2 è che prende il nome del file come una sequenza di byte quindi prende la rappresentazione a byte della stringa che definisce il nome del file. Qual è il problema di sicurezza in questo codice? Allora questo qui è un codice che vuol dare la possibilità di cancellare un file da parte di un'entità esterna che da il nome del file da cancellare. Quindi noi gli diamo il minimo insieme di privilegi per operare.

Qual è la differenza tra stringa e byte? La stringa è immutable. Immaginiamo che chi chiama il metodo m1 è un entità trusted potenzialmente, quindi gli sta dicendo :il nome del file è esattamente il file che io voglio cancellare allora abilita i permessi e lo cancello. La differenza tra usare il primo metodo e usare il secondo metodo è che le stringhe in java sono immutable, quindi non posso avere un attacco dove ho un'operazione che mi va a modificare in modo tra virgolette malevolo il nome del file invece. Invece se io lo do come un array di caratteri, gli array sono modificabili, quindi potremmo modificare il nome del file.

TOCTOU attack (Time of Check, Time of Use)

```
Class Good{  
    public void m1 (String filename) {  
        lot of checks on filename;  
        enablePermission (FileDeletionPermission);  
        delete filename; }  
  
    public void m2( byte[] filename) {  
        lot of checks on filename;  
        enablePermission (FileDeletionPermission);  
        delete filename; }  
}
```

m1 is **secure**, because
Strings are immutable
(assuming there are no TOCTOU
vulnerabilities in the underlying file
systems, eg due to symbolic links)

m2 is **insecure**,
because byte arrays
are **mutable**;
attackers can could
change the value of
filename after the
checks, in a multi-
threaded setting

Il punto è che il metodo m1 è sicuro perché usa una struttura dati che è immutable e questi tipi di attacco si chiamano "time of check, time of use" cioè nel senso che è a questo punto un attacco di questo tipo non è più possibile perché non posso andare a modificare la stringa, mentre il secondo metodo m2 è insicuro perché gli array sono mutable allora l'attaccante potrebbe fare un'operazione del cambio del nome del file dopo che è stato fatto il controllo e per questo si dice che il time of check è diverso dal time of use perché siamo in un contesto multi thread e quindi il thread dell'attaccante potrebbe fare questa operazione di modifica dell'array dei byte dopo che è stato fatto il controllo sul nome del file e che per questo li c'era scritto " a lot of checks of". Quindi questo è un pattern che vi fa capire come pure in un contesto di programmazione con politiche di sicurezza bisogna fare attenzione moltissimo alle strutture siano esse mutable o no.

Discussion

Privilege permissions: useful abstraction to program flexible access control policies.

Similarity between methods which temporarily raise privileges

- **Linux setuid root programs (Windows Local System Services) which can be started by any user, but then run in admin mode**
- **OS system calls invoked from a user program which cause a switch from user to kernel model**

Software Design tip

- All software entities (services) may be programmed to elevate the privileges of their clients
 - hopefully in a secure way...
 - If not: **privilege escalation attacks**
- **In any code review, such code obviously requires extra attention!**

Noteate che c'è anche una similarità tra il fatto di avere operazioni privilegiate quindi metodi che attivano certe operazioni e dei meccanismi che sono presenti nei sistemi operativi che aumentano il livello di privilegio un entità. Ad esempio il setuid del programma di root in linux può essere attivato da quello utente che poi va in posizione amministratore. In generale, nei sistemi operativi voi avete delle primitive che permettono a un generico utente di diventare amministratore per una particolare istanza, quindi è un qualcosa per cui le operazioni privilegiate hanno un senso anche a livello di sistema allora e un tip di software design? Posso programmarmi il diritto di aumentare i privilegi, ovviamente lo devo fare in un modo controllato perché se non lo faccio in modo controllato ho quella che si chiama un attacco di escalation dei privilegi, ciò vuol dire che sto dando dei privilegi e seguendo la sequenza delle chiamate nel metodo poi in realtà sto dando all'attaccante una valanga di diritti. Nell'ingegneria del software ci sono dei meccanismi di testing e dei meccanismi di code review che fanno particolarmente attenzione a questo aspetto, in modo tale che ci sia un controllo su come il codice è scritto per garantire che non avvenga un escalation dei privilegi.



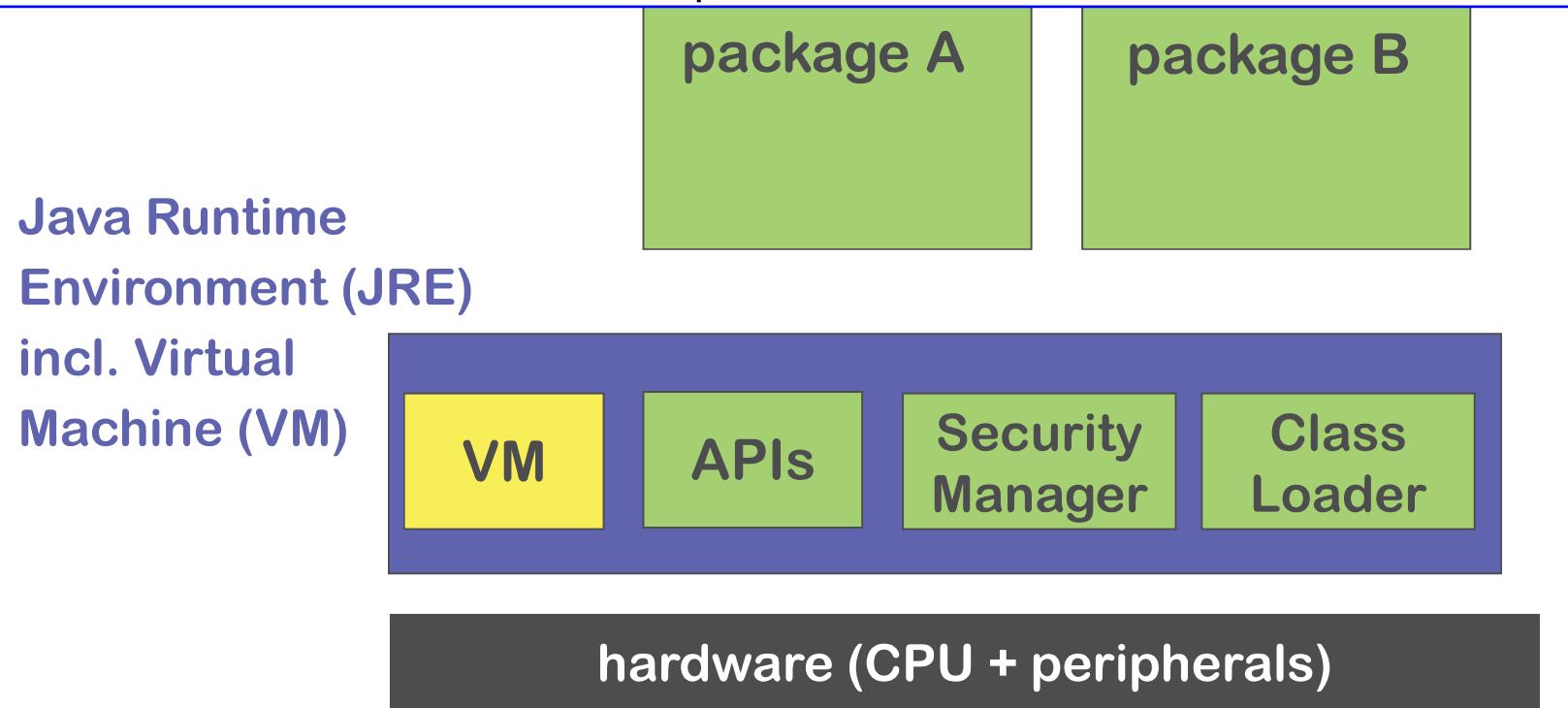
Summary: Java Security Guarantees

1. **memory safety**
2. **strong typing**
3. **visibility restrictions**
4. **immutable fields**
5. **unextendable classes**
6. **immutable objects, eg String, Boolean, Integer, URL**
7. **sandboxing based on stackwalking**

This allows security guarantees to be made even if part of the code is untrusted – or simply buggy

Similar guarantees for Microsoft .NET/C#, for Scala, ...

A questo punto noi possiamo dare un'idea ulteriormente raffinata, di che cosa vuol dire run time di ogni linguaggio di programmazione con politiche di sicurezza e di un linguaggio di programmazione che ha astrazione di sicurezza incluse. Qui ci sono diversi package, i diversi package sono eseguiti dal run time che contiene la VM, le API, ovviamente fanno parte del runtime, ma a questo punto abbiamo il security manager che è esattamente quello che è responsabile della politica di controllo degli accessi e similmente abbiamo il class loader, che ha il ruolo non solo di caricare le classi, ma a questo punto, una volta che carica del classi ha il compito di definire e di consultare il class file e di memorizzare la politica di sicurezza.



TCB for Java code-based access control

- **Byte Code Verifier** (BCV) typechecks the byte code
- **Java Virtual Machine** (JVM) executes the byte code (with some type-checking at run time)
- **SecurityManager** (SM) does the runtime access control by stack inspection
- **ClassLoader** (CL) downloads additional code, invoking BCV & updating policies for the SecurityManager

qual è la trust computer base di java così esteso? allora sicuramente abbiamo il byte code verifier, che è il componente della JVM che una volta che riceve il file byte coded lo puo ricevere sia come risultato della compilazione o dalla rete, indipendentemente da dove lo riceve fa un controllo, va a vedere se è conforme alle caratteristiche del byte code generato e certificato. La JAVA virtual machine esegue il byte code e ha dei controlli di sicurezza per garantire la type safety e la memory safety e abbiamo detto che ovviamente queste sono essenziali per fare il controllo degli accessi. Il security manager è quello che ha il compito di fare la stack inspection, quindi ovviamente deve essere trusted e infine il class loader che ha il compito di invocare il bytecode verifier e di andare a ispezionare il codice punto class per fare il setting del protection domain. Ovvero di mettere tutte quelle informazioni che poi saranno ispezionate dal security manager per fare il controllo degli accessi.

Implementation of the class `Class` in JDK1.1.1

```
package java.lang;

public class Class {

    private String[] signers;

    /** Obtain list of signers of given class */

    public String[] getSigners()

        { return signers;    }
```

What is the bug ?

How can it be fixed ?

Could it be prevented at language-level ?

Implementation of the class Class in JDK1.1.1

```
package java.lang;

public class Class {

    private String[] signers;

    /** Obtain list of signers of given class */

    public String[] getSigners()

    {
        return signers;
    }
}
```

What is the bug ? getSigners leaks reference to internal data structure

How can it be fixed ? getSigners should clone the array and return a clone

Could it be prevented at language-level ? By having immutable arrays, or type system for alias control

Implementation of the class Class in JDK1.1.1

```
package java.lang;

public class Class {

    private String[] signers;

    /*
```

```
pu
```

WE WILL DISCUSS THIS
EXAMPLE LATER
IN RUST

What is it?

Structure

How can we implement it?

a clone

Could it be prevented at language level? By having immutables arrays, or type

conversion

l'altro meccanismo è avere una gestione degli alias, cioè fare in modo che ci sia un meccanismo di protezione e quindi avere un unico proprietario, allora se ho un meccanismo di gestione del alias ho un ulteriore aspetto di safety all'interno del linguaggio che mi permette di garantire ancora più un numero maggiore di proprietà di sicurezza. La vedremo quando andremo a esaminare esattamente come rust affronta questo aspetto per avere ancora degli aspetti ulteriori di protezione a livello del linguaggio.

Discussion: nice ideas and techniques but

Some Java features are still the root of security problems:

- **Large TCB with large & complex attack surface**, growing over time
 - Many classes in the core Java API are in the TCB and can be accessed by malicious code
 - Security-critical components are implemented in Java & runs on the same VM, incl. ClassLoader and SecurityManager

Discussion

- Apart from logical flaws, there are risks of trusted code accidentally exposing a field as protected or sharing a reference to mutable object with untrusted code
- The possibility to download code over the internet is a dangerous capability, even if it is protected & controlled
 - it makes security flaws easy to exploit & devastating

What about unsafe languages?

abbiamo visto e hanno visto una l'idea di avere dei compartimenti che permettano ad encapsulare del codice, che sia untrusted e che possa interagire con del codice trusted abbiamo cercato di far vedere come questa caratteristica possa essere vista all'interno di un linguaggio di programmazione, andando a esaminare quali sono le caratteristiche del linguaggio di programmazione che permetta non solo di inserire a livello del linguaggio delle primitive che ci permettono di fare l'analisi delle proprietà che una certa porzione di codice possa avere e nel caso particolare, quali sono i permessi che vengono dati a delle componenti di codice, quelle che abbiamo chiamato usando la terminologia di Java le codebase e poi quali sono i meccanismi non solo per descrivere al livello di programma il controllo degli accessi, ma quali sono i meccanismi che devono essere presenti nel run time dell'esecuzione del programma, quindi nella macchina virtuale dell'esecuzione del linguaggio, i meccanismi che servono poi per fare in modo che venga valutato questa caratteristica. Abbiamo esaminato alcuni aspetti specifici di questa implementazione, di questa idea, nel caso del Java come linguaggio di programmazione abbiamo visto quindi il meccanismo della stacking inspection che permette di andare a controllare che il diritto di una chiamata dipende non solo dai diritti che io do al metodo attualmente in esecuzione, ma dal flusso di esecuzione. Perché l'idea che ci sta dietro, è che se io cerco di avere all'interno della stessa applicazione parti di codice che provengono da sorgenti con diritti diversi, devo andare a tracciare a runtime com'è il flusso delle invocazioni, in modo tale da scoprire da chi è stata attivata quella chiamata. Poi abbiamo visto quali sono i vincoli che ci permettono di programmare ulteriormente la politica di accesso e quindi abbiamo visto le operazioni privilegiate e a questo punto la domanda che uno si pone è: ma cosa succede per i linguaggi che non sono che non sono safe ? Abbiamo visto che la premessa di tutto questo è che il linguaggio deve essere type safety e memory safety e abbiamo fatto vedere la volta scorsa gli esempi e abbiamo portato le motivazioni al perché di questa osservazione.

Sandboxing

- Unsafe languages cannot provide sandboxing at the language level ... but
- An application written in an unsafe language could still use OS sandboxing by splitting the code across different processes (this is what Chrome does)
- An alternative approach:
use sandboxing support provided by underlying hardware,
to impose memory access restrictions inside a process

A questo punto si può affrontare l' approccio in due modi diversi: il primo approccio l'abbiamo già brevemente illustrato, con le caratteristiche di Google Chrome abbiamo fatto vedere come esempio un'architettura software di chrome come ogni tab che viene aperto nel browser e che a che vedere con il rendering di tutte le caratteristiche di interattività del browser, ogni tab corrisponde a un processo isolato quindi che cosa vuol dire? vuol dire che l'idea che io ho un applicazione e ne compongo il codice attraverso diversi sistemi che sono isolati uno dall'altro che hanno dei meccanismi standard di comunicazione e che poi si poggiano su una componente trusted, nel caso di chrome e del browser era il kernel del browser che si occupava ad esempio della gestione dello stack TCP/IP e dello stack del trasporto sicuro. un altro approccio, una soluzione differente che permette di affrontare lo stesso problema, cioè di isolare delle componenti, è quello di avere delle primitive hardware che ti aiutano nel fare questa operazione, cioè delle primitive hardware che permettono a livello della macchina hardware di dire: questa parte di codice non può essere accessibile se non da altre componenti che io ho definito

Example: security-sensitive code in larger program

secret.c

```
static int tries_left = 3;
static int PIN = 1234;
static int secret = 666;

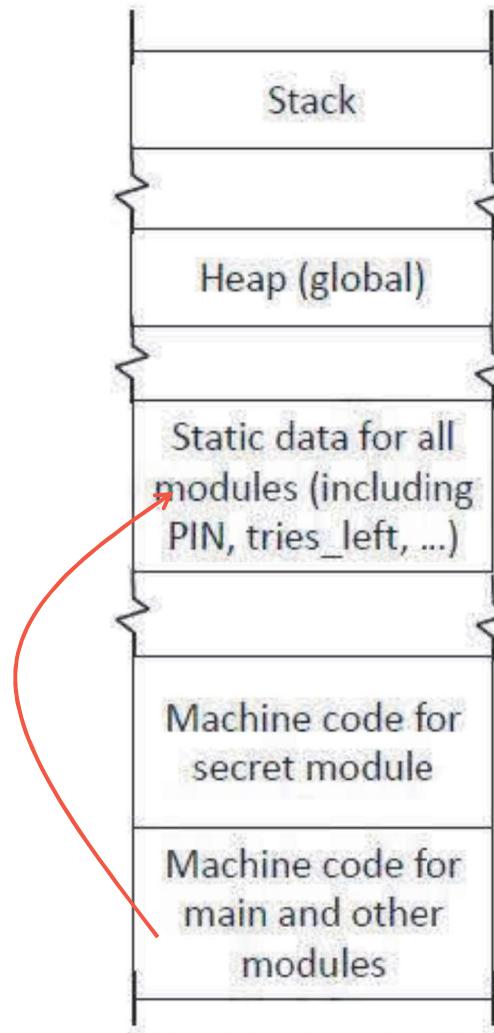
int get_secret (int pin_guess) {
    if (tries_left > 0) {
        if (PIN == pin_guess) {
            tries_left = 3; return secret; }
        else {
            tries_left--; return 0 ;}
    } }
```



main.c

```
# include "secret.h"
... // other modules
void main () {
...
}
```

Bugs or
malicious code
anywhere in the
program could
access the
high-security data



Example from [N. van Ginkel et al, Towards Safe Enclaves, HotSpot 2016]

Isolating security-sensitive code with secure enclaves

secret.c

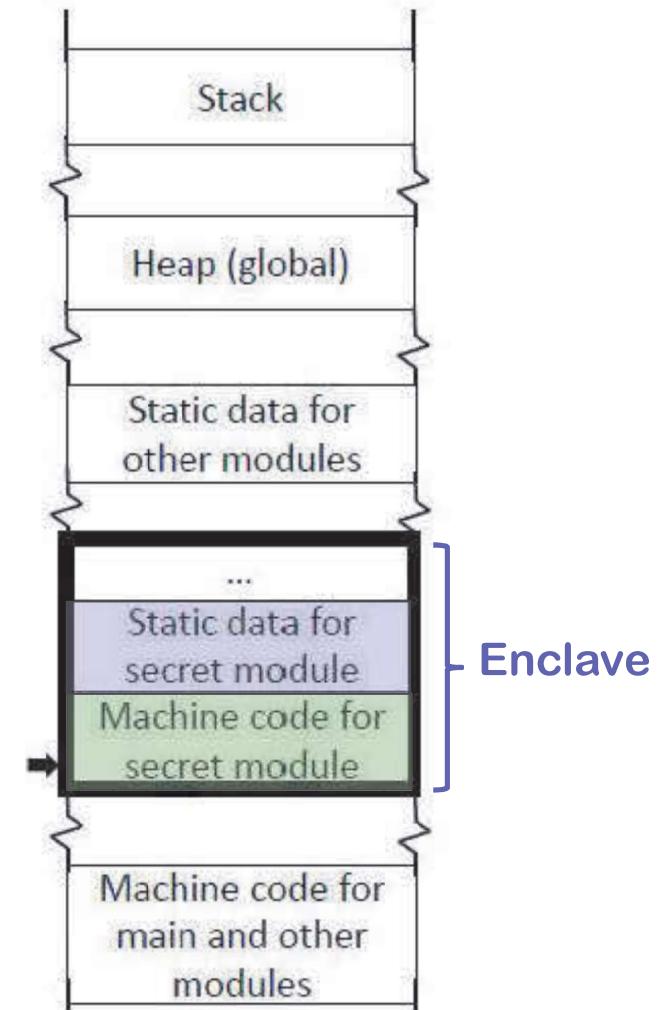
```
static int tries_left = 3;
static int PIN = 1234;
static int secret = 666;

int get_secret (int pin_guess) {
    if (tries_left > 0) {
        if ( PIN == pin_guess) {
            tries_left = 3; return secret; }
        else {
            tries_left--; return 0 ;}
    }
}
```



main.c

```
# include "secret.h"
... // other modules
void main () {
...
}
```



Isolating security-sensitive code with secure enclaves

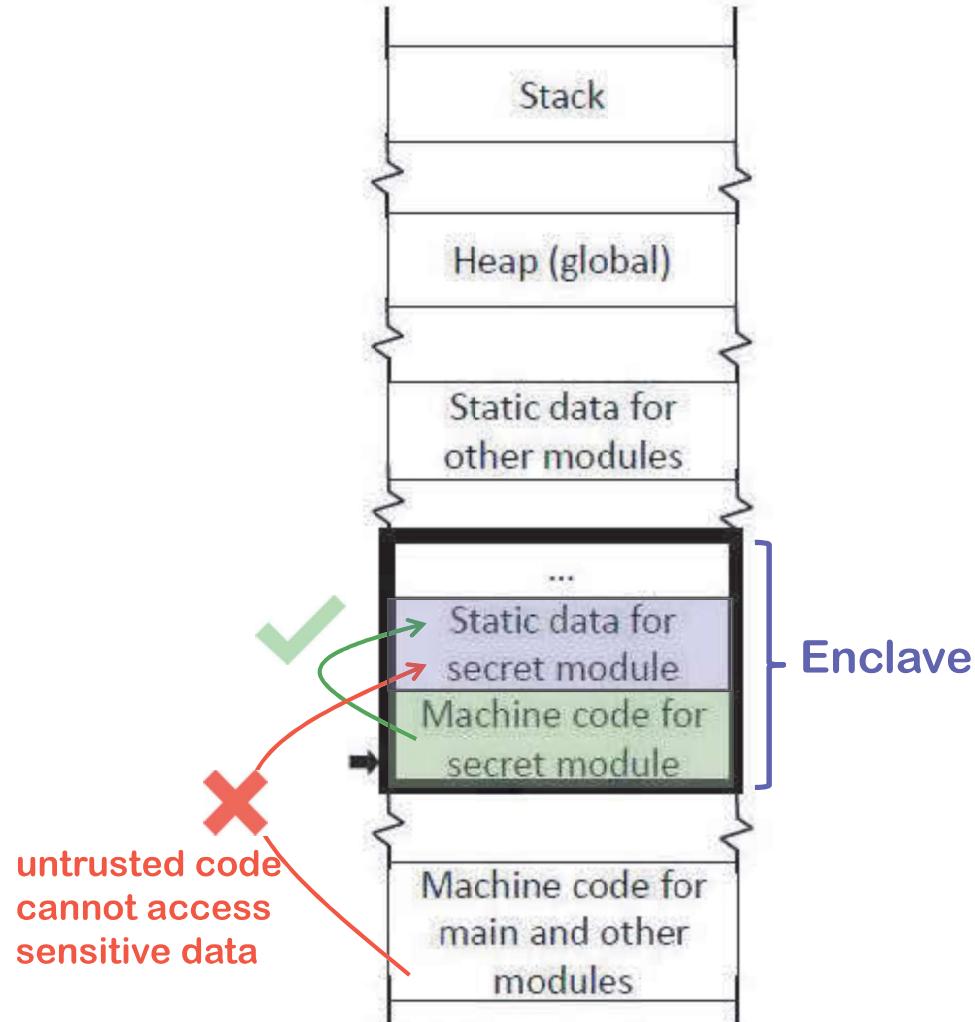
secret.c

```
static int tries_left = 3;
static int PIN = 1234;
static int secret = 666;

int get_secret (int pin_guess) {
    if (tries_left > 0) {
        if ( PIN == pin_guess) {
            tries_left = 3; return secret; }
        else {
            tries_left--; return 0 ;}
    }
}
```

main.c

```
# include "secret.h"
... // other modules
void main () {
    ...
}
```



Isolating security-sensitive code with secure enclaves

secret.c

```
static int tries_left = 3;
static int PIN = 1234;
static int secret = 666;

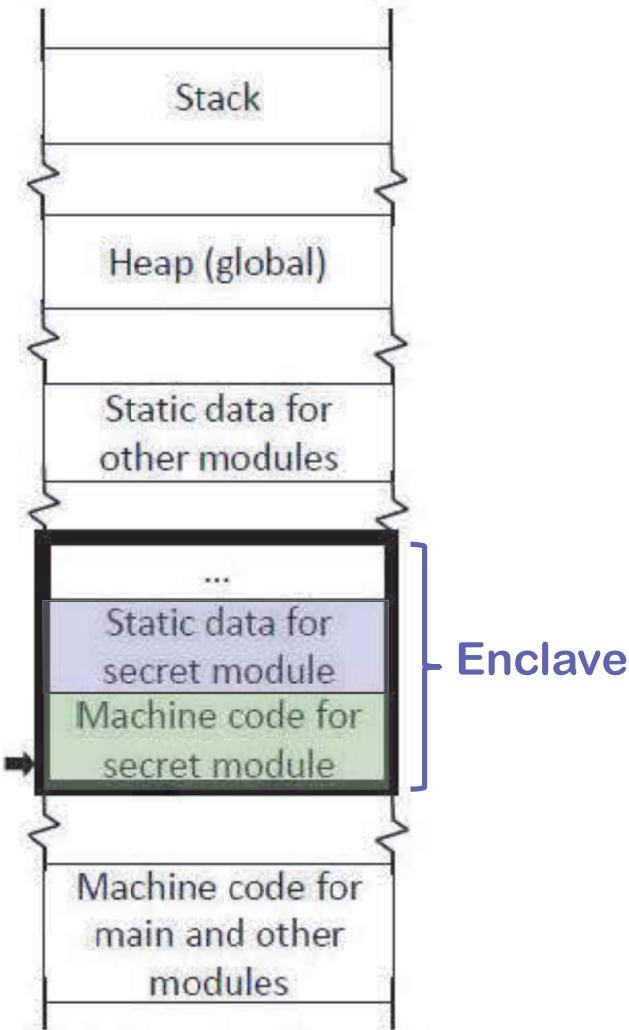
int get_secret (int pin_guess) {
    if (tries_left > 0) {
        if ( PIN == pin_guess) {
            tries_left = 3; return secret; }
        else {
            tries_left--; return 0 ;}
    }
}
```

main.c

```
# include "secret.h"
... // other modules
void main () {
...
}
```

Only allowed entry point
(for `get_secret`)

Untrusted code should not be
able to jump to the middle of
`get_secret` code (recall return-to-
libc & ROP attacks)



Il codice che è fuori dall'enclave non può accedere i dati all'interno delle enclave, il codice che è fuori dall'enclave può accedere soltanto a dei determinati punti d'accesso dell'enclave e non può avere degli accessi più liberali. Andiamo a esaminare le caratteristiche di questo: allora sicuramente è meno flessibile dello stack inspection perché? perché non traccia il flusso di esecuzione delle invocazioni, cioè quello che dice è: tu puoi accedere o non puoi accedere se sei al di fuori, quindi se fai una chiamata da un codice untrusted e poi chiavi il codice trusted non fai tutta la tracciatura di quello che avviene seguendo il flusso delle invocazioni dai metodi delle funzioni, quindi questo sicuramente è una parte che mettiamo sul lato negativo.

Secure Enclave

1. Enclaves isolates part of the code together with its data
 - Code outside the enclave cannot access the enclave's data
 - Code outside the enclave can only jump to valid entry points for code inside the enclave
2. Less flexible than stack walking:
 - Code in the enclave cannot inspect the stack as the basis for security decisions
 - Not such a rich collection of permissions, and programmer cannot define his own permissions
3. More secure, because
 - OS & Java VM (Virtual Machine) are not in the TCB
 - Also some protection against physical attacks is possible

Non ha la possibilità di programmarsi ed avere un meccanismo ad alto livello per programmarsi gli accessi, in modo particolare le operazioni privilegiate che nella nostra visione quando abbiamo descritto le caratteristiche di java della stack inspection avevamo detto che le operazioni privilegiate garantivano il principio di avere il minor numero di permessi, il principio del least privilege, e quindi permettevano di programmarsi la propria politica di controllo degli accessi. Quindi da questo punto di vista è negativo, dall'altro punto di vista, dato che la macchina dell'esecuzione del linguaggio, essendo il linguaggio che stiamo considerando non memory safety e non type safety, quindi fuori dal trust computer base e quindi avere dei meccanismi trusted è utile per fare l'esecuzione, inoltre permette anche di avere dei supporti per attacchi fisici.

allora tutto questo si basa su dei supporti hardware, in modo particolare il sistema dell' intel è quello sgx che fornisce esattamente questa caratteristica, fornisce una nozione di trusted execution environment che è confinato e ha dei vincoli su come le componenti di esecuzione al momento dell'esecuzione di un programma possono o non possono accedere alle parti del programma. Isola il codice non solo dagli altri programmi, ma anche dal sistema operativo, quindi definisce un evidente meccanismo di sicurezza, alcuni colleghi che sono molto più vicini alla comunità dell'open source, hanno un po di obiezioni su questa scelta dell'Intel, in modo particolare sul modello di business, cioè nel senso che soltanto codice che è certificato intel può essere messo all'interno dell'enclave e questo lascia un po' di perplessità.

Secure enclave: hardware support

- Intel SGX provides hardware support for secure enclaves
 - protecting confidentiality & integrity of enclave's code & data
 - providing a form of Trusted Execution Environment (TEE)
- SGX not only protects the enclave from the rest of the program, but also from the underlying Operating System!
- Some concerns about Intel's business model & level of control: will only code signed by Intel be allowed to run in enclaves?

esistono altri meccanismi, esistono anche degli altri brevetti, in modo particolare, c'è un brevetto di google del 2016 che affronta questo problema in un modo leggermente diverso, un po più leggero rispetto all enclave e l'idea è del fatto che ho un accesso a un controllo degli accessi andando a esaminare il valore del program counter quindi cosa ho? una mappa della memoria e so quale componente della memoria possa accedere utilizzando degli opportuni valori dell'indirizzo del programma counter, altri no.

EXECUTION AWARE MEMORY PROTECTION

- A more light-weight approach to get secure enclaves
 - access control based on the value of the program counter, so that some memory region can only be accessed by a specific part of the program code
- This provides similar encapsulation boundary inside a process as SGX
 - crypto keys can be made only accessible from the module with the encryption code
- The possible impact of a buffer overflow attack is then reduced

[Google, US patent 9395993 B2, July 2016]

[Koeberl et al., TrustLite: A security architecture for tiny embedded devices, European Conference on Computer Systems. ACM, 2014]

Vuol dire che sostanzialmente sto divenendo la memoria e le regioni, sto dando alla regione della memoria dei diritti e cerco di evitare andando ad esaminare il valore del program counter che ci possano essere dei canali nascosti tra regioni differenti che fanno emergere degli attacchi. Questo è sicuramente un modo diverso di affrontare un problema, non ha le stesse proprietà del meccanismo dell'intel e della sg un po più leggero, un po più flessibile e ha i suoi pro e ovviamente vi potete anche immaginare i contro, li vedete le informazioni del brevetto di google e l'articolo dove è stato presentato per la prima volta il meccanismo.

why we have
focused our
attention on
secure enclaves
and hardware
supports?

OUR FOCUS IS ON
PROGRAMMING LANGUAGES.



Attacks

- A recent wave of attacks have shown that hardware isolation mechanisms can be attacked via *software-exploitable side-channels*.
 - Over the past few years, many major isolation mechanisms have been successfully broken using side-channels
 - Foreshadow broke confidentiality of enclaved executions on Intel processors.

Jo Van Bulck, et al. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution." 27th USENIX Security Symposium, 2018, pp. 991– 1008.

allora immagino che voi vi domanderete perché uno all'interno di un corso che parla di linguaggio di programmazione ha dedicato un po' di tempo alla parte hardware? Il motivo è perché poi linguaggi di programmazione vengono compilati quindi non solo c'è il progetto del linguaggio, non solo le parti di analisi statica ma c'è anche la parte di generazione del codice e quindi vuol dire che uno deve sapere com'è fatta la macchina sotto per generare il codice, quindi sapere se usa un meccanismo di enclave o usa un meccanismo dove ha un isolamento usando il valore del program counter. La risposta meno ovvio è che ci sono stati recentemente degli attacchi e vedete questo è l'esempio che faccia un attacco di due anni fa, dove ci sono degli attacchi esattamente agli enclave utilizzando dei canali nascosti che possono essere utilizzati a livello software, ovvero sono dei flussi di informazione che possono emergere dall' interazione tra l'enclave che non erano previsti originariamente e che vengono fuori per caratteristiche della struttura, quindi questo cosa vuol dire? vuol dire che anche via software posso fare degli attacchi utilizzando questi canali nascosti di interazione tra le varie enclave a strutture che dovrebbero proteggere i dati.

Language-based Countermeasures

Provably Secure compilation techniques

M. Busi et al, Provably Secure Isolation for Interruptible Enclaved Execution on Small Microprocessors, IEEE 33rd Computer Security Foundations Symposium (CSF) 2020.

Questa cosa ha a che vedere su come si compila quindi tutta una parte di attività che ora ha a che vedere su come già nel software, su architetture che sono trusted, su architetture che forniscono già al livello dell'architettura hardware dei supporti, allora devo generare del codice in modo tale che ci siano dei meccanismi di interazione che permettano di amalgamare le proprietà di alto livello che io sto assumendo nel linguaggio con le proprietà di basso livello che sono fornite dalla macchina hardware, in modo tale che tutta la toolchain della compilazione affronti bene l'aspetto della sicurezza e questo è un esempio, ce ne sono tanti altri di lavori .

Avere compartimenti e isolare componenti processi a livello dello sviluppo del software è una cosa normale in tutti gli esempi di sviluppo software, in modo particolare abbiamo visto i sistemi operativi e con il meccanismo dei sistemi operativi ho l'isolamento con le primitive di controllo degli accessi dei sistemi operativi che permettono di definire il controllo degli accessi ruoli e permessi a livello delle applicazioni e dei processi del sistema operativo. Abbiamo visto un approccio linguistico, in modo particolare quello di Java del meccanismo di controllo degli accessi e abbiamo visto che l'assunzione principale per poter operare con quel meccanismo a livello di linguaggio è che linguaggio sia safe, ovvero almeno memory safe e type safe quindi abbiamo visto la soluzione di java che era poi la soluzione di C# la soluzione che è utilizzata in tantissimi altri linguaggi, ovvero il meccanismo del Sandboxing basato sulla nozione di permessi e di stack inspection.

Recap: different forms of compartmentalisation

- Conventional OS access control }
 - Language-level sandboxing in safe languages
 - eg Java sandboxing using stackwalking
 - Java VM & OS in the TCB
 - Hardware-supported enclaves in unsafe languages
 - eg Intel SGX enclaves
 - underlying OS possibly not in the TCB
- access control
of applications and
between applications
- access control
within an application

Abbiamo visto cosa vuol dire questo a livello della modifica della struttura della java virtual machine e quali sono le relazioni su la trust computer base del linguaggio in cui la java virtual machine del sistema operativo diventa una parte della trusted computed base. Abbiamo visto oggi stamane il discorso di avere dei meccanismi di isolamento, in modo particolare l' enclave sicure nel linguaggio di programmazione che sono unsafe, quindi quel linguaggio che non garantiscono la type safety e la memory safety, in modo particolare l'esempio della sgx dell' intel e l'esempio della partizione della struttura della memoria in regione, con un meccanismo di controllo, andando a esaminare il valore del program counter. Quindi queste due soluzioni, del sandbox al livello del linguaggio e le secure enclave per quanto riguarda il linguaggio safe, garantiscono quindi dei meccanismi di controllo più fini, cioè all'interno della struttura dei programmi, differentemente da quello che succede nei sistemi operativi.

Summary

- **Language-based sandboxing is a way to do access control within a application: different access right for different parts of code**
 - This reduces the TCB for some functionality
 - This may allows us to limit code review to small part of the code
 - This allows us to run code from many sources on the same VM and don't trust all of them equally
- **Hardware-based sandboxing can also achieve this also for unsafe programming languages**
 - Much smaller TCB: OS and VM are no longer in the TCB
 - But less expressive & less flexible
 - No stackwalking or rich set of permissions



Readings

- François Pottier, Christian Skalka, Scott F. Smith:
A systematic approach to static access control. ACM Trans. Program. Lang. Syst. 27(2): 344-382 (2005)
- Cédric Fournet, Andrew D. Gordon:
Stack inspection: Theory and variants. ACM Trans. Program. Lang. Syst. 25(3): 360-399 (2003)
- Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari:
Stack inspection and secure program transformations. Int. J. Inf. Sec. 2(3-4): 187-217 (2004)
- Dan S. Wallach, Edward W. Felten:
Understanding Java Stack Inspection. IEEE Symposium on Security and Privacy 1998: 52-63