Stack inspection

Allowing or disallowing a permission depends on the context in which the checkPermission method was called.

The access controller needs to examine the protection domains associated with all the classes on the stack.

The permission is granted only if all the protection domains on the stack have this permission.

In our old example, the BadClass.main() method for deleting a file calls the Runtime.exec() method which calls the AccessController.checkPermission() to check execute permission on /bin/rm.

Further, the BadClass.main() method itself may be called by some other method m() of class C.

We get the following stack.

AccessController.checkPermission()
Runtime.exec()
BadClass.main()
C.m()
...

The execute permission should be granted only if all the classes on the stack have that permission in their protection domain.

Hence the access controller checks that all frames from the top of the stack to the bottom have this permission in the protection domains of the respective classes.

Sometimes a trusted class may choose to give its permissions to lower frames on the stack.

E.g. an untrusted applet may call some routine to draw something on the screen, and the routine requires some local font file.

This is done using the doPrivileged() method.

```
untrustedclass { f() { ... trustedclass.draw() ...}}
trustedclass {
  public void draw {
    ...
    AccessController.doPrivileged (new PrivilegedAction () {
      public Object run () {
        // privileged code here
        ... <read font file> ...
      } }); }}
```

Instead of the doPrivileged() method

```
AccessController.doPrivileged (new PrivilegedAction () {
  public Object run () {
    <privileged code>
  }
});
```

earlier versions used beginPrivileged() and endPrivileged() calls.

```
AccessController.beginPrivileged();
<privileged code>
AccessController.endPrivileged();
```

To understand the stack inspection algorithm let us assume the following operations.

- enablePrivilege(T)
- disablePrivilege(T)
- checkPrivilege(T)
- revertPrivilege(T)

where T is a target (permission in the Java terminology) we wish to protect.

Actions taken by these operations:

- enablePrivilege(T) puts an enabledPrivilege(T) flag on the current stack frame if the current class has access to T according to the policy.
- disablePrivilege(T) puts a disabledPrivilege(T) flag on the current stack frame (and removes enabledPrivilege(T) flag if present).
- revertPrivilege(T) removes enabledPrivilege(T) and disabledPrivilege(T) flags from the current stack frame if present.
- checkPrivilege(T) examines the stack as follows ...

```
checkPrivilege (T) {
  for SF from top stack frame to bottom stack frame {
    if (policy doesn't allow the class in SF to access T) throw ForbiddenException;
    if (SF has enabledPrivilege (T) flag) return;
    if (SF has disabledPrivilege (T) flag) throw ForbiddedException;
  }
  return; // reached bottom of stack
}
```

The ABLP Logic   Abadi, Burrows, Lampson and Plotkin, 1993

We will model stack inspection using the (subset of) ABLP logic described below. The language contains

- Principals, modeling persons, organizations as well as cryptographic keys.
- Targets, modeling resources we wish to protect.
- Statements, modeling utterances of principals.

The ABLP Logic   Abadi, Burrows, Lampson and Plotkin, 1993

We will model stack inspection using the (subset of) ABLP logic described below. The language contains

- Principals, modeling persons, organizations as well as cryptographic keys.
- Targets, modeling resources we wish to protect.
- Statements, modeling utterances of principals.
  – The statement P says $\mathsf{Ok}(T)$ means that the principal P is authorizing access to target T.

The ABLP Logic — Abadi, Burrows, Lampson and Plotkin, 1993

We will model stack inspection using the (subset of) ABLP logic described below. The language contains

- Principals, modeling persons, organizations as well as cryptographic keys.
- Targets, modeling resources we wish to protect.
- Statements, modeling utterances of principals.
  - The statement P says $\mathsf{Ok}(T)$ means that the principal P is authorizing access to target T.
  - $P \mid Q$ says s means P says (Q says s), i.e. P quotes Q as saying s.

The ABLP Logic   Abadi, Burrows, Lampson and Plotkin, 1993

We will model stack inspection using the (subset of) ABLP logic described below. The language contains

- Principals, modeling persons, organizations as well as cryptographic keys.
- Targets, modeling resources we wish to protect.
- Statements, modeling utterances of principals.
  - The statement P says Ok(T) means that the principal P is authorizing access to target T.
  - $P \mid Q$ says s means P says (Q says s), i.e. P quotes Q as saying s.
  - $P \wedge Q$ says s means that both P and Q say s.

The ABLP Logic — Abadi, Burrows, Lampson and Plotkin, 1993

We will model stack inspection using the (subset of) ABLP logic described below. The language contains

- Principals, modeling persons, organizations as well as cryptographic keys.
- Targets, modeling resources we wish to protect.
- Statements, modeling utterances of principals.
  - The statement P says $\mathsf{Ok}(T)$ means that the principal P is authorizing access to target T.
  - $P \mid Q$ says s means P says (Q says s), i.e. P quotes Q as saying s.
  - $P \wedge Q$ says s means that both P and Q say s.
  - $P \Rightarrow Q$ means that P speaks for Q, i.e. P has at least as much authority as Q.

We assume a set of atomic statements and atomic principals.

principal P ::=

$AtomicPrincipal$

$P_1 \wedge P_2$

$P_1 \mid P_2$

statement s ::=

$AtomicStatement$

$\mathsf{s}_1 \wedge \mathsf{s}_2$

$\mathsf{s}_1 \rightarrow \mathsf{s}_2$

P says s_1

$P_1 \Rightarrow P_2$

Example Given some s we define following new statements.

$$\mathsf{s}_1 \equiv (Alice \wedge Bob) \ \mathsf{says} \ (Charlie \Rightarrow (Alice \wedge Bob))$$

$Alice$ and Bob declare $Charlie$ to be their representative.

Example Given some s we define following new statements.

$$\mathsf{s}_1 \equiv (Alice \wedge Bob)\ \mathsf{says}\ (Charlie \Rightarrow (Alice \wedge Bob))$$

$Alice$ and Bob declare $Charlie$ to be their representative.

$$\mathsf{s}_2 \equiv Charlie \mid Alice\ \mathsf{says}\ \mathsf{s}$$

$Charlie$ quotes $Alice$ as saying s.

Example Given some s we define following new statements.

$$\mathsf{s}_1 \equiv (Alice \wedge Bob)\ \mathsf{says}\ (Charlie \Rightarrow (Alice \wedge Bob))$$

Alice and *Bob* declare *Charlie* to be their representative.

$$\mathsf{s}_2 \equiv Charlie \mid Alice\ \mathsf{says}\ \mathsf{s}$$

Charlie quotes *Alice* as saying s.

$$\mathsf{s}_3 \equiv (Alice\ \mathsf{says}\ \mathsf{s}) \rightarrow \mathsf{s}$$

If *Alice* says s then it must be true.

Example Given some s we define following new statements.

$$\mathsf{s}_1 \equiv (\mathit{Alice} \wedge \mathit{Bob})\ \mathsf{says}\ (\mathit{Charlie} {\Rightarrow} (\mathit{Alice} \wedge \mathit{Bob}))$$

Alice and *Bob* declare *Charlie* to be their representative.

$$\mathsf{s}_2 \equiv \mathit{Charlie} \mid \mathit{Alice}\ \mathsf{says}\ \mathsf{s}$$

Charlie quotes *Alice* as saying s.

$$\mathsf{s}_3 \equiv (\mathit{Alice}\ \mathsf{says}\ \mathsf{s}) {\rightarrow} \mathsf{s}$$

If *Alice* says s then it must be true.

Intuitively, from $\mathsf{s}_1 \wedge \mathsf{s}_2 \wedge \mathsf{s}_3$ we should be able to prove s.

Example Given some s we define following new statements.

$$\mathsf{s}_1 \equiv (Alice \wedge Bob)\ \mathsf{says}\ (Charlie \Rightarrow (Alice \wedge Bob))$$

Alice and *Bob* declare *Charlie* to be their representative.

$$\mathsf{s}_2 \equiv Charlie \mid Alice\ \mathsf{says}\ \mathsf{s}$$

Charlie quotes *Alice* as saying s.

$$\mathsf{s}_3 \equiv (Alice\ \mathsf{says}\ \mathsf{s}) \rightarrow \mathsf{s}$$

If *Alice* says s then it must be true.

Intuitively, from $\mathsf{s}_1 \wedge \mathsf{s}_2 \wedge \mathsf{s}_3$ we should be able to prove s.

For this we require certain rules (axioms) for making proofs.

Axioms about statements

1 *If* `s` *is an instance of a theorem of propositional logic then* `s` *is true in ABLP logic.*

Axioms about statements

1 *If* s *is an instance of a theorem of propositional logic then* s *is true in ABLP logic.*

E.g. the ABLP statement

$$(P \text{ says } \mathsf{s}) \rightarrow (P \text{ says } \mathsf{s})$$

is an instance of the propositional logic statement

$$X \rightarrow X$$

Axioms about statements

1 *If* s *is an instance of a theorem of propositional logic then* s *is true in ABLP logic.*

E.g. the ABLP statement

$$(P \text{ says s}) \rightarrow (P \text{ says s})$$

is an instance of the propositional logic statement

$$X \rightarrow X$$

The ABLP statement

$$(P \text{ says s}) \wedge ((P \text{ says s}) \rightarrow \text{s}) \rightarrow \text{s}$$

is an instance of the propositional logic statement

$$(X \wedge (X \rightarrow Y)) \rightarrow Y$$

Axioms about statements

1 *If* s *is an instance of a theorem of propositional logic then* s *is true in ABLP logic.*

E.g. the ABLP statement

$$(P \ \mathsf{says\ s}) \rightarrow (P \ \mathsf{says\ s})$$

is an instance of the propositional logic statement

$$X \rightarrow X$$

The ABLP statement

$$(P \ \mathsf{says\ s}) \wedge ((P \ \mathsf{says\ s}) \rightarrow \mathsf{s}) \rightarrow \mathsf{s}$$

is an instance of the propositional logic statement

$$(X \wedge (X \rightarrow Y)) \rightarrow Y$$

Hence both ABLP statements are true.

2 *If* s *and* $\mathsf{s} \longrightarrow \mathsf{s}'$ *then* s'.

2 *If* s *and* $\mathsf{s} \rightarrow \mathsf{s}'$ *then* s'.

3 $(P \text{ says } \mathsf{s} \wedge P \text{ says } (\mathsf{s} \rightarrow \mathsf{s}')) \rightarrow P \text{ says } \mathsf{s}'$

We can draw conclusions from statements made by principals.

2 *If* s *and* $\mathsf{s} \rightarrow \mathsf{s}'$ *then* s'.

3 $(P \;\mathsf{says}\; \mathsf{s} \wedge P \;\mathsf{says}\; (\mathsf{s} \rightarrow \mathsf{s}')) \rightarrow P \;\mathsf{says}\; \mathsf{s}'$

We can draw conclusions from statements made by principals.

4 *If* s *then* P says s *for every principal* P.

True ABLP statements are supported by all principals.

Example

Given statement Alice says $(\mathsf{s}_1 \wedge \mathsf{s}_2)$ how do we conclude that Alice says s_1.

Example

Given statement *Alice* says $(s_1 \wedge s_2)$ how do we conclude that *Alice* says s_1.

We use the following steps.

$(s_1 \wedge s_2) \longrightarrow s_1$	by (1)
Alice says $((s_1 \wedge s_2) \longrightarrow s_1)$	by (4)
Alice says s_1	by (3)

Axioms about principals

5 $(P \wedge Q)$ says s $\equiv$ $(P$ says s$) \wedge (Q$ says s$)$

6 $(P \mid Q)$ says s $\equiv$ P says $(Q$ says s$)$

7 $(P = Q) \rightarrow (P$ says s $\equiv$ Q says s$)$

$=$ is equality on principals.

8 $(P_1 \mid (P_2 \mid P_3)) = ((P_1 \mid P_2) \mid P_3)$

Quoting is associative.

9 $(P_1 \mid (P_2 \wedge P_3)) = (P_1 \mid P_2) \wedge (P_1 \mid P_3)$

Quoting distributes over conjunction

10 $(P \Rightarrow Q) \equiv (P = P \wedge Q)$

11 $(P \text{ says } (Q \Rightarrow P)) \rightarrow (Q \Rightarrow P)$

A principal is free to choose a representative.

Example We want to conclude s from the three statements:

- $(Alice \wedge Bob)$ says $(Charlie \Rightarrow (Alice \wedge Bob))$
- $Charlie \mid Alice$ says s
- $(Alice$ says s$) \rightarrow$ s

$$(Alice \wedge Bob) \text{ says } (Charlie \Rightarrow (Alice \wedge Bob))$$

$$\rightarrow (Charlie \Rightarrow (Alice \wedge Bob)) \qquad \text{by (11)}$$

$$(Charlie \Rightarrow (Alice \wedge Bob)) \qquad \text{by (2)}$$

$$Charlie = (Charlie \wedge Alice \wedge Bob) \qquad \text{by (10)}$$

$$Charlie \text{ says } (Alice \text{ says s}) \qquad \text{by (6)}$$

$$(Charlie \wedge Alice \wedge Bob) \text{ says } (Alice \text{ says s}) \qquad \text{by (7,2)}$$

$Alice$ says ($Alice$ says s) by (5,1,2)

$Alice$ says (($Alice$ says s)$\rightarrow$s) by (4)

$Alice$ says s by (3)

s by (2)

Modeling Java stack inspection using ABLP

Wallach, Felten, 1998

Code can be digitally signed by a signer. We treat code, public keys and signers as principals. Stack frames created during execution of code are also treated as principals. Targets (resources to be protected) are also treated as principals.

Modeling Java stack inspection using ABLP

Wallach, Felten, 1998

Code can be digitally signed by a signer. We treat code, public keys and signers as principals. Stack frames created during execution of code are also treated as principals. Targets (resources to be protected) are also treated as principals.

If K is a public key of S then we have the statement

$$K \Rightarrow S \qquad \text{(S1)}$$

Modeling Java stack inspection using ABLP

Wallach, Felten, 1998

Code can be digitally signed by a signer. We treat code, public keys and signers as principals. Stack frames created during execution of code are also treated as principals. Targets (resources to be protected) are also treated as principals.

If K is a public key of S then we have the statement

$$K \Rightarrow S \tag{S1}$$

If some code C was signed and K is the corresponding public key then we have the statement

$$K \ \mathsf{says} \ (C \Rightarrow K) \tag{S2}$$

If F is the stack frame generated for executing code C then we have the statement

$$F \Rightarrow C \tag{S3}$$

Frame credentials Φ = set of all valid statements of the form S1,S2 and S3.

If F is the stack frame generated for executing code C then we have the statement

$$F \Rightarrow C \tag{S3}$$

Frame credentials Φ = set of all valid statements of the form S1,S2 and S3.

Note that from K says $(C \Rightarrow K)$ using (11) we can conclude $C \Rightarrow K$.

Further we can show transitivity of $\Rightarrow$: given $A \Rightarrow B$ and $B \Rightarrow C$ we have:
$A = A \wedge B$ by (10)
$B = B \wedge C$ by (10)
Hence $A = A \wedge B \wedge C = A \wedge C$
Hence we have $A \Rightarrow C$

If F is the stack frame generated for executing code C then we have the statement

$$F \Rightarrow C \tag{S3}$$

Frame credentials Φ = set of all valid statements of the form S1,S2 and S3.

Note that from K says $(C \Rightarrow K)$ using (11) we can conclude $C \Rightarrow K$.

Further we can show transitivity of $\Rightarrow$: given $A \Rightarrow B$ and $B \Rightarrow C$ we have:
$A = A \wedge B$ by (10)
$B = B \wedge C$ by (10)
Hence $A = A \wedge B \wedge C = A \wedge C$
Hence we have $A \Rightarrow C$

Hence from S1, S2 and S3 we can conclude $F \Rightarrow S$.

For each target T we treat $\mathsf{Ok}(T)$ as an atomic statement.

It means that access to T is permitted.

We consider the axiom

$$(T \text{ says } \mathsf{Ok}(T)) \rightarrow \mathsf{Ok}(T) \tag{S4}$$

A target is always free to grant permission to itself.

Targets are dummy principals. They never speak, but other (non-dummy) principals representing them may speak for them.

Target credentials $\mathcal{T}$ is the set of such axioms for all targets T.

Policy for a virtual machine M is defined by a set access credentials $\mathcal{A}_{\mathsf{M}}$ of statements of the form $P \Rightarrow T$ where P is a principal and T is a target.

This rule means that the local policy of virtual machine M allows P to access T.

Stacks

During execution, at any point of time, a stack frame F has a belief set $\mathcal{B}_F$

This is updated as follows.

Stacks

During execution, at any point of time, a stack frame F has a belief set $\mathcal{B}_F$

This is updated as follows.

Starting the program For the initial stack frame F_0

$\mathcal{B}_{F_0} = \{\mathsf{Ok}(T) \mid T \text{ is a target}\}$.

Stacks

During execution, at any point of time, a stack frame F has a belief set $\mathcal{B}_F$

This is updated as follows.

Starting the program For the initial stack frame F_0

$\mathcal{B}_{F_0} = \{\mathsf{Ok}(T) \mid T \text{ is a target}\}$.

Enabling privileges

If stack frame F calls enablePrivilege(T) then we update: $\mathcal{B}_F := \mathcal{B}_F \cup \{\mathsf{Ok}(T)\}$.

Stacks

During execution, at any point of time, a stack frame F has a belief set $\mathcal{B}_F$

This is updated as follows.

Starting the program For the initial stack frame F_0

$\mathcal{B}_{F_0} = \{\mathsf{Ok}(T) \mid T \text{ is a target}\}$.

Enabling privileges

If stack frame F calls enablePrivilege(T) then we update: $\mathcal{B}_F := \mathcal{B}_F \cup \{\mathsf{Ok}(T)\}$.

Function calls

Function call from stack frame F creates a new stack frame G.

$\mathcal{B}_G = \{F \ \mathsf{says}\ \mathsf{s} \mid \mathsf{s} \in \mathcal{B}_F\}$.

Disabling privileges

If stack frame F calls $\mathsf{disablePrivilege}(T)$ then we update

$\mathcal{B}_F := \mathcal{B}_F \setminus \{\mathsf{s} \mid \mathsf{Ok}(T) \text{ occurs in } \mathsf{s}\}$

Disabling privileges

If stack frame F calls $\mathsf{disablePrivilege}(T)$ then we update $\mathcal{B}_F := \mathcal{B}_F \setminus \{\mathsf{s} \mid \mathsf{Ok}(T) \text{ occurs in } \mathsf{s}\}$

Reverting privileges

If stack frame F calls $\mathsf{revertPrivilege}(T)$ then we update $\mathcal{B}_F := \mathcal{B}_F \setminus \{\mathsf{Ok}(T)\}$

Disabling privileges

If stack frame F calls $\mathsf{disablePrivilege}(T)$ then we update $\mathcal{B}_F := \mathcal{B}_F \setminus \{\mathsf{s} \mid \mathsf{Ok}(T) \text{ occurs in } \mathsf{s}\}$

Reverting privileges

If stack frame F calls $\mathsf{revertPrivilege}(T)$ then we update $\mathcal{B}_F := \mathcal{B}_F \setminus \{\mathsf{Ok}(T)\}$

Checking privileges

When F calls $\mathsf{checkPrivilege}(T)$ then we check that $\mathsf{Ok}(T)$ can be concluded from the set

$\Phi \cup \mathcal{T} \cup \mathcal{A}_{\mathsf{M}} \cup \{F \ \mathsf{says}\ \mathsf{s} \mid \mathsf{s} \in \mathcal{B}_F\}$.