A New Model for Protection Systems

Ben Mankin

ben.mankin@anarres.org

University of Bath

The Past

- Timesharing
- Error detection
  - Base and bound
  - Virtual machines
- Communication
  - IPC
  - External

Security models exist *within* this framework.

The Future

We would like:

- Flexible groupware
- Mobile code and data
- Agents
- Dynamic web applications
- Fewer buzzwords

These all require *new* security models!

The Requirements

Many security problems arise:

- Sharing data is difficult
- Viruses and malicious agents abound
- Mobile code and data cannot be trusted
- Administration is centralised

We have new **requirements**!

The Requirements

We need a new model for security which is:

- Network-based rather than host-based
- Decentralised
- Administratively decentralised
- Dynamically modifiable
- **Provably secure**

This is the new *security problem*.

The Security Problem

Some parts of the security problem have been solved.

- Reference monitoring
- Authentication
- Secure communication

We call upon these solutions to build our system.

The Security Problem

The problems we must solve include:

- Control – who may control an agent?
- Trust – whom can an agent call on for help?
- Mobility – to where may an agent move?
- Access – may an agent access a resource?

We need a concept of responsibility.

Security Domains

A "*Security Domain*" is:

- Not concrete
- Implemented only in metadata
- A denotation of **responsibility**
- Used for security decisions
- Similar in purpose to a traditional VM

Everything is in a domain.

Security Domains

Some approaches to Security Domains:

- JDK 1.1 Applets – individual sandboxes
- JDK 1.4 Applications – ProtectionDomains
- Signed device drivers – Windows discards all metadata and breaks the domain!
- TCPA – extension of the provider domain into the client system

The Calculus

. . . will not be presented in this talk!

- A computational model for adding metadata.
- Expressed in lambda calculus.
- Correctly manages domains within a computation.
- Can be easily implemented in any language.
- Without modification of application code?

The Calculus

Data Ctx Rand	$\dfrac{e_2 \rightarrow^S_{\cdot} e_2'}{w_1\, e_2 \rightarrow^S_{\cdot} w_1\, e_2'}$
Data Ctx Frame	$\dfrac{e \rightarrow^R_{\cdot} e'}{R[e] \rightarrow^S_{\cdot} R[e']}$
Data Red Grant	grant R in $e \rightarrow^S_{\cdot} e$
Data Ctx Untaint	$\dfrac{e \rightarrow^S_{\cdot} e'}{\text{untaint } R \text{ in } e \rightarrow^S_{\cdot} \text{untaint } R \text{ in } e'}$
Data Red Untaint Frame	untaint R in $P[w] \rightarrow^S_{\cdot} (P \cup (R \cap S))$[untaint R in w]
Data Red Untaint Value	untaint R in $v \rightarrow^S_{\cdot} v$
Data Red Frame Rator	$R[w_1]w_2 \rightarrow^S_{\cdot} R[w_1\, S[w_2]]$
Data Red Frame Rand	$(\lambda_R x.e)P[w] \rightarrow^S_{\cdot} \begin{cases} (\lambda_R x.e[x := P[x]])w & \text{if } R \subseteq P = \mathtt{true} \\ \mathtt{fail} & \text{if } R \subseteq P = \mathtt{false} \\ \text{(not reducible)} & \text{if } R \subseteq P = \mho \end{cases}$
Data Red Appl	$(\lambda_R x.e)v \rightarrow^S_{\cdot} \begin{cases} e[x := S[v]] & \text{if } R \subseteq S = \mathtt{true} \\ \mathtt{fail} & \text{if } R \subseteq S = \mathtt{false} \\ \text{(not reducible)} & \text{if } R \subseteq S = \mho \end{cases}$

OK, I lied.

The Calculus

Consequences of the calculus include:

- The modified calculus does not affect outcomes.
- Security checks are performed transparently and correctly.
- Principals are sets of privileges.
- Sets form a lattice with union and intersection.
- Principals form a lattice.

Practical Systems

There is an "*ideal*" protection system which:

- Satisfies common business requirements:
  - Expressive and permissive
  - Decentralised
  - Dynamic and flexible
  - Provably secure (in linear time)
- Is similar to RBAC
- Uses transitive relations

Practical Systems

Properties of the ideal model include:

- The access relationship is transitive.
- Everything is a principal.
- There exists a superuser.
- Principals may form a lattice?

The Lattice Model

Let principals be points of an (artificial) lattice.

- Principals need not be countable.
- Permissions need not be countable.
- Distribution is easy.
- The system is computationally simple.
- The basic operations required by the calculus are trivial.
- The basic operations required by the ideal model are trivial.

Implementation

There are two parts to an implementation:

- A domain mechanism for the target system.
  - `x = new Computation();`
  - `perl -T`
  - `LD_PRELOAD="secdomain.so"`
- A universal convention for the lattice.
  - `com.ibm.projectA.objectB`

Implementation

Options for the domain implementation:

- Virtual machine or interpreter
  - Java (Sun JVM, IBM RVM, Intel OLR)
  - .net (Microsoft CLR)
  - Perl (Parrot VM, Perl 5 interpreter)
  - Very fine grained implementation

Implementation

Options for the domain implementation:

- Virtual machine or interpreter
  - Java (Sun JVM, IBM RVM, Intel OLR)
  - .net (Microsoft CLR)
  - Perl (Parrot VM, Perl 5 interpreter)
  - Very fine grained implementation
- Operating system
  - Very coarse grained.
  - No view of mutator.

Implementation

Options for the domain implementation:

- Virtual machine or interpreter
  - Java (Sun JVM, IBM RVM, Intel OLR)
  - .net (Microsoft CLR)
  - Perl (Parrot VM, Perl 5 interpreter)
  - Very fine grained implementation
- Operating system
  - Very coarse grained.
  - No view of mutator.
- Application library
  - Fine or coarse grained.
  - Room for programmer error.

Scenarios

- Global modification of virtual machines.
  - All computations become secure.
  - Mobility and groupware become trivial.

Scenarios

- Global modification of virtual machines.
  - All computations become secure.
  - Mobility and groupware become trivial.
- Modification of a single host.
  - Non local input is untrusted.

Scenarios

- Global modification of virtual machines.
  - All computations become secure.
  - Mobility and groupware become trivial.
- Modification of a single host.
  - Non local input is untrusted.
- Protocol implemented on border only.
  - Legacy systems may exist within borders.
  - No defence against internal attacks.

Scenarios

- Global modification of virtual machines.
  - All computations become secure.
  - Mobility and groupware become trivial.
- Modification of a single host.
  - Non local input is untrusted.
- Protocol implemented on border only.
  - Legacy systems may exist within borders.
  - No defence against internal attacks.
- High level application library.
  - Some room for programmer error.
  - More coarse grained security.

Conclusions

- We have new requirements for security.
- We have built a theoretical model to satisfy these requirements.
- The system is proof-based and amenable to analysis.
- Nonintrusive implementations are possible.

Thankyou

Please put money in the tin.