A Survey of Mobile Code Security Techniques

Instructor

Dr. Roshan Thomas
NAI Labs at Network Associates
8000 Westpark drive, Suite 600
McLean, VA 22102-3105

Summary

In this tutorial, we survey various approaches to addressing security of mobile code and agents. Mobile code security issues can be classified into two broad problem areas, namely the malicious code problem and the malicious host problem. With malicious code, we are concerned with executing useful mobile code (such as JAVA applets) while protecting the hosts from malicious ones. On the other hand, with malicious hosts, we are concerned with the protection of agents against malicious servers. For the malicious code problem the techniques surveyed include code blocking, authentication, safe interpreters, fault isolation, code inspection and verification and wrappers. For the malicious host problem, we look at techniques to detect tampering of agents as well as to preserve secrecy.

Short bio of speaker

Dr. Roshan Thomas is a Senior Security Engineer at TIS Labs and has over 10 years of experience as a researcher in the areas of computer security, fault-tolerance, distributed database management and multilevel-secure object-oriented distributed computing. He is currently involved in research projects investigating security issues for mobile code as well as survivability metrics and models for a distributed security services infrastructure. Dr. Thomas was a Principal Investigator on several DARPA-funded research projects that developed approaches to modularize functionality of security components such as firewalls and routers as well as various distributed authorization and access control models. He has published his research in major security journals and conference proceedings. Before joining the TIS research team, Dr. Thomas was a Principal Computer Scientist at Odyssey Research Associates, Inc (ORA).

Who's watching your network

A Survey of Mobile Code Security Techniques

22nd National Information Systems Security Conference
October 18 - 21, 1999

Dr. Roshan Thomas
NAI Labs at Network Associates
8000 West Park Drive, Suite 600
McLean, VA 22102
(703) 356-2225 x 112
roshan_thomas@nai.com

NAI LABS
The Security Research Division of Network Associates, Inc.

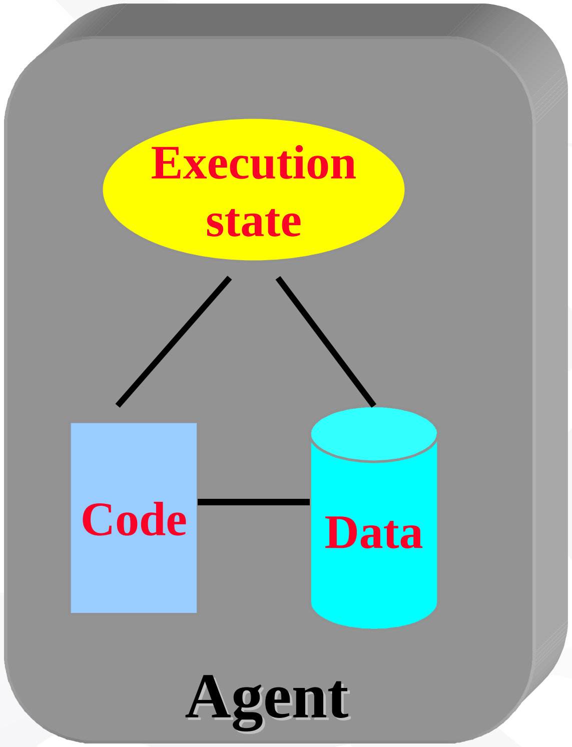
Outline of Presentation

- Defining mobile code
- Scope of mobile code security
- Techniques to prevent malicious code
- Techniques to prevent malicious hosts
- Summary and conclusions

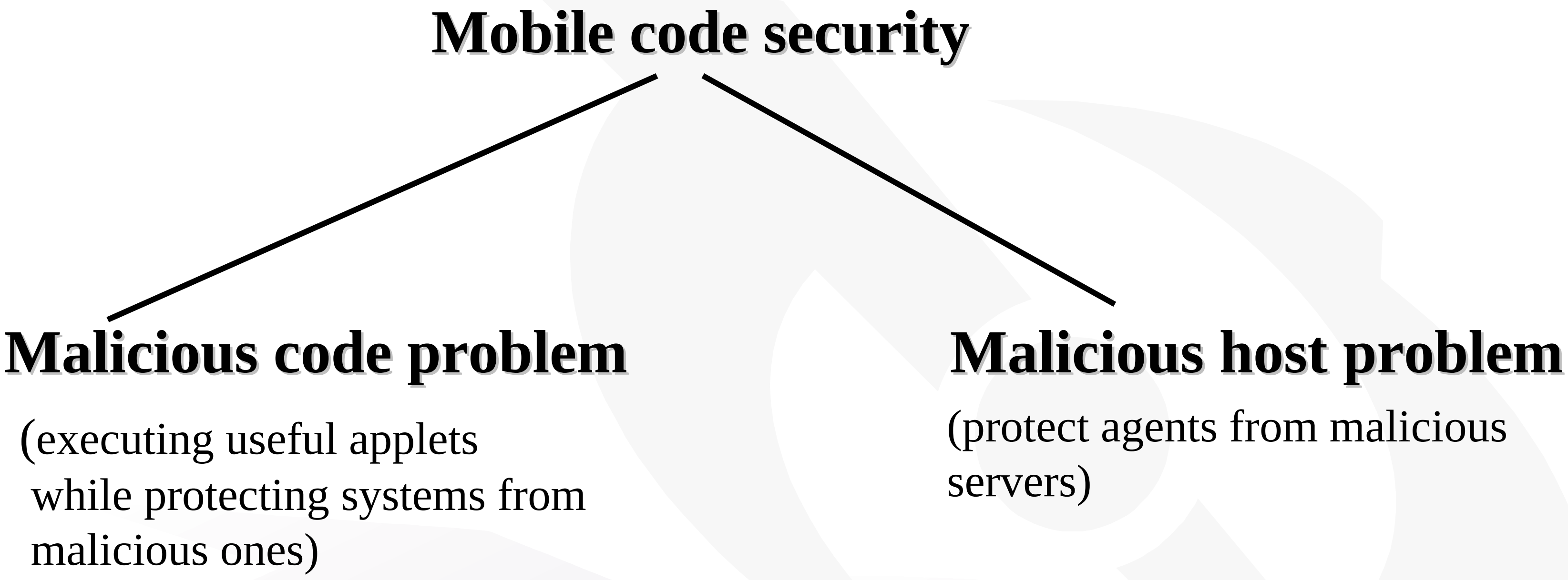
What is Mobile Code?

- **Mobile code is a general term used to refer to processes (executable code) that migrate and execute at remote hosts.**
- **Types of mobile code include:**
  - One-hop Agents (**weak mobility**)***, e.g.* Java applets.**
    - **Sent on demand from a server to a client machine and executed.**
    - **After execution, the agent's results or agent itself is returned to the agent owner that sent it.**
  - Multi-hop Agents **(strong mobility)**
    - **Sent out on the network to perform a series of tasks.**
    - **These agents may visit multiple agent platforms and communicate with other agents.**

Structure of an Agent



Scope of Mobile Code Security



Mobile code security

Malicious code problem

(executing useful applets while protecting systems from malicious ones)

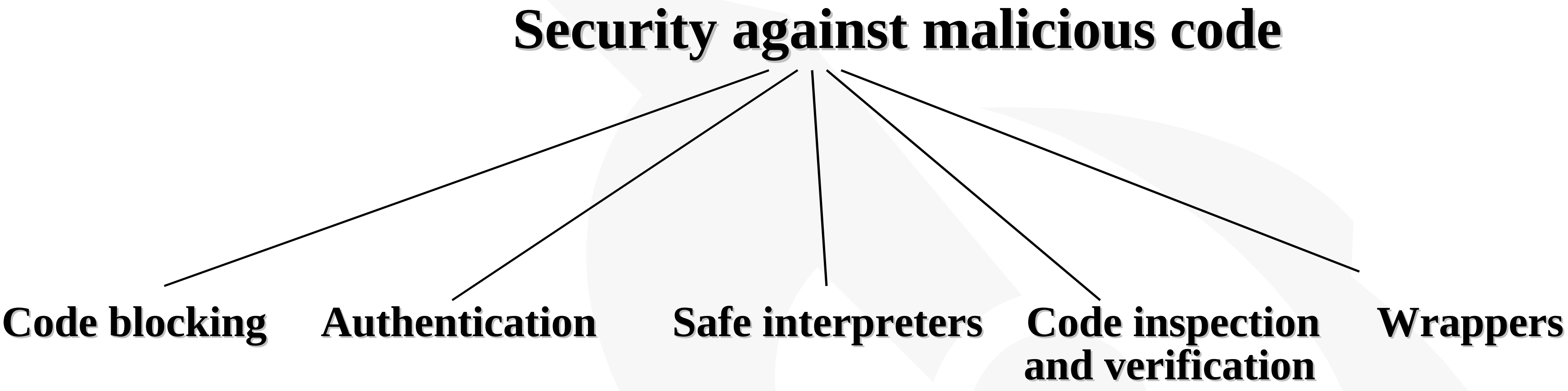
Malicious host problem

(protect agents from malicious servers)

Techniques to Prevent Malicious Code

Who's watching your network

Techniques to Prevent Malicious Code



Security against malicious code

- Code blocking
- Authentication
- Safe interpreters
- Code inspection and verification
- Wrappers

Techniques for Preventing Malicious Code: Code Blocking

Code blocking approaches

- **Disabling applications**
  - **E.g. switching off Java in Java-enabled browsers.**
  - **Relies on users complying with security policy.**
  - **Not easy to administer in a large environment.**
  - **Prevents intranet use of mobile code.**
- **Filtering**
  - **E.g. firewalls to filter out Web pages containing applets.**
  - **Does not rely on user compliance and management can be centralized.**
  - **Useful functionality at many popular web sites is denied to users.**

Code Blocking using Firewalls

Blocking strategies for JAVA **[Martin et. al , 1997]**

- Rewriting <applet> Tags
  - **Browser does not receive the <applet> and so no applet is fetched.**
  - **Be careful about parsing strategies.**
- Blocking by hex signatures
  - **Java class files start with the 4-byte hex signature CA FE BA BE**
  - **Apply in combination with <applet> blocker.**
- Blocking by filenames
  - **E.g. files with names ending in .class**
  - **Need to handle .zip files that encapsulate JAVA class files.**

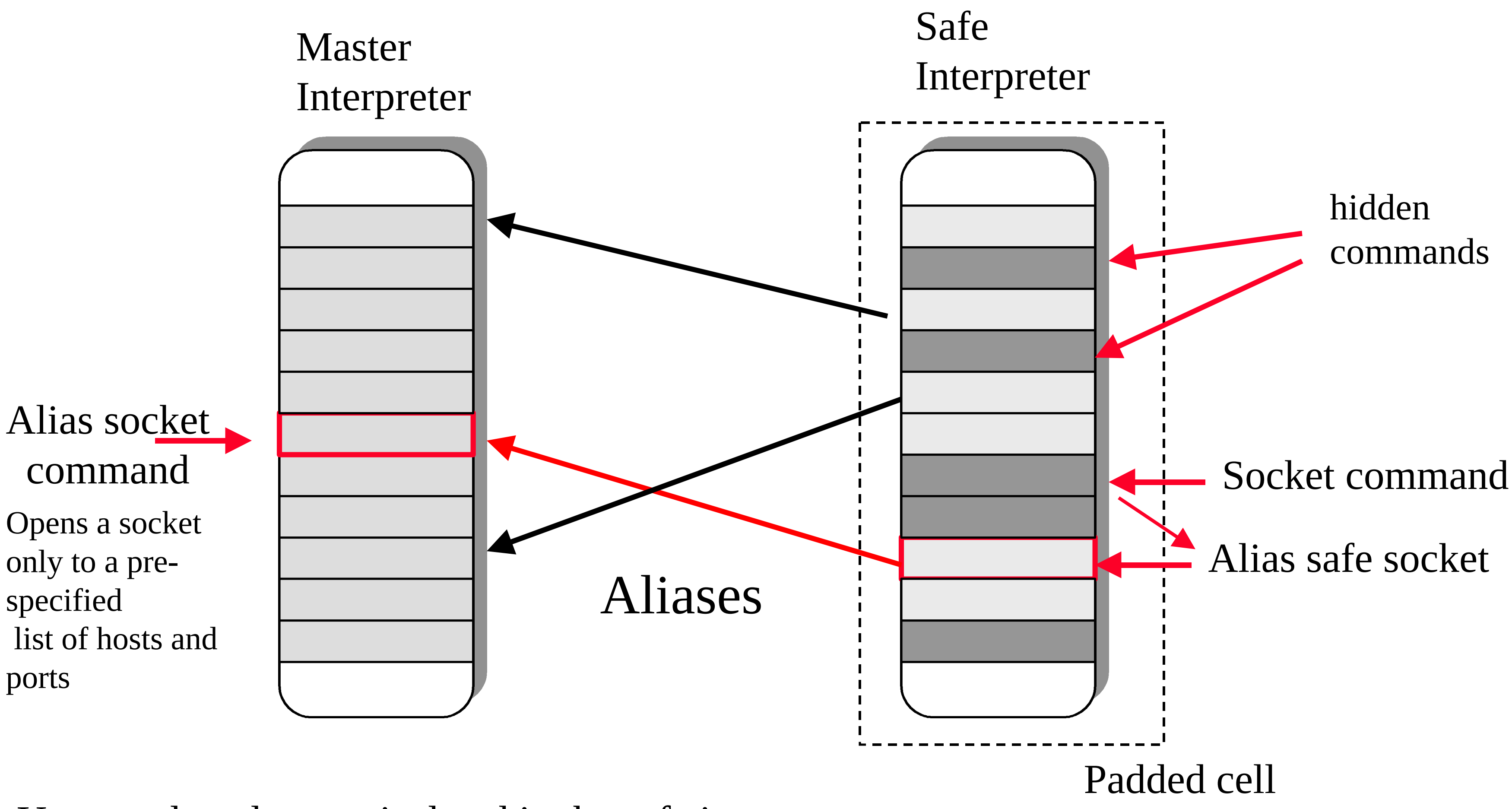
Techniques for Preventing Malicious Code: Authentication through Code Signing

- Based on the assurance obtained when the source of the code is trusted.
- On receiving mobile code, client verifies that it was signed by an entity on a trusted list.
- Used in JDK 1.1 and Active X.
  - Once signature is verified, code has full privileges.
- Problems
  - Trust model is all or nothing (trusted vs. untrusted).
  - To scale, we would need some public key infrastructure.
  - Limits users - even untrusted code may be useful and benign.
  - Code from a trusted source may still be unsafe and thus corrupt the host.

Techniques for Preventing Malicious Code: Safe Interpreters

- Instead of using compiled executables, interpret mobile code.
- Interpreter enforces a security policy.
- Each instruction is executed only if it satisfies the security policy.
- Examples of safe-interpreter systems
  - Safe-Tcl and extensions
  - Telescript/Odyssey
  - Java VM and extensions

Safe-Tcl and Security Policies



Master Interpreter

Safe Interpreter

hidden commands

Alias socket command

Opens a socket only to a pre-specified list of hosts and ports

Socket command

Alias safe socket

Aliases

Padded cell

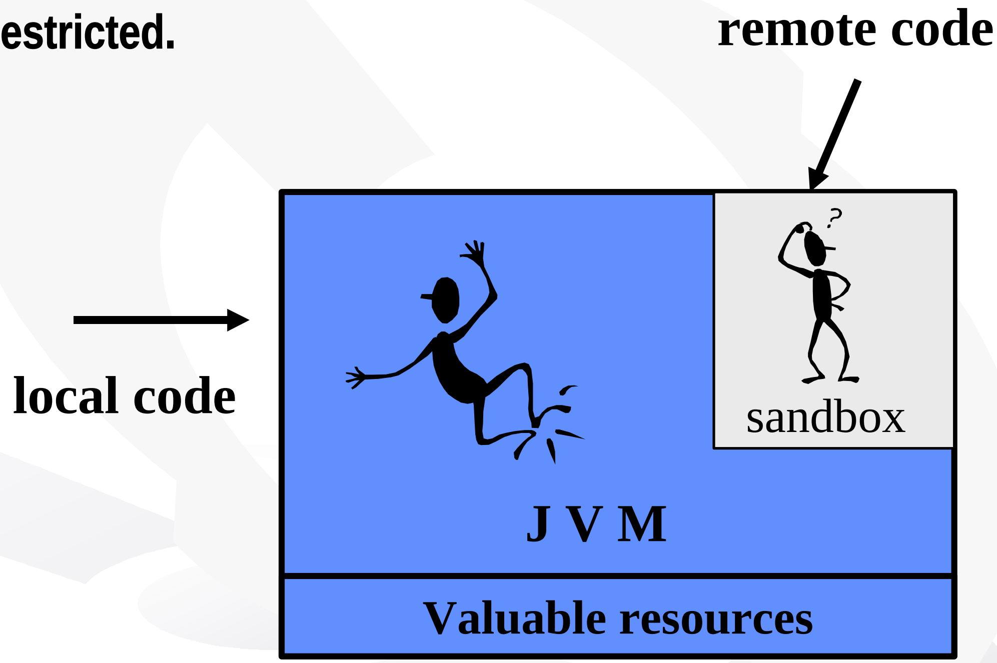
- Untrusted applets are isolated in the safe interpreter.
- Unsafe commands are hidden and cannot be invoked from the safe interpreter.
- Aliases, which are upcalls to the master interpreter, control use of unsafe commands.
- The master interpreter controls and manages aliases.

Safe Interpreters: Safe-Tcl *(cont'd)*

- Security policies in Safe-Tcl
  - A security policy consists of the commands available in safe interpreters using the policy(i.e. the set of aliases)
  - When an applet starts execution it requests a specific policy through an alias for loading policies..
  - If the master interpreter decides to permit the policy, it creates the associated aliases.
  - Composition of security policies is not safe and so an applet may use only a single security policy over its lifetime.
    - P1: It is safe for an applet to open network connections outside the firewall as as long as the applet cannot communicate with internal hosts.
    - P2: It is safe for an applet to read local files as along as there are no other external communications.
    - P1 and P2 do not compose safely as an applet that has both features can transmit local files outside the firewall, violating security.

Safe Interpreters: JAVA Security Model

- The sandbox security model
  - Black and white trust model.
  - Local code is trusted and has full access to system resources.
  - Downloaded remote code is restricted.



JAVA Sandbox Security Model *(cont’d)*

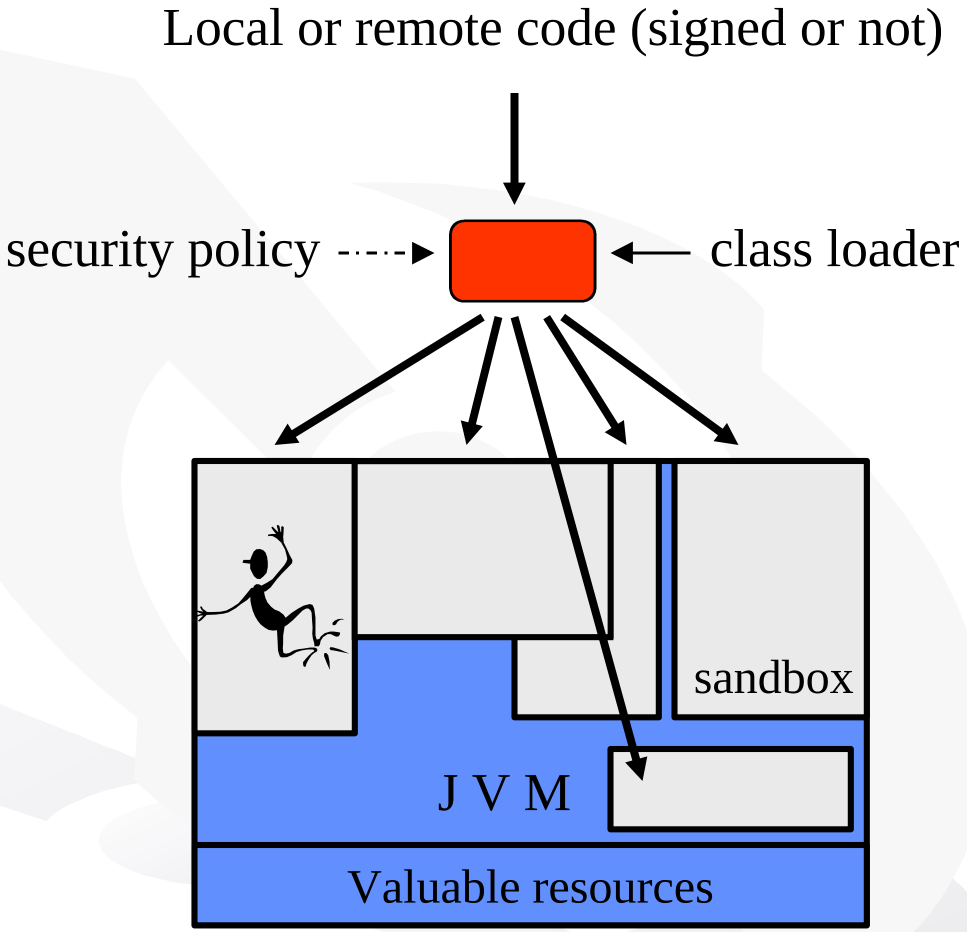
- The sandbox security model is built from
  - The Java applet class loader
    - Fetches remote applet’s code, enforces namespace separation.
  - The byte-code verifier
    - Checks byte code conformance to language specifications and applies built-in theorem prover.
  - The security manager
    - Mediates all system and dangerous methods that result in accesses to system resources.

Extensions to the Sandbox Model

- JDK 1.1.x
  - Supports digitally signed applets.
  - If signature can be verified, a remote applet is treated as local trusted code.
- JDK 1.2 (now renamed as JDK 2)
  - No concept of local trusted code. All code is subject to security controls.
  - Fine-grained domain-based and extensible access control.
  - Configurable security policy.
  - Extensible access control structure (typed and grouped permissions).

JDK 2 Security Model

- Each sandbox may have a different set of privileges.



Security Mechanisms in JDK 2

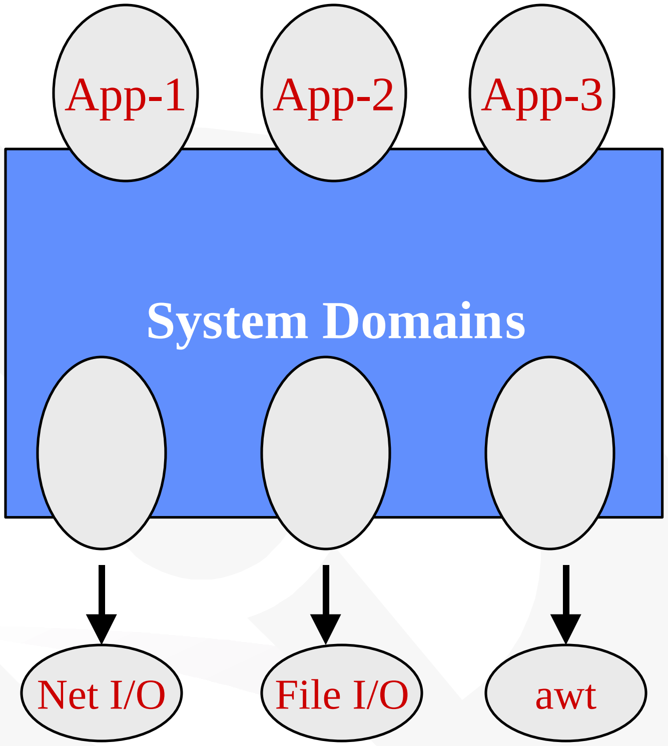
- Identity: **Every piece of code has an identity consisting of**
  - **Origin: location of the code as specified in a URL.**
  - **Signature: the public key of the private key used to sign the code.**
- Permissions:
  - **A permission consists of a target which is a file or directory and an action which is a read, write, execute, delete.**
  - **Permissions are subclassed from the abstract class** `java.security.Permission`
- Policy: **This is a mapping from an identity to a set of permissions.**

Example

```
grant codebase  “https://www.xyz.com/users/usr1”, signed by “*”
  {permission java.io.FilePermission “read, write’, “/folder1/tmp/*”;
   permission java.net.SocketPermission “connect”, “*.xyz.com”;}
```

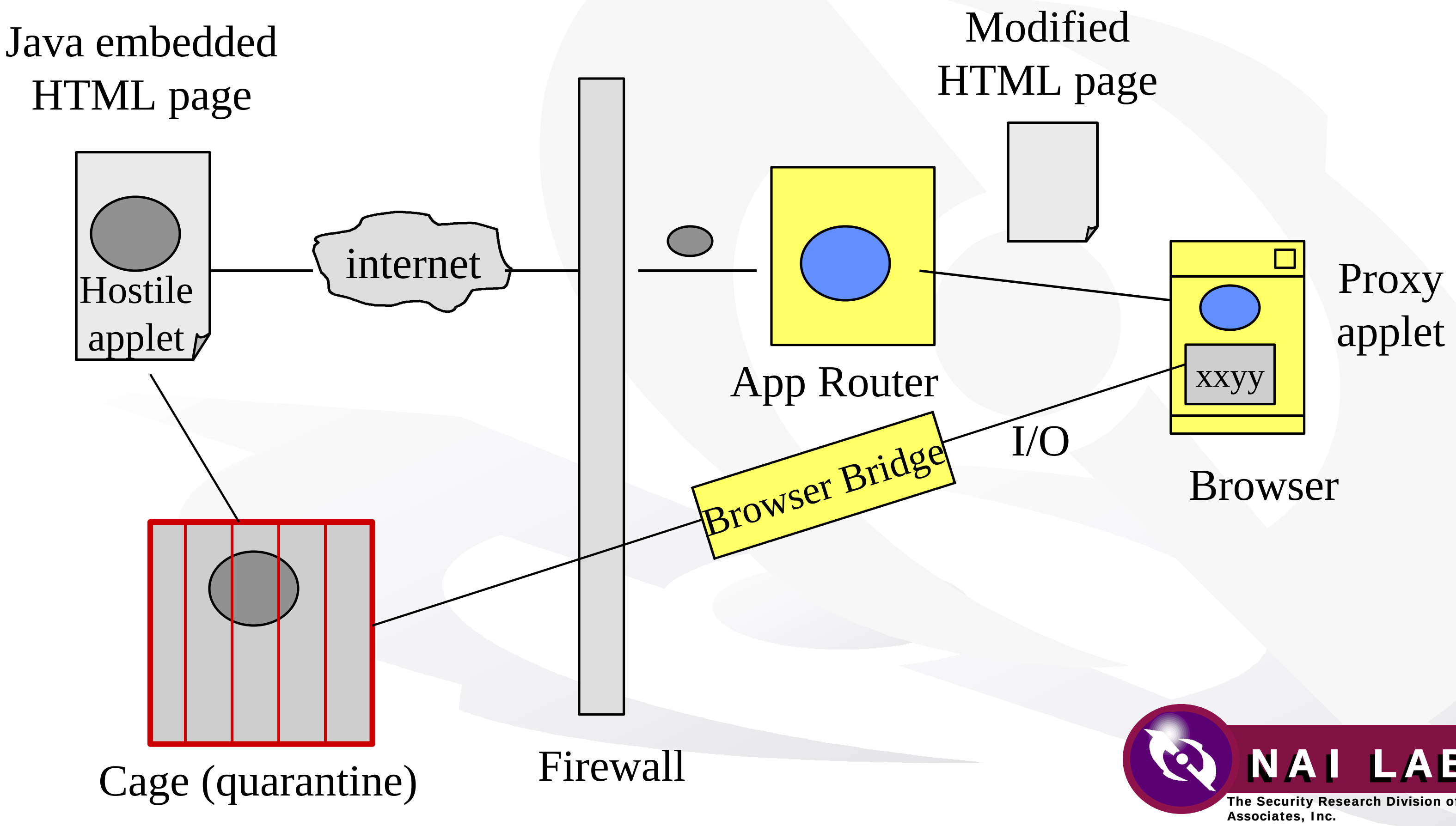
Sandboxes and Protection Domains

- Classes and objects belong to protection domains.
- Permissions are granted to domains.
- An execution domain may span several domains.
- The permission of a thread is the intersection of the permissions of all domains traversed (there are exceptions to this).
- A new thread inherits the security context of its parent.



Safe Interpreters: Sandboxing with Quarantine

- Internet applets are rerouted to a secure quarantine machine.
- Commercial solution offered by Digitivity (www.digitivity.com)
- A research prototype was built independently at AT&T Labs research.



Techniques for Preventing Malicious Code: Code Inspection

- These approaches intercept and inspect mobile code such as JAVA applets and Active X controls.
- Commercial solutions offered by
  - Finjan's SurfingGate 4
  - Network Associates WebScanX (folded into Virus Scan and Net Shield)
  - Trend Micro, Security7
- Typical features provided include:
  - Content inspection / byte-code scanning against known list of malicious code.
  - Validation of certificates and hash values.
  - Blocking of unwanted web sites..
  - In addition to HTTP, we can also monitor FTP traffic, email attachments, and compressed files for mobile code.

Techniques for Preventing Malicious Code: Wrappers

Server/gateway-side filtering followed by client-side wrapping.

- Trend Micro (www.antivirus.com)
  - Step 1: Proxy server does checks of certificates and hashes against block lists.
  - If applet passes step 1, then it is wrapped with enforcement code and a security policy and subsequently monitored by the client machine.
- Security7 (www.security7.com)
  - SafeGate: Server-side inspection engine.
  - SafeAgent: Local applications and resources are allocated into a special security zone by an administrator and kept isolated from downloaded code.

Who's watching your network

Techniques for Preventing Malicious Code: Code Verification with Proof-carrying Code

- With Proof-carrying code (PCC), a host can determine if it is safe to execute a program from an untrusted source.
- Host decides upon a safety policy which is then codified in the Edinburgh Logical framework (LF).
- Applet author has to generate a proof that the code confirms to the safety policy (certification).
- Code consumer validates the proof and executes the code if it passes (validation).
- Issues:
  - PCC sacrifices platform-independence for performance.
  - What program properties are expressible in LF logic is still an open research problem.

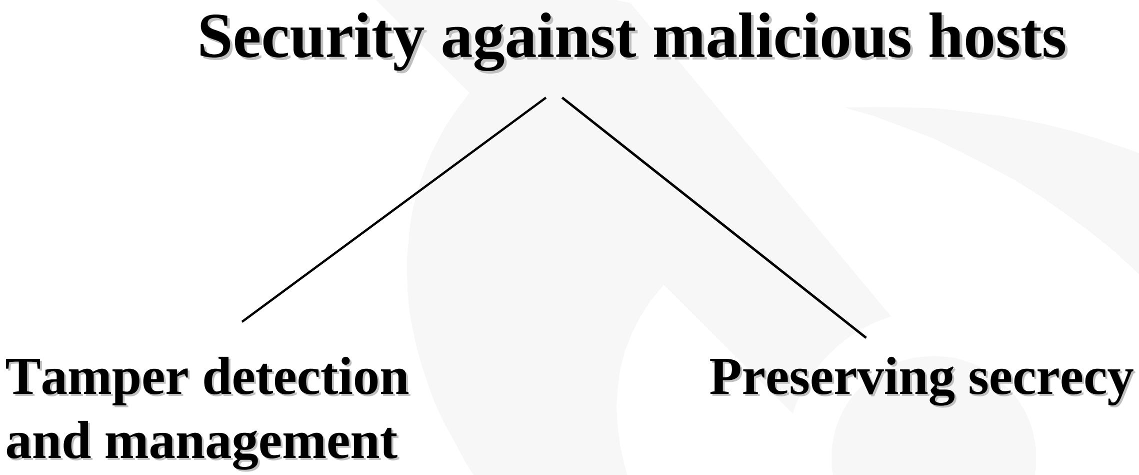
Techniques to Prevent Malicious Hosts

Threats by Malicious Hosts

- Leaking of code, data, control flow.
- Manipulation of code, data, control flow.
- Incorrect execution of code.
- Masquerading of the host.
- Denial of execution.
- Leaking and manipulation of the interaction with other agents.
- Returning wrong results of system calls issued by the agent.
- Tampering of agent itineraries.

Fritz Hohl. A model of attacks of malicious hosts against mobile agents,
4th Workshop on Mobile Object Systems (MOS '98).

Techniques to Prevent Malicious Hosts



Security against malicious hosts

Tamper detection and management

Preserving secrecy

Tamper Detection and Management Approaches

Tamper detection techniques include:

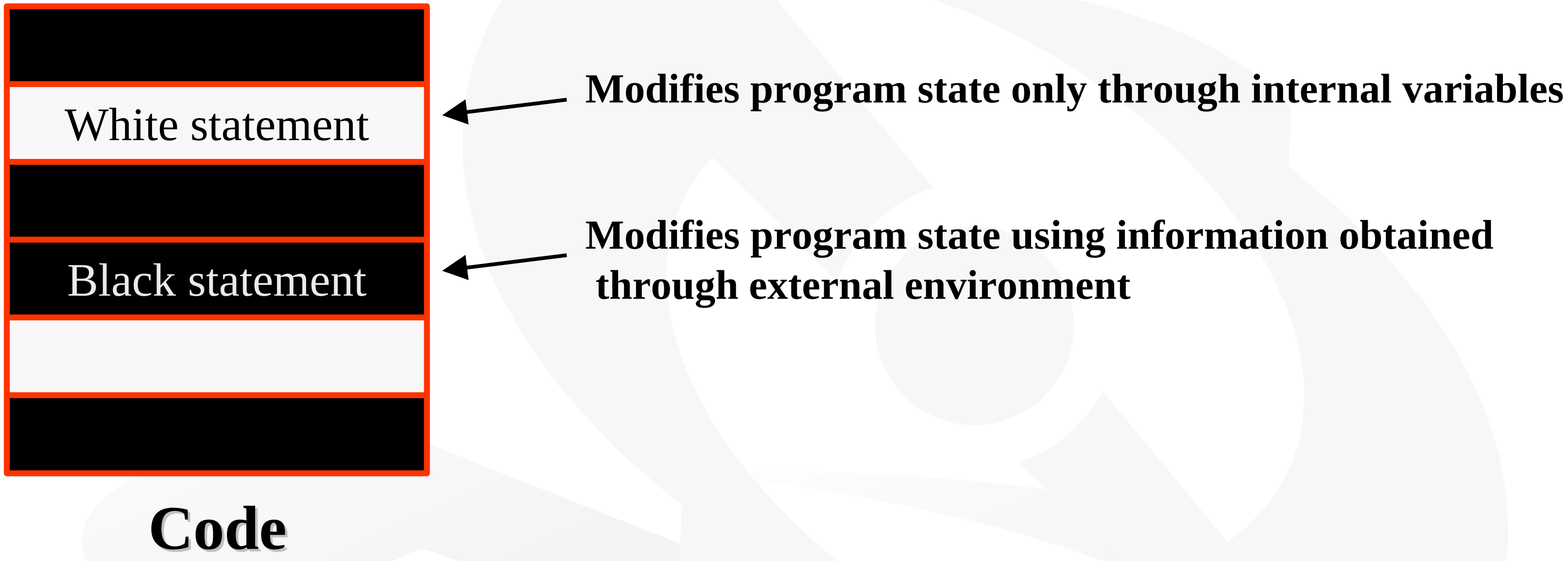
- **Execution tracing [Vigna]**
- **Partial Result Authentication Codes [Yee]**
- **Detection objects [Meadows]**
- **Protective assertions [Kassab and Voas]**

Tamper management techniques include:

- **Send agents only to trusted hosts.**
- **Multi-hop trust models.**
- **Tamper proofing with time-limited blackbox protection through code obfuscation and mess-up algorithms.**

Detecting Tampering: Execution Tracing

Use execution traces to verify program execution [Vigna].


White statement
Black statement
Modifies program state only through internal variables
Modifies program state using information obtained through external environment
Code

Detecting Tampering: Execution Tracing (*cont’d*)

- A trace Tc of the execution of a program C is composed of a sequence of pairs:

  < n, s >

  where n is unique id for a statement ;

  s is a signature ;

  - for black statements it contains the new values of internal variables after statement execution;
  - for white statements, it is empty.

Execution Tracing: Protocol Details

- A: **Agent owner,** B: **Agent platform,** C: **Mobile agent code**
- A' **: maybe the same as A or otherwise be a trusted digital notary**

When C terminates

- **B sends a signed message to A' containing a checksum of the program final state S1 and checksum of the trace Tc and a unique id iA.**

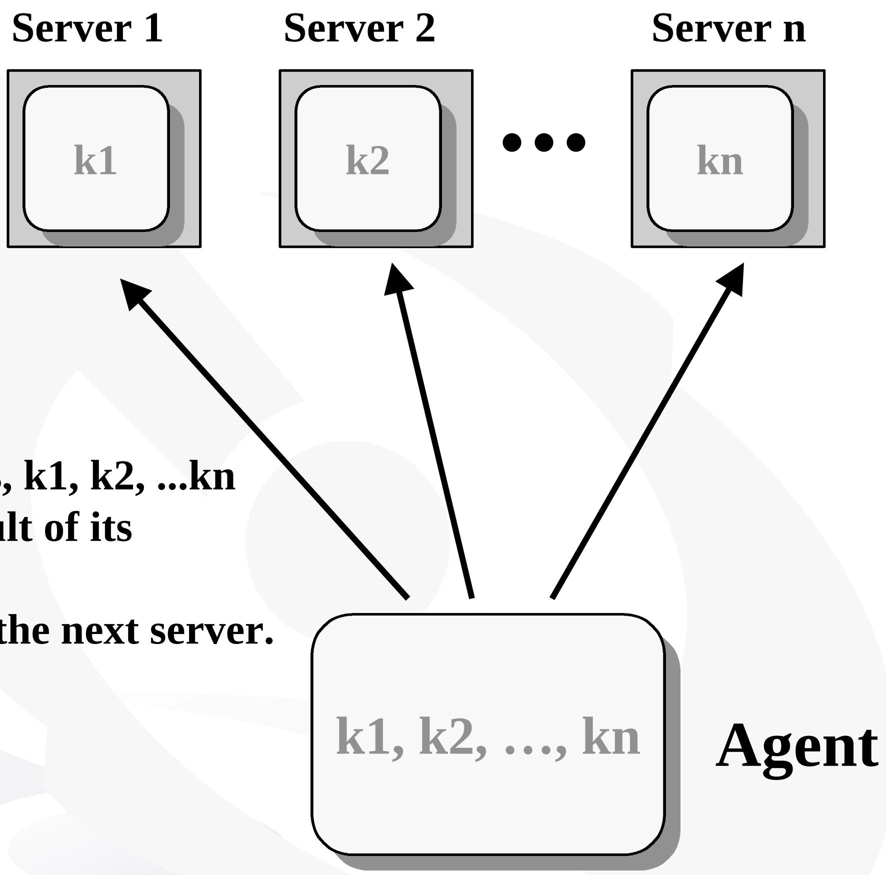
  m1: B --> A': B, Bs (H(S1), H(Tc), iA)

- **B sends a signed message to A containing the program final state S1, and iA.**

  m2: B --> A: B, Bs (Ap(S1), iA)

- **If A suspects that B cheated while executing C then**
  - **A can ask B to produce the trace and A' to produce the receipt messages.**
  - **A verifies the obtained trace with the value of H(Tc)**
  - **A replicates the execution of C following Tc.**
  - **The validation process should produce the final state S1; otherwise B cheated by modifying the code or some program variables.**

Detecting Tampering: Partial Result Authentication Codes



Solution proposed by Yee

- An agent is sent out with a set of secret keys, k1, k2, ...kn
- At server i, the agent uses ki to sign the result of its execution and producing a PRAC.
- Erase ki from agent state before moving to the next server.
- So what does this give us?

PRACs and Perfect Forward Integrity

- **A malicious server cannot forge the partial results from previous hops.**
- **PRACs allow an agent's owner (who has the keys k1, k2, …, kn) to cryptographically verify each partial result contained in a returning agent.**
- **These messages guarantee** perfect forward integrity:
  - If a mobile agent visits a sequence of servers S = s1, …, sn, and the first malicious server is sc, then none of the partial results generated at servers si, i < c, can be forged.
- **Yee also proposes various optimizations and variations**
  - **Use an initial key k1 and generate ki+1 from ki using a one-way function.**
  - **Publicly verifiable PRACs.**

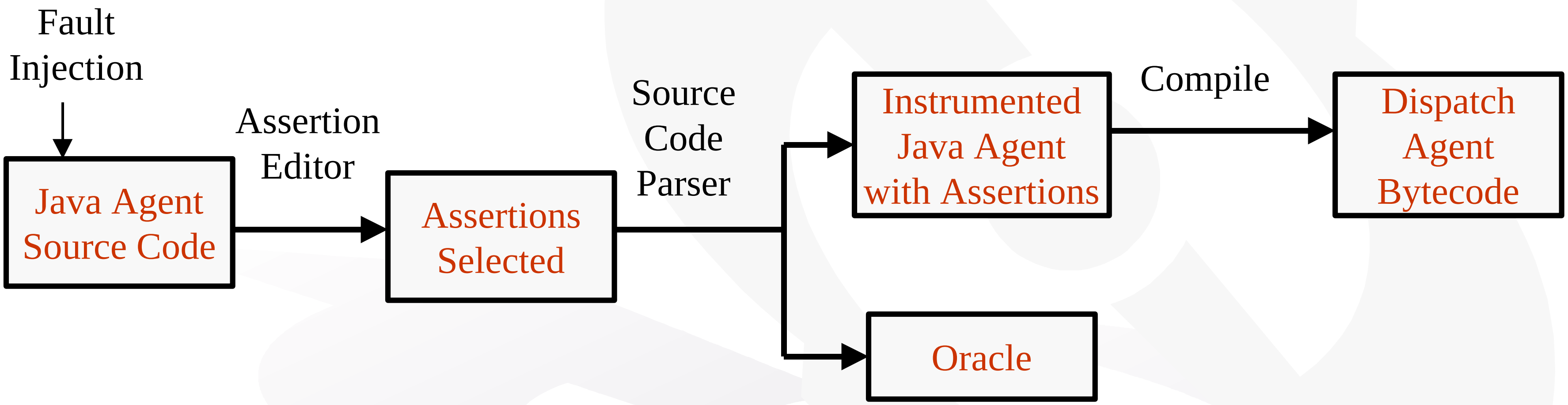
Detecting Tampering: Detection Objects

Meadows suggests the use of detection objects

- Dummy data items or attributes as part of agent’s state.
- Agents and host systems are unaware of these objects.
- If the detection objects have not been modified, then one can have reasonable confidence that legitimate data has not been corrupted.
- Issues:
  - Detection objects are application-specific.
  - Detection objects must be plausible enough to fool host systems and yet not adversely affect the agent’s computation/query results.
    - This may require query modification.
  - Answers to queries must still contain the detection objects so as to detect tampering.
  - Detection objects themselves may need to be updated frequently to avoid exposing them through the comparision of the results of several queries.

Detecting Tampering: Use of Protective Assertions

- Use assertions to reveal owner-specified agent state snapshots and increase agent observability.
- Assertions harden the agents by dynamically ensuring that the agent’s state remains acceptable.
- If assertions are bypassed by the host system, then the lack of information returning to the agent owner may indicate malicious activity.

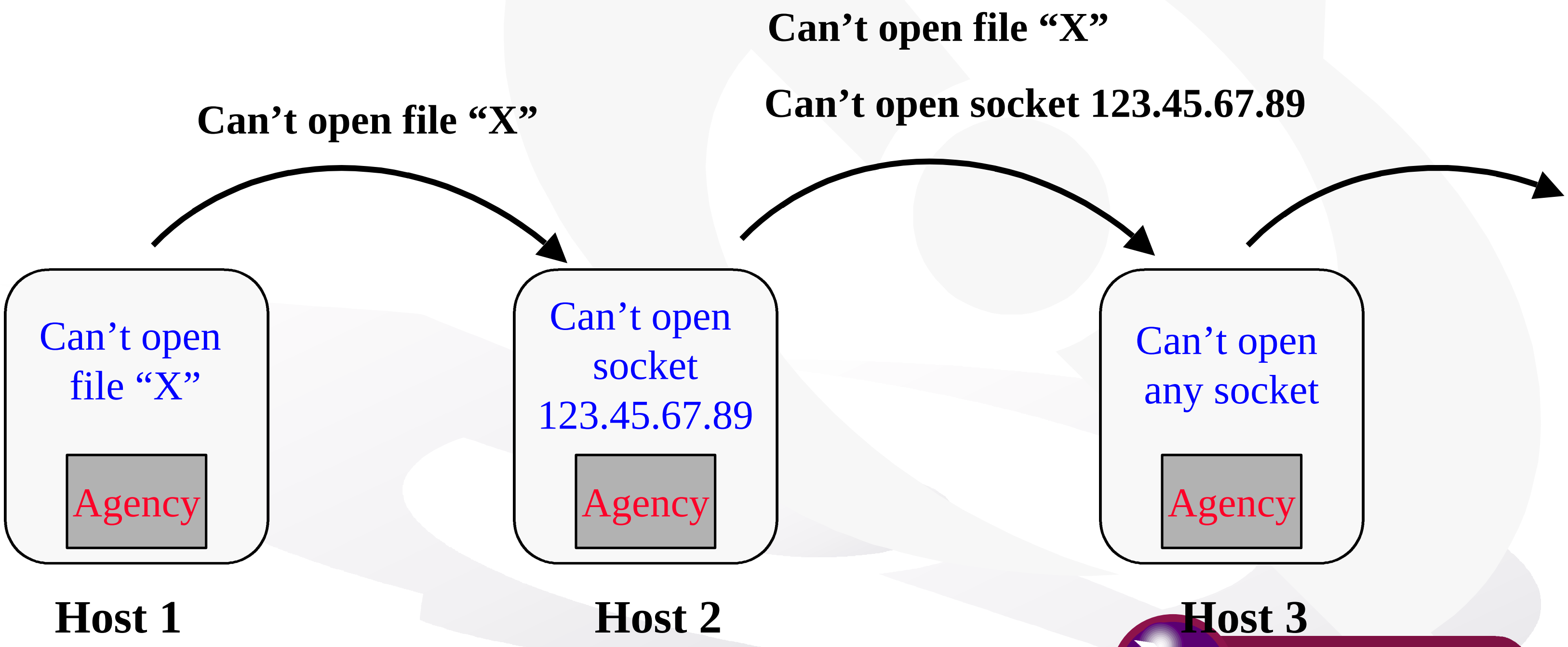


Process for Embedding Protective Assertions [Kassab and Voas]

Who's watching your network

Tamper Management: The Jumping Beans Multi-hop Trust Model

- Jumping beans (www.jumpingbeans.com) is a framework for mobilizing Java applications.
- Every host has an agency which is assigned a level of trust through an ACL.
- Worst case assumption, i.e. each host can be malicious.
- On each hop, a mobile application's ACL prior to the hop is merged with the ACL of the receiving agency in such a way that the security privileges decrease or remain the same.



NAI LABS

The Security Research Division of Network Associates, Inc.

Who's watching your network

Tamper-proofing: Time-limited Blackbox Protection through Code Obfuscation/Mess-up

Basic idea proposed by Hohl:

- Generate an executable agent from an agent specification so that it is not vulnerable to leakage and manipulation attacks.
- Such an agent has the following blackbox property.

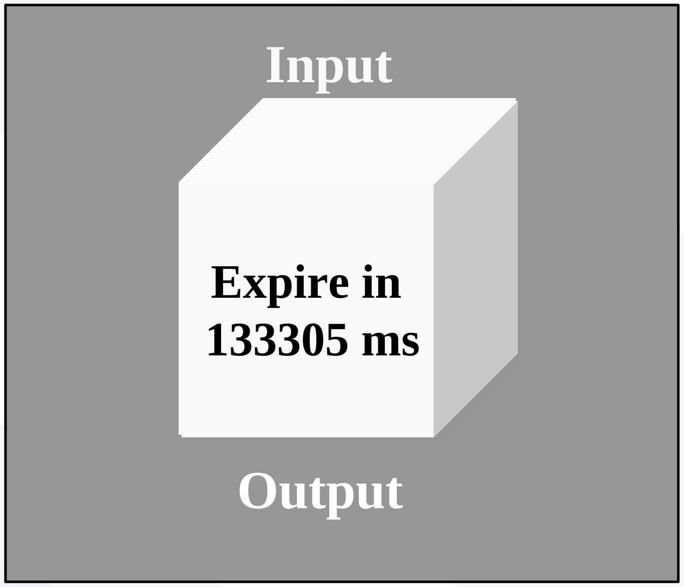
Time-limited Black box property

An agent is a time-limited blackbox if

1. for a certain known time interval
2. code and data of the agent specification cannot be leaked or modified

An attack after the protection interval

3. has no effect

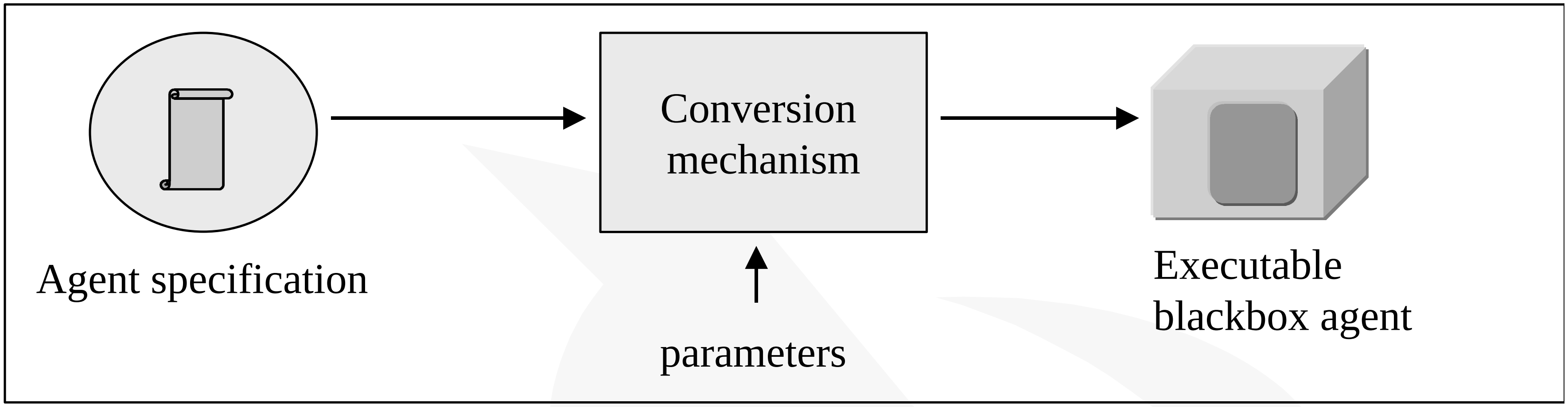


Input

Expire in 133305 ms

Output

Time-limited Blackbox Protection Approach



Agent specification

Conversion mechanism

parameters

Executable blackbox agent

- Don't allow the attacker to build a mental model of the agent in advance.
  - Create a new form of the agent dynamically at start of the protection interval.

- Make the process by which the attacker builds the mental model **time consumimg**. (we assume a lower bound on this time can be determined and is large enough for applications)
  - Use conversion algorithms based on code obfuscating and mess-up techniques, E.g:s
    - Hide the type of a statement by dynamically creating it at runtime.
    - Hide the location of a statement through dynamic code creation or by burying the statement in other statements.
    - Hide the type, value and location of data elements.

Preserving Secrecy: Encrypted Functions

- There is widespread belief that a host which executes a given program has full control over its execution, unless we use a trusted haven.
- For example [Chess]:

  "It is impossible to prevent agent tampering unless trusted (and tamper resistant) hardware is available. Without such hardware, a malicious host can always modify/manipulate the agent ..."

- Sanders and Tschudin Challenge these assumptions
  - Can a mobile agent conceal the program it wants to have executed?

Computing with Encrypted Functions

- **What is the problem?**

  Alice (agent owner**) has an algorithm to compute a function f.** Bob (host) **has an input *x* and is willing to compute f(*x*) for her, but Alice wants Bob to learn nothing substantial about f. Also, Bob should not need to interact with Alice during the computation of f(*x*).**

- **Protocol for "Non-interactive Computing with Encrypted Functions"**

  (E is some encryption function)

  (1) The owner of the agent encrypts f.

  (2) The owner creates a program P(E(f)) which implements E(f) and puts it in the agent.

  (3) The agent goes to the remote host, where it computes P(E(f))(x) and return home to the owner.

  (4) The owner decrypts P(E(f))(x) and obtains f(x).

Ongoing Research with Encrypted Functions

- Sander and Tschudin consider
  - representing the function f as a polynomial
  - showing certain classes of homomorphic encryption schemes would enable the protocol.
    - Why homomorphic schemes?
      - The encrypted program P_E constructed from a plain text program E has to be executable and therefore ordinary data encrpytion techniques cannot be applied. Also P and P_E have to be compatible with each other.
      - Mathematical analogue are algebraic structures where the compatible transformations are homomorphisms, i.e

        $h(x + y) = h(x) + h(y)$

Criteria for Evaluating Mobile Code Solutions

- Placement (server or client) and form of enforcement (centralized or distributed).
- Ease and scalability of security administration.
- Performance considerations.
- Expressive power/richness of of security policy.
- Ease of integration with existing applications/products.
- Degree of transparency (user awareness, compliance, intervention etc.) of the security solution to the user and the ability to customize this.
- Ability to coordinate with, as well as give/accept feedback to/from other security products (some degree of active security).
- Ability to adapt and learn based on history.
- Ability to audit operations.
- Platform independence.

Summary and Conclusions

- Increased interest in mobile code technology.
- Security remains a major impediment.
- Considerable progress in solving the malicious code problem.
- Research in solving the malicious host problem is still in its infancy.
- Ecommerce will be a major driver of technology.
- Industrial prototypes and solutions are emerging
  - IBM Aglets infrastructure.
  - Jumping Beans from Ad Astra Engineering.
  - Evolution of JAVA-based computing.