A Sandboxing Infrastructure for Alice ML

Bachelor Thesis

Andi Scharfstein, June 2006
Advisor: Andreas Rossberg

Programming Systems Lab
Saarland University

Open Programming

- The ability to dynamically import code at runtime

Open Programming

- The ability to dynamically import code at runtime
- Popular example: Java applets on web sites

Open Programming

- The ability to dynamically import code at runtime
- Popular example: Java applets on web sites
- Type safety guarantees that the program never goes wrong - but what about malicious programs?

Open Programming

- The ability to dynamically import code at runtime
- Popular example: Java applets on web sites
- Type safety guarantees that the program never goes wrong - but what about malicious programs?

➡ Open programming raises new security concerns

Open Programming: Dangers

- Resources may be corrupted (where „resources“ may mean any locally available set of data or system capabilities):

Open Programming: Dangers

- Resources may be corrupted (where „resources“ may mean any locally available set of data or system capabilities):
  - Deletion of data from hard disk

Open Programming: Dangers

- Resources may be corrupted (where „resources“ may mean any locally available set of data or system capabilities):
  - Deletion of data from hard disk
  - Publication of sensitive data

Open Programming: Dangers

- Resources may be corrupted (where „resources“ may mean any locally available set of data or system capabilities):
    - Deletion of data from hard disk
    - Publication of sensitive data
    - Abuse of system as spam server

Open Programming: Dangers

- Resources may be corrupted (where „resources“ may mean any locally available set of data or system capabilities):
  - Deletion of data from hard disk
  - Publication of sensitive data
  - Abuse of system as spam server
- How to prevent this from happening?

Naive approach

- Never grant access rights to local resources!

Naive approach

- Never grant access rights to local resources!
- May be too rigorous

Naive approach

- Never grant access rights to local resources!
- May be too rigorous
  - Write access may be necessary for saving files

Naive approach

- Never grant access rights to local resources!
- May be too rigorous
  - Write access may be necessary for saving files
  - Net access for „live“ functionality (e.g. football results)

Naive approach

- Never grant access rights to local resources!
- May be too rigorous
  - Write access may be necessary for saving files
  - Net access for „live“ functionality (e.g. football results)

➡ We need a flexible, general-purpose solution!

Sandboxes

- Provide a controlled (monitored) environment for execution of unknown code

Sandboxes

- Provide a controlled (monitored) environment for execution of unknown code
- Grant only access to resources considered „safe“

Sandboxes

- Provide a controlled (monitored) environment for execution of unknown code
- Grant only access to resources considered „safe“
- Dynamic distribution of access rights to client on a „need-to-have“ basis

Sandboxes

- Provide a controlled (monitored) environment for execution of unknown code
- Grant only access to resources considered „safe“
- Dynamic distribution of access rights to client on a „need-to-have“ basis
- In a given sandbox, only „smaller“ new sandboxes can be created (i.e., the child sandbox always inherits all limitations of its parent)

Sandboxes in Java

- Each program/applet has an associated security manager

Sandboxes in Java

- Each program/applet has an associated security manager
- Instance of class SecurityManager

Sandboxes in Java

- Each program/applet has an associated security manager
- Instance of class SecurityManager
- Supports calls like
  `security_manager.checkConnect(„google.com“, 80)`

Sandboxes in Java

- Each program/applet has an associated security manager
- Instance of class SecurityManager
- Supports calls like
  `security_manager.checkConnect(„google.com“, 80)`
- Is utilized by every security-sensitive API function

Sandboxes in Java

- Example: File Deletion

Sandboxes in Java

- Example: File Deletion
  - Application calls API function
    `file.delete();`

Sandboxes in Java

- Example: File Deletion
  - Application calls API function
    `file.delete();`
  - API function checks permissions (and possibly throws SecurityException)

```
SecurityManager sec = System.getSecurityManager();
sec.checkDelete(file.getName());
```

Sandboxes in Java

- Example: File Deletion

  - Application calls API function

    ```
    file.delete();
    ```

  - API function checks permissions (and possibly throws SecurityException)

    ```
    SecurityManager sec = System.getSecurityManager();
    sec.checkDelete(file.getName());
    ```

  - If permission is granted, API function deletes file and returns

    ```
    <delete file>
    return true;
    ```

Sandboxes in Java

- Example: File Deletion
  - Application calls API function
    ```
    file.delete();
    ```
  - API function checks permissions (and possibly throws SecurityException)
    ```
    SecurityManager sec = System.getSecurityManager();
    sec.checkDelete(file.getName());
    ```
  - If permission is granted, API function deletes file and returns
    ```
    <delete file>
    return true;
    ```
- Since applets only see API functions, no security breach is possible

Further Security Issues

- Clients should not be able to modify their security policies, e.g. by

Further Security Issues

- Clients should not be able to modify their security policies, e.g. by
  - gaining write access to permissions file or interpreter binary

Further Security Issues

- Clients should not be able to modify their security policies, e.g. by
  - gaining write access to permissions file or interpreter binary
  - executing unchecked native code

Further Security Issues

- Clients should not be able to modify their security policies, e.g. by
  - gaining write access to permissions file or interpreter binary
  - executing unchecked native code
  - creating their own security managers

Further Security Issues

- Clients should not be able to modify their security policies, e.g. by
  - gaining write access to permissions file or interpreter binary
  - executing unchecked native code
  - creating their own security managers
- Careful API design: insert checks at the proper place!

Sandboxes in Alice ML

- Open programming is supported by pickling

Sandboxes in Alice ML

- Open programming is supported by pickling
- Pickles never contain references to resources

Sandboxes in Alice ML

- Open programming is supported by pickling
- Pickles never contain references to resources
- Resource access is gained via the component manager structure:

```
COMPONENT_MANAGER = sig
  load: Url -> Component
  eval: Url * Component -> Package
  link: Url -> Package
... end
```

Sandboxes in Alice ML

- Open programming is supported by pickling
- Pickles never contain references to resources
- Resource access is gained via the component manager structure:

```
COMPONENT_MANAGER = sig
  load: Url -> Component
  eval: Url * Component -> Package
  link: Url -> Package
... end
```

- Component manager corresponds to Java class loader + security manager

Sandboxes in Alice ML

- Open programming is supported by pickling
- Pickles never contain references to resources
- Resource access is gained via the component manager structure:

```
COMPONENT_MANAGER = sig
  load: Url -> Component
  eval: Url * Component -> Package
  link: Url -> Package
... end
```

- Component manager corresponds to Java class loader + security manager
- Implicit inheritance of component managers along `import` chains

Sandboxes in Alice ML

- Access policies are created by implementing custom component managers (e.g. with different `link` functions)

Sandboxes in Alice ML

- Access policies are created by implementing custom component managers (e.g. with different `link` functions)
- Possible models:

Sandboxes in Alice ML

- Access policies are created by implementing custom component managers (e.g. with different `link` functions)

- Possible models:

  - Reject or accept requests based on URL syntax (e.g. „`x-alice:/lib/system/TextIO`“ might be rejected)

Sandboxes in Alice ML

- Access policies are created by implementing custom component managers (e.g. with different `link` functions)
- Possible models:
  - Reject or accept requests based on URL syntax (e.g. „`x-alice:/lib/system/TextIO`“ might be rejected)
  - Restrict module signatures (e.g. only supply the parts from `TextIO` that require read access to the stream)

Sandboxes in Alice ML

- Access policies are created by implementing custom component managers (e.g. with different `link` functions)
- Possible models:
  - Reject or accept requests based on URL syntax (e.g. „`x-alice:/lib/system/TextIO`“ might be rejected)
  - Restrict module signatures (e.g. only supply the parts from `TextIO` that require read access to the stream)
  - Wrap module functions with dynamic security checks (similar to Java)

Sandboxes in Alice ML

- Access policies are created by implementing custom component managers (e.g. with different `link` functions)
- Possible models:
  - Reject or accept requests based on URL syntax (e.g. „`x-alice:/lib/system/TextIO`“ might be rejected)
  - Restrict module signatures (e.g. only supply the parts from `TextIO` that require read access to the stream)
  - Wrap module functions with dynamic security checks (similar to Java)
- Introduce convenience functor `SECURITY_POLICY -> COMPONENT_MANAGER`

References

- Li Gong: Secure Java Class Loading
  IEEE Internet Computing, 2(6):56–61, 1998
- Sun Microsystems: Java 2 Platform SE 5.0 API Specification, 2004
- Xavier Leroy: Computer Security from a Programming Language and Static Analysis Perspective
  12th European Symposium on Programming, Lecture Notes in Computer Science, 2618:1–9, 2003
- Xavier Leroy: Java bytecode verification: algorithms and formalizations
  Journal of Automated Reasoning, 30(3–4):235–269, 2003
- Andreas Rossberg: The Missing Link – Dynamic Components for ML
  ICFP 2006
- Peter Sewell and Jan Vitek: Secure Composition of Insecure Components
  12th Computer Security Foundations Workshop, IEEE Computer Society Press, 1999