Secure Component Composition for Personal Ubiquitous Computing

ProgNet Workshop December 2003

16th December 2003

Dr. David Llewellyn-Jones, Prof. Madjid Merabti, Dr. Qi Shi, Dr. Bob Askwith

School of Computing and Mathematical Statistics
Liverpool John Moores University
James Parsons Building
Byrom Street
Liverpool L3 3AF

{D.Llewellyn-Jones, M.Merabti, Q.Shi, R.Askwith}@livjm.ac.uk
http://www.cms.livjm.ac.uk/PUCsec/
Overview

- Introduction
- Security Framework
- Component Composition
- Compose XML application
- Conclusions
- Future Work
Introduction: Component Applications

- All modern applications are built from components
  - They allow complex applications to be built from simpler parts
  - It suits the top-down procedural and OO paradigms of C, C++, Java etc.
  - Components can be reused, such as with OLE or DLLs

- From a security perspective it can also be very useful
  - Analysing smaller individual executables is quicker

- In a Ubiquitous Computing environment, devices can be seen as components of a larger networked structure
Introduction: The effect of combining components

- We intend to use the component structure of larger applications to develop a security framework for applications in a ubiquitous environment.
- Our work has looked at several areas, including the way properties of individual components can be established.
- Today I want to talk about how to deal with the composition of these components.
- We will therefore assume that the properties of the individual components are already known.
- The question then arises: if we know the security properties of the individual components, what happens when they are combined into a larger application?
Introduction: Sandboxing

- **What happens once we know the security properties of an application?**
  - If the application has serious security flaws we could elect not to execute it
  - If we know that an application has no known security flaws, we may assume that it is safe to run

- **More generally, we can use sandbox techniques to counteract the effect of the security flaws**
  - The aim is to maximise security, whilst minimising the restrictions on the application
  - The strongest restriction that can be applied is to prevent an application from running at all, but often much weaker restricts are effective
  - By knowing the security properties of an application, we can dynamically tailor the sandbox to provide the best environment

- **This is particularly useful in a dynamic environment**
A number of components enter the framework
- The application will be built by combining the individual components
- We assume the security properties for each component is already known

There are then three stages to the process
- The properties of the components are analysed together during component composition analysis. This establishes the security properties of the combined application and will be discussed in greater detail
- The components are combined into a complete application before execution
- Based on the security properties of the combined application, the code is executed within a secure sandbox wrapper.
Component Composition

- The intention is to establish the security properties for a combined application from the properties of its individual components
  - The security properties are properties of the individual components
- Security composition properties are characterised by
  - Separability: the idea that the property can be established for a component independent of its context
  - Properties of the inputs and outputs: the trace of a component defined by the data entering and leaving the component interface
  - Theoretical notions: secure composition properties tend to be strict, making practical implementation difficult
- Many theoretical security properties exist, for example
  - NI: non-interference
    - Definition: An interface \( E \) of a component is said to satisfy non-interference iff for any trace \( t \in T_E \) there exists a trace \( t' \in T_E \) such that \( t' \mid H_I_E = \emptyset \) and \( t' \mid (L_I_E \cup L_O_E) = t \mid (L_I_E \cup L_O_E) \).
  - BSD: backward strict deletion
  - BSI: backward strict insertion
Component Composition

- The separability of most security properties has a restricting influence on them.

- Results tend to take the form that "If components A and B both satisfy property X, then the composition of the components AB will also satisfy X."

- This is equivalent to saying that the security of a composed application is as weak as the weakest component.

- In general this is not the case. For example the case of PGP from the email client earlier. The Java sandbox is a particular example of this.
Component Composition

- Qi Shi and N Zhang have demonstrated a property “Composably Assured” (CA) that can be used in place of stronger security properties.
- Components in certain configurations are only required to satisfy CA in order for the composed application to satisfy the stronger security property.
- However, in this case the components must be configured in a tree structure.
- External interfaces are still required to satisfy the stronger security property.
- We want a system that can determine if such requirements have been satisfied.
Distinguishing trees
Component Composition

• We produce a practical implementation in which to use theoretical component composition results
• Our technique must distinguish between component topologies in a highly flexible manner
• An XML application is used to describe templates for composition topologies
• Topologies are compared against these templates, and if satisfied, the composed application will exhibit the properties defined in the XML template
• This allows sandbox techniques to be implement only when they are necessary to enforce security based on the properties of the composed application
A PROLOG-like language

• We use an XML application to define component topology templates
  – XML is a well-understood and standard technology
  – Readily available XML parsers could be used to undertake the laborious syntax parsing of the language

• In order to achieve maximum flexibility without sacrificing power, we use a Turing complete language to define templates

• However, the language had to fit the task; it had to be straightforward to define the more obvious class of topologies

• We used a Logic-like language (e.g. PROLOG) within the XML meta language
A PROLOG-like language

• Why use a language like PROLOG?
  – Logic languages use a set of rules to define a program. Unlike functional or OO languages such as C or Java, program flow is *not* defined strictly by the code.
  – Program flow is chosen at run-time by the interpreter and is constructed as a tree based on the rules in the program. The process is a little like BNF parsing, but more flexible.

1. `ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y)`
2. `ancestor(X,X)`
3. `parent(amy,bob)`

```
parent(X,Z), ancestor(Z,bob)       \{X=bob\}       success
  \______________________/
   \______________________/
  \______________________/
ancestor(bob,bob) \{X=amy\}       success
  \______________________/
   \______________________/
  \______________________/
parent(bob,Z), ancestor(Z,bob) \{X=amy\}       failure
```

```prolog
(1) ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y)
(2) ancestor(X,X)
(3) parent(amy,bob)
```
The Compose XML application

- The execution flow determines satisfaction of the component topology
  - We use the component topology to define the program execution flow. A template is satisfied if the program is able to fully run and terminate.
  - Although mathematically interesting, a number of practical difficulties can make PROLOG a hard language to use. However, since most of these relate to the method used to construct the control tree dictating execution flow, these difficulties do not apply in our case since the flow is predetermined by the component topology.
  - Simple trees can be defined in a very straightforward manner, without obvious programming techniques.
  - Nonetheless, it is Turing complete, ensuring that all computable templates can be defined.
The Compose XML application

- A template for a binary tree with components satisfying given security properties
- Sandbox techniques can be applied to the application or relaxed to suit the result

```
<configuration id="c3">
  <component id="c4">
    <input format=""/>
    <output format="id1 id1" cycle="disallow"/>
    <component id="c5">
      <input format="id1" cycle="disallow"/>
      <output config="c4" format="id1 id1" cycle="disallow"/>
      <output format=""/>
    </component>
  </component>
</configuration>
```
The Compose XML application

A more interesting example

– This template will be satisfied only if a prime number of components are connected linearly.
Conclusions

• Component Composition is an effective way to determine the security properties of larger applications

• To be effective, we require a method able to recognise the composition topology of components with certain properties

• We have developed an XML application designed to do this
  – The format is appropriate: simple structures can be defined easily
  – The format is flexible: it is Turing complete, so that all computable structures can be defined
  – We have a working implementation, ready to use within the larger framework
Future work

• Composable security properties are highly theoretical. We will attempt an integration of such existing security properties into our composition framework.

• We hope to develop new and more applicable security properties that will be integrated into both the composition and analysis aspects of the framework.

• Other areas of the framework require further work.

• The other areas must be combined with the Compose XML application to form an effective security framework.
The End

Thankyou for your time

http://www.cms.livjm.ac.uk/PUCsec/