An Object-Oriented Approach to Containing Mobile and Active Codes in Large-Scale Networks
Stephen S. Yau, Abhinav Prasad, X. Zhou
Computer Science and Engineering Department
Arizona State University
Tempe, AZ 85287-5406, USA

Abstract
An efficient method for containing mobile code and active code for the purpose of preventing security intrusion in large-scale networks is presented. This approach also provides flexibility in programming to make the networks more dependable from security point of view. It is scalable for large-scale networks, where mobile code communication is an integral part of the systems. The approach uses object-oriented components for the ease of deployment and easy use of features like selection of a specific protocol.

Keywords: mobile code, active code, code-containment, object-oriented components, large-scale networks, security intrusion prevention

1. Introduction
As the networks evolve into more complex forms, the need for containment of the mobile codes and upcoming active codes while providing necessary programming flexibility and scalability has been increasingly more important. The risk associated with downloading and executing an untrusted code is becoming increasingly important in a large-scale and complex network.

In this paper we will address the dependability of a system from this point of view. Typical applications can range from the ubiquitous Internet where Java applets and ActiveX are most common forms of mobile code. In addition, active codes, which represent recent advances in active networks [1], may also be mobile. The evolution of these large-scale networks are towards building programmability as part of the infrastructure. With this comes the need for security, need for easy and efficient programming of the network and the need for rapid and mass deployment of certain programs. This implies a need for component based system in this context of large scale distributed systems.

In this paper we will present an efficient containment approach for active as well as mobile codes, which is scalable for large-scale networks. This approach provides flexibility for programming the network to make more dependable from security point of view. Instead of restricting the security facility and the programming support to the language, we support it in middleware. To support the safe execution of the remote code we will use a method based on software fault isolation that is sandboxing [2]. To provide for the programming flexibility, an interface is provided which is used by the remote code to use the local resources. The access to resources will be defined in terms of an access expression which, if evaluates to be true, will imply a legal access to the resource. Any illegal moves will be caught and necessary action will be taken. The environment itself will consist of components specified according to the user and deployed safely on the host nodes.

2. Overview of Our Approach
Our approach is based on the software fault isolation method or the sandboxing method to provide code containment for untrusted code, which can be an active program, a mobile code or an extension code. We treat the faults as potential security hazard, which should be caught and...
contained. On the other hand, there should also be flexibility of programming and choice of programming language for the system to scale up. For this purpose, we develop an interface which has a set of methods required for the program to access the local resources or any other kinds of resources depending on the accessibility.

Of the existing popular mobile codes, Java applets are more suitable with respect to security as compared to ActiveX, which is based on authentication and runs with full user privilege and can be easily hacked [3]. The Java approach has the drawback of being inflexible and poor scalability in large-scale distributed system.

In a large-scale network, where the mobile code has the potential of corrupting the host machine or extension codes which are added to the system, which may contain malicious statements, providing security to the host and containment of such code is an important issue. Even in the current scenario there are possibilities of local user getting affected due to a malicious program. For example the famous Netscape security bug which can lead to adjustment of pointers of user stack and smash it. We address these issues by providing an approach based on sandbox [2] that is independent of language and has the benefit of easy deployment. The major advantage of this containment method is that it is simple and can be applied simultaneously to mobile code as well as extension codes or for that matter any code with low trust value. A mechanism is provided through interface methods for the active code or the mobile code to access the resources in a legal way through the use of interface calls.

The environment where the active or the mobile code executes and containment for it is provided is called secured execution environment. It is composed of an isolated segment for code and data called quarantine, which is initialized to a bit pattern every time a fresh execution begins. Any illegal moves out of this or any resource access not validated through access expression causes a trap in the execution environment. The interface methods contain methods designed with security in mind with respect to untrusted codes.

The Secured Execution Environment of our approach is shown Figure 1. It consists of two components:

![Figure 1: The Secured Execution Environment of our approach](image)

Memory isolation and access expression checker. The code, which is executing in this environment uses the interface APIs and works in isolated memory. The access expression checker causes a trap whenever an illegal access occurs.

3. Active Programs and Mobile Codes

Mobile codes represent a restricted form of active programs, which are more general and more powerful in nature as compared to the active programs. For the purpose of containing these codes, it is essential that checks be put on their access values. This means that any memory reference that does not satisfy access expression or any other access to resource not satisfying the expression value is illegal and is trapped. The definition of access expression allows a check for the severity of security violations. The mobile code or the active code that has downloaded itself at the host identifies itself through an authentication process. Then, depending on the code type, an Interface is chosen and the required execution environment created for it. The code is then modified for its containment, converted to runnable format and then executed in the
secured execution environment. Any trap occurred during the execution of code activates the required handling routine, which in many cases can be termination and flushout of the program.

Figure 2 shows the interaction of the mobile code with the secured execution environment.

![Diagram of mobile code interaction]

The mobile code uses interface APIs to interact with the local host machine resources.

4. Secured Execution Environment

This environment consists of the Interface APIs component, the memory isolation or the quarantine, the access expression checker and the secure communication module. We also plan the implementation as active code for fast deployment and many other benefits.

4.1 Interface APIs

The interface APIs provides the programming capability for the mobile code taking into consideration of the safety of the host as well as the mobile code. They can be regarded as a library where new methods can be added as they become important. These APIs will support the access to local or network resource or any other resource in a secure way. The selection of a library is based on the type of the code. These APIs also support communication between different untrusted programs or between trusted and untrusted programs.

These are public members of special classes handling I/O, memory functions, networking functions and other resource functions. They all call the methods of access expression checker to verify for its accessibility. For example, we have a skeleton networking class:

```cpp
Class Network{
    //…
    If (!Access_Expression_Checker.chkAPI(…))
        Trap(type);
    //…
};
```

4.2 The Memory Isolation

Memory isolation or the quarantine is required because the mobile or active code is not trusted and has to be dealt with separately from other codes, which are trusted. After a code has finished its execution and is no longer required to reside in the memory, the location occupied by it is initialized to some specific value to avoid any Trojan horses. To achieve this, the object code is modified to include guarded statements which check the jumps or other kinds of memory accesses, such as write to another segment which is not legal as calculated by access expression checker. Modifying object code has the advantage that it is not required to have code generated by a trusted compiler and worry about the trustworthiness of the compiler which generated the code.

![Diagram of memory isolation]

Figure 3: A scheme to achieve memory isolation
The modified code is executed in the execution unit and upon generation of any access violation the associated fault handler will be called. The scheme is shown in Figure 3.

4.3 Access Expression Checker

Access expression consists of terms which denote access to local and remote resource types, which are connected together to form a logical expression. The advantage of using this scheme is that it can be represented simply and gives a clear understanding about the access capability of a program. For example we can have for a program $p$ access expression as $\{\text{ReadAll + Write (file1, file3) + APIs (a1, a2, a6)}\}$. The expression is based on how much the user on the host trusts the mobile code or the active code.

4.4 Secure Communication Module

This component of the secured execution environment handles the secured transmission of data, various cryptographic protocols required for communication and encryption decryption schemes. The protocols can be selected based on the requirement and so are encryption decryption schemes key length and other factors determining the security needs. Establishing the proper protocol for the required need is very important, as that will have effect on the efficiency of the data transmission.

This module consists of different authentication protocols, encryption decryption methods and other cryptographic protocols.

![Secure Communication Module](image.png)

Figure 4: Secure Communication Module

5. Code Type

The mobile code type or the active code type is denoted by an integer, which represents that particular language throughout the distributed system or the network. When a new type of mobile code or active code gets supported by the system; an integer uniquely representing this language in which the mobile code was written is assigned. For example for a mobile code representing language C can be said to be of type 3 and this is known to all the nodes running this system as well the sender of the mobile code.

6. Procedure

6.1 Setup Secure Communication
6.1.1 A secured communication medium is set up between the sender of the mobile code and the host, which receives it. This is achieved by the secured communication module (Section 4.4).
6.1.2 Based on the requirement of the host and sender of mobile code, encryption decryption schemes are chosen.
6.1.3 Secured data communication over the network is established.

6.2 Setup secured execution environment
6.2.1 The secured execution environment is setup after the code is authenticated.
6.2.2 The type of code is determined.
6.2.3 APIs available for that type of code is determined.
6.2.4 Access expression depicting the memory access, file access, APIs allowed etc. setup
6.2.5 The code is then modified by adding guards (Section 7) and then loaded into a segment.

7. An Example

The sender of mobile code establishes a secure communication medium after negotiating with the Secure Communication Module at the remote host where the mobile
code will execute. Let us call the sender S and remote host where the mobile code or active code will execute as H. After the establishment of secure communication medium, S sends <Authentication Information, Code Type, Mobile Code> to the host H. This is shown below:

```
.....................
MOV a1, a2
ADD a1, a3
CMP a4, a5
READ file5, data[a5]
WRITE data[a5], file2
WRITE data[a5], file3
CALL api1
CALL api5
.....................
Code type
.....................
Authentication ID
.....................
```

If the authentication is successful the host H will be able to distinguish this code from others based on Authentication ID and code type. Code type is used to identify the type of the mobile code i.e. the language type. Hence it will generate an access expression for this code. Suppose it generates an expression:

```
{read(all)+write(file1,file2)+API(api1,api2,api3)}
```

This means that the code has read access to all files, write access to file1 and file2 and can use only api1, api2, api3. So any other type of access will be illegal for this code. Access to these resources of the host H can be negotiated by the sender of the code, which is recognized during the authentication process.

The code is then modified to prevent it from accessing anything other than the resources described by the access expression as follows:

```
//a1 to a4 in segment s1 and a5 in segment //s2
//Access to s1 only
...
```

Code to check a1 to a4 in s1
Code to modify a1 to a4 //a register is used //for indirect addresses)
MOV a1, a2
ADD a1, a3
Code to check for a5 in s2
Code to modify a5
CMP a4, a5
Code to check for file access and modify a5
READ file5, data[b5]
WRITE data[b5], file2
WRITE data[b5], file3
CALL api1
Trap
CALL api5
...
Authentication ID
...
Code type

8. Future Research

One of the research directions is the development of the API sets and a minimized set that would ensure a minimal number of reference to these APIs i.e. an API which is used once requires one reference, while another need to used twice for the same effect will need two references and will increase overhead in our case even though programmatically the later may be more efficient

We plan to implement our secured execution environment at the middleware level to avoid changing the operating system code. In this way the deployment of execution environment would be faster and easier. Moreover, it can be extended to demand loading of this environment. This will require securely sending the setup code for the execution environment over the network.

An active program, which can be called as setup program for the execution environment, sets up a secured channel and environment from the top to the bottom of the protocol layers. Security is required for each layer. We assume that the protocols, such as IPsec, implemented in the operating system are secured. The advantage of using an active program is to have easier and fast deployment. This active program which can
be downloaded from a trusted host (or a set of them) and is made out of components which are specific to security levels, protocols and other aspects.

This active program, which is encrypted and sent through a secured medium, sets up the Secured Execution Environment as showed in Figure 1. This environment then takes care of the remote application executing on the local machine with the privilege of local user. It also ensures secured transfer of data.

Through this secure channel to the local host and uses APIs from the interface for its program requirement. Other communications from this environment also go out through this secure communication module. This module can be used directly if supported by operating system or an implementation can be integrated with our system.

**Figure 5: Secured Environment System**

Figure 5 shows the complete secured environment system, which consists of five major components: the API interface, memory isolation, access expression checker, secure communication and the overall control. All the communication takes place through secure communication module. The mobile program interacts

**References**

[4] Ulana Legedza, David Wetherall and John Guttag, "Improving the performance of distributed applications using Active Networks", *IEEE Infocom* April 1998,