Defenses

System-Call based Intrusion Detection Systems (Host Based IDS)

Even if the attacker can take over the system, the attacker must make system calls to cause damage. The idea is to monitor the system calls made by the application to detect any deviation from the expected behavior.

![Syscall Monitor Diagram]

The syscall-monitor monitors each system call that the application makes. If the syscall looks normal, then it is allowed, otherwise the application is terminated. Do define “normal” behavior, the monitor requires a model which describes the normal behavior of the application.

Model types

For e.g. for the cp utility we can have the following model –

1. A set of system calls –
   
   Set \( M = \{ \text{open}, \text{read}, \text{write}, \text{close} \} \)
The monitor checks against the set that each syscall made during execution is in M. This approach ignores the following points –

   a. It could be that what syscall is allowed at a given moment, won’t be allowed later. And vice versa.

   b. The parameters of the syscalls. (fp of the read syscall has to be the same as the open syscall)

2. **A Finite State Automata** –

   The monitor keeps track of which state the application is in and which system calls are valid from that state. The following figures show the NFA for the cp utility. From the NFA, the DFA can be easily constructed by removing the epsilon transition.
How to construct models?

a. Manually

b. Trace based –

In this method, the application is run in a safe environment where it can’t be attacked. The traces of the system calls are collected. But since we can’t state with surety that one particular trace will always happen, we have to generalize the traces.

i. Construct n-gram model.

The n-gram model is a finite state machine which will remember the last (n-1) system calls made by the application.

For example, consider the following traces --
oorwrwrwcc
oorwrwcc
oorwcc
[o – open, r – read ... and so on]

A 3-gram model would look as follows –

![3-gram model for the traces](image)

Issues –

1. What should ‘n’ be? From empirical evidence, it is seen that ‘n’ should be 6.
2. How much trace is needed?

If \( n \) is too large, and the app isn’t executed enough number of times during model construction, then all states are not covered. This can lead to false alarms.

On the other hand if the number of states is too small, then the model loosely fits the application. This makes the application vulnerable to mimicry attacks, in which the attacker follows the model. For e.g. the attacker can do a buffer overflow attack to transfer control to the bad code. And once the bad code is running, it can imitate the syscall pattern of the model so that the monitor doesn’t detect it.

c. Static Model Construction

In Static Model Construction, the model is constructed by analyzing the source code or the binary. This involves constructing a Control Flow Automata on the basis of the syscalls in the application. In a Control Flow Automata, each statement (syscall) in the program is a transition, and the states are the points after each statement.

Consider the following programs and their corresponding CFA’s.

E.g.

```c
main()
{
    open();
    open();
    do {
        read();
        write();
    } while(...);
    close();
    close();
```
Similarly, the CFAs for programs with if-else constructs and function calls can also be constructed.