IDS system
keeping track of the system calls made by a monitored process and checking for abnormalities in the sequences of system calls made. When an attacker breaks into a process (through a buffer-overflow or some other means), he or she will need to make system calls in order to attempt to access the resources of the system that are under attack. As the system calls that the attacker will perform will likely be different than those performed by a process that is not under attack, it follows that by monitoring both healthy and broken processes, we can develop a scheme to identify those that might be under attack for further action to be taken.

Consider a process that makes the following system calls in the following order (where O = open, R = read, M = mmap, W = write, and C = close):
O R M M W R C
If we were to examine this stream of system calls with a window size of 5, we would see the following three sequences:
O R M M W
R M M W R
M M W R C

Userland

Userland Design overall views

1. User call Trace() function to start recording all the system call function been called when running.
    a. Trace() function is similar to system call ptrace function.
2. Call system function getList() to get the first list of correct order
3. Run another set of system call using trace() to record the list of system call in order
4. Compare the two list to see how many system calls is in right order
    a. For example, if system call function list A B C D E F G
    b. And another list is A B C D A D E
    c. Then the out is 1111000
    d. Just what is define about
5. Display if the system is being attack or not

Userland Function

Global variable:
List 1 // correct list
List2 // other list

function
// create two list
Void Createlist()

// call the system function trace() to trace all the system call function been trace
Void Run()

// call the system function getList() to get the list of system call function
Void getList()

// compare the two list to see how many system call function is in order
// list 1 is correct of function in correct order
// list 2 is other list of that compare to list 1
Void Cpmpare(int list1, int list2)

// display the if the system is being attack or not
Void display()

Kernel Space Design

Kernel Design overall view:

1. Kernel will trace which system call functions is been run after running the applicant
    - Insert function might be call with in trace function so user don't have to call it twice.
2. Each system call functions been called is insert into a system call list
    - There is no limited of how many system call function names will be store in
3. User space will have access to system call functions list such as (getList)
4. User space will also have access to system call function get count if needed

Kernel variable and Function:
All the following function will be added to the kernel header file.

Global variable:
List // content all the function call in order

Functions

```
// it will start trace the system call function been run
Void Trace();

// insert the function name into the list
// it take in the system call function name trace by trace function
// user space might not have access to the insert() function if insert() function is call within
// trace() function
Void Insert(str functionName)

// return the system call function name list
Int getList()

// return the number of system call been called
Int getCount()

// delete all the function name in the list and the list
Void Delete()
```