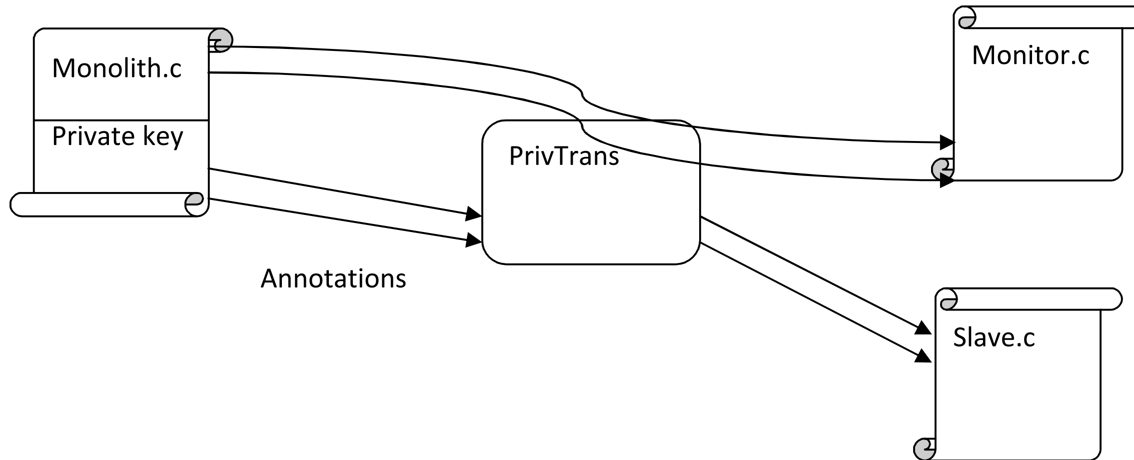
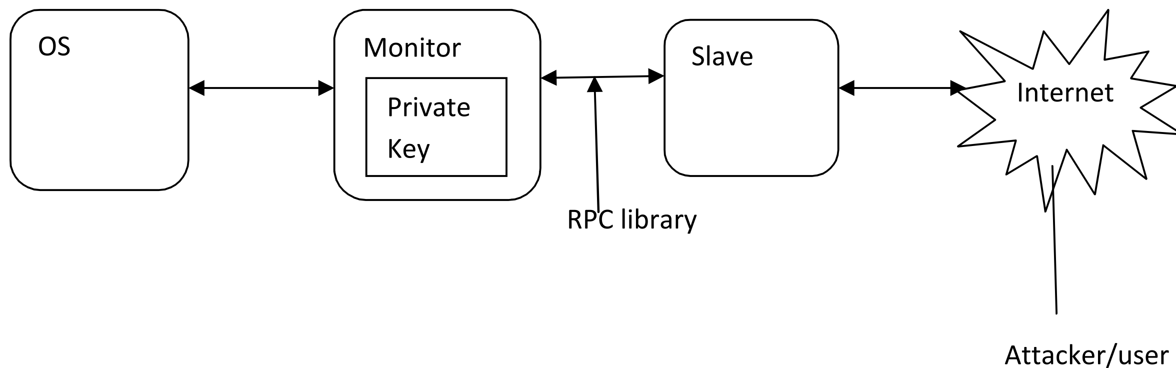
PrivTrans: Automatic Privilege Separator



Annotations:

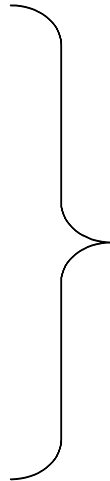
- Prescribe privileged operations
- E.g. Setuid() , chroot(),open(),bind() to privileged parts
- Some data items also have sensitive access data ; e.g. File handles, private keys
- Certain operations can be called by slave; RPC library needs to be defined for interaction between slave and monitor.



- If monitor exports function to slave for ex. SSH {keyxchange (), checkpass ()}, then policies need to be defined by monitor.
- Consider program :

Main.c

```
Rootmoduli ();
Keyxchange ();
Checkuser ();
If (pubkey mode) {
            Pubkeycheck ();
Syscheck ()
                  }
Else {
      Checkpass ();
     }
```

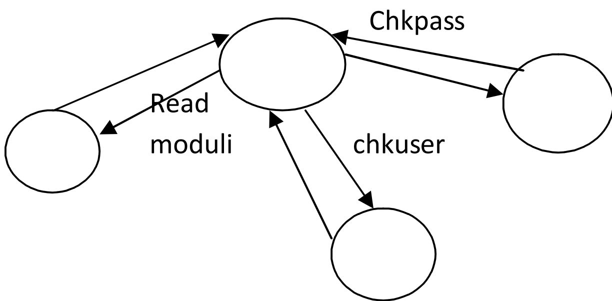


CFG => FSM=>remove un privileged calls

Model

Program:

```
While (1)
{
      Read pkt ();
     Switch (pkt.opcode)
     Case readmoduli:
            If (state == preauth0)
              Read moduli ();
            Break;
     Case……..
```



Policy is too loose as it depends on the state not in The control flow graph of program

File Descriptors in slave and monitor

-Suppose if File descriptor FD1 is in monitor and FD2 in slave

-prettyprint (int fd, char *b)

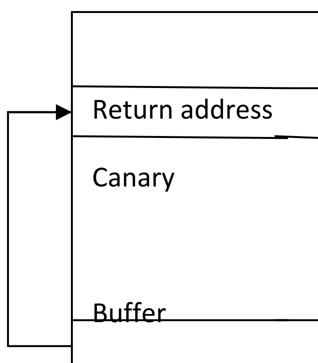
```
 {
     Write (fd, b, strlen (b));
}
```

- Slave can call prettyprint() through FD2 and monitor through FD1
- Pubfd -→slave

- Privfd→monitor
- Solution I : have two copies of prettyprint, one in each FD
- If there can be just one copy , prettyprint() should be made polymorphic and flag added
  Prettyprint (pubfd,"Yo!"); → slave FD
  Prettyprint (privfd,'Bye"); →Monitor
- Polymorphic:
  Prettyprint (int fd, int fd is_priv, char *b)
  {
    If (fd==is_priv)
  RPC (write (fd, 1, strlen (b)) ;
  }}
  Prettyprint (pubfd, 0,"Yo!");
  Prettyprint (privfd, 1,"Bye!");
- **Slave**
  Int pk1;   //handle to private key not private key itself
  Int pk2;
  RPC (sign (pk1, data))
- **Monitor.c**
  RSAkey pk1;   //Monitor has values ; looks up for mapping in the table
  RSAkey pk2;
- Slave keeps account of data item being int/handle
- If attacker takes over slave; no control to the monitor gained as only handles to file descriptors are present in slave.

Policy less vulnerable to static analysis, trace of all operations with different inputs can be observed but all paths might not be considered so analysis of trace is not effective.

BUFFER OVERFLOW
STACK GUARD



Gets (buf); →overflow→overwrites return address

Canary → value changed, program aborts before value of return address changes
Void foo ()

```
{
    Int canary;
    Char buf [64];
    Canary=canaries [12];
     Gets (buf);
    If (canary! =canaries [12])
        Abort ();
Return;
}
```

Possible assignments of canary:

- If canary is given constant value for ex canary = 0x1234, if attacker knows the value; would result in the attack.
- If canary is a global array ,attacker would guess value by brute force

Gcc compiler was modified to insert canary in all functions.

Advantages:

- **Good compatibility**
- **No work required on programmer's behalf**
  Negative: Overhead

Randomness of value of canary is an issue; leak;
Possible solutions: randomize canary at every call; define exception to check for canary

<u>POINTER GUARD</u>

NP	return addr
N P	
NP	buf

-Extra bit

-Exception when deref any NP

-handle function pointer

-no overflow of function pointer

Ret xor key
buffer

- Key : global , randomly chosen
- Pointer : xored going into memory , xored when retrieved from memory
- Int : not xored anytime
- Issue : pointer encryption , hassle