PREVENTION OF UNAUTHORISED DATA MODIFICATION SYSTEM USING SECURED MULTI KEY IMPLEMENTION

RAVI KUMAR, GOPAL PRASAD BEHERA, R.CHANDARSEKARAN

sinharavi91@yahoo.com

gopalbehera88@gmail.com

STUDENT,STUDENT,ASST.PROFESSOR

COMPUTER SCIENCE DEPARTMENT, BHARATH UNIVERSITY

CHENNAI, INDIA

Abstract- **We consider the problem of malicious modification of data in the server. We presenting a security mechanism designed to protect against unauthorized replacement or modification of data while still allowing authorized visit with nothing hiddenin the server. The key mechanism requiring any centralized public key structure. To prove our theory, we apply the approaches to file-system , implementing a prototype in Unix which protects operating system and experimental binaries on the server.**

Key words- ***Protection mechanisms, software release management and delivery, system interaction and application, access controls, file management, operating systems management.***

I. Introduction

In this paper, we reexamine the problem of how to authorize the modification of data . Instead of relying on the user to properly control updates to a saved data, we think on accessing the maker of the object to limit modifications to the object through a well-planned use of digital signatures and verification public keys[3]. Our approach does not rely on authenticating the end-user of the system on which the object to be updated resides, focusing instead on verifying that the creator of the updated object is authorized by the creator of the original object[7]. We focus exclusively on the action of replacing a digital object with a new version of that object (i.e., whether object Akþi is allowed to replace object Ak). We do not require a knowledge of who created the updated digital object, but instead ask was this individual authorized to create an updated version of this object? The approach we take is to associate with each digital object a digital signature of the object. This signature is checked by the enforcement mechanism when performing an update to the object. In essence, the object is self-signed; no centralized (or other) public key infrastructure is involved[10]. The core technology is a simple variation of self-signed executables. We use the term key-locking to refer to our proposal to avoid confusion with other schemes designed to limit the installation of digital objects based on the identity of the signing party. The proposed system allows objects to be easily upgraded, under the control of the (trusted) enforcement mechanism[1].

II. Related work

In EXISTING SYSTEM, we consider the problem of malicious modification of digital objects.

There is no authentication to view the file in server. So that the hacker can easily modify the data in the file. There is no security for the data store in the server. This will cause more problem to the client who's data are stored in the system[11].

In the PROPOSED SYSTEM, every Data Owner can access the data only after providing the public key. They change the public key after using the same key for n times. If the Data Owner wants to change their information, they've to provide their username, password and public key and private key. Also for security purpose the data will be save in the server as binary format. Also the Data Owner can set the access privileges for the users to view the data.

In the MODIFICATION PROCESS, we'll send the new public key to the Data Owner's mobile number as an SMS. Also another SMS will be send user's mobile to view the data that was send by the data owner.

III. PROPOSED SYSTEM

Paul C. Van Oorschot,Glenn Wuester

This paper presents a Unix kernel module, Digital Signature, which support administrators control practicable and Linkable Form (PLF) binary application and library loading based on the presence of a valid digital signature[13]. By foreclose attackers from regenerate libraries and sensitive, privileged system daemons with modified code, Digtal Signature adds the difficulty of conceal extracurricular activity such as access to agree systems. Digtal Signature gives administrators with an economic drive which mitigates the risk of running malicious code at run time. This dive adds unnecessary functionality previous inaccessible for the Linux operating system: kernel level RSA signature verification with caching and revocation of signatures.

We argue that application developers, while frequently seen as associate in the effort to create software with fewer security vulnerabilities, are not reliable associate[2]. They have precising skill sets which often do not include protection. Morely, we represent that it is wasteful and unrealistic to expect to be able to successfully teach all of the world's population of software developers to be protection experts. We propose more economic and impressive alternatives, concentrate on those software develop-ers who produce core functionality used by other software developers (e.g. those who develop popular APIs – Application Programming Interfaces)[12]. We handle the acquire of desig APIs which can be easily used in a secure fashion to support protection. We also developed two strang-man proposals which integrate protection into the work-flow of an software developer. Data tagging and unsuppressible information provides the basis for further work where the most natural use (path of least resistance) results in secure code. We believe there are facilities to co-opting software developers into programming properly[14].

The integrity of kernel code and data is fundamental to the integrity of the computer system. Any changing with the kernel data is an captivating place for rootkits written since despiteful modifications in the kernel are harder to identify compared to their user-level counterparts[15]. So far nevertheless, the simulate followed for changing is few to hiding captivating objects in user-space. This involves influence a subset of kernel data structures of linux that are related to intercepting user requests or affecting the user's view of the system. Hence, defense mechanisms are develop around detecting such conceal nature. The contribution of this paper is to show a new level of thiefy attacks that only available in kernel space and do not employ any hiding techniques traditionally used by rootkits[1]. These attacks are thiefy because the waste done to the system is not transparent to the user or intrusion detection systems installed on the system and are symbolic of a more systemic problem present through the kernel. Our goal in developing these contend prototypes was to show that such attacks are not only down-to-earth, but waste; they cannot be searched by the current generation of kernel integrity monitors, without more educated of the attackof the digital signature.

Now a day's architectures for intrusion shown force the IDS designer to make a difficult choice. May the IDS available on the host, it has an good view of what is going in that host's software, but is very highly susceptible to sudden fight[4]. On the other hand, if the IDS available in the network, it is more resistive to sudden fight, but has a very poor view of what is going in the host system, making it most susceptible to evasion. In this paper we present an structure that is having the visibility of a host-based IDS, but attractss the IDS outside of the host system for greater attack resistance[5]. We gain this through using of a virtual machine monitor. Using this technique allows us to isolate the IDS from the monitored host but still retain good visibility into the host's state of the developer . The VMM also offers us the special ability to completely mediate interface between the host software and the underlying hardware. We are presenting a detailed examination of our architecture, including Livewire, a prototype application. We show Livewire by applying a suite of simple intrusion detection policies and using them to detection of the real sudden attack.

We develop using digital signatures to secure binaries available on the system from changes by malware. While implementing to any file which is not intended to be changed by an end user, we focus on securinging user programs and libraries present on the system before infection will occur[2]. Our savings does not depends on a central trusted person or PKI, and can be incresingly deployed. While shown in the context of the Unix environment, our theory applies to other operating systems such as Windows like windows[3]..

Paul C. Van Oorschot,Glenn Wuester

In the **PROPOSED SYSTEM**, every Data Owner can access the data only after providing the public key. They change the public key after using the same key for n times. If the Data Owner wants to change their information, they've to provide their username, password and public key and private key. Also for security purpose the data will be save in the server as binary format. Also the Data Owner can set the access privileges for the users to view the data.

IV. FLOW CHART DIAGRAM

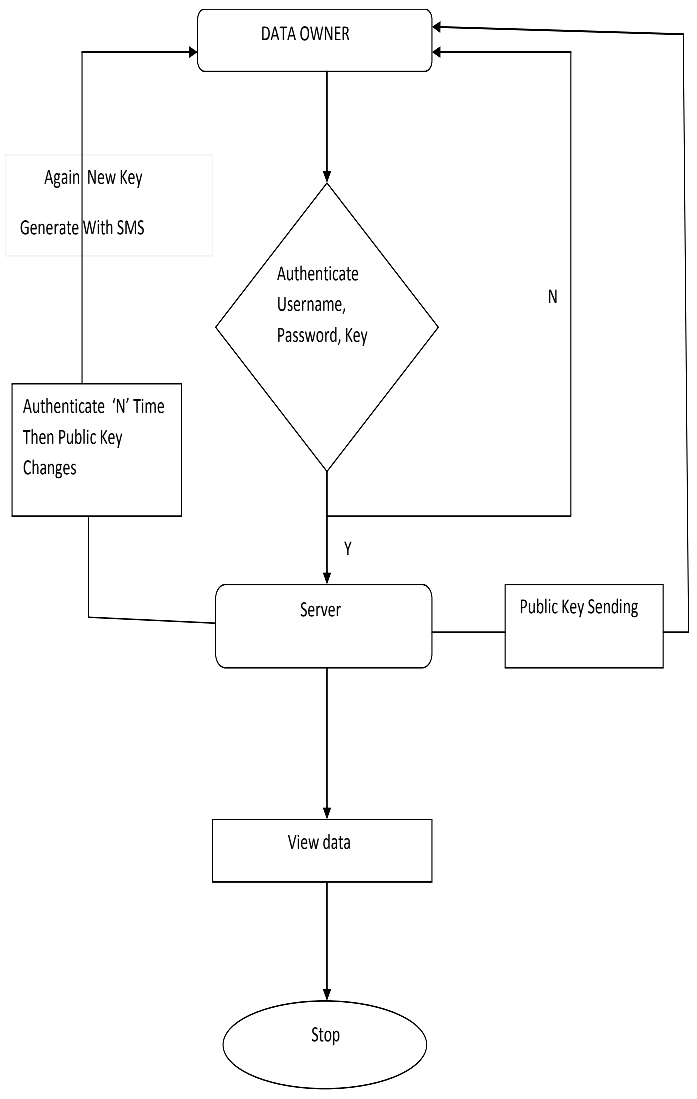

DATA OWNER
Again New Key
Generate With SMS
Authenticate Username, Password, Key
N
Authenticate 'N' Time Then Public Key Changes
Y
Server
Public Key Sending
View data
Stop

Fig-1

V. MODULE DESCRIPTION

Data Owner is the Person who is going to upload the data in the Server. To upload the data into the server, the Data Owner have be registered in the Server. Once the Data Owner registered in server, the space will be allotted to the Data Owner. So that the Data Owner can fetch the data in Server. Also the Data Owner can set the Access Privileges to the user[8].

A server is a computer program running to serve the requests of other programs, the "clients". Thus, the "server" performs some computational task on behalf of "clients". The clients either run on the same computer or connect through the network[10].

Here the Server acts as the main resource for the User and the Data Owner. Server is responsible for maintaining the Users and the Data Owners information. Server will prevent the unwanted users entering into the network. It also affirm the access permit of each and every user. The users have to be in their limits. The Data uploaded by the Data Owner will be stored in the server in the Binary format. So that it not possible to hack the data[11].

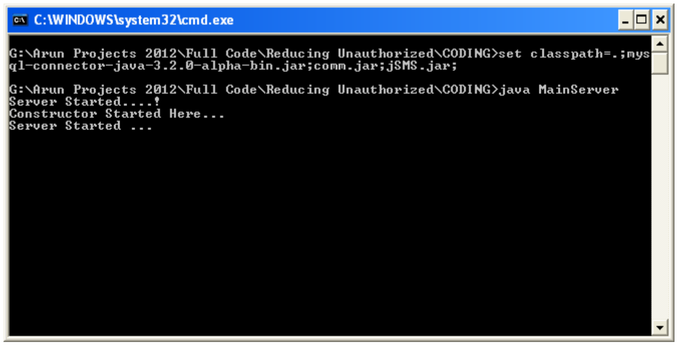
User is the person is going to see or download the data from the Server. To access the data from the Server, the users have to be registered with the Server. So that the user have to register their details like username, password. This is information will stored in the database for the future authentication. Also while registration phase, the access privileges of the users will be assigned[12].

The Data Owner will access the uploaded data using their public key[2]. While accessing the data the Server will request the Data Owner to enter the public Key. Once the public Key is valid then the Server allows the Data Owner to access the data. This will increase the level of security. After accessing the data for 'n' of times, the server will dynamically generate the new public key. So that the Data Owner have to enter that new Public Key while is accessing the data. By using the concept we can avoid the hacker from hacking the data[13].

If the data owner wants to update their public key, they have to provide their public key and private key. If these are valid the server will send the new key as an SMS alert to the Data Owner's mobile number. The mobile number will be get from the Data Owner during the registration phase itself. Once the Key are valid the server will generate the SMS. A GSM modem(Nokia PC Suite Mobile with the data cable) will be connected with the server. That GSM Modem will send the sms alert to the Concerned Data Owner[14].

This module is implemented to show how the user is going to retrieve the data from the server. Once the user requested the data, the request will hit the server and the server will respond to the request. The data will be viewed in the user's end. The is only able to view or download the data and not allowed to modify the data while viewing the in the server itself. So that we can prevent the user accessing beyond their limits[15].

VI. RESULTS



```
G:\Arun Projects 2012\Full Code\Reducing Unauthorized\CODING>set classpath=.;mys
ql-connector-java-3.2.0-alpha-bin.jar;comm.jar;jSMS.jar;

G:\Arun Projects 2012\Full Code\Reducing Unauthorized\CODING>java MainServer
Server Started....!
Constructor Started Here...
Server Started ...
```

Fig-2(Command prompt)

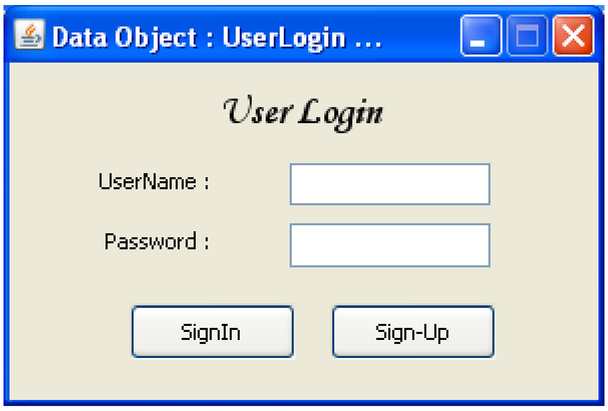
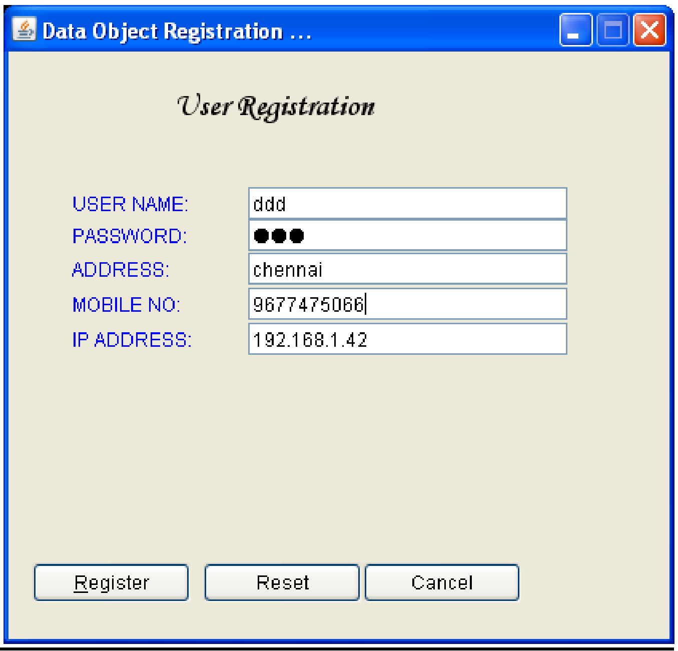


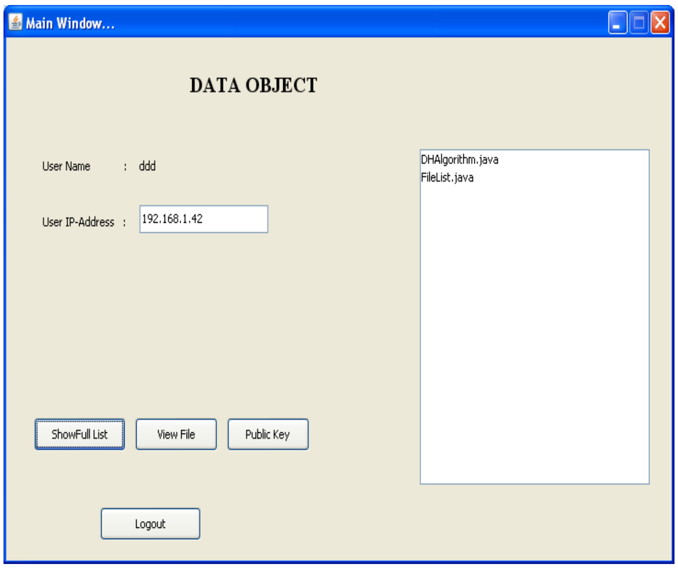
fig-3(User Login)


Data Object Registration ...

User Registration

USER NAME:	ddd
PASSWORD:	●●●
ADDRESS:	chennai
MOBILE NO:	9677475066
IP ADDRESS:	192.168.1.42

Fig-4(User Registration)


Main Window...

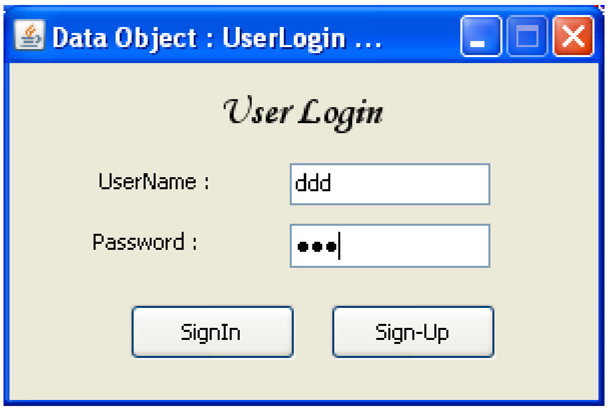
DATA OBJECT

User Name : ddd

User IP-Address : 192.168.1.42

DHAlgorithm.java
FileList.java

Fig-7(Full List)

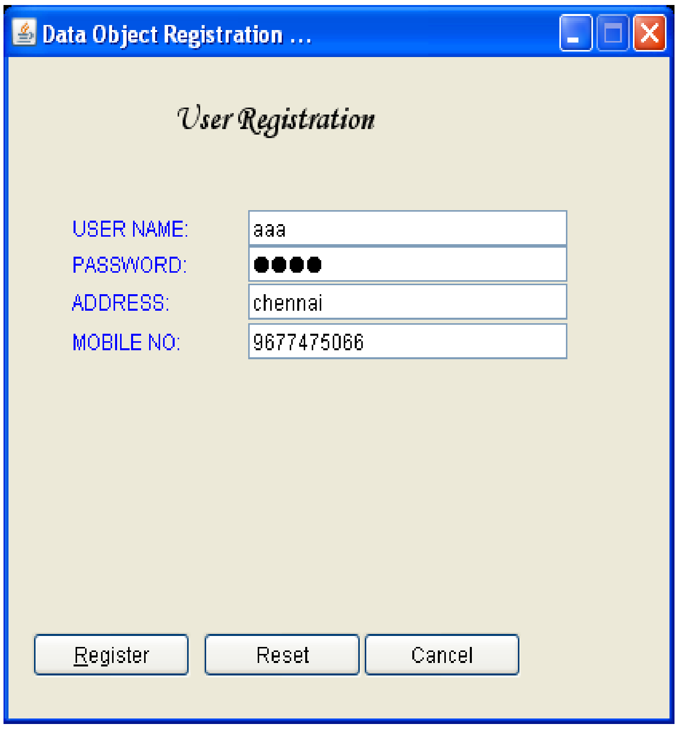

Data Object : UserLogin ...

User Login

UserName : ddd

Password : ●●●

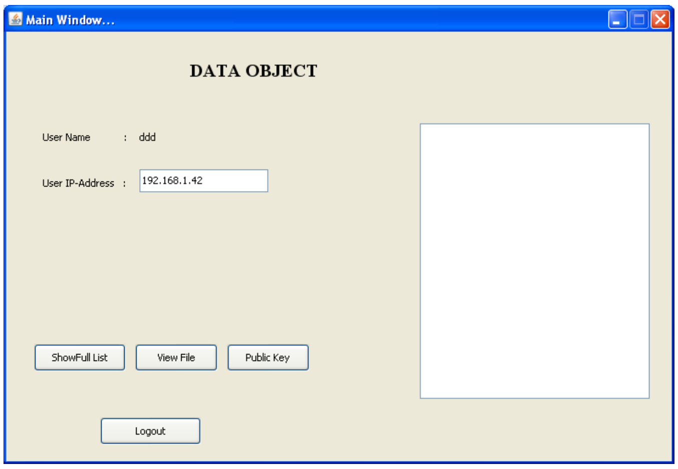
Fig-5(User Login2)


Data Object Registration ...

User Registration

USER NAME:	aaa
PASSWORD:	●●●●
ADDRESS:	chennai
MOBILE NO:	9677475066

Fig-8(User Registration)

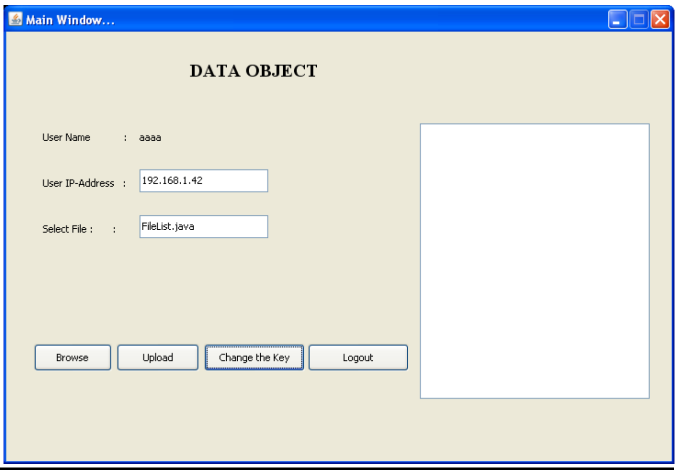

Main Window...

DATA OBJECT

User Name : ddd

User IP-Address : 192.168.1.42

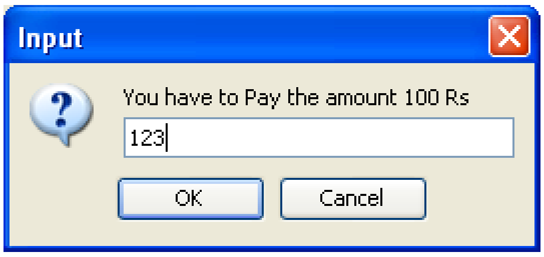
Fig-6(Main Window2)



DATA OBJECT

```
User Name        :  aaaa
User IP-Address  :  192.168.1.42
Select File :    :  FileList.java
```

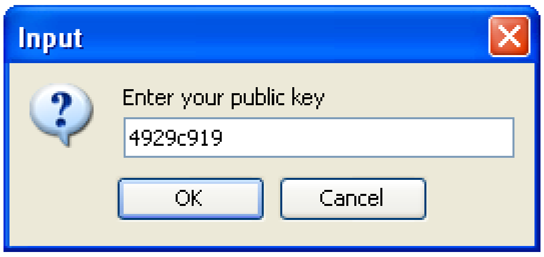
Fig-9(Main Window)



You have to Pay the amount 100 Rs

123

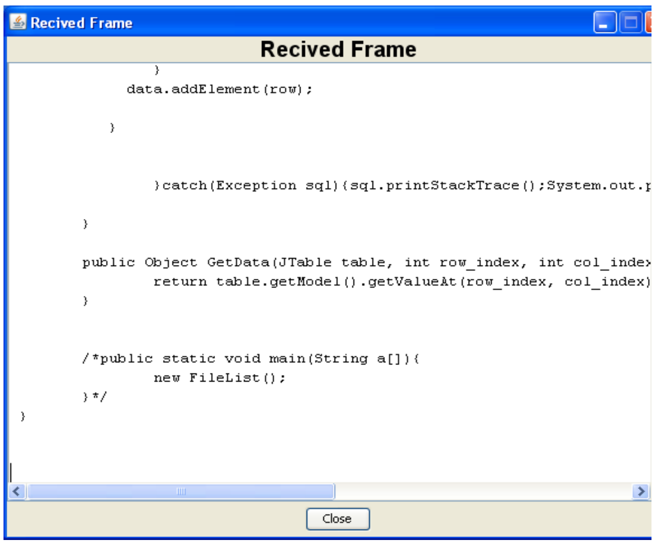
Fig-10(Input)



Enter your public key

4929c919

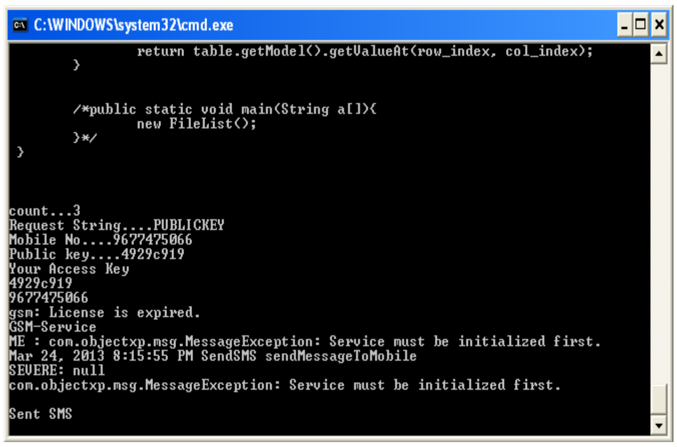
Fig-11(Output)



Recived Frame

```
                }
            data.addElement(row);

        }


                }catch(Exception sql){sql.printStackTrace();System.out.p

    }

    public Object GetData(JTable table, int row_index, int col_index
                return table.getModel().getValueAt(row_index, col_index)
    }


    /*public static void main(String a[]){
                new FileList();
    }*/
}
```

Fig-12(Received Frame)



```
                return table.getModel().getValueAt(row_index, col_index);
        }


        /*public static void main(String a[]){
                new FileList();
        }*/
 }



count...3
Request String....PUBLICKEY
Mobile No....9677475066
Public key....4929c919
Your Access Key
4929c919
9677475066
gsm: License is expired.
GSM-Service
ME : com.objectxp.msg.MessageException: Service must be initialized first.
Mar 24, 2013 8:15:55 PM SendSMS sendMessageToMobile
SEVERE: null
com.objectxp.msg.MessageException: Service must be initialized first.

Sent SMS
```

Fig-13(SMS Sending)

VII. CONCLUSION

Key-locking allows fine-grained control over entities allowed to replace a particular object. Key-locking can be used to protect against arbitrary modification of application binaries (as discussed in Section 3), and restrict package updates (as discussed in Section 5.1). Key-locking can also be used instead of usernames and passwords when pushing new versions of binary objects to a central server hosting digital objects created by users (as discussed in Section 2.7). Our application of key-locking to application binaries addresses a widespread problem: When binaries are being installed, the current (almost universal) situation is that the installer has write access to essentially the entire file-system —far too coarse a granularity from a security perspective. While bin-locking is not designed to protect all files or address all malware-related problems (indeed, a single solution to all such problems is unlikely to ever be found), we believe the prototype implementation validates the general approach and provides an important mechanism to help limit the abilities of malware. One aspect not widely addressed in the literature (to our knowledge) is the ability to transparently handle software application upgrades.

With many applications now receiving regular patches, dealing with upgrades in a smooth and nonintrusive manner is important. Key-locking provides a mechanism to enforce a separation between binary files belonging to different applications; even with privileges sufficient to install an application, binary files belonging to one application cannot be modified by an application originating from a different source.

VIII. REFERENCES

[1]http://git.kernel.org/?p=linux/kernel/git/stable/linux2.6s tabl ;a=commit;h=ae531c26c5c2a28ca1b35a75b39b3b256 850f2c8, Apr. 2008.

[2] http://www.gnu.org/software/grub/, Dec. 2008.

[3] http://www.bitsum.com/aboutwfp.asp, May 2007.

[4] Linux Operating System," FREENIX '01:

Proc. USENIX

[5]Enforcement (DTE)," Proc. Sixth USENIX Security Symp.,

July 1996.

[6]http://people.csail.mit.edu/rivest/sdsi11.html,

Oct. 1996.

[7] http://www.isi.edu/in-notes/rfc2692.txt, Sept. 1999.

[8] Digital signature principle-wikipedia.com
[9]DigSig-run time authentication at kernel level
[10]ww.usenix.org/event/lisa04/tech/full_papers/apvrille/apvrille.pdf
[11]key locking mechanism www.newmantools.com/key.htm
[12]www.usenix.org/event/hotsec07/tech/full_papers/.../wurster.pdf
[13]systematic threat www.cs.rutgers.edu/~iftode/**SystemicThreats**07.pdf
[14] G Marsaglia - See

http://stat.fsu.edu/pub/diehard, 1996.
[15] International Conference on Dependable Systems and Networks(DSN'05)..

[16] Paul C. Van Oorschot,Glenn Wuester- Reducing unauthorised modification of digital objects