Security Management Methods in Object-Oriented Database

Dejan Chandra Gope dejan.gope23@gmail.com

Dr. Md. Nasim Akhtar drnasim@duet.ac.bd

Department of Computer Science and Engineering
Dhaka University of Engineering and Technology (DUET)
Gazipur-1700, Dhaka, Bangladesh.

ABSTRACT: Security for object-oriented databases follows the traditional lines of discretionary access control, mandatory access control, and multilevel secure database systems. Security and integrity can be implemented in the object-oriented database model. We propose extensions to the basic data model to incorporate security and integrity. Our secrecy/integrity mechanism is based on the idea access control in function granularity is one of the features of many object-oriented databases. In those systems, the users are granted rights to invoke composed functions instead of rights to invoke primitive operations. Although primitive operations are invoked inside composed functions, the users can invoke them only through the granted functions. This achieves access control in abstract operation level. Access control utilizing encapsulated functions, however, easily causes many "security flaws" through which malicious users can bypass the encapsulation and can abuse the primitive operations inside the functions. In this paper, we develop a technique to statically detect such security flaws. First, we design a framework to describe security requirements that should be satisfied. Then, we develop an algorithm that syntactically analyzes program code of the functions and determines whether given security requirements are satisfied or not. This algorithm is sound, that is, whenever there is a security flaw, it detects it.

Keywords: Access Control, Security Models, Secure Database, Database Security, Integrity, Data Model.

1 INTRODUCTION

Access control mechanisms of current relational database management systems are based on discretionary policies governing the accesses of a subject to data based on the subject's identity and authorization rules. Common administration policies include centralized administration, by which only some privileged subjects may grant and revoke authorizations, and ownership administration. Ownership-based administration is often provided with features for administration delegation, allowing the owner of a data object to assign other subjects the right to grant and revoke authorizations. More sophisticated administration mechanisms can be devised such as joint administration, by which several subjects are jointly responsible for authorization administration. A number of extensions have been proposed with the goal of enriching the expressive power of the authorization these database models are more complex than the relational models.

User access to a database is either an action to get some information from the database, or an action to give some information to the database in order to make it reflected by the database

state. Access control is to impose restrictions on those actions in order to meet the requirements concerned with security. In many theoretical researches on security analysis, those two types of access are represented by read and write operations for simplicity. The basis of this simplification is the fact that any information flow between the users and the database originates in those operations. In practice, however, it is often the case that security requirements cannot be expressed in terms of such simple operations. We can impose these kinds of restrictions by defining appropriate functions and by authorizing users to invoke those functions instead of authorizing them to directly execute read or write operations. Those functions read the data but return a processed data through some computation, or they write the data following the required procedure. Although primitive read or write operations are invoked inside those functions, the users can invoke them only indirectly. In other words, the functions encapsulate those primitives into some procedures. The access control in the abstract operation level by using encapsulated functions (or "methods" in the object-oriented terminology) is one of important features of many object-oriented database systems. For example, suppose a stock company has a database about all stockbrokers of the company. In this company, each stockbroker is given a budget for his stock dealing and there is a regulation that the budget of each broker should not be higher than ten times his salary. One clerk is assigned a job to periodically examine whether the budget of each broker is illegally high against this regulation, but he should not be able to know the exact amount of the salary of each broker. Then, the database administrator defines a function that reads out the salary and the budget of a broker, compares them, and returns true or false. The clerk is authorized to invoke this function but is not authorized to directly read the salary data. In this situation, however, if that clerk can know the amount of the budget of some broker, he can know a little about the salary of that broker: "his salary is at least higher (lower) than this". If that user can change the amount of the budget to any value he wants, he can infer the exact amount of the salary by repeatedly changing the budget to several values and invoking the testing function. Those are security flaws. Another example is concerned with write access. Suppose the salary of each broker is updated once a week to a new value calculated from the budget given to him last week and the profit he made last week. Then, the database administrator defines a function that reads the budget and the profit of each broker, calculates a new salary value, and writes it in. An clerk is authorized to invoke this function. In this situation, if the employee is also able to change the budget of each broker to any value he wants, and as a consequence of it, can change the new salary value to any value he wants, then he can write any value he wants as the new salary. When we use encapsulation of functions for access control, many of these types of security flaws may occur.

In this paper, we develop a technique to statically detect those flaws. We authorize users to invoke functions, and we also describe security requirements as negative authorizations like "one should not be able to infer the result of this read operation", or "one should not be able to control the argument of this write operation". We call the former capability inferability and call the latter controllability. These two capabilities effectively correspond to the abilities to directly invoke read or write operation. Because read and write operations can be considered as special cases of function invocations, we can naturally generalize the notion of inferability onto returned values of any functions and the notion of controllability onto arguments of any functions. In fact, we sometimes want to encapsulate a composed function into another function and to describe requirements in terms of the encapsulated composed function, such as "one should not be able to infer the result of this function invocation". Then, security requirements are described in the following forms: the user u should not have inferability on the returned value of the function f, or the user u should not have controllability on the argument a of the function f. Inferability on returned values of functions and controllability on arguments of functions precisely represent two kinds of user abilities in database access: the ability to get data from the database and the ability to give data to the database.

2 RELATED WORK

There are many researches on statically determining whether a user can infer sensitive information in the database especially in the context of relational database systems. Those researches focus on whether users can know the existence of some entities or can make sensitive associations between entities or values in a database, while we focus on different aspect, i.e. whether a user can compute sensitive values from supplied values. Propose frameworks to detect the possibility of user inference on sensitive values through the knowledge on semantic dependency or on the integrity constraints defined in the database. On the other hand, our mechanism deals with dependency between arguments or returned values of functions, and data in the database which are referred to in those functions. Although all researches deal with dependency, functions can represent wider range of dependency than semantic dependency or integrity constraints. In fact, our mechanism can include integrity constraints by describing each integrity constraintinthe form of a function with no argument and returning a boolean value. On the contrary, a function cannot always be described in the form of a constraint because functions may have arguments, to which users can assign several values, and returned values can be any type. Arguments can be simulated by database values that the users can freely update. In our model, the notion of controllability, which is introduced to analyze the user's write capability, is also used to examine more elaborately how the user can update the data, and to investigate how it interacts with inferability. There are many researches on access control using "views" in object-oriented databases. Although the idea of access control using views defined by functions is essentially the same concept with access control using functions, those researches do not discuss security issues. Our technique can be applied to the verification of view definitions in those systems as well. In they provide a mechanism to automatically compute security levels of computed attributes in views in the context of relational databases. Their method is, however, simply to compute the least upper bound of security levels of all data used in the computation. Therefore, their method cannot be used for our purpose: to give users not total but to invoke in the query. (The query language is defined later partialinformationonsomedatathroughsomecomputations. inthissection.) proposed techniques to analyze program Bodies of access functions are described using the function code in order to detect all flow of information. Their methods definition language defined by the following syntax: is also to compute the least upper bound of the security levels of source data, and therefore, cannot be applied for our purpose. The rest of this paper is organized as follows. Section 2 briefly explains the basic data model used as the base of the development.

3 OBJECT-ORIENTED DATABASE MODEL

We assume an object-oriented database model based on:

1. Object and object identifier. Entities in the real world are modeled as objects in the database. The system assigns each object a unique object identifier.
2. Attributes and methods. Each object encapsulates a state and a set of behaviors. The state of an object is represented by a set of attribute values. Each attribute value may be a value from a primitive class (e.g., real, integer, string, etc.), an object identifier, or a collection. (A collection can be a set or list of object identifiers.) The behavior of an object is defined by a set of methods.The methods of an object are externally visible; attributes are not. Therefore, the only way to access or manipulate an attribute in an object is to invoke one of the object's methods.
3. Messages. To invoke a method in an object? a message must be sent to the object requestmg invocation of the method. A message is an object; specifically, each message is an instance of class message.

4. Classes and instances. A class groups a collection of objects which have the same set of methods and attributes! but which may differ in the values of those attributes. Each object in such a collection is an instance of the class.
5. Class hierarchy and inheritance. Each class may inherit the methods and attributes of other classes. The resulting structure is restricted to be a directed acyclic graph.

4 SECURITY AND AUTHORIZATION

The data stored in the database need protection from unauthorized access and malicious destruction or alteration, in addition to the protection against accidental introduction of inconsistency that integrity constraints provide. In this section, we examine the ways in which data may be misused or intentionally made inconsistent. Wethen present mechanisms to guard against such occurrences.

6 SECURITY VIOLATIONS

Among the forms of malicious access are:
• Unauthorized reading of data (theft of information)
• Unauthorized modification of data
• Unauthorized destruction of data
Database security refers to protection from malicious access. Absolute protection of the database from malicious abuse is not possible, but the cost to the perpetrator can be made high enough to deter most if not all attempts to access the database without proper authority.
To protect the database, we must take security measures at several levels:
• Database system Some database-system users may be authorized to access only a limited portion of the database. Other users may be allowed to issue queries, but may be forbidden to modify the data. It is the responsibility of the database system to ensure that these authorization restrictions are not violated.
• Operating system No matter how secure the database system is, weakness in operating-system security may serve as a means of unauthorized access to the database.
• Network Since almost all database systems allow remote access through terminals or networks, software-level security within the network software is as important as physical security, both on the Internet and in private networks.
• Physical Sites with computer systems must be physically secured against armed or surreptitious entry by intruders.
• Human Users must be authorized carefully to reduce the chance of any user giving access to an intruder in exchange for a bribe or other favors.
Security at all these levels must be maintained if database security is to be ensured. A weakness at a low level of security (physical or human) allows circumvention of strict high-level (database) security measures. In the remainder of this section, we shall address security at the database-system level. Security at the physical and human levels, although important, is beyond the scope of this text. Security within the operating system is implemented at several levels, ranging from passwords for access to the system to the isolation of concurrent processes running within the system. The file system also provides some degree of protection. The bibliographical notes reference coverage of these topics in operating-system texts.
Finally, network-level security has gained widespread recognition as the Internet has evolved from an academic research platform to the basis of international electronic commerce. The bibliographic notes list textbook coverage of the basic principles of network security. We shall present our discussion of security in terms of the relational-data model, although the concepts of this chapter are equally applicable to all data models.

7 AUTHORIZATION

We may assign a user several forms of authorization on parts of the database. For example,
• Read authorization allows reading, but not modification, of data.
• Insert authorization allows insertion of new data, but not modification of existing data.
• Update authorization allows modification, but not deletion, of data.
• Delete authorization allows deletion of data.

We may assign the user all, none, or a combination of these types of authorization. In addition to these forms of authorization for access to data, we may grant a user authorization to modify the database schema:
• Index authorization allows the creation and deletion of indices.
• Resource authorization allows the creation of new relations.
• Alteration authorization allows the addition or deletion of attributes in a relation.
• Drop authorization allows the deletion of relations.

The drop and delete authorization differ in that delete authorization allows deletion of tuples only. If a user deletes all tuples of a relation, the relation still exists, but it is empty. If a relation is dropped, it no longer exists. We regulate the ability to create new relations through resource authorization. A user with
resource authorization who creates a new relation is given all privileges on that relation automatically.
Index authorization may appear unnecessary, since the creation or deletion of an index does not alter data in relations. Rather, indices are a structure for performance enhancements.However, indices also consume space, and all database modifications are required to update indices. If index authorization were granted to all users, those who performed updates would be tempted to delete indices, whereas those who issued queries would be tempted to create numerous indices. To allow the database administrator to regulate the use of system resources, it is necessary to treat index creation as a privilege. The ultimate form of authority is that given to the database administrator. The database administrator may authorize new users, restructure the database, and so on. This form of authorization is analogous to that of a superuser or operator for an operating system.

8 AUTHORIZATION AND VIEWS

A view can hide data that a user does not need to see. The ability of views to hide data serves both to simplify usage of the system and to enhance security. Views simplify system usage because they restrict the user's attention to the data of interest. Although a user may be denied direct access to a relation, that user may be allowed to access part of that relation through a view. Thus, a combination of relational-level security and view-level security limits a user's access to precisely the data that the user needs. In our banking example, consider a clerk who needs to know the names of all customers who have a loan at each branch. This clerk is not authorized to see information regarding specific loans that the customer may have. Thus, the clerk must be denied direct access to the loan relation. But, if she is to have access to the information needed, the clerk must be granted access to the view cust-loan, which consists of only the names of customers and the branches at which they have a loan. This view can be defined in SQL as follows:

```
create view cust-loan as
(select branch-name, customer-name
from borrower, loan
where borrower.loan-number = loan.loan-number)
```

Suppose that the clerk issues the following SQL query:

```
select *
from cust-loan
```

Clearly, the clerk is authorized to see the result of this query. However, when the query processor translates it into a query on the actual relations in the database, it produces a query on borrower and loan. Thus, the system must check authorization on the clerk's query before it begins query processing.

Creation of a view does not require resource authorization. A user who creates a view does not necessarily receive all privileges on that view. She receives only those privileges that provide no additional authorization beyond those that she already had. For example, a user cannot be given update authorization on a view without having update authorization on the relations used to define the view. If a user creates a view on which no authorization can be granted, the system will deny the view creation request. In our cust-loan view example, the creator of the view must have read authorization on both the borrower and loan relations.

9 THE BASIC MODEL

In this section, we explain the basic model of database on which we develop our framework for access control. Although in this paper we assume a simple data model with mutable objects and classes, the only essential point of our development is that users access data objects by invoking functions. We believe that our mechanism can be translated onto several other data models, such as the relational data model with abstract data types.

The data model is defined as follows:

scm = ({e_name : [att:t,.....,att:t]},
{(f_name(arg:t,.....,arg:t):t, body})
t= b|c_name|{t}
db= ({(c_name, {obj})},
{(u_name, {f_name})})

A schema scm is a pair of a set of class definitions and a set of function definitions. Each class definition has the form c name : [att : t;:::;att : t] , which declares that instances of the class c name, i.e. objects belonging to c name, have attributes att of type t. Objects are mutable entities, and we can read out current values of their attributes, update their attributes, pass them to functions as arguments, and store them in attributes of other objects. Each function definition is a pair of a signature of the form f name(arg : t;:::;arg : t) : t and the definition of its body. The users access the database by invoking those functions. We call them access functions. We can interpret an access function also as a "method" by regarding the first argument as the receiver. Some additional consideration that would be needed if we introduced subtyping and overloading will be explained later. t stands for types in this model. t is either a basic type b such as integer, a class name c name that is interpreted as the type of its instances, or a set type of some type. A database db is a pair of (1) a set of pairs of a class name and its extension, i.e. a set of all its instances, and (2) a set of pairs of a user name and his capability list. A capability list is a set of all access function names (or names of special functions explained below) that the user is allowed to invoke in the query. (The query language is defined later in this section.) Bodies of access functions are described using the function definition language defined by the following syntax:

e::= c|a|fb(e,.....,e)|fa(e,.....,e)|r_att(e)|W_att(e,e)

cstands for constants, astands for the arguments of the access function. fb(e,:::,e) is an invocation of a basic function fb with arguments e,:::,e. Basic functions are primitive operations on basic types, such as addition on integers. fa(e,:::,e) is an invocation of another already defined access function fa. We do not consider recursive functions. r att and w att are special functions that read or write attributes of objects. For example, r salary(x) returns the current value of the attribute salary of the object x, and w salary(x, 100) writes 100 into the attribute salary of the object x and returns a special value null. Other than those constructs, the complete version of our development includes persistent global variables, to which we can

store any object, a special function to create new objects, let construct to define a local variable, and two special functions to get inputs from the console and to output values to the console. In this paper, however, we omit them for the brevity. The users issue query using the following SQL-like query language:
select item,.....,item from A1 € C1,.....,An € Cn where condition
In this syntax, C1;:::;Cn is a class name. A1;:::;An are called from-clause variables and are bound to each combination of instances of C1;:::;Cn. item is f(v,:::,v) where f is an access function or a special function (r att or w att) and where v is either a constant of some basic type or one of A1;:::;An. If object identifiers have some printable form, such as hid:730710i, we can also use them in place of v. In this development, however, we assume object identifiers do not have any printable form. We explain the reason of this choice later in Section 3. Items in a select clause are evaluated in order from left to right. condition consists of boolean terms connected by and and or where each boolean term has the form either of "f(v,::: ,v) op v" or "f(v,:::,v) op f (v,:::,v)". op is a binary predicate for basic types, such as <= for integers. For example, if the class Person:[name:string, age:int, :::] and the access function profile(x:Person):string are defined, and a user has r name, profile, and r age in his capability list, he can issue a query;select r_name(p), profile(p) from p € Person where r age(p) >20 which returns a set of pairs of a name and a profile for all Person instances whose age are greater then 20. selectconstruct can be nested, and set valued functions (or readoperations of set valued attributes) can be used in place ofclass names in from clause. For example, suppose Personhas an attribute child:{Person}. Then, the query belowreturns a set of names of children of a person named 'John':
select (select r_name(q) from q € child(p))
from p € Person where r_name(p)= 'John'.

10 SECURITY/INTEGRITY MECHANISM

A combined secrecy/integrity mechanism can be constructed based on the notion of protected groups. We assume that the system identifies and authenticates subjects, that the system can hide the existence of any object (e.g., classes, instances, subjects, messages) from any other object, and that a method can return a value that is indistinguishable from the "object not found" return value from the system. The secrecy and integrity of a protected group of objects is based in the interface object for that group. The interface object is the only object in the group which is allowed to invoke the methods GetACI and SetACI in method message, i.e., it is registered with the system for this privilege. When the interface object receives a message from some other object, that message either contains the access control information of the originating subject, or it contains no access control information, but does contain the object identifier of the source of the message. In the first case, the access control information is known. In the second case, the interface object can obtain the access control information based on the object identifier of the source of the message. The interface object can then set all further messages in this message chain to contain the access control information of the originating subject. If the source of the message is not a valid subject, the interface object can reject the message. The interface object also has access to the access control information of the target object of the message. This access control information may be stored in the object as additional attributes or in a separate object within the protected group. Using the access control information of the subject and the target object, the interface object can use the following general outline for each of its methods:
METHOD MethodName (Target, OtherParameters)
BEGIN
IF GetACI is-null THEN
SetACI (access control information of source object)
ENDIF

```
IF GetACI compares-favorably-with Target.ACI THEN
Invoke Target.MethodNcame (OtherParameters)
ELSE
RETURN ('Object not found')
ENDIF
END
```

In this approach, secrecy is a precondition that must be satisfied before access is allowed. The implementation of the comparison operator compares- favorably-with depends on the secrecy mechanism and the type of access control information. The implementation of traditional discretionary access control is straightforward in this setting. Only one-way protected groups are required. A class auth is included in the protected group. This class is responsible for checking whether a subject si is authorized to invoke a method m. On receipt of a message, the interface object uses auth to determine whether the source of the message is authorized to invoke the given method. Both grant/give-grant and cascading revocations can be implemented by incorporating more information into auth instances. Clark-Wilson integrity can be enforced with the one-way protected group approach in the following way. Methods in the interface object are the TPs and objects in the protected group are the CDIs. Access triples are stored in class auth. This is admittedly simplistic; more work needs to be done to further develop this approach. Another way to enforce Clark-Wilson style integrity is the Generalized Framework for Access Control;this can be applied to the objectoriented data model based on protected groups.

11 CONCLUESION AND FUTURE WORK

We defined a framework for access control in the abstract operation granularity, and developed an mechanism that detects security flaws caused by functions not hiding primitive operations inside them. The most important contribution of this research is that we introduced the notions of inferability on returned values and controllability on arguments, demonstrated that they properly model the problem of security flaws, investigated their properties, and gave the formal semantics of them. We think that these notions are proper generalization of traditional read/write capability and can work as a basis for various researches on access control in the function granularity. Althoughwe also showed a static analysis algorithm which is sound and sufficiently practical, it is not necessarily the only way to avoid security flaws. In fact, the algorithm shown in this paper is quite pessimistic. More accurate analysis with more complex computation could be developed using existing techniques for program analysis. Another alternative is to develop a mechanism to dynamically detect security flaws during execution of queries. Those are future issues. Our model achieves name-dependent access control, and a kind of context-dependent control. Content-dependent control, which is control depending on the contents of the actual data, could be also integrated by introducing some existing techniques, such as classes with predicates. The function definition language we defined in this research is quite simple language. Including more language features into it, such as conditional branch, recursion, and polymorphism, is also an important issue for future researches. We also assume a rather simple data model in this development. In how various data modeling concepts, such as versions or inheritance, affect the authorization mechanisms is discussed. The integration of the techniques we show in this paper and the mechanisms proposed in those researches is also an interesting issue.

REFERENCES

Dorothy E. Denning, Selim G. Akl, Mark Heckman, Teresa F. Lunt, Matthew Morgenstern, Peter G.Neumann, and Roger R. Schell. Views for multilevel database security. IEEE

Trans. on Soft. Eng.,13(2):129–140, Feb. 1987.
Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. CACM, 20(7):504–512, Jul. 1977.
Dorothy E. Denning, Peter J. Denning, and Mayer D.Schwartz. The tracker: A threat to statistical database security. ACM TODS, 4(1):76–96, Mar. 1979.
Dorothy E. Denning. A lattice model of secure information flow. CACM, 19(5):May 1976.
David Dobkin, Anita K. Jones, and Richard J. Lipton. Secure databases: Protection against user influence. ACM TODS, 4(1):97–106,Mar. 1979.
Eduardo B. Fern´andez, Rita C. Summers, and Christopher Wood. Database Security and Integrity,chapter 5,pages 55–64. The Systems Programming Series. Addison-Wesley, 1981.
Nurith Gal-Oz, Ehud Gudes, and Eudardo B. Fern´andez. A model of methods authorization in object-oriented databases. In Proc. of VLDB, pages 52–61, Aug. 1993.
Thomas Hinke and Harry S. Delugach. Aerie: An inference modeling and detection approach for databases. In Database Security VI: Status and Prospects, pages 179–194. IFIP WG 11.3, Aug. 1992.
Sandra Heiler and Stanley B. Zdonik. Object views: Extending the vision. In Proc. of IEEE ICDE, pages 86–93, Feb. 1990.
Sushil Jajodia and Ravi Sandhu. Toward a multilevel secure relational data model. In Proc. of ACM SIGMOD, pages 50–59,May 1991.
John B. Kam and Jeffrey D. Ullman. A model of statistiacl database and their security. ACM TODS, 2(1):1–10, Mar. 1977.
Rafiul Ahad, James Davis, and Stefan Gower. Supporting access control in an object-oriented database language. In Proc. of EDBT, volume 580 of LNCS, pages 184–200. Springer-Verlag, Mar. 1992.
Leland L. Beck. A security mechanism for statistical databases. ACM TODS, 5(3):316–338, Sep. 1980.
Elisa Bertino. Data hiding and security in objectoriented databases. In Proc. of IEEE ICDE, pages 338–347, Feb. 1992.
Rae K. Burns. Referential secrecy. In Proc. of IEEE Symp. on Research in Security and Privacy,pages 133–142, 1990.
Francis Y. Chin. Security in statistical databases for queries with small counts. ACM TODS, 3(1):92–104, Mar. 1978.
FrancisY. Chin andGultekin Ozsoyoglu.Auditing and inference control in statistical database. IEEE Trans. on Soft. Eng., 8(6):574–582, Nov. 1982.
Teresa F. Lunt, Dorothy E. Denning, Roger R. Schell, Mark Heckman, and William R. Shockley. The SeaView security model. IEEE Trans. on Soft. Eng, Jun. 1990.
Catherine Meadows and Sushil Jajodia. Ingetrity versus security in multi-level secure databases. In xSeaView security model. IEEE Trans. on Soft. Eng., Jun. 1990.
Matthew Morgenstern. Security and inference in multilevel database and knowlege-base systems. In Proc.of ACM SIGMOD, pages 357–371,Dec. 1987.
Subhasish Mazumdar, David Stemple, and Tim Sheard. Resolving the tension between integrity and security using a theorem prover. In Proc. of ACM SIGMOD, Sep. 1988.
Atsushi Ohori and Keishi Tajima. A polymorphic calculus for views and object sharing. In Proc. of ACM PODS, pages 255–266,May 1994.
Xaiolei Qian, Mark E. Stickel, Peter D. Karp, Teresa F. Lunt, and Thomas D. Garvey. In Detection and elimination of inference channels in multilevel relational database systems. Proc. of IEEE Symp. on Research in Security and Privacy, 1993.
Fausto Rabitti, Elisa Bertino, Won Kim, and Darrell Woelk. A model of authorization for next-generation database systems. ACM TODS, 16(1):88–131, Mar. 1991.
Neil C. Rowe. Inference-security analysis using resolution theorem-proving. In Proc. of IEEE ICDE, pages 410–416, Feb. 1989.

Dejan Chandra Gope is a student in the Department of Computer Science and Engineering at the University of Dhaka University of Engineering and Technology, Bangladesh. He received Bachelor of Science in Computer Science and Engineering (B.Sc) from the University of Dhaka University of Engineering and Technology in 2012. He is now studying Master of Science in Computer Science and Engineering (M.Sc) at the same University. His research interest lie in studying the role of computer vision and advanced human-computer interaction, Artificial Intelligence and Pattern Recognition.