To control access to data, the hierarchy of security levels is required. In the data secure language, the security levels are designed as four levels of security according to the security importance of variables. Table 1 shows the specification of the security levels designed in the data secure language.

Table 1. The hierarchy of security levels of data.

Security Level	Description
Public	The security level of public data that can be accessed freely regardless of security levels. Data is initialized Public if there is no security level assignment.
S3	The basic security level for secure data.
S2	The second security level of data that is higher level than S3.
S1	The top security level of data that should be kept securely from other levels.

The Public level is the class for variables of commonly used data. So, the data of the Public level can be freely accessed regardless of the security levels of users. Data that security levels are not assigned by the `setSecurityLevel()` command are initialized to be the Public security levels. The `setSecurityLevel()` command allows managers to set up important data that should be protected from exposure to unknown users. According to the security levels of data, access to sensitive data can be blocked by the security level management algorithm. The levels of secure data have a hierarchy according to the importance of data. The S1 level is the top security level for data that can be accessed only by users or systems with the S1 security level. On the other hand, The S3 level is the base security level for data that should be protected from public users. Between secure data, there is a hierarchy between levels, and users with lower security levels cannot access data with the higher security levels.

3.2. *The Structure of Security Level Management*

The security level management system consists of a data flow analyzer, data processor, expression analyzer, and security manager with access control list (ACL) [2, 9]. Figure 2 shows the structure of the security level management system for the data secure language designed in Section 3.1.

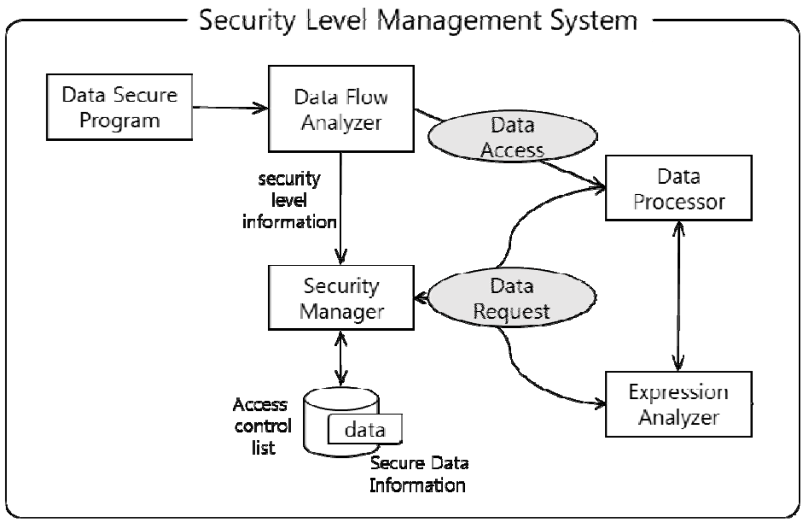

Fig. 2. The structure of security level management system for controlling accesses to secure data.

To ensure secure access to data used in a data secure program, the security levels of data are analyzed in the program. The data flow analyzer is for analyzing data that is propagated to the other destination. Because data have their security levels depending on the importance of the data, it is necessary to manage the movement of data by changing the security levels as it moves. If the analyzer does not modify the security levels as the data moves, the secure data may be leaked by the moved secure information. The access control list is a data structure that manages the value and security levels of variables in a data secure program to control accesses to secure data. This data structure manages the security levels and values of data defined in a data secure program.

The data processor and expression analyzer analyze the statements and expressions in a data secure program. They analyze and check through the security manager whether access to secure data is valid or not according to the security levels. They block and warn users if they find an access request that is incorrect access to secure data. The security manager controls the hierarchy of security levels through the access control list so that secure data is protected from being exposed to public users. If invalid access is requested, the security manager notifies the data processor or expression analyzer and supports to block the access to secure data.

3.3. *Design of Security Level Management Algorithm*

This section describes the design of security level management algorithms for the data secure language designed in section 3.1. Figures 3 and 4 show the algorithms for the data processor and expression analyzer described in Figure 2. The Data_Process algorithm analyzes the statements of data secure programs and identifies the flows of data to manage the security levels. It analyzes whether secure data are accessed correctly according to the security levels. If any security violation is found, it warns users and blocks access to secure data. The Exp_Analyze algorithm evaluates expressions and manages security levels of data. The algorithm returns the security levels of expression by analyzing the expression through the access control list.

```
Algorithm [Data_Process]
                    : processes statements in a data secure program
   stmt = getNextStatement( )
   while stmt != NULL do
      switch (stmt)
         case var = exp :                    // assignment statement
            level = Exp_Analyze(exp)
            updateACL (var, level)
         case if ( exp ) then stmt1 else stmt2 :     // if statement
            val = evaluate the value of exp
            if val == True then
               analyze and process stmt1
            else
               analyze and process stmt2
         case while ( exp ) do stmt1 :            // while statement
            val = evaluate the value of exp
            if val == True then
               analyze and process stmt1 and continue loop
            else
               continue
         case output (var, dst) :              // output statement
            level = getSecurityLevel(var)
            level_dest = Exp_Analyze(dst)
            if checkSecurityLevel(level, level_dest) == False then
               display warning and block the access to the data
         case setSecurityLevel(var, sl) :
                                          // security level management
            updateACL (var, sl)
         case changeSecurityLevel(var, sl) :
            level = getSecurityLevel(var)
            if checkSecurityLevel(level, sl) == False then
               display warning and block changing the access level
            else
               updateACL (var, sl)
      stmt = getNextStatement( )
```

Fig. 3. The Data_Process algorithm for processing data in statements in data secure programs.

The Data_Process algorithm analyzes the statements and manages the access control list up to date. For expressions in a statement, the algorithm calls the Exp_Analyze algorithm to find the security level of the expression. If the value of data is changed, the security level of the data is also updated through `updataACL( )`. The `output` command sends data to the destination `dst`. So, the security levels of the data and destination should be identified to keep secure data from being sent to an invalid destination. The algorithm identifies the security levels of the output data and destination `dst`, and checks the validity of the `output` command according to the security levels. If the `output` command sends secure data to an irrelevant destination, it displays a warning and blocks sending the data to the destination. In the case of `setSecurityLevel` and `changeSecurityLevel` commands, the security levels of data are initialized or updated after checking the validity of security levels. If the command tries to change the security level to a lower level than the previous level, the algorithm will display a warning and block the operation.

```
Algorithm [Exp_Analyze (exp)]
                  : analyzes expression and returns security level
   switch (exp)
      case constant or True or False :
         return Public
      case var :
         level = getSecurityLevel(var)
         return level
      case exp1 ± exp2 :
         level1 = Exp_Analyze (exp1)
         level2 = Exp_Analyze (exp2)
         if level1 > level2 then
             return level1
         else
             return level2
      case - exp1 :
         level = Exp_Analyze (exp1)
         return level
      case input (data, src) :
         level = Exp_Analyze (src)
         updateACL(data, level)
         return level

Algorithm [checkSecurityLevel(level1, level2)]
                                       : controls accesses to data
         if level1 >= level2 then
             return True
         else
             display warning
             return False
```

Fig. 4. The Exp_Analyze algorithm for evaluating expressions and returning the security levels of expressions.

The Exp_Analyze algorithm returns the security level of expression. If the expression is a constant or a logical value, the algorithm initializes the security level and returns the Public level. If the expression is a variable, the algorithm identifies the security level of the variable through the access control list and returns the security level. If the expression is a binary operation, the algorithm identifies the security levels of two expressions and returns the higher security levels of the two expressions. If the expression is a unary operation, the algorithm identifies the security level of the expression and returns the results. If the expression is an `input` command, the algorithm identifies the security level of the input data and updates the security level of the data to the access control list. The checkSecurityLevel algorithm manages the access control rules for secure data. The algorithm takes two security levels and ensures that the first data has access to the second data according to the two security levels. If the access is allowed, the algorithm returns True. Otherwise, the algorithm will display a warning and returns False.

The security level management algorithm provides a mechanism for protecting secure data from being exposed to public users according to the security levels in a data secure program. Besides, the algorithm analyzes the movement of data and security levels in a program and manages the security levels of data update according to the data change. The security level management algorithm can protect secure information in a data secure program from being exposed to unknown destinations according to the security levels.

4. Experimental Evaluation

4.1. *A Data Secure Program*

In this section, the security level management algorithm for the data secure language is implemented and experimented with a data secure program to validate the effectiveness of the proposed method.

```
1  SecureProgram {
2     dataP = 25;
3     dataS1 = input(100, S1);
4     dataS2 = input(0, S2);
5     dataS3 = input(300, S);
6     setSecurityLevel(dataS1, S1);
7     setSecurityLevel(dataS2, S2);
8     setSecurityLevel(dataS3, S3);
9     index = 0;
10    number = 5;
11    while (index <= number) {
12       dataS2 = dataS2 + index;
13       output(dataS2, dataS3)
14       index = index + 1;
15    }
16    output(dataP, dataS2)
17    dataP = dataS1
18    output(dataP, dataS2);
19 }
```

Fig. 5. An example of data secure program for evaluating the security level management algorithm.

Figure 5 shows an example program of the data security language. In the program, one public data `dataP` and three secure data are initialized from lines 2 to 8. In lines 9 and 10, two variables `index` and `number` are initialized for while loop. The while loop in line 11 is repeated 6 times according to the two variables, `index` and `number`. In the while loop, there is an operation to output secure data to a destination of the lower security level in line 13. So, the operation should be blocked to protect secure data from being exposed to a destination with low security level. After finishing the repetition of the while loop there are two identical `output` commands. However, there is an assignment statement that moves secure information `dataS1` to `dataP` in line 17. After the assignment statement, the `dataP` may have information on secure data `dataS1`. So, the `output` command in line 18 should be blocked securely without being exposed to data with the lower security levels than S1. Through security level management, this program can initialize the security levels of the variables to ensure the secure data can be kept safely.

4.2. *Experimental Results*

The security level management algorithm proposed in this paper has been implemented and it is evaluated with the data secure program described in section 4.1. The algorithm is implemented in the functional programming language Haskell [10] on the Microsoft Windows 7 operating system. The Haskell supports advanced functions for manipulating strings, so it provides an effective programming environment in the design of programming languages and analyzers for programs.

```
--[Initial Security Level Initialization]--

("dataP",(ID 25,Public))
("dataS1",(ID 100,S1))
("dataS2",(ID 0,S2))
("dataS3",(ID 300,S3))
("index",(ID 0,Public))
("number",(ID 5,Public))

  --[ Security Level Information after Execution]--

("dataP",(ID 100,S1))
("index",(ID 6,Public))
("dataS2",(ID 15,S2))
("number",(ID 5,Public))
("dataS3",(ID 300,S3))
("dataS1",(ID 100,S1))

  --[Unauthorized Data Access Warnings]--

[Warning] Access request to Secure data.
  From: Name: [dataS3]   id: [300]   SL: S3
  To: Name: [dataS2]   id: [0]   SL: S2
[Warning] Access request to Secure data.
  From: Name: [dataS3]   id: [300]   SL: S3
  To: Name: [dataS2]   id: [1]   SL: S2
[Warning] Access request to Secure data.
  From: Name: [dataS3]   id: [300]   SL: S3
  To: Name: [dataS2]   id: [3]   SL: S2
[Warning] Access request to Secure data.
  From: Name: [dataS3]   id: [300]   SL: S3
  To: Name: [dataS2]   id: [6]   SL: S2
[Warning] Access request to Secure data.
  From: Name: [dataS3]   id: [300]   SL: S3
  To: Name: [dataS2]   id: [10]   SL: S2
[Warning] Access request to Secure data.
  From: Name: [dataS3]   id: [300]   SL: S3
  To: Name: [dataS2]   id: [15]   SL: S2
[Warning] Access request to Secure data.
  From: Name: [dataS2]   id: [15]   SL: S2
  To: Name: [dataP]   id: [100]   SL: S1
```

Fig. 6. The experimental results of security level management algorithm applied to the data secure program in section 4.1.

Figure 6 shows the experimental results of the security level management system for the data secure program in Figure 5. The result shows the initial assignments of security levels of variables, final security levels of variables after executing the program, and warnings for insecure access operations that are not allowed to access secure data according to the security levels. It is confirmed that the initial security levels follow the initial definition of the variables in the benchmark program. The final security level of `dataP` is different from the initial level because the information in secure data `dataS1` is moved to `dataP` in line 17 of the program. So, to protect the information in `dataP` moved from the secure data `dataS1`, the security level of `dataP` is promoted from Public to S1. In the program, the `output` command in line 13 tries to output `dataS2` of security level S2 to the `dataS3` which has the security level S3. So, data of security level S2 should be protected from being exposed to the `dataS3`. Because the `output` command in line 13 is executed 6 times in the while loop, there are 6 warnings for preventing the `dataS2` from being exposed to the `dataS3`. The final warning is for the `output` command in line 18 of the program. The two `output` commands in lines 16 and 18 are identical, but there is a data movement between the two `output` commands. Before the data movement in line 17, the `dataP` has a Public security level. So, the `output` command in line 16 is not an access violation according to the security levels between `dataP` and `dataS2`. After executing the assignment in line 17, the `dataP` may have information on `dataS1` with security level S1. So, the `output` command in line 18 should be blocked to protect the information with security level S1 from being exposed to security level S2. The final warning is for the `output` command in line 18.

From the evaluation results, we have shown that the security level management algorithm can effectively protect the secure data from being exposed to data with low security levels. Besides, information moved from secure data is successfully protected from being exposed to others by maintaining the security levels of moved data. Because information security is important in information systems, the technology for protecting secure data is needed. In this paper, we design a security level management algorithm for enhancing data security of systems so that the systems can keep and protect secure data according to the security levels of data. The proposed approach is expected to be applied in systems to enhance data security by managing secure data without being exposed.

5. Conclusion

In recent information environments, a vast amount of information is produced and used in information systems. The secure information should be kept safely without being exposed to unknown destinations. This paper presents the security level management algorithm for preserving data safely according to the security levels of data. We design a data secure language for describing secure data by assigning security levels to the data. We define the hierarchy of security levels to manage secure data according to the importance of data. For the data secure programs, we design the security level management algorithm to protect secure data not to be accessed from insecure data. The algorithm identifies the accesses to secure data, and it blocks accesses to secure data from low security levels. The algorithm can also manage the security levels of moved data. So, when secure data is propagated to data with low security levels, the secure data are successfully protected by changing the security levels of moved data. The experimental results show that the presented algorithm can effectively block accesses to secure data in the data secure program according to security levels.

In recent information systems, the number of data is growing exponentially, and data management is becoming an essential technology for maintaining the system. The approach proposed in this paper can be effectively applied to manage secure data without being exposed to unknown destinations. It is expected that the proposed approach can be used in the design and implementation of secure systems that require secure data management.

Acknowledgments

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(Ministry of Education) (No. NRF-2017R1D1A1B03034769).

References

[1] Romuald Thion, Access Control Models, Chapter 37, Cyber Warfare and Cyber Terrorism, 2007.
[2] John Barkley, Comparing simple role based access control models and access control lists, The second ACM workshop on Role-based access control, pages 127-132, 1997.
[3] Chang N. Zhang, Cungang Yang, Information flow analysis on role-based access control model, Information Management & Computer Security, Vol. 10 Issue: 5, pp.225-236, 2002.
[4] Srdjan Marinovic, Robert Craven, Jiefei Ma, Naranker Dulay, Rumpole: A flexible break-glass access control model. The 16th ACM Symposium on Access Control Models and Technologies. pages 73-82. 2011.
[5] Luis Cruz-Piris, Diego Rivera, Ivan Marsa-Maestre, Enrique de la Hoz, and Juan R. Velasco, Access Control Mechanism for IoT Environments Based on Modelling Communication Procedures as Resources, Sensors, Vol 18, No. 3, Mar. 2018.
[6] Mehdi Adda, Jabril Abdelaziz, Hamid Mcheick, Rabeb Saad, Toward an Access Control Model for IOTCollab, Procedia Computer Science, Volume 52, Pages 428-435, 2015,
[7] Arvind Kumar Bansal, Introduction to Programming Languages, Chapman and Hall, 2013.
[8] Massimo Merro, The WHILE programming language, http://profs.sci.univr.it/~merro/files/WhileExtra_l.pdf
[9] Messaoud Benantar, Access Control Systems: Security, Identity Management and Trust Models, Springer, 2006.
[10] Programming Language Haskell, http://www.haskell.org/