W12 Lab: Bell-LaPadula

Team Members:

Jeremy Duong
Austin Lundberg
Timothy Stephenson
Gabriel Sanahuano
Sophia Pearson
David Headrick

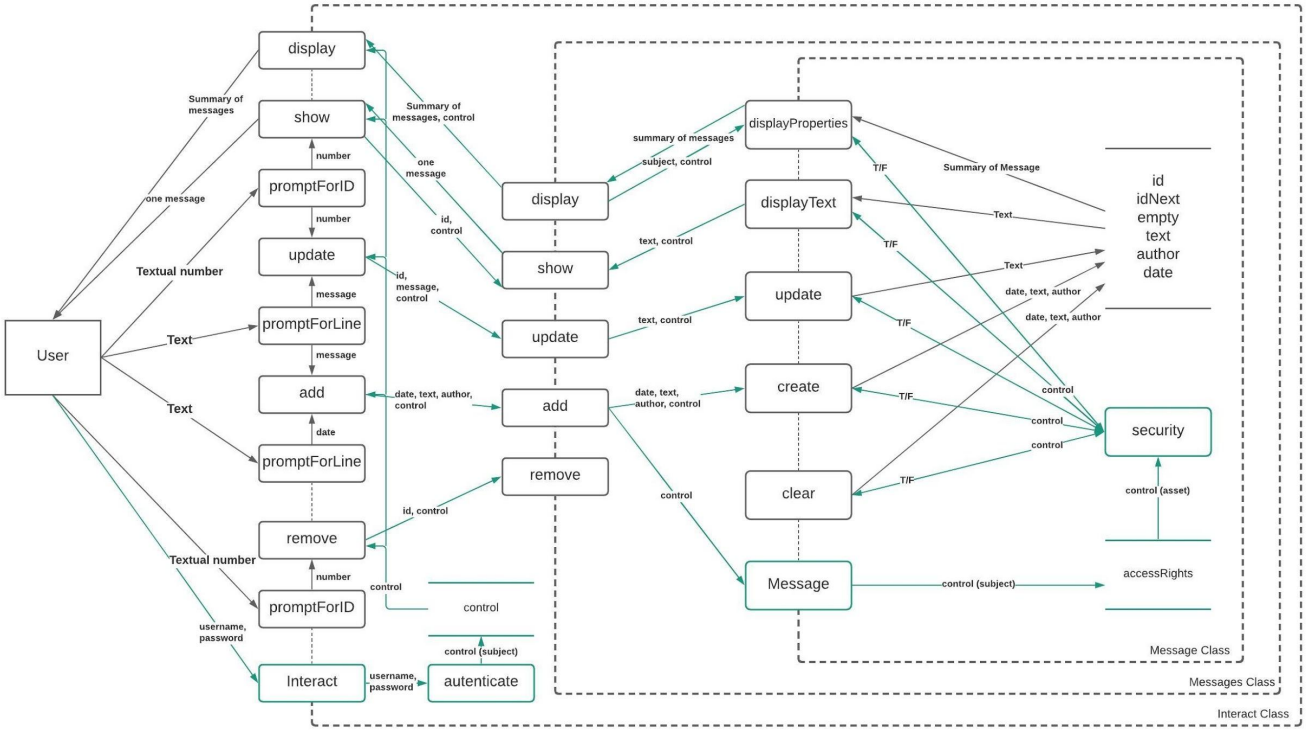
Summary

This report provides a summary of the application of the Bell-LaPadula access control model on an existing program. As evidence of the successful implementation, this report presents a modified version of an original Data Flow Diagram, code snippets of important functions and data structures, and a dozen or so test cases with explanation, expected results, and actual outputs. **In the event that a new privilege level needs to be added to the program, do the following:**

- **Add the new Control level to the Control enum**
- **Add an insert statement in the ControlMap constructor with the appropriate string-value pair**
- **If the added Control level is the highest or lowest level, modify the code in the ControlMap constructor to set lowestLevel or highestLevel to the appropriate string-value pair.**

Bell-LaPadula Implementation

DFD



Justification

The above DFD is the current representation with the security check updates. The updates to the DFD are highlighted in turquoise. The security check (securityConditionRead and securityConditionWrite) are represented by the security process. Security is implemented on an asset-level basis (within the Message class). Therefore, the Message asset cannot be unwrapped without first being able to pass the security check; the Message class trust boundary was strengthened. When a new message is created, the accessRights member variable of the Message class is initialized to assist with future read/write access. The authenticate function is called upon creation of an Interact instance (session) which then initializes the control member variable (representing the subject's control). This member variable is then used by the Interact method and passed as needed through the levels to the security process. This implements the Bell-LaPadula model that operates on a confidentiality basis using the model elements of authenticate(), securityConditionRead(), securityConditionWrite(), and a control data type(enum).

Policy Justification

The code was designed with the ControlMap class which only requires slight changes that can be easily implemented to the needs of the organization. This allows for an easy and intuitive design. Each message has an author and security level specified in the enum Control data type. With every login, one of the enum values will be assigned based on the ID provided. If more security levels are desired, they can be added to the enum Control data type declaration.

Code Snippets

Control data type

```
/*************************************************************************
 * The Control data type
 * The data structure used to determine confidentiality levels
 * ***********************************************************************/
enum Control {
    PUBLIC, CONFIDENTIAL, PRIVILEGED, SECRET
};
```

Explanation:

The Bell-LaPadula model implements security checks using a control data type. The above code implements the control data type using an enum. The elements are listed from least confidentiality access to most for easier comparison in the security functions.

ControlMap class

```
/*************************************************************************
 * Map the string representation of control level to the Control representation
 * to make it easy to add control levels
 * To add more control levels, add more insert statements to the constructor
 * Also remember to insert the value into the enum above.
 * ***********************************************************************/
class ControlMap {
private:
    std::map<std::string, Control> controlMap;
    std::pair<std::string, Control> lowestLevel;
    std::pair<std::string, Control> highestLevel;
public:
    ControlMap() {
        //Add more insert statements to this to include more control levels
        controlMap.insert({"Public", PUBLIC});
        controlMap.insert({"Confidential", CONFIDENTIAL});
        controlMap.insert({"Privileged", PRIVILEGED});
        controlMap.insert({"Secret", SECRET});
```

```
        //If the highest or lowest level changes, please also change these:
        this->lowestLevel = {"Public", PUBLIC};
        this->highestLevel = {"Secret", SECRET};
    };

    std::map<Control, std::string> getMap();

    // Return Control type representation of Control given the string
    Control getControlNum(const std::string &controlText);

    // Return string representation of Control
    std::string getControlString(const Control &controlNum);

    // Get the highest control level
    Control getHighestLevel() { return this->highestLevel.second; }

    // Get the lowest control level
    Control getLowestLevel() { return this->lowestLevel.second; }
};
```

Explanation:

The ControlMap class was written to assist with converting a string to a Control data type and vice versa. This conversion occurs when reading from and writing to the .txt file. In the event that a privilege level is added to the program, both the Control enum and the ControlMap constructor would need to be updated to allow for full functionality.

Security Read/Write functions

```
/*********************************************************************
 * Security Condition Read implementation
 * Determine if the subject can read the asset given the control
 * levels of the asset and the subject
 * ********************************************************************/
bool securityConditionRead(const Control &assetControl,
                           const Control &subjectControl)
{
    return subjectControl >= assetControl; //Read: down allowed, up restricted
}

/*********************************************************************
 * Security Condition Write implementation
 * Determine if the subject can write to the asset given the control
 * levels of the asset and the subject
 * ********************************************************************/
bool securityConditionWrite(const Control &assetControl,
                            const Control &subjectControl)
```

```
{
    return subjectControl <= assetControl; //Write: down restricted, up allowed
}
```

Explanation:

The code above shows proper implementation of the Bell-LaPadula read and write functions which will compare the subjects control level to the assets control level and return a true or false depending on the situation. This will securely validate any request for an asset before the asset is given.

Authenticate function inside interact class

```
/************************************************
 * INTERACT :: AUTHENTICATION
 * authenticate the user: find their control level
 ************************************************/
Control Interact::authenticate(const string & userName,
                               const string & password) const
{
   int id = idFromUser(userName);
   ControlMap controlMap;

   //If username and password are not recognized, assume the lowest level
   Control subjectControl = controlMap.getLowestLevel();
   if (ID_INVALID != id && password == string(users[id].password))
      subjectControl = users[id].subjectControl;
   return subjectControl;
}
```

Explanation:

The authenticate function was modified to implement the return of the proper subjectControl value when the ID is given. If the ID does not match existing user ID's the user will default to returning "public" access. The subjectControl values come from the enum Control data type declared.

Test Cases

1. AdmiralAbe (Secret):
    a. Tries to update a message from Walter the Weatherman (Public).
    b. Expected result: AdmiralAbe can **not** perform this update since this would fail the securityConditionWrite() check. This would be a violation of the "write-down restricted" clearance.

Output:

```
<AdmiralAbe> u
Select the message ID to update: 103
```

```
Please provide a message: updating ...
Cannot update this message!
```

2. AdmiralAbe (Secret):
    a. Tries to update a message **to** Fleet Admiral Chester W. Nimitz (Secret).
    b. Expected result: AdmiralAbe can successfully write a message to Fleet Admiral Chester W. Nimitz because this would **not** violate the “write-down restricted” clearance.

    Output:

```
<AdmiralAbe> u
Select the message ID to update: 106
Please provide a message: updating
Update successful!
```

3. AdmiralAbe (Secret):
    a. Tries to update a message from Winston Churchill (Confidential).
    b. Expected result: AdmiralAbe can **not** update a message from Winston Churchill because this would violate the “write-down restricted” clearance.

    Output:

```
<AdmiralAbe> u
Select the message ID to update: 100
Please provide a message: updating...
Cannot update this message!
```

4. CaptainCharlie (Privileged):
    a. Tries to display a message from J Robert Oppenheimer (Secret).
    b. Expected result: CaptainCharlie can **not** display this message because it would violate the “read-up restricted” protocol.

    Output:

```
<CaptainCharlie> s
Select the message ID to display: 109
Cannot find message!
```

5. CaptainCharlie (Privileged):
    a. Tries to display all of the messages.
    b. Expected Result: CaptainCharlie can **not** display all of the messages; rather he will only see messages that are Privileged, Confidential and Public because of the “read-down permitted” clearance protocol.

    Output:

```
<CaptainCharlie> d
        [100] Message from Winston Churchill at 16 May 1940
        [101] Message from President Franklin D. Roosevelt at 29 December 1940
        [102] Message from Seaman Smith at 5 December 1941
        [103] Message from Walter the Weatherman at 7 December 1941
        [104] Message from President Franklin D. Roosevelt at 8 December 1941
        [105] Message from Lt. Kenneth Taylor at 9 December 1941
        [107] Message from Captain Buckmaster at 7 June 1942
        [108] Message from General George S. Patton at 1 February 1943
```

6. SeamanSam (Confidential):
    a. Tries to display a message from J. Robert Oppenheimer (Secret).
    b. Expected result: SeamanSam can **not** see J. Robert Oppenheimer's message. Because of the "Read-up is restricted" clearance protocol, SeamanSam cannot read anything that is Privileged or Secret.

Output:

```
<SeamanSam> s
Select the message ID to display: 109
Cannot find message!
```

7. SeamanSam (Confidential):
    a. Tries to edit a Public message from Franklin D. Roosevelt.
    b. Expected result: SeamanSam will **not** edit the Public message because it is a document of lower clearance ("Write-down restricted" clearance protocol).

Output:

```
<SeamanSam> u
Select the message ID to update: 101
Please provide a message: updating...
Cannot update this message!
```

8. SeamanSue (Confidential):
    a. Tries to add a new message to the file.
    b. Expected result: SeamanSue (Confidential clearance) can write to a highly sensitive document of Confidential, Privileged, or Secret level, because "Write-up is permitted".

Output:

```
<SeamanSue> a
Please provide a date: 26 March 2021
```

```
Please provide a message: Sue's message...

<SeamanSue> s
Select the message ID to display: 110
        Message: Sue's message...
```

9. SeamanSue (Confidential):
    a. Tries to delete a privileged level message from the file.
    b. Expected result: SeamanSue should be able to delete any message from the file as “write-up privilege is permitted”.

    Output:

```
<SeamanSue> r
Select the message ID to delete: 108
Message deleted.
```

10. SeamanSly (Confidential):
    a. Tries to display all the messages.
    b. Expected result: only Confidential and Public messages show up, Secret and Privileged do **not** show. This is because an individual of low clearance (confidential) can **not** read a document of higher clearance (secret, privileged). This is the “read-up is restricted” clearance.

    Output:

```
<SeamanSly> d
        [100] Message from Winston Churchill at 16 May 1940
        [101] Message from President Franklin D. Roosevelt at 29 December 1940
        [102] Message from Seaman Smith at 5 December 1941
        [103] Message from Walter the Weatherman at 7 December 1941
        [104] Message from President Franklin D. Roosevelt at 8 December 1941
        [105] Message from Lt. Kenneth Taylor at 9 December 1941
```

11. SeamanSly (Confidential):
    a. Tries to show a public message.
    b. Expected result: The message should display normally as “read-down permission” access is allowed.

    Output:

```
<SeamanSly> s
Select the message ID to display: 101
        Message: No man can tame a tiger into a kitten by stroking it.
```

12. Jeremy (Public):
    a. Tries to display all messages
    b. Expected Result: Displays only the public accessible messages and **not** any other higher privilege messages for “read-up restricted” access.

Output:

```
<Jeremy> d
        [101] Message from President Franklin D. Roosevelt at 29 December 1940
        [103] Message from Walter the Weatherman at 7 December 1941
        [104] Message from President Franklin D. Roosevelt at 8 December 1941
```

13. David (Public):
    a. Tries to update a message of higher privilege.
    b. Expected result: The message should be updated successfully to allow for “write-up permitted” access.

Output:

```
<David> u
Select the message ID to update: 109
Please provide a message: The war is over, stop fighting!
Update successful!
```