Black Box & White Box Testing

Student: Patrick Walsh

Date: 5/2/2021

School: University of Maryland Global Campus

Course: SDEV 360 7380

Professor: Dr. Alla Webb

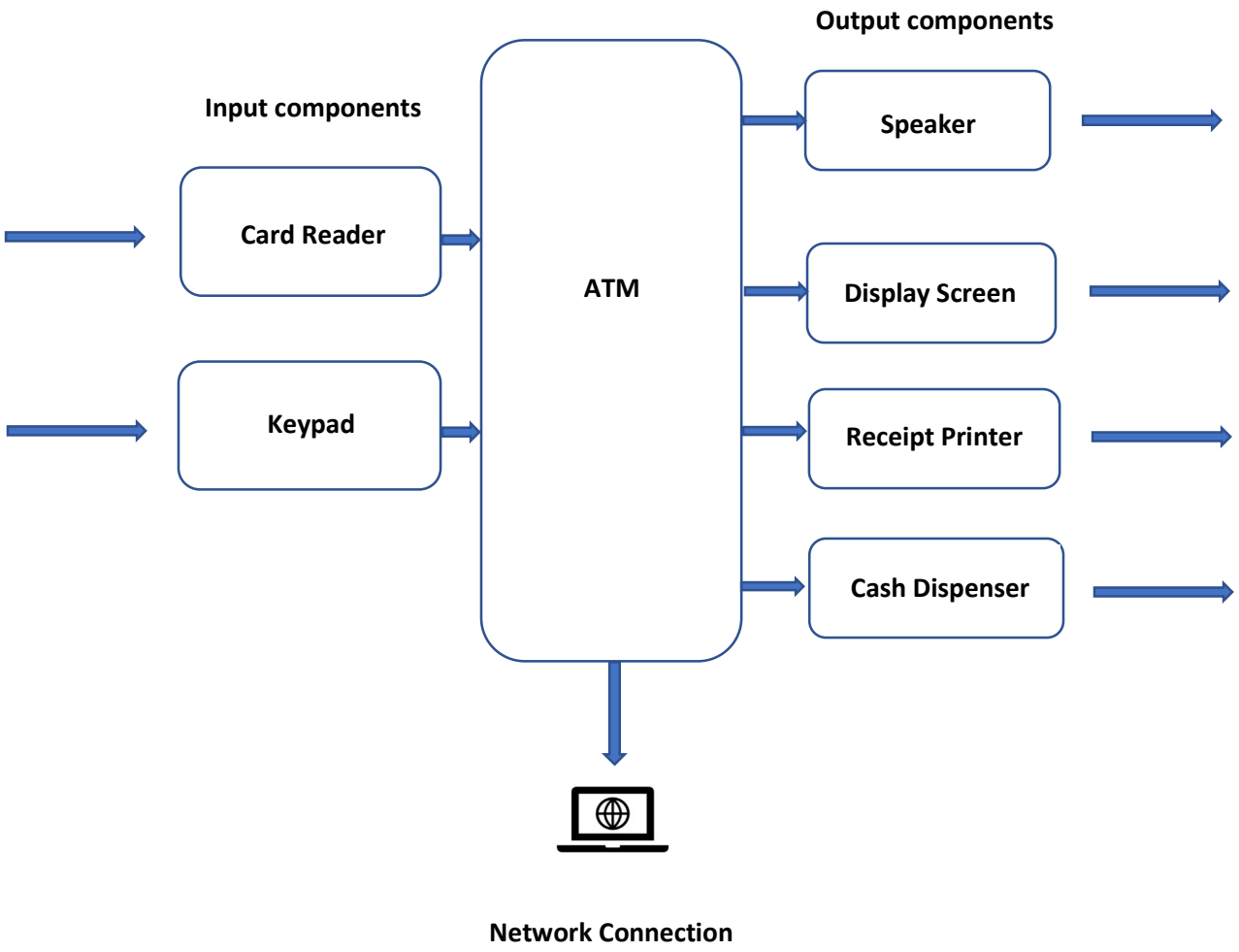
Architectural components of an ATM

Automatic Teller Machines (ATM) allow a user physically present at the ATM to perform various financial transactions, including checking the balance of a checking or savings account, making cash withdrawals and deposits, and transferring funds (Agarwal, 2021). To perform these actions, the user must enter a debit card into the ATM's card reader and enter a Personal Identification Number (PIN) through the keypad.

The physical components of an ATM generally consist of two input components and four output components (Agarwal, 2021). The input components are the card reader and keypad, and the output components consist of a speaker, display screen, receipt printer, and cash dispenser (Agarwal, 2021). The ATM also requires a network connection in order to send user inputs to a bank or financial transaction center and return results such as showing account balance or dispensing money (Agarwal, 2021). The architectural design of an ATM in figure 1 visually shows the components described. At each of these components exists the possibility for threat vulnerabilities which could be exploited by malicious actors, but this paper will focus on the card reader and keypad using a brute force attack to guess the user's PIN.

Figure 1

Architectural Design of an ATM



Note. High-level overview of the basic components of an ATM, showing the two input components, four output components, and a network connection.

Black Box vs. White Box Testing

Black box testing is an approach to testing that uses the design specifications without knowing the internal code of the program (Software Testing Help, 2021). A black box testing scenario

will generate multiple test cases with inputs and expected outputs, verifying the functionality of the program by comparing the expected output with the actual output. White box testing, on the other hand, examines the internal code of the program to generate test cases (Software Testing Help, 2021). White box testing requires that a tester have access to the internal code of the application, allowing them to analyze the program infrastructure. White box testing can often be carried out earlier in the Software Development Lifecycle (SDLC) than black box testing because black box testing requires the tester to use a Graphical User Interface (GUI) while white box testing can be done on the raw code (Software Testing Help, 2021).

Attack: Brute force PIN entry

This black box test case will examine multiple use-cases that address a brute force PIN entry attack carried out at the card reader and keypad components of the ATM. In this test scenario, the tester will simulate an attacker who is attempting to infiltrate a user's account by guessing the PIN. It is assumed for the purposes of this test scenario that the attacker has stolen the user's ATM debit card or has illegally purchased a stolen card. The attacker can attempt to guess the user's PIN manually or use a PIN brute force application. Such an application can be used to guess several thousand PIN combinations within a few seconds to shorten the amount of time it takes to guess the correct PIN.

Mitigation: The ATM's software should implement a timeout limit that prevents many PIN guesses in a short period of time. The ATM should also limit the number of incorrect PINs so that the account is locked out after a certain number of wrong entries. These two mitigation techniques should prevent most brute force attacks.

Black box testing

In test 1, the PIN is entered correctly, and the test passes successfully. In test 2, the attacker uses a brute force PIN guessing software to try different PIN combinations until the correct one is guessed. In test 3, the mitigation techniques are applied and the same actions from test 2 are repeated. The ATM should limit the number of PIN guesses to once per every second and should lock out the account after 5 wrong guesses.

Test case	Input	Expected output	Actual output	Test passed?
1	Enter debit card into card reader and enter the correct PIN '4437'into the keypad.	Should see message on screen that says, 'PIN accepted'.	Message appears that says, 'PIN accepted'.	True
2	Enter debit card into card reader and use brute force PIN guessing software to quickly try different combinations of PINs until the correct PIN is entered.	Should see several messages on screen that say, 'Incorrect PIN' until finally the message changes to 'PIN accepted'.	Message appears on screen that says, 'Incorrect PIN' several times before message changes to 'PIN accepted'.	False
3	Mitigation is applied to ATM's software, then actions taken in test 2 are repeated.	Should see messages display on screen that say, 'Incorrect PIN' once per second. This message should appear 5 times before another message displays that says, 'Maximum attempts reached. Account locked'.	Messages display once per second saying, 'Incorrect PIN'. This occurs 5 times before another message displays saying, 'Maximum attempts reached. Account locked'.	True

White box testing

The code below shows how the ATM software should react when various inputs are given. After the ATM welcome screen, the program uses a brute_force() function to guess the user's PIN. The PIN is 4 digits long.

```
print("WELCOME TO THE ATM!\nPLEASE ENTER A PIN\n\n")


def brute_force(pin):
    """
    Takes in PIN as 4 character string and returns a
    randomly generated combination of 4 digits.
    """
    # PIN is 4 digits long
    characters = [secrets.choice(NUMBERS), secrets.choice(NUMBERS), secrets.choice(NUMBERS),
                  secrets.choice(NUMBERS)]
    for i in range(4):  # generates random PIN with secrets module
        pin += characters[secrets.randbelow(len(characters) - 1)]

    return pin
```

The program is currently set to not mitigate brute force attacks:

```
using_mitigation = False  # if set to False, will not employ brute force mitigation
```

When the program is executed, the brute_force() function is used to hack into the ATM by guessing the PIN. In the highlighted portion of the code inside the while loop, the program checks if the PIN is correct. If it is, then the ATM displays the proper success message and the program terminates. If the PIN is incorrect, the program displays an error message and then attempts another PIN using the brute_force() function.

```
while True:
    if using_mitigation:
        time.sleep(1)  # limits attempts to once per se
        if attempt_num > 4:  # locks account after 5 fa
            print('Maximum attempts reached. Account lo
            break

    if brute_force(pin_guess) == PIN_CORRECT:
        print('PIN accepted')
        print('Attempt number:', attempt_num)
        break
    else:
        print(brute_force(pin_guess), 'Incorrect PIN')
        attempt_num += 1
```

Below is an example output:

```
brute_force_PIN
5554 Incorrect PIN
2225 Incorrect PIN
5020 Incorrect PIN
1111 Incorrect PIN
0007 Incorrect PIN
1696 Incorrect PIN
6644 Incorrect PIN
4999 Incorrect PIN
6566 Incorrect PIN
9935 Incorrect PIN
4669 Incorrect PIN
7744 Incorrect PIN
7122 Incorrect PIN
3535 Incorrect PIN
9993 Incorrect PIN
9484 Incorrect PIN
PIN accepted
Attempt number: 16316

Process finished with exit code 0
```

As seen in the output, the program successfully guessed the correct PIN ‘4437’ on attempt number 16,316. The program only took 1-2 seconds to run since the brute_force() function can try thousands of different combinations per second.

Now the program will mitigate brute force attacks by setting ‘using_mitigation’ to True:

```
using_mitigation = True  # if set to False, will not employ brute force mitigation
```

Now the program will only allow a PIN to be entered once per second. The program is also set to lock out the account after 5 failed login attempts:

```
if using_mitigation:
    time.sleep(1)  # limits attempts to once per second
    if attempt_num > 4:  # locks account after 5 failed login attempts
        print('Maximum attempts reached. Account locked.')
        break
```

Below is an example output of the program being run with mitigation in place:

```
WELCOME TO THE ATM!
PLEASE ENTER A PIN


0066 Incorrect PIN
2626 Incorrect PIN
0355 Incorrect PIN
4448 Incorrect PIN
7877 Incorrect PIN
Maximum attempts reached. Account locked.

Process finished with exit code 0
```

References

Agarwal, T. (2021, January 2). *Automated Teller Machine: Block Diagram, Types & Its Working*. ElProCus - Electronic Projects for Engineering Students. https://www.elprocus.com/automated-teller-machine-types-working-advantages/

Software Testing Help. (2021, April 30). *Key Differences Between Black Box Testing and White Box Testing*. https://www.softwaretestinghelp.com/black-box-vs-white-box-testing/