Assignment 1: Code Vulnerabilities, Secure coding and Android Security Programming

FIT5003 Software Security S2 2020

Faculty of Information Technology, Monash University

Submission Guidelines

- **Deadline:** Assignment 1 Report submission is due on **Sunday 11 th October 2020, 23:00**.
- **Demonstration Deadline:** The demonstration via Panopto platform or the interview must be shared/-conducted before Sunday 11 th October 2020, 23:00.
- **Submission Files:**
  1. A report in PDF file format of maximum 6 pages as a reference. Appendices and References are excluded from the page count.
  2. Appropriate c code as an answer to relevant tasks
  3. Appropriate Android Studio Project compressed as a zip or rar file

  Notes:

  1. A handwritten document is **not** acceptable and will **not** be marked even if converted and submitted electronically.

- **Individual Assignment:** This is an individual assignment so each student should work on the assignment tasks alone.
- **Submission Platform:**
  - Electronic submission via Moodle for the report.
  - Electronic submission via Moodle for the assignment task code and apks
  - Share of video interview files with your tutor via the Monash panopto service
- **Filename Format:** Name your files for different assignment tasks as follows:
  1. Submission via moodle: report_SID.pdf
  2. Sharing via Panopto: A1_interview_SID as title, in case of an interview. **Please follow the provided structure for avoiding marki reduction**
  3. Submission via moodle of assignment task files: use A1_task_number_SID and the appropriate file extension for the relevant task
- **Late Submission Policy:** Submit a special consideration form to formally request a late submission. For this semester, special consideration requests should be send directly to the faculty and not just the tutor team. However, do inform the teaching team of your request.
- **Late Submission Penalty:** A late submitted assignment without prior approval will receive a late penalty of 20% deduction per day (including Saturday and Sunday) or part thereof, after the due date and time.

- **Plagiarism:** It is an academic requirement that your submitted work be original. Zero marks will be awarded for the whole submission if there is any evidence of copying, collaboration, pasting from websites, or copying from textbooks.
  Note: Plagiarism policy applies to all assessments.
- **Grading Procedure:**
  - To receive a grade for the assignment you must demonstrate and explain your work by creating a video recording of **maximum 20 minutes** using **Panopto platform** and share it with your tutor.
  - You must only demonstrate what you have submitted via report.
  - If you have any privacy concern regarding the Panopto platform then you need to raise it with your tutor by **Monday 5th October 2020**. Requests for interviews after this date will not be accepted.
  - You can use the report and any other notes you have prepared beforehand to help you explain and demonstrate your work.
- **IT Use Policy**: Your submission must comply with Monash University's IT Use Policy.

Marks

- This assignment is worth **30%** of the total unit marks.
- The assignment is marked out of **30** nominal marks.

For all the following tasks you must demonstrate the attack on the provided VM in a recorded (Panopto or Zoom) where you explain the attack while performing it.

1 Task 1: Vulnerabilities and Secure Coding (10 Marks)

The provided VM runs the following services:

- **Task 1.1**: On TCP port 7070, the server accepts appropriate input from a remote client in order to execute a specific linux command. The server program appears in three different versions:
  - **version 1**: The server accepts from a client appropriate input and performs an `nslookup` linux command to get information about a specific domain.
  - **version 2**: The server accepts from a client appropriate input and performs a `ping` linux command to find out if some ip address is reachable from the server. Only IP addresses should be allowed
  - **version 3**: The server accepts from a client appropriate input and performs a `curl` linux command to get information about a specific url and check if it is reachable from the server.
  - **version 4**: The server should accept from the client appropriate input to view the files of the folder `test`

  You can find the code for this server in Listing-1. The program has a vulnerability.

  1. a) Identify the vulnerability. [1 Mark]
  2. b) Exploit the vulnerability in order to [2 marks]:
     - **version 1:** Recover the `/etc/shadow` file in Linux.
     - **version 2:** Get a reverse shell from the server back to a malicious client thus giving this client shell access to the server
     - **version 3:** change the privilege level to read, write and execute for all files in the folder called `hidden`
     - **version 4:** add a user to the linux system called `nuser`
  3. c) Change the code so it is no longer vulnerable to the identified attack. [3 Marks]

- **Task 1.2**:On TCP port 6061 to 6065 (depending on the version), the server runs a network wrapper for a legacy program without networking capability to allow access through network to this program simple functionality. The legacy code authenticates users and retrieves the content of corresponding file for the authenticated user. You can find the code for this server in Listing-2 and Listing-3. The legacy program is vulnerable to several vulnerabilities. Note that there are 5 versions of the above server, each one of them is a little different from the others.

  1. a) Exploit some vulnerabilities and get a reverse shell from the server to your client. [4 Marks]
  2. b) Exploit some vulnerability and get access to the file without having knowledge of a username and a password. [3 Marks]
  3. c) Change the code so it is no longer vulnerable to the identified attacks. [3 Marks]

Info: How to choose the correct version to work with?
In the link that exists in the Moodle, place your studentID and then click on the download button to get the versions for the above tasks that you will be using for the assignment. In the provided VM the files of each version have the following names:

- **T1,1 version 1 :** filename to execute `codeserver1`
- **T1,1 version 2 :** filename to execute `codeserver2`
- **T1,1 version 3 :** filename to execute `codeserver3`
- **T1,1 version 4 :** filename to execute `codeserver4`
- **T1,1 version 1 :** filename to execute `codeserver1`
- **T1.2 version 1 :** filename to execute `asgm1_serv1` and port 6061
- **T1.2 version 2 :** filename to execute `asgm1_serv2` and port 6062
- **T1.2 version 3 :** filename to execute `asgm1_serv3` and port 6063
- **T1.2 version 4 :** filename to execute `asgm1_serv4` and port 6064
- **T1.2 version 5 :** filename to execute `asgm1_serv5` and port 6065

Info: To setup the provided VM (it is an oracle VirtuaBox VM) just double click the file or use file open from the oracle VirtuaBox to include it in the oracle VirtuaBox. Then you can setup the network tab in the VM setting to set the network adapter to Host only adapter mode. You can use a simple client (eg. using netcat) in another VM (eg. one of the VMs that we used in lab 3) to connect to the server inside the assignment 1 provided VM. You need to also set the client VM in Host-only adapter Mode. Alternatively, you can set both VMs in the same NAT network (as was described in the setup instructions of lab 3). The username and the password in the assignment 1 provided VM is `student` and `fit5003` respectively.
To start any of the servers (i.e. a version of the server) in the main folder of the user account you can find all possible versions of the task 1.1 and 1.2 Use the versions that are assigned to you by executing `sudo ./filename`

2 Task 2: Android Java-Based application (15 Marks)

In the Android Studio project that is provided, there is a sketched Android Application that is using a series of Keystore files stored inside the android app internal storage area in order to perform some security related operations. The user should be able to login with a username and password and then get access to a secure SMS service. The Android Application consists of two Activities, a public one (called

PrivateUserActivity and a private one called PrivateActivity. When the application start, the activity (PrivateUserActivity) is loaded. The PrivateUserActivity provides text areas for the user to add his username and password and two buttons, an OK button that when pressed transfers the username and password information to the PrivateActivityand a CANCEL button that closes the App. The Application has been created only for a few users and each one of them has his/her own keystore file.
Authentication Mechanism: The PrivateActivity collects the username and password information from the PrivateUserActivity, generates its hash value (uses SHA256) after concatenating username and password strings and uses the first 5 digits of the hash function result as the password of the provided keystore. If the keystore is accessible then the the user is authenticated otherwise a Toast message is generated and the App returns to the PrivateUserActivity.

Inside the keystore there should be entries of at least 3 certificates (public keys) of other users (SMS receivers) and the user's public key cryptography key pair. The PrivateActivity has a textfield where a mobile phone number can be placed (for sending securely an SMS), a textfield where the SMS message can be written and a textfield where the alias of the public key inside the keystore to be used for secure communication must be provided by the user. Apart from the above, there is also a textview area where information about the alias can be seen.

When the SECURE SMS button is pressed, the following SMS securing mechanism is taking place:

1. The public key of the sms message receiver is chosen (using aliases inside the keystore that have the receiver's trusted certificate)
2. A random symmetric key is generated (for AES 256 in CBC mode)
3. The SMS message is encrypted using the generated secret key (AES 256 in CBC mode)
4. The secret key is encrypted using the receiver's public key
5. The encrypted SMS and encrypted secret key are concatenated
6. The result is encoded into a base64 string and the result is sent as an SMS
7. After the submission of the message, the digital signature of the message is generated (using the user's private key)
8. This information is been sent as an intent result to the PrivateUserActivity and is visualized using the Toast message mechanism.

Info: A Android Studio starting project is provided in the unit's Moodle to assist you in the App development. For compatibility reasons, it is suggested to used Android Oreo and test the program in a virtual Nexus 5 device.

1. **Task 2.1 (10 Marks)** Implement the functionality of the Activity PrivateActivity as well as any additional functions needed in PrivateUserActivity so that:
   - it collects and processes the information coming from PrivateUserActivity
   - it verifies the username and password using the above described mechanism.
   - it extracts from the keystore the information regarding the keys and certificates and shows in the textview area of the activity the key aliases, the certificate type and the cipher that is been used.
   - the SMS secure transmission mechanism is implemented
   - the provided digital signature is returned to PrivateActivity and is printed in the application screen using the Toast class (see relevant code inside the provided Android project)
2. **Task 2.2 (5 Marks)**: Based on the existing design approach and functionality that appears on the provided Android Application as well as the code that you have developed to solve Task 2.1 explain possible design issues that can compromise the security of this Android Application as well as possible solutions to those problems.

3 APPENDIX: C code listings

3.1 Starting C Code of task 1.1: command server (nslookup execution example)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>

#define PORT 7070

int exec_command(int sock, char *buf) {
    char command[300];
    // redirecting the stdout to the socket
    close(STDOUT_FILENO);
    dup2(sock, STDOUT_FILENO);

    // The following line is different for each linux command to be executed.
    sprintf(command, "/usr/bin/nslookup %s",  buf);

    system(command);
    return 0;
}

void main()
{
    struct sockaddr_in server;
    struct sockaddr_in client;
    int clientLen;
    int sock,newsock;
    char buf[1500];
    pid_t pid,current = getpid();
    int ret_val;

    sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock < 0) {
     perror("Error opening socket");
     exit(1);
    }
    memset((char *) &server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(PORT);

    ret_val = bind(sock, (struct sockaddr *) &server, sizeof(server));
    if (ret_val < 0) {
        perror("ERROR on binding");
        exit(1);
    }

    listen(sock, 5);
    clientLen = sizeof(client);

    while (1) {
        newsock = accept(sock, (struct sockaddr *) &client, &clientLen);
```

```
        if (newsock < 0) {
         perror("Error on accept");
         exit(1);
        }

        if (fork() < 0) {
         perror("Error on fork");
         close(sock);
exit(1);
        }
pid = getpid();
// printf("child pid for new connectin %d", pid);
        if (pid == current) {
continue;
}
else {
         bzero(buf, 1500);
         recvfrom(newsock, buf, 1500-1, 0, (struct sockaddr *) &client, &clientLen);
         exec_command(newsock, buf);
         close(newsock);
         exit(0);
        }

    }
    close(sock);
}
```

3.2 Starting C Code of task 1.2 (legacy application running by the server)

The code is the same in all versions with different values of `DUMMY_SIZE` and `BUF_SIZE`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/ip.h>


void myprintf(char *msg)
{
    char dummy[DUMMY_SIZE]; memset(dummy, 0, DUMMY_SIZE);
    printf(msg);
}

int check_authentication(char* username, char *password) {
         char password_buffer[BUF_SIZE];
         int auth_flag[1];

         auth_flag[0] = 0;
         strcpy(password_buffer, password);

         printf("checking␣the␣following␣␣username:␣");
         myprintf(username);

         if(strcmp(password_buffer, "passwd1")==0 & strcmp(username, "user1") == 0)
                 auth_flag[0] = 1;
         if(strcmp(password_buffer, "passwd2")==0 & strcmp(username, "user2") == 0)
```

```
                auth_flag[0] = 1;
        if(strcmp(password_buffer, "passwd3")==0 & strcmp(username, "user3") == 0)
                auth_flag[0] = 1;
        if(strcmp(password_buffer, "passwd4")==0 & strcmp(username, "user4") == 0)
                auth_flag[0] = 1;
        if(strcmp(password_buffer, "passwd5")==0 & strcmp(username, "user5") == 0)
                auth_flag[0] = 1;
        if(strcmp(password_buffer, "passwd6")==0 & strcmp(username, "user6") == 0)
                auth_flag[0] = 1;
        if(strcmp(password_buffer, "passwd7")==0 & strcmp(username, "user7") == 0)
                auth_flag[0] = 1;
        if(strcmp(password_buffer, "admino")==0 & strcmp(username, "admin") == 0)
                auth_flag[0] = 1;
        return auth_flag[0];
}

int main(int argc, char *argv[]) {
        int result=0;
   FILE *fp;
   char buff[255];
        if(argc < 3) {
                printf("Usage:␣%s␣<username>␣<␣password>\n", argv[0]);
                exit(0);
        }

        result=check_authentication(argv[1], argv[2]);

        if(result!=0) {
                printf("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
                printf("␣␣␣␣␣␣Access␣Granted.\n");
                printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
           fp = fopen("./fit5003.txt", "r");
           fgets(buff, 255, (FILE*)fp);
           printf("secret␣data:␣%s\n", buff );
           fclose(fp);
        } else {
                printf("\nAccess␣Denied.\n");
   }
}
```

3.3 Starting C Code of task 1.2 (server)

The server is almost the same in all versions (please check comment in the code)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>

#define PORT 6060

int exec_command(int sock, char *buf) {

    char *val0;
```

```
    char *val1;
    char command[5000];
    // redirecting the stdout to the socket
    close(STDOUT_FILENO);
    dup2(sock, STDOUT_FILENO);
    val0= strtok(buf, " ");
    val1= strtok(NULL, "\n");

// The following line is slightly different in each version
// since a different authenticate program is executed in each version
   char * argvlist[] = {"authenticate",val0, val1, NULL};

    printf("1 :%s, 2: %s|",val1, val0);

    execve("./authenticate", argvlist, NULL);

    return 0;
}

void main()
{
    struct sockaddr_in server;
    struct sockaddr_in client;
    int clientLen;
    int sock,newsock;
    char buf[35500];
    pid_t pid,current = getpid();
    int ret_val;

    sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    if (sock < 0) {
     perror("Error opening socket");
     exit(1);
    }
    memset((char *) &server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(PORT);

    ret_val = bind(sock, (struct sockaddr *) &server, sizeof(server));
    if (ret_val < 0) {
        perror("ERROR on binding");
        exit(1);
    }

    listen(sock, 5);
    clientLen = sizeof(client);

    while (1) {
        newsock = accept(sock, (struct sockaddr *) &client, &clientLen);
        if (newsock < 0) {
         perror("Error on accept");
         exit(1);
        }

        if (fork() < 0) {
```

```
         perror("Error on fork");
         close(sock);
exit(1);
        }
pid = getpid();
printf("child pid for new connectin %d", pid);
        if (pid == current) {
continue;
}
else {
bzero(buf, 35500);
recvfrom(newsock, buf, 35500-1, 0, (struct sockaddr *) &client, &clientLen);
exec_command(newsock, buf);
close(newsock);
exit(0);
        }
    }
    close(sock);
}
```