Group Assignment

Database Design, Implementation and Security

IE2042- Database Management Systems for Security

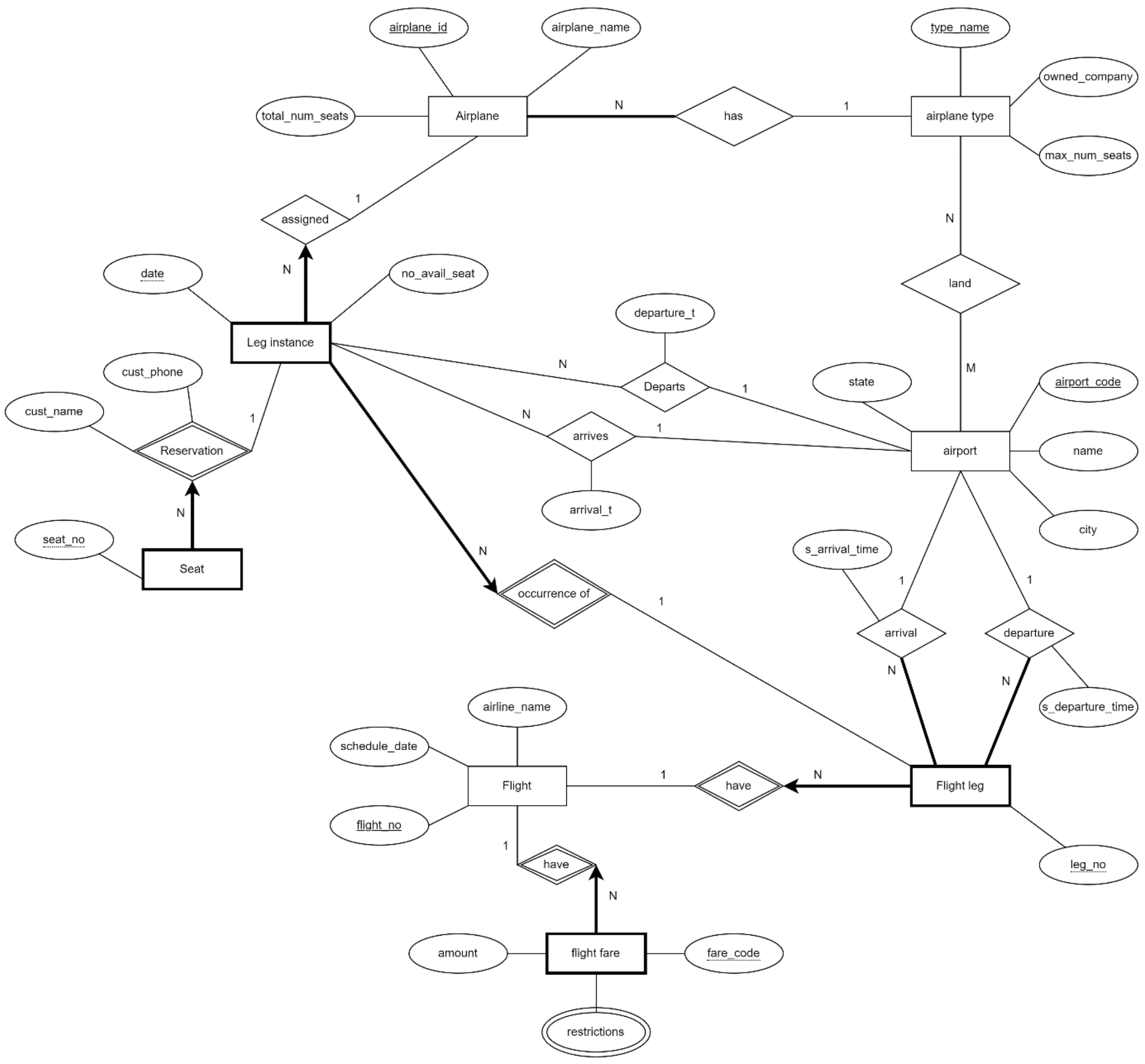
Student Name	Registration Number
D.S.C. Wijesuriya	IT21155802
P.G.E Janith Sandamal	IT21166860
W.N. Dilsara	IT21182600
Premakanthan. N	IT21197550

Table of contents

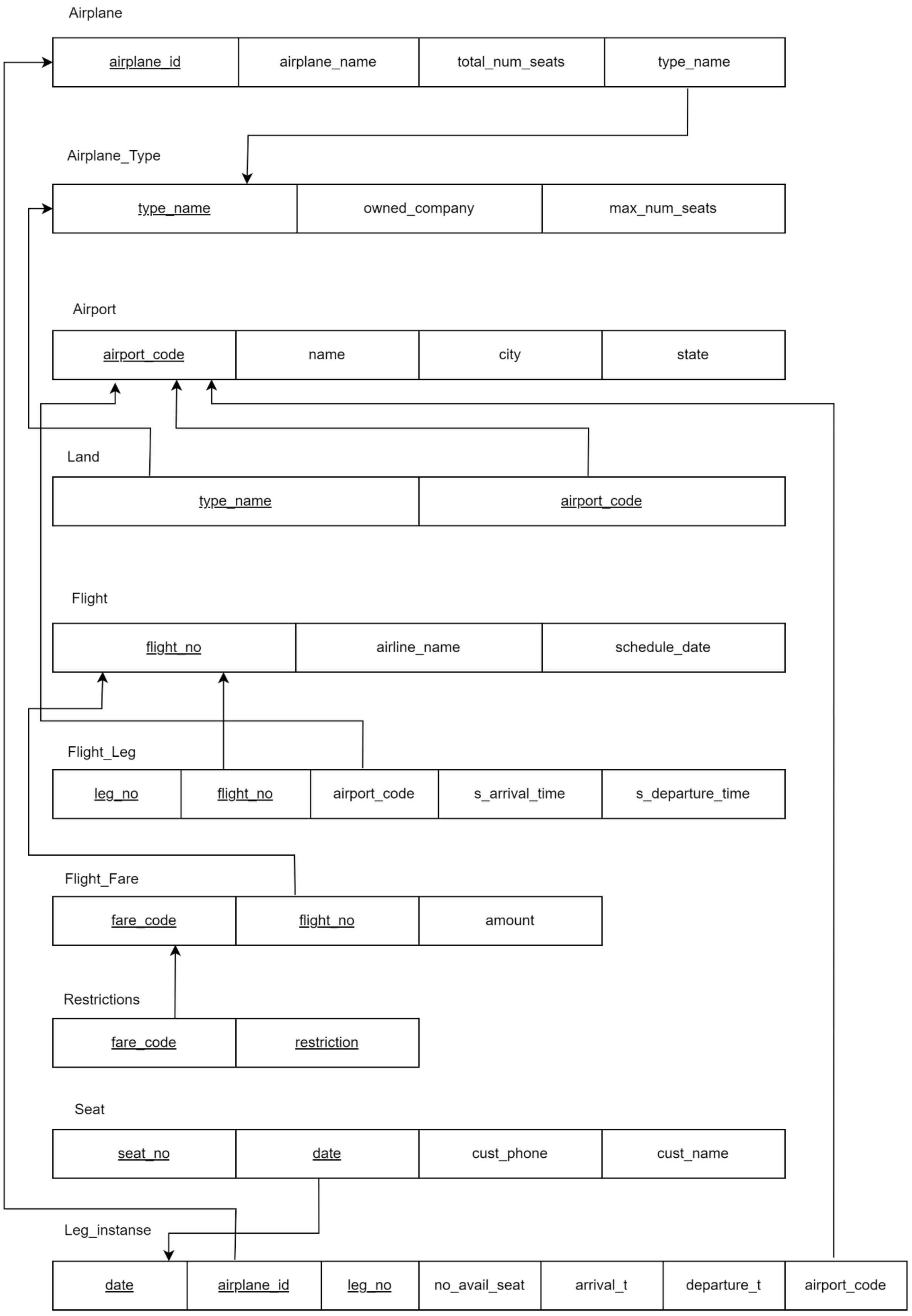
Assumptions

1. A leg instance is a particular occurrence of a leg on a particular date.
2. A leg is a nonstop portion of a flight
3. It is assumed that restrictions of the flight fare can contain multiple values.

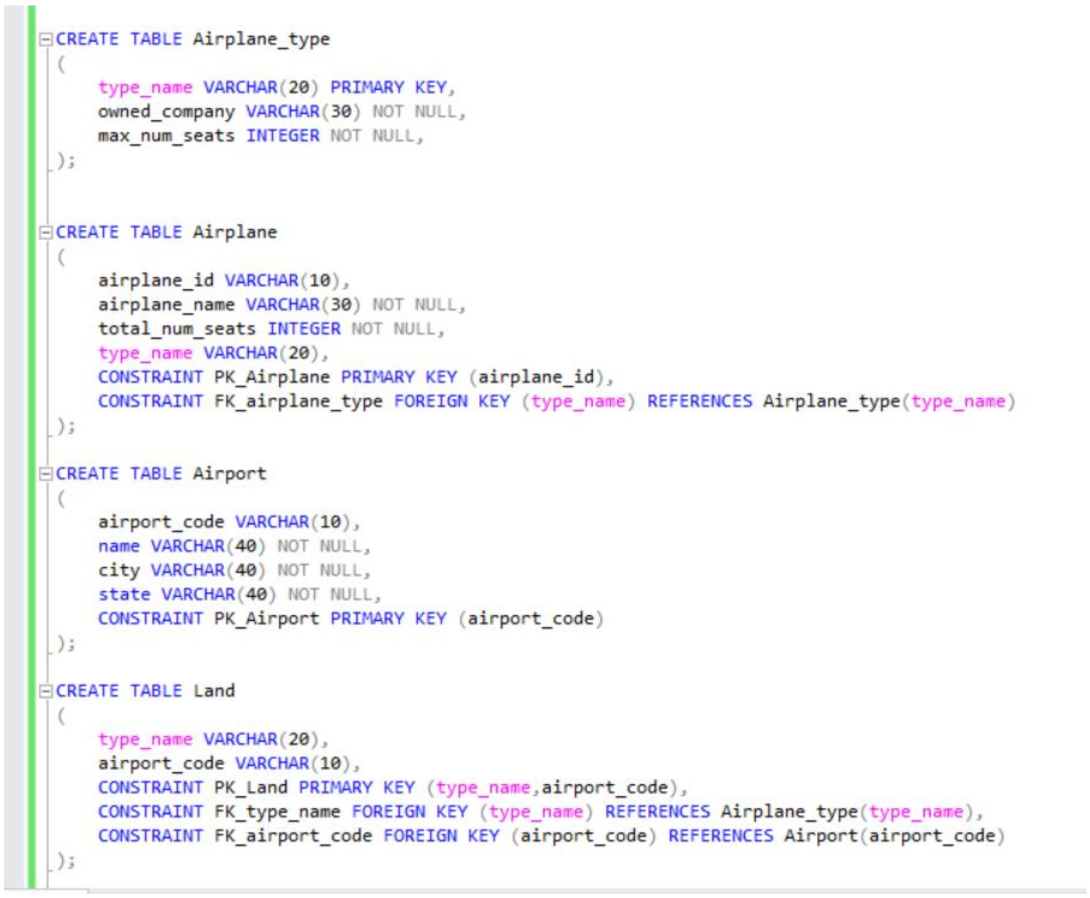
ER Diagram



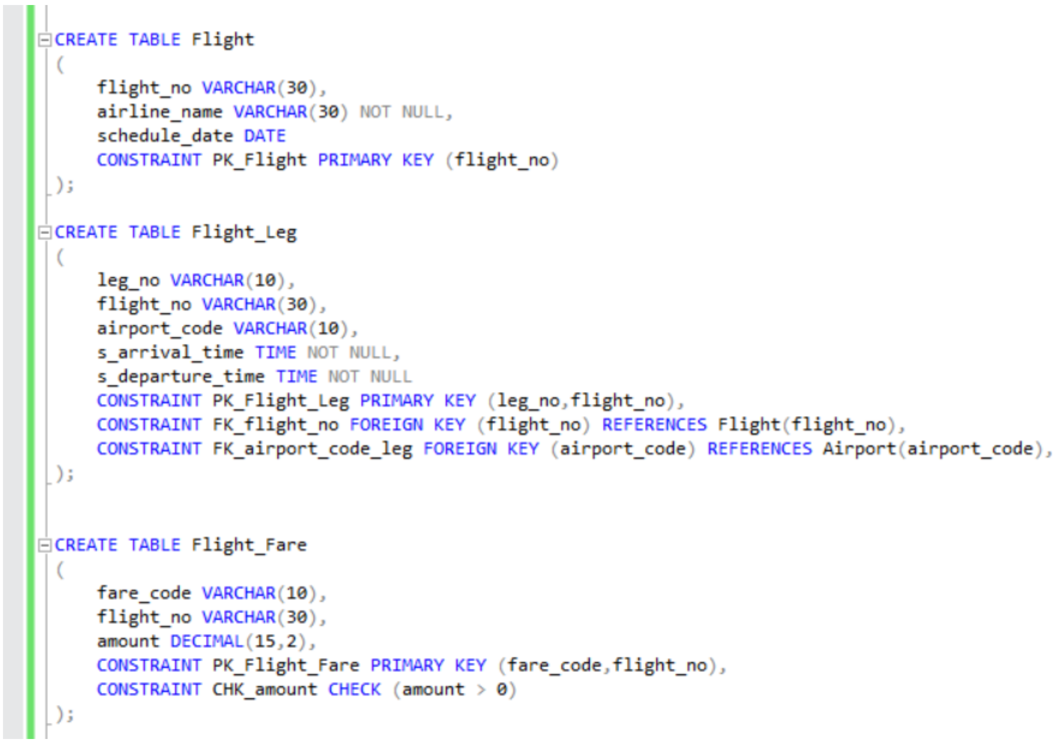
Logical Model



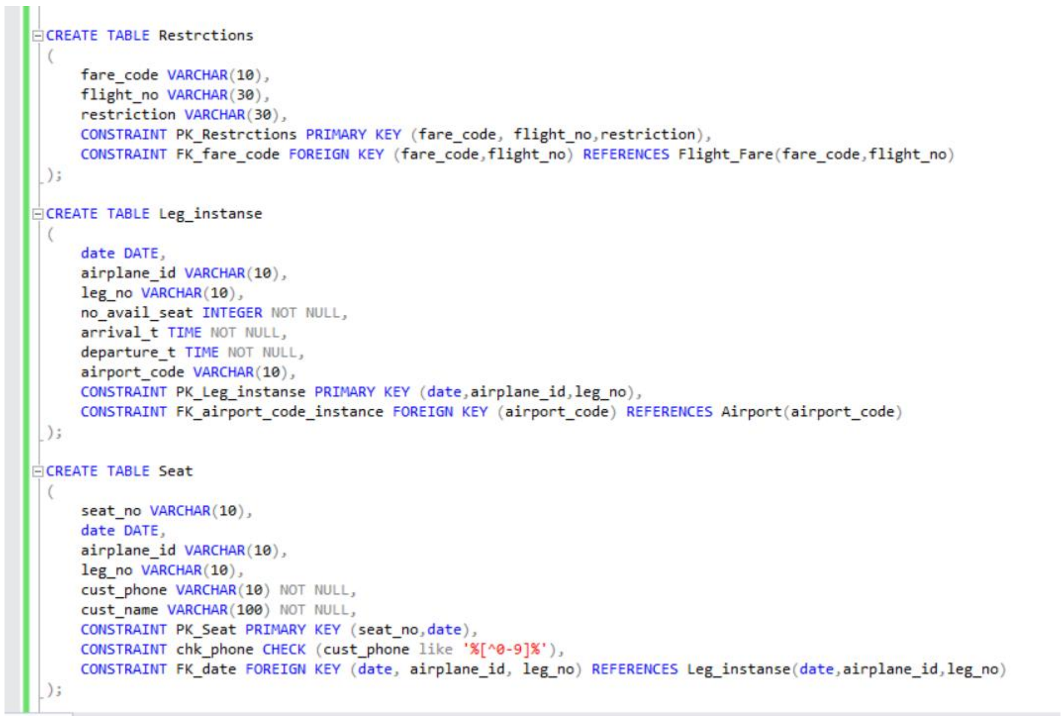
Implementation of Logical model



```sql
CREATE TABLE Airplane_type
(
    type_name VARCHAR(20) PRIMARY KEY,
    owned_company VARCHAR(30) NOT NULL,
    max_num_seats INTEGER NOT NULL,
);

CREATE TABLE Airplane
(
    airplane_id VARCHAR(10),
    airplane_name VARCHAR(30) NOT NULL,
    total_num_seats INTEGER NOT NULL,
    type_name VARCHAR(20),
    CONSTRAINT PK_Airplane PRIMARY KEY (airplane_id),
    CONSTRAINT FK_airplane_type FOREIGN KEY (type_name) REFERENCES Airplane_type(type_name)
);

CREATE TABLE Airport
(
    airport_code VARCHAR(10),
    name VARCHAR(40) NOT NULL,
    city VARCHAR(40) NOT NULL,
    state VARCHAR(40) NOT NULL,
    CONSTRAINT PK_Airport PRIMARY KEY (airport_code)
);

CREATE TABLE Land
(
    type_name VARCHAR(20),
    airport_code VARCHAR(10),
    CONSTRAINT PK_Land PRIMARY KEY (type_name,airport_code),
    CONSTRAINT FK_type_name FOREIGN KEY (type_name) REFERENCES Airplane_type(type_name),
    CONSTRAINT FK_airport_code FOREIGN KEY (airport_code) REFERENCES Airport(airport_code)
);
```

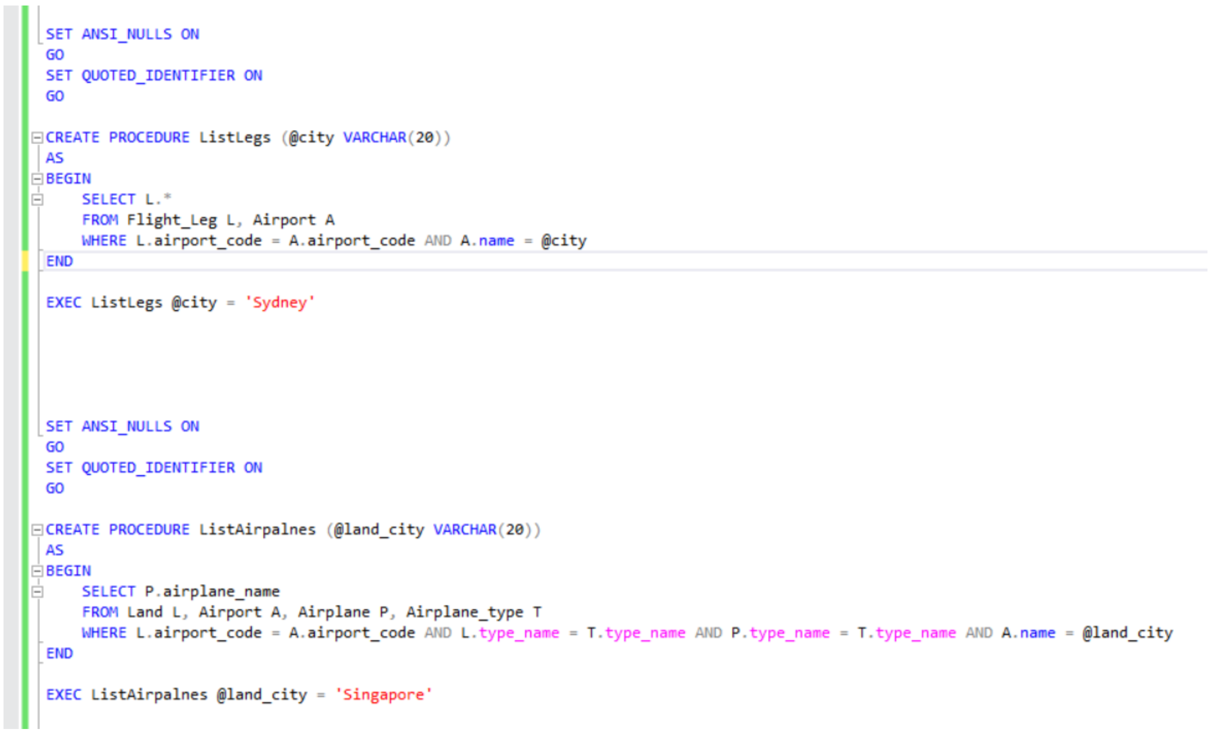


```sql
CREATE TABLE Flight
(
    flight_no VARCHAR(30),
    airline_name VARCHAR(30) NOT NULL,
    schedule_date DATE
    CONSTRAINT PK_Flight PRIMARY KEY (flight_no)
);

CREATE TABLE Flight_Leg
(
    leg_no VARCHAR(10),
    flight_no VARCHAR(30),
    airport_code VARCHAR(10),
    s_arrival_time TIME NOT NULL,
    s_departure_time TIME NOT NULL
    CONSTRAINT PK_Flight_Leg PRIMARY KEY (leg_no,flight_no),
    CONSTRAINT FK_flight_no FOREIGN KEY (flight_no) REFERENCES Flight(flight_no),
    CONSTRAINT FK_airport_code_leg FOREIGN KEY (airport_code) REFERENCES Airport(airport_code),
);

CREATE TABLE Flight_Fare
(
    fare_code VARCHAR(10),
    flight_no VARCHAR(30),
    amount DECIMAL(15,2),
    CONSTRAINT PK_Flight_Fare PRIMARY KEY (fare_code,flight_no),
    CONSTRAINT CHK_amount CHECK (amount > 0)
);
```

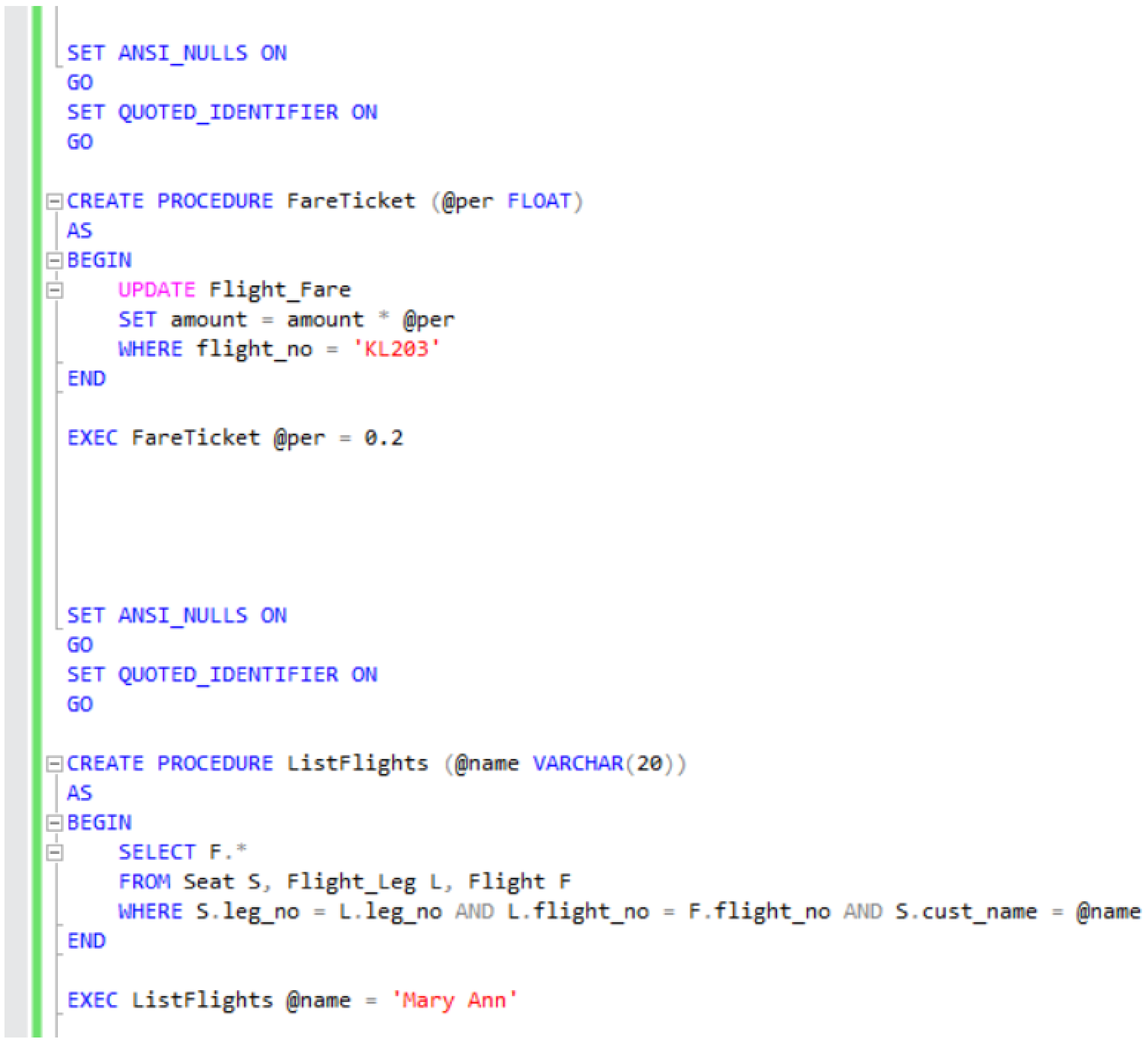


```sql
CREATE TABLE Restrctions
(
    fare_code VARCHAR(10),
    flight_no VARCHAR(30),
    restriction VARCHAR(30),
    CONSTRAINT PK_Restrctions PRIMARY KEY (fare_code, flight_no,restriction),
    CONSTRAINT FK_fare_code FOREIGN KEY (fare_code,flight_no) REFERENCES Flight_Fare(fare_code,flight_no)
);

CREATE TABLE Leg_instanse
(
    date DATE,
    airplane_id VARCHAR(10),
    leg_no VARCHAR(10),
    no_avail_seat INTEGER NOT NULL,
    arrival_t TIME NOT NULL,
    departure_t TIME NOT NULL,
    airport_code VARCHAR(10),
    CONSTRAINT PK_Leg_instanse PRIMARY KEY (date,airplane_id,leg_no),
    CONSTRAINT FK_airport_code_instance FOREIGN KEY (airport_code) REFERENCES Airport(airport_code)
);

CREATE TABLE Seat
(
    seat_no VARCHAR(10),
    date DATE,
    airplane_id VARCHAR(10),
    leg_no VARCHAR(10),
    cust_phone VARCHAR(10) NOT NULL,
    cust_name VARCHAR(100) NOT NULL,
    CONSTRAINT PK_Seat PRIMARY KEY (seat_no,date),
    CONSTRAINT chk_phone CHECK (cust_phone like '%[^0-9]%'),
    CONSTRAINT FK_date FOREIGN KEY (date, airplane_id, leg_no) REFERENCES Leg_instanse(date,airplane_id,leg_no)
);
```

Stored procedures



```sql
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE ListLegs (@city VARCHAR(20))
AS
BEGIN
    SELECT L.*
    FROM Flight_Leg L, Airport A
    WHERE L.airport_code = A.airport_code AND A.name = @city
END

EXEC ListLegs @city = 'Sydney'




SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE ListAirpalnes (@land_city VARCHAR(20))
AS
BEGIN
    SELECT P.airplane_name
    FROM Land L, Airport A, Airplane P, Airplane_type T
    WHERE L.airport_code = A.airport_code AND L.type_name = T.type_name AND P.type_name = T.type_name AND A.name = @land_city
END

EXEC ListAirpalnes @land_city = 'Singapore'
```



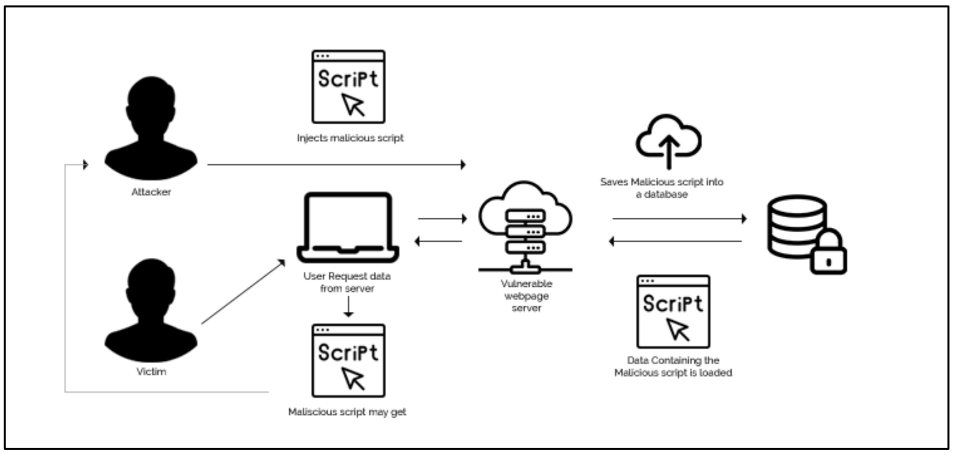
```sql
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE FareTicket (@per FLOAT)
AS
BEGIN
    UPDATE Flight_Fare
    SET amount = amount * @per
    WHERE flight_no = 'KL203'
END

EXEC FareTicket @per = 0.2




SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE ListFlights (@name VARCHAR(20))
AS
BEGIN
    SELECT F.*
    FROM Seat S, Flight_Leg L, Flight F
    WHERE S.leg_no = L.leg_no AND L.flight_no = F.flight_no AND S.cust_name = @name
END

EXEC ListFlights @name = 'Mary Ann'
```

Database Vulnerabilities

A database is a collection of structured information and data. Any weakness in the database which can be exploited can be called a database vulnerability. Some of the most common vulnerabilities are extensive user and group privileges, audit trail tracking, SQL injection, buffer overflow, and database backups.

SQL Injection

This is one of the most usual attacks which uses malicious SQL code to manipulate databases in order to access secret information. This information can be either sensitive personal details or important organizational data. In some cases, a successful SQL injection exploit can even issue commands to the operating system. Successful SQL injection exploits can read sensitive data from the database, modify database data, carry out administrative database operations, recover the content of a file that is present in the database management system, and even execute administrative operations on the database.


Script
Injects malicious script
Attacker
Saves Malicious script into a database
User Request data from server
Vulnerable webpage server
Victim
Script
Data Containing the Malicious script is loaded
Script
Maliscious script may get

Techniques of SQL injection

There are mainly three types

1. In-Band Injection
   This is one of the easiest ways of exploitation. This type of attack takes place when the intruder uses the communication medium to perform the attack and gather confidential information. There are two ways to perform this injection.

   a. Error-based injection

This method carries out operations that trigger error messages to be produced by the database. These error messages may contain information that the attacker might utilize to learn more about the database's structure.

   b. Union-based injection
      This is the simplest way to receive more information from the injection. It uses the UNION operator to combine multiple statements and receives a single response.

2. Out-of-Band Injection
   When the attacker can't utilize the same channel to initiate the attack and collect data, or when a server is too slow or unstable for these tasks to be carried out, out-of-band SQLi is used. Only servers with commands that initiate DNS or HTTP requests are able to perform this.

3. Blind Injection
   In order to gain further insight into the server's architecture, the attacker delivers data payloads to the server and monitors how it responds and behaves. Blind SQLi is the name of this technique since the hacker cannot see information about the in-band attack because the data is not communicated from the website database to the attacker. Blind SQL injections are slower to perform but could be just as destructive because they depend on the server's behavior and reaction.

Impact of SQL injection

Users' login passwords, business secrets, or transaction information may be stolen from databases utilized by programs or apps that are insecure. SQL injection flaws must always be repaired, they shouldn't be left unpatched. The consequences of SQL injection can be analyzed through the CIA triad.

1. Confidentiality
   Most of the databases contain sensitive and confidential information, so when an injection is made the confidentiality of the information is lost.

2. Authentication
   It might be feasible to log in as another user without having previously known the password if weak SQL commands are utilized to verify usernames and passwords.

3. Authorization
   If the authorization aspect is affected in the database, the intruder might be able to get into the system as a different user.

4. Integrity
   When the injection attack is made, the sensitive information is read and possible alterations are made. So, the integrity factor is being compromised due to SQL injection.

Mitigation Techniques of SQL injection

It is very important to practice mitigation techniques to reduce the severity of the attack. The main two types of mitigation techniques are sanitization and validation.

1. Validation
   It is the process of checking if the input data meets a set of criteria or is submitted in the expected format. Testing the length, format, range, and permitted characters are all examples of validation tests.

2. Sanitization
   It is the process of modifying the input data to make sure that it is valid. This can be done in two methods namely backlisting and escaping. Blacklisting is removing or deleting unnecessary characters. Escaping is replacing unsuitable characters with safe ones.

Countermeasures to overcome SQL injection

1. Use stored Programs
   A stored procedure is a compiled SQL command that is preserved in a database server and can be used repeatedly by a third-party program. Since input arguments are always regarded as a real text value rather than a command when using a stored procedure, SQL injection is avoided.

2. Provide the least privilege
   To interface with a database, software needs user credentials in order to execute SQL commands like insert, update, search, delete, and drop. The application user should only be granted the necessary database privileges in order to reduce the impact of a SQL injection attack. As required, more rights should be granted. An SQL injection attack's potential impact is diminished with low database privileges.

3. Using Web Application Firewall
   Users are exposed to risks when they launch a website on the Internet since it is prone to assault. A web application firewall (WAF) is required to secure websites. It is an application firewall that protects HTTP applications.

Sample queries

The following query retrieves information about the specific item 999 or returns all item names and descriptions since 1 = 1 is always true.

```
SELECT ItemName, ItemDescription
FROM Items
WHERE ItemNumber = 999 OR 1=1
```

The following query changes SQL commands by using improperly filtered characters.

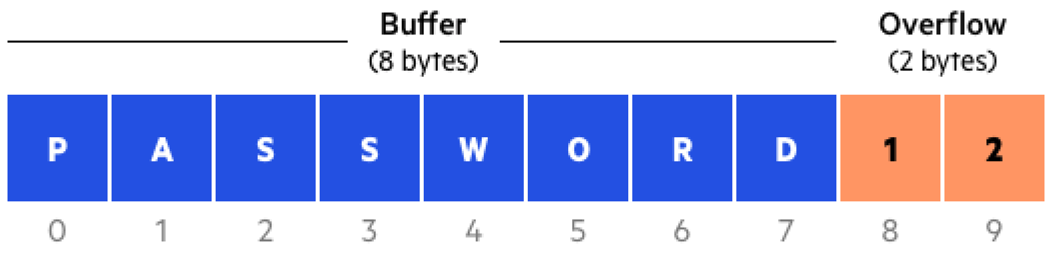
```
SELECT ItemName, ItemDescription
FROM Items
WHERE ItemNumber = 999; DROP TABLE USERS
```

The following query will merge nonrelated SELECT queries and receive data from different tables.

```
SELECT ItemName, ItemDescription
FROM Items
WHERE ItemID = '999' UNION SELECT Username, Password FROM Users;
```

Buffer Overflow

Data is temporarily stored in buffers, which are areas of memory, while it is being transported from one place to another. When the amount of data exceeds the memory buffer's storage capacity, a buffer overflow occurs. The application that is trying to copy the data to the buffer as a result overwrites nearby memory locations.



Techniques of Buffer Overflow

1. Stack Overflow
   When a computer software tries to use more memory from the call stack that has been assigned to it, a sort of buffer overflow error known as a stack overflow occurs. Local function variables and return address information are stored on the call stack, also known as the stack segment, which is a fixed-sized buffer used during program execution.
2. Heap Overflow
   It occurs when a portion of memory is allotted to the heap and data is loaded to this memory without the data's bounds being checked. This may result in the overwriting of some crucial heap data structures, which may then result in the overwriting of the virtual function table.

Impact of Buffer Overflow

A buffer overflow poses hazards to confidentiality, integrity, and availability by granting an attacker access to various regions of the internal memory and ultimately taking control of how the program is executed.
All forms of software are susceptible to buffer overflows. They frequently happen as a result of incorrect inputs or inadequate buffer space allocation. The software may perform erratically, produce inaccurate results, make memory access mistakes, or crash if the transaction overwrites executable code.

Mitigations of Buffer Overflow

1. Writing secure code
   The best defense against buffer overflow issues is to write secure code. Programmers must be aware of dangerous functions and avoid utilizing them whenever possible while writing programs in languages that are vulnerable to buffer overflow flaws. Although this is the greatest method of preventing buffer overflows, it could be challenging to upgrade older applications and programs that only run on older operating systems.

2. Data Execution Prevention
   Attackers frequently inject malicious code into locations like stacks and heaps when utilizing buffer overflows to gain unauthorized application execution. As a result, data execution prevention, a new buffer overflow mitigation strategy, is presented. Hardware and software levels can both enable data execution prevention.

3. Make use of compiler warnings
   Compilers frequently issue warnings and advise using secure alternatives when creating new software that uses vulnerable routines. These adjustments can be made by developers throughout the development stage fast.

4. Stack canaries
   Once stack-based buffer overflows gained popularity, compilers included new features to safeguard crucial stack data, like return addresses. These "canaries" are arbitrary values produced randomly during each program execution; they are stacked and often validated immediately prior to returning to the calling routines. The program will stop running and an error would be raised if there is a stack overflow and user input replaces the canary.

5. Address space layout randomization
   Changing up the base addresses of the library and other memory regions, such as the stack, makes it more difficult for an attacker to attack. Because return-oriented programming chains rely significantly on the locations of instructions from this library is loaded into the program, this makes it more difficult for an attacker to construct return-oriented programming chains.

Countermeasures of Buffer Overflow

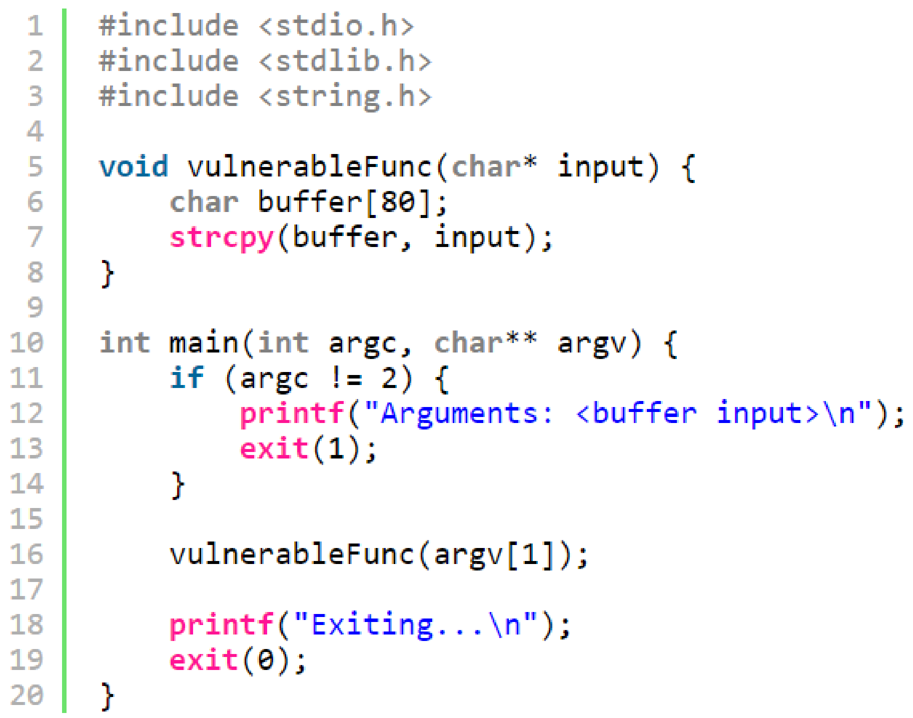
It is unquestionably useful to be able to identify buffer overflow problems in source code. However, removing them from a source code necessitates reliable detection and expertise with secure buffer handling procedures.

1. Use a different programming language
   Weaknesses in C are made possible by its lack of robust object type and direct memory access. Usually, languages that do not share these features are immune. To reduce overflow vulnerabilities, Java, Python, and.NET, among other languages and platforms, don't need any unique checks or modifications.

2. Use safe string handling functions
   The strcopy and strcat functions, copy a string into a buffer and add one buffer's contents to another. These two display the risky habit of not checking any buffer constraints and, if given enough bytes, writing past the buffer's boundaries. The following table shows the safe alternative for the functions

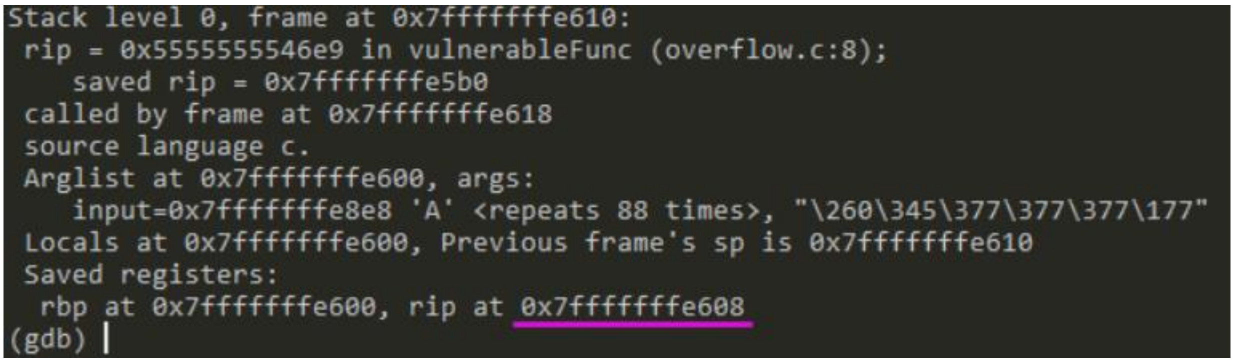
Function	**Alternative**
strcpy	strlcpy*
strcat	strlcat*
printf	sprintf*
gets	fgets

Sample code



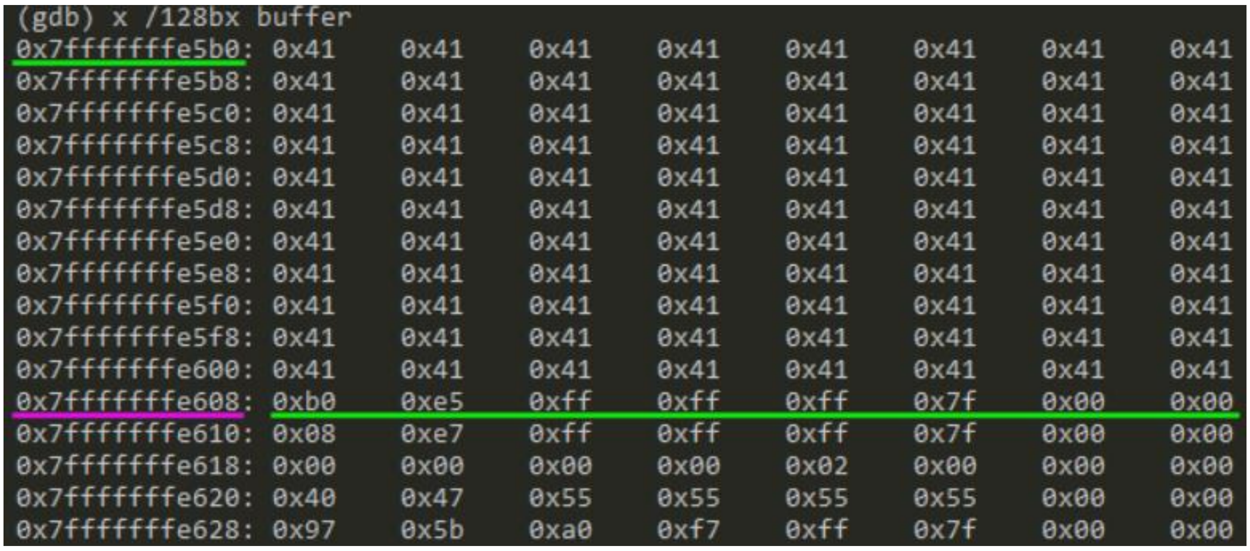
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void vulnerableFunc(char* input) {
    char buffer[80];
    strcpy(buffer, input);
}

int main(int argc, char** argv) {
    if (argc != 2) {
        printf("Arguments: <buffer input>\n");
        exit(1);
    }

    vulnerableFunc(argv[1]);

    printf("Exiting...\n");
    exit(0);
}
```

Information about the stack frame



```
Stack level 0, frame at 0x7fffffffe610:
 rip = 0x5555555546e9 in vulnerableFunc (overflow.c:8);
    saved rip = 0x7fffffffe5b0
 called by frame at 0x7fffffffe618
 source language c.
 Arglist at 0x7fffffffe600, args:
    input=0x7fffffffe8e8 'A' <repeats 88 times>, "\260\345\377\377\377\177"
 Locals at 0x7fffffffe600, Previous frame's sp is 0x7fffffffe610
 Saved registers:
  rbp at 0x7fffffffe600, rip at 0x7fffffffe608
(gdb)
```

Local Variables of Programs



```
(gdb) x /128bx buffer
0x7fffffffe5b0: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffe5b8: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffe5c0: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffe5c8: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffe5d0: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffe5d8: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffe5e0: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffe5e8: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffe5f0: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffe5f8: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffe600: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffe608: 0xb0    0xe5    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffe610: 0x08    0xe7    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffe618: 0x00    0x00    0x00    0x00    0x02    0x00    0x00    0x00
0x7fffffffe620: 0x40    0x47    0x55    0x55    0x55    0x55    0x00    0x00
0x7fffffffe628: 0x97    0x5b    0xa0    0xf7    0xff    0x7f    0x00    0x00
```

s