97`a`+20`21-1`=117=0x75`HEX`

Байт-стаффинг - это техника, используемая в протоколах передачи данных для того, чтобы избежать конфликтов между специальными символами (например, флагами начала/конца кадра) и данными. В вашем коде:

def byte\_stuffing(self, data):

stuffed\_data = data.replace(b'@u', b'\x1B\x01')

return stuffed\_data

def unstuffing(self, data):

unstuffed\_data = data.replace(b'\x1B\x01', b'@u')

return unstuffed\_data

Алгоритм байт-стаффинга здесь довольно прост:

Если в данных встречается последовательность байтов @u (которая используется как флаг кадра), она заменяется на \x1B\x01.

При приеме данных происходит обратная операция: \x1B\x01 заменяется обратно на @u.

Это нужно, чтобы предотвратить случайное появление флага @u внутри данных, что могло бы привести к неправильной интерпретации структуры кадра.

2. Заполнение пакета:

if len(data) < 21:

# Если данные короче 21 байта, добавляем терминатор и заполняем нулями

padded\_data = data + TERMINATOR + b'\x00' \* (20 - len(data))

else:

# Если данные ровно 21 байт, оставляем как есть

padded\_data = data

Если данные короче 21 байта:

1. Добавляется терминатор (TERMINATOR, который равен b'\x04').

2. Оставшееся пространство заполняется нулевыми байтами (b'\x00').

3. 0x00 и его интерпретация:

0x00 - это нулевой байт. В ASCII это непечатный символ NUL.

Когда мы используем 0x00 для заполнения, это просто способ обеспечить фиксированную длину поля данных.

Чтобы отличить нули заполнения от нулей в данных, используется терминатор. Всё после терминатора считается заполнением.

4. Структура кадра:

frame = flag + destination\_address + source\_address + stuffed\_data + fcs

flag: b'@u' (2 байта)

destination\_address: b'\x00' (1 байт)

source\_address: номер COM-порта (1 байт)

stuffed\_data: до 21 байта данных (включая терминатор и заполнение)

fcs: b'\x00' (1 байт) - контрольная сумма (в данном случае не используется)

5. Прием данных:

if TERMINATOR in unstuffed\_data:

actual\_data = unstuffed\_data.split(TERMINATOR)[0]

else:

actual\_data = unstuffed\_data

При приеме данных ищется терминатор. Если он найден, берутся только данные до терминатора. Это позволяет отделить реальные данные от заполнения.

Основные шаги алгоритма байт стаффинга

Определение специальных символов: Выбираются символы, которые могут быть интерпретированы как управляющие (например, символ @u).

Замена специальных символов: Перед отправкой данных специальные символы заменяются на последовательность байтов, которая не будет интерпретирована как управляющая.

Обратная замена: После приема данных специальные последовательности байтов заменяются обратно на исходные символы.

Пример работы

Ввод данных: Пользователь вводит строку "Hello, @u world!".

Преобразование в байты: Строка преобразуется в байты: b'Hello, @u world!'.

Заполнение данных: Строка заполняется нулями до 21 байта: b'Hello, @u world!\x00\x00\x00\x00\x00'.

Байт стаффинг: Символ b'@u' заменяется на b'\x1B\x01': b'Hello, \x1B\x01 world!\x00\x00\x00\x00\x00'.

Создание пакета: Пакет создается с использованием метода create\_frame.

Структура пакета

Пакет будет иметь следующую структуру:

Flag: b'@u' (2 байта)

Destination Address: b'\x00' (1 байт)

Source Address: bytes([int(self.ser\_out.port[3:])]) (1 байт)

Data: b'Hello, \x1B\x01 world!\x00\x00\x00\x00\x00' (21 байт)

FCS: b'\x00' (1 байт)

Пример вывода

В окне состояния будет отображена структура отправленного пакета:

Отправлен кадр: 0x40 0x75 0x00 0x03 0x48 0x65 0x6C 0x6C 0x6F 0x2C 0x20 0x1B 0x01 0x20 0x77 0x6F 0x72 0x6C 0x64 0x21 0x00 0x00 0x00 0x00 0x00 0x00 0x00

В окне вывода будет отображена строка "Hello, @u world!" после обратного преобразования символа b'\x1B\x01' обратно в b'@u'.

Что изменится при проверке работы

При проверке работы байт стаффинга вы увидите, что:

Символ b'@u' был заменен на b'\x1B\x01' перед отправкой.

Символ b'\x1B\x01' был заменен обратно на b'@u' после приема.

Это показывает, что байт стаффинг работает корректно и избегает путаницы с управляющими символами.