



CS177 Computer Security Discussion

Spring 2020 - Week 2

Dongyu (Hector) Meng
Apr 6th



Logistics

- Discussion: Monday 5-5:50pm
- One discussion section for all students
- Discussions will be recorded for review
- Discussion QA: After lecture till 6:50pm
 - General questions about discussions and lectures
 - For hands-on help with programming assignments attend lab hours



Project 1: Simple Network Client

- Implement a very simple (key, value)-storage system following to protocol specification
- Server (S) and client (C) exchange messages in rounds
- Message format
- Due end of day next Wednesday



Project 1: Simple Network Client

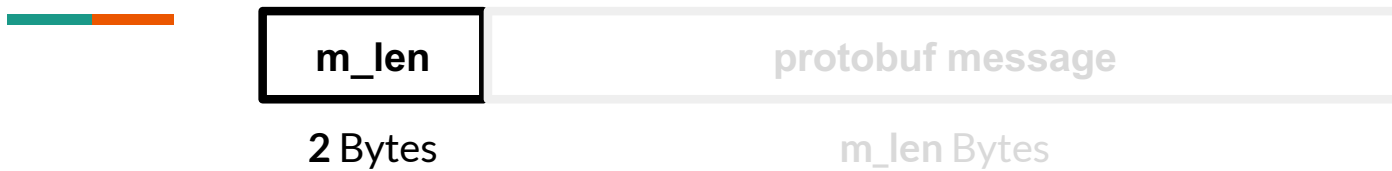
1. **Task Request Message (C->S):** Asking for a task.
2. **Task Message (S->C):** Specifying a specific task that C must carry out.
3. **Task Response Message (C->S):** It is sent once the client has performed the given task, and it contains the task's result or status.

You'll need to initiate another round from the client side after a round is done.

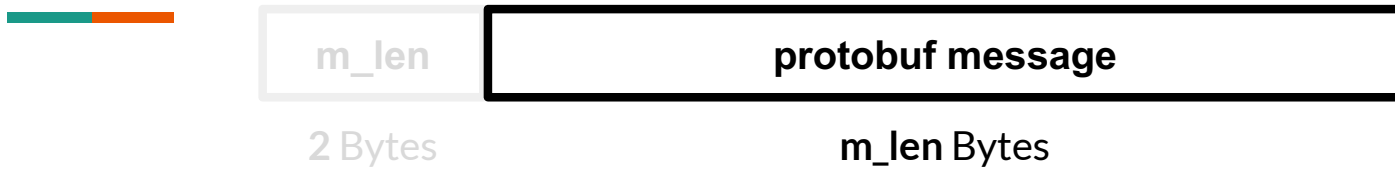


2 Bytes

m_len Bytes



- Unsigned short, 2 bytes
- Big-endian
 - `big_endian_pack(4) == '\x00\x04'`
 - For most computers, the native endianness is little-endian
 - `big_endian_unpack(small_endian_pack(4)) == ?`
 - **1024**, and your program will hang there...
 - `struct` if you're using Python



- Type
- Other optional fields



Protobuf

*Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for **serializing structured data** – think XML, but smaller, faster, and simpler. You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using **a variety of languages**.*



Protobuf

- XML (size, speed/readability, structure interleaving)
- JSON (size, speed, type/readability)
- Choose tools that suit your needs.



```
syntax = "proto2";
```

```
message Person {  
  required string name = 1;  
  required int32 id = 2;  
  optional string email = 3;  
}
```

protoc




Or C++, C#, Dart, Java, Go...

Python class



import

```
# Your Python application  
from generated_python_file import Person  
  
person = Person()  
person.name = "Sue Maslino"  
person.id = 177  
data = person.SerializeToString()  
...  
recv_end_person = Person()  
recv_end_person.ParseFromString(data)
```



```
syntax = "proto2";

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
}
```

Most of the times you, as a programmer, have control over both the serialization and deserialization ends, so the `.proto` file is shared by both ends.

In this project, however, you don't know about the server side definition for sure. Interesting...



```
syntax = "proto2";
```

```
message Message_1 {  
    required string first = 1;  
    required string second = 2;  
}
```

```
message Message_2 {  
    required string second = 1;  
    required string first = 2;  
}
```

```
m1 = Message_1()  
m1.first = "FIRST"  
m1.second = "SECOND"  
data = m1.SerializeToString()
```

```
m2 = Message_2()  
m2.ParseFromString(data)  
print(f"m2 first, tag 2: {m2.first}")  
print(f"m2 second, tag 1: {m2.second}")
```

```
* bed @mdy-personal INSERT ~/discussion  
python example.py  
m2 first, tag 2: SECOND  
m2 second, tag 1: FIRST
```



```
syntax = "proto2";
```

```
message Message_1 {  
    required string name = 1;  
    optional string gender = 2;  
    Optional string addr = 3;  
}
```

```
message Message_2 {  
    required string name = 1;  
    Optional string addr = 2;  
}
```

```
m1 = Message_1()  
m1.name = "someone"  
m1.addr = "somewhere"  
data = m1.SerializeToString()
```

```
m2 = Message_2()  
m2.ParseFromString(data)  
print(m2.name)  
print(m2.addr)
```



Protobuf

- The abstraction (structure of message) may evolve over time
- e.g. serialization end: past, deserialization end: now
- So these problems are not all imaginary



Server-Side Message Definition

- First of all, there are more than one reasonable definition!
- Starts with type as tag 1 (or field number 1)
- Each additional field of that message type takes the next tag in turn
- Think about how you can match it from the client side
- Think about how you can reuse definitions and save some typing



Socket Programming with Python

- The `socket` library
 - `socket.socket(socket.AF_INET, socket.SOCK_STREAM)`
- Receive full message
 - `sock.recv(4096)` does not give you 4096 bytes for sure
 - Keep receiving until you have the full message
 - Doesn't really happen for packets as small as we have in this proj
 - We make sure that it will happen tho...



Socket Programming with Python

- Straightforward wrapper of C implementation
- socket as fd
- Communication between user land and kernel land
- What happens when you call `sock.recv(4096)`?
- What if the kernel buffer is full but the other end still sends more data?



Socket Programming with Python

<https://realpython.com/python-sockets/>