ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

HW3-HW4 Overview

COM-402: Information Security and Privacy

De-Anonymization Attacks

I Know What You Did Last Summer

De-anonymization Attacks

1a - Rating on same date:

1. Create `user_name_to_hash` and `movie_hash_to_name` dictionaries
2. Iterate over `all anonymized ratings`
3. Iterate over `all public ratings`
4. If the dates of the anonymized and public ratings match → add an entry to both dictionaries

De-anonymization Attacks

1b - Frequency attack

1. Sort both `movie_name` and `movie_hash` according to number of ratings
    a. This will give you a possibility to map between movies: `movie_name[0] <-> movie_hash[0]`
2. Create a list of the hashes of all publicly rated movies
3. Search in the anonymized database for a user that has ratings for all these hashes
4. Use the `movie_hash -> movie_name` to find all movies of the user

De-anonymization Attacks

<u>**1c - Randomized ratings**</u>

1. Create a *`user_ratings`* vector and put a *`1`* for each date a user rated a movie in the public database
2. Convolute *`user_ratings`* with *`[1, 2, 3, 4, …, 14, 13, 12, 11, …, 1]`* to reflect the probability-distribution
3. For each hashed user in the anonymized database, create a vector as in step 1: *`hashed_user_ratings`*
4. Find the hashed user with the maximum dot-product of its *`hashed_user_ratings`* vector with the public vector *`user_ratings`*

1c - Randomized ratings

0	0	1	2	**3**	2	1	0	0	0

*

0
1
0
0
1
0
0

De-anonymization Defense (HW4/Ex1)

1. Drop all data that is not important to the application
2. If linkable data are important for later, use a salt to hash them

Password hashing - hw3 & hw4 (ex2)

- **Registration**:
    - User creates an account
    - User’s password is hashed and stored in db
- **Login**:
    - User enters a password which is then hashed
    - Hashed password is checked against the hash stored in db

Password hashing - hw3 & hw4 (ex2)

- **Password cracking:**
  - Brute-force attacks → simple but have limitations
  - Dictionary attacks → much more effective than BF attacks
  - Lookup tables → extension of dictionary attack
  - Rainbow tables → lookup table with time-memory trade-off

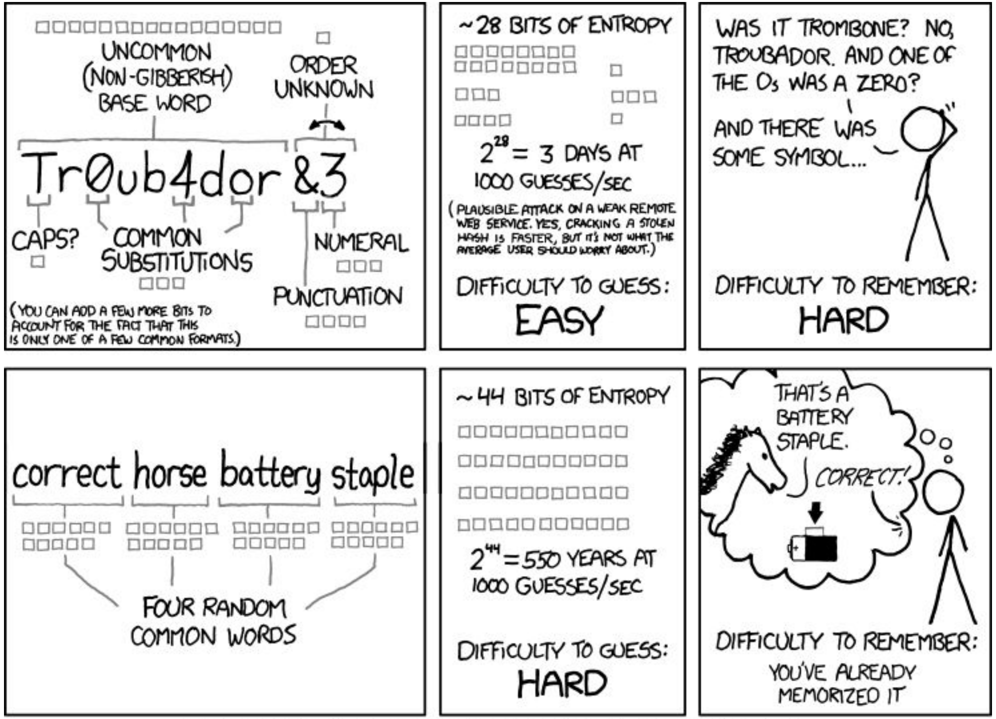
Password hashing - hw3 & hw4 (ex2)

- Lookup and rainbow tables are powerful because each password is hashed in the same way
- **Salting** → append a random string to the password before hashing
- Salting makes rainbow/lookup tables ineffective, so cracking bunch of password quickly is not practical
- Common mistakes: **salt reuse** and **short salt**
- Salt and pepper (see Dropbox’s post)

Password hashing - hw3 & hw4 (ex2)

- Attacker can still use a dictionary or brute-force attack to crack a single password hash
- GPUs, ASICs, etc. can compute many hashes per second
- Slow hash functions: *`bcrypt, scrypt, PBKDF2`*
- This slows down the attacks, but still doesn't prevent them
    - Keyed hashes - HMAC

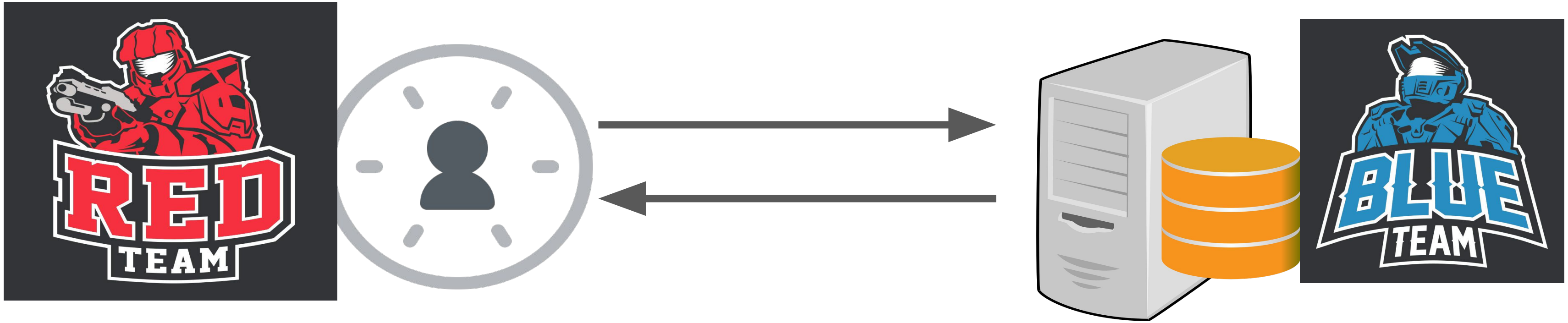
Password hashing - hw3 & hw4 (ex2)


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor &3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

SQL Injection and Defense - hw3 & hw4 (ex3)

- User input from client-facing programs must be prepared before they are used in database queries
  - What if these programs use the input as given by user? ***SQL injection attacks in hw3***
  - How to <u>*prepare*</u> user input? ***Parameterized queries (prepared statements) in hw4***



SQL Injection - hw3 ex3a

- **Give as user input a string that executes multiple queries: the intended one and another one to fetch interesting data**
    - Where to attack? Roam around to find a page that queries the db using user input **and outputs db results**
    - Here /personalities queries the db we need, and it queries ‘id’ in table users: /personalities?id=2
    - Guesswork: program probably executes: SELECT id,name from personalities WHERE id = ` “ + user_input + “`“
    - Example of attack string: SELECT id,name from personalities WHERE id = ` “ + 2` UNION SELECT name,message from contact_messages WHERE mail LIKE `%james@bond% + “`“

SQL Injection - hw3 ex3b

- **Give as user input a string that executes multiple queries: the intended one and another one to fetch interesting data**
  - Find inspector_derrick’s password length - need true / false response.
  - Where to attack? /messages inspects the db we need
  - SELECT name,message FROM contact_messages WHERE name LIKE `” + % `AND LENGTH ((“ + **pwdq**+ “)) = `” + str(some_int) + “` AND `1`=`1” + “`”
  - Pwdq is an sql query: SELECT password FROM users WHERE name LIKE `inspector_derrick`”
  - Why use AND `1`=`1` ?
  - Guess password (similarly to guessing length) character by character using SUBSTRING and a given index

SQL Injection Defense

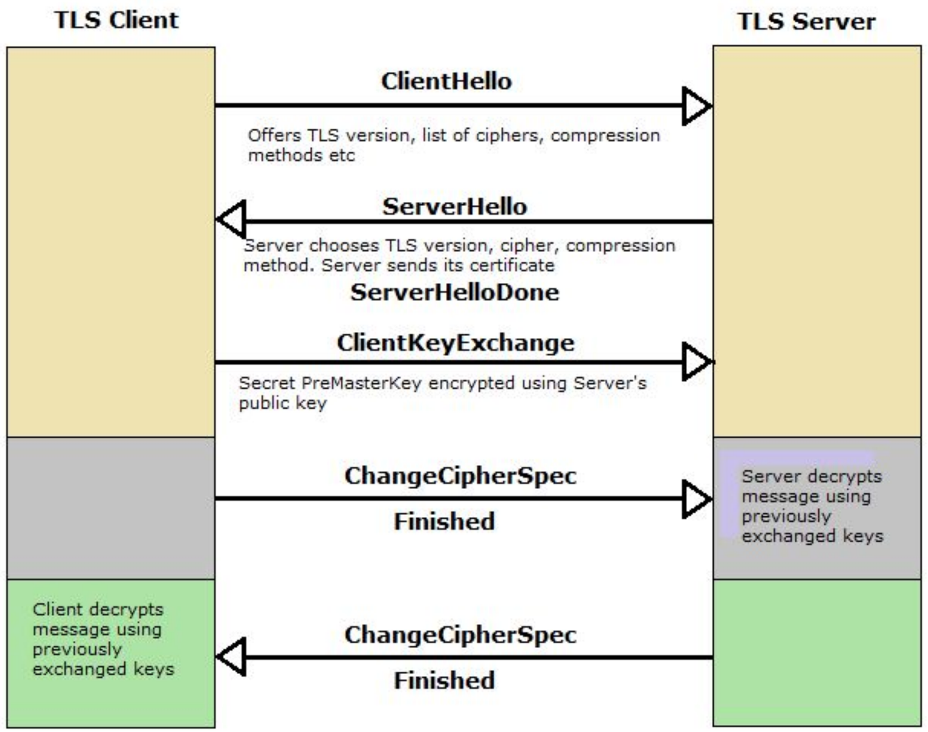
- Use parameterized/prepared queries for user input
  - Parameterized queries use placeholders instead of embedded inputs
  - `Define db cursor: db.cursor()`
  - `cursor.execute(sql,param)`
- Escape user input - prone to errors and might not cover all cases.
- Other techniques
  - Whitelist Input Validation
  - Least Privilege

Downgrade Dance (HW3/ex4) - Ceyhun

Downgrade Dance

- SSL → obsolete and insecure & TLS → SSL’s successor
- Backwards compatibility w/ legacy systems
- TLS client implementations do not rely on the TLS version negotiation mechanism alone:
  - Intentionally reconnect using a downgraded protocol if handshake fails
  - Retries with lower TLS version

TLS Handshake & How to downgrade



TLS Client

TLS Server

ClientHello

Offers TLS version, list of ciphers, compression methods etc

ServerHello

Server chooses TLS version, cipher, compression method. Server sends its certificate

ServerHelloDone

ClientKeyExchange

Secret PreMasterKey encrypted using Server's public key

ChangeCipherSpec

Finished

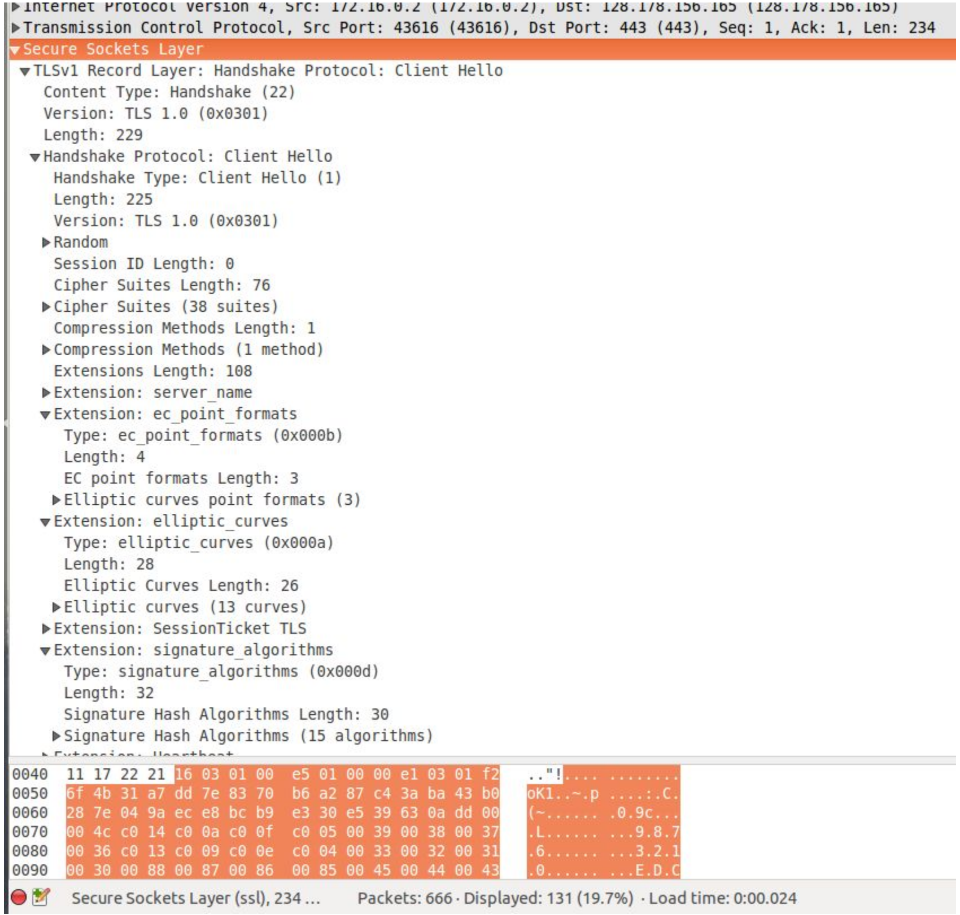
Server decrypts message using previously exchanged keys

Client decrypts message using previously exchanged keys

ChangeCipherSpec

Finished

Client Hello

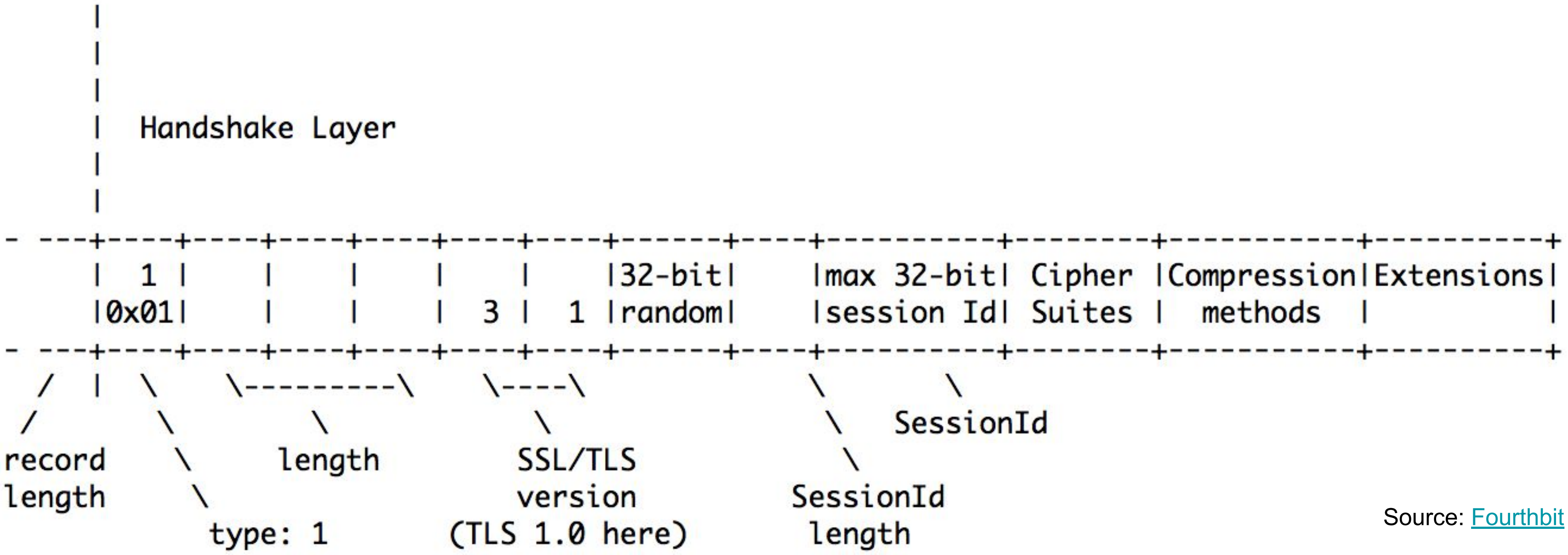


```
▶Internet Protocol Version 4, Src: 172.16.0.2 (172.16.0.2), Dst: 128.178.156.165 (128.178.156.165)
▶Transmission Control Protocol, Src Port: 43616 (43616), Dst Port: 443 (443), Seq: 1, Ack: 1, Len: 234
▼Secure Sockets Layer
 ▼TLSv1 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 229
  ▼Handshake Protocol: Client Hello
     Handshake Type: Client Hello (1)
     Length: 225
     Version: TLS 1.0 (0x0301)
   ▶Random
     Session ID Length: 0
     Cipher Suites Length: 76
   ▶Cipher Suites (38 suites)
     Compression Methods Length: 1
   ▶Compression Methods (1 method)
     Extensions Length: 108
   ▶Extension: server_name
   ▼Extension: ec_point_formats
      Type: ec_point_formats (0x000b)
      Length: 4
      EC point formats Length: 3
    ▶Elliptic curves point formats (3)
   ▼Extension: elliptic_curves
      Type: elliptic_curves (0x000a)
      Length: 28
      Elliptic Curves Length: 26
    ▶Elliptic curves (13 curves)
   ▶Extension: SessionTicket TLS
   ▼Extension: signature_algorithms
      Type: signature_algorithms (0x000d)
      Length: 32
      Signature Hash Algorithms Length: 30
    ▶Signature Hash Algorithms (15 algorithms)
   ▶Extension: Heartbeat

0040  11 17 22 21 16 03 01 00  e5 01 00 00 e1 03 01 f2   .."!.... ........
0050  6f 4b 31 a7 dd 7e 83 70  b6 a2 87 c4 3a ba 43 b0   oK1..~.p ....:.C.
0060  28 7e 04 9a ec e8 bc b9  e3 30 e5 39 63 0a dd 00   (~...... .0.9c...
0070  00 4c c0 14 c0 0a c0 0f  c0 05 00 39 00 38 00 37   .L...... ...9.8.7
0080  00 36 c0 13 c0 09 c0 0e  c0 04 00 33 00 32 00 31   .6...... ...3.2.1
0090  00 30 00 88 00 87 00 86  00 85 00 45 00 44 00 43   .0...... ...E.D.C

Secure Sockets Layer (ssl), 234 ...    Packets: 666 · Displayed: 131 (19.7%) · Load time: 0:00.024
```

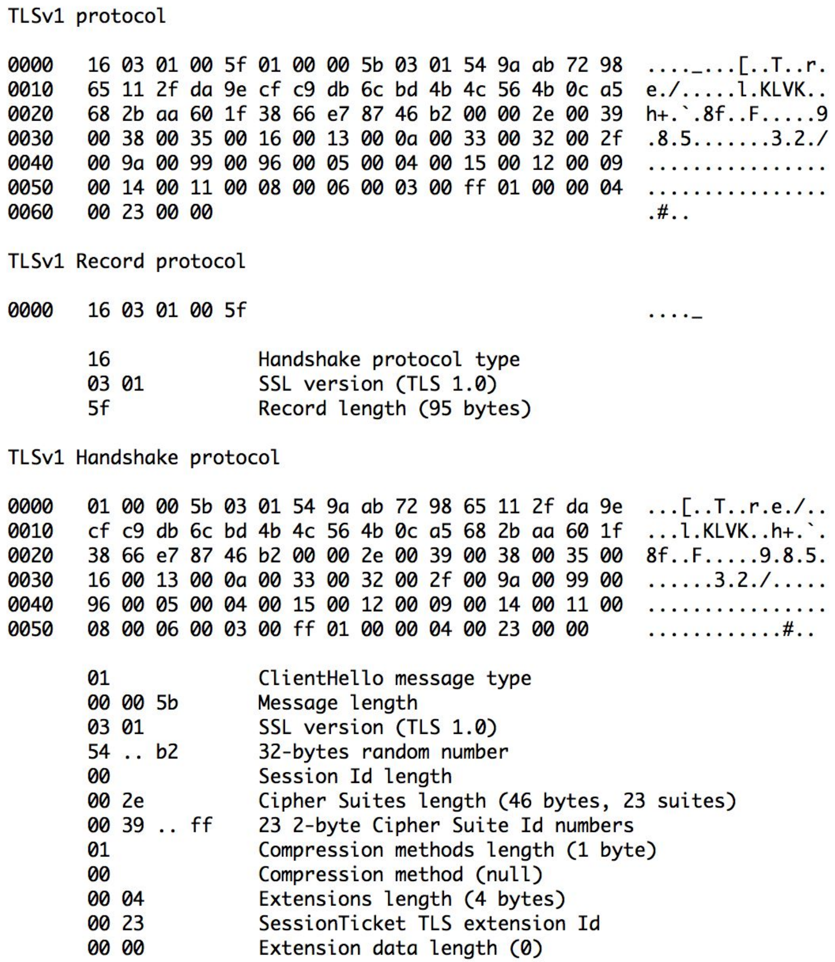
ClientHello



```
     |
     |
     |
     |  Handshake Layer
     |
     |
- ---+----+----+----+----+----+----+------+----+----------+--------+-----------+----------+
     |  1 |    |    |    |    |    |32-bit|    |max 32-bit| Cipher |Compression|Extensions|
     |0x01|    |    |    |  3 |  1 |random|    |session Id| Suites |  methods  |          |
- ---+----+----+----+----+----+----+------+----+----------+--------+-----------+----------+
  /  |  \    \---------\    \----\             \       \
 /       \        \            \                \   SessionId
record    \     length        SSL/TLS            \
length     \                  version         SessionId
            type: 1       (TLS 1.0 here)       length
```

Source: Fourthbit

ClientHello



```
TLSv1 protocol

0000   16 03 01 00 5f 01 00 00 5b 03 01 54 9a ab 72 98   ...._...[..T..r.
0010   65 11 2f da 9e cf c9 db 6c bd 4b 4c 56 4b 0c a5   e./.....l.KLVK..
0020   68 2b aa 60 1f 38 66 e7 87 46 b2 00 00 2e 00 39   h+.`.8f..F.....9
0030   00 38 00 35 00 16 00 13 00 0a 00 33 00 32 00 2f   .8.5.......3.2./
0040   00 9a 00 99 00 96 00 05 00 04 00 15 00 12 00 09   ................
0050   00 14 00 11 00 08 00 06 00 03 00 ff 01 00 00 04   ................
0060   00 23 00 00                                       .#..

TLSv1 Record protocol

0000   16 03 01 00 5f                                    ...._

       16             Handshake protocol type
       03 01          SSL version (TLS 1.0)
       5f             Record length (95 bytes)

TLSv1 Handshake protocol

0000   01 00 00 5b 03 01 54 9a ab 72 98 65 11 2f da 9e   ...[..T..r.e./..
0010   cf c9 db 6c bd 4b 4c 56 4b 0c a5 68 2b aa 60 1f   ...l.KLVK..h+.`.
0020   38 66 e7 87 46 b2 00 00 2e 00 39 00 38 00 35 00   8f..F.....9.8.5.
0030   16 00 13 00 0a 00 33 00 32 00 2f 00 9a 00 99 00   ......3.2./.....
0040   96 00 05 00 04 00 15 00 12 00 09 00 14 00 11 00   ................
0050   08 00 06 00 03 00 ff 01 00 00 04 00 23 00 00      ............#..

       01             ClientHello message type
       00 00 5b       Message length
       03 01          SSL version (TLS 1.0)
       54 .. b2       32-bytes random number
       00             Session Id length
       00 2e          Cipher Suites length (46 bytes, 23 suites)
       00 39 .. ff    23 2-byte Cipher Suite Id numbers
       01             Compression methods length (1 byte)
       00             Compression method (null)
       00 04          Extensions length (4 bytes)
       00 23          SessionTicket TLS extension Id
       00 00          Extension data length (0)
```

Source: Fourthbit

How to force the server to downgrade?

- Configure `iptables` and `NFQUEUE` (`--dport 443`)
  - `iptables -t nat -A POSTROUTING -j MASQUERADE`
  - `iptables -A FORWARD -s 172.16.0.2 -p tcp --dport 443 -j NFQUEUE --queue-num 0`
  - `route add default gateway 172.16.0.3`
- Look for a `ClientHello`:
  - If the version bytes are `x0303 (TLS 1.2)` or `x0302 (TLS 1.1)`
    - Drop packet
    - Send a `FIN-ACK` packet to the server
    - **Set IP_SRC_ADDR to `172.0.0.3` (aka attacker)**
    - You could use the dropped packet as a basis and only change the `FLAGS` and `IP_SRC_ADDR`
  - If the version bytes are `x0301 (TLS 1.0)`, let it through

Preventing Protocol Downgrade Attacks

- Unnecessary protocol downgrades are undesirable

- **Signaling Cipher Suite Value (SCSV):**
  - Present in the `ClientHello` message as a backwards-compatible signal to server
  - Tell the server that the connection should only be established if the highest protocol version supported by the server is identical to or lower than that of what it sees in the `ClientHello`
  - Server responds with inappropriate fallback

Secure Nginx and Friends

HTTP Strict Transport Security (HSTS)

- A mechanism to tell web browsers that should only connect using HTTPS
- Basically, a field in the HTTP response header: `Strict-Transport-Security`
- Prevents HTTPS-to-HTTP downgrade attacks
- Does not redirect by itself, only notify

HW4/ex4 - steps to solve

- **Goal:** run NGINX with your generated certificate
- Create two server entries in `default.conf` to listen on port `80 (HTTP)` and `443 (HTTPS)`
- Configure redirect from HTTP to HTTPS with the response code 301
- Add the following headers using `add_header`:
    - `Strict-Transport-Security "max-age=315536000; includeSubDomains" always;`
    - Only TLSv1.2: `ssl_protocols TLSv1.2;`
    - Another field to protect against XSS: `X-XSS-Protection "1; mode=block";`