Network Security Challenge 09 - Hard

In this challenge we introduce the new TETRA9000 protocol. It is losely based on the real-world radio standard TETRA[1], however over 9000 times better. The TETRA protocol was recently analyzed and broken by security researchers from MidnightBlue, who published a paper[2] and held a talk[3] on the topic. *It is highly recommended to look into either or both because the exploits are very much inspired by techniques discovered by MidnightBlue.*

Our protocol works as follows: A client (*mobile station*, also called Alice in code) connects to a server (*base station*, also called Bob in code). Both share a secret pre-shared key, which is used for symmetric cryptography (ChaCha20). The client first sends a `HELLO` message with the IV of the keystream set to zero as default. The server replies firstly with a `TIMESTAMP` message, which tells the time in the beginning of the connection and is used as the starting point for the packet counter. This packet counter is incremented on every acknowledged packet and is used as IV for the cipher. After the timestamp message, the server sends (using the new IV) a `PASSWD` message, which contains a random 4 Byte password. The client acknowledges the reception of this message with an `ACK` message. The packet exchange ends here, but there are more packet types described below.

As this protocol emulates an over-the-air interface, you can take a Man-in-the-Middle position. Simply connect to port `20209` for the client and `20309` for the server on `netsec.net.in.tum.de`. You can find the source code on the scoreboard as usual.

IMPORTANT: PLEASE READ THE FOLLOWING VERY CAREFULLY

- Please develop your exploit locally. **ONLY SUBMIT IT WHEN YOU ARE READY!** Exploits might take some time on this challenge and we don't want you to block the autograder queue.
- Please don't overwhelm our server and build in `time.sleep(0.1)` between your packets. This also (usually) makes sure that packets are delivered invidually, which is important for post packets.
- Make sure to have a command for `flag` in your `PATH` or modify the server code.
- Please go watch either the talk or read the paper at the relevant points, it will help you.
- If you believe that you have found an unintended solution, please write to `netsec@net.in.tum.de`.
- Your exploit may take some time, the reference exploit took 180s on average, the timeout is set to 400s.

Your task is, obviously, to decrypt the password and retrieve the flag.
Good luck!

[1] https://en.wikipedia.org/wiki/Terrestrial_Trunked_Radio
[2] https://www.usenix.org/system/files/usenixsecurity23-meijer.pdf
[3] https://media.ccc.de/v/37c3-11761-all_cops_are_broadcasting

Message formats: all packets begin with 4 bits of type and 4 bits of flag. The next bytes and flags are specific to the message type.

- Type `0x1 HELLO`: No flags, no payload, signals start of connection to server. Only accepted by server.
- Type `0x2 TIMESTAMP`: No flags, payload is 4 byte unsigned integer encoding a timestamp-like nonce.
- Type `0x3 PASSWD`: No flags, payload is 4 byte password characters. Triggers `ACK` by client. Increases packet counter on client.
- Type `0x4 ACK`: No flags, no payload. Signals reception of message by client. Only accepted by server. Increases packet counter on server.
- Type `0x5 DATA`: Flag `checksum_available` to signal checksum (CRC32 checksum truncated to one byte) after payload. After header comes 1 byte unsigned integer to signal length, after which comes variable length payload. Only accepted by client, triggers `ACK`. Increases packet counter on client.
- Type `0x6 PING`: No flags, no payload, triggers `ACK` by both server and client without increasing packet counter.
- Type `0x7 FLAG`: No flags, payload is 42 bytes of unencrypted fresh flag. Only sent by server, not accepted by client.

It is advisable to have a Linux system at hand for the challenges. Our servers and clients are written in Python.