Assignment 2
Harigovind Raghunath 22B0057

The script is written in Lua and works by exploiting the fact that Tor SNI is recognisable to a certain extent.
The script finds Tor packets using SNI(explained later) and then marks it by showing Tor! In the protocol field. If the steps in the README are followed for coloring rules then the packets will also have a white background with red text for easy identification. Furthermore a protocol subtree is added with the following info-
Tor Traffic Detected
    Tor Traffic Detected: True
    Detection Type: Known Tor IP

Once a Tor packet is identified from its TLS handshake using SNI then all traffic to and from that **destination IP** is marked as Tor traffic as we have essentially identified the guard node that we have connected to.

Detecting using Tor SNI

The following is a snippet of the source code from Tor which deals with generating the SNI(found in the file tor\src\lib\tls\tortls_nss.c) -

```
 if (!is_server) {
    /* Set a random SNI */
    char *fake_hostname = crypto_random_hostname(4,25,
"www.",".com");
    SSL_SetURL(tls->ssl, fake_hostname);
    tor_free(fake_hostname);
  }
  SECStatus s = SSL_ResetHandshake(ssl, is_server ?
PR_TRUE : PR_FALSE);
  if (s != SECSuccess) {
    tls_log_errors(tls, LOG_WARN, LD_CRYPTO, "resetting
handshake state");
  }
```

```
crypto_random_hostname(int min_rand_len, int max_rand_len, const char *prefix,
                       const char *suffix)
{
  char *result, *rand_bytes;
  int randlen, rand_bytes_len;
  size_t resultlen, prefixlen;

  if (max_rand_len > MAX_DNS_LABEL_SIZE)
    max_rand_len = MAX_DNS_LABEL_SIZE;
  if (min_rand_len > max_rand_len)
    min_rand_len = max_rand_len;

  randlen = crypto_rand_int_range(min_rand_len, max_rand_len+1);

  prefixlen = strlen(prefix);
  resultlen = prefixlen + strlen(suffix) + randlen + 16;

  /* (x+(n-1))/n is an idiom for dividing x by n, rounding up to the nearest
   * integer and thus why this construction. */
  rand_bytes_len = ((randlen*5)+7)/8;
  if (rand_bytes_len % 5)
    rand_bytes_len += 5 - (rand_bytes_len%5);
  rand_bytes = tor_malloc(rand_bytes_len);
  crypto_rand(rand_bytes, rand_bytes_len);

  result = tor_malloc(resultlen);
  memcpy(result, prefix, prefixlen);
  base32_encode(result+prefixlen, resultlen-prefixlen,
                rand_bytes, rand_bytes_len);
  tor_free(rand_bytes);
  strlcpy(result+prefixlen+randlen, suffix, resultlen-(prefixlen+randlen));

  return result;
```

From this we can basically infer the following things -

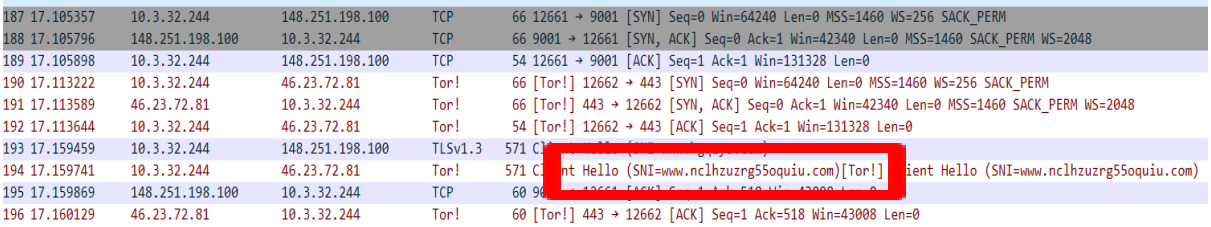
- The SNI is random
- It always has the prefix “www.” and ends with “.com”
- It uses base32 encoding to generate the random string
- Tor generates a random sequence of bytes with a length between 4 and 25 bytes

Base32 encoding converts 5 bits of data into 1 character. This means that every byte (8 bits) of random data produces 8/5 = 1.6 Base32 characters. 4 bytes of random data → (4 × 8) ÷ 5 = 6.4 → rounded up to **7 Base32 characters**. 25 bytes of random data → (25 × 8) ÷ 5 = **40 Base32 characters**.

Base32 encoding works in blocks of 5 bytes (40 bits), which encode into 8 Base32 characters. However, if the random byte length is not a multiple of 5, the last block will be a partial block. Tor does not use padding in its Base32 encoding, so the length of the Base32 string depends on the number of bits in the last block.
If the random byte length is a multiple of 5 (e.g., 5, 10, 15, 20, 25 bytes), the Base32 string will have a length that is a multiple of 8 (e.g., 8, 16, 24, 32, 40 characters).
If the random byte length is not a multiple of 5, the last block will encode fewer than 8 characters. The valid lengths for the Base32 string depend on how many bits are left in the last block:
1 byte (8 bits): Encodes to 2 Base32 characters.
2 bytes (16 bits): Encodes to 4 Base32 characters.
3 bytes (24 bits): Encodes to 5 Base32 characters.
4 bytes (32 bits): Encodes to 7 Base32 characters.
Combining full and partial blocks, the valid lengths for the Base32 string are:7, 8, 15, 16, 23, 24, 31, 32, 39, 40 characters.

So using the above info for each packet the script checks if its a potential Tor packet by -

- The dissector calls sni_field(), which retrieves the TLS Server Name Indication (SNI) value (if present) from the packet
- The function is_tor_sni_pattern(sni_str) is then invoked with the SNI string. This function performs several checks:
    - It verifies that the SNI fits the pattern www.<domain>.com
    - It ensures the <domain> part (the random Base32-encoded segment) falls within a valid length range (7–40 characters)
    - It iterates through each character of the domain part to guarantee that only valid Base32 characters (letters a–z and digits 2–7) are present.
    - It confirms that the length matches one of the expected values or follows a valid partial block pattern in Base32 encoding.
- If all these conditions are met, it is highly likely that the SNI was generated by Tor, so is_tor_sni_pattern returns true.
- The destination IP is marked as a Tor IP



The client Hello with the random SNI can be seen here