Day 4 / ASR-EHD – WriteUp by AV1ct0r

Peter is a little bit paranoid: he always uses encrypted connections. To be sure algorithms are secure Peter uses his own client. He even gave us a traffic dump which was made while using his custom client. Is Peter's connection really secure?

https://hackquest.zeronights.org/downloads/task4/8Jdl3f_client.tar
https://hackquest.zeronights.org/downloads/task4/d8f3ND_dump.tar

1. Открываем файлик client в IDA Pro и видим, что он умеет скачивать часть файла flag.jpg с сервера https://ssltest.a1exdandy.me:443/ . Какую часть файла качать (с какого по какой байт) берется из командной строки.

```
signed __int64 __fastcall main(int argc, char **argv, char **a3)
{
  size_t v4; // rsi
  __int64 v5; // ST48_8
  int v6; // [rsp+10h] [rbp-450h]
  int v7; // [rsp+14h] [rbp-44Ch]
  __int64 v8; // [rsp+20h] [rbp-440h]
  __int64 v9; // [rsp+28h] [rbp-438h]
  __int64 v10; // [rsp+30h] [rbp-430h]
  __int64 v11; // [rsp+38h] [rbp-428h]
  __int64 v12; // [rsp+40h] [rbp-420h]
  char ptr; // [rsp+50h] [rbp-410h]
  unsigned __int64 v14; // [rsp+458h] [rbp-8h]

  v14 = __readfsqword(0x28u);
  if ( argc != 3 )
    return 0xFFFFFFFFLL;
  v6 = atoi(argv[1]);
  v7 = atoi(argv[2]);
  if ( v6 < 0 || v7 < 0 || v7 <= v6 )
    return 0xFFFFFFFFLL;
  v8 = 0LL;
  v9 = 0LL;
  v10 = 0LL;
  OPENSSL_init_ssl(0LL, 0LL);
  OPENSSL_init_crypto(2048LL, 0LL);
  v11 = ENGINE_get_default_DH(2048LL, 0LL);
  if ( v11 )
  {
    if ( (unsigned int)ENGINE_init(v11) )
    {
      v12 = ENGINE_get_DH(v11);
      if ( v12 )
      {
        v8 = DH_meth_dup(v12);
        if ( v8 )
        {
          if ( (unsigned int)DH_meth_set_generate_key(v8, dh_1) )
          {
            if ( (unsigned int)ENGINE_set_DH(v11, v8) )
            {
              v5 = TLSv1_2_client_method(v11, v8);
              v10 = SSL_CTX_new(v5);
              if ( (unsigned int)SSL_CTX_set_cipher_list(v10, "DHE-RSA-AES128-SHA256") )
              {
                v9 = BIO_new_ssl_connect(v10);
                BIO_ctrl(v9, 100LL, 0LL, (__int64)"ssltest.a1exdandy.me:443");
                if ( BIO_ctrl(v9, 101LL, 0LL, 0LL) >= 0 )
                {
                  BIO_ctrl(v9, 101LL, 0LL, 0LL);
                  BIO_printf(v9, "GET /flag.jpg HTTP/1.1\n", argv);
                  BIO_printf(v9, "Host: ssltest.a1exdandy.me\n");
                  BIO_printf(v9, "Range: bytes=%d-%d\n\n", (unsigned int)v6, (unsigned int)v7);
                  v4 = (signed int)BIO_read(v9, &ptr, 1024LL);
                  fwrite(&ptr, v4, 1uLL, stdout);
                }
                else
                {
                  v4 = 1LL;
```

```
                fwrite("Can't do connect\n", 1uLL, 0x11uLL, stderr);
              }
            }
            else
            {
              v4 = 1LL;
              fwrite("Can't set cipher list\n", 1uLL, 0x16uLL, stderr);
            }
          }
          else
          {
            v4 = 1LL;
            fwrite("Can't set DH methods\n", 1uLL, 0x15uLL, stderr);
          }
        }
        else
        {
          v4 = 1LL;
          fwrite("Can't set generate_key method\n", 1uLL, 0x1EuLL, stderr);
        }
      }
      else
      {
        v4 = 1LL;
        fwrite("Can't dup dh meth\n", 1uLL, 0x12uLL, stderr);
      }
    }
    else
    {
      v4 = 1LL;
      fwrite("Can't get DH\n", 1uLL, 0xDuLL, stderr);
    }
  }
  else
  {
    v4 = 1LL;
    fwrite("Can't init engine\n", 1uLL, 0x12uLL, stderr);
  }
}
else
{
  v4 = 1LL;
  fwrite("Can't get DH\n", 1uLL, 0xDuLL, stderr);
}
if ( v11 )
{
  ENGINE_finish(v11, v4);
  ENGINE_free(v11);
}
if ( v8 )
  DH_meth_free(v8, v4);
if ( v10 )
  SSL_CTX_free(v10, v4);
if ( v9 )
  BIO_free_all(v9, v4);
return 0LL;
}
```

Картинки с флагом на сервере не оказалось, зато в dump.pcap оказалось куча ssl-трафика, предположительно с кусками картинки. После быстрой проверки сервера на heartbleed (чтобы стырить приватный ключик для расшифровки трафика) было выяснено, что сервер не уязвим. Кроме того, в SSL сессиях согласно дампу трафика и клиенту, используется шифр DHE-RSA-AES128-SHA256, в котором RSA используется только для подписи, а обмен ключами происходит по схеме Диффи-Хеллмана (приватный RSA ключик сервера в таком режиме нам не поможет).

2. Немного подирбастив сервер нашел файлик https://ssltest.a1exdandy.me/x , который является простеньким вредоносом, зашитый в него адрес админки - 0x82C780B2697A0002 (0x82C780B2:0x7a69 = 178.128.199.130:31337). При подключении к порту 31337, было выяснено, что сервер поддерживает 3 команды, некоторые из которых просят дополнительные аргументы

   nc 178.128.199.130 31337

Yet another fucking heap task...
Command: 1-3

1 - Index: - Size:
2 - Index:
3 - Index: - Length:

Но дальше ничего сделать не получилось с этим портом, и, скорее всего, это был отвлекающий таск.

3. Посмотрев внимательно client, увидел, что в нем используется кастомизированный генератор секретов Диффи-Хеллмана:

```
int __fastcall rnd_work(__int64 a1)
{
  __int64 v1; // rsi
  unsigned int i; // [rsp+10h] [rbp-10h]

  rnd_read();
  BN_bin2bn(&RANDOM_512, 512LL, a1);
  BN_lshift1(a1, a1);
  v1 = (unsigned int)BITS_ind[0];               // BITS_ind        dd 4096, 4095, 4081,
4069, 0
  if ( (unsigned int)BN_is_bit_set(a1, (unsigned int)BITS_ind[0]) )
  {
    for ( i = 0; i <= 4; ++i )
    {
      if ( (unsigned int)BN_is_bit_set(a1, (unsigned int)BITS_ind[i]) )
      {
        v1 = (unsigned int)BITS_ind[i];
        BN_clear_bit(a1, v1);
      }
      else
      {
        v1 = (unsigned int)BITS_ind[i];
        BN_set_bit(a1, v1);
      }
    }
  }
  if ( (unsigned int)((signed int)((unsigned __int64)BN_num_bits(a1) + 7) / 8) > 0x200 )
  {
    printf("Err!", v1);
    exit(0);
  }
  BN_bn2binpad(a1, &RANDOM_512, 512LL);
  return rnd_write();
}
```

Изначально секрет (512 байт) читается из /dev/urandom и сохраняется в файл state. При каждом следующем запросе с секретом происходит вот такая магия:

```
XOR = 2**4096 + 2**4095 + 2**4081 + 2**4069 + 1
CMP = 2**4096

state *= 2
if state > CMP:
  state ^= XOR
```

Секрет как длинное число сдвигается на 1 бит влево, и если старший бит был 1, то число ксорится с константой из 5 ненулевых бит (XOR).

Посмотрев pcap увидел, что параметры Диффи-Хеллмана, прилетающие от сервера постоянны:

dh_g = 2

dh_p =
23390802492779255177134184370397517812355114045331724403582725611989933627587
39401628497740832343323137697741404356266201556242992633613057758919052185866
70655715893288485709389705595840459536959184197888703535377147531607239137521
00704810651892577111770521339703456940346854154884020022465250463024557548779
12628500832530428925635954562125372206923099547410895937384190821069805333212
42052260848103390783970996421645754599588489631366724152747516143702550329379
81786588147095719801999313216854607209552815027819569749983631505548263472693
03406621084722334380799951438407554891259374905474388779304738382511246753225
9

А при каждом установлении соединения клиент посылает свою публичную часть секрета Диффи-Хеллмана. Сравнивая публичные части секретов соседних сессий можно восстановить начальный секрет клиента, а затем все последующие секреты для каждой сессии:
Если старший бит секрета равен 0, то на следующей сессии секрет станет просто в 2 раза больше, а публичная часть возведется в квадрат по модулю p. Таким образом удалось восстановить начальный секрет (то, что прочиталось из /dev/urandom) по модулю p:

21203026657408131340081649553555077103988039053928613582810186903734586942020
59974533258150533645955531600047907594359958275925171784741886651113321894206
50868610567156950459495593726196692754969821860322110444674367830706684288723
40092471871874457207271644500778995507253233899654346028749977313778507161517
43117746595491095419046545686731437095871841282202774713181557577997594708295
97214195494764332668485009525031739326801550115807698375007112649770412032760
12205452700064519182799525264971495134695518061983478353178741199860061007517
54947469532366281256131779971456508591639859841594686748546999019270801439778
13208682753148280937687469933353788992176066206254339449062166596095349440088
42929113567330833424580437523011509515917231297567943275016324693626660307731
42208130420480630339273456135652271843330915345510718240335351594835411759588
67122974738255966511008607723675431569961127852005437047813822454112416864211
12032301600826785372273131102623332323512192296970201633716433685382659808285
55920071267273520411249112210484981418416257653902044607252315814169911527691
76243658310857769293168120450725070030636638954553866903537931113666283836250
52531879862287234783939119793946829512406062996125070817249996611040652734 7

а из него несложно посчитать секреты для всех остальных сессий.

И вот тут появились проблемы:

A) Wireshark не умеет расшифровывать SSL зная секреты Диффи-Хеллмана и готовых решений не нашлось. Надо самим посчитать общий секрет Диффи-Хеллмана (он же pre-master key сессии), а по нему с помощью большого велосипеда (не думал в SSL есть велосипеды) найти master key сессии. Дальше можно сделать SSLKEYLOG файл, в который записать client random (есть в каждой ssl сессии) и master key, указать его в настройках WireShark для расшифровки SSL и теоретически профит.

   Но возникло еще несколько проблем

B) PHP считал слишком медленно (не используйтся функции bcadd, bcpowmod…), решил переписать на питоне.
C) Формулу расчета master key по pre-master key в человеческом виде найти не удалось, сорцы ssl понимаются очень тяжело, заставить openssl вывести результаты промежуточных расчетов тоже не смог. В итоге использовал такой код https://golang.org/src/crypto/tls/prf.go#L145 , описание

https://www.cryptologie.net/article/340/tls-pre-master-secrets-and-master-secrets/ и какие-то RFC:

```
master_secret = PRF(pre_master_secret, "master secret",
                    ClientHello.random + ServerHello.random)
                    [0..47];
```

В итоге спустя полдня смог накодить такое (по мне, не обошлось без велосипедов):

```
for i in xrange(0, 4264):


  dh_secret = pow(srv_pubkeys[i], state, dh_p)
  dh_secret = hex(dh_secret)[2:-1]
  if len(dh_secret) % 2 :
    dh_secret = "0"+dh_secret
  while dh_secret[0:2] == "00":
    dh_secret = dh_secret[2:]
  dh_secret = dh_secret.decode("hex")

  seed = "master secret"+(cl_random[i].strip() + srv_random[i].strip()).decode("hex")
  A = seed
  master_key = ""
  for j in xrange(0, 2):
    A = hmac.new(dh_secret, A, hashlib.sha256).digest()
    master_key += hmac.new(dh_secret, A+seed, hashlib.sha256).digest()

  master_key = master_key[0:48].encode("hex")
  print "CLIENT_RANDOM " + cl_random[i].strip() + " " + master_key

  state *= 2
  if state > CMP:
    state ^= XOR
```

D) Чтобы выдирать различные client random, … из сессий Wireshark использовался экспорт в csv и поиск в сыром трафике того, что в csv попало как “…”.

E) Для расшифровки 4264 сессиий WireShark решил скушать много гигов оперативы (8 ему не хватило), но ничего, можно все запустить на мощном компьютере, а не на слабом ноуте. Однако при экспорте http-объектов (расшифрованных кусков картинки) WireShark может сохранить только первые 1000 файлов, а дальше у него нумерация заканчивается. В итоге пришлось разбивать pcap на 5 частей по 1000 tcp-сессий в каждом. В итоге получилась такая красивая картинка после склейки всех кусочков:


ZERO
NIGHTS
COME PLAY WITH US
zn19{672be8dd061e3014cd14c17b60fb7048}