Отчёт по реверс-инжинирингу криптографической функции - Lab3

Matthew Rusakov m.rusakov@innopolis.university SD-03

May 2025

Предисловие

Я проанализировал бинарник тремя способами:

- Статический + динамический анализ (динамический анализ не выдал результатов)
- Анализ дизасемблированных функций в ghidra
- Анализ бинарника с помощью доп. утилит

1 Анализ с помощью команд в терминале (статический анализ)

Команда

```
strings 2 | grep -iE "aes|rsa|des|encrypt|decrypt|crypto|sha|md5"
```

не дала никакого аутпута

Команда

```
rabin2 -i 2
```

вывела

```
# Импорты
[Imports]
nth vaddr      bind   type   lib name
-----------------------------------
1   0x00000000 GLOBAL FUNC       __libc_start_main
2   0x00000000 WEAK   NOTYPE     _ITM_deregisterTMCloneTable
3   0x00001070 GLOBAL FUNC       puts
4   0x00001080 GLOBAL FUNC       __stack_chk_fail
5   0x00001090 GLOBAL FUNC       memcmp
6   0x00000000 WEAK   NOTYPE     __gmon_start__
7   0x00000000 WEAK   NOTYPE     _ITM_registerTMCloneTable
8   0x00000000 WEAK   FUNC       __cxa_finalize
```

Команда

```
objdump -x 2 | grep -i openssl
```

не вывела ничего

Вывод Бинарник не использует стандартные криптографические библиотеки (OpenSSL, libcrypto и т. д.), так как:
strings не нашёл упоминаний AES, RSA, DES, SHA и т. д.
rabin2 -i не показал импортов криптографических функций.
objdump тоже не нашёл ссылок на OpenSSL.
Это значит, что шифрование, скорее всего:

- Самописное (например, простой XOR, ROT, замены).
- Встроенное в код (без внешних библиотек).
- Использует базовые алгоритмы (CRC, Base64, кастомные шифры).

Для более глубокого анализа я решил дизасемблировать бинарник в ghidra

2 Анализ дизассемблированных функций в Ghidra

В ходе анализа было исследовано несколько функций, извлечённых с помощью дизассемблирования бинарного файла в Ghidra. Цель — определить структуру, поведение и назначение данных функций. Основное внимание уделяется функциям, связанным с криптографической обработкой данных. Как показал анализ, исследуемый код представляет собой собственную реализацию блочного шифра, архитектурно схожую с AES (Advanced Encryption Standard).

2.1 Функция `FUN_00101caf`

Основная управляющая функция, выполняющая следующие шаги:

1. Инициализация двух массивов: `local_a0` и `local_60`, содержащих по 16 слов (64 байта).
2. Инициализация ключевого материала (`local_c0`, `local_d0` и др.).
3. Вызов функции `FUN_0010167a` — установка начального ключа.
4. Вызов функции `FUN_0010177e` — основной криптографический процесс.
5. Сравнение результата с эталоном (`memcmp`).

Выводится сообщение `SUCCESS` или `FAILURE` в зависимости от результата.

2.2 Функция `FUN_0010167a`

Функция копирует 128-битный ключ из `param_3` в область памяти по смещению `+0xf0` внутри структуры состояния. Используется для установки внутреннего состояния перед шифрованием.

3 Функция FUN_0010177e

Реализует цикл побайтового шифрования данных, аналогичный AES CTR mode:

- Генерирует ключевой поток путём вызова FUN_001012d2.
- Инкрементирует счётчик (128-битный nonce/counter).
- XOR-ит ключевой поток с входными данными.

Алгоритм работает с блоками по 16 байт, каждый байт обрабатывается в цикле с шагом XOR.

3.1 Функция FUN_001012d2

Реализует один раунд симметричного блочного шифра:

1. Применяет подстановку байтов (S-box), используя таблицу DAT_00102140.
2. Производит перестановку байтов (аналог ShiftRows в AES).
3. Выполняет линейную трансформацию, аналогичную MixColumns.
4. Добавляет раундовый ключ через функцию FUN_0010128c.
5. Повторяет 14 раундов, аналогично AES-128.

3.2 Функция FUN_0010128c

Реализация AddRoundKey, то есть побайтового XOR между состоянием (16 байт) и 16-байтным раундовым ключом из расписания. Индекс раунда задаётся аргументом param_1.

3.3 Функция FUN_001012c3

Реализует умножение байта на 2 в поле GF(2^8):

$$xtime(b) = \{\, b \ll 1, b = 0 (b \ll 1) \oplus 0x1B, b = 1$$

Данная операция используется в линейной трансформации MixColumns.

4 S-box (DAT_00102140)

S-box по адресу 0x00102140 соответствует таблице подстановок, применяемой в AES:

- DAT_00102140[0x00] = 0x63 — соответствует AES S-box[0x00].
- Применяется при преобразовании каждого байта состояния.

Вывод

В результате анализа установлено, что представленные функции реализуют блочный шифр, чрезвычайно схожий с AES:

- Используются фазы: `SubBytes`, `ShiftRows`, `MixColumns`, `AddRoundKey`.
- Применяется S-box, идентичный AES.
- Алгоритм построен на 14 раундах, как в AES-128.

Таким образом, можно сделать вывод, что бинарный модуль реализует кастомную или облегчённую версию AES, возможно, с модифицированным раундовым расписанием.

5 Использование автоматических утилит

Я попробовал установить утилиту binwalk, которая сможет определить, что за механизм шифрования используется в бинарнике

```
binwalk -B 2
```

аутпут:

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             ELF, 64-bit LSB shared object, AMD x86-64, version 1 (SYSV)
8256          0x2040          AES Inverse S-Box
8512          0x2140          AES S-Box
```

- ELF-файл (64-битный, LSB, для x86-64) — это стандартный формат исполняемых файлов в Linux
- Найден обратный S-box AES (AES Inverse S-box), используемый при расшифровке
- Найден прямой S-box AES (AES S-box), используемый при шифровании

Вывод Обнаруженные S-box'ы подтверждают, что в бинарнике реализован AES или AES-подобный шифр. Смещения (0x2040 и 0x2140) указывают на начало массивов, которые используются в подстановочных преобразованиях (SubBytes). Это полностью совпадает с тем, что мы ранее увидели в дизассемблированных функциях: S-box по адресу 0x00102140 и его применение в FUN_001012d2.

Список литературы

[1] GitHub Link: https://github.com/MattWay224/reverse-engineering-course В этом репозитории можно найти все лабы и информацию про каждое задание в каждой лабе