УДК 004

Парсинг двоичных ELF-файлов на Python с применением LIEF

Фатеенков Данила Витальевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В статье описан процесс работы с двоичными форматами ELF и PE. Для работы используется язык программирования Python и необходимые модули, такие как ELF. В статье также описан формат файлов ELF и PE и для чего они были созданы, рассмотрены основные параметры и характеристики, заданные в файлах таких форматов.
Ключевые слова: двоичные файлы, парсинг, ELF, Python, struct, LIEF

Parsing ELF binary files in Python using LIEF

Fateenkov Danila Vitalievich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article describes how to work with ELF and PE binary formats. It uses the Python programming language and necessary modules, such as ELF. The article also describes ELF and PE file formats and what they were created for, the main parameters and characteristics set in the files of such formats are discussed.
Keywords: binary files, parsing, ELF, Python, struct, LIEF

1. Введение

1.1 Актуальность

Актуальность статьи обусловлена тем, что ELF является одним из основных форматов исполняемых двоичных файлов в UNIX подобных системах (Linux и другие). Важно понимать структуру файлов такого формата, так как они могут использоваться также и для распространения вирусов и нежелательной информации на компьютере пользователя.

В настоящее время существуют различные редакторы исполняемых файлов (в том числе и те, которые предназначены для работы с ELF-файлами), но зачастую нельзя получить подробную информацию о той или иной части файла. Например, hex-редакторы не предоставляют подробную информацию об импортируемых и экспортируемых функциях в ELF-файле.

Поэтому может возникнуть необходимость провести парсинг ELF-файла. Для решения такой задачи подойдёт язык программирования Python и модуль LIEF.

1.2 Обзор исследований

Н.К. Мищенко описала применение классификатора Байеса для идентификации ELF-файлов [1]. Для реализации поставленной задачи использовался MATLAB и в результате был выбран наиболее оптимальный параметр для классификации.

С.И. Штеренберг и А.В. Красов провели исследование, направленное на нахождение вариантов применения языка программирования Assembler для скрытия вирусов внутри ELF-файлов [2].

Ю.А. Тихонова в своей статье описала методы модификации ELF-файлов для внедрения цифрового водяного знака [3]. Внедрение ЦВЗ обусловлено защитой ПО от незаконного копирования и распространения.

И.Е. Кривцова, К.И. Салахутдинова и И.В. Юрин рассмотрели метод защиты ПО на основе идентификации ELF-файлов по их сигнатурам специальной структуры [4].

С.И. Штеренберг описал методы использования языка программирования низкого уровня (в контексте статьи это Assembler) для стеганографического вложения [5].

1.3 Цель исследования

Цель – провести парсинг двоичного файла с расширением ELF.

2. Материалы и методы

Для реализации поставленной цели используется язык программирования Python и сторонний модуль LIEF.

3. Результаты и обсуждения

ELF является форматом исполняемых двоичных файлов, которые используются в UNIX-системах (например, в Linux). Стандарт формата был расширен для использования на 64-разрядных платформах и стал одним из основных в UNIX-системах. Стандарт ELF различает 3 типа файлов:

1. Перемещаемый файл - хранит инструкции и данные, которые могут быть связаны с другими объектными файлами;

2. Разделяемый объектный файл - содержит инструкции и данные и может быть связан с другими перемещаемыми файлами и разделяемыми объектными файлами;

3. Исполняемый файл - содержит полное описание, позволяющее системе создать образ процесса. В том числе: инструкции, данные, описание необходимых разделяемых объектных файлов.

ELF файл состоит из следующих частей:

1. Заголовок. Всегда расположен в начале файла и содержит описание структуры и характеристики файла. К характеристикам относятся тип файла, версия формата файла, архитектура процессора и др. Заголовок имеет следующий размер: 52 байта для 32-битных и 64 для 64-битных файлов;

2. Данные. Эта часть делится на 3 части: таблица заголовков программы, таблица заголовков секций и сами данные.

Все характеристики имеют описание и своё наименование в системе. Структура заголовка файла:

1. e_ident – общая характеристика файла или «магическое число». Состоит из 16-и байт: первые 4 определяют формат файла, а остальные представляют описание файла;
2. e_type – тип объектного файла;
3. e_machine – архитектура процессора;
4. e_version – версия объектного файла;
5. e_entry – точка входа, через которую система передаёт управление при запуске процесса;
6. e_phoff – смещение в таблице заголовков файла;
7. e_shoff – смещение в таблице секций файла;
8. e_flags – специальные биты памяти, которые связаны с файлом и зависят от процессора;
9. e_ehsize – размер заголовка в байтах;
10. e_phentsize - размер одного заголовка программы.

Это не все параметры, входящие в структуру заголовка файла, другие также будут рассмотрены в дальнейшем. Во многих языках программирования можно увидеть следующую структуру заголовка ELF файла:

```
typedef struct {
    unsigned char e_ident[16];
    uint16_t e_type;
    uint16_t e_machine;
    uint32_t e_version;
    uint64_t e_entry;
    uint64_t e_phoff;
    uint64_t e_shoff;
    uint32_t e_flags;
    uint16_t e_ehsize;
    uint16_t e_phentsize;
    uint16_t e_phnum;
    uint16_t e_shentsize;
    uint16_t e_shnum;
    uint16_t e_shstrndx;
} Elf64_Ehdr;
```

Чтобы убедиться в том, что файл представлен в формате ELF, можно посмотреть заголовок в любом hex-редакторе и прочитать первые 4 байта. Если они равны «7F 45 4C 46», то при декодировании будет получена строка «.ELF». Как было описано выше – заголовок присутствует в самом начале файла и первые 4 байта всегда будут равны «7F 45 4C 46» в ELF файле (см. рис. 1).

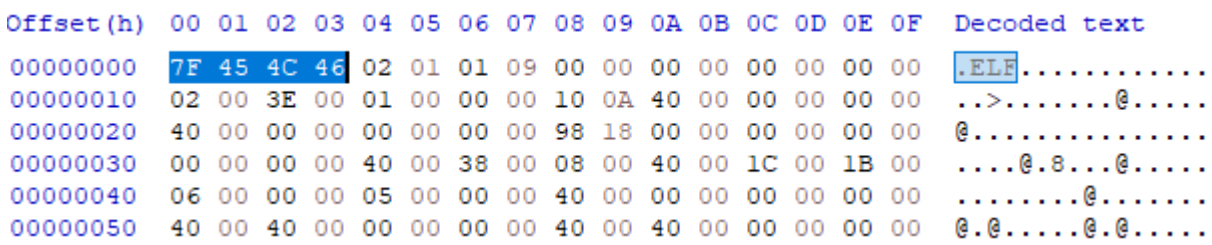

Рисунок 1. Сигнатура файла в hex-редакторе

Для работы с двоичными данными, в Python есть отдельный модуль struct. Но его может быть недостаточно для обработки ELF файлов, поэтому хорошим решением будет также использовать сторонний модуль LIEF. Установить его можно через pip (см. рис. 2).

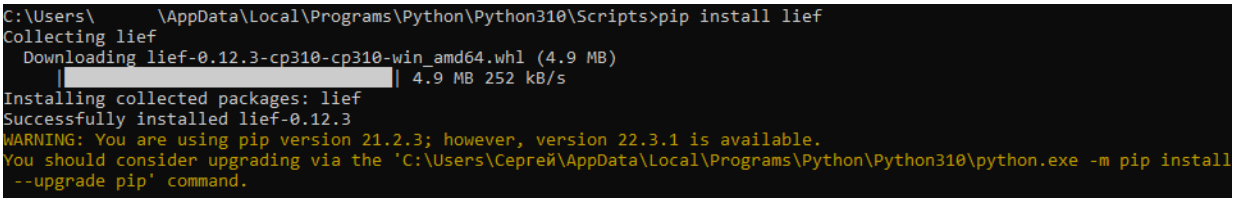

Рисунок 2. Процесс установки модуля LIEF

Для того, чтобы провести парсинг файла, необходимо подать название необходимого файла в функцию “lief.parse()”. Важно отметить, что файл должен быть расположен в одной директории с кодом Python:

```
elf = "elf-FreeBSD-x86_64-echo"
binary = lief.parse(elf)
```

После проведения парсинга, можно обратиться к заголовку файла через метод “header”:

```
print(binary.header)
```

В результате будет выведена информация из заголовка ELF файла (см. рис. 3). Информация включает в себя описание всех параметров заголовка, которые были описаны выше.

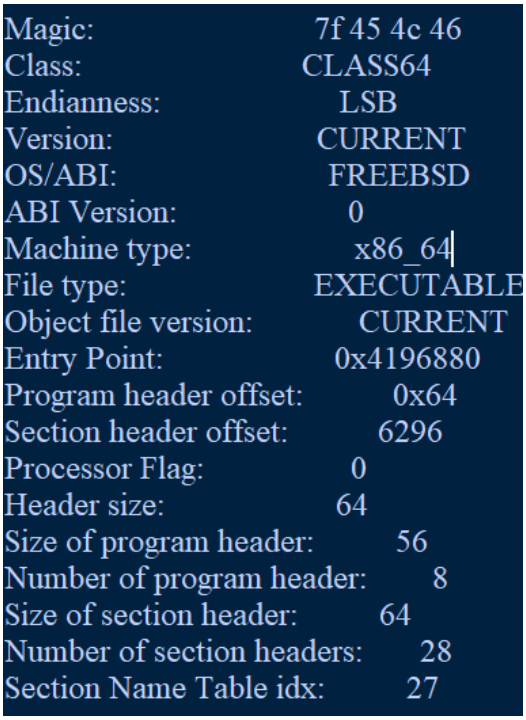

Рисунок 3. Данные заголовка ELF файла

Модуль LIEF позволяет вывести данные некоторых параметров заголовка по отдельности, если в этом есть необходимость. Для этого предложен модулем ряд методов:

1. arm_flags_list – список элементов класса «ARM_EFLAGS»;
2. entrypoint – адрес точки входа;
3. file_type – тип файла;
4. header_size – размер заголовка файла;
5. hexagon_flags_list – список элементов класса «HEXAGON_EFLAGS»;
6. identity – общая характеристика файла (представлена в виде списка);
7. identity_abi_version – версия ABI или application binary interface (интерфейс двоичного приложения);
8. identity_class – класс ELF файла;
9. identity_data – кодировка данных ELF файла;
10. identity_version – версия заголовка;
11. identity_os_abi – версия OS ABI, выводится также при вызове метода «header» (см. рис. 3);
12. machine_type – тип архитектуры;

Также присутствуют методы для получения данных о секциях и сегментах обрабатываемого файла.

Стоит учитывать, что сегменты и секции имеют другую структуру и поэтому работа с ними немного отличается. Во многих языках программирования, в которых можно обрабатывать ELF файлы, представлена следующая структура заголовков секций:

```
typedef struct {
```

```
        uint32_t sh_name; (название секции)
        uint32_t sh_type; (тип секции)
        uint64_t sh_flags; (флаги секции)
        uint64_t sh_addr; (адрес загрузки секции)
        uint64_t sh_offset; (сдвиг)
        uint64_t sh_size; (размер в байтах)
        uint32_t sh_link; (связь с другой секцией)
        uint32_t sh_info; (дополнительная информация)
        uint64_t sh_addralign; (выравнивание секции)
        uint64_t sh_entsize; (размер записи в байтах)
    } Elf64_Shdr;
```

Чтобы получить список всех секций в ELF файле, необходимо вызвать метод «sections»:

```
print(binary.sections)
```

Но в таком случае будет получена ссылка на объект в памяти (см. рис. 4). Секции можно вывести через массив.

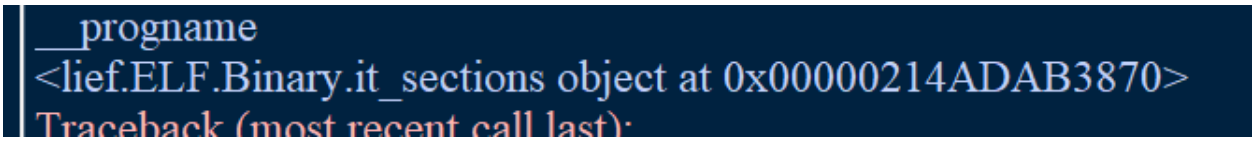

Рисунок 4. Попытка вывода секций через print

Для обработки информации секций можно использовать следующие методы из модуля LIEF:

1. name – название секции;
2. type – тип секции;
3. flags – флаги секции;
4. content – содержимое секции. Представлено в виде массива чисел;
5. offset – сдвиг секции;
6. size – размер секции в байтах;
7. segments – массив сегментов, связанных с секцией;
8. virtual_address – виртуальный адрес секции.

Используя данные методы и функции можно получить всю основную информацию о той или иной секции (см. рис. 5).

Информация, хранящаяся в ELF-файле, разделена и распределена в секции. Каждая секция имеет свое имя (оно является уникальным в списке). Некоторые секции хранят информацию ELF-файла (например, таблицы строк), другие секции хранят информацию, которая используется для отладки. Назначений для информации может быть много.

Таблица заголовков секций представляет собой массив структур Elf32_Shdr, который был описан ранее.

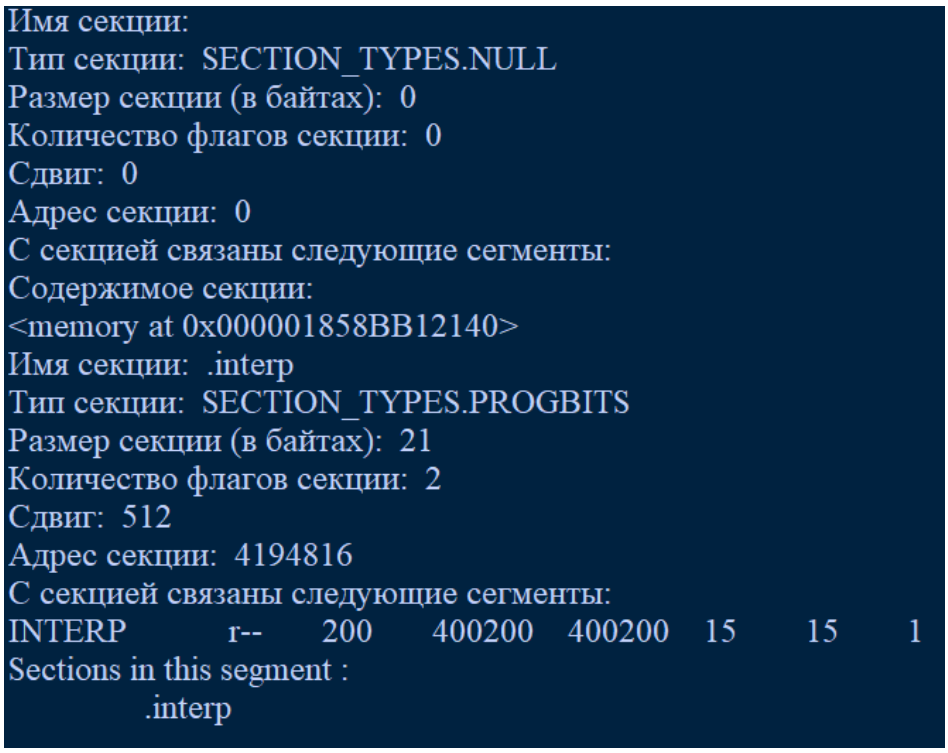

Рисунок 5. Информация о секция ELF файла

Из полученной информации можно перейти к парсингу сегментов. Структура заголовков сегментов следующая:

```
typedef struct {
    uint32_t p_type; (тип сегмента)
    uint32_t p_flags; (флаги сегмента)
    uint64_t p_offset; (сдвиг сегмента)
    uint64_t p_vaddr; (виртуальный адрес сегмента)
    uint64_t p_paddr; (физический адрес сегмента)
    uint64_t p_filesz; (размер сегмента)
    uint64_t  p_memsz;  (занимаемый размер памяти
    сегментом)
    uint64_t p_align; (выравнивание сегмента)
} Elf64_Phdr;
```

Обработка сегментов осуществляется с использованием следующих методов:

1. content – содержимое сегмента. Также представлено в виде массива чисел;
2. file_offset – сдвиг сегмента;
3. flags – флаги сегмента;
4. physical_address – физический адрес сегмента;
5. sections - итератор над секцией, с которой взаимодействует сегмент;
6. type – тип сегмента;
7. virtual_address – виртуальный адрес сегмента;

8. virtual_size – размер сегмента;

Получить список сегментов можно с помощью метода «segments»:

```
print(binary.segments)
```

Будет возвращена ссылка на список сегментов (см. рис. 6).

```
OS_ABI.FREEBSD
<lief.ELF.Binary.it_segments object at 0x0000021BD2C0B030>
```

Рисунок 6. Ссылка на список сегментов ELF файла

Благодаря информации о сегментах можно установить взаимосвязь между сегментами и секциями с помощью метода «sections». Данный метод выводит список всех секций, так или иначе связанных с рассматриваемым сегментом (см. рис. 7). Если вывести список секций, то можно увидеть – какие секции относятся к тому или иному сегменту.

```
Тип сегмента:  SEGMENT_TYPES.LOAD
Сдвиг:  0
Количество флагов сегмента:  SEGMENT_FLAGS.???
Физический адрес сегмента:  4194304
Секции, взаимодействующие с сегментом:
 <lief.ELF.Segment.it_sections object at 0x000002B7E94B70F0>
Виртуальный адрес сегмента:  4194304
Размер сегмента:  4836
```

Рисунок 7. Информация о сегменте, полученная в ходе парсинга

В случае обращения к секциям через «sections» можно получить список параметров секций, которые можно рассмотреть детальнее (см. рис. 8).

```
Тип сегмента:  SEGMENT_TYPES.LOAD
Сдвиг:  0
Количество флагов сегмента:  SEGMENT_FLAGS.???
Физический адрес сегмента:  4194304
Секции, взаимодействующие с сегментом:

.interp          PROGBITS      400200   15      200      3.74899   ALLOC                INTERP LOAD
.note.tag        NOTE          400218   30      218      2.70795   ALLOC              LOAD NOTE
.hash            HASH          400248   ac      248      1.26143   ALLOC             LOAD
.gnu.hash        GNU_HASH      4002f8   34      2f8      3.39089   ALLOC                LOAD
.dynsym          DYNSYM        400330   240     330      1.33304   ALLOC                LOAD
.dynstr          STRTAB        400570   11d     570      4.45732   ALLOC              LOAD
.gnu.version     HIOS          40068e   30      68e      1.69576   ALLOC              LOAD
.gnu.version_r   GNU_VERNEED   4006c0   40      6c0      2.5036    ALLOC                LOAD
.rela.dyn        RELA          400700   18      700      1.13869   ALLOC INFO_LINK      LOAD
.rela.plt        RELA          400718   1b0     718      1.56451   ALLOC INFO_LINK      LOAD
.init            PROGBITS      4008c8   13      8c8      3.26083   ALLOC EXECINSTR      LOAD
.plt             PROGBITS      4008dc   130     8dc      3.80308   ALLOC EXECINSTR      LOAD
.text            PROGBITS      400a10   758     a10      6.11452   ALLOC EXECINSTR      LOAD
.fini            PROGBITS      401168   e       1168     3.23593   ALLOC EXECINSTR      LOAD
.rodata          PROGBITS      401180   38      1180     3.86623   ALLOC              LOAD
.eh_frame_hdr    PROGBITS      4011b8   3c      11b8     3.22253   ALLOC                LOAD GNU_EH_FRAME
.eh_frame        ARM_EXIDX     4011f8   ec      11f8     4.33039   ALLOC              LOAD
Виртуальный адрес сегмента:  4194304
Размер сегмента:  4836
```

Рисунок 8. Информация о сегменте из ELF файла

Помимо сегментов также можно получить информацию о динамических вхождениях в файл, которые содержатся в сегменте PT_DYNAMYC. Они расположены в секции .dynamic и получить список таких элементов можно через метод «dynamic_entries». В результате будет получен список, состоящий из ссылок на динамические вхождения (см. рис. 9).

Рисунок 9. Ссылка на список динамических вхождений в ELF файле

Если вывести каждый элемент списка по отдельности, то можно получить название динамического вхождения и его адрес в файле (см. рис. 10).

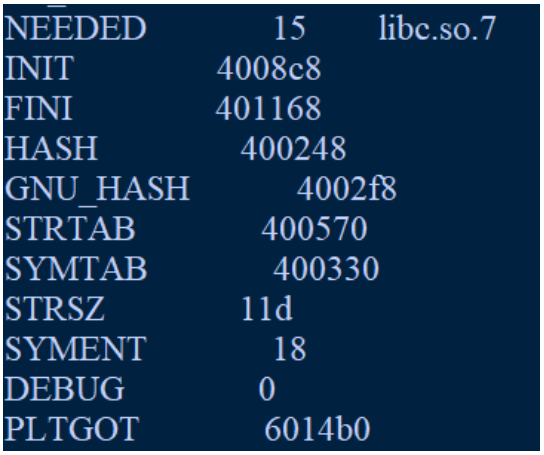

Рисунок 10. Список динамических вхождений в ELF файле

Помимо сегментов, секций и вхождений можно получить информацию о символах файла. В контексте обработки ELF файлов, символы - это ссылки на какой-либо элемент кода (например, глобальную переменную или функцию). В большинстве разделяемых библиотек и динамически подключаемых исполняемых файлов существует две таблицы символов. К символам относятся как стандартные функции для организации ввода и вывода данных в приложении, так и переменные, которые не видит пользователь во время работы программы.

Получить список символов можно с помощью «symbols». В рассматриваемом ELF файле встречаются следующие символы: malloc, cap_ioctls_limit, cap_rights_limit, _Jv_RegisterClasses, atexit, err, strerror, _init_tls, cap_fcntls_limit, strlen, exit, write, writev, strcmp.

В этом списке можно выделить много знакомых разработчику функций и переменных. Все они являются частями тех или иных библиотек.

Получить информацию о том или ином символе можно с помощью следующих методов из модуля LIEF:

1. name – название символьной ссылки;
2. imported – импортирован ли символ. Возвращает либо True, либо False;
3. exported – экспортирован ли символ. Также возвращает True или False;
4. information – данное свойство определяет, какой тип имеет символ и какие имеет атрибуты привязки. Свойство представлено в виде числового значения;
5. shndx – индекс секции, с которой связан символ;
6. size – размер символа. Многие символы имеют заранее определённые размеры и если он не известен, то параметр будет равен нулю;
7. symbol_version – версия символа. Представлена в виде объекта SymbolVersion;
8. type – классификация символа;
9. value – значение символа;
10. visibility – видимость символа;

Также можно узнать – какие функции были экспортированы и импортированы в ELF файле. Для этого существуют методы «exported_functions» и «imported_functions», соответственно (см. рис. 11). Как можно заметить, список импортированных функций может совпадать со списком символов в ELF файле. Ниже представлен пример получения списка импортированных функций:

```
binary = lief.parse(elf)
print("Импортированные функции:")
imported = binary.imported_functions
if len(imported) == 0:
    print("Импортированных функций нет.")
else:
    print("Всего: ",len(imported))
for func in imported:
    print(func)
```

```
Импортированные функции:
Всего: 18
malloc
cap_ioctls_limit
cap_rights_limit
atexit
err
strerror
_init_tls
cap_fcntls_limit
```

Рисунок 11. Список импортированных функций

Если необходимости в углубленном анализе ELF файла нет, то можно получить всю основную информацию, используя метод «abstract». Он возвращает главную информацию об ELF файле, включая информацию о заголовках, символах и функциях:

```
print(binary.abstract)
```

```
Header
======
Magic:                          7f 45 4c 46
Class:                          CLASS64
Endianness:                     LSB
Version:                        CURRENT
OS/ABI:                         FREEBSD
ABI Version:                    0
Machine type:                   x86_64
File type:                      EXECUTABLE
Object file version:            CURRENT
Entry Point:                    0x4196880
Program header offset:          0x64
Section header offset:          6296
Processor Flag:                 0
Header size:                    64
Size of program header:         56
Number of program header:       8
Size of section header:         64
Number of section headers:      28
Section Name Table idx:         27

Sections
========
                  NULL          0        0        0        0
.interp           PROGBITS      400200   15       200      3.74899   ALLOC                    INTERP LOAD
```

Рисунок 12. Сохранённая в виде текстового файла информация об ELF файле

Модуль LIEF также предлагает функции для изменения ELF файлов, но делать это рекомендуется только в случае, если разработчик знает, с каким файлом работает в данный момент.

LIEF также может работать с файлами формата PE (Portable Executable). Файлы данного формата имеют свою структуру и не предназначены для UNIX-систем.

В данной статье был рассмотрен модуль LIEF для Python и проведён парсинг файла с расширением ELF. Были описаны структуры частей ELF файлов и функции модуля LIEF, которые нужны для обработки параметров описанных структур.

Библиографический список

1. Мищенко Н.К. Способ идентификации ELF-файлов на основе классификатора Байеса // Альманах научных работ молодых ученых Университета ИТМО: XLVIII научная и учебно-методическая конференция Университета ИТМО, Санкт-Петербург, 29 января – 01 2019 года. – Санкт-Петербург: федеральное государственное автономное образовательное учреждение высшего образования "Национальный исследовательский университет ИТМО", 2019. С. 38-42.
2. Штеренберг С.И., Красов А.В. Варианты применения языка ассемблера

для заражения вирусом исполнимого файла формата ELF // Информационные технологии и телекоммуникации. 2013. Т.1. № 3. С. 61-71.
3. Тихонова Ю.А. Внедрение ЦВЗ в файл формата ELF // Молодежная научная школа кафедры "Защищенные системы связи". 2020. Т.1. № 1 (1). С. 64-67.
4. Кривцова И.Е., Салахутдинова К.И., Юрин И.В. Метод идентификации исполняемых файлов по их сигнатурам // Вестник государственного университета морского и речного флота им. адмирала С.О. Макарова. 2016. № 1 (35). С. 215-224.
5. Штеренберг С.И. Методика применения языка ассемблер для стеговложения информации в исполняемые файлы // T-Comm: Телекоммуникации и транспорт. 2016. Т.10. № 6. С. 42-47.
6. LIEF URL: https://lief-project.github.io. - Дата обращения: 26.01.2023.