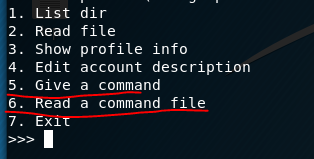
**Решение заданий категории Pwn.**

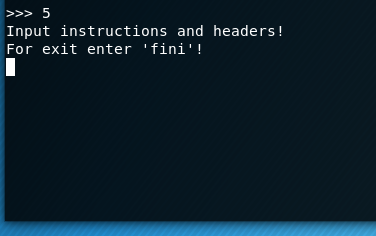
**Задание №3. «Login panel: Got a shell»**

После предыдущего задания мы получаем доступ к функции чтения файлов. С помощью данной функции мы можем получить исполняемый файл сервиса. Для это можно написать простой скрипт. На машине будет хексовый дамп сервиса, сохранённый в текстовом файле. Скачав бинарник команда может анализировать его в IDA Pro.

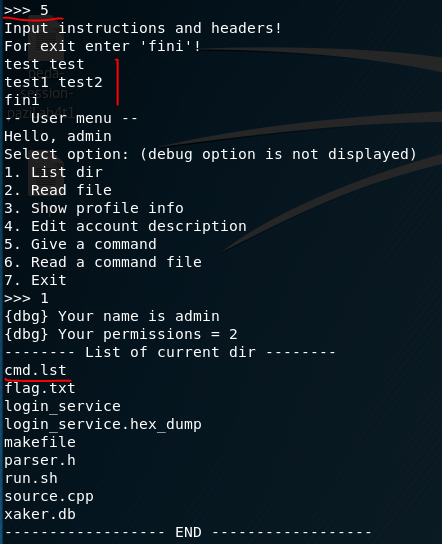
С повышением повышенных привилегий внутри ограниченной оболочки появляется доступ не только к функции чтения файла, но и к новым функция, таким как «Дать команду» и «Прочитать файл с командами».



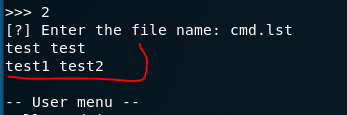
После выполнения функции 5 (дать команду) появляется поле ввода, которое регламетирует, что необходимо ввести заголовки и инструкции, а для завершения ввести «fini».



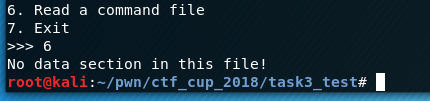
После ввода каких-либо данных в папке появится новый файл с именем «cmd.lst».



При этом, данный файл содержит всё, что было в него передано кроме последней записи «fini».

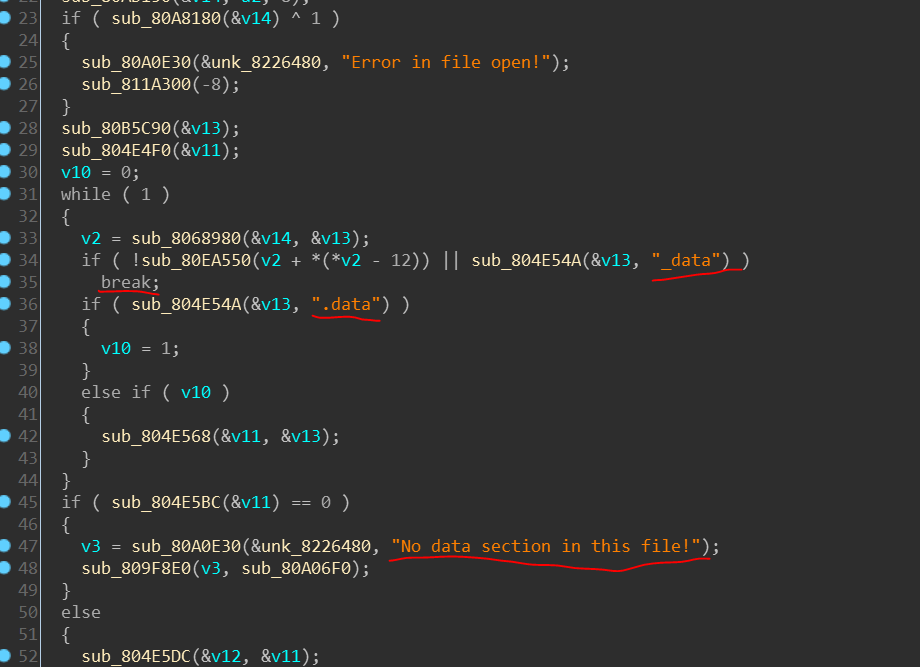


Однако при попытке прочтения командного файла с помощью функции 6 возникает ошибка и соединение разрывается.

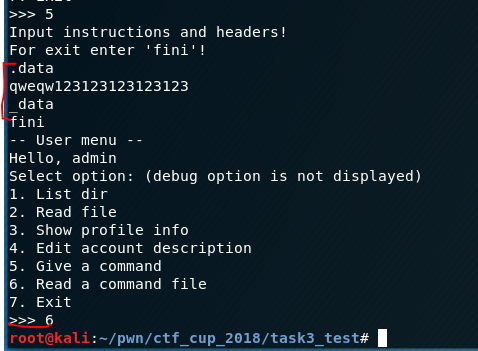


Это даёт вектор исследование исполняемого файла и сужает количество функций и методов, которые необходимо анализировать.

Находим строку, которая соответствует ошибке и выходим на функцию проверки заголовка «data».

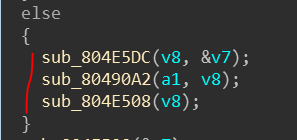


При беглом анализе данной функции можно заметить, что если была встречена строка «.data», то выставляется некий флаг, при наличие которого вызывается функция sub\_804E568. Проанализировав данную функцию, понимаем, что это push\_back в вектор, которым является переменная v11, при этом если размер данного вектора равен 0, то делается вывод, что в секции «data» пусто. Это значит, что должны присутствовать 2 заголовка «.data» и «\_data», между которыми должны быть данные. На основании анализа изменим командный файл.

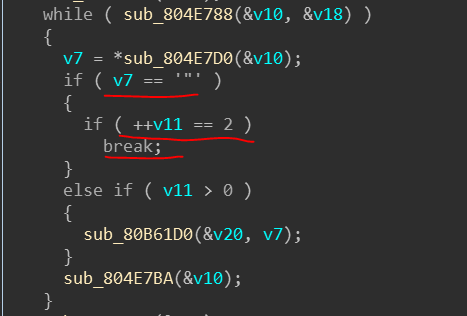


На этот раз сообщение об ошибке не было, но разрыв соединения случился, разрыв может свидетельствовать об фатальной ошибке, приведшей к завершению работы приложения.

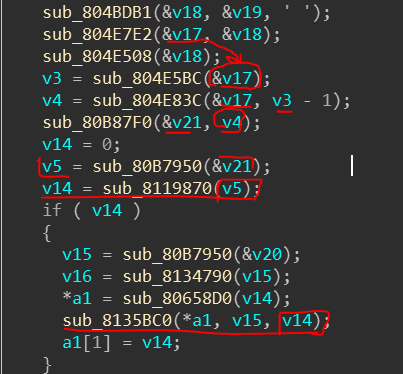
Теперь предстоит анализ блока else, в который передаётся управление, когда размер вектора больше нуля, то есть в секции данных есть какие-либо строчки.



Есть 3 функции, наибольшее внимание из которых представляет функция sub\_80490A2. В данной функции можно встретить интересный цикл с подсчётом количества двойных кавычек. А также функцию, с аргументом в виде символа пробела. При углубленном анализе, можно понять, что функция с аргументом проблема является функцией разбиения строки по разделителю, которым и является пробел.



После этих вывод следует достаточно долгий статический анализ (но можно и воспользоваться динамикой, скорей всего, динамика будет удобнее) следующего блока функций.



Стоит обратить внимание на функция sub\_8135BC0, в которой участвуют три аргумента, при анализе данной функции, можно понять, что это функция memcpy(). Однако стоит обратить внимание на довольно необычный размер. Дело в том, что размер высчитывается путём передачи в функцию sub\_8119870, аргумента являющегося последней подстрокой при разбиении строки по проблему.

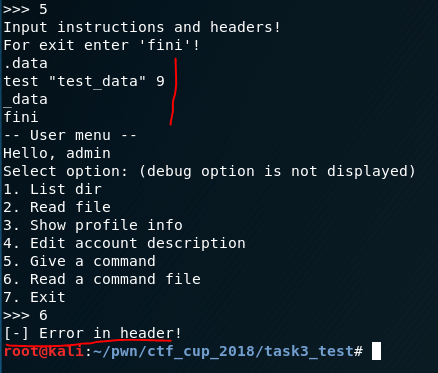
Анализ функции sub\_8119870 в динамике позволит понять, что это функция atoi. Все эти данные приводят к мысли, что в секции «data» хранятся строки следующего вида:

.data

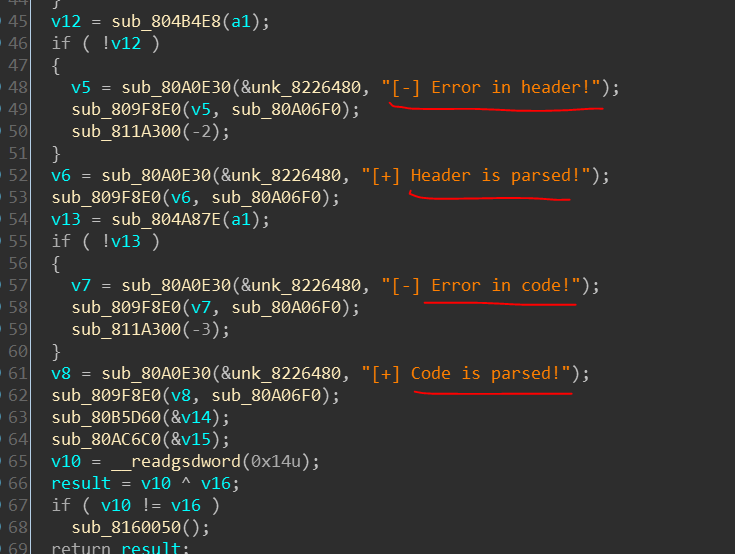
<name> \*двойная кавычка\* <data> \*двойная кавычка\* размер

\_data

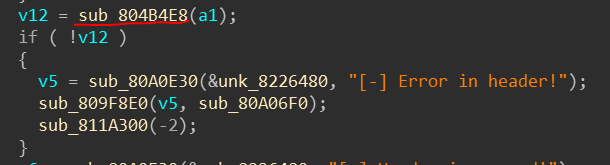
Попробуем сделать такой вид в файле.



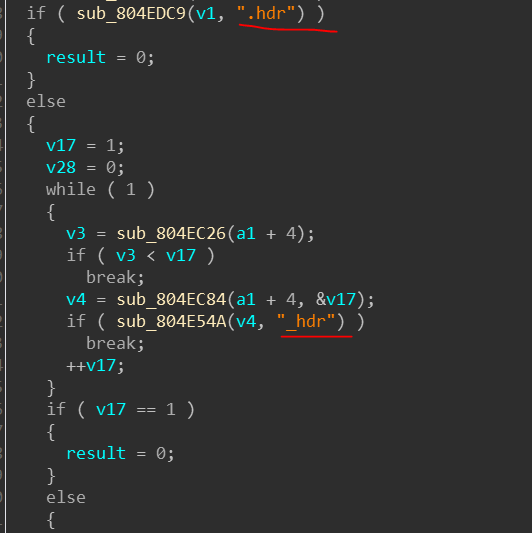
Наше предположение – верно, мы получаем новую ошибку, ошибка в заголовке. Найдём данную строку в базе.



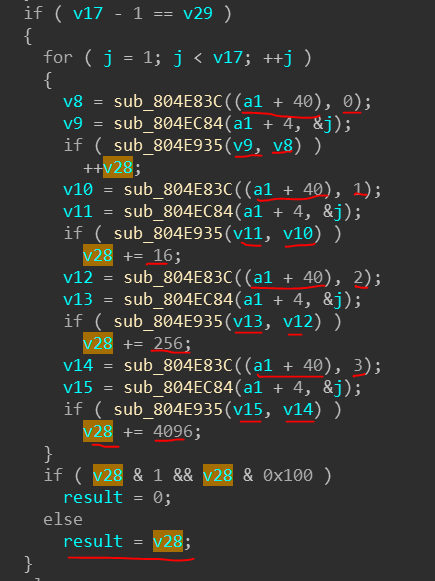
Как можно заметить, при открытие командной файла происходит некоторый парсинг его содержимого.



Просмотрим функцию sub\_804B4E8, которая занимается парсингом заголовка.



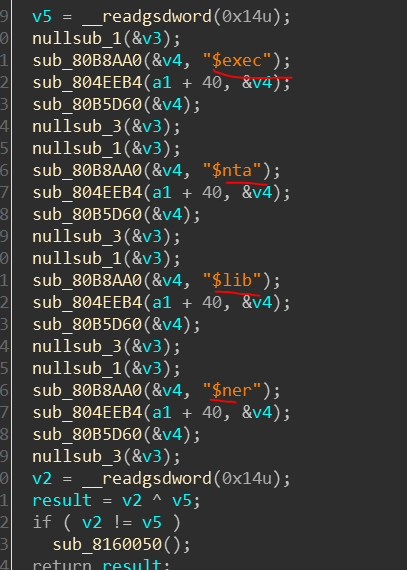
В начале функции встречаем уже знакомый нам формат заголовков секции «.hdr» и «\_hdr», но можно догадаться на основе прошлого опыта, что между ними необходимо поставить ещё что-то. Листая код далее можно наткнуться на интересный блок инструкций.



Переменная v28 возвращается из функции и судя по прибавлению к ней довольно заметный констант (0x10, 0x100, 0x1000) после очередного блока if, можно предположить, что данная переменная является некоторым флагом, который сигнализирует об том или ином значение, присутствующем в секции заголовка. Значения для проверки берутся из памяти текущего объекта, то есть где-то была произведена их инициализация, просмотрев данный участок кода в динамике можно будет спокойно определить, что это за значения. Функция их инициализирующая находится в функции sub\_804B98C.

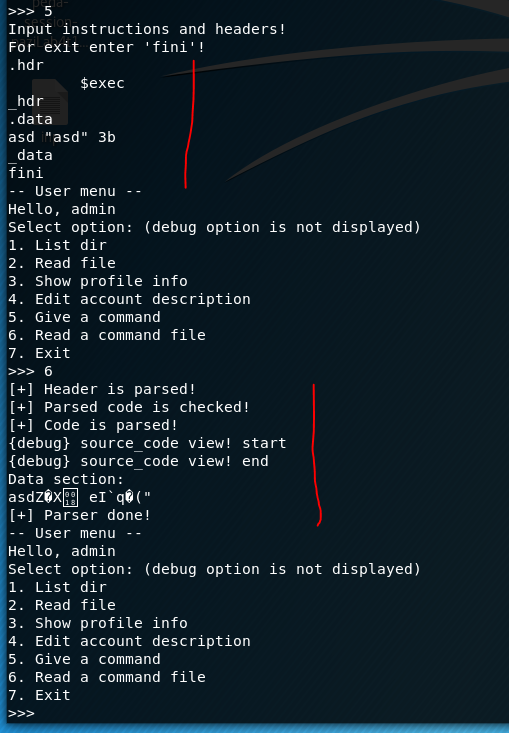


В ней видно, какие константы проверяются в блоке указанном выше.

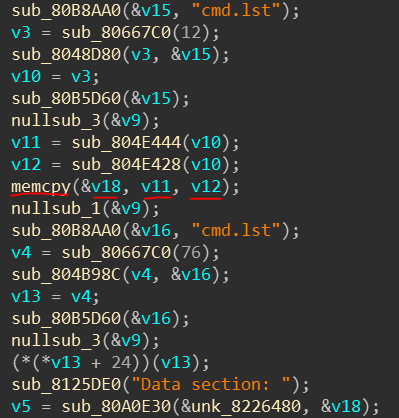


Добавим некоторые из них в командный файл и увидим, что всё теперь выполняется корректно, а также выдаются отладочные сообщения, по которым можно понять, что существует ещё одна секция (но она не обязательна к заполнению).

Отображения секции данных немного не корректно, в том плане, что отображаются байты за пределом переданных 3 байт. Этот факт наталкивает на мысль подробного изучения функции парсинга секции данных.



Однако, уже на данном этапе можно попробовать передать строчку в секции данных неправильно указав её размер (довольно типичное действие для исследователя) и получить явное место, в котором присутствует уязвимость.

На самом деле функция парсинга секции данных, как и функция парсинга секции заголовка и кода вызываются в одном месте, в функции sub\_804C040. 

В данной функции мы видим командный файл, вывод строки об содержании секции данных на экран а также функцию memcpy, которая копирует на стек некоторый массив с размером v12, при этом указатель на источник и размер получаются путём вызовом различных функций от одного аргумента (вызываются различные методы для одного и того-же объекта). Анализ данных методов и возвращаемых ими значений приводит к тому, что размер копируемых данных – есть размер указанный в командном файле, а источник данных – значение между двойными кавычками (это двольно просто глянуть динамикой). При этом буфер куда происходит копирование расположен на стек и ограничен 256 байтами.

Таким образом мы обнаруживаем переполнение буфера на стеке, однако анализ защитных механизмов исполняемого файла выдаёт нам то, что стек защищается с помощью стековой канарейки, а также пристуствует NX, который запрещает выполнение кода не стеке.

Стековая канарейка обходится путём использованя уязвимости Format String, обнаруженной на прошлом этапе, а NX обходится ROP-цепочкой, которая гененрируется довольно просто, так как файл собран статически (возможно ROP-цепочку надо будет немного подкорректировать, у меня Ropper выдал не совсем верную цепочку).

После получения канарейки с помощью format-string и генерации верной ROP цепочки несложно составить финальный эксплоит (файл «exploit\_3.py»).

Демонстрация работы приведена на рисунке ниже.

