ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА
(РОСАВИАЦИЯ)

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)

Кафедра основ радиотехники и защиты информации

А.А. Антонов

ПРАКТИЧЕСКИЕ АСПЕКТЫ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Учебно-методическое пособие
по выполнению лабораторных работ

для студентов
специальности 10.02.05
очной формы обучения

Москва
ИД Академии Жуковского
2021

УДК 004.056
ББК 001.8
А72

Рецензент:
Петров В.И. – канд. техн. наук, доцент

Антонов А.А.
А72 Практические аспекты информационной безопасности [Текст] : учебно-методическое пособие по выполнению лабораторных работ / А.А. Антонов. – М.: ИД Академии Жуковского, 2021. – 32 с.

Данное учебно-методическое пособие издается в соответствии с рабочей программой учебной дисциплины «Практические аспекты информационной безопасности» по учебному плану для студентов специальности 10.02.05 очной формы обучения.

Рассмотрено и одобрено на заседаниях кафедры 22.04.2021 г. и методического совета 22.04.2021 г.

УДК 004.056
ББК 001.8

В авторской редакции

Подписано в печать 28.05.2021 г.
Формат 60х84/16 Печ. л. 2 Усл. печ. л. 1,86
Заказ № 782/0519-УМП44 Тираж 40 экз.

Московский государственный технический университет ГА
125993, Москва, Кронштадтский бульвар, д. 20

Издательский дом Академии имени Н. Е. Жуковского
125167, Москва, 8-го Марта 4-я ул., д. 6А
Тел.: (495) 973-45-68
E-mail: zakaz@itsbook.ru

Лабораторная работа № 1

Анализ безопасности веб-приложений, разработанных на основе CMS платформы

1. Цель работы - научиться использовать стандартное сканирующее программное обеспечение в целях проверки безопасности веб-приложения.

2. Краткие теоретические сведения

В теоретических сведениях описаны методы использования инструментов, для более обширного описания данных инструментов вы можете обратиться к текстам лекций.

Для удобства исследователей создаются отдельные сборки для исследований или так называемые дистрибутивы. Чаще всего это Linux на базе Debian с уже установленным программным обеспечением для анализа уязвимостей и их эксплуатации. В такой сборке может содержаться как базовое программное обеспечение, так и довольно специфическое программное обеспечение, которое редко применяется, но всё же включается в сборки.

На сегодняшний день таких сборок не очень много, основными из них считаются: Kali Linux–как стандарт, Parrot OS–как более усовершенствованная версия Kali.

Мы используем Kali Linux, как дистрибутив для решения практических и лабораторных работ, практически всё программное обеспечение, которое будет рассматривать в курсе содержится в базовой сборке Kali Linux, то есть после установки данной операционной системы, вы будете иметь практически все программы необходимые в данном курсе.

После запуска KaliLinux в левом верхнем углу можно увидеть надпись “Applications”, наведя на которую появиться выпадающее меню с различными пунктами, каждый пункт представляет собой тематику при исследовании безопасности, наведя на один из пунктов можно получить краткий список инструментов, которые необходимы для данной тематики.

Однако, для того чтобы узнать весь список инструментов, предустановленных на вашей версии дистрибутива целесообразно обратиться на официальный сайт, там же можно найти описание каждого из инструментов, либо ссылку на официальный сайт инструмента, если он существует.

Первым делом, когда мы заходим на сайт, мы хотим примерно понять, на чём он написан, что на нём стоит в плане управляющей системы, какие порты открыты/закрыты, какие версии ПО, которое запущено на данных портах и прочую общую информация.

Для анализа веб-приложения необходимо выполнить следующие шаги.

а) Проанализировать сайт с помощью утилиты whatweb.

Для того, чтобы понять на чём написано веб-приложение можно использовать утилиту **whatweb,** для того чтобы использовать данную утилиту

достаточно открыть терминал набрать данную команду и указать после неё адрес исследуемого сайта.

Данный инструмент используется для автоматизирования сбора общей информации о тестируемой цели, может облегчить понимание функционирования веб-приложения и сформировать некоторые векторы атак.

Пример использования приведён на Рисунок 1.

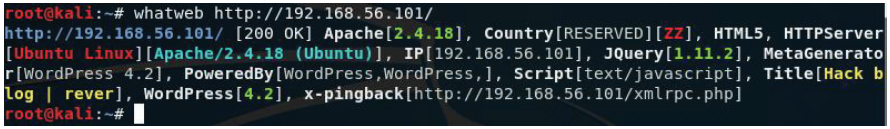

Рисунок 1 - Пример использования whatweb.

Исходя из полученных данных с помощью данной утилиты, необходимо сделать выводы об исследуемом приложении и записать их в отчёт по выполнению лабораторной работы.

Далее нас интересует информация об открытых портах, сервисах, которые запущены и версиях этих сервисов. Для выполнения данной задачи мы будем использовать сетевой сканер – ***nmap***, данный сканер можно использовать из терминала, вообще он довольно мультифункционален и умеет многие вещи, помимо обычных сканирований портов и определения версий.

Сначала проверяются какие порты открыты, для этого напишем следующую команду: nmap 195.133.218.79 (ip-адрес сайта можно получить с помощью whatweb)

Следующая команда данной утилиты используется если первая команда не сработала, но вы знаете что хост доступен nmap –Pn 195.133.218.79 (-Pn используется если первая команда не сработала, но вы знаете что хост доступен, или возможно хост блокирует пинги), как представлено на Рисунке 2.

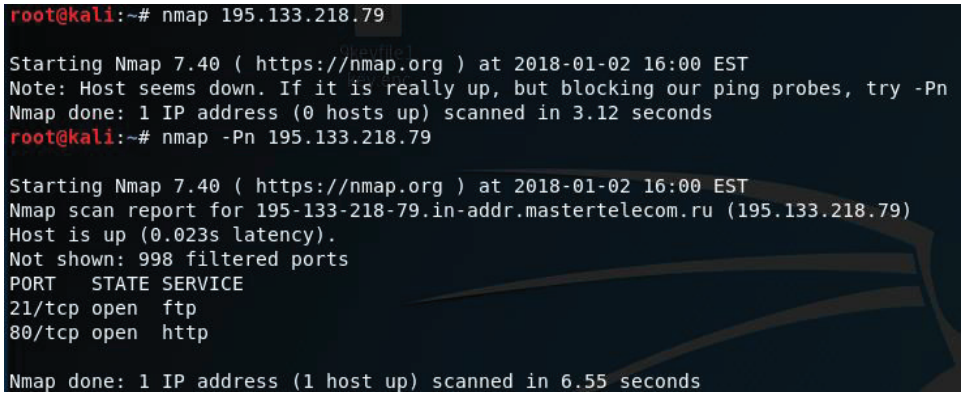

Рисунок 2 - Пример утилиты nmap.

На рисунке мы видим результат работы nmap. При стандартных опциях nmap сканирует лишь небольшой диапазон часто используемых портов, поэтому вывод не всегда может совпадать с тем, что действительно открыто на хосте, например, порты за значением 10000 скорее всего не попадут в вывод при стандартных опциях сканирования.

Для того чтобы указать отдельный порт или диапазон портов можно использовать опцию “-p”, например:

nmap –p80 127.0.0.1 – сканирует только 80 порт на локальной машине

nmap –p21,80 127.0.0.1 – сканирует только 21 и 80 порт на локальной машине

nmap –p1-10,80 127.0.0.1 – сканирует диапазон с 1 по 10 и 80 порт на локальной машине

nmap –p1-65535 127.0.0.1 – сканирует диапазон с 1 по 65535 порт, это всевозможные порты, больше чем 65535 портов – нет.

Также стоит отметить, что при таком типе сканирования – по умолчанию сканируются толькоTCP-порты, для сканирования UDP-портов необходимо специально указать ключ – “-sU”, для указания TCPтоже есть ключ – “-sS”, примеры:

nmap –sU –p123 127.0.0.1 – сканирует 123 UDP порт

nmap –sS –p1-80 127.0.0.1 – сканирует диапазон TCP портов с 1 по 80.

Теперь, когда мы знаем, как произвести сканирование любого порта, нам необходимо решить следующую задачу – определение сервиса, который запущен на данном порту и его версии. Для этого у nmap’а есть прекрасная опция “-sV” – что означает определить версию сервиса, запущенного на порту.

Используем следующую команду: nmap –Pn –sV –p21,80 195.133.218.79, результат представлен на рисунке 3.

```
root@kali:~# nmap -Pn -sV -p21,80 195.133.218.79

Starting Nmap 7.40 ( https://nmap.org ) at 2018-01-03 06:45 EST
Nmap scan report for 195-133-218-79.in-addr.mastertelecom.ru (195.133.218.79)
Host is up (0.022s latency).
PORT   STATE SERVICE VERSION
21/tcp open  ftp     ProFTPD 1.3.4a
80/tcp open  http    Apache httpd 2.2.15 ((CentOS) DAV/2 PHP/5.3.3)
Service Info: OS: Unix

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.77 seconds
```

Рисунок 3 - Пример утилиты nmap

Теперь мы знаем, какие именно сервисы запущены на портах и их версии. После этого мы можем поискать известные уязвимости для данных версий и попробовать применить их. Однако это может затянуться на долго, поэтому нам нужно автоматизировать это. Недавно для nmap был разработан скрипт,

который автоматически находит все известные уязвимости и выдаёт ссылки на них, при сканировании хоста.

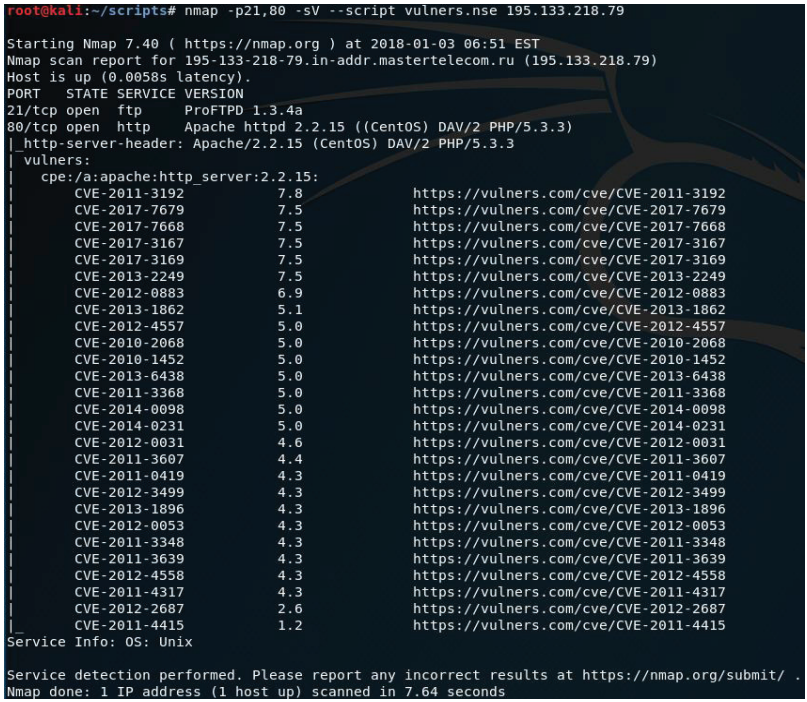

Рисунок 4 - Пример утилиты github’e

Данный скрипт не входит в стандартные скрипты nmap’a (хотя возможно в новых версиях он будет включён в сборку). Для запуска скрипта используется ключ --script (два тире), далее идёт путь к скрипту, либо если он в текущей директории, то просто указывается его полное имя, результатом работы скрипта является список всех публичных уязвимостей для данной версии сервиса, результат представлен на рисунке 4.

После использования whatweb и nmap, очень часто можно получить уже неплохие векторы атак, заметно также, как много информации можно получить в довольно короткий срок, всё это зачастую занимает не больше 5-10 минут. Однако есть сканирования, которые могут занимать довольно много времени, например, сканирования директорий сайта.

Принцип довольно простой, по заранее составленному словарю происходят запросы к сайту с целью поиска файла или директории, если файл или директория были обнаружены это фиксируется в отчёте сканера.

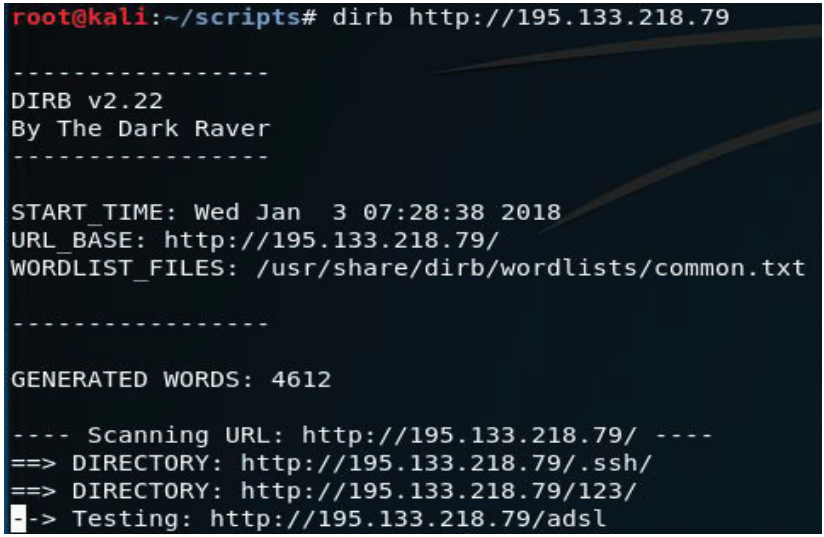

Рисунок 5 - Пример директивы dirbuster

Запуск стандартного сканирования осуществляется с помощью данной команды: dirb http://example.com, результат представлен на рисунке 5.

Также, вместо домена может быть ip-адресс сайта. Результат работы показан на слайде, если была найдена директория, то она помечается стрелкой с двумя равно слева, если же был найден файл, то после него пишется статус код ответа от сервера, если файл доступен, то статус код будет равен 200 и будет указан верный размер файла. Данный сканер может дать очень много полезной информации и найти много интересного, однако он ограничивается словарём, всё что не входит в словарь обнаружено не будет, также это является активным сбором информации и в принципе уже наказуемо.

После всех сканирований нужно приступать к попыткам обнаружения уязвимостей на самом сайте. Начать стоит с CMS системы (Аббревиатура расшифровывается как «Content Management System», то есть система управления контентом (содержимым) сайта), если она публичная, например, Wordpress, Joomla и другие. Для Wordpress и Joomlae есть специальные сканеры уязвимостей, которые сканируют сайт, определяют версию, установленные плагины и ищет по базе известных уязвимостей, в каких местах сайта есть уязвимые компоненты.

Для Wordpress сканер называется – wpscan

Для Joomla сканер называется – joomscan.

б) Просканировать сайт специальным сканером уязвимостей (в зависимости от типа CMS-платформы).

Далее необходимо выбрать подходящий автоматический сканер уязвимостей для исследуемой цели. Автоматические сканеры уязвимостей также рассматривались в лекциях по безопасности веб-приложений. Например, для популярной CMS платформы – Wordpress существует открытый сканер wpscan, который входит в состав дистрибутива Kali Linux.

Исходя из полученных данных с помощью данной утилиты, необходимо сделать выводы об исследуемом приложении и записать их в отчёт по выполнению лабораторной работы. Также необходимо указать общее количество уязвимостей и количество крайне критичных уязвимостей.

Пример использования показан на рисунке 6.

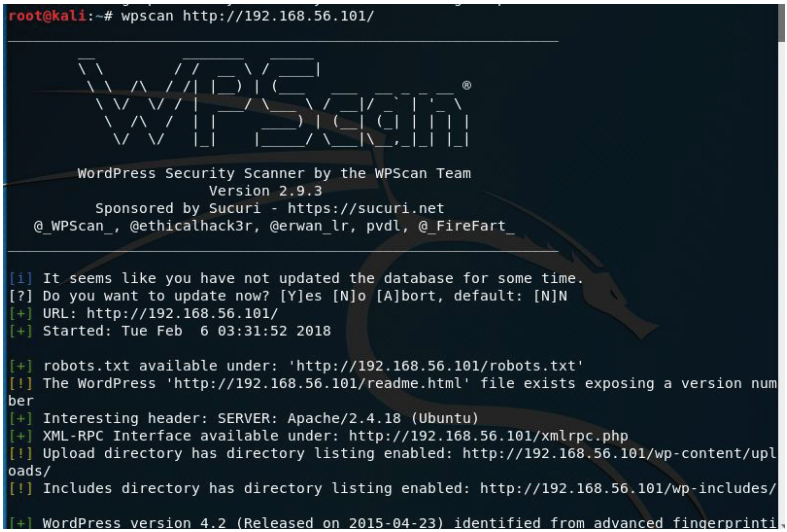

Рисунок 6 - Пример использования сканера wpscan.

в) Провести ручной анализ сайта и найти контрольные значения.

Выполнение данного пункта заключается в ручном анализе сайта и поиска на сайте контрольных значений, представленных в форме специального слова и заключённого в фигурные скобки контрольного значения в формате md5-хеша. Ручной анализ может заключаться в просмотре исходного кода страниц, анализа доступных страниц сайта, нахождение учётных записей пользователей. При проведении анализа необходимо обнаружить как можно больше контрольных значений и внести их в виде скриншотов и описания к ним, каким образом они были получены.

г) Произвести атаку полного перебора для пароля одного из пользователей приложение Burp Suite.

Выполнение данного пункта не обязательно должно заканчиваться компрометацией одного из аккаунтов веб-приложения, так как атаки полного перебора довольно часто занимают большой объём времени. Поэтому в данном пункте необходимо производить атаку по словарю, что ускорит работу атаки полного перебора.

Для простого перебора можно использовать уже применяемую нами ранее утилиту – wpscan которая также может производить перебор паролей. Стоит отметить, что для осуществления перебора необходимо знать имя учётной записи, что в прочем находится без особо труда применительно к Wordpress.

Пример команды для атаки полного перебора приведён на рисунке 7*Рисунок* .

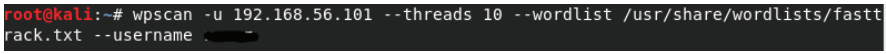

Рисунок 7 - Запуск атаки полного перебора.

После выполнения своих стандартных функцию wpscan начнёт производить атаку полного перебора по заданному словарю.

Ключи, используемые при запуске следующие:

-u – URL цели.

--threads – количество потоков для выполнения атаки

--wordlist – путь до словаря, файла с паролями

--username – имя пользователя для которого проводится атака.

В случае нахождения пароля будет выдано сообщение об успешном проведении атаки, а также в конце напечатана таблица, если вы проводили атаку для нескольких пользователей, как представлено на рисунке 8*Рисунок* .

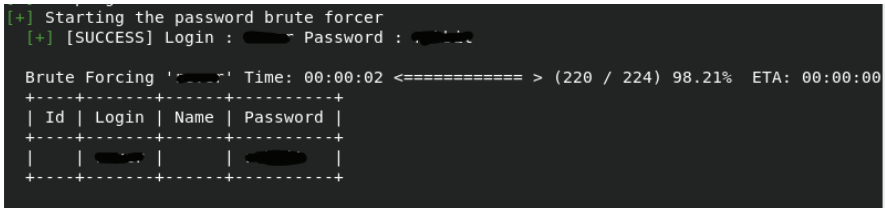

Рисунок 8 - Успешная реализация атаки.

При выполнение данного пункта необходимо провести атаку полного перебора с использованием предложенных инструментов.

При использовании инструментов отличных от wpscan, даже при неудачном проведение атаки — это будет учитываться более весомо, чем удачная атака с помощью wpscan. В отчёт должны войти скриншоты запуска атаки и её результатов, а также выводы по проделанной атаке. Успешность атаки не считается важным критерием.

д) Произвести эксплуатацию известных уязвимостей и описать их.

При помощи данных полученных на втором этапе лабораторной работы, необходимо произвести попытки эксплуатации одной из найденных уязвимостей.

При этом требуется описать данную уязвимость, отметить, существует ли для неё средства управления в открытом доступе, что требуется для успешной реализации данной уязвимости, если реализовать её в данных условиях не представляется возможных – то пояснить принципиальные причины невозможности эксплуатации.

Все действия и анализ необходимо сопровождать скриншотами и логическими пояснениями. Выбор открытой уязвимости производится вами лично, однако он должен быть отличен от выбора вашего соседа.

3. Порядок выполнения работы.

1. Изучить теоретические сведения.
2. Загрузить операционную систему Kali linux с заранее подготовленной загрузочной флэш карты. Необходимо воспользоваться горячими клавишами «F12» и «Esc» для выбора загрузки операционной системы.
3. Используя командную строку браузера ввести адрес: 172.21.45.151/pazi, открыть лабораторную работу № 1, где представлено теоретическое описание и задания.
4. Проанализировать сайт с помощью утилиты whatweb.
5. Просканировать сайт специальным сканером уязвимостей
6. Провести автоматический перебор директорий с помощью dirb.
7. Провести ручной анализ сайта и найти контрольные значения.
8. Произвести атаку полного перебора для пароля одного из пользователей.
9. Произвести эксплуатацию известных уязвимостей и описать их.
10. Составить отчет о лабораторной работе.

4. Содержание отчета

1. Титульный лист.
2. Задание.
3. По каждому пункту составить подробное текстовое описание с изображениями (скриншотами).
4. Вывод, который должен содержать оценки безопасности тестируемого веб-приложения и рекомендации для устранения ошибок безопасности. Стиль оформления отчёта – научно-технический

5. Литература

1. А.С. Масалков: Особенности киберпреступлений в России. Инструменты нападения и защита информации. М.: ДМК-Пресс, 2018.

2. А.А. Бирюков: Информационная безопасность: защита и нападение. М.:ДМК Пресс, 2012.

3. А.С. Панов: Реверсинг и защита программ от взлома. Спб.: БХВ-Петербург, 2006.

4. Семенов Ю.А. Обзор уязвимостей, некоторых видов атак и средств защиты (http://book.itep.ru/6/intrusion.htm).

Лабораторная работа № 2

Анализ безопасности веб-приложений. Методы обхода аутентификации

1. Цель работы - получить навыки обхода аутентификации, познакомиться с базовыми уязвимостями веб-приложений и с их использованием.

2. Краткие теоретические сведения

Основные понятия

Для изучения уязвимостей SQLi рассмотрим основы баз данных SQL.

База данных (БД) - представленная в объективной форме совокупность самостоятельных материалов (статей, расчётов, нормативных актов, судебных решений и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью ЭВМ.

Система управления баз данных (СУБД) - совокупность программных и лингвистических средств, обеспечивающих управление созданием и использованием баз данных.

Функции СУБД:

- управление данными во внешней памяти (на дисках);
- управление данными в оперативной памяти с использованием дискового кэша;
- журнализация изменений, резервное копирование и восстановление базы данных после сбоев;
- поддержка языков БД

Значения данных, хранимые в реляционной базе данных, являются типизированными, т. е. известен тип каждого хранимого значения.

SQL (от англ. Structured Quary Language — язык структурированных запросов) — универсальный компьютерный язык управления данными в системах управления базами данных (СУБД) основанных на реляционной модели. SQL является непроцедурным языком. Это значит то, что на языке можно описать манипуляции с данными, но проинструктировать, как ему это сделать нельзя, так как в языке отсутствуют свойственные алгоритмическим языкам конструкции, такие как метки, условные переходы, операторы цикла и др.

Веб-приложения нередко используют SQL-базы данных, т. е. такие БД, взаимодействие с которыми осуществляется через запросы на языке SQL.

Веб-приложение формирует и отправляет базе SQL-запросы, которые могут зависеть от параметров, пришедших от клиента (взятых из HTTP-запроса). SQL-запросы текстовые и простой способ сделать запрос, зависящий от параметра - просто подставить этот параметр в текст запроса.

SQLi это один из представителей класса уязвимостей injection, которые получаются, если в какие-то управляющие команды подставляются параметры-данные от пользователя и эти данные могут выйти из контекста, куда подставляются, перестав быть просто данными, и поменять смысл команды.

Таким образом, в конце своей инъекции можно добавить символы однострочного SQL-комментария, тогда всё что идет дальше (как правило, остаток исходного SQL-запроса) будет считаться комментарием и будет игнорироваться.

LocalFileInclude (LFI) — уязвимость, которая позволяет удаленному пользователю получить доступ к нужной информации с помощью специально сформированного запроса к произвольным файлам.

Что делит LFI на две ветки: выполнение содержимого подключаемого файла и чтение содержимого подключаемого файла.

Уязвимости класса LFI чаще всего встречается в различных менеджерах файлов.

RCE (англ. RemoteCodeExecution) – компьютерная уязвимость, при которой происходит удаленное выполнение кода на взламываемом компьютере, сервере и т.п. RCE это гарантированный способ взлома сайтов и веб-приложений. RCE - является одной из самых опасных уязвимостей.

Возможность удаленного внедрения кода в серверный скрипт в 100% случаев приводит к взлому ресурса. С помощью уязвимости RCE злоумышленник сразу получает доступ к серверу атакуемого сайта, размещая на нем веб-шеллы, или любой другой вредоносный код.

Возможность эксплуатации RCE возникает из-за грубейших ошибок разработки сайта, отсутствия фильтрации передающих параметров, использование небезопасных функций и приемов программирования.

Для выполнения лабораторной необходимо изучить следующие шаги.

а) Анализ cookie-параметра

Cookie — небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. Веб-клиент (обычно веб-браузер) всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP-запроса.

Применяется для сохранения данных на стороне пользователя, на практике обычно используется для:

- аутентификации пользователя;
- хранения персональных предпочтений и настроек пользователя;
- отслеживания состояния сеанса доступа пользователя;
- ведения статистики о пользователях.

Данный тип параметра можно изменять на стороне клиента, однако обрабатывается данный параметр на сервере, а не в браузере клиента, что может позволить произвести атаку на сервер или обход каких-либо ограничений при существующей уязвимости в данном параметре.

Не редкость, когда приложения обладают уязвимыми cookie-параметрами, хранящими данные об уровне привилегий вашей учётной записи, при использовании слабой криптографии или простого кодирования данных, это может повлечь за собой плохие последствия.

Пример слабо защищённых cookie-параметров представлен на рисунке 9.

Рисунок 9 - Пример запроса с параметром cookie.

б) Особенности языка PHP при обработке данных.

Известно, что язык PHP обладает своеобразной функциональностью и логикой. Зачастую результаты обработки данных с помощью некоторых встроенных функций вызывает множество вопросов о корректности действий и логике работы языка. Однако несмотря на многие факторы против данного языка он до сих пор активно используется и развивается.

Типичным примером не совсем верной логики языка является отсутствие в некоторых логических операции транзитивности, пример на представлен рисунке 10.

```
php > $var = "teststring";
php > var_dump($var == 0);
bool(true)
php > var_dump($var == False);
bool(false)
php > var_dump(0 == False);
bool(true)
php >
```

Рисунок 10 - Пример необычного поведения кода PHP.

Также это демонстрирует и атаки типа Request Injection, сутью которых является изменение типа передаваемого значения, в надежде на отсутствие проверок типа передаваемых значений или использования уязвимых функций, при обработке введённых значений.

Например, передаются 2 параметра, при этом считается, что эти параметры обязательно будут являться строками. Данное упрощение может быть введено разработчиком, если эти параметры принимаются из стандартной формы ввода логина и пароля. Однако разработчик не учитывает, что тип данных, который к нему придёт может являться совсем не строкой, а, например, массивом.

Вот так передаются строки:

url/index.php?login=user&paswd=paswd

Тот же скрипт, но пользователь решил передать не строки, а массивы:

url/index.php?login[]=test&passwd[]=test2

При отсутствии должных обработок типов и при использовании уязвимых функций, данный запрос может произвести обход аутентификации или вызвать сбой в работе приложения.

в) Уязвимость Magic Hash.

Из-за ошибки в PHP при работе с хэшами в некоторых ситуациях злоумышленник может подобрать пароль к учетной записи пользователя, обойти аутентификацию и другие средства обеспечения безопасности, полагающиеся на хеширование.

Проблема возникает при использовании оператора равно-равно (==) во время сравнения двух хэшей. PHP использует кодировку base16, в результате получаем строку, подобную этой: “0e812389…”.

Если строка начинается с «0e» и после нее следуют только цифры, PHP будет интерпретировать эту строку как число с плавающей точкой. В итоге, условие if (0e462097431906509019562988736854 == 0) будет истиной.

Если хеш пароля пользователя в базе данных начинается с 0e, злоумышленник может воспользоваться любой строкой, которая при сравнении будет интерпретирована как 0, и успешно авторизоваться в системе без знания пароля. То есть необходимо найти строку/число, хэш функция которого будет интерпретироваться как число с плавающей точкой.

г) SQLi - обнаружение и базовые навыки эксплуатации

Внедрение SQL-кода (англ. SQL injection) — один из распространённых способов взлома сайтов и программ, работающих с базами данных, основанный на внедрении в запрос произвольного SQL-кода.

Внедрение SQL, в зависимости от типа используемой СУБД и условий внедрения, может дать возможность атакующему выполнить произвольный запрос к базе данных (например, прочитать содержимое любых таблиц, удалить, изменить или добавить данные), получить возможность чтения и/или записи локальных файлов и выполнения произвольных команд на атакуемом сервере.

Атака типа внедрения SQL может быть возможна из-за некорректной обработки входных данных, используемых в SQL-запросах.

Рассмотрим пример использования целочисленного и строкового параметра.

Допустим, серверное ПО, получив входной параметр id, использует его для создания SQL-запроса.

Рассмотрим следующий PHP-скрипт:

```
$id = $_REQUEST['id'];
$res = mysqli_query("SELECT * FROM news WHERE id_news = " . $id);
```

Если на сервер передан параметр id, равный 5 (например так: *http://example.org/script.php?id=5*), то выполнится следующий SQL-запрос:

```
SELECT * FROM news WHERE id_news = 5.
```

Но если злоумышленник передаст в качестве параметра id строку -1 OR 1=1 (например, так: *http://example.org/script.php?id=-1+OR+1=1*), то выполнится запрос: SELECT * FROM news WHERE id_news = -1 OR 1=1

Таким образом, изменение входных параметров путём добавления в них конструкций языка SQL вызывает изменение в логике выполнения SQL-запроса (в данном примере вместо новости с заданным идентификатором будут выбраны все имеющиеся в базе новости, поскольку выражение 1=1 всегда истинно).

Предположим, серверное программное обеспечение, получив запрос на поиск данных в новостях параметром search_text, использует его в следующем SQL-запросе (здесь параметры экранируются кавычками):

```
$search_text = $_REQUEST['search_text'];
$res = mysqli_query("SELECT id_news, news_date, news_caption, news_text, news_id_author
FROM news WHERE news_caption LIKE('%$search_text%')").
```

Сделав запрос вида http://example.org/script.php?search_text=Test мы получим выполнение следующего SQL-запроса:

```
SELECT id_news, news_date, news_caption, news_text, news_id_author FROM news
WHERE news_caption LIKE('%Test%')
```

Но, внедрив в параметр search_text символ кавычки (который используется в запросе), мы можем кардинально изменить поведение SQL-запроса.

Например, передав в качестве параметра search_text значение ')+and+(news_id_author='1, мы вызовем к выполнению запрос:

```
SELECT id_news, news_date, news_caption, news_text, news_id_author FROM news  WHERE news_caption LIKE('%') and (news_id_author='1%')
```

3. Порядок выполнения работы.

1. Изучить теоретические сведения.
2. Загрузить операционную систему Kali linux с заранее подготовленной загрузочной флэш карты. Необходимо воспользоваться горячими клавишами «F12» и «Esc» для выбора загрузки операционной системы.
3. Используя командную строку браузера ввести адрес: 172.21.45.151/pazi, открыть лабораторную работу № 2, где представлено теоретическое описание и задания.
4. Выполнить шесть заданий обращаясь к серверу.
5. При выполнении первого задания использовать расширения для чтения cookie-файлов и кодировку BASE64.
6. При выполнении второго, третьего, четвертого и пятого заданий использовать особенности языка PHP при обработке данных и внедрение SQL-кода.
7. При выполнении шестого задания целесообразно анализировать уязвимости связанные с Magic Hash.
8. Предложить варианты устранения найденных уязвимостей.
9. Сформулировать выводы по работе и составить отчёт.

4. Содержание отчета

1. Титульный лист.
2. Задание.
3. По каждому пункту составить подробное текстовое описание с изображениями (скриншотами).
4. Вывод, который должен содержать оценки безопасности тестируемого веб-приложения и рекомендации для устранения ошибок безопасности. Стиль оформления отчёта – научно-технический

5. Литература

1. А.С. Масалков: Особенности киберпреступлений в России. Инструменты нападения и защита информации. М.: ДМК-Пресс, 2018.
2. А.А. Бирюков: Информационная безопасность: защита и нападение. М.:ДМК Пресс, 2012.
3. А.С. Панов: Реверсинг и защита программ от взлома. Спб.: БХВ-Петербург, 2006.
4. Семенов Ю.А. Обзор уязвимостей, некоторых видов атак и средств защиты (http://book.itep.ru/6/intrusion.htm).

Лабораторная работа № 3

Анализ безопасности приложений. Безопасность механизмов защиты программного обеспечения от копирования

1. Цель работы - получить практические навыки в анализе двоичных приложений, научиться анализировать двоичную реализацию простейших алгоритмов.

2. Краткие теоретические сведения

Для практической реализации необходимо использовать дизассемблер IDA, который является продуктом Hex-Rays. По своей сути IDA-это дизассемблер рекурсивного спуска.

Рекурсивный спуск фокусируется на концепции потока управления, которая определяет, должна ли инструкция быть разобрана или нет, исходя из того, ссылается ли она на другую инструкцию. Для понимания рекурсивного спуска полезно классифицировать инструкции в соответствии с тем, как они влияют на указатель на команду процессора.

Чтобы преодолеть один из самых больших недостатков рекурсивного спуска, IDA использует большое количество эвристических методов для идентификации дополнительного кода, который, возможно, не был найден в процессе рекурсивного спуска.

Помимо самого процесса дизассемблирования, IDA прилагает усилия не только для того, чтобы отличить байты данных от байтов кода, но и для того, чтобы точно определить, какой тип данных представлен этими байтами данных.

Хотя код, который вы просматриваете в IDA, написан на ассемблере, одна из основных целей нарисовать картину, максимально приближенную к исходному коду.

Установка IDA на Windows - это очень простой процесс. Запуск установщика Windows проведет вас через несколько информационных диалогов, только один из которых требует каких-либо размышлений:

IDA содержит справочную документацию в стиле Windows, но это в первую очередь обзор пользовательского интерфейса IDA и подсистемы сценариев.

Приведем краткое описание каждого из подкаталогов в рамках установки IDA:

Каталог cfg содержит различные конфигурационные файлы, в том числе базовый конфигурационный файл IDA ida.cfg, файл конфигурации GUI idagui.cfg, а также текстовый файл конфигурации пользовательского интерфейса idatui. cfg.

Каталог ide содержит основные файлы, необходимые для работы со встроенным скриптовым языком IDA-IDA.

Каталог ids содержит файлы символов (IDS files на языке IDA), которые описывают содержимое общих библиотек, на которые могут ссылаться двоичные файлы, загруженные в IDA. Эти файлы идентификаторов содержат сводную информацию, в которой перечислены все записи, экспортируемые из данной библиотеки. Эти записи включают в себя информацию, описывающую тип и количество параметров, требуемых функцией, тип возвращаемого значения (если таковой имеется) функции и информацию о соглашении вызова, используемом функцией.

Каталог loaders содержит расширения IDA, которые используются в процессе загрузки файлов для распознавания и анализа известных форматов файлов, таких как PE или ELF-файлы.

Каталог plugins содержит модули IDA, предназначенные для обеспечения дополнительного и в большинстве случаев определяемого пользователем поведения IDA.

Каталог procs содержит процессорные модули, поддерживаемые установленной версией IDA. Процессорные модули обеспечивают возможность перевода с машинного языка на ассемблерный язык в рамках IDA и отвечают за создание языка ассемблера, отображаемого в пользовательском интерфейсе IDA.

Каталог sig содержит подписи для существующего кода, который IDA использует для различных операций сопоставления шаблонов. Именно благодаря такому сопоставлению шаблонов IDA может идентифицировать последовательности кода как известный библиотечный код, что потенциально экономит вам значительное количество времени в процессе анализа. Используемые подписи генерируются с помощью технологии быстрой идентификации и распознавания библиотек IDA (FLIRT).

Каталог til содержит информацию о библиотеке типов, которую IDA использует для записи макета структур данных, специфичных для различных библиотек компилятора.

Можно скачать программу IDA Pro Free (для некоммерческого использования) **с** официального сайта с рядом ограничений (версия, достаточная для первоначального знакомства с функционалом дизассемблера).

Для выполнения лабораторной необходимо детально изучить следующие пункты.

а) Сбор общей информации о файле.

При получении исполняемого файла, неизвестно его предназначение, кроме того, что он может выполниться и производить какие-либо действия. Для того чтобы узнать, какие точно действия он производит и каким образом, потребуется провести исследование, которое начинается с общего анализа файла.

Первое что необходимо узнать это платформа для его исполнения и разрядность файла. Данную информацию можно получить при анализе файла в шестнадцатеричном редакторе. При анализе заголовка файла, очень можно

понять платформу, для которой предназначен данный двоичный файл, также в заголовочной секции можно найти признаки принадлежности к той или иной разрядности.

На *Рисунок* рисунке 11*Рисунок* видно, что данный файл предназначен для Linux-подобных операционных систем и имеет разрядность 32-бита, однако будет запускаться и в 64-битной версии операционной системы.

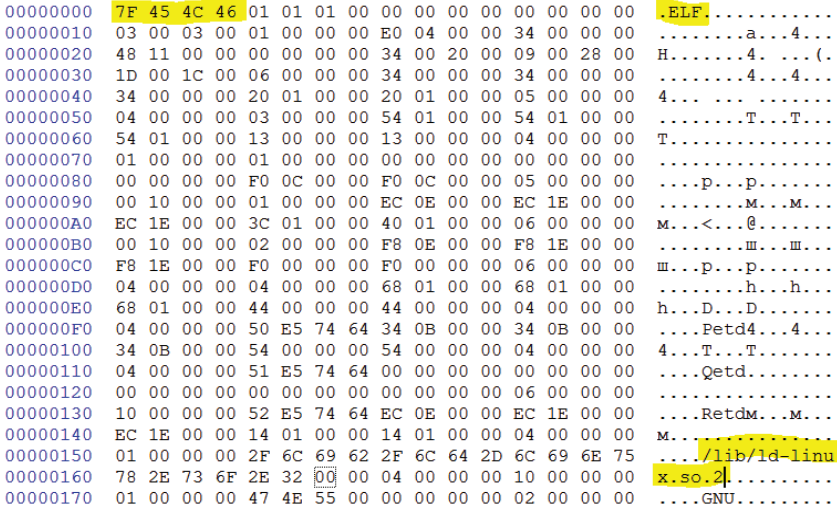

Рисунок 11 - Шестнадцатеричный дамп заголовка исполняемого файла

После определения платформы и разрядности исполняемого файла, можно производить загрузку данного файла в дизассемблер IDA Pro (далее IDA) для дальнейшего исследования.

После загрузки исполняемого файла в IDA чаще всего проверяются имеющиеся строки, для анализа некоторых функций.

Удобно использовать строки, на которые есть ссылки из этих функций. При отсутствии символьной информации, строки являются незаменимым помощником в распознавании важных функций и обработчиков.

Для получения полного списка строк можно воспользоваться сочетанием клавиш Shift+F12. После нажатия данных клавиш будет отображена отдельная вкладка со списком всех строк и их адресами, как представлено наРисунок 12.

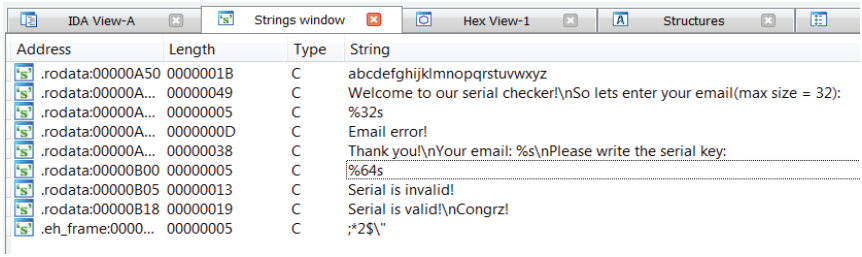

Рисунок 12 - Список строк, используемых в файле

После анализа полученной информации, касающейся строк, можно переходить непосредственно к восстановлению логики работы приложения.

б) Статический анализ дизассемблированного кода

Существуют различные подходы к решению данного вопроса, при анализе крупных проектов не принято восстанавливать логику работы всей системы, обычно исследуются конечные части, выполняющие несколько функций. При таком анализе обратная разработка начинается с какой-либо конечной функции, действие которой вы однозначно можете определить.

Однако в случае данной лабораторной работы можно начинать обратную разработку прямо с функции main(), так как функционал приложения крайне мал и однообразен, поэтому запутаться в функциях невозможно.

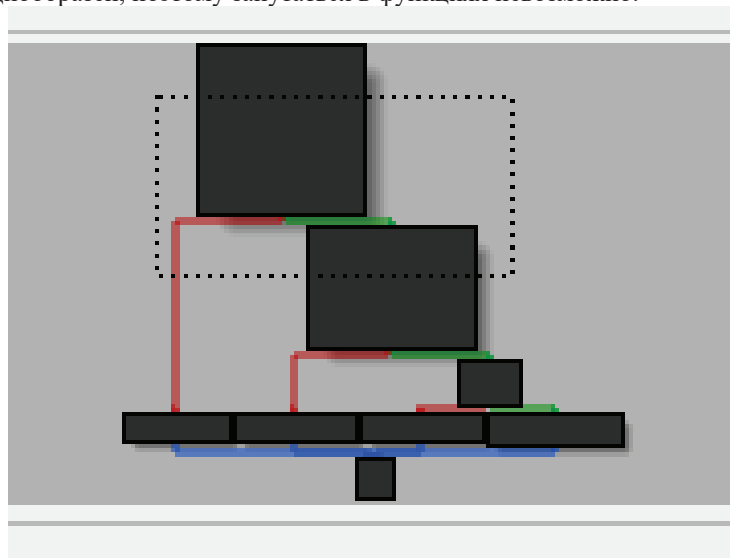

Рисунок 13 - Блок схема функции

Для удобного исследования кода рекомендуется применять различные типы отображения в IDA. Типы отображения кода можно менять с помощью клавиши пробел.

Режим графового отображения кода позволяет быстро ориентироваться между логическими блоками и быстрее анализировать логические условия, как представлено на рисунке 13.

При исследовании программ часто нужно обращать внимание на обработку вводимых данных, т.к. большая часть уязвимостей и ошибок в работе связана с недостаточно надежной и хорошей обработкой введённых данных.

При анализе приложений подобных приложению в данной лабораторной работе требуется чётко понимать, когда, как и куда поступают введённые вам данные. Рассмотрим реализацию передачи электронного адреса в функцию проверки в данной лабораторной работе как представлено на рисунке 14.

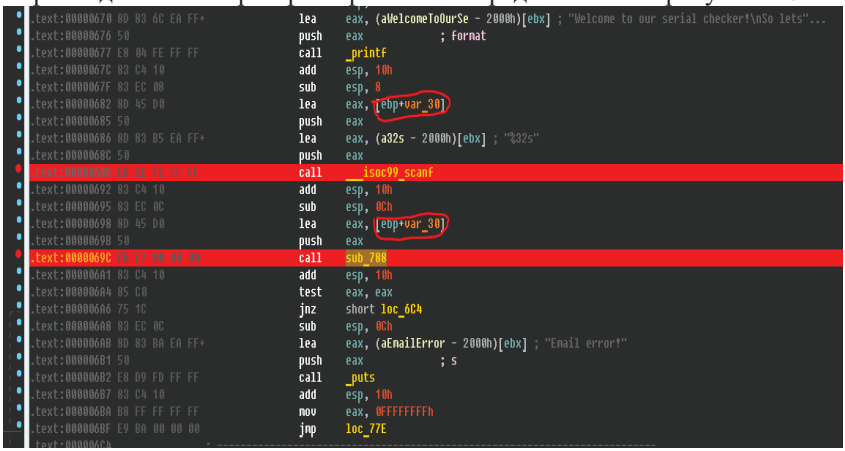

Рисунок 14 - Фрагмент кода функции main

В приведённом фрагменте кода функции main(), мы видим, что в функцию scanf() передаётся два параметра, один из которых - формат вводимых данных, а второй - это адрес буфера на стеке, куда и будут сохранены введённые данные. Передача параметров в функции осуществляются через стек.

После считывания наших данных происходит коррекция стека, и на стек поступает аргумент для следующей функции, имя для которой мы не имеем, так как она нестандартная и была написана программистом.

Однако при анализе аргумента можно заметить, что он является адресом на введённый буфер, хранимый на стеке, также можно заметить, что результат работы данной функции влияет на логическое выражение и разбивает ход исполнения на 2 возможных варианта, одним из которых является вариант с выводом фразы “Email error!”. Данных предположений и наблюдений будет

достаточно для формирования вывода, что данная функция является функцией проверки введённого адреса электронной почты.

в) Преобразование алгоритмов с языка низкого уровня на язык программирования высокого уровня.

После анализа алгоритмов, реализованных в исполняемом файле, часто возникает потребность реализовать их на языке программирования высокого уровня, реализация может быть абсолютно идентичная по функциональности, но иметь другие параметры или носить иной смысл. IDA предоставляет удобные упрощения для анализа алгоритмов на ассемблере, одно из таких упрощений уже было рассмотрено выше - графическое представление функции. Графы могут помочь в определении циклов и условий.

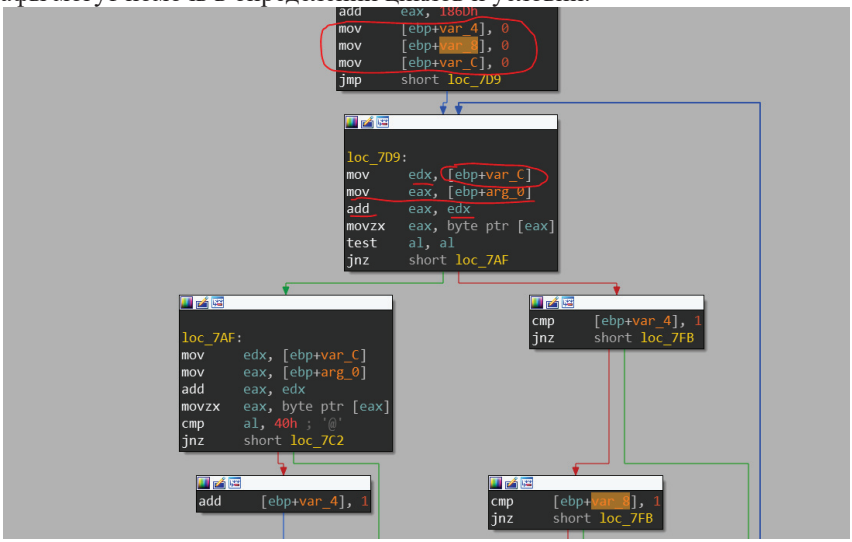

Рисунок 15 - Графическое отображение функции

Из Рисунок 15 видно, что перед входом в цикл происходит обнуление трех локальных переменных, одна из которых является счётчиком цикла. Счётчик цикла помещается в регистр **edx** в следующем блоке, далее прибавляется к адресу, помещенному в **eax**. В **eax** помещается адрес переданного в функцию аргумента, которым является адрес введённой строки. Далее в **eax** помещается очередной символ строки, и проверяется на Null-byte, так как все строки заканчиваются нулем и это будет служить сигналом для выхода из цикла.

Далее, по ветке выхода из цикла, можно обнаружить два сравнения локальных переменных с единицей.

При успешном прохождении данных сравнений произойдёт переход на ветку, представленную на Рисунок 16.

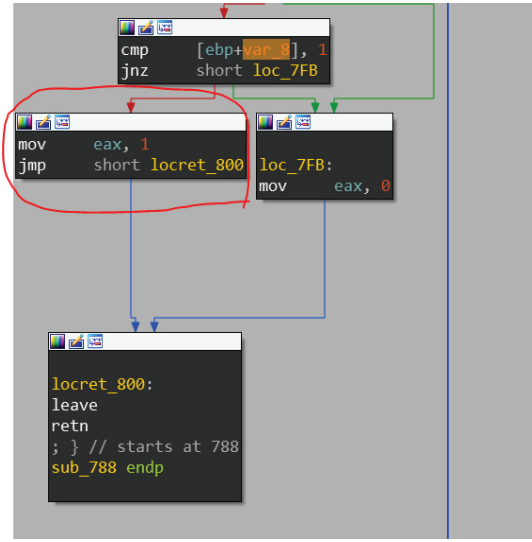

Рисунок 16 - Ветка выхода из функции

Данная ветка приводит к завершению работы функции с кодом «1», что означает, что проверка электронной почты прошла успешно.

То есть для верного прохождения проверки необходимо, чтобы данные локальные переменные были равны единице.

Для этого рассмотрим ветку, относящуюся к телу цикла на рисунке 17.

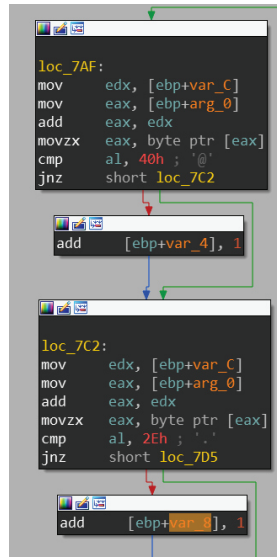

Рисунок 17 - Ветка тела цикла

В первом блоке тела цикла можно заметить те же инструкции, что были перед входом в цикл, соответственно очередное значение строки помещается в младший байт регистра **eax** и сравнивается с константой 40h, которая является ASCII кодом знака “@”, который используется в адресах электронной почты. Если символ строки равен данной константе, то локальная переменная увеличивается на единицу. То же самое происходит и в следующем блоке. Так как мы знаем, что для успешного выполнения нашей функции необходимо, чтобы обе локальные переменные были равны единице, значит, что в введенном нами адресе электронной почты должны обязательно присутствовать только по одному символу “.” и “@”.

После анализа функции можно получить готовый алгоритм проверки адреса, что позволит реализовать его на одном из языков программирования высокого уровня, например, на Python.

Реализация данной функции на Python приведена на рисунке 18.

```
1  def checker( inpEmail ):
2      _dFlag = 0
3      _aFlag = 0
4
5      for i in inpEmail:
6
7          if i == '.':
8              _dFlag += 1
9          elif i == '@':
10             _aFlag += 1
11
12     return ( _dFlag == 1 and _aFlag == 1 )
13
```

Рисунок 18 - Реализация данной функции на Python.

3. Порядок выполнения работы.

1. Получить индивидуальное задание на выполнение лабораторной работы.
2. Проанализировать алгоритм проверки электронной почты и пояснить принцип его работы.
3. Проанализировать алгоритм проверки серийного ключа и пояснить принцип его работы.
4. Произвести верный ввод ключа для одного любого адреса электронной почты.
5. Реализовать программу для автоматической генерации секретного ключа для любого введённого адреса электронной почты.

6. Сформулировать выводы по работе и составить отчёт.

4. Содержание отчета

1. Титульный лист.
2. Описание по каждому пункту проведённых вами действий с изображениями (скриншотами), исходный код программы генератора.
3. Вывод лабораторной работы, рекомендации для устранения ошибок безопасности, стиль оформления отчёта – научно-технический

5. Литература

1. А.С. Масалков: Особенности киберпреступлений в России. Инструменты нападения и защита информации. М.: ДМК-Пресс, 2018.
2. А.А. Бирюков: Информационная безопасность: защита и нападение. М.:ДМК Пресс, 2012.
3. А.С. Панов: Реверсинг и защита программ от взлома. Спб.: БХВ-Петербург, 2006.
4. Семенов Ю.А. Обзор уязвимостей, некоторых видов атак и средств защиты (http://book.itep.ru/6/intrusion.htm).

Лабораторная работа № 4

Анализ безопасности приложений. Базовые бинарные уязвимости

1. Цель работы - получить практические навыки в анализе двоичных приложений на наличие уязвимостей, научиться использовать уязвимости бинарных приложений и разобраться в концепциях их работы.

2. Краткие теоретические сведения

а) Общие понятия о бинарных уязвимостях.

Бинарные уязвимости являются одним из самых сложных классов уязвимостей в современном мире практической информационной безопасности связанной с анализом кода. Данные уязвимости сложно обнаружить и зачастую очень сложно использовать полностью. Первым усложняющим задачу фактором является анализ двоичного кода, а не исходного кода. При анализе двоичного кода даже большой командой профессионалов всё равно велик шанс упустить важные детали из виду.

Хорошим примером может служить операционная система Windows которую исследуют и разрабатывают большое количество профессионалов и несмотря на это уязвимостей в ней находят очень часто, в том числе и критические. Сложно переоценить значимость данных уязвимостей, ведь бинарные уязвимости также и находят в критически важных инфраструктурах, например, в автоматизированных системах, которые обеспечивают целые города благами жизни, или регулируют ядерное оружие. Всё высоконагруженное и высокотехничное оборудование работает на программном обеспечении разработанном на компилируемых языка и соответственно сразу потенциально подвержено такого рода уязвимостям.

Существует два больших класса уязвимостей: кода и архитектуры.

Уязвимости кода непосредственно связаны с неправильными обработками данных или недостаточной проверкой получаемых значений.

Уязвимости архитектуры представляют собой реализацию каких-либо возможностей системы вне задуманных целей для это возможности. Другими словами, вы используете те функции, которые вам доступны, но не в тех целях, для которых они подразумевались, при этом проверки целей не происходит концептуально.

б) Переполнение буфера.

Данная уязвимость считается самой популярной при исследованиях, несмотря на то, что разработано уже множество методик, позволяющих компилятору защищаться от подобного рода уязвимостей, они всё равно обнаруживаются и эксплуатируются, особенно в старом программном

обеспечении, которое не редко входит в состав нового программного обсечения как некие полезные модули.

Суть данной уязвимости заключается в том, что размер входного буфера данных зачастую не проверяется при перемещение его из одного места в другое или просто при считывании. При этом данная уязвимость не обязательно должна быть связана с переполнением стека или кучи с возможностью выполнения кода, также можно и перезаписывать некоторые локальные переменные при возникновении переполнений буфера. Пример уязвимости переполнения буфера представлен на рисунке 19.

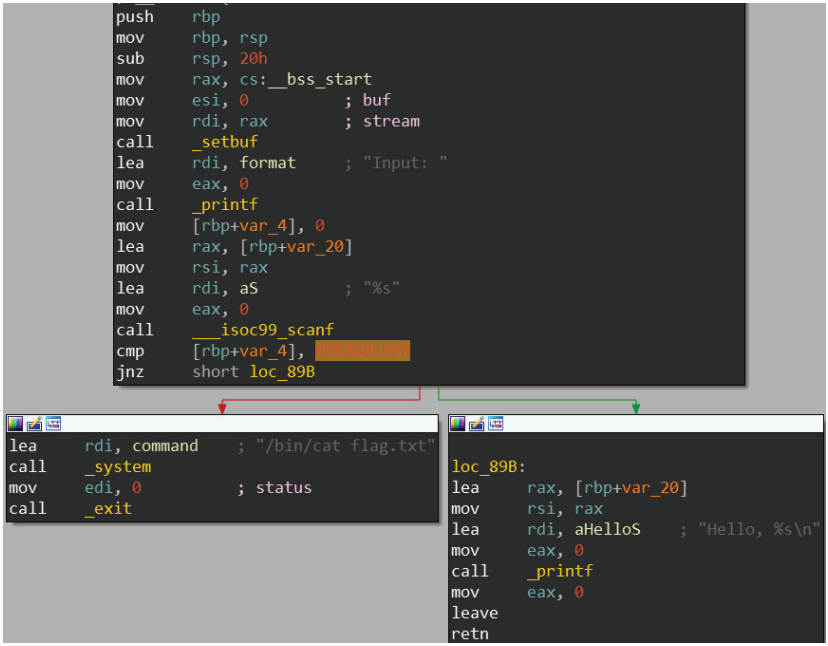

Рисунок 19 - Пример уязвимости переполнения буфера.

На данном рисунке представлен пример переполнения буфера, при этом в данном случае при переполнение будет произведена перезапись локальной переменной, но если попытаться перезаписывать значения дальше вплоть до адреса возврата, то произойдёт ошибка, связанная с нарушением целостности стека. Эта ошибка связана с наличием защищающих механизмов в компиляторе. Одним из таких механизмов является так называемая стековая канарейка, которая находится перед адресом возврата и соответственно нельзя перезаписать адрес возврата не перезаписав её, а её перезапись влечёт за собой аварийное завершение работы программы, так как обработчик обнаружит нарушение целостности стека. Это довольно простая технология очень хорошо

влияет на защиту программного обеспечения, если программист вдруг допустил ошибку.

Перезапись переменной произойдёт потому, что в функцию scanf() не был явно передан аргумент отвечающий за количество считываемых символов, однако массив в который данные символы будут считываться ограничен 16 байтами, которые располагаются на стеке. Там-же и располагается локальная переменная и если правильно высчитать смещение и передать необходимое количество байт для достижения адреса данной переменной, то последующие байты перезапишут значение данной переменной.

В данном примере это показано довольно искусственно, т.к. по условиям видно, что для того чтобы выполнилась команда чтения флага одна из локальных переменных должна быть равна значению **0xdeadbeef**, однако данная переменная нигде не изменяет своё значение из чего можно сделать вывод что от нас требуется изменить это значение при помощи переполнения буфера.

Для того, чтобы высчитать верное количество, байт которое нужно передать, чтобы попасть на перезапись необходимой переменной можно воспользоваться IDA и посмотреть разбиение стека для данной функции.

Рисунок 20 - Разбиение стека функции main.

Данную операцию можно выполнить с помощью горячих клавиш “Ctrl+K”, после их нажатия откроется новая вкладка, в которой будет указано разбиение стека для данной функции.

Как можно заметить по рисунку 16, от начала нашего буфера, который хранит введённые значения до адреса необходимой переменной 0x20-0x4 = 0x16 – байт, в переводе на десятичную систему 32-4=28 байт. То есть после передачи 28 байт следующие байты будут перезаписывать данную переменную.

Данный вывод довольно просто проверяется при помощи запуска исполняемого файла с передачей в него необходимых параметров, как на рисунке 21.

```
root@kali:~/pazi_pwn/task1# python -c 'print "a"*26 + "\xef\xbe\xad\xde"' | ./a.
out
Input: Hello, aaaaaaaaaaaaaaaaaaaaaaaaﾭ
root@kali:~/pazi_pwn/task1# python -c 'print "a"*28 + "\xef\xbe\xad\xde"' | ./a.
out
Input: PAZI{FLAG}
root@kali:~/pazi_pwn/task1# cat flag.txt
PAZI{FLAG}
root@kali:~/pazi_pwn/task1#
```

Рисунок 21 - Эксплуатация уязвимости.

Данный пример довольно искусственен, однако хорошо демонстрирует базовый принцип уязвимости переполнения буфера.

в) Уязвимость форматной строки.

Суть данной уязвимости заключается в том, что введённые пользователем данные могут передаваться в функции вывода на экран, при этом формат данных не будет указан явно, что позволяет пользователю выводит те данные на экран которые ему необходимы, например, содержание стека функции.

В контексте атаки данная уязвимость может быть полезна, например, для получения как раз той самой стековой канарейки, про которую было написано выше, при её получении мы можем перезаписать её верным значением и после перезаписать необходимым нам значением адрес возврата.

Рассмотрим пример. На рисунке 18 видно, что введённые пользователем данные в программ считываются и добавляются в буфер, далее данный буфере предаётся в функцию printf() однако, формат передаваемых данных никак не указывается, что позволяет манипулировать выводом данной функции в различных целях. Целью данного примера является возможность чтения контрольного значения PAZI{flag} которое можно заметить в функции.

На практике можно с пользой применять данную уязвимость, например, если она была найдена во встраиваемой системе под которую нет отладчика, с помощью данной уязвимости можно следить за значениями на стеке.

Рисунок 22 - Пример уязвимости форматной строки.

Для выполнения цели предусмотренной данным искусственным примером достаточно произвести определённый форматный ввод и немного обработать вывод. Ввод представлен на рисунке 23, после получения объёма шестнадцатеричных цифр можно выделить в них ASCII-коды формата флага и найти флаг.

```
root@kali:~/pazi_pwn/task1# ./task2
Input: 0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%
08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p
,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,0x%08p,
Your input:
0x0x7f7c021a7683,0x0x7f7c021a8760,0x0x7f7c01eea720,0x0x7f7c023aa700,0x   (nil),0
x0x5a41500000000000,0x0x7d67616c667b49,0x0x302c703830257830,0x0x78302c7038302578
,0x0x2578302c70383025,0x0x302578302c703830,0x0x38302578302c7038,0x0x703830257830
2c70,0x0x2c7038302578302c,0x0x302c703830257830,0x0x78302c7038302578,0x0x2578302c
70383025,0x0x302578302c703830,0x0x38302578302c7038,0x0x7038302578302c70,0x0x2c70
38302578302c,0x0x302c703830257830,0x0x78302c7038302578,0x0x2578302c70383025,0x0x
302578302c703830,0x0x38302578302c7038,0x0x7038302578302c70,0x0x2c7038302578302c,
0x0x302c703830257830,root@kali:~/pazi_pwn/task1# 
```
Рисунок 23 - Пример эксплуатации уязвимости форматной строки.

3. Порядок выполнения работы.

1. Получить индивидуальное задание на выполнение лабораторной работы.
2. Обнаружить уязвимости в представленных исполняемых файлах и описать каждую из уязвимостей, пояснив принципы её работы и критичность.
3. Обеспечить использование вами найденных уязвимостей и описать принципы их эксплуатации.
4. Сформулировать выводы по работе и составить отчёт.

4. Содержание отчета

1. Титульный лист.
2. Описание по каждому пункту проведённых вами действий с изображениями (скриншотами), пояснение каждой из найденных уязвимостей.
3. Вывод лабораторной работы, рекомендации для устранения ошибок безопасности, стиль оформления отчёта – научно-технический

5. Литература

1. А.С. Масалков: Особенности киберпреступлений в России. Инструменты нападения и защита информации. М.: ДМК-Пресс, 2018.
2. А.А. Бирюков: Информационная безопасность: защита и нападение. М.:ДМК Пресс, 2012.
3. А.С. Панов: Реверсинг и защита программ от взлома. Спб.: БХВ-Петербург, 2006.
4. Семенов Ю.А. Обзор уязвимостей, некоторых видов атак и средств защиты (http://book.itep.ru/6/intrusion.htm).