

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное бюджетное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт
Факультет приборостроения,
информационных технологий и
электроники

Кафедра
«Информационная безопасность
систем и технологий»

Специальность 10.05.03 -

Информационная безопасность
автоматизированных систем

Специализация -

Защищенные автоматизированные
системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ЛАБОРАТОРНЫЙ СТЕНД ДЛЯ ИССЛЕДОВАНИЯ УЯЗВИМОСТЕЙ
ПРОГРАММНЫХ СРЕДСТВ ЗАЩИЩЕННЫХ ТЕЛЕКОММУНИКАЦИОННЫХ
СИСТЕМ

Студент

 Зверев О. В.

Руководитель к.т.н.доцент

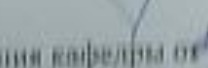
 Борисенко С. Н.

Нормоконтролер к.т.н.доцент

 Иванов А. Н.

Рецензент начальник сектора АО «ПНИЭИ»

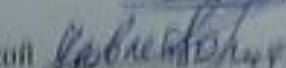
 Зорин А. Н.

Работа допущена к защите (протокол заседания кафедры от  № 16)

Заведующий кафедрой

С. Л. Зефирова

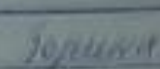
Работа защищена с отметкой

 (протокол заседания ГЭК

от  № 16)

А. Н. Иванов

Секретарь ГЭК

Подпись  И. Л. Орлов
Ведущий инженер  И. Л. Орлов

Получено 10.11.2011
Отдел кадров

Оглавление

Введение.....	4
1 Обзор возможных уязвимостей программных средств.....	5
1.1 Переполнение буфера.....	5
1.2 Переполнение целых чисел.....	8
1.3 Состояние «гонки» в программном обеспечении.....	15
2 Обзор возможных уязвимостей веб-приложений по OWASP TOP 10.....	17
3 Конфигурирование виртуального окружения для разработки, размещения и эксплуатации программных средств, содержащих уязвимости	25
4 Разработка программного кода, содержащего уязвимости.....	38
4.1 Разработка программных средств, содержащих уязвимость переполнение буфера.....	38
4.2 Разработка программных средств, содержащего уязвимость целочисленное переполнение.....	40
4.3 Разработка программных средства, содержащего уязвимость состояние «гонки».....	41
5 Описание процесса эксплуатации уязвимостей программного кода	43
5.1 Процесс эксплуатации уязвимости переполнение буфера.....	43
5.2 Процесс эксплуатации уязвимости целочисленное переполнение....	45
5.3 Процесс эксплуатации уязвимости состояние «гонки»... ..	48
6 Описание угроз безопасности информации возникающих при разработке программных средств и методы разработки безопасных программных средств.....	58
7 Рекомендации по проведению лабораторного практикума по исследованию уязвимостей программного кода.....	61
8 Вредные психофизиологические факторы, влияющие на опорно-двигательную систему специалиста по информационной безопасности. Методы и средства физической культуры, снижающие их воздействие.....	61

8.1 Вредные психофизиологические факторы, влияющие на опорно-двигательный аппарат специалиста по защите информации.....	61
8.2 Методы и средства физической культуры, снижающие их пагубное воздействие.....	63
Заключение	65
Список используемых источников.....	66
Приложение А Состав лабораторного виртуального стенда, использующегося при выполнении проект.....	68
Приложение Б Обзор уязвимостей программных средств	70
Приложение В Угрозы безопасности информации, возникающие при разработке программных средств согласно ГОСТ Р 58412-2019	72
Приложение Г Методы защиты от угроз безопасности, возникающих при разработке программных средств согласно ГОСТ Р 56939-2016.....	74
Приложение Д Схема эксплуатации программных средств имеющего уязвимость состояние гонки	76
Приложение Е Схема эксплуатации программных средств имеющего уязвимость переполнение буфера.....	78
Приложение Ж Схема эксплуатации программного кода имеющего уязвимость целочисленное переполнение.....	80

Введение

Актуальность и практический аспект исследования уязвимостей программного кода связан с тем, что в современном обществе злоумышленники способны похитить информацию пользователя, навредить правильной работоспособности программы путем атак на программное обеспечение.

Методы обеспечения безопасной эксплуатации программного обеспечения является важнейшей задачей использования программного обеспечения. Также это необходимо для предотвращения хищения информации пользователя и стабильной работоспособности программного обеспечения.

Для решения проблемы уязвимостей программного обеспечения было выявлено множество атак, в которых были использованы уязвимости, рассмотренные в данной работе. Особенностью данных атак в основном является сбой или неправильность работоспособности программы.

В данной работе были рассмотрены некоторые атаки на программное обеспечение, анализ безопасной работоспособности программного обеспечения путем предотвращения данных атак.

Целью выпускной квалификационной работы является разработка лабораторного стенда по исследованию уязвимостей программного кода, разработки рекомендаций для выполнения лабораторных работ с использованием лабораторного стенда по исследованию уязвимостей программного кода.

Стенд предназначен для проведения лабораторных работ по дисциплине «Защита программных средств защищенных телекоммуникационных систем».

1 Обзор возможных уязвимостей программного кода

1.1 Переполнение буфера

Переполение буфера – это проблема всех низкоуровневых языков программирования. Возникает данная уязвимость потому, что в целях эффективности данные и информация о потоке выполнения программы перемешаны, а в низкоуровневом языке разрешен прямой доступ к памяти. Языки программирования. С и С++ больше других подвержены переполению буфера. Результаты переполения буфера могут быть различные, например, крах программы или получение противником полного контроля над приложением, а если приложение запущено от имени пользователя с высоким уровнем доступа (root, Administrator или System), то и над всей операционной системой и другими пользователями. Если рассматриваемое приложение – это сетевая служба, то переполение буфера может привести к распространению червя. Первый, получивший широкую известность Интернет–червь, эксплуатировал ошибку в сервере finger, он так и назывался – «finger–червь Роберта Т. Морриса» (или просто «червь Морриса»). После того как в 1988 году Интернет был на самом низком уровне, должны были научиться избегать уязвимости переполения буфера, но этого до сих пор нередко появляются сообщения о такого рода ошибках в самых разных программах.

Чаще всего уязвимость переполение буфера встречается в программах, написанных на языке программирования С. Язык С++ так же небезопасен, так как основан на этом языке. Однако, использование стандартной библиотеки шаблонов STL позволяет свести риск некорректной работы со строками к минимуму, а более строгий компилятор С++ помогает программисту избегать некоторых ошибок. Даже если программа составлена на чистом языке С, все же

рекомендуется использовать компилятор C++, чтобы можно было выявить как можно больше ошибок.

В языках более высокого уровня, которые появились некоторое время спустя, программист уже не имеет прямого доступа к памяти, хотя за это и приходится расплачиваться производительностью. В такие языки, как Java, C# и Visual Basic, уже встроены строковый тип, массивы с контролем выхода за границы и запрет на прямой доступ к памяти (в стандартном режиме). Казалось бы, в таких языках переполнение буфера невозможно, но правильнее было бы считать, что данная уязвимость лишь гораздо менее вероятна. Это происходит потому, что эти языки реализованы на C или C++, а ошибка в реализации может стать причиной переполнения буфера. Еще один потенциальный источник проблемы заключается в том, что на какой-то стадии все эти высокоуровневые языки должны обращаться к операционной системе, а уж она-то почти наверняка написана на C или C++. Язык C# позволяет обойти стандартные механизмы .NET, объявив небезопасный участок с помощью ключевого слова `unsafe`. Да, это упрощает взаимодействие с операционной системой и библиотеками, которые написаны на C/C++, но одновременно открывает возможность допустить обычные для C/C++ ошибки. Даже если программирование осуществляется преимущественно на языках высокого уровня, следует тщательно контролировать данные, передаваемые внешним библиотекам.

В программах на языках C/C++ есть достаточное количество способов переполнить буфер.

Ниже представлены строки, с помощью которых был реализуем `finger-червь Морриса`:

```
char buf[20] ;  
gets (buf) ;
```

Не существует способа вызвать команду `gets` для чтения из стандартного ввода без риска переполнить буфер. Вместо этого стоит воспользоваться командой `fgets`. Следующий способ вызвать переполнение буфера – это

воспользоваться функцией `strcpy`. Ниже представлен фрагмент кода, вызывающего уязвимость переполнение буфера с помощью функции `strcpy`.

```
char buf[20];
char prefix[] = "http://";
strcpy(buf, prefix);
strncat(buf, path, sizeof(buf));
```

Проблема заключается в неудачном интерфейсе функции `strncat`. Этой функции необходимо указывать количество символов свободных в буфере, а не общую длину буфера. Далее представлен еще один фрагмент кода, приводящий к переполнению буфера:

```
char buf[MAX_PATH];
sprintf(buf, "%s – %d\n", path, errno);
```

Функцию `sprintf` почти невозможно использовать безопасно. Для Microsoft Windows было выпущено оповещение о критической ошибке, которая была связана с применением `sprintf` для отладочного протоколирования.

```
char buf [ 32] ;
strncpy(buf, data, strlen(data));
```

В последнем аргументе передана длина входного буфера, а не размер целевого буфера. Один из распространенных способов столкнуться с проблемой – считать байты вместо символов. Если работает с кодировкой ASCII, то между ними нет разницы, но в кодировке Unicode один символ представляется двумя байтами. Ниже представлен пример:

```
_snwprintf(wbuf, sizeof(wbuf), «%s\n», input);
```

Следующее переполнение несколько интереснее:

```
bool CopyStructs(InputFile* pInFile, unsigned long count)
{
    unsigned long i;
    m_pStructs = new Structs[count];
    for(i = 0; i < count; i++)
    {
```

```
if(!ReadFromFile(pInFile, &(m_pStructs[i])))  
break;  
}  
}
```

В данном отрывке кода оператор `new[]` делает примерно то же, что и такой код:

```
ptr = malloc(sizeof(type) * count);
```

Если значение `count` поступает от пользователя, то нетрудно задать его так, чтобы при умножении возникло переполнение. После этого будет выделен буфер гораздо меньшего размера, чем необходимо, и противник сможет его переполнить. В компиляторе C++, который поставляется в составе Microsoft Visual Studio 2005, производится внутренняя проверка для недопущения таких ошибок. Такая же проблема может возникнуть во многих случаях использования функции `calloc`, которая выполняет такую же операцию. В этом и состоит опасность многих ошибок, связанных с переполнением буфера:

- разрушение стека (`stack smashing`) — стандартный метод переполнения буфера для изменения адреса возврата функции;
- перенаправление указателя (`pointer subterfuge`) — перезапись локального указателя с целью поместить данные в нужное место;
- атака на регистр (`register attack`) — перезапись значения, хранимого в регистре (например в EBP), для получения управления;
- захват VTable (`VTable hijacking`) — изменение локальной ссылки на объект так, чтобы вызов VTable приводил к запуску нужной функции;
- захламление обработчиков исключений (`exception handler clobbering`) — изменение кода обработки исключения, заставляющее систему выполнить подставленный взломщиком код;
- выход индекса за границы диапазона (`index out of range`) — использование индекса массива, который не проверяется на соответствие разрешенному диапазону;

– переполнения кучи (heap overflow) — принуждение диспетчера кучи к выполнению умысла хакера.

1.2 Переполнение целых чисел

Суть проблемы переполнения целых чисел заключается в том, что какой бы формат для представления чисел ни выбрать, существуют операции, которые при выполнении компьютером дают не тот же результат, что при вычислениях на бумаге. Существуют исключения - в некоторых языках программирования реализованы целочисленные типы переменных, но это встречается редко.

В других языках программирования, таких как Ada реализованы целочисленные типы переменных с проверкой диапазона, и если ими пользоваться постоянно, то вероятность ошибки снижается. В языках программирования C и C++ применяются настоящие целые типы. В современных версиях Visual Basic все числа представляются типом Variant, где хранятся как числа с плавающей точкой; если объявить переменную типа int и записать в нее результат деления 5 на 4, то получится не 1, а 1.25. У Perl свой подход. В C# проблема усугубляется тем, что иногда этот язык требует использования целых чисел со знаком, но затем производится проверка с помощью ключевого слова «checked».

Результатом переполнения целых чисел может быть все, что угодно: логическая ошибка, аварийная остановка программы, или исполнение произвольного кода. Большинство современных атак направлены на то, чтобы заставить приложение допустить ошибку при выделении памяти, после чего противник сможет воспользоваться переполнением. Даже если работать на языках, отличных от C/C++, все равно гарантий защиты от переполнений целого нет. Логический просчет, который возник в результате усечения целого, стал причиной ошибки в сетевой файловой системе NFS (Network File System),

из-за которого любой пользователь мог получить доступ к файлам от имени root.

Поскольку язык C сравнительно низкого уровня, то он приносит безопасность в жертву быстродействию и готов преподнести целый ряд неприятностей при работе с целыми числами. Большинство других языков на такое не способны, а язык программирования C, проделывает небезопасные вещи, только если ему явно разрешить.

Есть несколько типичных ситуаций, которые приводят к переполнению целых чисел. Одна из самых частых – незнание с порядком приведений и неявными приведениями, которые осуществляют некоторые операторы. Пример операции проведения представлен ниже.

```
const long MAX_LEN = 0x7fff;  
short len = strlen(input);  
if (len < MAX_LEN)
```

Если не обращать внимания на усечение, то существует вопрос: в каком порядке производятся приведения типов при сравнении len и MAX_LEN? Стандарт языка гласит, что повышающее приведение следует выполнять перед сравнением; следовательно, len будет преобразовано из 16-разрядного целого со знаком в 32-разрядное целое со знаком. Это простое приведение, так как оба типа знаковые. Чтобы сохранить значение числа, оно расширяется с сохранением знака до более широкого типа. В данном случае мог бы получиться такой результат:

```
len = 0x100;  
(long)len = 0x00000100;
```

Или

```
len = 0xffff;  
(long)len = 0xffffffff;
```

Поэтому если противник сумеет добиться того, чтобы len превысило 32К, то len станет отрицательным и останется таковым после расширения до 32

битов. Следовательно, после сравнения с MAX_LEN программа пойдет по неверному пути.

Целое со знаком в более широкое целое со знаком. Меньшее значение расширяется со знаком, например приведение (char)0x7f к int дает 0x0000007f, но (char)0x80 становится равно 0xffffffff80.

Целое со знаком в целое без знака того же размера. Комбинация битов сохраняется, значение может измениться или остаться неизменным. Так, (char)0xff (-1) после приведения к типу unsigned char становится равно 0xff, но ясно, что -1 и 255 – это не одно и то же.

Целое со знаком в более широкое целое без знака. Здесь сочетаются два предыдущих правила. Сначала производится расширение со знаком до знакового типа нужного размера, а затем приведение с сохранением комбинации битов. Это означает, что положительные числа ведут себя ожидаемым образом, а отрицательные могут дать неожиданный результат.

Например, (char) -1 (0xff) после приведения к типу unsigned long становится равно 4 294 967 295 (0xffffffff).

Целое без знака в более широкое целое без знака. Это простейший случай: новое число дополняется нулями, что ожидаемо. Соответственно, (unsigned char)0xff после приведения к типу unsigned long становится равно 0x000000ff.

Целое без знака в целое со знаком того же размера. Так же как при приведении целого со знаком к целому без знака, комбинация битов сохраняется, а значение может измениться в зависимости от того, был ли старший (знаковый) бит равен 1 или 0.

Целое без знака в более широкое целое со знаком. Так же как при приведении целого без знака к более широкому целому без знака, значение сначала дополняется нулями до нужного без знакового типа, а затем приводится к знаковому типу. Значение не изменяется, так что ничего в этом случае не бывает.

Понижающее приведение. Если в исходном числе хотя бы один из старших битов был отличен от нуля, то мы имеем усечение, что вполне может привести

к печальным последствиям. Число без знака станет отрицательным или произойдет потеря информации. Если речь не идет о битовых масках, следует проверять, было ли усечение.

Преобразования при вызове операторов. Большинство программистов не подозревают, что одного лишь обращения к оператору достаточно для изменения типа результата. Обычно ничего страшного не происходит, но граничные случаи могут неприятно удивить. Ниже представлен код на C++, который показывает данную проблему:

```
template <typename T>
void WhatIsIt(T value)
{
    if((T)-1 < 0)
        printf("Со знаком");
    else
        printf("Без знака");
    printf(" – %d бит\n", sizeof(T)*8);
}
```

Если хотя бы один операнд имеет тип `unsigned long`, то оба операнда приводятся к типу `unsigned long`. `Long` и `int` – это два разных типа, но на современных машинах они оба имеют длину 32 бита, поэтому компилятор считает их эквивалентными.

В иных случаях, когда длина операнда составляет 32 бита или меньше, операнды расширяются до типа `int`, и результатом является значение типа `int`.

Как правило, ничего неожиданного при этом не происходит, и неявное приведение в результате применения операторов может помочь избежать некоторых переполнений. Но бывают и исключения. Во-первых, в системах, где имеется тип 64-разрядного целого, ожидаемо скорое приведение `unsigned short` и `signed short` к `int`, а операторное приведение не нарушает корректность результата, то `unsigned int` и `signed int` будут приводиться к 64-разрядному типу

(`_int64`). Но это так не работает до тех пор, когда стандарт C/C++ не станет трактовать 64-разрядные целые так же, как остальные.

Следующая неожиданность заключается в том, что поведение изменяется в зависимости от оператора. Все арифметические операторы (`+`, `-`, `*`, `/`, `%`) подчиняются приведенным выше правилам. Но им же подчиняются и поразрядные бинарные операторы (`&`, `|`, ^A); поэтому `(unsigned short) | (unsigned short)` дает `int`. Те же правила в языке C распространяются на булевские операторы (`&&`, `||` и `!`), в то время как в C++ возвращается значение встроенного типа `bool`. Некоторые унарные операторы модифицируют тип, а некоторые — нет. Оператор дополнения до единицы (`~`) изменяет тип результата, следовательно `((unsigned short)0)` дает `int`, в то время как операторы префиксного и постфиксного инкремента и декремента (`++`, `--`) типа не меняют.

Код для проверки того, возникнет ли переполнение при сложении двух 16-разрядных целых без знака представлен ниже:

```
bool IsValidAddition(unsigned short x, unsigned short y)
{
    if(x + y < x)
        return false;
    return true;
}
```

Если результат сложения двух положительных чисел оказывается меньше какого-то слагаемого, значит что-то не в порядке. Точно такой же код должен работать и для чисел типа `unsigned long`. Компилятор оптимизирует всю функцию так, что она будет возвращать `true`.

Результат выполнения операции `unsigned short + unsigned short - int`. Неважно какие значения целых без знака, результат не может переполнить тип `int`, поэтому сложение всегда выполняется правильно. После чего, `int` сравнивается с `unsigned short`. Значение `x` приводится к типу `int` и оно уже, никогда не будет больше `x + y`. Чтобы исправить эту ошибку, следует привести результат обратно к `unsigned short`:

```
if((unsigned short)(x + y) < x)
```

Поразрядные операции AND, OR и XOR (исключающее или) должны работать, но расширение со знаком не позволяет правильно это делать. Рассмотрим пример:

```
int flags = 0x7f;
```

```
char LowByte = 0x80;
```

```
if ((char)flags ^ LowByte == 0xff)
```

Результатом операции должно быть 0xff, именно с этим значением и производится сравнение, но компилятор делает все иначе и приводит оба операнда к типу int. Для поразрядных операций выполняется приведение к int, если операнды имеют более узкий тип. Поэтому flags расширяется до 0x0000007f, и тут ничего плохого нет, но LowByte расширяется до 0xffffffff80, в результате операции получается значение 0xfffffffff.

Если переменные не имеют знака, результат будет положительным числом, но высшие разряды могут перейти в низшие. Две беззнаковые 16-разрядные целочисленные переменные, которые содержат значение 40 000. Если они перемножаются, а результат сохраняется в другой беззнаковой 16-разрядной целочисленной переменной, то видимым произведением будет 4096. Результат неверен, но этот факт не обнаруживается.

Возможность вызывать неопределяемые числовые переполнения. Один из способов атаки заключается в том, что программе предоставляется 2 допустимых (но больших) параметра, таких, что любая операция сложения или умножения вызовет переполнение. Некоторые программы для работы с графикой поддерживают параметры командной строки, которые задают высоту и ширину файла с изображением и размер, к которому будет преобразовано входное изображение. Если ширина и высота были выбраны, чтобы вызвать переполнение, то программа неправильно вычислит сколько памяти ей понадобится для хранения изображения, и вызовет процедуру malloc для распределения, отводя для него слишком маленький буфер. Теперь ситуация созрела для атаки за счет переполнения буфера. Такие же средства атаки можно

применить, когда сумма или произведение положительных целых чисел со знаком в результате оказываются отрицательным целым числом.

Выбор в качестве средства атаки условия состязательности. Программа согласно установке бита SETUID принадлежит пользователю root и взломщик хочет воспользоваться ее привилегиями для записи в файл паролей. Он не обладает разрешениями на запись в файл паролей, но при обзоре кода можно заметить то, что SETUID-программа не предназначена для записи в файл паролей, она лишь осуществляет запись в файл имеющим название `my document` в текущем рабочем каталоге. Пользователь может иметь этот файл в своем текущем рабочем каталоге, но это не означает, что он действительно имеет права записи в этот файл. Например, файл может быть символьной ссылкой на другой файл, который вообще не принадлежит пользователю, например на файл паролей.

1.3 Состояние «гонки» в программном обеспечении

Состояние гонки — это ошибка проектирования многопоточной системы или приложения, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода.

Состояние гонки — это «плавающая» ошибка, которая проявляется в случайные моменты времени и «пропадающая» при попытке ее локализовать. Данная уязвимость может приводить к нежелательному поведению данной программы в процессе ее исполнения из-за временных задержек в работе.

Программа содержит последовательность кода, который может исполняться параллельно с другим кодом и требует временный исключительный доступ к разделяемому (shared) ресурсу. Но существует временное окно, в котором разделяемый ресурс может быть изменен другой последовательностью кода, которая исполняется параллельно. Пример состояния гонок представлен на рисунке 1.

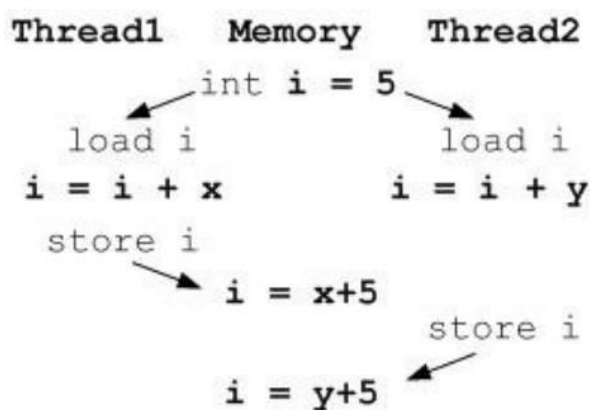


Рисунок 1 – Пример состояния гонки

Ниже представлен пример кода на языке программирования Java:

```

volatile int x;
//Поток 1:
while (!stop) {
x++;
...
}
//Поток 2: while (!stop) {
if (x%2 == 0) System.out.println("x=" + x);
...
}

```

Считается, что $x=0$. Выполнение программы происходит в следующем порядке:

Оператор `if` в потоке 2 проверяет x на четность.

Оператор « $x++$ » в потоке 1 увеличивает x на единицу.

Оператор вывода в потоке 2 выводит « $x=1$ », хотя, казалось бы, переменная проверена на четность.

Изначально в уязвимом драйвере `n_hdlc` для передаваемых данных использовался самописный односвязный список буферов. При ошибке передачи данных в указатель `n_hdlc.tbuf` записывается адрес текущего буфера. Это так и работало до тех пор, пока в 2009 году был принят коммит `be10eb75893`, который добавил поддержку сброса данных и внес состояние

гонки при обращении к `n_hdlc.tbuf`. Если при отправке данных произошла ошибка, то в последующем функции и обращаются к `n_hdlc.tbuf` и при одновременном исполнении могут дважды поместить один и тот же буфер в список `tx_free_buf_list`. Это приводит к двойному освобождению памяти. Буферы данных представлены структурой `n_hdlc_buf` и выделяются в slab-кеше `kmalloc-8192`.

2 Обзор возможных уязвимостей веб-приложений по OWASP TOP 10

Open Web Application Security Project (OWASP) – это некоммерческая организация, а также открытое интернет-сообщество, целью которого является защита организаций посредством разработки безопасного кода, тестирования на проникновение и сопровождения разрабатываемых приложений на всех этапах проекта. OWASP использует различные ресурсы (люди, технологии, процессы), чтобы решить существующие и возникающие проблемы в разработке безопасных приложений. Это происходит с помощью внедрения библиотек и применения инструментов безопасности, предоставляемых OWASP. Чтобы обеспечить долгосрочный успех проекта, все участники организации OWASP являются добровольцами, включая сам совет директоров OWASP, руководителей отделений, проектов и самих участников. Наиболее опасные уязвимости представлены в таблице 1.

Таблица 1 – ТОП 10 OWASP

Название уязвимости	Описание уязвимости
A1:2017 Инъекции	Уязвимости связанные, например, с реализацией SQL,MySQL, OS и LDAP, возникают, когда непроверенные данные отправляются интерпретатору в составе команды или запроса. Вредоносные данные могут заставить интерпретатор выполнить непредусмотренные команды или обратиться к данным без прохождения соответствующей авторизации.
A2:2017 Недостатки аутентификации	Функции приложений, связанные с аутентификацией и управлением сессиями, часто некорректно реализуются, позволяя злоумышленникам скомпрометировать пароли, ключи или сессионные токены, а также эксплуатировать другие ошибки реализации для временного или постоянного перехвата учетных записей пользователей.
A3:2017 Разглашение конфиденциальных данных	Большинство веб-приложений и API имеют плохую защиту критичных финансовых, медицинских или персональных данных. Злоумышленники могут похитить или изменить эти данные, а затем осуществить

Продолжение таблицы 1.

Название уязвимости	Описание уязвимости
A3:2017 Разглашение конфиденциальных данных	мошеннические действия с кредитными картами или персональными данными. Конфиденциальные данные требуют дополнительных мер защиты, например их шифрования при хранении или передаче, а также специальных мер предосторожности при работе с браузером.
A4:2017 Внешние сущности XML (XXE)	Старые или плохо настроенные XML-процессоры обрабатывают ссылки на внешние сущности внутри документов. Эти сущности могут быть использованы для доступа к внутренним файлам через обработчики URI файлов, общие папки, сканирование портов, удаленное выполнения кода и отказ в обслуживании.
A5:2017 Недостатки контроля доступа	Действия, разрешенные аутентифицированным пользователям, зачастую некорректно контролируются. Злоумышленники могут воспользоваться этими недостатками и получить несанкционированный доступ к учетным записям других пользователей или конфиденциальной информации, а также изменить пользовательские данные или права доступа.
A6:2017 Некорректная настройка параметров безопасности	Неправильная конфигурация безопасности является распространенной ошибкой. Это происходит из-за использования стандартных параметров безопасности, неполной или специфичной настройки, открытого облачного хранения, некорректных HTTP-заголовков и подробных сообщений об ошибках, содержащих критичные данные. Все ОС, фреймворки, библиотеки и приложения должны быть не только настроены должным образом, но и своевременно корректироваться и обновляться.
A7:2017 Межсайтовое выполнение сценариев (XSS)	XSS имеет место, когда приложение добавляет непроверенные данные на новую веб-страницу без их соответствующей проверки или преобразования, или когда обновляет открытую страницу через API браузера, используя предоставленные пользователем данные, содержащие HTML- или JavaScript-код. С помощью XSS злоумышленники могут выполнять сценарии в браузере жертвы, позволяющие им перехватывать пользовательские сессии, подменять страницы сайта или перенаправлять пользователей на вредоносные сайты.

Продолжение таблицы 1

Название уязвимости	Описание уязвимости
A8:2017 Небезопасная десериализация	Небезопасная десериализация часто приводит к удаленному выполнению кода. Ошибки десериализации, не приводящие к удаленному выполнению кода, могут быть использованы для атак с повторным воспроизведением, внедрением и повышением привилегий.
A9:2017 Использование компонентов с известными уязвимостями	Компоненты, такие как библиотеки, фреймворки и программные модули, запускаются с привилегиями приложения. Эксплуатация уязвимого компонента может привести к потере данных или перехвату контроля над сервером. Использование приложениями и API компонентов с известными уязвимостями может нарушить защиту приложения и привести к серьезным последствиям.
A10:2017 Недостатки журналирования и мониторинга	Недостатки журналирования и мониторинга, а также отсутствие или неэффективное использование системы реагирования на инциденты, позволяет злоумышленникам развить атаку, скрыть свое присутствие и проникнуть в другие системы, а также изменить, извлечь или уничтожить данные. Проникновение в систему обычно обнаруживают только через 200 дней и, как правило, сторонние исследователи, а не в рамках внутренних проверок или мониторинга.

Более подробно, какие бывают источники угроз, а так же последствия таких атак на компании и организации приведено в таблице 2.

Таблица 2 –Сведения об уязвимостях с указанием вектора атак,недостатков безопасности и последствий.

Уязвимость	Векторы атак	Недостатки безопасности	Последствия
A1: 2017	Практически любой источник данных может быть вектором для атаки: переменные среды, параметры, внешние и внутренние веб-сервисы, а также все типы пользователей. Внедрение становится возможным, если злоумышленник может отправить вредоносные данные интерпретатору.	Внедрение особенно распространены в старом коде. Уязвимости часто обнаруживаются в запросахSQL, LDAP, XPath или NoSQL системных командах, обработчиках XML, заголовках SMTP, языках выражений и запросахORM. Быстро и легко обнаружить и фаззеры могут помочь злоумышленникам найти такие уязвимости	Реализация может привести к потере данных, их подтверждению или разглашению третьим лицам, а также отказу в обслуживании. В некоторых случаях управление узлом может быть полностью перехвачено. Последствия для бизнеса зависят от критичности приложения и данных
A2: 2017	Злоумышленники имеют доступ к сотням тысяч действительных комбинаций имен и паролей для атак на учетные записи, спискам стандартных учетных данных администраторов, инструментам для автоматизации атак методом подбора и атак по словарям. Атаки на сессии хорошо изучены, особенно в части действующих токенов сессий.	Недостатки аутентификации очень распространены из-за исполнения и реализации большинства средств идентификации и контроля доступа. Управление сессиями является основой аутентификации и контроля доступа и присутствует во всех приложениях с контролем состояния. Атакующие могут обнаружить недостатки вручную и эксплуатировать их, используя автоматизированные инструменты, списки паролей и атаки по словарю.	Для компрометации системы злоумышленнику достаточно получить доступ к нескольким обычным или одной администраторской учетной записи. В зависимости от области исп. приложения результатом может стать отмывание денег, мошенничество в сфере социального обеспечения или кража персональных данных, а также разглашение охраняемой законом, конфиденциальной информации.

Продолжение таблицы 2

Уязвимость	Векторы атак	Недостатки безопасности	Последствия
A3: 2017	<p>Большинство веб-приложений и API имеют плохую защиту критичных финансовых, медицинских или персональных данных. Злоумышленники могут похитить или изменить эти данные, а затем осуществить мошеннические действия с кредитными картами или персональными данными. Конфиденциальные данные требуют дополнительных мер защиты, например их шифрования при хранении или передаче, а также специальных мер предосторожности при работе с браузером.</p>	<p>На протяжении последних лет данная атака является самой распространенной и опасной. Чаще всего встречается отсутствие шифрования конфиденциальных данных, а при наличии часто используются ненадежные алгоритмы, протоколы, шифры, методы хранения хешированных паролей или методы создания и управления ключами. Также легко обнаружить уязвимости на стороне сервера для передаваемых данных, но не хранимых</p>	<p>Из-за уязвимости часто страдают все персональные данные (медицинские записи, учетные данные, данные кредитных карт), которые должны быть защищены по закону, например, в соответствии с Общим регламентом ЕС по защите данных (GDPR) или локальными законами о неприкосновенности данных</p>
A4: 2017	<p>Злоумышленники могут эксплуатировать уязвимые обработчики XML через загрузку XML или внедрение вредоносного контента в XML-документы, используя уязвимый код, в зависимости или компоненты</p>	<p>Большинство старых обработчиков XML позволяют задавать внешние сущности, URI, которые разменовываются и вычисляются при обработке XML. Инструменты SAST позволяют обнаружить уязвимость путем проверки зависимостей и конфигурации. Инструменты DAST требуют дополнительных операций, выполняемых вручную, для обнаружения</p>	<p>Подобные уязвимости могут использоваться для получения данных, выполнения удаленных запросов с сервера, сканирования внутренней системы, провоцирования отказа в обслуживании, а также осуществления других атак.</p>

Продолжение таблицы 2

Уязвимость	Векторы атак	Недостатки безопасности	Последствия
A4: 2017		и эксплуатации уязвимости. Тестирующим, выполняющим проверки вручную, необходимо обучать ХХЕ тестированию, поскольку подобные проверки не проводятся.	
A5: 2017	Эксплуатация контроля доступа является основным навыком злоумышленников. Инструменты SAST и DAST могут обнаружить отсутствие контроля доступа, но не могут проверить его работоспособность при его наличии. Наличие контроля доступа можно обнаружить вручную, а его отсутствие можно обнаружить автоматически в некоторых фреймворках.	Уязвимости, связанные с контролем доступа, довольно распространены из-за отсутствия автоматического обнаружения и эффективного функционального тестирования разработчиками. Контроль доступа обычно не проверяется автоматическими статическими или динамическими тестами. Тестирование вручную — наилучший способ обнаружения отсутствия или неэффективности контроля доступа, включая методы HTTP.	Технические последствия: выполнение злоумышленником действий с правами пользователя или администратора; использование пользователем привилегированных функций; создание, просмотр, обновление или удаление любых записей.
A6: 2017	Злоумышленники часто пытаются эксплуатировать неисправленные уязвимости, настроенные по умолчанию учетные записи, незащищенные файлы для получения НСД или информации о системе.	Настройка безопасности может быть выполнена некорректно на любом уровне приложения, включая сетевые службы, платформы, веб-службы, сервер, базу данных, фреймворки, код, а также предустановленные виртуальные машины, контейнеры или хранилища. Для	Подобные уязвимости позволяют злоумышленникам получить несанкционированный доступ к системным данным или функциям, а также могут привести к полной компрометации системы

Продолжение таблицы 2

Уязвимость	Векторы атак	Недостатки безопасности	Последствия
A6 2017		поиска уязвимых настроек, настроенных по умолчанию учетных записей, неиспользуемых служб, устаревших параметров и т. п. можно использовать автоматизированные сканеры.	
A7: 2017	Автоматизированные инструменты могут обнаруживать и эксплуатировать все три вида межсайтового выполнения сценариев, более того, фреймворки для их эксплуатации можно найти в открытом доступе	Межсайтовое выполнение сценариев (XSS) является второй по распространенности уязвимостью из Топ-10 OWASP и обнаруживается в двух третях всех приложений. Автоматизированные инструменты могут обнаруживать XSS автоматически, особенно в случае проработанных технологий, таких как PHP, J2EE / JSP и ASP.NET.	Межсайтовое выполнение сценариев будет иметь последствия средней степени тяжести в случае отраженного XSS или XSS на основе объектной модели документа и серьезные последствия в случае межсайтового выполнения хранимых сценариев с удаленным выполнением кода в браузере пользователя, например, кража учетных данных, перехват сессий или установка вредоносного ПО.
A8: 2017	Эксплуатировать десериализацию сложно, поскольку готовые эксплойты редко можно использовать без их изменения или доработки	Данная уязвимость включена в Топ-10 по результатам отраслевых исследований, а не по количественным показателям. Некоторые инструменты могут обнаруживать ошибки десериализации, но для их подтверждения обычно	Последствия ошибок десериализации нельзя недооценивать. Подобные ошибки могут привести к удаленному выполнению кода, одной из самых опасных уязвимостей.

Продолжение таблицы 2

A9: 2017	Несмотря на простоту поиска уже готовых эксплойтов для большинства известных уязвимостей, некоторые из них требуют создания специальных средств для их эксплуатации.	Данная уязвимость является очень распространенной. Шаблоны для разработчиков, содержащие большое количество компонентов, могут привести к непониманию того, какие компоненты реально используются в приложении или API. Некоторые сканеры, такие как retire.js, могут помочь с обнаружением уязвимостей, но определение сложности их эксплуатации потребует дополнительных усилий	Несмотря на то, что не все уязвимости приводят к серьезным последствиям, причиной некоторых масштабных взломов стали именно компоненты, содержащие известные уязвимости. В зависимости от защищаемых активов подобная угроза может оказаться на вершине вашего списка.
A10: 2017	Эксплуатация недостатков журналирования и мониторинга лежит в основе почти всех крупных взломов. При проведении атак злоумышленники полагаются на отсутствие контроля и своевременного реагирования на инциденты	Одним из способов определить качество мониторинга является анализ журналов после проведения теста на проникновение. Для определения возможного ущерба все действия тестируемых должны регистрироваться.	Большинство атак начинаются с анализа уязвимостей. Возможность проведения подобного анализа повышает вероятность удачной эксплуатации уязвимости

3 Конфигурирование виртуального окружения для разработки, размещения и эксплуатации программ, содержащих уязвимости

Для конфигурирования виртуальной среды были использованы средства виртуализации Oracle VM VirtualBox. Создана виртуальная машина с названием server для операционной системы Linux (Debian) с оперативной памятью 1024 Мб, это показано на рисунке 2.

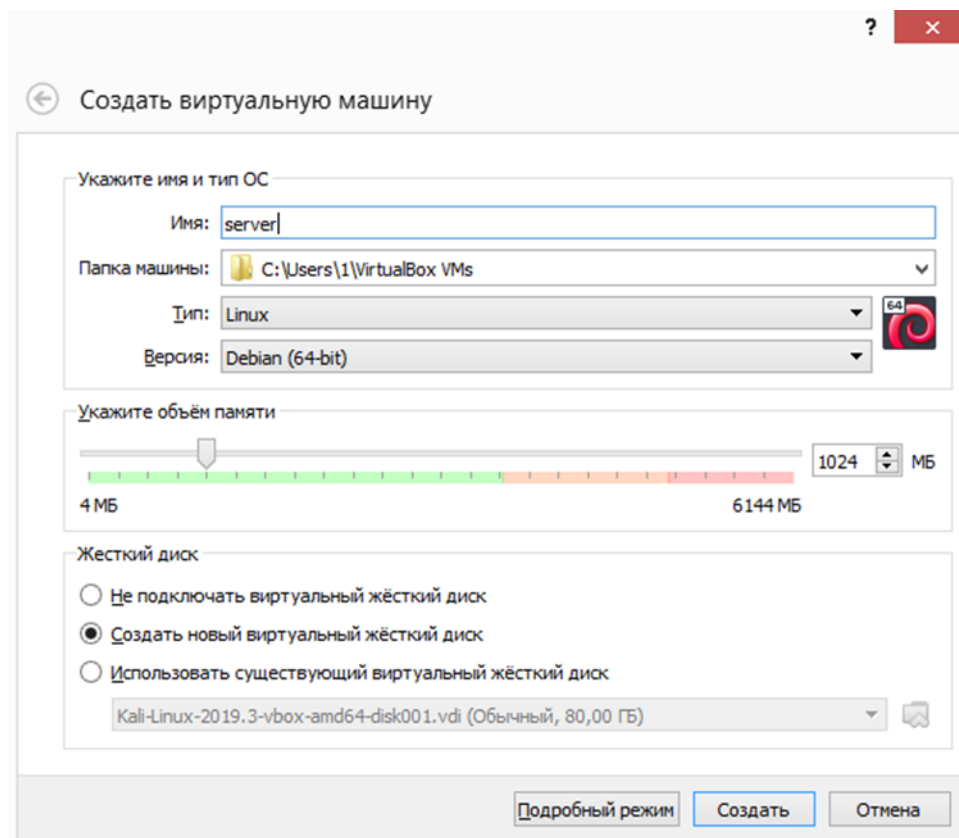


Рисунок 2 – Создание виртуальной машины

Следующим этапом был подключен жесткий диск на 8 Гб и смонтирован образ операционной системы Linux(Debian) для ее установки, который был скачен с официального сайта разработчика. Следует выполнить ряд действий для настройки системы, таких как выбор языка, часового пояса, пароля для суперпользователя, имя пользователя и пароль для входа был задан serv, так же было распределено пространство жесткого диска, куда будет установлена операционная система, это показано на рисунке 3.

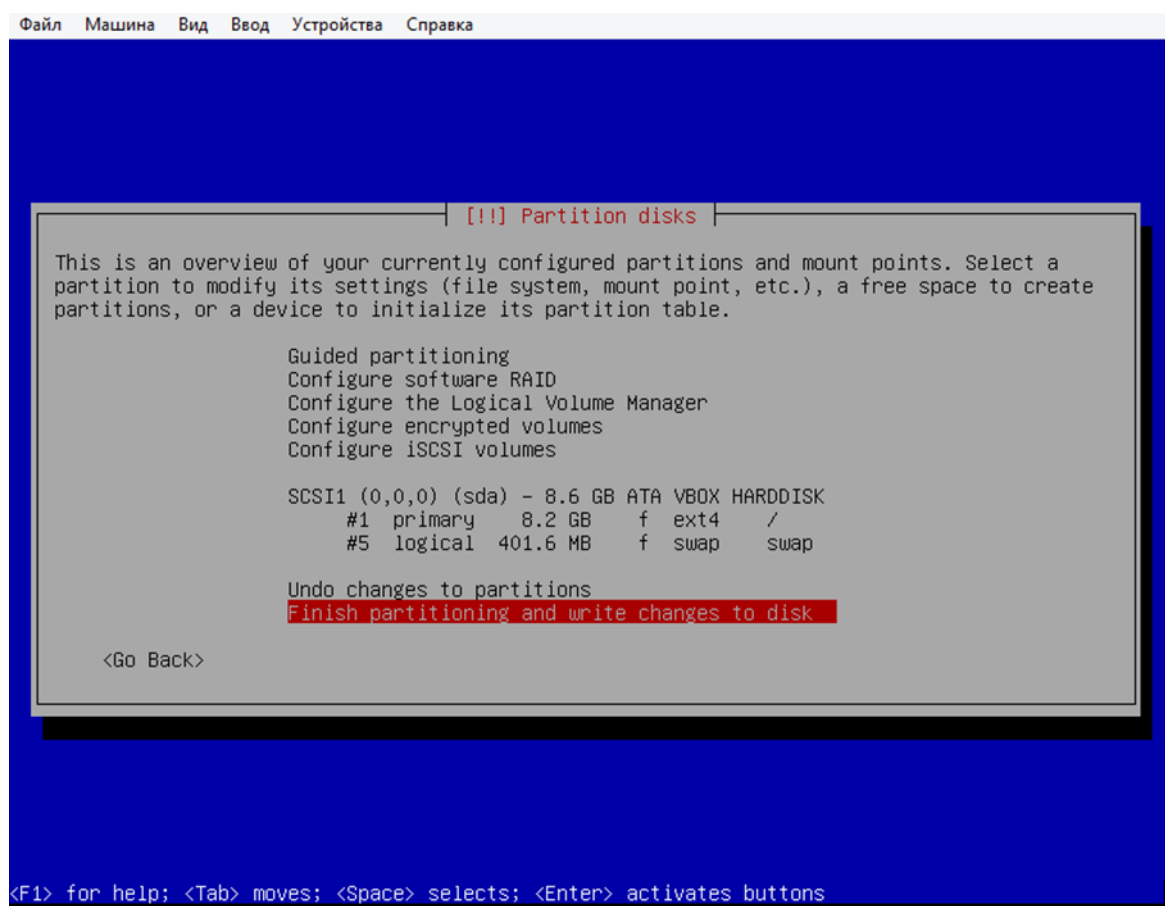


Рисунок 3 – Распределение пространства жесткого диска

По окончании установки системы, необходимо выполнить перезагрузку виртуальной машины, после этого ввести логин и пароль заданные в начале установки, в нашем случае логин и пароль serv.

После входа в систему необходимо проверить наличие обновлений, для этого была использована команда в терминале `sudo apt-get update` и `sudo apt-get upgrade`. Когда все необходимые файлы будут скачены и установлены, система будет готова к работе. Для нашего веб-сервера, который будет работать внутрилокальной сети необходимо поменять настройки сетевого адаптера в конфигурации виртуальной машины на сетевой мост. Вход в систему serv и настройка сетевого адаптера показана на рисунках 4 и 5.

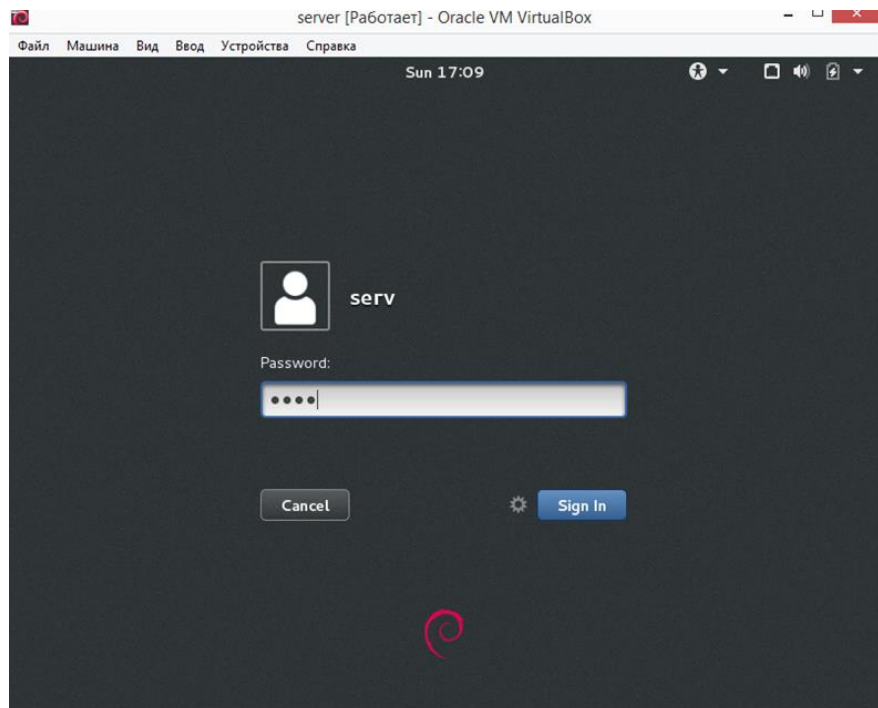


Рисунок 4 - Вход в учетную запись serv

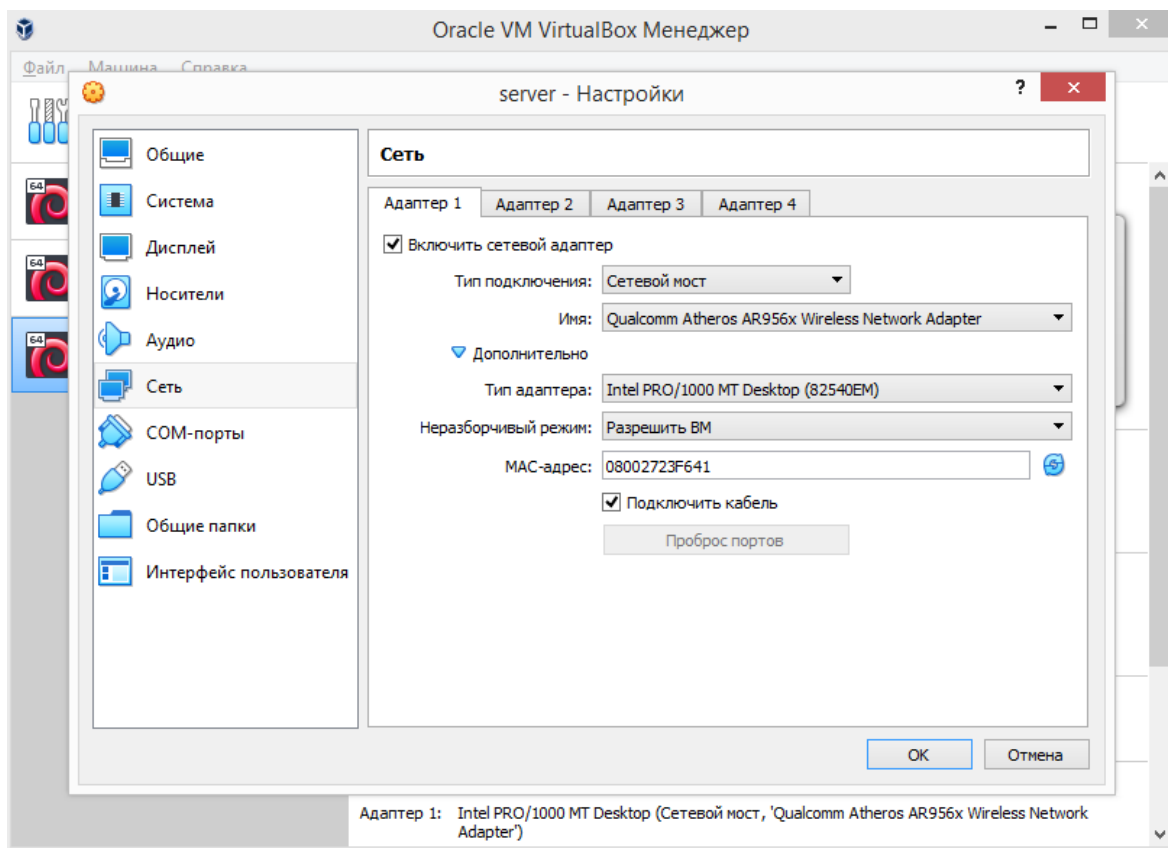
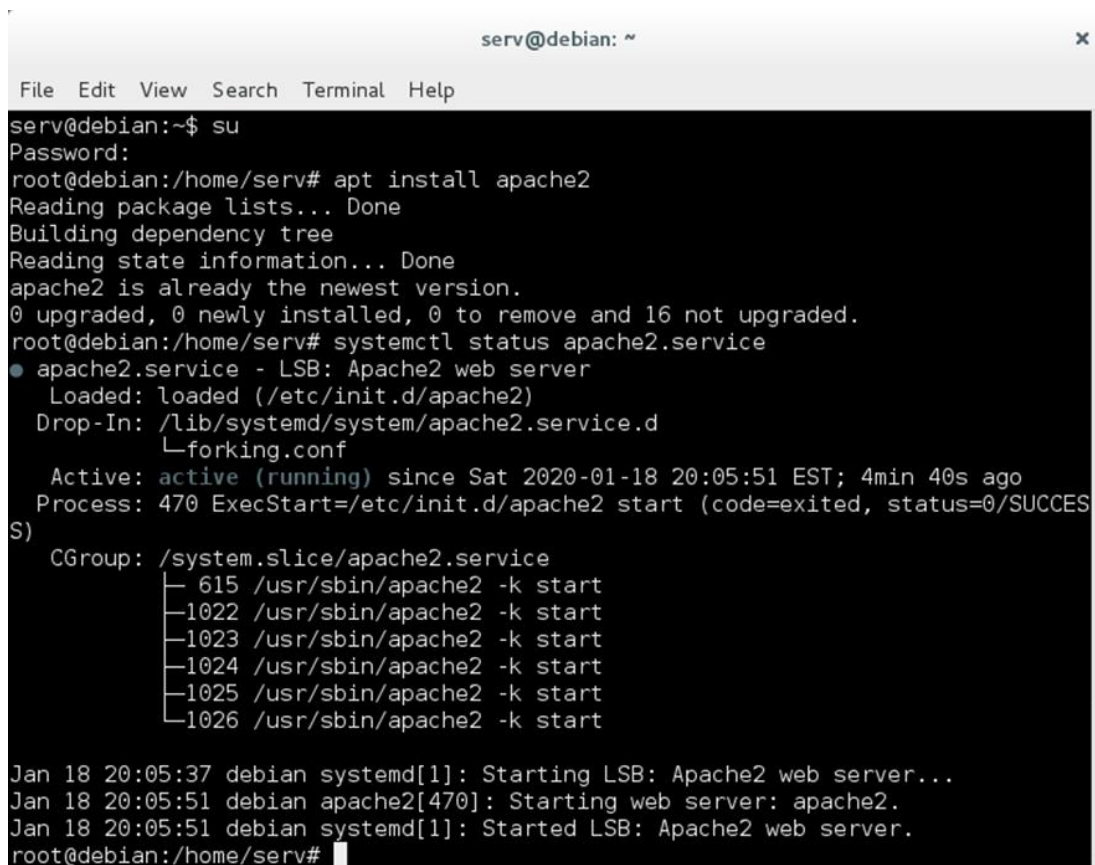


Рисунок 5 - Настройки сетевого адаптера виртуальной машины server

Для тестирования безопасности сервера необходимо аналогично установить еще одну виртуальную машину с операционной системой Kali

Linux. На этом конфигурирование виртуальной среды для размещения веб-сервера завершено.

Первое включение операционной системы показало, что нужно установить дополнительные инструменты для работы веб-сервера. Для начала был установлен веб-сервер Apache2 из официальных репозитория Debian и проверена его работоспособность, это показано на рисунке 6.



```
serv@debian: ~  
File Edit View Search Terminal Help  
serv@debian:~$ su  
Password:  
root@debian:/home/serv# apt install apache2  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
apache2 is already the newest version.  
0 upgraded, 0 newly installed, 0 to remove and 16 not upgraded.  
root@debian:/home/serv# systemctl status apache2.service  
● apache2.service - LSB: Apache2 web server  
   Loaded: loaded (/etc/init.d/apache2)  
   Drop-In: /lib/systemd/system/apache2.service.d  
            └─forking.conf  
   Active: active (running) since Sat 2020-01-18 20:05:51 EST; 4min 40s ago  
   Process: 470 ExecStart=/etc/init.d/apache2 start (code=exited, status=0/SUCCESS)  
   CGroup: /system.slice/apache2.service  
           └─ 615 /usr/sbin/apache2 -k start  
              1022 /usr/sbin/apache2 -k start  
              1023 /usr/sbin/apache2 -k start  
              1024 /usr/sbin/apache2 -k start  
              1025 /usr/sbin/apache2 -k start  
              1026 /usr/sbin/apache2 -k start  
Jan 18 20:05:37 debian systemd[1]: Starting LSB: Apache2 web server...  
Jan 18 20:05:51 debian apache2[470]: Starting web server: apache2.  
Jan 18 20:05:51 debian systemd[1]: Started LSB: Apache2 web server.  
root@debian:/home/serv#
```

Рисунок 6 – Установка и проверка Apache2.

Следующим этапом был настроен файл конфигурации веб-сервера, который находится по пути:

`/etc/apache2/apache2.conf`

Так как сервер является локальным, то была изменена глобальная переменная `ServerName` на значение `localhost` и выполнена перезагрузка Apache, для того чтобы изменения вступили в силу.

Для проверки работы веб-сервера Apache, был открыт браузер FireFox и набран адрес `localhost`, результат работы показан на рисунке 7.

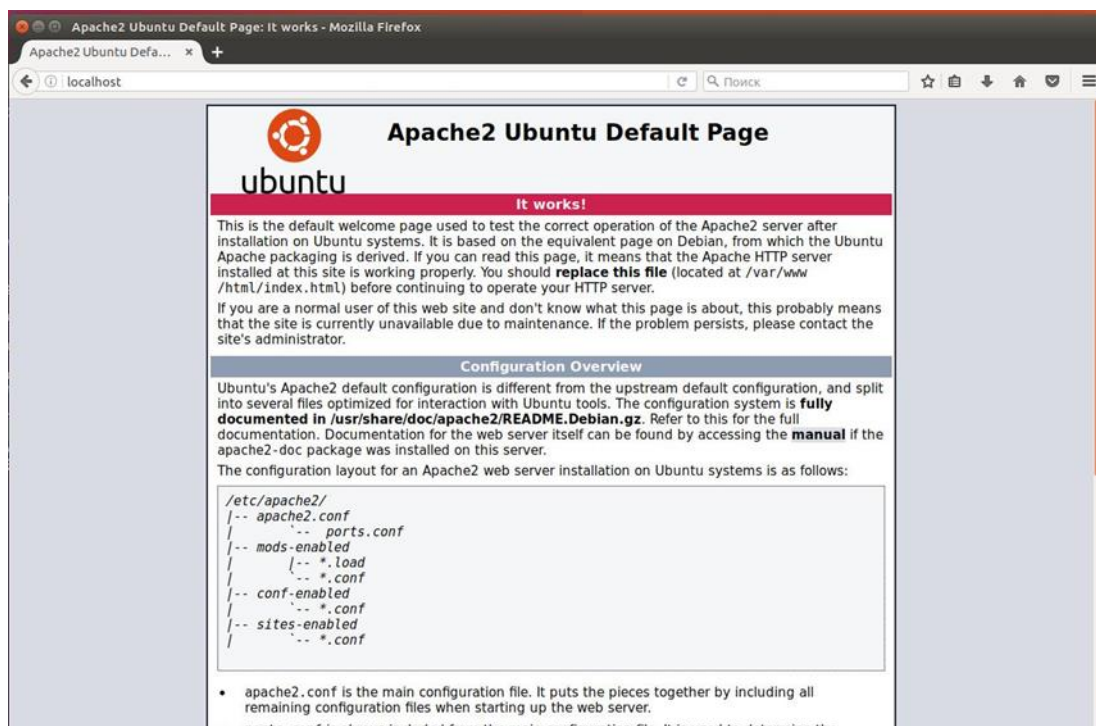


Рисунок 7 – Результат работы сервера Apache

После установки и настройки веб-сервера Apache нужно установить систему управления базами данных MySQL, для этого введем команду

```
apt install mysql-server
```

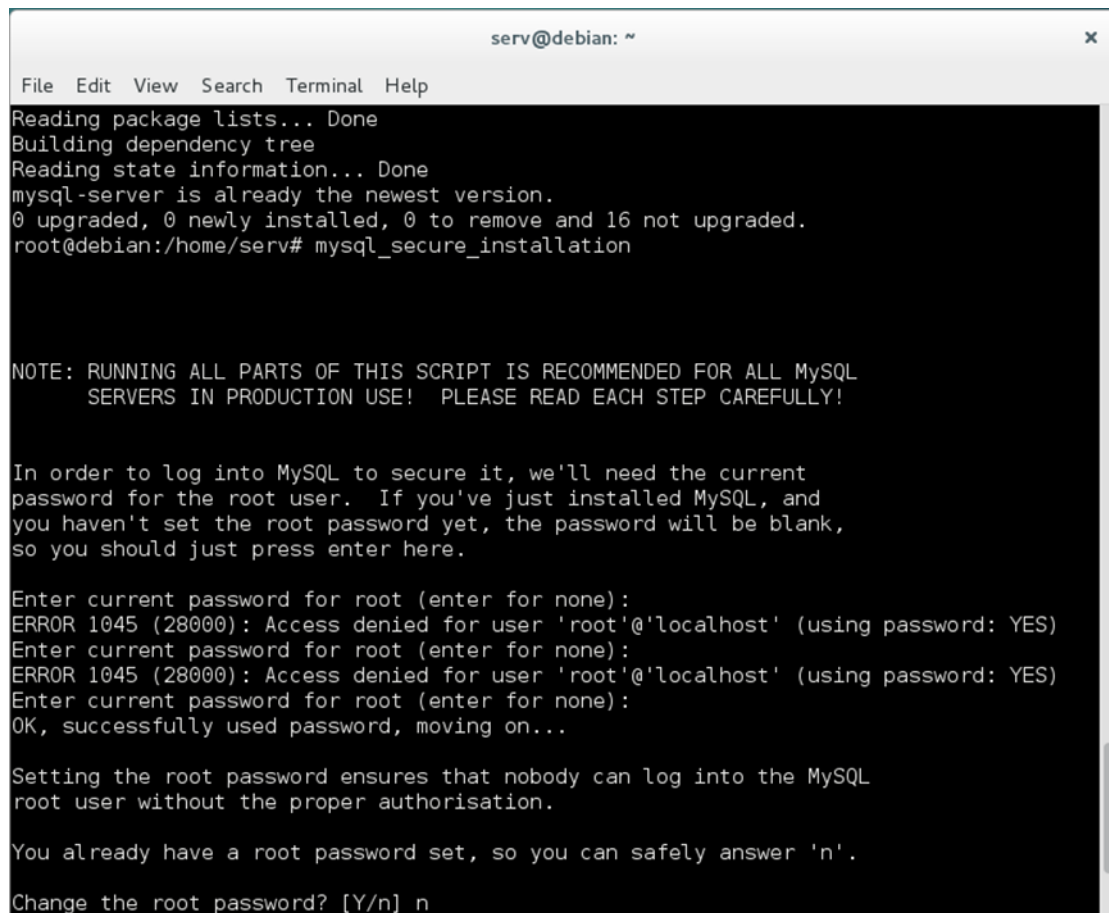
и дождемся конца установки.

После завершения установки, был запущен простой скрипт безопасности, который удалит некоторые настройки по умолчанию и немного заблокирует доступ к нашей системе. Интерактивный скрипт запускается командой `mysql_secure_installation`.

Первое, что требует от нас скрипт это ввести пароль администратора от СУБД MySQL. После ввода пароля будем предложено его изменить на более сложный.

По умолчанию система имеет анонимного пользователя, позволяющего любому войти в СУБД MySQL даже если для него не было создано пользовательского аккаунта. Это сделано в целях тестирования и упрощения установки. Его следует удалить перед переходом в реальное рабочее окружение. Следующим пунктом будет вопрос о разрешении администратору подключаться только с localhost. Это гарантирует, что кто-то из сети не сможет

угадать пароль root'a. Для тестирования нашего сервера оставим эту возможность включен. На рисунках 8 и 9 показаны создание пароля для администратора и настройка прав доступа



```
serv@debian: ~
File Edit View Search Terminal Help
Reading package lists... Done
Building dependency tree
Reading state information... Done
mysql-server is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 16 not upgraded.
root@debian:/home/serv# mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MySQL
SERVERS IN PRODUCTION USE! PLEASE READ EACH STEP CAREFULLY!

In order to log into MySQL to secure it, we'll need the current
password for the root user. If you've just installed MySQL, and
you haven't set the root password yet, the password will be blank,
so you should just press enter here.

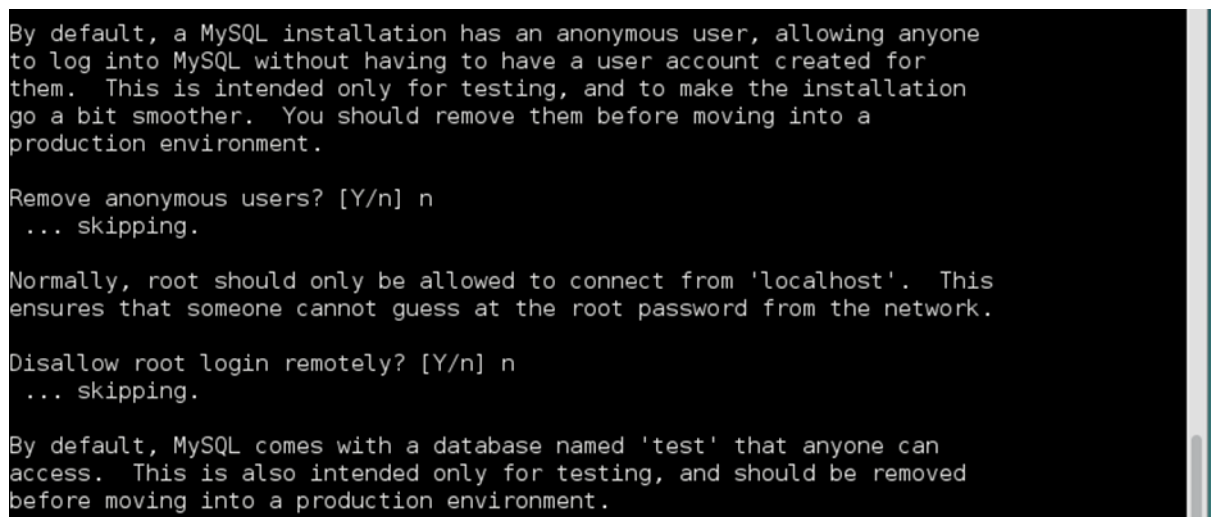
Enter current password for root (enter for none):
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
Enter current password for root (enter for none):
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password ensures that nobody can log into the MySQL
root user without the proper authorisation.

You already have a root password set, so you can safely answer 'n'.

Change the root password? [Y/n] n
```

Рисунок 8 – Настройка безопасности MySQL



```
By default, a MySQL installation has an anonymous user, allowing anyone
to log into MySQL without having to have a user account created for
them. This is intended only for testing, and to make the installation
go a bit smoother. You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] n
... skipping.

Normally, root should only be allowed to connect from 'localhost'. This
ensures that someone cannot guess at the root password from the network.

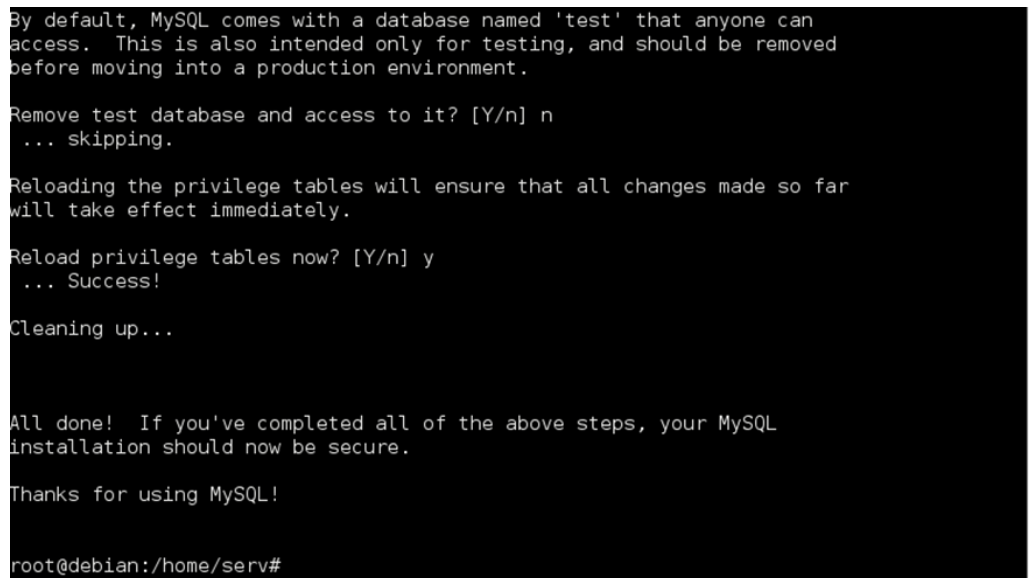
Disallow root login remotely? [Y/n] n
... skipping.

By default, MySQL comes with a database named 'test' that anyone can
access. This is also intended only for testing, and should be removed
before moving into a production environment.
```

Рисунок 9 – Настройка безопасности MySQL

По умолчанию СУБД MySQL поставляется с базой данных 'test', к которой может любой получить доступ. Это также сделано в целях

тестирования и она должна быть удалена перед переходом в реальное рабочее окружение. Последним пунктом будет предложено перезагрузить таблицу привилегий, чтобы изменения немедленно вступили в силу, это показано на рисунке 10.



```
By default, MySQL comes with a database named 'test' that anyone can
access. This is also intended only for testing, and should be removed
before moving into a production environment.

Remove test database and access to it? [Y/n] n
... skipping.

Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.

Reload privilege tables now? [Y/n] y
... Success!

Cleaning up...

All done! If you've completed all of the above steps, your MySQL
installation should now be secure.

Thanks for using MySQL!

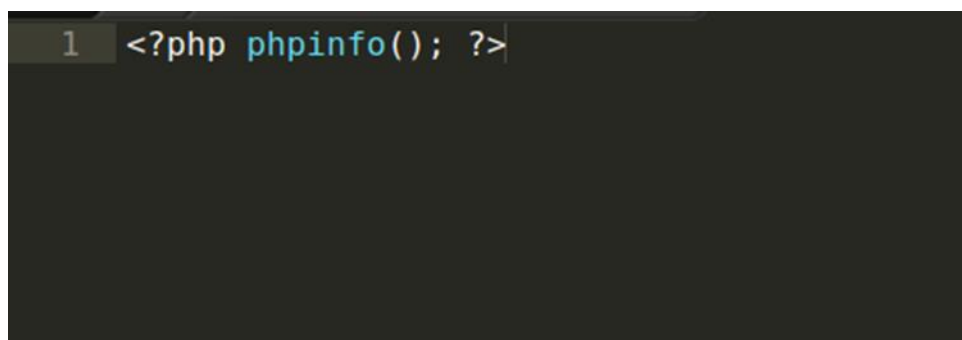
root@debian:/home/serv#
```

Рисунок 10 – Перезагрузка таблицы привилегий MySQL

После установки и настройки СУБД MySQL необходимо установить интерпретатор языка PHP. Для этого в терминале от имени администратора следует ввести команду:

```
apt-get install libapache2-mod-php5 php5-mysql php5-curl php5-gd php-mail –
у
```

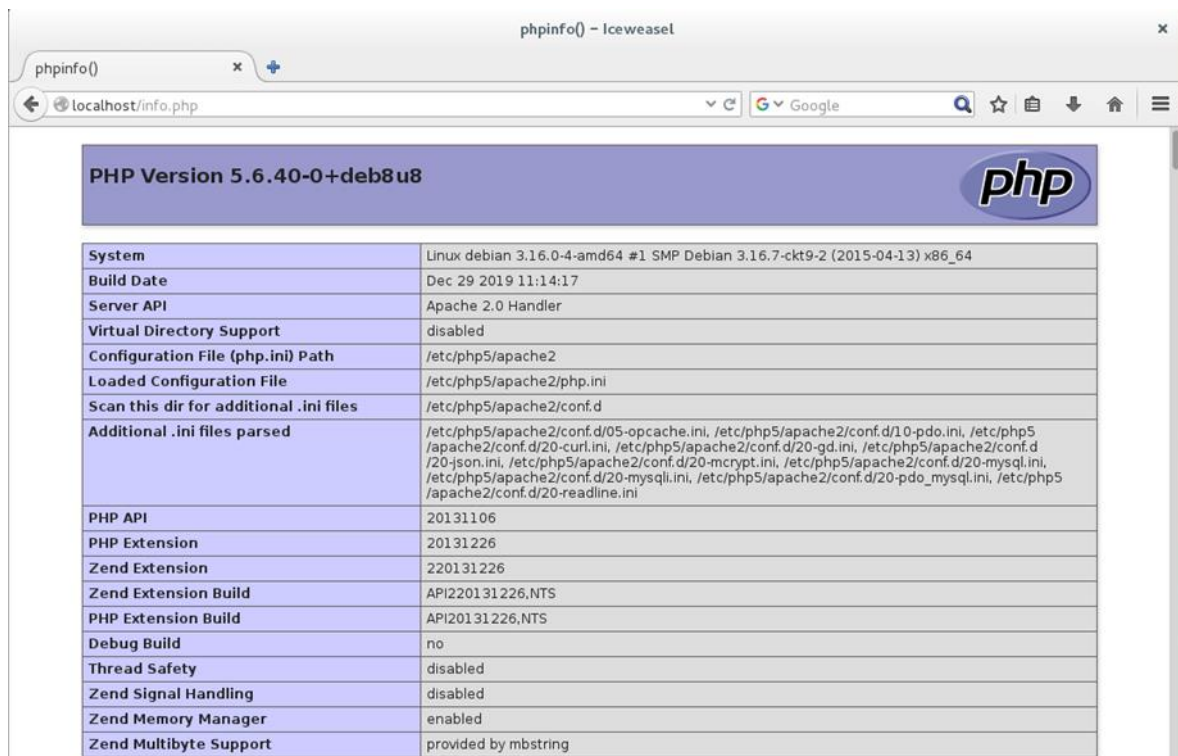
После установки интерпретатора языка PHP, необходимо проверить его работоспособность, для этого был создан пустой файл в корневой директории сервера /var/www/html/info.php, в который был добавлен php код показанный на рисунке 11.



```
1 <?php phpinfo(); ?>
```

Рисунок 11 – Код на языке PHP

Данный код показывает работоспособность интерпретатора PHP. Для его запуска необходимо открыть браузер и ввести в адресной строке путь к созданному файлу, в нашем случае это localhost/info.php, работоспособность показана на рисунке 12.



PHP Version 5.6.40-0+deb8u8	
System	Linux debian 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt9-2 (2015-04-13) x86_64
Build Date	Dec 29 2019 11:14:17
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/05-opcache.ini, /etc/php5/apache2/conf.d/10-pdo.ini, /etc/php5/apache2/conf.d/20-curl.ini, /etc/php5/apache2/conf.d/20-gd.ini, /etc/php5/apache2/conf.d/20-json.ini, /etc/php5/apache2/conf.d/20-mcrypt.ini, /etc/php5/apache2/conf.d/20-mysql.ini, /etc/php5/apache2/conf.d/20-mysqli.ini, /etc/php5/apache2/conf.d/20-pdo_mysql.ini, /etc/php5/apache2/conf.d/20-readline.ini
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API220131226,NTS
PHP Extension Build	API20131226,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring

Рисунок 12 – Информация о PHP на сервере

Так же опционально для администрирования из веб-браузера был установлен phpMyAdmin следующей командой : `apt install php-gettext phpmyadmin`. Во-время установки phpMyAdmin необходимо выбрать веб-сервер Apache.

От настройки базы данных с помощью `dbconfig-common` отказываемся, т.к. база данных у нас уже установлена..

Для загрузки конфигурации phpMyAdmin веб-сервером Apache необходимо открыть файл `/etc/apache2/apache2.conf` и добавить в самый конец строку `Include /etc/phpmyadmin/apache.conf` . После данной операции нужно сохранить файл и перезапустить сервер командой `systemctl restart apache2`.

Теперь phpMyAdmin будет доступен по адресу `localhost/phpmyadmin`
 Настройка сервера phpMyAdmin показана на рисунках 13-15

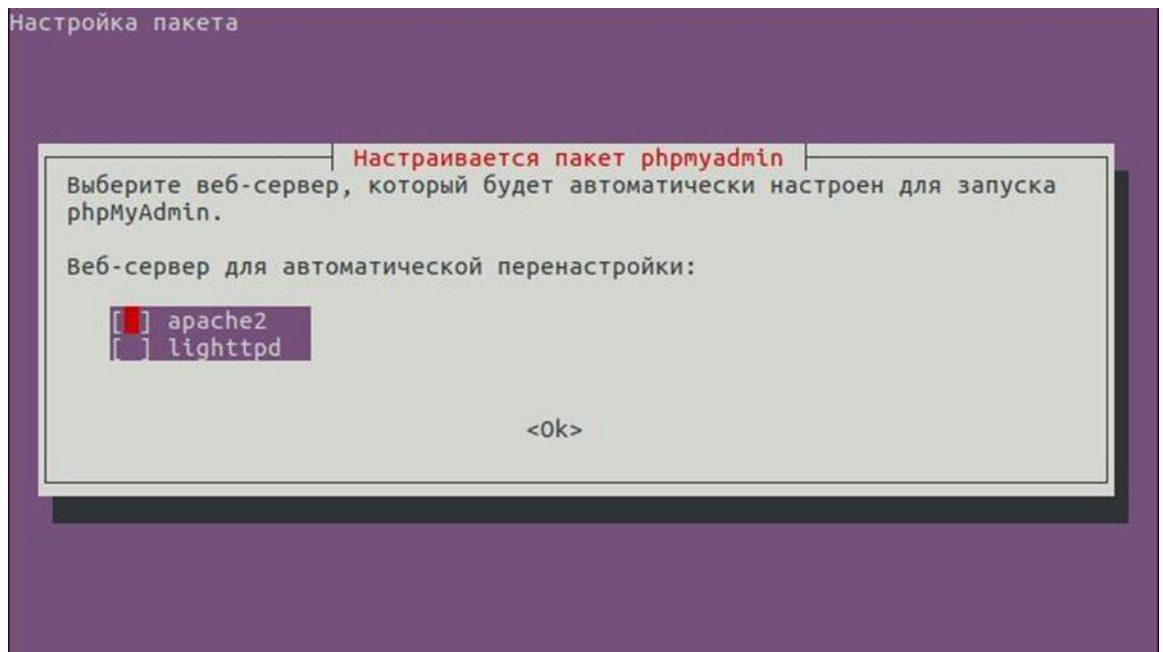


Рисунок 13 – Настройка пакета phpMyAdmin

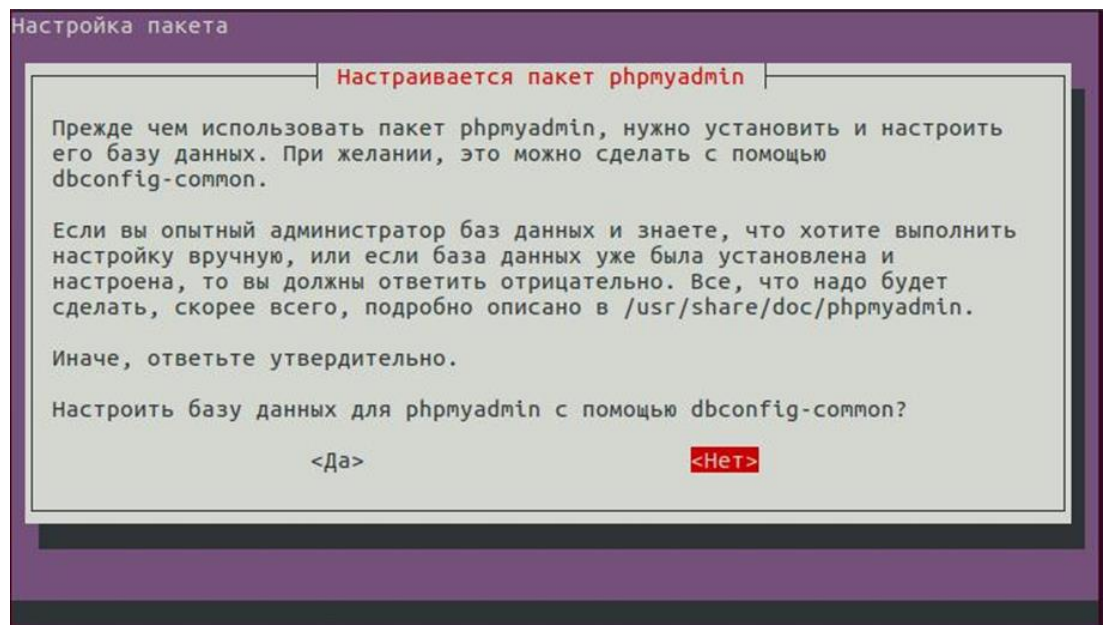


Рисунок 14 – Настройка пакета phpMyAdmin

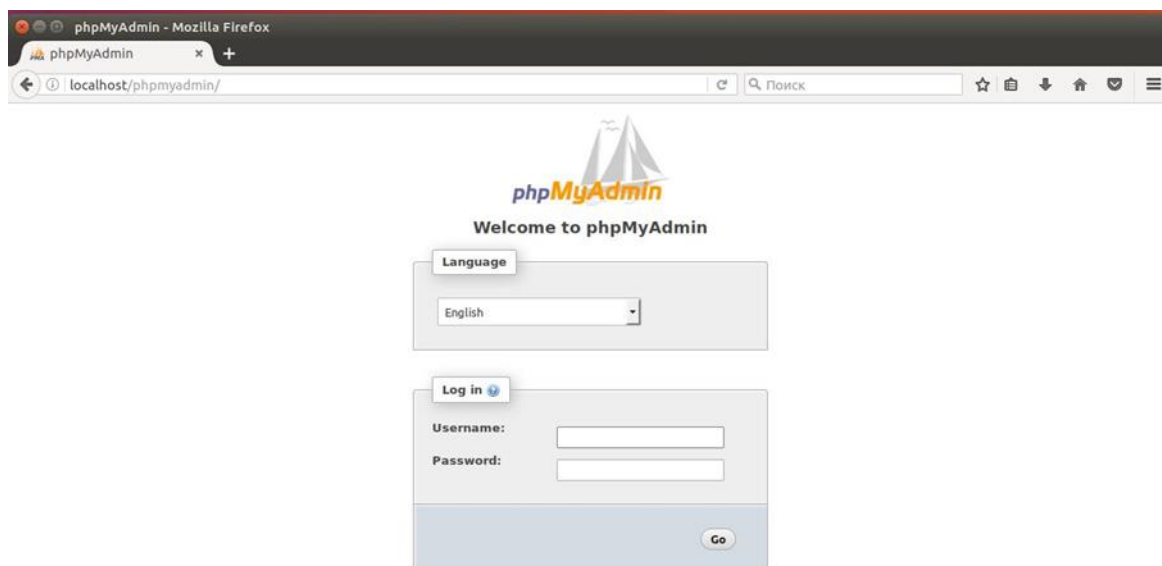


Рисунок 15 – Панель управления phpMyAdmin

Для исследования бинарных уязвимостей было сконфигурировано виртуальное окружение для размещения и эксплуатации бинарных уязвимостей и способов защиты от них. Виртуальное окружение для развертывания лабораторного стенда включает в себя следующее программно-аппаратное обеспечение: компьютер со средствами виртуализации Oracle VM Virtual Box, в которой была создана виртуальная машина с установленной операционной системой Linux(Debian), которая имеет название Debian 8 и оперативную память 512 Мб. Внешний вид главного экрана Oracle VM Virtual Box после завершения подготовки новой ВМ.

Затем был подключен жесткий диск на 2 Гб и смонтирован образ операционной системы Linux(Debian) для ее установки, который был скачен с официального сайта разработчика. Следует выполнить ряд действий для настройки системы, таких как выбор языка, часового пояса, имени и пароля пользователя для входа. Так же было распределено пространство жесткого диска, куда будет установлена операционная система..

Создание и настройка виртуальной машины показана на рисунках 16-17

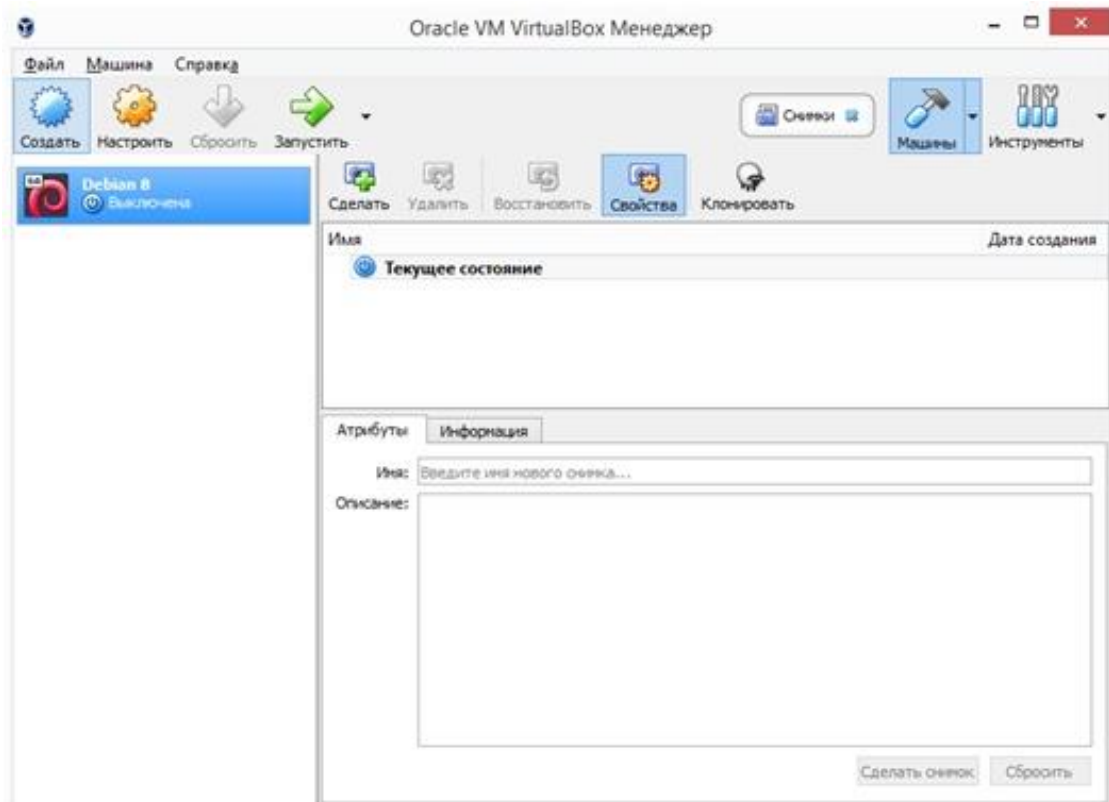


Рисунок 16 – Внешний вид средства виртуализации после создания VM

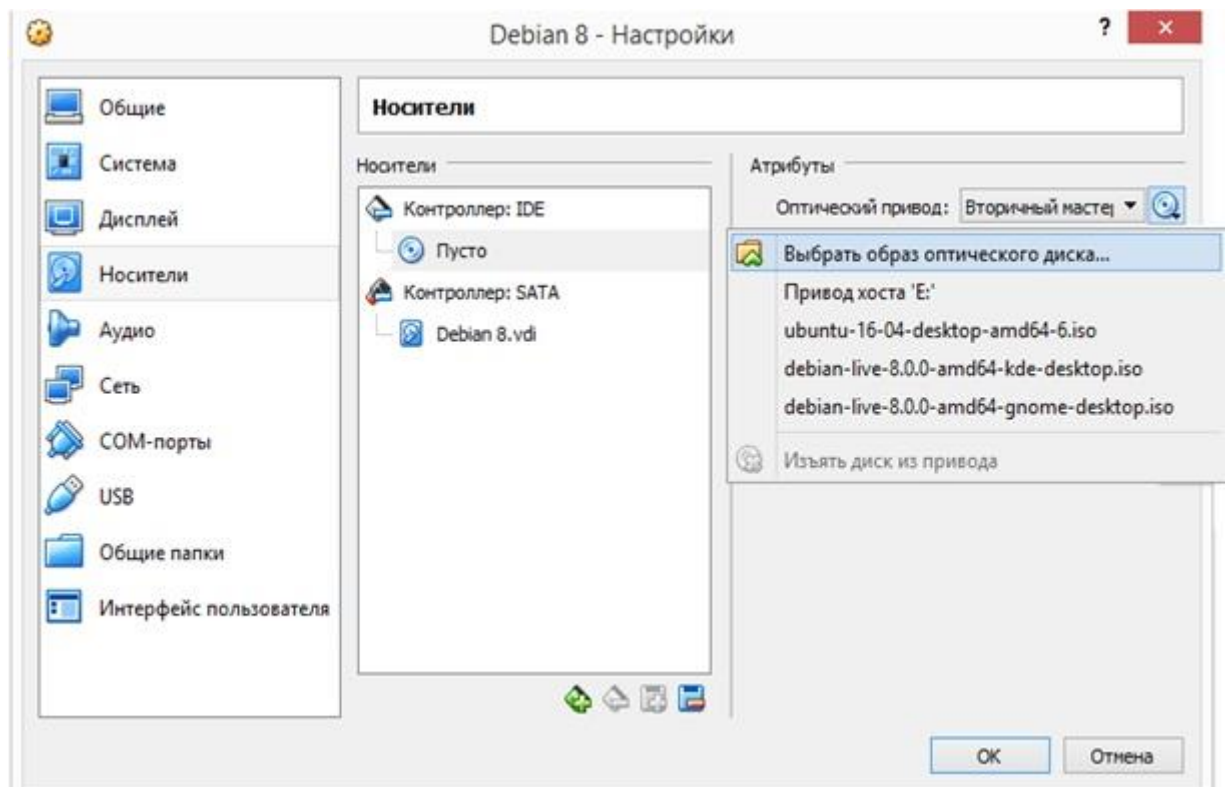
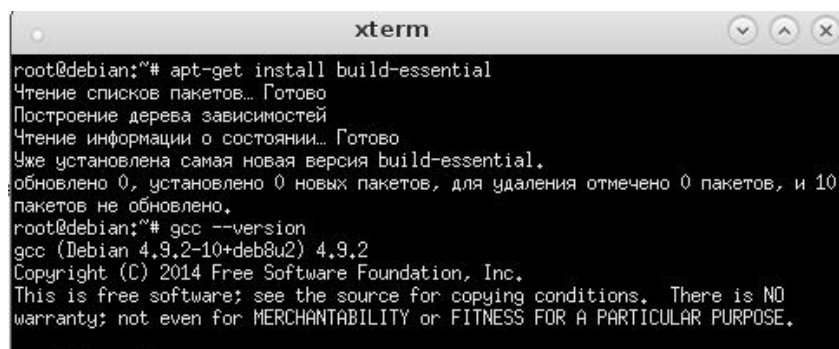


Рисунок 17 – Настройки VM

После окончания установки системы, необходимо дождаться перезагрузки виртуальной машины, после этого ввести логин и пароль,

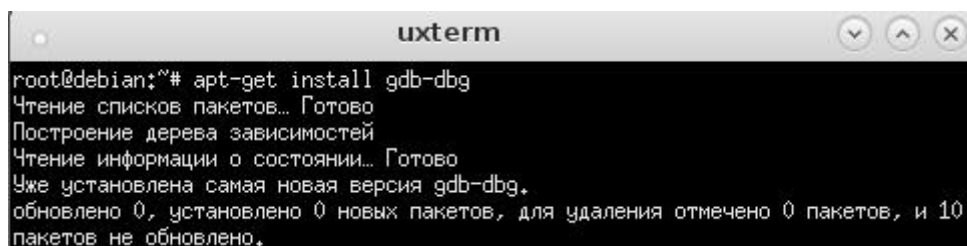
заданные вначале установки и войти в систему. Следующим действием был установлен компилятор GCC (GNU C compiler) с помощью команды `apt install build-essential`, который является одним из наиболее простых и гибких в управлении компиляторов. После чего была произведена проверка установки компилятора. Это показано на рисунке 18.



```
root@debian:~# apt-get install build-essential
Чтение списков пакетов... Готово
Построение дерева зависимостей
Чтение информации о состоянии... Готово
Уже установлена самая новая версия build-essential.
обновлено 0, установлено 0 новых пакетов, для удаления отмечено 0 пакетов, и 10
пакетов не обновлено.
root@debian:~# gcc --version
gcc (Debian 4.9.2-10+deb8u2) 4.9.2
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Рисунок 18 – Установка компилятора GCC

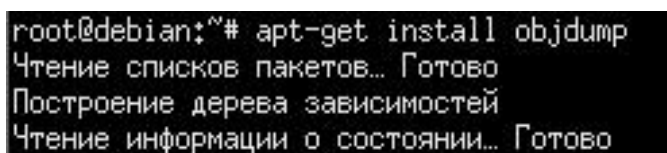
Следующим этапом был установлен отладчик `gdb-dbg` при помощи команды `apt-get install gdb-dbg`. Это представлено на рисунке 19.



```
root@debian:~# apt-get install gdb-dbg
Чтение списков пакетов... Готово
Построение дерева зависимостей
Чтение информации о состоянии... Готово
Уже установлена самая новая версия gdb-dbg.
обновлено 0, установлено 0 новых пакетов, для удаления отмечено 0 пакетов, и 10
пакетов не обновлено.
```

Рисунок 19 – Установка отладчика gdb-dbg

Следующим этапом была установлена утилита `objdump`. Это было сделано для отображения информации объектных файлов запускаемой программы. Это показано на рисунке 20.



```
root@debian:~# apt-get install objdump
Чтение списков пакетов... Готово
Построение дерева зависимостей
Чтение информации о состоянии... Готово
```

Рисунок 20 – Установка утилиты objdump

Следующим этапом была проведена проверка утилиты `objdump` на тестовом файле. Результаты представлены на рисунке 21.

```

root@debian:~# readelf a.out -a
Заголовок ELF:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00
  Класс:                               ELF64
  Данные:                               дополнение до 2, little endian
  Версия:                               1 (current)
  OS/ABI:                               UNIX - System V
  Версия ABI:                           0
  Тип:                                  EXEC (Исполняемый файл)
  Машина:                               Advanced Micro Devices X86-64
  Версия:                               0x1
  Адрес точки входа:                   0x400410
  Начало заголовков программы:          64 (байт в файле)
  Начало заголовков программы:          4776 (байт в файле)
  Флаги:                                0x0
  Размер этого заголовка:                64 (байт)
  Размер заголовков программы:          56 (байт)
  Число заголовков программы:            8
  Размер заголовков раздела:             64 (байт)
  Число заголовков раздела:              30
  Индекс табл. строк загол. раздела: 27

```

Рисунок 21 – Результат работы утилиты objdump

4 Разработка программных средств содержащего уязвимости

4.1 Разработка программных средств, содержащего уязвимость переполнение буфера

Создание программы на языке C, реализующей замену одной операции в исследуемой функции. В качестве примера предлагается рассмотреть подмену одного байта и произвести изменение работы самой процедуры.

```
void foo(void) {  
    int i=0;  
    i++;  
    printf("i: %d\n", i);  
}
```

Данная программа создает переменную `i` и присваивает ей значение 0, затем выводит его в `stdout`.

Для того чтобы создать программу, которая реализует замену одной операции в данной функции, необходимо посмотреть инструкции, которые генерирует компилятор для функции `foo()`. Ниже представлен полный код программы с использованием функции `foo()`:

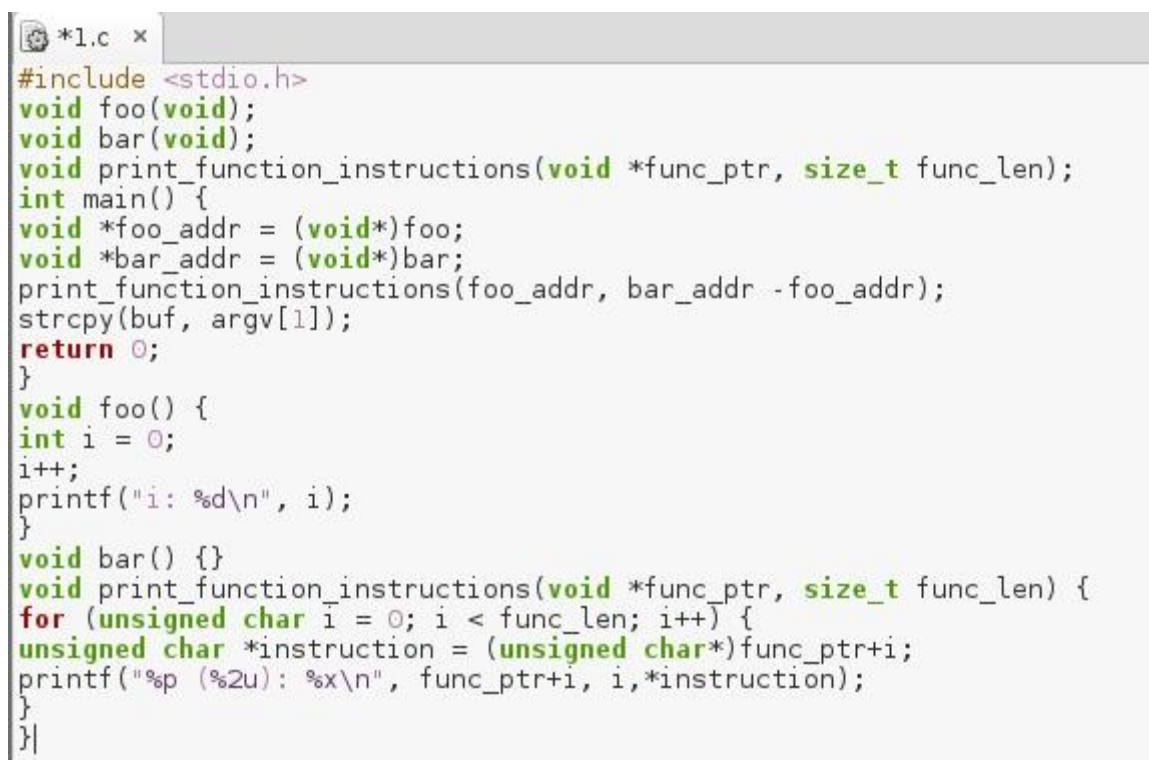
```
#include <stdio.h>  
  
void foo(void);  
  
int main(void) {  
    return 0;  
}  
  
void foo(void) {  
    int i=0;  
    i++;
```

```
printf("i: %d\n", i);
}
```

Имея указатель на функцию `foo()`, необходимо получить указатель, который имеет тип `unsigned char` на байт, который необходимо поменять.

```
unsigned char *instruction = (unsigned char*)foo_addr + 18;
*instruction = 0x2A;
```

На рисунке 22 приведен текст получившейся программы.



```
*1.c x
#include <stdio.h>
void foo(void);
void bar(void);
void print_function_instructions(void *func_ptr, size_t func_len);
int main() {
    void *foo_addr = (void*)foo;
    void *bar_addr = (void*)bar;
    print_function_instructions(foo_addr, bar_addr - foo_addr);
    strcpy(buf, argv[1]);
    return 0;
}
void foo() {
    int i = 0;
    i++;
    printf("i: %d\n", i);
}
void bar() {}
void print_function_instructions(void *func_ptr, size_t func_len) {
    for (unsigned char i = 0; i < func_len; i++) {
        unsigned char *instruction = (unsigned char*)func_ptr+i;
        printf("%p (%2u): %x\n", func_ptr+i, i,*instruction);
    }
}
```

Рисунок 22 – Код программы, имеющий уязвимость переполнение буфера

Затем необходимо скомпилировать исходный код программы с помощью команды: `$ gcc -std=c99 -D_BSD_SOURCE -o foo foo.c` и запустить.

В приведенном выше коде используется небезопасная функция `strcpy`, которая позволяет записать больше данных, чем вмещает выделенный под них массив. Если запустить данную программу в системе Linux с аргументом, длина которого превышает 100 байт, скорее всего, работа программы будет аварийно завершена, а пользователь получит сообщение об ошибке. Вывод работающей программы представлен на рисунке 23.

```
root@debian:~# gcc prog.c
root@debian:~# gcc -o prog prog.c
root@debian:~# ./prog
Ошибка сегментирования
```

Рисунок 23 – Результат работы программы

4.2 Разработка программного кода, содержащего уязвимость целочисленное переполнение

Классически целочисленное переполнение может быть эксплуатировано через переполнение буфера. Получившийся код представлен на рисунке 24

```
#include <limits.h>

void func(int si_a, int si_b) {
    int result;
    if (si_a > 0) {
        if (si_b > 0) {
            if (si_a > (INT_MAX / si_b)) {

            }
        } else {
            if (si_b < (INT_MIN / si_a)) {

            }
        }
    } else {
        if (si_b > 0) {
            if (si_a < (INT_MIN / si_b)) {

            }
        } else {
            if ( (si_a != 0) && (si_b < (INT_MAX / si_a))) {

            }
        }
    }
    result = si_a * si_b;
}
```

Рисунок 24 – Код программы, имеющий уязвимость целочисленное
переполнение

Данный пример проверяет, могут ли быть переменные сложены без переполнения. Это может произойти, так как переменные знакового типа.

4.3 Разработка программно кода, содержащего уязвимость состояние «ГОНКИ»

Программа, которая содержит уязвимость из-за состояния гонки показана на рисунке 25



```
prog1.c x
#include <stdio.h>
#include <unistd.h>
#include <cstdlib>
#define DELAY 10000
int main()
{
    char* fn = "/tmp/XYZ";
    char buffer[60]; FILE* fp;
    long int i;

    scanf("%50s", buffer );
    if(!access(fn, W_OK))
    {
        for (i=0; i < DELAY; i++)
        {
            int a = i^2;
        }
        fp = fopen(fn, "a+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp); fclose(fp);
    }
    else printf("No permission \n");
}
```

Рисунок 25 – Код программы, содержащий уязвимость состояние гонки

Выше представлена часть программы Set-UID - установка ID пользователя во время выполнения и является правами доступа в Unix, которые разрешают пользователям запускать исполняемые файлы с правами владельца исполняемого файла. Данная программа добавляет строку, которая была введена пользователем в конец временного файла /tmp/XYZ. Так как программа выполняется с root-привилегиями, она проверяет, разрешен ли реальному пользователю доступ к файлу /tmp/XYZ. Для этого программа запускает процесс `access()`. После того, как она убедится в том, что у пользователя достаточно прав, программа открывает файл и записывает введенные данные в него. Здесь есть состояние гонки: во временном окне (которое симулируется задержкой между проверкой (`access`) и использованием (`fopen`)). Есть вероятность того, что файл, используемый процессом `access`, отличается от

файла, используемого процессом foren, не смотря на то что они имеют одинаковое имя /tmp/XY. Если злоумышленник может каким-либо образом сделать файл /tmp/XYZ жесткой ссылкой, которая указывает на /etc/shadow, значит он может добавить пользовательский ввод к /etc/shadow.

5 Описание процесса эксплуатации уязвимостей программных средств

5.1 Процесс эксплуатации уязвимости переполнение буфера

Переполнение буфера стека перезаписывает данные, которые разработчик не собирался перезаписывать, обеспечивая полный контроль над программой и её выходными данным. Код программы приведен на рисунке 26.

```
1  #include <stdlib.h>
2  #include <unistd.h>
3  #include <stdio.h>
4
5  int main(int argc, char **argv)
6  {
7      volatile int modified;
8      char buffer[64];
9
10     modified = 0;
11     gets(buffer);
12
13     if(modified != 0) {
14         printf("You have changed the 'modified' variable\n");
15     } else {
16         printf("Try again?\n");
17     }
18 }
```

Рисунок 26 – Код программы

5.2 Процесс эксплуатации уязвимости целочисленное переполнение

В приведённой выше функции показано, что массив символьного типа с именем `buffer` создаётся с размером 64. Переменная `modified` равна 0, и функция `gets()` вызывается с переменной `buffer` в качестве аргумента. Оператор `IF`, проверяет, не равно ли значение `modified` нулю. Переменная `modified` имеет значение, отличное от 0. Документация функции `gets()` показана на рисунке 27.

`gets()` reads a line from `stdin` into the buffer pointed to by `s` until either a terminating new-line or EOF, which it replaces with `'\0'`. No check for buffer overrun is performed (see `BUGS` below).

Рисунок 27 – Определение функции `gets()`

Функция `gets()` просто принимает пользовательский ввод. Однако функция не проверяет, действительно ли пользовательский ввод вписывается в

структуру данных, в которой мы его храним (в данном случае это `buffer`), и, таким образом, мы можем переполнить структуру данных и повлиять на другие переменные и данные стека. Кроме того, поскольку известно, что все переменные хранятся в стеке, и что представляет собой переменная `modified` (0), всё, что нужно сделать, — это ввести достаточно данных, для того, чтобы перезаписать переменную `modified`. Злоумышленник вводит слишком много текста и может перезаписать переменную `modified` и всё остальное в стеке, включая указатели возврата. Таким образом, если злоумышленник сможет взять под контроль стек программы, он сможет эффективно контролировать всю программу и заставить её делать то, что он хочет. Например можно перезаписать указатель возврата функции в стеке на пользовательский, который указывает на вредоносную функцию. На рисунке 28 показан процесс эксплуатации данной уязвимости.

```
#include "IntegerOverflow.h"

#ifdef ALLOC_PRAGMA
# pragma alloc_text(страница, TriggerIntegerOverflow)
# pragma alloc_text(страница, IntegerOverflowIoctlHandler)
#endif // ALLOC_PRAGMA

# pragma auto_inline (выкл)

NTSTATUS TriggerIntegerOverflow (в Pvoid UserBuffer, в размере SIZE_T) {
    ULONG Count = 0;
    NTSTATUS Status = STATUS_SUCCESS;
    ULONG BufferTerminator = 0xBAD00000;
    ULONG KernelBuffer[BUFFER_SIZE] = {0};
    Size_t TerminatorSize = sizeof (BufferTerminator);

    ProbeForRead(UserBuffer, sizeof(KernelBuffer), (ULONG)__alignof(KernelBuffer));

    DbgPrint ("[ + ] UserBuffer: 0x %p \n", UserBuffer);
    DbgPrint ("[ + ] размер UserBuffer: 0x %X \n", размер);
    DbgPrint ("[ + ] KernelBuffer: 0x %p \n", &KernelBuffer);
    DbgPrint ("[ + ] KernelBuffer Size: 0x %X \n", sizeof (KernelBuffer));

#ifdef SECURE
    if (Size > (sizeof (KernelBuffer) - TerminatorSize)) {
        DbgPrint ("[ - ] недопустимый размер UserBuffer: 0x %X \n", размер);

        Status = STATUS_INVALID_BUFFER_SIZE;
        возвращенный статус;
    }
    else
        DbgPrint ("[ + ] запуск целочисленного переполнения \n");

    if ((Size + TerminatorSize) >> sizeof (KernelBuffer)) {
        DbgPrint ("[ - ] недопустимый размер UserBuffer: 0x %X \n", размер);

        Status = STATUS_INVALID_BUFFER_SIZE;
        возвращенный статус;
    }
}
#endif

// Выполните операцию копирования
while (Count < (Size / sizeof (ULONG))) {
    если (*(PULONG)UserBuffer != Буфертерминатор) {
        KernelBuffer[Count] = * (PULONG)UserBuffer;
        UserBuffer = (PULONG)UserBuffer + 1;
        Считать++;
    }
    остальное {
        передов;
    }
}
}
__except (EXCEPTION_EXECUTE_HANDLER) {
    Status = GetExceptionCode();
    DbgPrint ("[ - ] код исключения: 0x %X \n", статус);
}
```

Рисунок 28 Процесс эксплуатации уязвимости целочисленное переполнение

По комментариям следует сделать вывод о том, что в программе реализована уязвимость целочисленное переполнение, которое стало возможным из-за того, что программист не предусмотрел передачу в драйвер буфера слишком большого объема. Любой размер от 0xffffffffc до 0xfffffffff позволяет обойти проверку. Следует обратить внимание, что операция копирования завершается, если встречается символ завершения (хотя должно быть 4-байтовое выравнивание). Следовательно, не обязательно указывать размер буфера равный передаваемому размеру.

Параметр `InBufferSize`, передаваемый в функцию `DeviceIoControl`, размером `DWORD` или 4 байта. В 64-битном драйвере по адресу `HEVD!TriggerIntegerOverflow+97` выполняется следующая проверка: `ffff800`bb1c5ac7 lea r11,[r12+4] fffff800`bb1c5acc cmp r11,r13`. В коде выше происходит сравнение 64-битных регистров (префикс/суффикс для преобразования к 32-битному представлению не используется). Соответственно, регистр `r11` никогда не переполнится, поскольку туда устанавливается значение `0x100000003`, из чего можно сделать вывод, что данную уязвимость нельзя использовать на 64-битных машинах.

5.3 Процесс эксплуатации уязвимости состояние «гонки»

Код программы, содержащий уязвимость состояние гонки приведен на рисунке 29

```
for (;;) {  
    long tmo1 = 0;  
    long tmo2 = 0;  
  
    if (loop % 2 == 0)  
        tmo1 = loop % MAX_RACE_LAG_USEC;  
    else  
        tmo2 = loop % MAX_RACE_LAG_USEC;
```

Рисунок 29 – Код программы, содержащий уязвимость состояние гонки

Счетчик loop увеличивается каждую итерацию, изменяются и переменные tmo1 и tmo2. Они задают задержку в конкурирующих потоках исполнения, которые: синхронизируются на pthread_barrier; ожидают заданное количество микросекунд в холостом цикле; взаимодействуют с интерфейсом драйвера n_hdlc.

Такой способ столкновения потоков исполнения позволяет достичь состояния гонки. Далее, открываем псевдотерминал (ptmd) и устанавливаем для него протокол N_HDLC. Текст получившейся программы показан на рисунке 30.

```
ptmd = open("/dev/ptmx", O_RDWR);
if (ptmd < 0) {
    perror("[-] open /dev/ptmx");
    goto end;
}

ret = ioctl(ptmd, TIOCSETD, &ldisc);
if (ret < 0) {
    perror("[-] TIOCSETD");
    goto end;
}
```

Рисунок 30 – Установка протокола N_HDLC

Установка N_HDLC для последовательной линии приводит к автоматической загрузке драйвера n_hdlc, который поставляется как модуль ядра. Этот же эффект может быть достигнут и с помощью сервиса ldattach. Далее следует приостановить вывод данных псевдотерминала и записать в него один буфер. Функция n_hdlc_send_frames() не может выполнить отправку данного буфера и сохраняет его адрес в n_hdlc.tbuf.

Были запущены два потока, которые могут исполняться на всех доступных ядрах процессора: поток 1 - сбрасывает данные псевдотерминала с помощью ioctl; поток 2 - возобновляет приостановленный вывод с помощью ioctl. Эти потоки могут дважды поместить в tx_free_buf_list тот единственный буфер, который был записан в псевдотерминал. Далее, было возвращено исполнение

эксплоита на нулевое ядро процессора и вызывает возможную ошибку двойного освобождения памяти. Код получившейся программы показан ниже, на рисунке 31.

```
ret = sched_setaffinity(0, sizeof(single_cpu), &single_cpu);
if (ret != 0) {
    perror("[-] sched_setaffinity");
    goto end;
}

ret = close(ptmd);
if (ret != 0) {
    perror("[-] close /dev/ptmx");
    goto end;
}
```

Рисунок 32 – Ошибка двойного освобождения памяти

Функция `n_hdlc_release()` проходит по спискам `n_hdlc_buf_list` и освобождает память, занятую под буферы `n_hdlc_buf`. Если гонка была выиграна, то здесь должно произойти двойное освобождение памяти (double-free error).

Эта ошибка успешно обнаруживается с помощью динамического детектора ошибок памяти Kernel Address Sanitizer (KASAN), который сообщает о ней как об использовании после освобождения, который происходит прямо перед вторым вызовом `kfree()`.

6 Описание угроз безопасности информации возникающей при разработке программных средств и методы разработки безопасных программных средств

В соответствии с ГОСТ 19781 «Обеспечение систем обработки информации программное. Термины и определения» под программным обеспечением понимают совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ.

При проведении мероприятий по защите конфиденциальной информации, а также при их планировании и создании безопасного ПО, а также её поддержанием. Все этапы по разработке и функционированию должны соответствовать методическим указаниям и требованиям НПА, а также не нарушать законодательство РФ.

В связи с этим безопасная разработка ПО должна осуществляться на основании следующих государственных стандартов:

- ГОСТ Р 56939-2016 Защита информации. Разработка безопасного программного обеспечения. Общие требования;
- ГОСТ 2.001—2013 Единая система конструкторской документации. Общие положения;
- ГОСТ 19.101—77 Единая система программной документации. Виды программ и программных документов;
- ГОСТ 19.201—78 Единая система программной документации. Техническое задание. Требования к содержанию и оформлению;
- ГОСТ 19.402—78 Единая система программной документации. Описание программы;
- ГОСТ 19.404—79 Единая система программной документации. Пояснительная записка. Требования к содержанию и оформлению;
- ГОСТ Р ИСО/МЭК 15408-1—2012 Информационная технология. Методы и средства обеспечения безопасности. Критерии оценки

безопасности информационных технологий. Часть 1. Введение и общая модель;

- ГОСТ Р ИСО/МЭК 15408-2—2013 Информационная технология. Методы и средства обеспечения безопасности. Критерии оценки безопасности информационных технологий. Часть 2. Функциональные компоненты безопасности;

- ГОСТ Р ИСО/МЭК 15408-3—2013 Информационная технология. Методы и средства обеспечения безопасности. Критерии оценки безопасности информационных технологий. Часть 3. Компоненты доверия к безопасности;

- ГОСТ Р ИСО 10007—2019 Менеджмент организации. Руководящие указания по управлению конфигурацией;

- ГОСТ Р ИСО/МЭК 12207—2010 Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств;

- ГОСТ Р ИСО/МЭК 27001—2006 Информационная технология. Методы и средства обеспечения безопасности. Системы менеджмента информационной безопасности;

- ГОСТ Р ИСО/МЭК 27034—1 —2014 Информационная технология. Методы и средства обеспечения безопасности. Безопасность приложений. Часть 1. Обзор и общие понятия.

Предотвращение появления и устранение уязвимостей программы может быть достигнуто путем реализации разработчиком программного обеспечения (ПО) мер по разработке безопасного ПО в процессах жизненного цикла ПО, установленных ГОСТ Р ИСО/МЭК 12207.

Разработчик ПО должен:

- определить и документировать цели организации в области создания безопасного ПО и меры по разработке безопасного ПО, реализация которых направлена на достижение поставленных целей;

- обеспечить реализацию и проводить внутренние проверки мер по разработке безопасного ПО из базового набора мер по разработке безопасного ПО, в процессах жизненного цикла ПО, установленных ГОСТ Р ИСО/МЭК 12207, а также работать над улучшением процессов, связанных с разработкой безопасного ПО, если по какой либо причине не возможно выполнение мер из базового набора то должно быть описаны причины по которым, это не было сделано, а также может применить компенсирующие меры и добавить описание этих мер, и сравнить их с мерами из базового набора;

- предусмотреть выделение ресурсов, необходимых для реализации мер по разработке безопасного ПО. и обеспечить реализацию этих мер;

- проводить внутренние проверки выполнения мер по разработке безопасного ПО с разработкой требований и их планированию;

- проводить улучшения процессов, связанных с разработкой безопасного ПО на основе несоответствий, выявленных в ходе внутренних проверок и изменения целей разработчика ПО в области разработки безопасного ПО;

- создать руководство по разработке безопасного ПО в котором описать области действия руководства и перечень и описание мер по разработке безопасного ПО. подлежащих, реализации в среде разработки ПО.

Руководство по разработке безопасного ПО утверждается руководством организации, и доводится до всех сотрудников организации, имеющих отношение к разработке безопасного ПО.

Разработка документации разработчика ПО, связанной с реализацией мер по разработке безопасного ПО, может быть направлена на организацию работ по созданию безопасного ПО. выполняемых в рамках процессов жизненного цикла ПО.

Перечень документации разработчика ПО, могут входить эксплуатационные документы, а также документ:

- содержащий требования по безопасности, предъявляемые к разрабатываемому ПО;
- содержащий сведения о результатах моделирования угроз безопасности информации;
- содержащий сведения о проекте архитектуры программы;
- описывающий используемые инструментальные средства;
- содержащий информацию о прослеживаемости исходного кода программы к проекту архитектуры программы;
- содержащий порядок оформления исходного кода программы;
- содержащий сведения о результатах проведения статического анализа исходного кода программы;
- содержащий сведения о результатах проведения экспертизы исходного кода программы;
- содержащий сведения о результатах проведения функционального тестирования программы;
- содержащий сведения о результатах проведения тестирования на проникновение;
- содержащий описание процедуры передачи ПО пользователю;
- содержащий описание процедур отслеживания и исправления обнаруженных ошибок ПО и уязвимостей программы;
- содержащий описание процедуры поиска разработчиком ПО уязвимостей программы;
- описывающий реализацию и использование процедуры уникальной маркировки каждой версии ПО;
- описывающий использование системы управления конфигурацией ПО;

- описывающий меры, используемые для защиты инфраструктуры среды разработки ПО;
- содержащий сведения об обучении сотрудников.

Требования к содержанию указанных документов представлены. Требования к количеству и номенклатуре документов не предъявляется. Разработчик ПО может скомпоновать необходимые сведения по своему усмотрению. Документы могут быть выполнены в виде бумажных или электронных документов. Для организации работ, выполняемых в рамках процесса эксплуатации ПО, разработчик ПО должен передать пользователю эксплуатационные документы.

Для организации работ, выполняемые в процессах жизненного цикла ПО. Документация разработчика ПО должна содержать:

- описание используемой методологии моделирования угроз безопасности информации;
- список выявленных потенциальных угроз безопасности информации (при выявлении);
- описание проектных решений и/или указаний по применению разрабатываемого ПО, направленных на нейтрализацию выявленных потенциальных угроз безопасности информации.

Входными данными для процесса моделирования угроз безопасности информации являются в первую очередь сведения о проекте архитектуры программы (предполагаемых компонентах программы, их интерфейсах и концепции их совместного выполнения), в том числе информация о заимствованных у сторонних разработчиков ПО компонентах и информация из открытых источников (например, из банка данных угроз безопасности информации Федеральной службы по техническому и экспортному контролю (ФСТЭК России), связанная с типовыми сценариями компьютерных атак и угрозами безопасности моделирование угроз безопасности информации:

При выполнении конструирования и комплексирования ПО разработчик ПО должен реализовать следующие меры:

- использование при разработке ПО идентифицированных инструментальных средств;
- создание программы на основе уточненного проекта архитектуры программы;
- создание (выбор) и использование при создании программы порядка оформления исходного кода программы;
- статический анализ исходного кода программы;
- экспертиза исходного кода программы;

Меры по разработке безопасного программного обеспечения, реализуемые при выполнении квалификационного тестирования программного обеспечения:

- функциональное тестирование программы;
- тестирование на проникновение;
- динамический анализ кода программы;
- фаззинг-тестирование программы;

Меры по разработке безопасного программного обеспечения, реализуемые при выполнении инсталляции программы и поддержки приемки программного обеспечения:

- обеспечение защиты ПО от угроз безопасности информации, связанных с нарушением целостности в процессе его передачи пользователю;
- поставка пользователю эксплуатационных документов.

Меры по разработке безопасного программного обеспечения, реализуемые при решении проблем в программном обеспечении в процессе эксплуатации:

- реализация и использование процедуры отслеживания и исправления обнаруженных ошибок ПО и уязвимостей программы;
- систематический поиск уязвимости программы.

Меры по разработке безопасного программного обеспечения, реализуемые в процессе менеджмента документацией и конфигурацией программы:

- реализация и использование процедуры уникальной маркировки каждой версии ПО;
- использование системы управления конфигурацией ПО.

Меры по разработке безопасного программного обеспечения, реализуемые в процессе менеджмента инфраструктурой среды разработки программного обеспечения:

- защита от несанкционированного доступа к элементам конфигурации;
- резервное копирование элементов конфигурации;
- регистрация событий, связанных с фактами изменения элементов конфигурации.

Меры по разработке безопасного программного обеспечения, реализуемые в процессе менеджмента людскими ресурсами:

- периодическое обучение сотрудников;
- периодический анализ программы обучения сотрудников.

При выполнении инсталляции программы и поддержки приемки ПО разработчик ПО должен реализовать следующие меры:

- обеспечение защиты ПО от угроз безопасности информации, связанных с нарушением целостности, в процессе его передачи пользователю;
- поставка пользователю эксплуатационных документов.

Реализация мер способствует достижению следующих целей:

- обеспечение соответствия экземпляра ПО. Переданного разработчиком, и экземпляра ПО. полученного пользователем;
- обеспечение пользователя эксплуатационными документами в объеме, достаточном для правильной настройки и безопасного применения программы.

В результате успешной реализации мер:

- ПО поставляется пользователю, при этом пользователем может быть обнаружено любое расхождение между оригиналом ПО и полученной версией;

- пользователю поставляются эксплуатационные документы в объеме, достаточном для правильной настройки и безопасного применения программы.

При выполнении решения проблем в ПО разработчик ПО должен реализовать следующие меры:

- реализация и использование процедуры отслеживания и исправления обнаруженных ошибок ПО и уязвимостей программы;
- систематический поиск уязвимостей программы.

Реализация мер способствует достижению цели устранения ошибок ПО и уязвимостей программы, выявляемых в процессе эксплуатации ПО.

В результате успешной реализации мер ошибки ПО и уязвимости программы, обнаруженные в процессе эксплуатации ПО, регистрируют, анализируют и устраняют.

Для организации работ, выполняемых в процессах жизненного цикла ПО и подтверждения соответствия требованиям настоящего стандарта документация разработчика ПО должна содержать описание методов приема и обработки сообщений от пользователей в ситуациях, когда неизвестна ранее уязвимость программы используется для проведения компьютерной или сетевой атаки на информационную систему пользователя.

В экстренных ситуациях разработчик ПО должен быть способен к выпуску обновлений ПО в обход стандартной процедуры выпуска новых версий ПО. Если экстренный выпуск обновлений ПО невозможен, разработчик ПО должен предложить альтернативные способы временного решения проблемы, включая использование пользователем дополнительных средств защиты.

Для организации работ, выполняемых в процессах жизненного цикла ПО и подтверждения соответствия требованиям настоящего стандарта документация разработчика ПО должна содержать:

- перечень экстренных ситуаций, при которых возможен выпуск обновлений ПО в обход стандартной процедуры выпуска новых версий ПО;

- описание альтернативных (если экстренный выпуск обновлений ПО невозможен) способов временного решения проблемы, связанной с экстренной ситуацией.

Разработчик ПО должен проводить систематический поиск уязвимостей программы. Поиск уязвимостей программы следует проводить с использованием:

- моделирования угроз безопасности информации;
- статического анализа кода программы;
- экспертизы исходного кода программы;
- функционального тестирования программы;
- тестирования на проникновение;
- динамического анализа кода программы;
- фаззинг-тестирования программы.

Разработчику ПО следует обеспечить конфиденциальность информации, связанной с выявленными уязвимостями программы.

При выполнении менеджмента документацией и конфигурацией программы разработчик ПО должен реализовать следующие меры:

- реализация и использование процедуры уникальной маркировки каждой версии ПО;
- использование системы управления конфигурацией ПО.

Система управления конфигурацией ПО должна идентифицировать элементы конфигурации, которые связаны с реализацией функций безопасности ПО (при наличии требований безопасности, предъявляемых к ПО).

Для организации работ, выполняемых в процессах жизненного цикла ПО и подтверждения соответствия требованиям настоящего стандарта документация разработчика ПО должна содержать перечень элементов конфигурации, которые связаны с реализацией функций безопасности ПО.

Реализация мер способствует достижению следующих целей:

- создание программы на основе уточненного проекта архитектуры программы с использованием идентифицированных инструментальных средств с определёнными опциями (настройками);
- выявление и удаление недостатков программы (потенциально уязвимых конструкций) в исходном коде программы;
- формирование исходных данных для выполнения динамического анализа кода программы, фаззинг-тестирования программы и тестирования на проникновение в рамках процесса квалификационного тестирования ПО.

В результате успешной реализации мер:

- программа должна быть создана с учетом требований по безопасности, установленных в процессе анализа требований к ПО;
- при создании программы должны быть использованы только идентифицированные разработчиком ПО инструментальные средства с определёнными опциями (настройками);
- в исходном коде программы должны быть выявлены и устранены недостатки программы (потенциально уязвимые конструкции);
- необходимо сформировать исходные данные (перечень выявленных потенциально уязвимых конструкций в исходном коде программы), используемые при проведении динамического анализа кода программы, фаззинг-тестирования программы и тестирования на проникновение.

7 Рекомендации по проведению лабораторного практикума по исследованию уязвимостей программного кода

Программа, которая использует уязвимость для разрушения защиты другой программы, называется эксплойтом. Наибольшую опасность представляют эксплойты, предназначенные для получения доступа к уровню суперпользователя или, другими словами, повышения привилегий. Эксплойт переполнения буфера достигает этого путём передачи программе специально изготовленных входных данных. Такие данные переполняют выделенный буфер и изменяют данные, которые следуют за этим буфером в памяти.

Если программа не проверяет длину введённого нового пароля, то любые данные, длина которых превышает размер выделенного для их хранения буфера, будут просто записаны поверх того, что находилось после буфера. Злоумышленник может вставить в эту область памяти инструкции на машинном языке, например, шелл-код, выполняющие любые действия с привилегиями суперпользователя — добавление и удаление учётных записей пользователей, изменение паролей, изменение или удаление файлов и т.д. Если исполнение в этой области памяти разрешено и в дальнейшем программа передаст в неё управление, система исполнит находящийся там машинный код злоумышленника.

Правильно написанные программы должны проверять длину входных данных, чтобы убедиться, что они не больше, чем выделенный буфер данных. Однако программисты часто забывают об этом. В случае если буфер расположен в стеке и стек «растёт вниз» (например в архитектуре x86), то с помощью переполнения буфера можно изменить адрес возврата выполняемой функции, так как адрес возврата расположен после буфера, выделенного выполняемой функцией. Тем самым есть возможность выполнить произвольный участок машинного кода в адресном пространстве процесса. Использовать переполнение буфера для искажения адреса возврата возможно

даже если стек «растёт вверх» (в этом случае адрес возврата обычно находится перед буфером).

Даже опытным программистам бывает трудно определить, насколько то или иное переполнение буфера может быть уязвимостью. Это требует глубоких знаний об архитектуре компьютера и о целевой программе. Было показано, что даже настолько малые переполнения, как запись одного байта за пределами буфера, могут представлять собой уязвимости.

Переполнения буфера широко распространены в программах, написанных на относительно низкоуровневых языках программирования, таких как язык ассемблера, Си и С++, которые требуют от программиста самостоятельного управления размером выделяемой памяти. Устранение ошибок переполнения буфера до сих пор является слабо автоматизированным процессом. Системы формальной верификации программ не очень эффективны при современных языках программирования.

Многие языки программирования, например, Perl, Python, Java и Ada, управляют выделением памяти автоматически, что делает ошибки, связанные с переполнением буфера, маловероятными или невозможными. Perl для избежания переполнений буфера обеспечивает автоматическое изменение размера массивов. Однако системы времени выполнения и библиотеки для таких языков всё равно могут быть подвержены переполнениям буфера, вследствие возможных внутренних ошибок в реализации этих систем проверки. В Windows доступны некоторые программные и аппаратно-программные решения, которые предотвращают выполнение кода за пределами переполненного буфера, если такое переполнение было осуществлено. Среди этих решений — DEP в Windows XP SP2, OSsurance и Anti-Execute.

Обработчик ошибок, который проверяет размер передаваемого буфера, представлен на рисунке 33.

```

#include <iostream>
#include <string>
using namespace std;

int main()
{
    setlocale(LC_ALL, "ru");
    int buf = 10; // размер буфера
    string in;
    cout << "Введите входную строку: ";
    cin>>in;

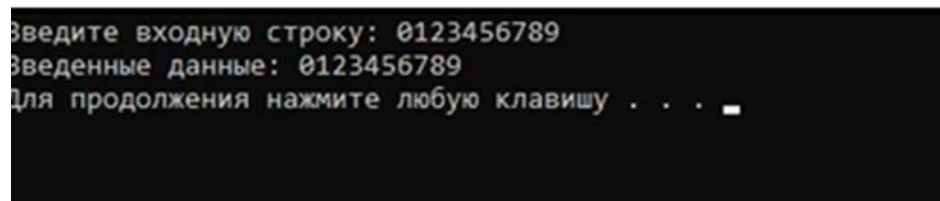
    if (in.size() > buf)
    {
        cout << "Ошибка! Переполнение буфера" << endl;
    }
    else
    {
        cout << "Введенные данные: " << in << endl;
    }

    system("pause");
    return 0;
}

```

Рисунок 33 – Код программы, проверяющий размер передаваемого буфера

Результаты работы обработчика ошибок представлены на рисунках 34 и 35.

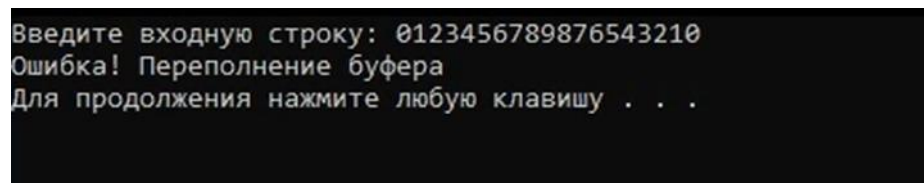


```

Введите входную строку: 0123456789
Введенные данные: 0123456789
Для продолжения нажмите любую клавишу . . .

```

Рисунок 34 – Непереполненный буфер



```

Введите входную строку: 0123456789876543210
Ошибка! Переполнение буфера
Для продолжения нажмите любую клавишу . . .

```

Рисунок 35 – Переполненный буфер

8 Вредные психофизиологические факторы, влияющие на опорно-двигательный аппарат специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

8.1 Вредные психофизиологические факторы, влияющие на опорно-двигательный аппарат специалиста по защите информации.

Любая профессиональная деятельность несет в себе определенную степень риска профессиональных заболеваний.

Казалось бы, у людей, вынужденных проводить перед монитором компьютера долгие часы, работа спокойная и «физически легкая», поэтому профессиональные болезни им не грозят. Однако на самом деле это не так. Любая малоподвижная и сидячая деятельность создает множество предпосылок для развития серьезных заболеваний.

Однако эта деятельность имеет свои специфические минусы, самой большой проблемой специалиста по ИБ является малоподвижный образ жизни. Ведь творческая работа порой настолько поглощает, что ты и не задумываешься, сколько часов проводишь перед монитором.

Основные вредные факторы, влияющие на состояние здоровья людей, работающих за компьютером:

- сидячее положение в течение длительного времени;
- воздействие электромагнитного излучения монитора;
- утомление глаз, нагрузка на зрение;
- перегрузка суставов кистей;
- стресс при потере информации;
- пыль и грязь: аллергия и кишечные инфекции.

Работая за компьютером, человек должен долгое время пребывать в вынужденном положении, двигательная активность сильно ограничивается, что, естественно, оказывает негативное воздействие на позвоночник и мышцы спины. Длительное статическое положение тела приводит к постепенной

атрофии мышц шеи, спины, грудной клетки и брюшного пресса. Ослабленные мышцы оказываются не в состоянии поддерживать позвоночник в правильном физиологическом положении и в итоге возникают нарушения осанки, переходящие в дальнейшем в искривления позвоночника. Чаще всего позвоночник искривляется в боковой плоскости, и это заболевание называется сколиозом. Еще одна причина развития нарушения осанки и искривления позвоночника - сидение на рабочем месте в неправильной позе. Вначале человек стремится контролировать себя и держать правильную осанку, но, погрузившись в процесс работы, утрачивает контроль над положением корпуса, и со временем вообще перестает обращать внимание на то, как он сидит. Все это создает условия, как для возникновения заболевания, так и для его прогрессирования.

Многочасовое нахождение в статичной позе ухудшает обменные процессы в веществе межпозвонковых дисков, что вызывает их преждевременное старение. Межпозвоночные диски утрачивают упругость, теряют воду и постепенно растрескиваются. Этот патологический процесс приводит к развитию остеохондроза. Дальнейшее прогрессирование заболевания заканчивается тем, что через образовавшиеся трещины начинается постепенно выпадение пульпозного ядра, т. е. образуется межпозвоночная грыжа. При межпозвоночной грыже нередко наблюдается сдавление и реактивное воспаление нервных корешков, отходящих от спинного мозга, называемое радикулитом.

Работа с компьютерной мышью и клавиатурой в течение длительного времени вызывает значительное перенапряжение сухожильно-связочного аппарата, мышц предплечья и кистей рук. В процессе работы руки специалиста ИБ или оператора ПК несут достаточно низкую физическую нагрузку, а перенапряжение связано с выполнением однотипных движений в течение всего трудового дня. Перенапряжение может послужить причиной возникновения достаточно серьезного нервно-мышечного расстройства.

На подушечках пальцев имеется огромное количество нервных окончаний. Эти чувствительные окончания в процессе работы с клавиатурой постоянно раздражаются, что приводит к нарушению передачи нервных импульсов по проводящим путям от кончиков пальцев к коре головного мозга. В результате появляются судороги мышц предплечья и кистей рук, а также нарушается мелкая моторика кисти. Английские врачи, первые описавшие данное заболевание, назвали его *repetitive strain injury (RSI)*, что переводится как повторяющаяся травма, растяжение.

8.2 Методы и средства физической культуры, снижающие их воздействие

В регламентированные перерывы с целью снижения развивающегося у пользователей нервно-эмоционального напряжения, утомления мышечного скелета, целесообразно выполнять комплексы специальных профилактико-реабилитационных упражнений.

В целях профилактики и предупреждения заболеваний специалистов по защите информации необходимо проводить специальную оздоровительную систему профреабилитации от последствий гиподинамии. При этом должны выполняться комплексные меры, включающие физические упражнения, водно-воздушные процедуры, массаж, прием витаминов и микроэлементов, минеральных веществ, аутогенную тренировку.

Лучший способ повышения двигательной активности – спорт и активный отдых.

Существует много форм физической культуры, которые используются для нормализации функционального состояния человека, а так же для профилактики заболеваний:

- подвижные спортивные игры являются хорошим средством физического развития, нормализация психоэмоционального состояния, совершенствования опорно-двигательного аппарата и мускулатуры;

- ходьба и бег. Имеют огромное для профилактики опорно-двигательного аппарата и оздоровление организма;

- велосипедный спорт способствует усилению обмена веществ, тренировке опорно двигательной системы и др., велопрогулки показаны при заболеваниях сердечно - сосудистой, дыхательной систем и нарушении обмена веществ, а также при последствии травм суставов ног (для разработки тугоподвижности и тренировки мышц). Зимой велопрогулки заменяются упражнениями на велотренажерах;

- плавание - отличное тренирующее средство и закаливающее. Если плавание сочетать с прогулками, играми, то оздоравливающий эффект еще более возрастает. Плавание усиливает деятельность мышечной системы и обмен веществ, а при травмах и заболеваниях позвоночника ведет к исчезновению болей и улучшению подвижности в суставах. Осторожность следует проявлять при заболеваниях сердца, легких, ЛОР - органов, особенно в осенне-зимнее время из-за опасности переохлаждения и обострения или возникновения простуды.

Как видно тихая работа программиста очень опасна, но правильная оценка угроз и здоровый образ жизни уменьшают их до минимума и зачастую исключают болезни у специалистов по защите информации.

Заключение

Уязвимости программного кода представляют большую опасность для пользователей устройств. Основная причина этого в том, что используя уязвимости программного кода злоумышленники могут копировать, изменять, удалять или блокировать информацию пользователя.

Основной задачей дипломного проекта было создание лабораторного стенда для исследования программного кода, имеющие уязвимости, а также конфигурирование виртуального окружения для размещения и эксплуатации бинарных уязвимостей и способов защиты от них. Разработку программного кода, содержащего приведенные уязвимости.

Был произведен обзор наиболее распространенных уязвимостей программного кода, таких как переполнение буфера, целочисленное переполнение и состояние гонки в программном коде.

Были разработаны рекомендации по проведению лабораторного практикума по исследованию уязвимостей программного кода с использованием разработанного лабораторного стенда.

По результатам работы был сделан вывод о том, что созданный и описанный лабораторный стенд полностью соответствует для изучения уязвимостей программного кода.

Таким образом, все пункты технического задания были выполнены полностью.

Список используемых источников

1 Информ.-справочный портал [Электронный ресурс]: Группа параллельных и распределенных информационных систем <http://pdos.csail.mit.edu>;

2 19 смертных грехов, угрожающих безопасности программ. Как не допустить типичных ошибок / Дэвид Лебланк, Майкл Ховард, Джон Виега» - ДМК Пресс; Москва; 2011; Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/4812> (дата обращения: 30.01.2020). — Режим доступа: для авториз. пользователей;

3 Информ.-справочный портал [Электронный ресурс]: Как устроены дыры в безопасности <https://habr.com/ru/post/266591/>;

4 Защищенный код / Майкл Ховард, Дэвид Лебланк - 2-е издание, исправленное, Москва 2005; Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/4812> (дата обращения: 30.01.2020). — Режим доступа: для авториз. пользователей;

5 Информ.-справочный портал [Электронный ресурс]: Платформа для разработчиков. <https://github.com/>;

6 Дневник охотника за ошибками. Путешествие через джунгли проблем безопасности программного обеспечения / К. Тобиас ; перевод с английского А.Н. Киселев. — Москва : ДМК Пресс, 2013. — 240 с. — ISBN 978-5-94074-374-3. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/4812> (дата обращения: 30.01.2020). — Режим доступа: для авториз. пользователей.

7 Информ.-справочный портал [Электронный ресурс]: Виды компьютерных уязвимостей <https://it-black.ru/vidy-kompyuternykh-uyazvimostey/>;

8 Информ.-справочный портал [Электронный ресурс]: Классификация угроз информационной безопасности <https://en.ppt-online.org/125155>;

9 Информ.-справочный портал [Электронный ресурс]: Поиск уязвимостей в программах с помощью анализаторов кода <https://www.viva64.com/ru/a/0028/>;

10 Информ.-справочный портал [Электронный ресурс]: Переполнение

буфера: причины, эффективность и методы решения проблемы.
<https://fb.ru/article/461145/>.

Приложение А

(обязательное)

Состав лабораторного виртуального стенда, использующегося при
выполнении проекта

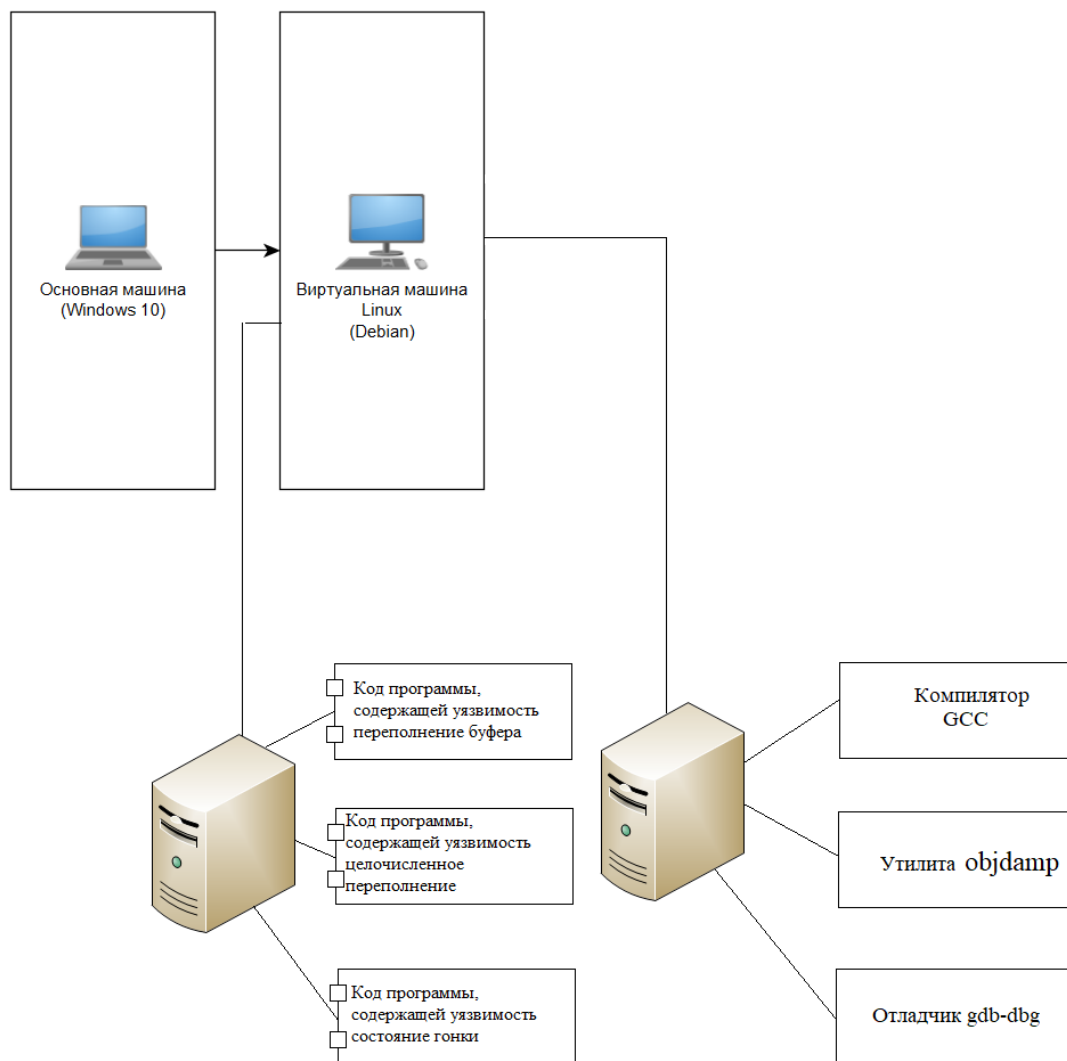


Рисунок А.1 – Состав лабораторного виртуального стенда

Приложение Б
(обязательное)
Обзор уязвимостей программных средств



Рисунок Б.1 – Обзор уязвимостей программных средств

Приложение В

(обязательное)

Угрозы безопасности информации возникающие, при разработке
программных средств согласно ГОСТ Р 58412-2019

ГОСТ Р 58412-2019



Рисунок В.1 - Угроз безопасности информации, возникающие при разработке программных средств согласно ГОСТ Р 58412-2019

Приложение Г
(обязательное)

Методы защиты от угроз безопасности, возникающих при разработке
программных средств согласно ГОСТ Р 56939-2016

ГОСТ Р 56939-2016



Рисунок Г.1 - Методы защиты от угроз безопасности, возникающих при разработке программных средств согласно ГОСТ Р 56939-2016

Приложение Д
(обязательное)

Схема эксплуатации уязвимостей программных средств, имеющих
уязвимость состояние гонки

Схема эксплуатации программного кода, имеющего уязвимость состояние гонки

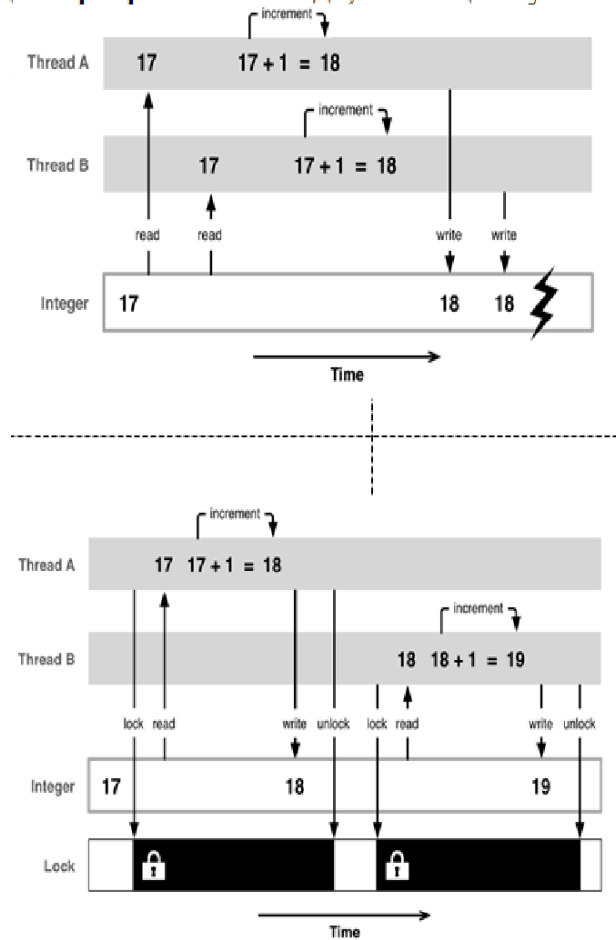


Рисунок Д.1 – Схема эксплуатации уязвимостей программных средств, имеющих уязвимость состояние гонки

Приложение Е
(обязательное)

Схема эксплуатации программных средств, имеющих уязвимость
переполнение буфера

Схема эксплуатации программного кода имеющего уязвимость «Переполнение буфера»

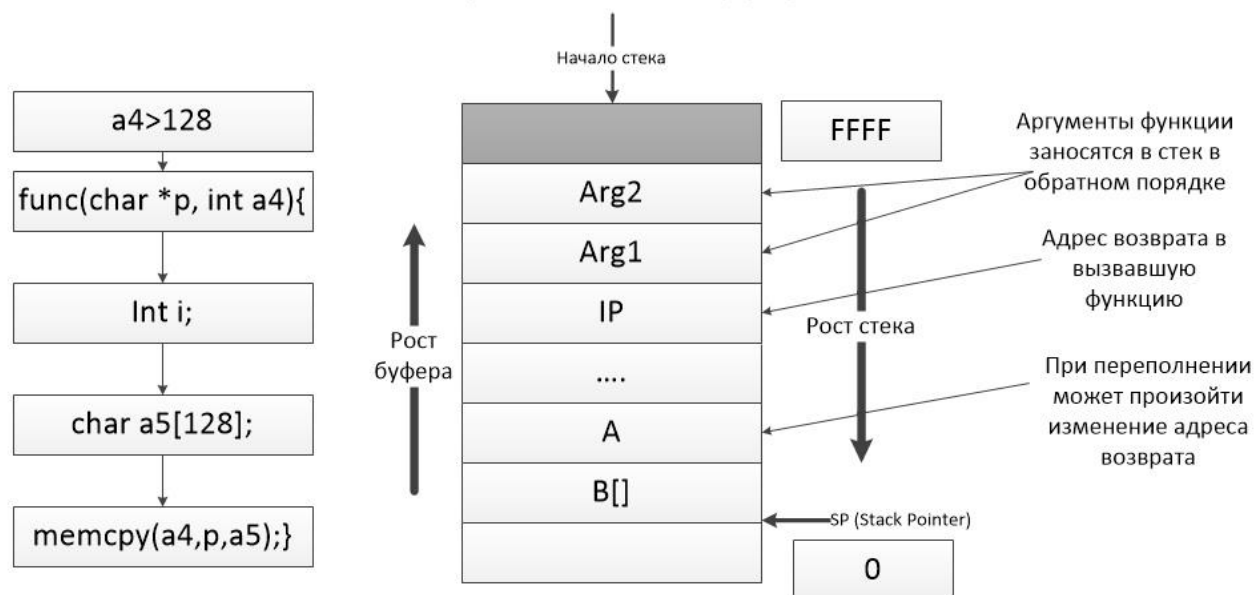


Рисунок Е.1 - Схема эксплуатации уязвимостей программных средств,
имеющих уязвимость переполнение буфера

Приложение Ж
(обязательное)

Схема эксплуатации программного кода, имеющего уязвимость целочисленное
переполнение

Схема эксплуатации программного кода, имеющего уязвимость целочисленное переполнение

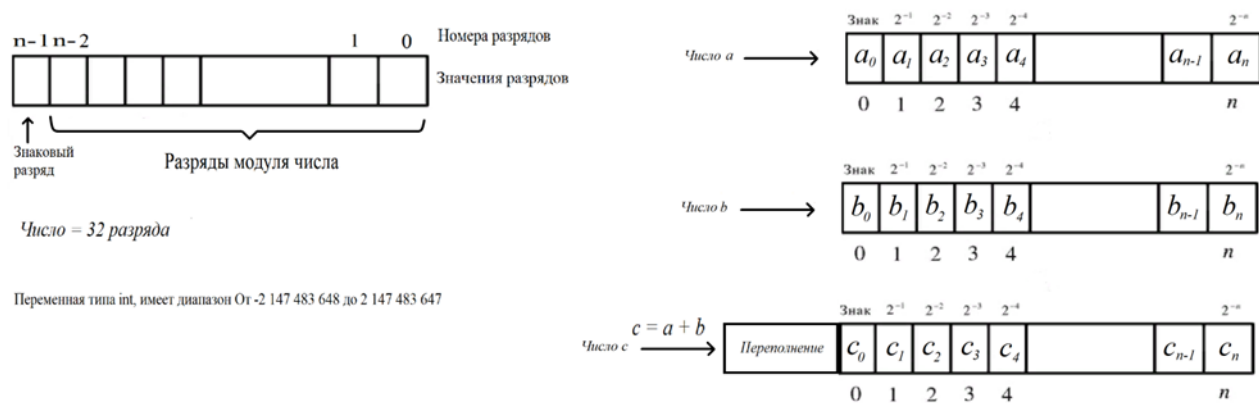


Рисунок Ж.1 - Схема эксплуатации уязвимости программного кода, имеющего уязвимость целочисленное переполнение