Цели и задачи лабораторной работы

Цель лабораторной работы: изучение способов эксплуатации уязвимостей переполнения буфера на стеке и связанных с ними уязвимостей форматной строки.

1 Теоретические сведения

См. лекцию об уязвимостях.

2 Поиск и эксплуатация ROP-цепочек

2.1 Утилита ropper

Для поиска гаджетов используются утилиты наподобие Ropper (https://github.com/sashs/Ropper/tree/master) и RopGadget. Данные утилиты способны как осуществлять поиск гаджетов, так и составлять полноценные ROP-цепочки. Далее будет рассмотрена утилита Ropper.

Входными данными для анализа является исполняемый файл/разделяемая библиотека, в котором будет осуществляться поиск (флаг -f). Поскольку в самом исполняемом файле достаточного количества гаджетов может не оказаться, поиск гаджетов может проводиться в библиотеках, которые использует исполняемый файл. Для отображения списка используемых библиотек используется утилита ldd (использование: *ldd <исполняемый файл>*).

Помимо простого поиска гаджетов утилита может осуществлять автоматического составления ROP-цепочки с генерацией заготовки Python-скрипта, выводящего строку, пригодную для подачи в атакуемое приложение. При этом скрипт предназначен для выполнения в Python 2, но может быть немного доработан для запуска через Python 3.

Ropper генерирует только заготовку скрипта, поскольку утилита не знает 1) размер буфера, который необходимо переполнить до того, как будет перезаписан адрес возврата; 2) адрес загрузки библиотеки.

Адреса загрузки библиотек можно узнать в gdb с помощью команды *info proc mappings* (программа уже должна быть запущена). Известный адрес загрузки библиотеки вставляется в скрипт либо передается в Ropper с помощью аргумента *-I <адрес>*.

Для преодоления буфера перед адресом возврата наиболее простым вариантом является модификация скрипта (см. переменную *rop*, которая изначально равна пустой строке).

2.2 Эксплуатация ROP-цепочек

Ropper может генерировать только ROP-цепочки для вызовов execve и mprotect. Первый вызов запускает на выполнение другую программу. Второй вызов может использоваться для изменения разрешений для области памяти (позволяет снять заперт на исполнение кода в стеке/куче).

Поскольку для решения задачи Л/Р достаточно добиться вывода “Access granted”, вместо получения экземпляра оболочки /bin/sh (что обычно и является целью shell-кодов, но требует дополнительных действий) можно выполнить скрипт, выводящий требуемую строку. В этом случае, для построения ROP-цепочки нужно передать в Ropper аргумент *--chain “execve cmd=<путь к скрипту>"*. Для упрощения путь к скрипту следует сделать кратным 8 байтам.

Скриптом с точки зрения Unix является текстовый файл, начинающийся со строки вида *#!<путь к интерпретатору>*. Ниже приведен текст скрипта Python3, который ничего не выводит:

```
#!/usr/bin/python3
1+1
```

Поскольку скрипт должен иметь разрешения на выполнение, для файла скрипта необходимо выполнить команду *chmod u+x <путь к скрипту>*.

2.3 Возможные проблемы при построении ROP-цепочек

В некоторых случаях Ropper может генерировать некорректную ROP-цепочку, в рамках которой на стек дополнительно кладутся либо со стека излишне убираются данные (к примеру, если один из гаджетов содержит инструкцию `leave`, которая не компенсируется инструкциями других гаджетов либо дополнительными данными в самой цепочке). В частности, такое поведение наблюдалось при выполнении л//р в Ubuntu 24.04 и не наблюдалось в Ubuntu 22.04

Решением данной проблемы может быть замена ОС или модификация ROP-цепочки с учетом добавления/отъема данных путем замены гаджета, добавления компенсирующих гаджетов или добавления в саму ROP-цепочку данных. Для поиска гаджетов с требуемой инструкцией в Ropper используется параметр *--search* (например, *--search ‘pop rbx’*).

Другой возможной проблемой является некорректная интеграция имени файла в ROP-цепочку. К примеру, Ropper может попытаться добавить к имени файла лишние /, если длина пути не кратна 8. В этом случае можно изменить имя файла и перегенерировать цепочку, либо

вручную поправить имя файла в цепочке, сдвинув его к началу и добавив в конце нулевые байты вместо /.

3 Задание на лабораторную работу

Задание на лабораторную работу: добиться вывода на экран надписи “Access granted” путем построения ROP-цепочки.

Итоговый порядок выполнения работы:

1) Отключить ASLR.
2) Написать скрипт на Python/bash, который выводит “Access granted”.
3) Разрешить выполнение скрипта с помощью *chmod*.
4) Определить загружаемые библиотеки с помощью *ldd*, выбрать целевую библиотеку.
5) Найти адрес загрузки библиотеки с помощью IDA/GDB.
6) Сгенерировать скрипт генерации ROP-цепочки с помощью Ropper.
7) Определить размер переполняемого буфера.
8) Поправить скрипт генерации с учетом размера переполняемого буфера.
9) Сгенерировать строку и передать ее на вход программы.
10) Если получили SIGBUS/SIGSEGV, проверьте в отладчике (IDA/GDB), что выполнение доходит до инструкции syscall:
    1) Если не доходит – исправить ROP-цепочку (см. пункт 2.3);
    2) Если доходит до syscall, но вызов не приводит к нужному результату – проверить корректность имени файла и другие аргументы в соответствующих регистрах.
    3) goto 8;
11) Если выведено Access Granted – break.

Примечание: в целом, можно пойти дальше и получить экземпляр оболочки /bin/sh, но такой вариант требует наличия открытого потока ввода, через который будут считываться команды. Если вы перенаправите поток ввода из файла, вводить команды вы не сможете. Есть обходной путь – перенаправить вывод из другой команды, но вот какой – можете подумать сами.