Отчёт по третьему разделу внешнего курса

Артём Дмитриевич Петлин

Содержание

# 1. Цель работы

Выполнить третий раздел внешнего курса “Системный администратор Linux с нуля”.

# 2. Задание

Задания восьмого, девятого, десятого и одиннадццатого модулей, а также тесты.

# 3. Теоретическое введение

* Модуль 8. Настройка сети и SSH
* Модуль 9: Управление пакетами
* Модуль 10: Управление логами
* Модуль 10: Управление логами

# 4. Выполнение практических заданий

|  |
| --- |
| Рисунок 1: модуль 8 |

Настраиваем статический IP на тестовом сервере вручную.

|  |
| --- |
| Рисунок 2: модуль 8 |

В файле /etc/network/interfaces прописываем конфигурацию для интерфейса eth0.

|  |
| --- |
| Рисунок 3: модуль 8 |

Проверяем доступность внешних ресурсов через ping.

|  |
| --- |
| Рисунок 4: модуль 8 |

Удаляем текущий IP-адрес с интерфейса. Запускаем любой веб-сервер на 80 порту (например, nginx) и проверяем, что он работает.

|  |
| --- |
| Рисунок 5: модуль 8 |

С помощью утилиты dig узнаем IP-адреса популярных сервисов.

|  |
| --- |
| Рисунок 6: модуль 8 |

Проверяем, слушает ли SSH-порт на сервере.

|  |
| --- |
| Рисунок 7: модуль 8 |

Изменяем порт SSH и запрещаем вход по паролю.

|  |
| --- |
| Рисунок 8: модуль 8 |

Настраиваем подключение по ключу и авторизуемся.

|  |
| --- |
| Рисунок 9: модуль 8 |

Разрешаем SSH-доступ только для определенных пользователей. Убеждаемся, что для другого пользователя подключение не сработает.

|  |
| --- |
| Рисунок 10: модуль 8 |

Настраиваем UFW для ограничения доступа только для SSH.

|  |
| --- |
| Рисунок 11: модуль 8 |

Настраиваем UFW, чтобы доступ по SSH был только из вашей локальной сети.

|  |
| --- |
| Рисунок 12: модуль 8 |

|  |
| --- |
| Рисунок 13: модуль 8 |

Используя fail2ban, создаем jail, который будет блокировать IP-адрес после трех успешных запросов к nginx.

|  |
| --- |
| Рисунок 14: модуль 8 |

Проверяем, как работает блокировка при множественных запросах.

|  |
| --- |
| Рисунок 15: модуль 9 |

Находим и устанавливаем утилиту htop.

|  |
| --- |
| Рисунок 16: модуль 9 |

Удаляем утилиту htop, затем устанавливаем заново с полной очисткой (purge).

|  |
| --- |
| Рисунок 17: модуль 9 |

Выполняем полное обновление системы.

|  |
| --- |
| Рисунок 18: модуль 9 |

Настраиваем автоматическую установку обновлений безопасности.

|  |
| --- |
| Рисунок 19: модуль 9 |

Удаляем устаревшие и неиспользуемые зависимости.

|  |
| --- |
| Рисунок 20: модуль 9 |

Устанавливаем пакет вручную через dpkg, предварительно скачав его с сайта.

|  |
| --- |
| Рисунок 21: модуль 9 |

Анализируем, какие зависимости потребуются.

|  |
| --- |
| Рисунок 22: модуль 10 |

|  |
| --- |
| Рисунок 23: модуль 10 |

Просматриваем содержимое основного системного журнала и журнала аутентификации, используя утилиту для постраничного просмотра.

|  |
| --- |
| Рисунок 24: модуль 10 |

Выводим последние 20 записей из бинарного системного журнала с помощью journalctl.

|  |
| --- |
| Рисунок 25: модуль 10 |

Проверяем наличие и расположение файла журнала ошибок веб-сервера Nginx.

|  |
| --- |
| Рисунок 26: модуль 10 |

Изучаем любую одну строку из файла /var/log/auth.log и вручную определяем в ней: временную метку, имя сервиса (процесса) и описание события.

|  |
| --- |
| Рисунок 27: модуль 10 |

Проверяем текущий статус служб rsyslog и systemd-journald с помощью systemctl.

|  |
| --- |
| Рисунок 28: модуль 10 |

|  |
| --- |
| Рисунок 29: модуль 10 |

Генерируем тестовое сообщение с помощью утилиты logger. Затем находим это сообщение сначала в выводе journalctl, а потом в файле /var/log/syslog. Используя journalctl, фильтруем и выводим только те события, которые имеют уровень важности error (err) или более критичный. Заглядываем в конфигурационный файл rsyslog (например, /etc/rsyslog.d/50-default.conf) и находим строку, отвечающую за направление сообщений от категорий (facility) auth и authpriv в файл /var/log/auth.log. Отправляем в системный журнал сообщение с явно указанным уровнем важности warning. Убеждаемся, что оно появляется в выводе journalctl при фильтрации по этому уровню.

|  |
| --- |
| Рисунок 30: модуль 10 |

Находим все строки в файле /var/log/auth.log, в которых упоминается неудачная попытка входа по паролю (Failed password), не обращая внимания на регистр символов. Строим конвейер команд, который сначала найдет все успешные SSH-подключения (Accepted password) в /var/log/auth.log, а затем извлечет из этих строк только IP-адреса подключившихся клиентов. Составляем список уникальных IP-адресов, с которых были зафиксированы неудачные попытки входа в систему. Собираем статистику и выведите пять IP-адресов, с которых было зафиксировано наибольшее количество успешных входов по SSH.

|  |
| --- |
| Рисунок 31: модуль 10 |

Используя journalctl, отображаем все системные журналы за последние 15 минут.

|  |
| --- |
| Рисунок 32: модуль 10 |

Находим в журнале аутентификации /var/log/auth.log все попытки входа от имени несуществующих пользователей.

|  |
| --- |
| Рисунок 33: модуль 10 |

Определяем IP-адрес, с которого было совершено наибольшее количество неудачных попыток подбора пароля (Failed password). После определения наиболее подозрительного IP-адреса из предыдущего шага, извлекаем из /var/log/auth.log абсолютно все записи, связанные с этим IP, и сохраняем их в отдельный файл incident\_report.log. Рассчитываем контрольную сумму SHA256 для созданного файла incident\_report.log, чтобы зафиксировать его целостность и доказать, что он не изменялся после сбора. Проверяем, какие команды выполнялись с правами суперпользователя (через sudo) вашим текущим пользователем.

|  |
| --- |
| Рисунок 34: модуль 10 |

Изучаем существующую конфигурацию logrotate для системного менеджера пакетов (apt или dpkg) в директории /etc/logrotate.d/. Определяем, как часто происходит ротация, сколько архивных копий хранится и используется ли сжатие.

|  |
| --- |
| Рисунок 35: модуль 10 |

Создаем собственный конфигурационный файл /etc/logrotate.d/testapp для управления вымышленным лог-файлом /var/log/testapp.log. Настраиваем его на ежедневную ротацию, хранение четырех архивных копий и сжатие старых логов.

|  |
| --- |
| Рисунок 36: модуль 10 |

Проверяем созданную конфигурацию logrotate на синтаксические ошибки, выполнив «сухой запуск» в режиме отладки.

|  |
| --- |
| Рисунок 37: модуль 10 |

Принудительно запускаем ротацию для лога testapp и проверьте результат: убеждаемся, что старый лог был сжат и переименован, а на его месте появился новый пустой файл. Пишем строку конфигурации для rsyslog, которая будет пересылать абсолютно все логи (*.*) по протоколу TCP на удаленный сервер с адресом logs.example.com и стандартным портом 514.

|  |
| --- |
| Рисунок 38: модуль 11 |

Устанавливаем Podman и проверяем установку.

|  |
| --- |
| Рисунок 39: модуль 11 |

|  |
| --- |
| Рисунок 40: модуль 11 |

Запускаем контейнер.

|  |
| --- |
| Рисунок 41: модуль 11 |

Запускаем Nginx и проверяем, что он работает.

|  |
| --- |
| Рисунок 42: модуль 11 |

Разворачиваем сайт в Nginx, используя локальную папку с HTML-файлами.

|  |
| --- |
| Рисунок 43: модуль 11 |

Запускаем контейнер Nginx с ограничением памяти в 100 МБ и проверяем его работу.

|  |
| --- |
| Рисунок 44: модуль 11 |

|  |
| --- |
| Рисунок 45: модуль 11 |

|  |
| --- |
| Рисунок 46: модуль 11 |

Исследуем поведение контейнера при исчерпании выделенной памяти.

|  |
| --- |
| Рисунок 47: модуль 11 |

|  |
| --- |
| Рисунок 48: модуль 11 |

|  |
| --- |
| Рисунок 49: модуль 11 |

|  |
| --- |
| Рисунок 50: модуль 11 |

Создаем и запускаем кастомный веб-сайт на базе nginx с использованием Podman.

|  |
| --- |
| Рисунок 51: модуль 11 |

Сохраняем образ и переносим его на другой сервер.

# 5. Выполнение тестовых заданий

|  |
| --- |
| Рисунок 52: тест |

Мы используем команду ip a add для временного добавления IP-адреса к сетевому интерфейсу. Директива auto в файле /etc/network/interfaces указывает системе, какие интерфейсы нужно автоматически активировать (поднимать) во время загрузки. Для удаления конкретного IP-адреса с интерфейса мы используем команду ip a del с указанием полного адреса с маской и имени интерфейса.

|  |
| --- |
| Рисунок 53: тест |

Настройки, заданные с помощью команды ip, являются временными и действуют только до перезагрузки системы. При статической настройке сети в файле /etc/network/interfaces мы указываем шлюз по умолчанию с помощью директивы gateway в блоке настроек интерфейса.

|  |
| --- |
| Рисунок 54: тест |

Мы используем современную утилиту ss с ключами -t , -u , -l, -n, -p для получения полного списка открытых портов и связанных с ними процессов. Ключ -p в команде ss заставляет ее отображать идентификатор процесса (PID) и его имя, которое использует данный сокет.

|  |
| --- |
| Рисунок 55: тест |

Мы используем команду dig для выполнения DNS-запросов и получения подробной информации о доменных именах, включая их IP-адреса. Для быстрой проверки доступности удаленного TCP-порта мы используем утилиту netcat с ключами -v и -z.

|  |
| --- |
| Рисунок 56: тест |

По умолчанию демон SSH-сервера прослушивает входящие подключения на TCP-порту 22. Главный файл конфигурации для серверной части SSH находится по пути /etc/ssh/sshd\_config. Файл ssh\_config предназначен для клиентской части. Для временной остановки системного сервиса мы используем команду systemctl stop.

|  |
| --- |
| Рисунок 57: тест |

Для настройки аутентификации по SSH-ключу мы копируем содержимое файла публичного ключа в файл authorized\_keys на сервере. Директива PermitRootLogin no в файле /etc/ssh/sshd\_config запрещает прямое подключение к серверу по SSH под учетной записью root, что является важной мерой безопасности.

|  |
| --- |
| Рисунок 58: тест |

После настройки правил мы активируем межсетевой экран UFW командой ufw enable. Это применяет правила и включает автозагрузку файрвола при старте системы. Основные настройки jails Fail2ban, которые определяют правила блокировки, хранятся в файле /etc/fail2ban/jail.conf. Чтобы удалить правило по его номеру из списка, выведенного командой ufw status numbered, мы используем команду ufw delete [номер].

|  |
| --- |
| Рисунок 59: тест |

Параметр bantime в конфигурации jail Fail2ban определяет длительность блокировки IP-адреса в секундах после превышения лимита попыток. Для ограничения доступа к порту по источнику мы используем синтаксис ufw allow from [источник] to any port [порт]. Это разрешает подключения только с указанной подсети.

|  |
| --- |
| Рисунок 60: тест |

Команда apt autoremove удаляет пакеты, которые были установлены автоматически как зависимости и больше не нужны. Сами целевые пакеты удаляются командой remove, а их конфиги остаются. Команда apt purge удаляет пакет вместе с его конфигурационными файлами. Зависимости, установленные с ним, при этом остаются в системе. Мы регулярно выполняем apt update, чтобы обновить локальную базу данных пакетов. Без этого система не будет знать о новых версиях пакетов.

|  |
| --- |
| Рисунок 61: тест |

Пакет unattended-upgrades позволяет нам настроить автоматическую установку обновлений безопасности и других пакетов без ручного вмешательства. После apt update мы выполняем apt upgrade, чтобы установить все доступные обновления для установленных пакетов. Команда apt full-upgrade выполняет более интеллектуальное обновление, которое может удалять obsolete пакеты или устанавливать новые зависимости, что иногда необходимо для полного обновления системы.

|  |
| --- |
| Рисунок 62: тест |

При возникновении проблем с зависимостями мы используем команду apt –fix-broken install, чтобы попытаться автоматически исправить нарушенные зависимости. Принудительное удаление пакета, от которого зависят другие программы, приводит к “разрыву зависимостей”. Для фильтрации вывода других команд и поиска конкретного пакета мы используем конвейер с grep.

|  |
| --- |
| Рисунок 63: тест |

Мы предпочитаем устанавливать .deb файлы через apt, так как он автоматически разрешает и устанавливает все зависимости. APT проверяет целостность и подлинность пакетов из репозиториев с помощью GPG-ключей, что защищает систему от установки модифицированных или вредоносных пакетов. В корпоративной среде, если нужного пакета нет в утвержденных репозиториях, правильным действием является запрос на его добавление через систему тикетов, а не самостоятельная установка из непроверенных источников.

|  |
| --- |
| Рисунок 64: тест |

Мы используем логи в первую очередь для этих трех целей: найти причину неисправности, выяснить обстоятельства взлома и определить “узкие места” в системе. Ошибка 502 обычно указывает на проблему связи веб-сервера с backend-процессом. В современных системах логи хранятся в двух основных местах: классические текстовые файлы в /var/log/ и централизованный бинарный журнал systemd, который мы просматриваем с помощью journalctl.

|  |
| --- |
| Рисунок 65: тест |

Обычно systemd-journald действует как первичный сборщик логов, а затем, при наличии настроек, перенаправляет сообщения в демон syslog для постоянного хранения в привычных текстовых файлах в /var/log/. Сбой запуска критичной системной службы — это значимое негативное событие, которое классифицируется уровнем error, а не просто информационным сообщением. Категория в syslog указывает на подсистему или программу-источник сообщения.

|  |
| --- |
| Рисунок 66: тест |

Чтобы uniq мог корректно подсчитать повторяющиеся строки, они должны следовать друг за другом. В awk $1, $2, $3 и т.д. обозначают первое, второе, третье и последующие поля в строке. Мы используем конвейер: первый grep отфильтровывает строки с “error”, а второй grep с ключом -v удаляет из этого результата строки, содержащие “healthcheck”.

|  |
| --- |
| Рисунок 67: тест |

Большое количество неудачных попыток входа за короткий промежуток времени, особенно для стандартных имен пользователей, является классическим признаком автоматизированной brute-force атаки. Расчет хеша фиксирует текущее состояние файла. Команды, выполненные через sudo, подробно логируются в журналах аутентификации.

|  |
| --- |
| Рисунок 68: тест |

Основная задача logrotate — ротация, архивация и удаление старых лог-файлов по заданному расписанию и правилам. Удаленный сбор логов обеспечивает целостность журналов. Grafana Loki спроектирован именно для этого сценария.

|  |
| --- |
| Рисунок 69: тест |

Команда podman info выводит подробную информацию о среде Podman: версию, конфигурацию, хранилища, сеть и т.д., что помогает нам проверить корректность его настройки. Ключевое архитектурное отличие Podman в том, что он использует архитектуру без демона. Ключ –rm автоматически удаляет контейнер сразу после того, как он завершит свою работу.

|  |
| --- |
| Рисунок 70: тест |

Безопасность Podman для многопользовательских сред обусловлена его способностью работать в rootless-режиме. Как уже было отмечено, Podman не требует постоянно работающего фонового демона.

|  |
| --- |
| Рисунок 71: тест |

Мы используем команду podman run для запуска нового контейнера. Флаг -v используется для монтирования директорий или файлов с хостовой машины внутрь контейнера. Комбинация флагов -i и -t позволяет нам запустить контейнер в интерактивном режиме с псевдо-TTY.

|  |
| --- |
| Рисунок 72: тест |

По умолчанию podman ps показывает только работающие контейнеры. Чтобы увидеть все контейнеры, включая остановленные, мы добавляем флаг -a. Флаг –rm заставляет Podman автоматически удалять контейнер сразу после его остановки.

|  |
| --- |
| Рисунок 73: тест |

Для мониторинга потребления ресурсов работающими контейнерами в реальном времени мы используем команду podman stats. Флаг -d запускает контейнер в фоновом режиме. После запуска управление возвращается в терминал, а контейнер продолжает работать независимо. После изменения unit-файлов systemd мы выполняем systemctl daemon-reload.

|  |
| --- |
| Рисунок 74: тест |

Когда контейнер запущен как systemd-сервис, его логи интегрируются в общий журнал systemd. Флаг –memory устанавливает максимальный лимит оперативной памяти, который может использовать контейнер.

|  |
| --- |
| Рисунок 75: тест |

Для загрузки Docker-образов из удаленного реестра мы используем команду podman pull. Файл policy.json определяет политику доверия для образов контейнеров. В нем мы настраиваем, из каких реестров разрешено скачивать образы, требуется ли для них цифровая подпись и какие ключи являются доверенными. Команда podman build используется для сборки собственного образа контейнера.

|  |
| --- |
| Рисунок 76: тест |

Мы используем podman build -t myapp . для создания собственного образа из инструкций в Dockerfile, который находится в текущей директории. Результатом будет образ с именем myapp. Если в политике доверия (policy.json) для определенного реестра или образа указана директива signedBy, то Podman будет проверять наличие и валидность цифровой подписи у образа.

# 6. Оценки тестов

|  |
| --- |
| Рисунок 77: тест 1 |

|  |
| --- |
| Рисунок 78: тест 2 |

|  |
| --- |
| Рисунок 79: тест 3 |

|  |
| --- |
| Рисунок 80: тест 4 |

|  |
| --- |
| Рисунок 81: тест 5 |

|  |
| --- |
| Рисунок 82: тест 6 |

|  |
| --- |
| Рисунок 83: тест 7 |

|  |
| --- |
| Рисунок 84: тест 8 |

|  |
| --- |
| Рисунок 85: тест 9 |

|  |
| --- |
| Рисунок 86: тест 10 |

|  |
| --- |
| Рисунок 87: тест 11 |

|  |
| --- |
| Рисунок 88: тест 12 |

|  |
| --- |
| Рисунок 89: тест 13 |

|  |
| --- |
| Рисунок 90: тест 14 |

|  |
| --- |
| Рисунок 91: тест 15 |

|  |
| --- |
| Рисунок 92: тест 16 |

|  |
| --- |
| Рисунок 93: тест 17 |

# 7. Выводы

Мы выполнили третий раздел внешнего курса “Системный администратор Linux с нуля”.

# Список литературы

1. https://study.selectel.ru/members/courses/course756726784647