Лабораторная работа №5.

Блокировка и ограничение доступа пользователей, контроль уязвимостей в системах Linux

*Цель работы:* Ознакомление с методами блокировки пользовательских учетных записей, ограничением доступа пользователей через PAM и другие инструменты, а также использование Trivy для контроля за уязвимостями.

1. Блокировка пользовательских учетных записей

Блокировка учетной записи может быть необходима в случае подозрительной активности или компрометации.

Способ 1. Блокировка и разблокировка пользователей с помощью команды *passwd*

Команда *passwd* работает с паролями учетной записи пользователя.

Зайдем под учетной записью суперпользователя и создадим пользователя *blockuser* c паролем *P@ssw0rd*:

# useradd blockuser # passwd blockuser

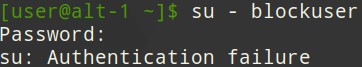
Чтобы заблокировать пользователя *blockuser*, запустим команду *passwd*

с опцией -l (--lock):

# passwd -l blockuser

Теперь выйдем из учетной записи суперпользователя. Попробуем зайти под учетной записью *blockuser*:

$ su - blockuser



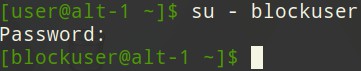
Зайдем под учетной записью суперпользователя.

Чтобы разблокировать пользователя *blockuser*, запустим команду

*passwd* c опцией -u (--unlock): # passwd -u blockuser

Выйдем из учетной записи суперпользователя. Попробуем зайти под учетной записью *blockuser*:

$ su - blockuser



Способ 2. Блокировка и разблокировка пользователей с помощью команды *usermod*

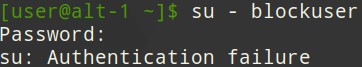
Команда *usermod* используется для изменения учетных записей пользователей.

Чтобы заблокировать пользователя *blockuser*, запустим команду

*usermod* с опцией -L (--lock): # usermod -L blockuser

Теперь выйдем из учетной записи суперпользователя. Попробуем зайти под учетной записью *blockuser*:

$ su - blockuser



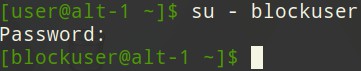
Зайдем под учетной записью суперпользователя.

Чтобы разблокировать пользователя *blockuser*, запустим команду

*usermod* c опцией -U (--unlock): # usermod -U blockuser

Выйдем из учетной записи суперпользователя. Попробуем зайти под учетной записью *blockuser*:

$ su - blockuser



Способ 3. Блокировка и разблокировка пользователей с помощью команды *chage*

Команда *chage* используется для изменения информации об истечении срока действия пароля пользователя.

Используем команду *chage* для автоматической блокировки неактивного пользователя после определенного количества дней бездействия.

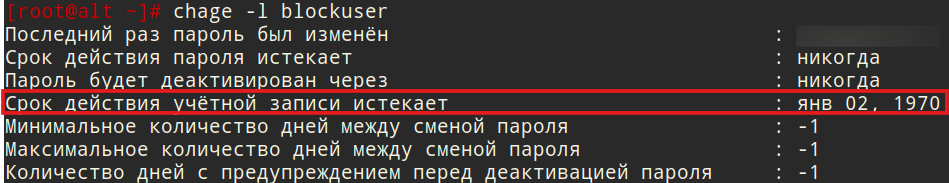
Заблокируем пользователя *blockuser*:

# chage -E 1 blockuser

По сути, мы установили дату истечения срока действия 2 января 1970

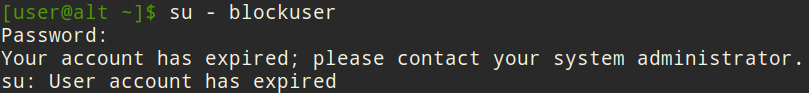
года.

Выведем информацию о пароле пользователя *blockuser*: # chage -l blockuser



Выйдем из учетной записи суперпользователя. Попробуем зайти под учетной записью *blockuser*:

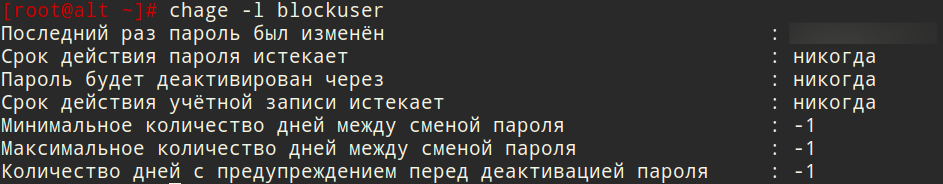
$ su - blockuser



Зайдем под учетной записью суперпользователя.

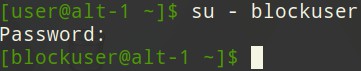
Разблокируем пользователя *blockuser*, удалив дату истечения срока действия учетной записи:

# chage -E -1 blockuser

Выведем информацию о пароле пользователя *blockuser*: # chage -l blockuser

Выйдем из учетной записи суперпользователя. Попробуем зайти под учетной записью *blockuser*:

$ su - blockuser



1. Ограничение возможности входа пользователей
   1. Настройка PAM для ограничения доступа пользователей

Настройка PAM для ограничения доступа пользователей может быть выполнена с помощью изменения конфигурационных файлов, обычно расположенных в */etc/pam.d/*.

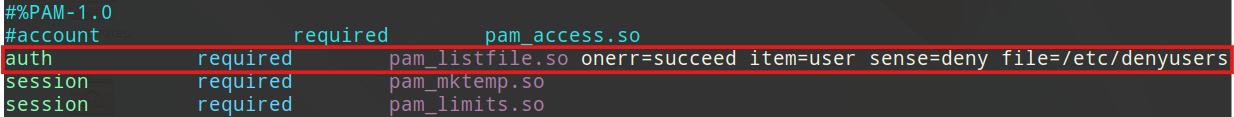
Несколько модулей PAM были разобраны в предыдущих лабораторных работах. Дополним список модулями *pam\_listfile.so* м *pam\_time.so*.

Пример 1. Ограничение доступа пользователей с помощью модуля pam\_listfile.so

Запретим авторизацию пользователя *blockuser*. Для этого воспользуемся модулем *pam\_listfile.so*. Данный модуль позволяет ограничить доступ для пользователей на основе файла со списком.

Зайдем под учетной записью суперпользователя. Отредактируем файл */etc/pam.d/system-auth-common*: # vim /etc/pam.d/system-auth-common

Добавим следующую строку:

auth required pam\_listfile.so onerr=succeed item=user sense=deny file=/etc/denyusers

Модуль *auth* указывает на процесс аутентификации пользователя.

Флаг *required* означает, что данный модуль должен быть выполнен. Если он не пройдет, аутентификация не будет успешной, но другие модули все равно будут выполняться.

Параметр *onerr=succeed* параметр указывает, что если произойдет ошибка (например, файл не найден), то модуль будет считать, что проверка прошла успешно.

Параметр *item=user* указывает, что проверяемым элементом будет имя пользователя.

Параметр *sense=deny* указывает, что если имя пользователя найдено в файле, то доступ будет запрещен.

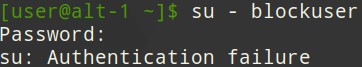
Параметр *file=/etc/denyusers* указывает на путь к файлу, в котором перечислены пользователи, которым запрещен доступ. Каждый пользователь должен быть записан с новой строки.

Теперь создадим файл */etc/denyusers*: # vim /etc/denyusers

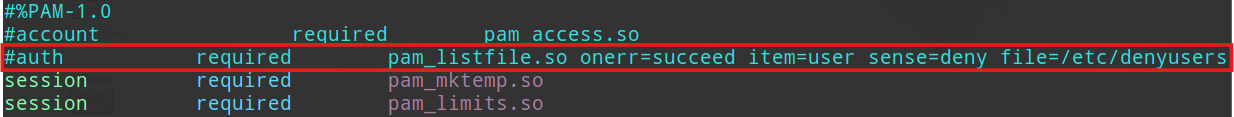
Запишем в созданный файл пользователя *blockuser*: blockuser

Выйдем из учетной записи суперпользователя. Попробуем зайти под учетной записью *blockuser*:

$ su - blockuser

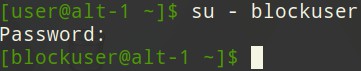


Зайдем под учетной записью суперпользователя. Закомментируем строку в файле */etc/pam.d/system-auth-common*:

# auth required pam\_listfile.so onerr=succeed item=user sense=deny file=/etc/denyusers

Выйдем из учетной записи суперпользователя. Попробуем еще раз зайти под учетной записью *blockuser*:

$ su - blockuser



Пример 2. Ограничение доступа пользователей с помощью модуля pam\_time.so

Разрешим доступ пользователю blockuser только с 20:00 до 21:00 по будням. Для этого воспользуемся модулем *pam\_time.so*. Модуль *pam\_time.so* позволяет задавать временные ограничения для доступа пользователей.

Откроем файл */etc/security/time.conf*:

# vim /etc/security/time.conf

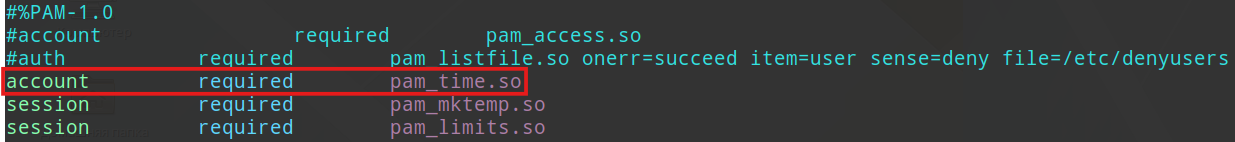
Добавим следующее правило:

\*;\*;blockusers;MoTuWeThFr2000-2100

Отредактируем файл */etc/pam.d/system-auth-common*: # vim /etc/pam.d/system-auth-common

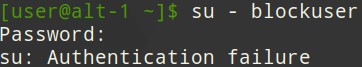
Добавим следующую строку:

account required pam\_time.so



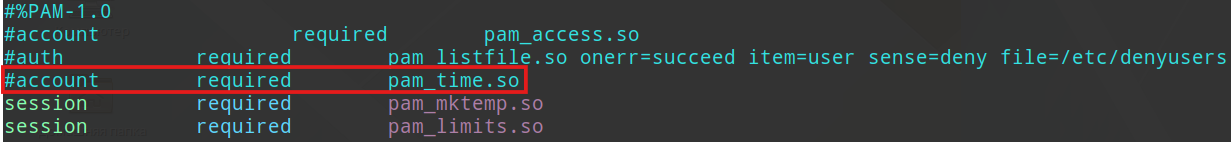
Выйдем из учетной записи суперпользователя. Попробуем зайти под учетной записью *blockuser*:

$ su - blockuser



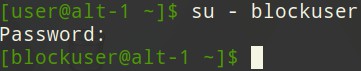
Зайдем под учетной записью суперпользователя. Закомментируем строку в файле */etc/pam.d/system-auth-common*:

#account required pam\_time.so



Выйдем из учетной записи суперпользователя. Попробуем еще раз зайти под учетной записью *blockuser*:

$ su - blockuser



* 1. Использование fail2ban для защиты от брутфорс-атак

*Fail2ban* — локальный сервис, который отслеживает логи служб, работающих на сервере, и при обнаружении подозрительной активности блокирует IP-адреса возможных злоумышленников.

Для защиты от брутфорс-атак (взломов путём перебора паролей) Fail2ban можно настроить, например, для защиты службы SSH.

Установим пакет *fail2ban*:

# apt-get update && apt-get install fail2ban

Запускаем и добавляем в автозагрузку сервис *fail2ban*: # systemctl enable --now fail2ban

Защита для SSH начнет работать сразу после установки. По умолчанию *fail2ban* будет на 10 минут блокировать IP-адреса, с которых в течение 10 минут было выполнено 5 неудачных попыток авторизации.

Настройки *fail2ban* хранятся в конфигурационном файле

*/etc/fail2ban/jail.conf*.

# cat /etc/fail2ban/jail.conf | less

Настройки разделены по секциям. В [DEFAULT] указываются общие параметры, которые определяют работу *fail2ban* в целом и применяются для большинства служб. Специфические настройки для той или иной службы прописываются внутри соответствующей секции (например, [sshd], [apache-auth], [vsftpd]).

Менять какие-либо параметры напрямую в файле *jail.conf* не рекомендуется. Вместо этого необходимо создать новый файл, разместив его в директории */etc/fail2ban/jail.d/ssh.conf*.

Важно! Перед созданием файла необходимо переключить сбор логов на

*systemd*.

Для этого отредактируем файл /*etc/fail2ban/jail.conf*: # vim /etc/fail2ban/jail.conf

Заменим следующую строку:

before = paths-altlinux.conf На строку:

before = paths-altlinux-systemd.conf

Теперь можно размещать свои настройки в

*/etc/fail2ban/jail.d/название.conf*.

Создадим файл */etc/fail2ban/jail.d/ssh.conf* и добавим в него следующие строки:

[sshd] enabled = true findtime = 120

maxretry = 3

bantime = 43200

Такая настройка означает, что если с какого-либо IP-адреса в течение 2 минут будут выполнены 3 неудачные попытки авторизоваться, *fail2ban* заблокирует этот адрес на 12 часов.

В секции *DEFAULT* могут быть указаны такие параметры, как период блокировки (bantime), количество попыток (maxretry) и пр.

В файле */etc/fail2ban/jail.d/ssh.conf* указаны следующие параметры:

*enabled* — обязательный параметр, определяющий включение или отключение секции. Для включения должно быть установлено значение true, оно указано по умолчанию;

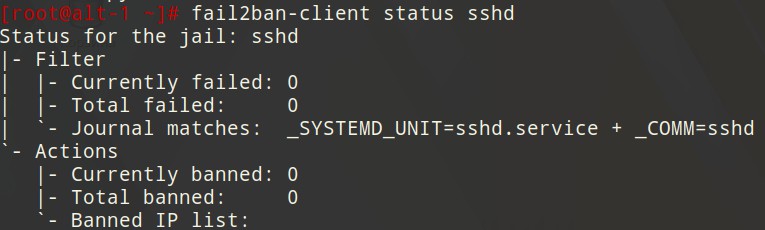
*bantime* — продолжительность бана в секундах, то есть период, на который подозрительный IP-адрес будет заблокирован;

*maxretry* — количество неудачных попыток в течение периода findtime, после которых будет выполнена блокировка;

*findtime* — период в секундах, в течение которого действие (в данном случае — неудачная попытка подключения) должно повториться определенное количество раз (maxretry), после чего будет выполнена блокировка.

После создания файла необходимо перезапустить службу fail2ban: # systemctl restart fail2ban

Информацию о работе модуля sshd с помощью следующей команды: # fail2ban-client status sshd



Проверим работы сервиса *fail2ban*.

Попробуем подключиться к пользователю *user* с *alt-2* на *alt-1*.

|  |  |
| --- | --- |
| Имя хоста | IP-адрес |
| alt-1 | 192.168.10.10/24 |
| alt-2 | 192.168.10.20/24 |

Примечание! Локальная сеть между машинами настроена по умолчанию.

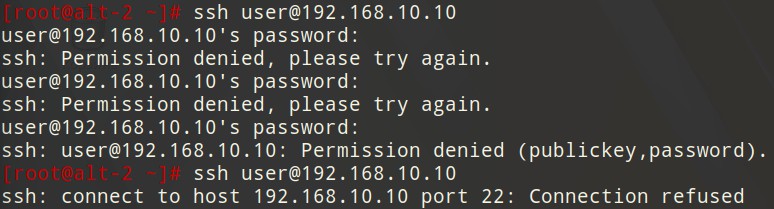
Важно! Предварительно необходимо проверить статус службы sshd: # systemctl status sshd

Если служба не запущена, необходимо запустить ее: # systemctl enable --now sshd

Зайдем под учетной записью суперпользователя на хосте *alt-2*.

Подключаемся по протоколу *ssh* к пользователю *user* с *alt-2* на *alt-1* и намеренно вводим неверных пароль:

[# ssh user@192.168.10.10](mailto:%23sshuser@192.168.10.10)

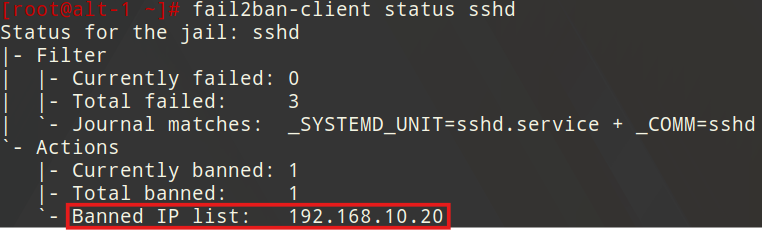


После 3-й неудачной попытки подключиться к пользователю *user* с *alt-2*

на *alt-1* возможно будет только через 12 часов.

Дополнительно выведем информацию о работе модуля sshd с помощью следующей команды:

# fail2ban-client status sshd



* 1. Использование /sbin/nologin для предотвращения входа некоторых пользователей

Скрипт */sbin/nologin* используется для предотвращения входа пользователей в систему. Он обычно назначается в качестве оболочки для учетных записей, которым не требуется вход в систему. Когда пользователь пытается войти в систему с такой оболочкой, они получают сообщение о том, что вход запрещен, и сессия завершается.

С помощью команды *usermod* с опцией *-s* назначим */sbin/nologin* в качестве оболочки для пользователя *blockuser*:

# usermod -s /sbin/nologin blockuser

Проверим изменение оболочки пользователя *blockuser* в файле

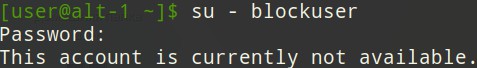
*/etc/passwd*:

# cat /etc/passwd | grep blockuser



Выйдем из учетной записи суперпользователя. Попробуем зайти под учетной записью *blockuser*:

$ su - blockuser



Зайдем под учетной записью суперпользователя.

Вернем возможность пользователю *blockuser* входить в систему. Для этого сменим оболочку пользователя *blockuser* на */bin/bash*: # usermod -s /bin/bash blockuser

Проверим изменение оболочки пользователя *blockuser* в файле

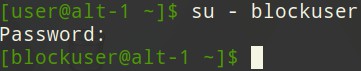
*/etc/passwd*:

# cat /etc/passwd | grep blockuser



Выйдем из учетной записи суперпользователя. Попробуем зайти под учетной записью *blockuser*:

$ su - blockuser



1. Контроль за уязвимостями с помощью *Trivy*

*Trivy* — сканер уязвимостей в образах контейнеров, файловых системах и репозиториях Git. Кроме того, trivy может находить ошибки в файлах конфигурации, запрограммированные конфиденциальные данные, использование несовместимых лицензий в проекте.

Установим пакет *trivy*:

# apt-get update && apt-get install trivy

Синтаксис утилиты *trivy*:

*trivy* <команда> [--scaners <сканер1,сканер2>] <цель>

Доступные команды:

*image (i)* — сканировать образ контейнера;

*filesystem (fs)* — сканировать локальную файловую систему; *repository (repo)* — сканировать git-репозиторий (удаленно); *vm* — сканировать образ виртуальной машины;

*kubernetes (k8s)* — сканировать кластер кубернетес;

*aws* — сканировать учётную запись AWS; *config* — сканировать файлы конфигурации; *rootfs* — сканировать rootfs;

*sbom* — сканировать используемые пакеты ОС и программные зависимости (SBOM);

*completion* — сгенерировать скрипт автозаполнения для указанной оболочки;

*module* — управление модулями; *plugin* — управление плагинами; *server* — режим сервера;

*version* — вывести версию.

Сканеры:

*vuln* — известные уязвимости (CVE) (по умолчанию);

*config* — проблемы с IAC и неправильные настройки;

*secret* — конфиденциальная информация и секреты (по умолчанию);

*license* — лицензии на программное обеспечение.

Для получения дополнительной информации по применению *trivy*

используем опцию *--help*: # trivy --help

Уровни уязвимостей:

UNKNOWN — уязвимости или риски, для которых нет достаточной информации для определения их уровня серьезности.

LOW — уязвимости, которые имеют минимальное влияние на безопасность системы и могут быть легко устранены.

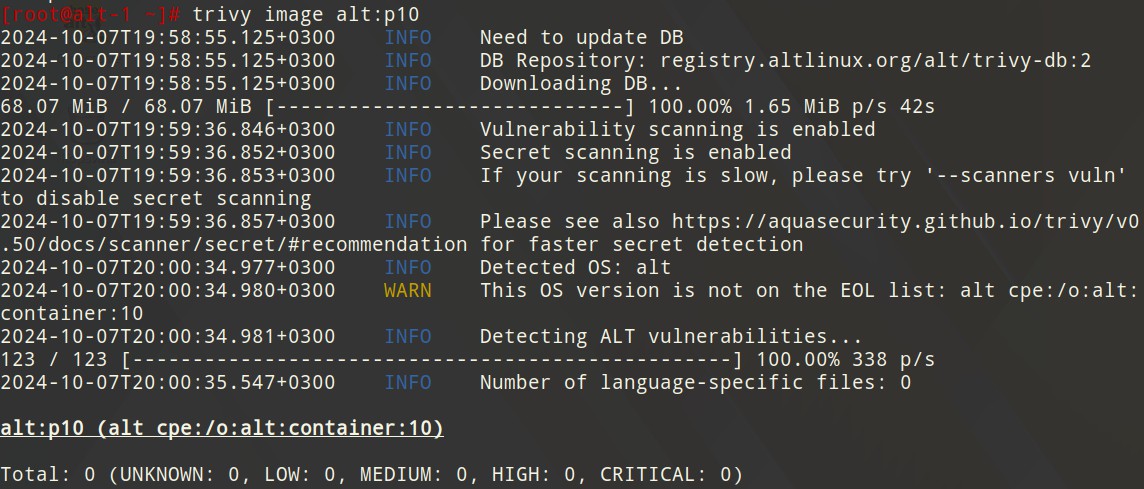
MEDIUM — уязвимости, которые могут представлять некоторую угрозу, но требуют дополнительных условий для эксплуатации или имеют ограниченное воздействие.

HIGH — серьезные уязвимости, которые могут привести к значительным последствиям, если они будут использованы злоумышленниками.

CRITICAL — очень серьезные уязвимости, которые могут позволить злоумышленникам полностью контролировать систему или данные, требующие немедленного внимания.

Разберем несколько примеров использования сканера *Trivy*: Отсканируем образ контейнера *alt:p10* на уязвимости:

# trivy image alt:p10



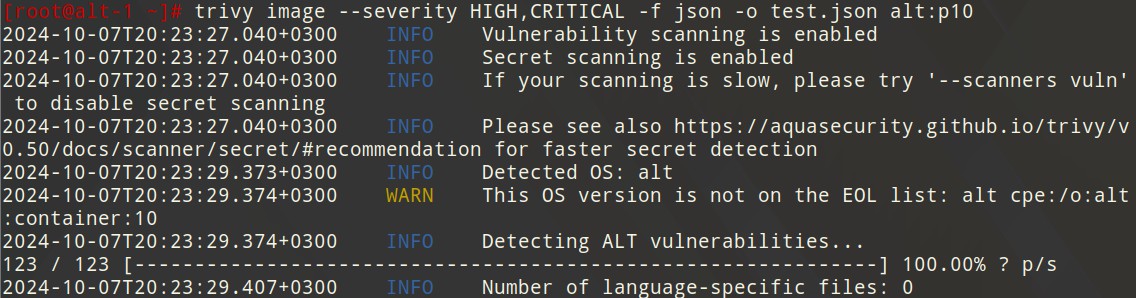
По результатам сканирования образ контейнера *alt:p10* не содержит известных уязвимостей.

*Тег* — метка, которая используется для обозначения конкретной версии или варианта образа Docker. Теги позволяют пользователям легко управлять и идентифицировать различные версии одного и того же образа.

Теги обычно имеют формат *<имя\_образа>:<тег>*.

Отсканируем образ контейнера *alt:p10* на наличие уязвимостей HIGH и CRITICAL с сохранением результата в формате JSON в файл:

# trivy image --severity HIGH,CRITICAL -f json -o test.json alt:p10

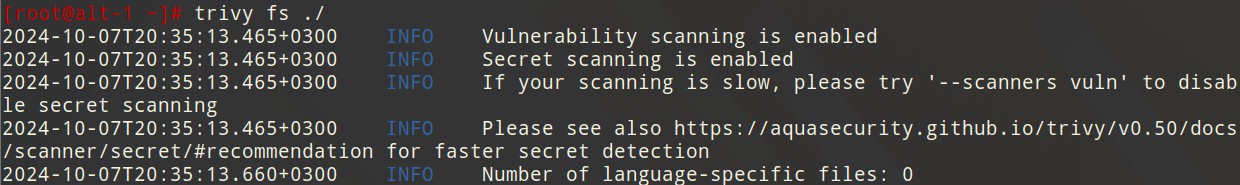


Выведем содержимое файла *test.json*:

# cat test.json | less

Отсканируем все файлы по определенному пути (в текущей директории):

# trivy fs ./



1. Анализ данных из ALTRepo API

Проект *ALTRepo* предназначен для агрегации информации о дистрибутивах и пакетной базе, а также для предоставления интерфейса к базе данных, используемой в задачах разработки, тестирования и развития системы.

Все данные проекта хранятся в ALTRepo DB, доступ к которой предоставляется посредством REST API: [https://rdb.altlinux.org](https://rdb.altlinux.org/).

REST API (Representational State Transfer Application Programming Interface) — архитектурный стиль для разработки сетевых приложений, который использует подход REST для взаимодействия между клиентом и сервером.

Основные характеристики REST API включают:

1. Статус вне сеанса (Stateless).

Каждый запрос от клиента к серверу должен содержать всю необходимую информацию для понимания и обработки запроса. Сервер не сохраняет никакой информации о состоянии клиента между запросами.

1. Система ресурсов (Resources).

В REST API все ключевые элементы представляются как ресурсы, которые могут быть доступны через уникальные идентификаторы (URI).

1. Использование стандартных методов HTTP.

REST API обычно использует стандартные методы HTTP для выполнения операций над ресурсами:

GET: Получение данных.

POST: Создание нового ресурса.

PUT: Обновление существующего ресурса. DELETE: Удаление ресурса.

1. Представление ресурсов.

Ресурсы могут иметь разные представления (форматы), такие как JSON, XML и другие. Клиент и сервер могут обмениваться данными в выбранном формате.

1. Кэширование.

Ответы от сервера могут быть закэшированы клиентом или промежуточными серверами, что помогает снизить нагрузку на сервер и улучшить производительность.

Перейдем в ALTRepo API и разберем его основные разделы.

|  |  |
| --- | --- |
| Раздел | Описание |
| api | Основные функции API, которые могут предоставлять базовые операции с системами и сущностями ALTRepo. |
| auth | Методы авторизации и проверки прав доступа пользователей. |
| task | Методы для работы с задачами, такими как создание, обновление и получение статуса задач. |
| task/progress | API для отслеживания прогресса выполнения задач, например, выполнения сборок или других длительных операций. |
| package | API для работы с пакетами, предоставляющее информацию о доступных пакетах в репозиториях. |
| packageset | Методы для работы с наборами пакетов, позволяющие управлять группами пакетов и их конфигурациями. |
| acl | API, связанное с системой контроля доступа (ACL), управляет правами доступа к различным объектам системы. |
| bug | Раздел для работы с ошибками, позволяет запрашивать и отправлять информацию об ошибках, связанных с пакетами. |
| dependencies | API для работы с зависимостями пакетов, которое |

|  |  |
| --- | --- |
|  | помогает отслеживать, какие зависимости необходимы для установки или работы пакетов. |
| file | Методы для работы с файлами пакетов, например, для получения информации о конкретных файлах, входящих в состав пакетов. |
| image | API для работы с образами, например, с дисковыми образами или другими бинарными объектами, доступными в системе. |
| export | Раздел, отвечающий за экспорт данных из системы. Позволяет выгружать информацию о пакетах, зависимостях и других сущностях в различных форматах. |
| errata | API для получения информации об исправлениях и обновлениях, предоставляющее данные о важных исправлениях и изменениях в пакетах. |
| vuln | Раздел для управления уязвимостями, предоставляющий информацию о найденных уязвимостях в пакетах или системах. |
| license | API для работы с лицензиями, которое позволяет запрашивать информацию о лицензиях на используемые пакеты. |
| site | Общие методы для работы с веб-сайтом или интерфейсом системы, возможно, включающие инструменты для управления внешним интерфейсом или настройками системы. |

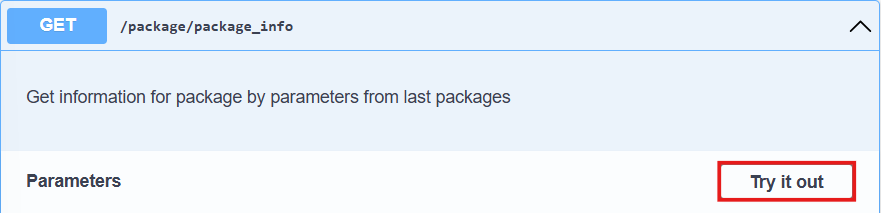
Запросим информацию о пакете *git*. Git — система контроля версий, используемая для управления проектами и совместной работы.

Ссылка на скачивание пакета: <https://packages.altlinux.org/en/sisyphus/srpms/git/>

В ALTRepo API перейдем в раздел *package* и выберем пункт

*GET: /package/package\_info*.

Нажмем на кнопку “Try it out”:



Заполним поля *package name* и *package version* и нажмем кнопку “Execute”:

Результат поиска включает в себя HTTP-запрос к API с ответом в формате JSON:

curl -X 'GET' \ '[https://rdb.altlinux.org/api/package/package\_info?name=git&version=2.42.2&arch=](https://rdb.altlinux.org/api/package/package_info?name=git&version=2.42.2&arch=x86_64&source=false&full=false) [x86\_64&source=false&full=false](https://rdb.altlinux.org/api/package/package_info?name=git&version=2.42.2&arch=x86_64&source=false&full=false)' -H 'accept: application/json'

Структура ответа:

1. Request body (тело запроса)

В разделе *request\_args* указаны аргументы, с которыми был выполнен запрос. В данном случае:

name: "git" — название пакета, который запрашивался; version: "2.42.2" — версия пакета;

arch: "x86\_64" — архитектура пакета.

Другие параметры, такие как release, source, branch, packager\_email и т.д., не были указаны в запросе (имеют значение null или false).

1. Response body (тело ответа)

length: 5 — число найденных пакетов (в данном случае 5 версий пакета). В массиве *packages* содержатся подробные сведения о каждом пакете.

1. Response headers (заголовки ответа)

alt-svc — параметры альтернативного сервиса. content-length: 6263 — длина тела ответа в байтах.

content-type: application/json — тип возвращаемого контента. date — дата и время ответа сервера.

server: nginx — сервер, на котором выполняется запрос.

x-cache-status: MISS — кэширование на стороне сервера не было выполнено.

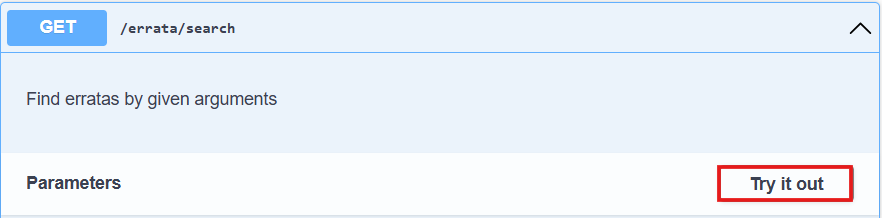
1. Request duration (время выполнения запроса)

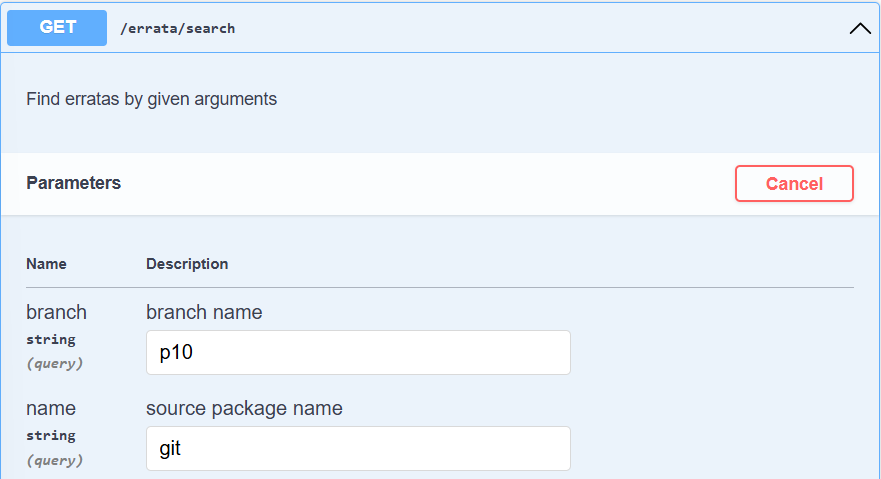
1790 ms — время, потраченное сервером на выполнение запроса.

Теперь запросим информацию об об исправлениях и обновлениях пакета *git*.

В ALTRepo API перейдем в раздел *errata* и выберем пункт GET: /errata/search.

Нажмем на кнопку “Try it out”:



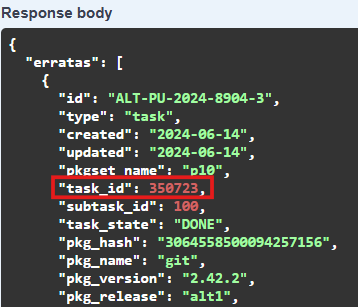
Заполним поля *branch name* и *source package version* и нажмем кнопку “Execute”:

Результат поиска включает в себя HTTP-запрос к API с ответом в формате JSON:

curl -X 'GET' \ '<https://rdb.altlinux.org/api/errata/search?branch=p10&name=git>' \

-H 'accept: application/json'

В теле ответа найдем идентификатор последней задачи:

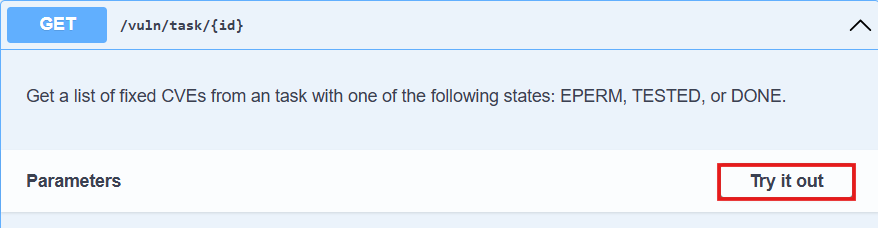


task\_id = 350723

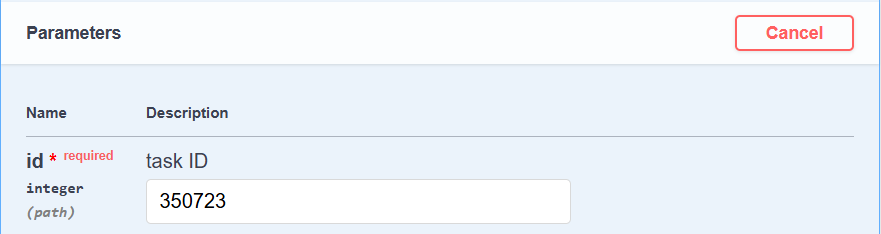
Используя данный идентификатор, запросим информацию об уязвимостях в пакете *git*.

В ALTRepo API перейдем в раздел *vuln* и выберем пункт GET: /vuln/task/{id}.

Нажмем на кнопку “Try it out”:



Заполним поле *task ID* и нажмем кнопку “Execute”:

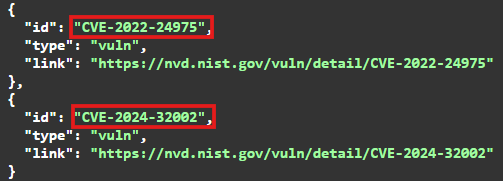


Результат поиска включает в себя HTTP-запрос к API с ответом в формате JSON:

curl -X 'GET' \

'<https://rdb.altlinux.org/api/vuln/task/350723>' -H 'accept: application/json'

В теле ответа найдем идентификаторы уязвимостей, представленные в формате CVE (Common Vulnerabilities and Exposures):

CVE-2022-24975 CVE-2024-32002