Тут титульник

Введение

В современном информационном обществе безопасность веб-серверов является одним из важнейших аспектов информационной безопасности. Веб-серверы играют ключевую роль в предоставлении доступа к веб-ресурсам и обработке пользовательских запросов. В этом контексте особое внимание уделяется защите веб-серверов от различных атак и уязвимостей.

Одним из самых популярных веб-серверов на сегодняшний день является Nginx. Nginx («Engine X») представляет собой легкий, высокопроизводительный сервер, который широко используется для обслуживания веб-сайтов, проксирования и балансировки нагрузки. Он также предлагает множество возможностей для обеспечения безопасности веб-сервера и защиты от различных видов атак.

Целью данной курсовой работы является исследование и анализ мер безопасности, которые могут быть применены для защиты веб-сервера Nginx. Мы рассмотрим основные уязвимости и атаки, с которыми может столкнуться веб-сервер, а также изучим различные методы и техники, которые могут быть использованы для повышения безопасности сервера Nginx.

В ходе исследования будут рассмотрены следующие аспекты защиты веб-сервера Nginx:

- Конфигурация сервера: Мы изучим важные параметры и настройки, которые могут повлиять на безопасность сервера, такие как ограничение доступа к файлам и директориям, использование SSL-сертификатов и применение правил файервола.
- Защита от DDoS-атак: Рассмотрим различные методы обнаружения и предотвращения распределенных атак отказа в обслуживании (DDoS), которые могут оказаться вредными для сервера и доступности веб-сайта.
- Фильтрация трафика: Изучим возможности фильтрации и обработки трафика, включая использование списка контроля доступа (ACL), блокировку IP-адресов, применение белых и черных списков.
- Мониторинг и регистрация: Рассмотрим важность непрерывного мониторинга и регистрации событий сервера для обнаружения подозрительной активности и быстрого реагирования на потенциальные угрозы.

В заключении работы будут представлены рекомендации и практические рекомендации по улучшению безопасности веб-сервера Nginx на основе полученных результатов и анализа. Безопасность веб-серверов является важной задачей для всех организаций, которые предоставляют свои услуги в сети интернет, и эта курсовая работа представляет собой ценный вклад в области информационной безопасности.

Почему необходимо следить за защитой веб-сервера

В последние годы количество атак на веб-серверы значительно выросло. С развитием технологий и все большей зависимости от онлайн-сервисов и электронной коммерции, веб-серверы стали привлекательной целью для злоумышленников. Некоторые факторы, способствующие увеличению количества атак на веб-серверы, включают:

- Распространение злоумышленников: С возрастанием числа злоумышленников, умеющих проводить атаки на веб-серверы, риск стал более значимым. Легко доступные инструменты и ресурсы в Интернете позволяют даже неопытным злоумышленникам осуществлять атаки на веб-серверы.
- Уязвимости веб-приложений: Часто веб-приложения содержат уязвимости, которые могут быть использованы злоумышленниками для атак на сервер. Отсутствие должной обработки и валидации входных данных, уязвимости в коде приложения или ошибки в конфигурации сервера могут предоставить злоумышленникам доступ к серверу.
- Развитие новых типов атак: Злоумышленники постоянно разрабатывают новые методы и техники для атак на веб-серверы. Это включает в себя более сложные DDoS-атаки, передовые методы SQL-инъекций, улучшенные техники фишинга и другие инновационные способы эксплуатации уязвимостей.
- Коммерческая ценность данных: Веб-серверы часто содержат ценную информацию, такую как финансовые данные, персональные данные пользователей, коммерческие секреты и другие конфиденциальные сведения. Захват или кража таких данных может принести значительную финансовую прибыль злоумышленникам, стимулируя их к проведению атак.
- Распространение ботнетов: Ботнеты, состоящие из множества зараженных компьютеров или устройств, используются для проведения массовых атак на веб-серверы. Эти ботнеты контролируются злоумышленниками и могут использоваться для запуска DDoS-атак или для эксплуатации уязвимостей на сервере.

Вот примеры нескольких атак, произведенных на крупные компании за последние годы:

- Атака на Sony PlayStation Network (2011): В 2011 году Sony PlayStation Network столкнулся с одной из наиболее серьезных атак на веб-инфраструктуру. Злоумышленники скомпрометировали сетевую инфраструктуру, что привело к утечке личных данных более 77 миллионов пользователей, включая имена, адреса электронной почты, пароли и финансовую информацию.
- Атака на Equifax (2017): В 2017 году компания Equifax, одна из трех крупнейших кредитных бюро в США, столкнулась с серьезной атакой. Злоумышленники эксплуатировали уязвимость веб-приложения, что позволило им получить доступ к личным данным около 147 миллионов человек, включая имена, социальные номера, даты рождения и номера кредитных карт.
- Атака на Yahoo (2013-2014): В 2013-2014 годах Yahoo столкнулся с масштабной атакой, в результате которой были скомпрометированы данные более 3 миллиардов пользователей. Злоумышленники получили доступ к учетным записям, включая имена, адреса электронной почты, хэшированные пароли и секретные вопросы безопасности.

Это всего лишь некоторые примеры успешных атак на веб-инфраструктуру, и с течением времени появляются новые методы и уязвимости, которые могут быть эксплуатированы злоумышленниками. Важно постоянно обновлять свои знания о безопасности и применять соответствующие меры для защиты веб-серверов и веб-приложений.

Основные рекомендации к защите веб-серверов

За стандартами защиты веб-серверов следят различные организации и стандартизационные органы, такие как OWASP (Open Web Application Security Project), NIST (National Institute of Standards and Technology) или ENISA (European Union Agency for Cybersecurity), а также сообщества экспертов в области информационной безопасности. Они играют важную роль в разработке и установлении лучших практик и рекомендаций по обеспечению безопасности веб-серверов.Вот основные из них:

- Обновляйте программное обеспечение: Регулярно обновляйте операционную систему, веб-сервер и все установленные компоненты и приложения до последних версий. Обновления часто содержат исправления уязвимостей, которые могут быть использованы злоумышленниками.
- Используйте сильные пароли: Установите сложные пароли для учетных записей администратора и других пользователей, а также для базы данных. Избегайте использования стандартных паролей и регулярно меняйте пароли.
- Применяйте фильтрацию трафика: Используйте межсетевые экраны (firewalls) и системы обнаружения вторжений (IDS/IPS) для фильтрации и контроля входящего и исходящего сетевого трафика. Это поможет блокировать подозрительные пакеты данных и защитить сервер от многих типов атак.
- Защитите от DDoS-атак: Реализуйте механизмы для обнаружения и смягчения DDoS-атак, такие как использование услуги облачной защиты от DDoS или настройка сетевых устройств для отсеивания вредоносного трафика.
- Применяйте принцип наименьших привилегий: Ограничьте привилегии учетных записей, чтобы минимизировать потенциальные последствия компрометации. Пользовательские учетные записи должны иметь только необходимые права доступа к ресурсам.
- Фильтруйте входящие данные: Валидируйте и санитизируйте входящие данные, особенно если они передаются в базу данных или выполняются на сервере. Это поможет предотвратить SQL-инъекции и XSS-атаки.
- Шифруйте соединение: Используйте протокол HTTPS с помощью сертификатов SSL/TLS для защиты передачи данных между клиентом и сервером. Это поможет предотвратить перехват информации и подделку данных.
- Регулярно создавайте резервные копии: Регулярное резервное копирование данных позволяет восстановить сервер в случае успешной атаки или сбоя. Убедитесь, что резервные копии хранятся в надежном месте, отдельно от основного сервера.
- Мониторинг и журналирование: Внедрите системы мониторинга, которые следят за активностью сервера и обнаруживают подозрительные или необычные события. Хороший журнал событий поможет вам исследовать инциденты и принять меры по предотвращению будущих атак.

Это лишь некоторые рекомендации, и полная защита веб-сервера требует комплексного подхода, учета специфических потребностей вашего сервера и постоянного обновления знаний о безопасности.

О выборе веб-сервера

Правильный выбор веб-сервера обеспечивает оптимальное функционирование веб-приложения, удовлетворяя требованиям бизнеса и ожиданиям пользователей. Он позволяет эффективно обрабатывать запросы, предоставлять контент быстро и без проблем, а также адаптироваться к растущим потребностям и нагрузке.

Кроме того, правильный выбор веб-сервера имеет прямое отношение к безопасности вашего веб-приложения. Надежный веб-сервер обеспечивает защиту от различных угроз и атак, минимизируя риски утечки данных или компрометации системы.

В своей работе я буду использовать Nginx (Engine-X) — это мощный веб-сервер и прокси-сервер, который выполняет ряд функций и может быть использован в различных сценариях. Вот некоторые основные области применения Nginx:

- Веб-сервер: Одной из основных функций Nginx является обслуживание веб-содержимого. Он способен обрабатывать статические файлы, такие как HTML, CSS, JavaScript и изображения. Благодаря своей высокой производительности и эффективному использованию ресурсов, Nginx позволяет эффективно обслуживать большое количество запросов, особенно в высоконагруженных средах.
- Обратный прокси: Nginx часто используется в качестве обратного прокси-сервера, который принимает запросы от клиентов и перенаправляет их на соответствующие веб-серверы. Это позволяет балансировать нагрузку между несколькими серверами и повышает отказоустойчивость, так как приложения на серверах могут быть легко масштабируемы.
- Балансировка нагрузки: Nginx предоставляет возможности балансировки нагрузки, которые позволяют распределять запросы равномерно между несколькими серверами. Это помогает оптимизировать использование ресурсов и обеспечивает более высокую доступность веб-приложений.
- Кэширование: Nginx поддерживает функцию кэширования, которая позволяет сохранять статические ресурсы, такие как изображения или файлы CSS/JavaScript, в оперативной памяти или на диске. Это сокращает нагрузку на сервер и ускоряет время загрузки страниц для повторных запросов.
- SSL/TLS терминирование: Nginx может выполнять функцию терминирования SSL/TLS, что позволяет осуществлять шифрование и расшифровку данных между клиентом и сервером. Это обеспечивает безопасную передачу данных и защиту от перехвата информации.
- Проксирование API: Nginx может быть использован для проксирования запросов к внутренним или внешним API. Это позволяет контролировать доступ, управлять авторизацией и маршрутизацией запросов к API.
- Управление статическими файлами и медиа-контентом: Nginx может использоваться для эффективного доставки статических файлов и медиа-контента, таких как видео или аудиофайлы. Это позволяет обеспечить быструю и надежную доставку контента конечным пользователям.

Сочетание высокой производительности, гибкости и богатого функционала делает Nginx популярным выбором для веб-серверов, обратных прокси и балансировщиков нагрузки во многих веб-приложениях и средах разработки.

Далее будут представлены основные настройки для nginx сервера, применяемые для защиты сервера.

Основные настройки Nginx

О структуре файла конфигурации

Файл конфигурации Nginx, известный как nginx.conf, определяет основные настройки и параметры работы веб-сервера Nginx. Вот общая структура файла конфигурации nginx.conf:

- Директивы глобального блока (http): В этом блоке определяются глобальные настройки для всего веб-сервера. Включает в себя директивы, такие как user, worker_processes, events и другие, которые задают общие параметры работы сервера.
- Блок events: Здесь определяются параметры событийной модели, такие как количество рабочих процессов (worker_processes), метод обработки событий и другие настройки, связанные с обработкой событий сервером.
- Блок http: В этом блоке определяются основные параметры и настройки HTTP-протокола. Он включает в себя блок server, который может быть повторен несколько раз для определения разных виртуальных хостов и их настроек.
- Блок server: Каждый блок server определяет настройки для конкретного виртуального хоста или сервера. В этом блоке определяются параметры, такие как listen, server_name, location и другие, которые управляют поведением сервера для конкретного хоста.
- Блок location: Блок location определяет настройки для обработки запросов, соответствующих определенному пути URL. Здесь можно определить параметры, такие как root, proxy_pass, rewrite и другие, чтобы настроить обработку запросов для конкретного пути.
- Другие блоки и директивы: В файле конфигурации nginx.conf могут быть определены и другие блоки и директивы для специфических настроек и модулей, таких как SSL/TLS, кэширование, сжатие и другие.

Структура файла конфигурации nginx.conf может различаться в зависимости от конкретных потребностей и настроек веб-сервера. Рекомендуется внимательно изучить документацию Nginx и следовать принятой структуре и синтаксису для корректной настройки сервера.

Сетевые ограничения

Эти настройки позволяют вам определить различные параметры работы сервера Nginx, такие как порты, доступ по IP-адресам, размеры буферов, таймауты и другие параметры. Изменение этих настроек позволяет адаптировать сервер к конкретным требованиям приложения и повысить его производительность и безопасность.

```
http{

    #limit concurrency
    limit_conn_zone $server_name zone=per_vhost:5m;
    limit_conn_zone $binary_remote_addr zone=per_ip:5m;


     server{
```

```
        listen       81;


        deny   192.168.0.170;
        deny   192.168.0.171;
        allow   192.168.0.174;

        server_name  localhost;

        #buffer sizes
        client_body_buffer_size 16k;
        client_header_buffer_size 1k;
        client_max_body_size 8m;
        large_client_header_buffers 2 1k;

        #timeouts
        client_body_timeout 12;
        client_header_timeout 12;

        #keepalive
        keepalive_timeout 65;
        send_timeout 10;

        #server version info off
        server_tokens off;
        ...
        location  ~* \.(css|js|jpg|png|gif)$ {
              ...
              limit_conn per_ip 1;
              ...
        }
    }
}
```

Рассмотрим каждую из указанных настроек в контексте Nginx:

- **limit_conn_zone $server_name zone=per_vhost:5m** и **limit_conn_zone $binary_remote_addr zone=per_ip:5m**: Эти директивы определяют ограничения на количество одновременных подключений для каждого виртуального хоста (per_vhost) и для каждого IP-адреса клиента (per_ip). Здесь указаны размеры зон памяти (5m), которые используются для отслеживания подключений.
- **listen 81**: Эта директива указывает, что сервер Nginx будет слушать входящие подключения на порту 81.
- **deny 192.168.0.170, deny 192.168.0.171** и **allow 192.168.0.174**: Эти директивы управляют доступом к серверу на основе IP-адреса клиента. Клиенты с IP-адресами 192.168.0.170 и 192.168.0.171 будут запрещены, а клиент с IP-адресом 192.168.0.174 будет разрешен.
- **server_name localhost**: Эта директива определяет имя сервера, к которому применяются настройки в данном блоке конфигурации. В данном случае, сервер будет отвечать на запросы с хостом "localhost".
- **client_body_buffer_size 16k**, **client_header_buffer_size 1k, client_max_body_size 8m, large_client_header_buffers 2 1k**: Эти настройки связаны с размерами буферов, используемых для обработки тела запроса клиента (client_body_buffer_size), заголовков клиента (client_header_buffer_size), максимального размера тела запроса клиента (client_max_body_size) и больших буферов для заголовков клиента (large_client_header_buffers).

- **client_body_timeout 12**, **client_header_timeout 12**: Эти настройки определяют таймауты ожидания запроса от клиента для тела (client_body_timeout) и заголовков (client_header_timeout).
- **keepalive_timeout 65**, **send_timeout 10**: Эти настройки определяют таймауты соединения keep-alive (keepalive_timeout) и отправки данных на сервер (send_timeout).
- **server_tokens off**: Эта директива отключает отправку информации о версии сервера в заголовках ответа, чтобы уменьшить возможность идентификации сервера в случае потенциальных атак.
- **location ~* \.(css|js|jpg|png|gif)$ { ... }**: Это блок конфигурации для обработки запросов к статическим файлам с расширениями .css, .js, .jpg, .png и .gif. В данном случае, внутри блока могут быть указаны дополнительные настройки, например, limit_conn per_ip 1;, которая ограничивает количество одновременных подключений с одного IP-адреса до 1.

Кэширование запросов

Кэширование запросов в Nginx - это процесс сохранения результатов запросов на сервере, чтобы при последующих запросах на тот же ресурс сервер мог возвращать результаты из кэша, без необходимости выполнения полной обработки запроса.

При использовании кэширования, Nginx может значительно сократить нагрузку на сервер, улучшить скорость ответа и снизить задержки для конечных пользователей. Когда клиент отправляет запрос, Nginx проверяет наличие соответствующей записи в кэше. Если запись присутствует и не устарела, сервер может немедленно вернуть результат клиенту без обращения к бэкенд-серверу.

```
http{

    include         fastcgi_params;
    include         fastcgi.conf;

    #fastCGI
    fastcgi_cache_path /etc/nginx/cache levels=1:2 keys_zone=microcache:10m max_size=500m inactive=10m;
    fastcgi_cache_key "$scheme$request_method$host$request_uri";
    fastcgi_ignore_headers Cache-Control Expires Set-Cookie ;
    add_header caching $upstream_cache_status;

    server{
        ...
        location ~* \.(css|js|jpg|png|gif)$ {
            ...
            expires 1M;
            ...
        }
        ...
        location /testphp {
            fastcgi_cache microcache;
            fastcgi_cache_valid 200 60m;
            fastcgi_pass 127.0.0.1:9000;
            ...
        }
    }
}
```

В приведенном примере конфигурации Nginx, рассмотрим несколько соответствующих настроек для кэширования:

- **fastcgi_cache_path /etc/nginx/cache levels=1:2 keys_zone=microcache:10m max_size=500m inactive=10m**: Эта настройка определяет путь к директории, где будут храниться кэшированные данные (/etc/nginx/cache), уровни директорий (levels=1:2), зону ключей (keys_zone=microcache:10m), максимальный размер кэша (max_size=500m) и время неактивности после которого записи в кэше считаются устаревшими (inactive=10m).
- **fastcgi_cache_key "$scheme$request_method$host$request_uri"**: Эта директива определяет ключ, по которому происходит кэширование запросов FastCGI. Ключ формируется на основе схемы ($scheme), метода запроса ($request_method), хоста ($host) и URI запроса ($request_uri).
- **fastcgi_cache microcache**: Эта директива указывает, что запросы, соответствующие данной локации, должны быть кэшированы в зоне ключей microcache.
- **fastcgi_cache_valid 200 60m**: Эта директива определяет время жизни кэшированной записи с кодом ответа 200 (успешный ответ) в течение 60 минут.

Таким образом, с помощью этих настроек Nginx может кэшировать ответы FastCGI и возвращать их непосредственно из кэша, минуя обращение к бэкенд-серверу, если записи в кэше существуют и не устарели. Это позволяет снизить нагрузку на сервер и сократить время обработки запросов, повышая производительность и улучшая отзывчивость веб-приложения.

Шифрование запросов

Шифрование и SSL (Secure Sockets Layer) являются важными аспектами безопасности веб-серверов. Nginx предоставляет возможность использовать SSL/TLS для шифрования соединения между клиентом и сервером.

SSL/TLS - это протоколы, обеспечивающие шифрование данных и аутентификацию для безопасной передачи информации по сети. Они используют криптографические алгоритмы для защиты конфиденциальности, целостности и подлинности данных.

```
http{
   server{
      listen       443 ssl;


      #ssl
      ssl_certificate /etc/nginx/ssl/nginx.crt;
      ssl_certificate_key /etc/nginx/ssl/nginx.key;
      ssl_session_cache    shared:SSL:1m;
      ssl_session_timeout
      ssl_ciphers  HIGH:!aNULL:!MD5;
      ssl_prefer_server_ciphers  on;
   }
}
```

В данном примере конфигурации Nginx демонстрируется использование шифрования запросов с помощью протокола SSL/TLS. Давайте рассмотрим каждую из указанных настроек:

- l**isten 443 ssl;**: Эта директива указывает на прослушивание порта 443 (стандартный порт для HTTPS) и использование протокола SSL/TLS для шифрования соединения между клиентом и сервером.
- **ssl_certificate /etc/nginx/ssl/nginx.crt;** и **ssl_certificate_key /etc/nginx/ssl/nginx.key;**: Эти директивы указывают пути к сертификату и приватному ключу, необходимым для установки безопасного соединения. В данном случае, указываются пути к файлам сертификата (nginx.crt) и приватного ключа (nginx.key).
- **ssl_session_cache shared:SSL:1m;** и **ssl_session_timeout**: Эти директивы определяют настройки кэша сеансов SSL/TLS. shared:SSL:1m указывает, что кэш сеансов должен быть разделен между несколькими воркерами и иметь размер 1 мегабайт. ssl_session_timeout 5m; определяет время жизни сеансов в кэше.
- **ssl_ciphers HIGH:!aNULL:!MD5;**: Эта директива определяет список шифров, которые могут быть использованы при установке SSL/TLS-соединения. В данном случае, используются шифры с высоким уровнем безопасности и отключены анонимные шифры и шифры, использующие алгоритм MD5.
- **ssl_prefer_server_ciphers on;**: Эта директива указывает серверу предпочитать шифры, предложенные клиентом, при установке SSL/TLS-соединения.

Эти настройки позволяют использовать SSL/TLS для шифрования запросов между клиентом и сервером, обеспечивая безопасность и защиту передаваемых данных от перехвата или нежелательного доступа.

Сжатие ответов сервера

Использование сжатия респонзов с помощью gzip позволяет уменьшить размер передаваемых данных между сервером и клиентом, что улучшает скорость загрузки страницы и экономит пропускную способность сети. Это особенно полезно при передаче больших текстовых файлов, стилей CSS и JavaScript, которые часто могут быть сжаты существенно без потери качества.

```
http{
   server {
      #gzip
      gzip on;
      gzip_min_length 100;
      gzip_comp_level 3;


      gzip_types text/plain;
      gzip_types text/css;
      gzip_types text/javascript;


      gzip_disable "msie6";
   }
}
```

В данном примере конфигурации Nginx демонстрируется использование сжатия респонзов с помощью gzip. Давайте рассмотрим каждую из указанных настроек:

- **gzip on;**: Эта директива включает сжатие респонзов с помощью gzip.
- **gzip_min_length 100;**: Эта директива указывает минимальный размер ответа, который будет сжиматься. В данном случае, ответы, размером менее 100 байт, не будут сжиматься.
- **gzip_comp_level 3;**: Эта директива задает уровень компрессии для сжатия gzip. Уровень 3 является стандартным и обеспечивает хороший баланс между скоростью и степенью сжатия.
- **gzip_types text/plain; gzip_types text/css; gzip_types text/javascript;**: Эти директивы указывают типы контента, которые могут быть сжаты с помощью gzip. В данном случае, текстовые файлы (text/plain), CSS-файлы (text/css) и JavaScript-файлы (text/javascript) будут сжиматься.
- **gzip_disable "msie6";**: Эта директива позволяет отключить сжатие для устаревших браузеров, в данном случае для Internet Explorer 6. Такие браузеры не всегда могут правильно обрабатывать сжатые респонзы, поэтому можно отключить сжатие для них.

Базовая аутентификация

Базовая аутентификация (Basic Authentication) - это простой механизм аутентификации, который использует комбинацию имени пользователя и пароля для ограничения доступа к ресурсам сервера. В Nginx базовая аутентификация может быть легко настроена с использованием модуля ngx_http_auth_basic.

```
http{


   server{
      location /profile {
         auth_basic "Restricted folder";


         #base auth
         auth_basic_user_file /etc/nginx/creds/.htpasswd;
         access_log /var/log/nginx/new.log upstream_time;
      }
   }
}
```

В данной конфигурации Nginx используется базовая аутентификация для ограничения доступа к пути /profile. Давайте рассмотрим каждую из указанных настроек:

- **location /profile { ... }**: Директива location определяет путь к ресурсу, для которого будет применяться базовая аутентификация. В данном случае, это путь /profile.
- a**uth_basic "Restricted folder";**: Эта директива устанавливает сообщение, которое будет отображаться в диалоговом окне аутентификации браузера, при попытке доступа к ограниченной папке /profile. В данном случае, сообщение будет "Restricted folder".
- **auth_basic_user_file /etc/nginx/creds/.htpasswd;**: Эта директива указывает путь к файлу .htpasswd, который содержит пары "имя пользователя:зашифрованный пароль". В данном случае, файл .htpasswd находится по пути

/etc/nginx/creds/.htpasswd. Этот файл должен быть создан и содержать допустимые учетные данные для аутентификации пользователей.

- **access_log /var/log/nginx/new.log upstream_time;**: Эта директива настраивает запись журнала доступа в файл /var/log/nginx/new.log с указанием времени выполнения каждого запроса (upstream_time). Журнал доступа содержит информацию о запросах к ресурсу /profile.

Таким образом, при доступе к пути /profile, браузер будет запрашивать у пользователя имя пользователя и пароль. Введенные учетные данные будут проверяться на соответствие с данными из файла .htpasswd. Если аутентификация прошла успешно, пользователю будет разрешен доступ к ограниченной папке /profile, а информация о запросах будет записываться в указанный журнал доступа.

Эта конфигурация позволяет ограничить доступ к конкретной папке на сервере с использованием базовой аутентификации, что обеспечивает дополнительный уровень защиты и контроля доступа к конфиденциальной информации.

Ограничения по геоданным

Nginx GeoIP2 - это модуль, который позволяет использовать информацию о географическом расположении клиентов веб-сервера на основе их IP-адресов. Этот модуль использует базу данных GeoIP2, которая содержит информацию о стране, регионе, городе, координатах и других атрибутах, связанных с конкретными IP-адресами. Для работы с данным модулем необходимо загрузить в конфигурацию сервера модуль ngx_http_geoip2, а так же предоставить серверу доступ к базе геоданных, в нашем случае использовалась GeoLite2-City.mmdb.

```
http{


  #geoip
  geoip2 /usr/share/GeoIP/GeoLite2-City.mmdb {
    auto_reload 60m;
    $geoip2_metadata_city_build metadata build_epoch;
    $geoip2_data_country_name country names en;
    $geoip2_data_country_code country iso_code;
    $geoip2_data_city_name city names en;
    $geoip2_data_region_name subdivisions 0 names en;
    $geoip2_data_state_code  subdivisions 0 iso_code;
  }
  geoip blocking
  map "$geoip2_data_country_code:$geoip2_data_state_code" $allowed_reg {
     default no;
     ~^RU: yes; #Россия
     ~^BY: yes; #Белоруссия
     ~^AM: yes; #Армения
     ~^KZ: yes; #Казахстан
     ~^KG: yes; #Кыргызстан
     UA:40 yes; #Севастополь
     UA:43 yes; #Крым
     UA:14 yes; #Донецкая область
     UA:09 yes; #Луганская область
     UA:23 yes; #Запорожская область
     UA:65 yes; #Херсонская область
     GE:AB yes; #Абхазия
  }
  server{
```

```
    }
}
```

Давайте рассмотрим каждую из указанных настроек:

- **geoip2 /usr/share/GeoIP/GeoLite2-City.mmdb { ... }**: Директива geoip2 указывает путь к файлу базы данных GeoIP2, который содержит информацию о геолокации IP-адресов. В данном случае, файл находится по пути /usr/share/GeoIP/GeoLite2-City.mmdb. Дополнительно указаны переменные, которые будут содержать информацию о стране, регионе, городе и других атрибутах.
- **auto_reload 60m;**: Эта настройка указывает интервал автоматической перезагрузки базы данных GeoIP2. В данном случае, база данных будет перезагружаться каждые 60 минут.
- **$geoip2_metadata_city_build metadata build_epoch;**: Эта переменная содержит информацию о времени последнего обновления базы данных GeoIP2.
- **$geoip2_data_country_name country names en;**: Эта переменная содержит название страны на основе IP-адреса клиента.
- **$geoip2_data_country_code country iso_code;**: Эта переменная содержит код страны (двухбуквенный ISO-код) на основе IP-адреса клиента.
- **$geoip2_data_city_name city names en;**: Эта переменная содержит название города на основе IP-адреса клиента.
- **$geoip2_data_region_name subdivisions 0 names en;**: Эта переменная содержит название региона на основе IP-адреса клиента.
- **$geoip2_data_state_code subdivisions 0 iso_code;**: Эта переменная содержит код региона (двухбуквенный ISO-код) на основе IP-адреса клиента.
- **geoip blocking;**: Эта директива включает блокировку доступа на основе географической локации. Если клиент находится в запрещенном регионе, его доступ будет заблокирован.
- **map "$geoip2_data_country_code:$geoip2_data_state_code" $allowed_reg { ... }**: Эта директива определяет переменную $allowed_reg, которая будет содержать значение yes или no в зависимости от географической локации клиента. В данном случае, используется регулярное выражение для определения разрешенных регионов. Если соответствующий регион найден в базе данных GeoIP2, переменная будет иметь значение yes, в противном случае - no.
- **if ($allowed_reg = no) { return 444; }**: Эта конструкция проверяет значение переменной $allowed_reg. Если значение равно no, то выполняется директива return 444, которая прекращает обработку запроса и возвращает ошибку 444 ("No Response") клиенту. Это позволяет блокировать доступ к серверу для клиентов из запрещенных регионов.

Обратите внимание, что данная конфигурация может быть дополнена другими настройками сервера внутри блока server { ... }.

Балансировка нагрузки

Nginx предоставляет несколько алгоритмов балансировки нагрузки, которые определяют способ распределения входящих запросов между серверами в группе. Каждый алгоритм имеет свои особенности и может быть выбран в зависимости от требований вашей

системы. Ниже приведены основные алгоритмы балансировки нагрузки, поддерживаемые Nginx:

- **Round Robin** (Поочередный выбор):
  - Алгоритм по умолчанию.
  - Запросы распределяются по серверам в группе в порядке их указания.
  - При каждом новом запросе выбирается следующий сервер в порядке списка.
  - Простой и равномерный способ распределения нагрузки.
- **Least Connections** (Выбор сервера с наименьшим количеством активных соединений):
  - Запросы направляются на сервер с наименьшим количеством активных соединений.
  - Позволяет распределить нагрузку более равномерно, учитывая текущую нагруженность серверов.
- **IP Hash** (Хеширование по IP-адресу):
  - Каждый клиентский IP-адрес сопоставляется с конкретным сервером.
  - Позволяет обеспечить сохранение состояния сессии для клиента на протяжении всего времени взаимодействия с сервером.
  - Гарантирует, что все запросы от одного клиента будут направлены на один и тот же сервер.
- **Generic Hash** (Общее хеширование):
  - Запросы хешируются с использованием произвольного ключа, указанного в конфигурации.
  - Позволяет гибко настраивать способ хеширования для распределения нагрузки на основе определенных параметров запроса или других данных.

```
http{
   upstream php_servers {
         #hash $scheme$request_uri;
         #ip_hash;
         least_conn;
         #random two least_conn;
         #least_time header;
         #least_time last_byte;
         server localhost:10001;
         server localhost:10002;
         server localhost:10003;
      }
   server{
      listen       81;
      listen       443 ssl;



      # proxy the PHP scripts to Apache listening on 127.0.0.1:9000
      # testing load balancers


      location /testphp {
         fastcgi_cache microcache;
         fastcgi_cache_valid 200 60m;
         fastcgi_pass 127.0.0.1:9000;
```

```
        proxy_set_header proxy_header_to_server nginx;
        add_header proxy_header nginx;
        proxy_pass http://php_servers;
    }
  }
}
```

В данном примере конфигурации Nginx используется алгоритм балансировки нагрузки "Least Connections" для группы серверов php_servers. Давайте рассмотрим, что делает каждая настройка:

- **least_conn**: Этот алгоритм направляет запросы на сервер с наименьшим количеством активных соединений. Он распределяет нагрузку равномерно между серверами, учитывая их текущую нагруженность. Серверы localhost:10001, localhost:10002 и localhost:10003 указаны в качестве серверов для балансировки.
- **listen 81; и listen 443 ssl;**: Эти директивы указывают Nginx слушать соединения на портах 81 и 443 с использованием SSL/TLS.
- **location /testphp**: В этом блоке настраивается обработка запросов, которые соответствуют пути /testphp. Они проксируются на серверы из группы php_servers с использованием балансировки нагрузки. Также присутствуют дополнительные настройки, такие как кеширование (fastcgi_cache) и передача заголовков (proxy_set_header, add_header).

В данной конфигурации алгоритм балансировки нагрузки "Least Connections" выбран для равномерного распределения запросов между серверами localhost:10001, localhost:10002 и localhost:10003. Это позволяет достичь более эффективного использования ресурсов и более высокой отказоустойчивости системы.

Логирование

Nginx логирование играет важную роль, позволяя отслеживать и анализировать различные события, ошибки и активность сервера. Nginx предлагает различные типы логов, которые можно настроить в файле конфигурации nginx.conf. Давайте рассмотрим основные типы логов и их назначение:

- **Access logs** (логи доступа): Они записывают информацию о каждом запросе, поступающем на сервер Nginx. Access logs содержат информацию, такую как IP-адрес клиента, время запроса, HTTP-метод, запрошенный URL, код состояния ответа и объем переданных данных. Эти логи полезны для мониторинга активности сервера и анализа трафика.
- **Error logs** (логи ошибок): Они регистрируют различные ошибки, возникающие в процессе обработки запросов. Error logs включают сообщения об ошибках, критические события и предупреждения, связанные с работой сервера. Эти логи помогают в выявлении проблем и их диагностике для обеспечения стабильности и безопасности сервера.
- **Application logs** (логи приложений): Если вы используете Nginx в качестве прокси или обратного прокси для приложений, таких как веб-серверы или приложения на основе фреймворка, то вы можете настроить логирование событий, связанных с вашими приложениями. Это позволяет вам отслеживать действия и проблемы, возникающие в приложениях.

Для каждого из этих типов логов в Nginx можно настроить формат записей, место хранения файлов логов и уровень подробности записываемых сообщений. Это позволяет администраторам настроить логирование согласно своим требованиям и предпочтениям.

```
http{
   #log formatting
   log_format upstream_time '$remote_addr - $remote_user [$time_local] '
                            '"$request" $status $body_bytes_sent '
                            '"$http_referer" "$http_user_agent"'
                            'rt=$request_time uct="$upstream_connect_time" uht="$upstream_header_time"
urt="$upstream_response_time"';


   log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                     '$status $body_bytes_sent "$http_referer" '
                     '"$http_user_agent" "$http_x_forwarded_for"';


   #access_log  logs/access.log  main;
   server{


      #logs off
      error_log off;
      access_log off;
      #access_log  logs/host.access.log  main;




      location /profile {
         error_log /var/log/nginx/errors_profile.log main
         access_log /var/log/nginx/access_profile.log upstream_time;
      }


   }
}
```

В представленной конфигурации Nginx определены два формата логов: upstream_time и main. Давайте разберем, что делают эти настройки:

- **log_format upstream_time**: Этот формат логов определяет пользовательский формат записей для логов доступа. Он содержит следующие переменные и данные:
- **$remote_addr**: IP-адрес клиента
- **$remote_user**: имя пользователя (если используется базовая аутентификация)
- **$time_local**: локальное время запроса
- **"$request"**: сам запрос
- **$status**: код состояния ответа сервера
- **$body_bytes_sent**: размер ответа в байтах
- **"$http_referer"**: HTTP-заголовок Referer (если присутствует)
- **"$http_user_agent"**: HTTP-заголовок User-Agent
- **rt=$request_time**: время обработки запроса
- **uct="$upstream_connect_time"**: время установки соединения с бэкенд-сервером
- **uht="$upstream_header_time"**: время получения заголовков ответа от бэкенд-сервера

- **urt="$upstream_response_time"**: время получения ответа от бэкенд-сервера
- **log_format main**: Этот формат логов также определяет пользовательский формат записей для логов доступа. Он содержит переменные и данные, такие как IP-адрес клиента, имя пользователя (если есть), локальное время запроса, сам запрос, код состояния ответа сервера, размер ответа, HTTP-заголовки Referer и User-Agent, а также заголовок X-Forwarded-For (если присутствует).

Внутри блока server для пути /profile определены настройки логирования:

- **error_log /var/log/nginx/errors_profile.log main**: Указывает путь к файлу, в который будут записываться ошибки, связанные с обработкой запросов для данного пути.
- **access_log /var/log/nginx/access_profile.log upstream_time;**: Указывает путь к файлу, в который будут записываться логи доступа для данного пути, используя формат upstream_time для форматирования записей логов.

Обратите внимание, что логирование в блоке server может быть включено или выключено с помощью директив error_log и access_log. В представленной конфигурации логирование в целом выключено для данного сервера, но включено и настроено для пути /profile.

Настраивая логирование в Nginx, вы можете контролировать формат записей, выбирать, какие данные включать, и указывать файлы, в которые записывать логи. Это помогает в мониторинге и отладке сервера, а также в получении полезной информации о запросах и ошибках.