Яновский В.В. Реализация механизма защиты от REPLAY-атак в HTTP-запросах // Международный журнал информационных технологий и энергоэффективности. – 2025. – Т. 10 № 4(54) с. 47–54

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.4

РЕАЛИЗАЦИЯ МЕХАНИЗМА ЗАЩИТЫ ОТ REPLAY-АТАК В HTTP-ЗАПРОСАХ

Яновский В.В.
ФГБОУ ВО «МИРЭА - РОССИЙСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ», Москва, Россия (119454, г. Москва, Пр-т Вернадского, д. 78, стр.4), e-mail: yanovsky.dev@yandex.ru

Настоящая статья посвящена задаче обеспечения безопасности HTTP-запросов веб-приложений от Replay-атак. Рассмотрен подход, основанный на многоступенчатой проверке временных меток, контрольных сумм и уникальности запросов. Предложена архитектура системы, включающая специализированные middleware-компоненты и распределённый кеш для предотвращения повторного использования запросов. Проведён количественный анализ комбинаторной сложности предложенного метода, подтверждающий его высокую устойчивость к попыткам перебора. Представлены практические рекомендации по выбору оптимального интервала хранения контрольных сумм в распределённом кеше с учётом требований безопасности и производительности

Ключевые слова: Клиент-серверное взаимодействие, HTTP-запросы, информационная безопасность, Replay-атаки, контрольная сумма, промежуточное ПО.

IMPLEMENTATION OF A MECHANISM TO PROTECT AGAINST REPLAY ATTACKS IN HTTP REQUESTS

Yanovskiy V.V.
MIREA - RUSSIAN TECHNOLOGICAL UNIVERSITY, Moscow, Russia (119454, Moscow, avenue. Vernadsky, 78, b. 4), e-mail: yanovsky.dev@yandex.ru

This paper addresses the issue of securing HTTP requests in web applications against replay attacks. An approach based on multi-stage validation of timestamps, checksums, and request uniqueness is proposed. The suggested system architecture incorporates specialized middleware components and a distributed cache to prevent the reuse of requests. A quantitative analysis of the combinatorial complexity of the proposed method confirms its robustness against brute-force attacks. Practical recommendations regarding the optimal duration for storing checksums in a distributed cache are provided, balancing security requirements and system performance.

Keywords: Client-server interaction, HTTP requests, information security, Replay attacks, checksum, middleware.

Введение

В условиях глобальной цифровизации и стремительного роста количества веб-приложений вопросы информационной безопасности приобретают особую значимость. HTTP-запросы, являясь фундаментальным элементом взаимодействия в сети, подвержены различным видам атак, способным привести к существенным финансовым и репутационным потерям. Одной из наиболее распространённых угроз является повторное воспроизведение запросов, что создаёт предпосылки для реализации атак, направленных на подмену данных и осуществление мошеннических операций.

Современные механизмы защиты, включая использование протокола HTTPS и различных методов аутентификации на основе токенов, не всегда обеспечивают необходимый уровень безопасности от атак, направленных на повторное использование или модификацию передаваемых данных. В этой связи актуальной задачей становится разработка и исследование

новых подходов к защите HTTP-запросов и транзакций, обеспечивающих повышение их устойчивости к потенциальным угрозам.

Целью статьи является разработка архитектуры системы защиты HTTP-запросов от Replay-атак на основе многоуровневой проверки временных меток, контрольных сумм и уникальности запросов.

Постановка проблемы

Несмотря на распространённое применение существующих средств защиты, таких как HTTPS и токенов аутентификации, многие веб-приложения остаются уязвимыми к Replay-атакам [1]. Проблема заключается в том, что злоумышленник способен перехватить ранее выполненный HTTP-запрос и осуществить его повторную передачу, тем самым получая возможность совершать противоправные действия от имени легитимного пользователя. Подобные атаки способны повлечь серьёзные последствия, в том числе финансовый ущерб, компрометацию конфиденциальных пользовательских данных и репутационные риски для организаций. В связи с этим возникает необходимость разработки и исследования новых подходов к защите от Replay-атак, обеспечивающих точное распознавание и отклонение повторных запросов, устойчивость к попыткам модификации передаваемых данных и эффективное использование системных ресурсов.

Архитектура системы

Для решения обозначенной проблемы разработана архитектура, направленная на безопасную обработку HTTP-запросов. Предлагаемая архитектура включает взаимодействие клиентской и серверной частей. Серверная часть разделена на несколько отдельных компонентов, каждый из которых выполняет специализированную функцию во время анализа и обработки поступающих запросов.

В процессе обеспечения защиты от Replay-атак задействованы следующие компоненты:

1) клиентская часть. Её функции заключаются в генерации случайного значения, получении актуального времени и вычислении контрольной суммы (checksum), основанной на уникальном алгоритме. Если пользователи имеют доступ к алгоритму генерации контрольной суммы, требуется применить методы обфускации для защиты алгоритма от анализа;

2) служба синхронизации времени (TimeProvider). Данный компонент предоставляет актуальное время, которое синхронизируется между клиентской и серверной частями, обеспечивая согласованность и исключая возможные конфликты во временных метках;

3) серверная часть. Реализована с помощью нескольких middleware-компонентов и модуля бизнес-логики:

- middleware проверки времени (TimeComparisonMiddleware). Данный компонент контролирует соответствие временной метки, указанной в запросе, текущему системному времени с допустимым отклонением;
- middleware проверки контрольной суммы (ChecksumComparisonMiddleware). Сверяет контрольную сумму, вычисленную сервером, с контрольной суммой, переданной клиентом;
- middleware проверки уникальности контрольной суммы (ChecksumExistenceMiddleware). Проверяет уникальность поступившего запроса, взаимодействуя с распределённым кешем;

- бизнес-логика. Центральный компонент, отвечающий за непосредственную обработку запросов после прохождения всех проверок;

4) распределённый кеш (Distributed Cache). Выполняет функцию временного хранилища контрольных сумм, исключая возможность повторного использования ранее отправленных запросов.

Процесс взаимодействия компонентов системы включает следующие этапы:

1) генерация и отправка запроса на стороне клиента:

- клиент генерирует случайное значение и получает актуальное время от TimeProvider;
- объединяет полученные данные с помощью специального алгоритма для формирования контрольной суммы;
- формирует итоговый запрос, включающий сгенерированное случайное значение, метку времени и контрольную сумму, а также любые другие обязательные данные;
- отправляет сформированный запрос на сервер;

2) обработка запроса в TimeComparisonMiddleware:

- middleware получает актуальное время от TimeProvider;
- сравнивает временную метку из запроса с текущим временем;
- если временная разница превышает допустимый порог X единиц времени, middleware отклоняет запрос;
- в случае допустимой разницы middleware передаёт запрос далее;

3) проверка запроса в ChecksumComparisonMiddleware:

- middleware повторно вычисляет контрольную сумму, используя данные запроса (случайное значение и временную метку);
- сравнивает её с контрольной суммой, переданной от клиента;
- при несовпадении контрольных сумм запрос отклоняется;
- при совпадении запрос переходит к следующему этапу проверки;

4) проверка уникальности запроса в ChecksumExistenceMiddleware:

- middleware проверяет наличие контрольной суммы в распределённом кеше;
- если сумма обнаружена, запрос считается повторным и отклоняется;
- если сумма отсутствует, она сохраняется в кеш на заданный интервал X единиц времени, чтобы предотвратить повторную обработку;
- запрос направляется в модуль бизнес-логики;

5) бизнес-логика. После успешного прохождения всех вышеуказанных этапов осуществляется непосредственная обработка запроса в соответствии с заложенными в системе функциями.

Схема предложенной архитектуры отражена на Рисунке 1.

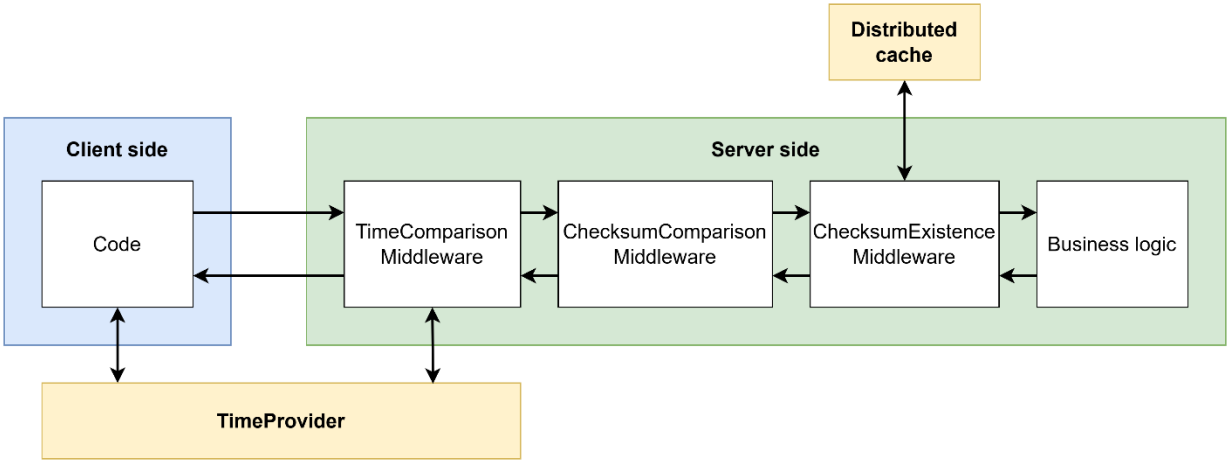

Рисунок 1 – Архитектурная схема защиты от Replay-атак

Разработанная архитектура обеспечивает надежную защиту за счёт многоуровневой проверки запросов в промежуточном ПО (middleware). Данный подход позволяет исключить возможность повторного использования запросов, а также противостоять Replay-атакам посредством следующих механизмов:

- проверки актуальности временной метки, предотвращающей обработку устаревших запросов;
- проверки контрольной суммы, предоставляющей защиту от модификаций запросов;
- проверки уникальности запросов, гарантирующей, что одни и те же запросы не будут выполняться повторно.

Совместное применение вышеперечисленных механизмов обеспечивает эффективную защиту от Replay-атак и высокий уровень безопасности клиент-серверного взаимодействия.

Поставщик времени

Поставщик времени играет важную роль в клиент-серверном взаимодействии, обеспечивая синхронизацию времени между всеми компонентами распределённой системы. Такая синхронизация является критически важной для согласованности событий и действий в системе, где требуется использовать единый временной стандарт [1]. Применение поставщика времени позволяет получить унифицированные и точные временные метки, что значительно облегчает процессы аудита и анализа, а также способствует повышению безопасности, исключая атаки, связанные с манипуляцией временными метками. Кроме того, он гарантирует корректную работу механизмов, имеющих строгие временные рамки, таких как управление сроком действия токенов.

Рекомендуется использовать в качестве стандартного поставщика времени серверную часть, которая отвечает за проверку HTTP-запросов. Такой подход минимизирует расходы, связанные с обращением к внешним API. Однако, если прямая связь между участниками системы невозможна, следует использовать внешние источники времени.

Алгоритм объединения строк

Существует большое разнообразие алгоритмов объединения строк, однако особую ценность представляют алгоритмы, позволяющие легко изменять способ объединения без

существенных вмешательств в основной код приложения. К таким алгоритмам относятся методы объединения строк с использованием seed или функций [3]. С точки зрения реализации, оба подхода похожи, однако главное отличие заключается в способе задания опорных элементов: при использовании seed они заданы заранее, а в случае функций они вычисляются динамически, что предпочтительнее для больших наборов данных.

Использование seed при объединении строк позволяет четко контролировать метод комбинирования элементов и легко изменять его при необходимости, не затрагивая базовую архитектуру приложения. Это особенно значимо для обеспечения безопасности и обновления логики обработки данных со временем.

Основные характеристики подхода:

1) контролируемый процесс объединения:

- seed задаёт конкретную последовательность чередования символов случайной строки и временной метки, обеспечивая непредсказуемость и гибкость;
- обновление seed полностью изменяет итоговый порядок объединения.

2) простота внесения изменений:

- обновление seed не требует модификации кода алгоритма;
- генерация нового seed позволяет оперативно адаптировать алгоритм к изменяющимся требованиям безопасности и бизнес-логики.

Описание алгоритма:

- seed определяет порядок чередования символов, поступающих из временной метки и случайной строки. В seed указываются элементы перечисления (например, Time и RandomString), определяющие, из какого источника необходимо выбрать следующий символ для формирования итоговой строки;
- алгоритм последовательно обрабатывает элементы seed, добавляя символы из соответствующих источников. Если текущий элемент seed совпадает со значением Time, выбирается символ из временной метки, если RandomString – из случайной строки. Индексы берутся циклически, позволяя формировать строки произвольной длины;
- благодаря гибкой архитектуре можно оперативно менять seed, обеспечивая безопасное, уникальное и воспроизводимое объединение данных.

Для количественной оценки вариаций возможных строк, генерируемых описанным алгоритмом, следует провести анализ комбинаторных характеристик исходных данных. Алгоритм состоит из трех основных этапов: генерации случайной строки, получения временной метки и их последующего объединения с помощью seed. Каждый из этих этапов влияет на конечное количество вариаций.

Приведём пример расчета. Пусть длина seed составляет 100 символов, из которых половина относится к временной метке (длина 26 символов), а половина – к случайной строке (длина 20 символов). Тогда количество возможных комбинаций вычисляется по следующей формуле:

$$C(l_t, s_t, l_r, s_r) = {l_t}^{s_t} \times {l_r}^{s_r} = 26^{50} \times 20^{50} \approx 6.3 \times 10^{135}$$

где:

$C(l_t, s_t, l_r, s_r)$ – количество комбинаций,

l_t – длина временной метки,

s_t – количество вхождений Time в seed,

l_r – длина случайной строки,

s_r – количество вхождений RandomString в seed.

Оценка времени, необходимого для перебора всех возможных комбинаций при скорости обработки одной комбинации за 10 микросекунд, представлена следующей формулой:

$$T = k \times t = 6.3 \times 10^{135} \times 10^{-5} = 6.3 \times 10^{130} \text{сек.} = \frac{6.3 \times 10^{130}}{31.536 \times 10^{6}} \text{г.} \approx$$

$$\approx 1.996 \times 10^{123} \text{ г.}$$

где:

T – время для полного перебора всех комбинаций,

k – количество комбинаций,

t – время обработки одной комбинации.

Таким образом, анализ показывает, что практическая реализация перебора всех возможных вариантов является неосуществимой ввиду огромных временных затрат (порядка квадрагинтиллионов лет). Следовательно, описанный алгоритм гарантирует высокую степень безопасности, поскольку создаёт уникальные и неповторяющиеся комбинации данных.

Время хранения контрольной суммы

Для определения периода хранения контрольной суммы и допустимого временного интервала необходимо выполнить следующие шаги:

1) установить нижнюю границу временного интервала – это время, требуемое на передачу и проверку запроса от момента его формирования на стороне клиента до момента проверки на стороне сервера.

2) установить верхнюю границу временного интервала – максимальное время, по истечении которого требуется очистить кеш для предотвращения переполнения оперативной памяти.

Предположим, что нижняя временная граница составляет примерно 1200 мс, при этом её величина может варьироваться в зависимости от скорости соединения пользователей, особенностей реализации клиентской части и сложности обфускации.

Для вычисления верхней границы необходимо определить три ключевых показателя:

1) объём памяти, требуемый для хранения одной контрольной суммы;

2) объём оперативной памяти, выделенной для кеширования;

3) интенсивность поступления запросов в секунду.

В качестве системы кеширования рассматривается Redis. Чтобы определить объём памяти для хранения одной контрольной суммы, следует сохранить контрольную сумму в кеш и измерить её размер в байтах [4]. Результаты оценки представлены на Рисунке 2.

```
# redis-cli
127.0.0.1:6379> SET rap:bd475ba69641d2e4f2d31e43d968f73282e2da5dd9126afcd54592fd30794e18 "ok" EX 20
OK
127.0.0.1:6379> MEMORY USAGE rap:bd475ba69641d2e4f2d31e43d968f73282e2da5dd9126afcd54592fd30794e18
(integer) 128
127.0.0.1:6379> SET rap:bd475ba69641d2e4f2d31e43d968f73282e2da5dd9126afcd54592fd30794e18 "ok" EX 60
OK
127.0.0.1:6379> MEMORY USAGE rap:bd475ba69641d2e4f2d31e43d968f73282e2da5dd9126afcd54592fd30794e18
(integer) 128
```

Рисунок 2 – Размер памяти для хранения одной контрольной суммы

На рисунке видно, что на каждую контрольную сумму выделяется 128 байт.

Остальные значения невозможно определить точно, поэтому предположим следующие параметры: сервер располагает 32 Гб оперативной памяти, выделенными для Redis, и обрабатывает 50 000 запросов в секунду. Тогда расчёт максимального времени хранения данных выглядит следующим образом:

$$T_{max} = \frac{m_r}{m_c \times k_{rps}} = \frac{32\ \text{Гб.}}{128\ \text{б.} \times 50\ 000} = \frac{34\ 359\ 738\ 368\ \text{б.}}{128\ \text{б.} \times 50\ 000} \approx 5\ 369\ \text{сек.}$$

где:

T_{max} – максимальное время хранения,

m_r – объём памяти, выделенный под Redis,

m_c – объём памяти на одну контрольную сумму,

k_{rps} – количество поступающих запросов в секунду.

В результате расчетов получен диапазон от 1,2 сек. до 5 369 сек. Любое значение в этом интервале может быть приемлемо для практического использования. В условиях разного качества интернет-соединения у пользователей рекомендуется выбрать оптимальное значение на уровне примерно 10 секунд [5].

Заключение

В статье предложена и детально описана архитектура системы, направленной на защиту HTTP-запросов от Replay-атак за счёт многоуровневой проверки на промежуточных этапах обработки. Ключевыми элементами системы являются поставщик времени, middleware-компоненты для проверки временных меток, контрольных сумм и их уникальности, а также распределённый кеш для хранения контрольных сумм. В результате количественного анализа доказана высокая степень безопасности предложенного подхода, обусловленная невозможностью практического перебора всех комбинаций контрольных сумм за разумное время.

Рассмотрены рекомендации по выбору оптимального временного интервала для хранения контрольных сумм в зависимости от характеристик системы. Таким образом, представленная архитектура позволяет существенно повысить устойчивость веб-приложений к Replay-атакам,

снижая риски финансовых и репутационных потерь и обеспечивая надёжность и безопасность информационного обмена.

Список литературы

1. A Guide to Replay Attacks And How to Defend Against Them. — Текст : электронный // Packetlabs : [сайт]. — URL: https://www.packetlabs.net/posts/a-guide-to-replay-attacks-and-how-to-defend-against-them/ (дата обращения: 02.12.2024).
2. Srivastava, G. K. Time Synchronization in Distributed Systems / G. K. Srivastava. — Текст : электронный // Medium : [сайт]. — URL: https://medium.com/stackspacearena/time-synchronization-in-distributed-systems-ceebf657d874 (дата обращения: 13.12.2024).
3. Marsaglia, G. Seeds for Random Number Generators / G. Marsaglia // Communications of the ACM. — 2003. — Т. 46, № 5. — С. 90–93. — DOI: 10.1145/769800.769827.
4. Al-Allawee, A.; Lorenz, P.; Abouaissa, A.; Abualhaj, M. A Performance Evaluation of In-Memory Databases Operations in Session Initiation Protocol / A. Al-Allawee, P. Lorenz, A. Abouaissa, M. Abualhaj // Network. — 2023. — Т. 3, № 1. — С. 1–14. — DOI: 10.3390/network3010001.
5. Gafurova, D. The Importance of Internet Speed on the Quality of Public Service Systems / D. Gafurova // Economics and Education. — 2023. — Т. 24, № 4. — С. 302–306. — DOI: 10.55439/ECED/vol24_iss4/a50.

References

1. A Guide to Replay Attacks And How to Defend Against Them. — Текст : электронный // Packetlabs : [сайт]. — URL: https://www.packetlabs.net/posts/a-guide-to-replay-attacks-and-how-to-defend-against-them/ (дата обращения: 02.12.2024).
2. Srivastava, G. K. Time Synchronization in Distributed Systems / G. K. Srivastava. — Текст : электронный // Medium : [сайт]. — URL: https://medium.com/stackspacearena/time-synchronization-in-distributed-systems-ceebf657d874 (дата обращения: 13.12.2024).
3. Marsaglia, G. Seeds for Random Number Generators / G. Marsaglia // Communications of the ACM. — 2003. — Т. 46, № 5. — С. 90–93. — DOI: 10.1145/769800.769827.
4. Al-Allawee, A.; Lorenz, P.; Abouaissa, A.; Abualhaj, M. A Performance Evaluation of In-Memory Databases Operations in Session Initiation Protocol / A. Al-Allawee, P. Lorenz, A. Abouaissa, M. Abualhaj // Network. — 2023. — Vol. 3, № 1. — pp. 1–14. — DOI: 10.3390/network3010001.
5. Gafurova, D. The Importance of Internet Speed on the Quality of Public Service Systems / D. Gafurova // Economics and Education. — 2023. — Vol. 24, № 4. — pp. 302–306. — DOI: 10.55439/ECED/vol24_iss4/a50.