

УДК 004.056:004.738.5

Величко А.В.¹, Киричек Г.Г.²

¹ студ. гр. КНТз-513 НУ «Запорізька Політехніка»

² доц. НУ «Запорізька Політехніка»

МЕХАНІЗМИ ЗАХИСТУ ВЕБДОДАТКІВ У DJANGO ІЗ ВИКОРИСТАННЯМ MIDDLEWARE

Забезпечення безпеки вебдодатків є невід'ємною складовою сучасної веб-розробки, оскільки зростання кількості автоматизованих запитів і атак здатне спричинити перевантаження серверів та порушення стабільності функціонування систем [1]. Одним із ефективних підходів до розв'язання цієї проблеми є впровадження механізмів контролю трафіку та обмеження частоти звернень користувачів. У фреймворку Django такі можливості реалізуються за допомогою middleware, що забезпечує проміжну обробку HTTP-запитів і відповідей [2].

Метою роботи є дослідження можливостей middleware у Django та розробка механізму захисту вебдодатку шляхом обмеження кількості запитів від клієнтів і блокування підозрілої активності. Об'єктом дослідження є процес обробки HTTP-запитів та відповідей у вебдодатках. Предметом є методи реалізації контролю доступу та фільтрації трафіку на рівні middleware. Для досягнення поставленої мети необхідно: провести аналіз сучасних методів забезпечення безпеки вебдодатків, способів протидії автоматизованим запитам і мережевим атакам; дослідити архітектурні особливості обробки HTTP-запитів і відповідей у фреймворку Django; визначити принципи реалізації контролю трафіку та обмеження частоти звернень користувачів; розробити механізм захисту вебдодатку на основі middleware із функціями фільтрації запитів і блокування підозрілої активності, а також оцінити ефективність запропонованого механізму за показниками навантаження і стійкості до атак.

Middleware забезпечує централізовану обробку вхідних і вихідних даних між клієнтом і серверною частиною застосунку, що дає змогу реалізувати механізми фільтрації, логування та обмеження частоти звернень без модифікації основної логіки. Такий підхід відповідає концепції gate

limiting, яка широко застосовується для захисту вебсервісів від надмірного навантаження [1] та підвищує керованість системи. У межах роботи розроблено компонент, який аналізує активність користувачів за IP-адресами: виконується підрахунок кількості звернень у заданому часовому інтервалі, після чого у разі перевищення встановленого порогу застосовується тимчасове блокування. Додатково реалізовано ведення журналу подій, що забезпечує фіксацію як звичайних звернень, так і випадків обмеження доступу [3]. Налаштування системи логування у вебдодатку здійснюється із використанням стандартного модуля logging мови Python, інтегрованого у фреймворк Django через конфігурацію LOGGING у файлі settings.py.

```
LOGGING = {'version': 1,
           'disable_existing_loggers': False,
           'handlers': {
               'file': {
                   'level': 'INFO',
                   'class': 'logging.FileHandler',
                   'filename': 'visitor.log',    },
               'banned_file': {
                   'level': 'INFO',
                   'class': 'logging.FileHandler',
                   'filename': 'banned_ip.log',
               },
           },
           'loggers': {
               'visitor_logger': {'handlers': ['file'], 'level':
               'INFO'}, 'banned_ip_logger': {'handlers': ['banned_file'],
               'level': 'INFO'},
           },
           }
```

Це дозволяє централізовано збирати дані про активність користувачів, контролювати перевищення лімітів запитів та перенаправляти повідомлення різних рівнів важливості в окремі файли або зовнішні системи моніторингу [2]. Основна логіка обмеження частоти HTTP-запитів реалізована у проміжному класі middleware фреймворку Django, що перехоплює кожен запит до його обробки (view) та здійснює перевірку активності клієнта.

```
import logging
from django.http import JsonResponse
from django.utils.timezone import now
from collections import defaultdict
from datetime import timedelta
logger = logging.getLogger('visitor_logger')
banned_logger = logging.getLogger('banned_ip_logger')
REQUEST_LIMIT = 50
TIME_WINDOW = 20
BAN_TIME = 300
```

```

ip_data = defaultdict(list)
banned_ip = {}
def get_client_ip(request):
    x_forwarded_for=
request.META.get('HTTP_X_FORWARDED_FOR')
    if x_forwarded_for:
        return x_forwarded_for.split(',')[0]
    return request.META.get('REMOTE_ADDR')
class SaveIPAddressMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
    def __call__(self, request):
        ip = get_client_ip(request)
        curr = now()
        if ip in banned_ip and curr < banned_ip[ip]:
            return JsonResponse({'error': 'IP Blocked'}, status=403)
        ip_data[ip] = [
            t for t in ip_data[ip]
            if t > curr - timedelta(seconds=TIME_WINDOW)
        ]
        if len(ip_data[ip]) >= REQUEST_LIMIT:
            banned_ip[ip] = curr + timedelta(seconds=BAN_TIME)
            banned_logger.info(f"BAN: {ip}")
        return JsonResponse({'error': 'Too many requests'},
status=429)
        ip_data[ip].append(curr)
        logger.info(f"{ip} {request.path}")
        return self.get_response(request)

```

Запропонований підхід забезпечує ефективне обмеження надмірної активності клієнтів і зниження навантаження на сервер. Використання логування надає можливість подальшого аналізу поведінки користувачів і виявлення потенційних загроз безпеці. У високонавантажених системах подібні механізми реалізуються із застосуванням зовнішніх систем збереження стану, зокрема Redis [3], що забезпечує масштабованість рішення. Серед ключових переваг запропонованого підходу доцільно відзначити простоту реалізації, повну інтеграцію зі стандартними засобами Django та відсутність потреби у сторонніх залежностях [4].

За результатами дослідження встановлено, що використання middleware у Django є ефективним і доступним засобом реалізації базових механізмів захисту вебдодатків. Запропонований підхід забезпечує контроль трафіку, обмеження підозрілої активності та підвищення загальної стабільності системи, а також характеризується гнучкістю щодо подальшого розширення й адаптації до вимог конкретного проєкту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Hoffman A. Web application security. O'Reilly Media, Inc, 2024. 395 p.
2. Melé, A. Django 5 By Example: Build powerful and reliable Python web applications from scratch. Packt Publishing Ltd., 2024. 781 p.
3. Woldman, T. Hands-On Microservices with Django: Build cloud-native and reactive applications with Python using Django 5. Packt Publishing Ltd., 2024.
4. Kleppmann, M. Designing data-intensive applications. O'Reilly Media, 2019. 140 p.