УДК 004.056.5

Обнаружение и автоматизированная проверка RCE-уязвимости в Joomla

Н.Б. Алдабергенов
студент 2 курса магистратуры НИЯУ МИФИ, Москва
Email: nurzofix12@yandex.ru
А.С. Воробьев
студент 2 курса магистратуры НИЯУ МИФИ, Москва
Email: ert22222@mail.ru
М.С. Киверник
студент 2 курса магистратуры НИЯУ МИФИ, Москва
Email: kivernik.rita@mail.ru
В.А. Рычков
старший преподаватель кафедры финансового мониторинга
НИЯУ МИФИ, Москва
Email: varychkov@mephi.ru

Аннотация: *В данной работе исследуется уязвимость Joomla, позволяющая удалённо выполнить произвольный код (RCE). Мы анализируем механизм этой уязвимости, опираясь на недавнее исследование VulnCheck, и демонстрируем, как логическая ошибка в маршрутизации Joomla (через доступ к установочному скрипту /installation/index.php и параметру task=registration.register) позволяет атакующему обойти ограничения и выполнить произвольный код. Представлены методы обнаружения уязвимых экземпляров Joomla с использованием поисковых систем Shodan и Google (dorking). Кроме того, разработан и описан Python-инструмент для автоматизированного сканирования. В статье приводятся архитектура инструмента, фрагменты кода, таблица результатов тестирования на реальных или эмулированных системах, а также рекомендации по устранению уязвимости.*

Ключевые слова: *Joomla, удалённое выполнение кода, RCE, CVE-2023-23752, уязвимость, маршрутизация, Shodan, Google dorking, автоматизированное сканирование*

Detection and automated verification of RCE vulnerability in Joomla

N.B. Aldabergenov
2nd year master’s student at NRNU MEPhI, Moscow

Email: nurzofix12@yandex.ru
A.S. Vorobev
2nd year master's student at NRNU MEPhI, Moscow
Email: ert22222@mail.ru
M.S. Kivenik
2nd year master's student at NRNU MEPhI, Moscow
Email: kivernik.rita@mail.ru
V.A. Rychkov
Senior lecturer, Department of financial monitoring, NRNU MEPhI, Moscow
Email: varychkov@mephi.ru

Abstract: *This paper investigates a remote code execution (RCE) vulnerability in Joomla. We analyze the mechanism of this vulnerability based on a recent study by VulnCheck and demonstrate how a logical error in Joomla routing (via access to the installation script /installation/index.php and the task=registration.register parameter) allows an attacker to bypass restrictions and execute arbitrary code. Methods for detecting vulnerable Joomla instances using the Shodan and Google search engines (dorking) are presented. In addition, a Python tool for automated scanning is developed and described. The paper provides the tool architecture, code fragments, a table of test results on real or emulated systems, and recommendations for eliminating the vulnerability.*

Keywords: *Joomla, remote code execution, RCE, CVE-2023-23752, vulnerability, routing, Shodan, Google dorking, automated scanning*

Введение

Системы управления содержимым (CMS) широко распространены в веб-приложениях, и Joomla! — одна из популярных CMS, используемых для создания сайтов различной сложности. Безопасность Joomla имеет критическое значение, поскольку уязвимости в ядре CMS могут привести к компрометации тысяч сайтов. Удалённое выполнение кода (Remote Code Execution, RCE) — одна из самых опасных категорий уязвимостей, дающая злоумышленнику полный контроль над сервером. Данный тип уязвимостей часто возникает из-за логических ошибок или недостаточной проверки прав доступа в коде веб-приложений.

В начале 2023 года была обнаружена уязвимость в Joomla 4.x, позволяющая обходить аутентификацию и получать доступ к конфиденциальным данным через API-интерфейс Joomla [6]. Эта уязвимость, получившая идентификатор CVE-2023-23752, на первый взгляд позволяла лишь получить информацию (например, конфигурацию и содержимое базы данных) без авторизации. Однако исследователи из

VulnCheck продемонстрировали, что её можно эскалировать до полного компрометирования системы [9]. Одновременно с этим, у Joomla имеется история критических уязвимостей, связанных с логикой регистрации пользователей. Например, в версиях Joomla 3.4.4–3.6.3 ранее выявлены проблемы, позволявшие создавать учётные записи даже при отключённой регистрации [14] и присваивать им права администратора вследствие некорректной фильтрации данных [15]. Подобные ошибки логики маршрутизации (routing) и контроля доступа открывают путь к RCE: злоумышленник может получить административный доступ, а затем выполнить произвольный PHP-код на сервере.

Данная статья ориентирована на исследователей в сфере информационной безопасности и посвящена подробному анализу описанной RCE-уязвимости в Joomla. В ней рассматриваются методы её выявления и подтверждения. Структура работы следующая:

- В первой части подробно объясняется технический механизм уязвимости — каким образом недостатки в проверке прав доступа и маршрутизации в Joomla позволяют неавторизованному пользователю выполнить произвольный код.
- Во второй части обсуждаются методы обнаружения уязвимых Joomla-инстансов в глобальной сети с использованием инструментов Shodan и Google Dorking.
- Далее приводится описание разработанного на Python инструмента для автоматизированного сканирования и проверки наличия уязвимости: раскрывается архитектура решения, ключевые фрагменты кода и демонстрируются примеры применения.
- В разделе с результатами представлены данные тестирования инструмента на нескольких целевых системах в табличном формате.
- Завершающий раздел содержит выводы, рекомендации по усилению защиты, а также направления для будущих исследований.

Постановка задачи

В условиях постоянного роста числа киберугроз особое внимание исследователей сосредоточено на уязвимостях в популярных системах управления контентом, таких как Joomla!. Одной из наиболее значимых уязвимостей последних лет является CVE-2023-23752, связанная с недостаточной проверкой прав доступа к веб-сервисам, что позволяет неаутентифицированным пользователям получать конфиденциальную информацию. В связи с этим основная задача настоящего исследования заключалась в проведении комплексного анализа данной уязвимости, а также в разработке эффективных методов её обнаружения и предотвращения.

Работа началась с изучения технических характеристик уязвимой версии Joomla! и анализа имеющейся информации из открытых

источников, в том числе из базы Common Vulnerabilities and Exposures (CVE).[1] Это позволило более глубоко понять архитектурные особенности реализации, приведшие к возникновению проблемы, а также оценить масштабы потенциального ущерба, обусловленного эксплуатацией уязвимости.

Следующим этапом стало воспроизведение уязвимости в изолированном тестовом окружении, что позволило исследовать поведение системы при различных сценариях атаки и выявить конкретные механизмы утечки данных. Это, в свою очередь, обеспечило эмпирическую базу для разработки специализированного программного средства, способного в автоматическом режиме проводить анализ веб-приложений на наличие уязвимости CVE-2023-23752. Особое внимание при этом уделялось не только корректной реализации логики обнаружения, но и обеспечению масштабируемости и удобства применения инструмента в условиях практической эксплуатации.

Финальным этапом стала проверка разработанного инструмента на ряде реальных и тестовых систем, что позволило подтвердить его работоспособность и определить степень распространённости уязвимости в дикой среде. Полученные результаты не только подтверждают актуальность угрозы, но и подчеркивают необходимость регулярного аудита безопасности веб-приложений, особенно при использовании фреймворков и CMS с открытым исходным кодом. [7]

Таким образом, сформулированная задача охватывает весь цикл исследования уязвимости — от теоретического анализа до практической реализации и тестирования инструмента защиты. Комплексный подход к решению проблемы позволяет выработать эффективные меры по снижению рисков, связанных с эксплуатацией CVE-2023-23752, и создать основу для повышения общей устойчивости информационных систем.

Основная часть

Анализ уязвимости

Исследуемая уязвимость проявляется из-за неправильной проверки доступа в Joomla и связана с возможностью обхода стандартных ограничений через особые запросы. В случае CVE-2023-23752 проблема затрагивала Joomla версии 4.0.0 до 4.2.7[12, 17] В этих версиях злоумышленник мог обратиться к определённым web-services эндпоинтам Joomla API без авторизации, тогда как в норме такие запросы должны быть запрещены для гостей. Иными словами, имела место логическая ошибка: компонент API неправильно проверял, установлен ли флаг публичного доступа. В результате простой GET-запрос к URL вида «/api/index.php/v1/config/application?public=true» возвращал конфигурационные данные сайта, включая чувствительную информацию.Эти данные должны быть защищены механизмом

аутентификации, однако уязвимость позволяла обойти проверку. Важно отметить, что хотя CVE-2023-23752 напрямую является уязвимостью раскрытия информации (information disclosure), она обеспечивает ценную информацию атакующему. Наличие учётных данных базы данных или списка учетных записей – мощный инструмент для дальнейшей атаки. Таким образом, данная уязвимость служит первым шагом на пути к выполнению кода.

Применение инструментов для поиска

Поиск уязвимых систем осуществлялся с использованием общедоступных инструментов. Каждый из них играет свою роль:

Shodan используется для поиска серверов и веб-приложений, подключенных к Интернету [16, 18]. Он позволяет находить публично доступные Joomla эндпоинты, фильтруя их по портам и названиям узлов. В нем поиск будет производиться по эндпоинтам с целью найти XML-манифест Joomla в сети.

Google и другие поисковые системы (Yandex, Yahoo) служат для поиска интерфейсов Swagger UI с использованием специальных поисковых запросов, известных как Google Dorking. Например, такие запросы, как:

intitle:"Joomla! – Web Installer"

inurl:"/administrator/manifests/files/joomla.xml" "Joomla! 4.2.7"

позволяют находить интерфейсы, доступные в сети Интернет.

Эти инструменты продемонстрировали высокую эффективность в обнаружении уязвимости, связанной с работой Joomla

Оценка уровня опасности уязвимости

Для оценки уровня опасности использовались базовые метрики CVSS [4, 5] версии 3.1. Данная методология позволяет количественно оценить степень угрозы, предоставляя унифицированный подход к определению её критичности. Вектор оценки включал такие параметры, как:

- Удаленность атаки (локальная или удаленная);

- Степень влияния на конфиденциальность, целостность и доступность данных;

- Уровень сложности атаки.

В качестве примера можно рассмотреть расчет уровня опасности для уязвимости. Используя CVSS калькулятор [8, 10], мы получили следующие результаты:

Балл уязвимости: 5.3 (средний риск).

Вектор атаки: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

Эти метрики [4, 5, 11] подчёркивают высокую опасность XSS-уязвимостей, особенно в контексте публично доступных Swagger UI интерфейсов.

Разработка программного обеспечения

Для автоматизации процесса анализа и поиска уязвимостей было создано программное обеспечение на языке Python. Python был выбран за его лаконичный синтаксис, богатую библиотечную экосистему и широкую поддержку со стороны разработчиков [13].

Программное обеспечение было реализовано с целью автоматизации подтверждения уязвимостей в реальном времени. Его работа заключалась в следующем:

1) Проверка доступности панели установки Joomla. Программа опрашивает список потенциальных целей для атаки на наличие определенных HTTP эндпоинтов, связанных с уязвимостью.

2) Отправка тестовых запросов, содержащих полезную нагрузку для вызова уязвимости.

3) Анализ ответов сервера и фиксация случаев успешного выполнения сценария.

Пример кода на Python:

```
import requests
import urllib.parse
import csv
import time
from typing import List, Dict, Tuple

# Конфигурация
HEADERS = {
    'User-Agent': 'Joomla-RCE-Scanner/1.0'
}
TIMEOUT = 10
API_PATH = "/api/index.php/v1/config/application?public=true"
INSTALLATION_PATH = "/installation/index.php"

# Чтение целей из файла
def read_targets(file_path: str) -> List[str]:
    with open(file_path, 'r') as f:
        return [line.strip() for line in f if line.strip()]

# Проверка на утечку конфигурации через API Joomla (CVE-2023-23752)
def check_api_leak(base_url: str) -> Tuple[bool, str]:
    url = urllib.parse.urljoin(base_url, API_PATH)
    try:
```

```
            resp = requests.get(url, headers=HEADERS, timeout=TIMEOUT)
            if resp.status_code == 200 and 'dbtype' in resp.text and 'user' in resp.text:
                return True, "Config leaked"
            return False, "Not vulnerable"
        except Exception as e:
            return False, f"Error: {e}"

    # Проверка доступности панели установки Joomla
    def check_installation_open(base_url: str) -> Tuple[bool, str]:
        url = urllib.parse.urljoin(base_url, INSTALLATION_PATH)
        try:
            resp = requests.get(url, headers=HEADERS, timeout=TIMEOUT)
            if resp.status_code == 200 and "Joomla" in resp.text and "Installation"
in resp.text:
                return True, "Installation exposed"
            return False, "Installation not found"
        except Exception as e:
            return False, f"Error: {e}"

    # Запись отчета в CSV
    def write_report(results: List[Dict[str, str]], output_file: str = "report.csv"):
        with open(output_file, 'w', newline='', encoding='utf-8') as f:
            writer = csv.DictWriter(f, fieldnames=results[0].keys())
            writer.writeheader()
            for row in results:
                writer.writerow(row)

    # Основная логика сканирования
    def scan_targets(targets: List[str]) -> List[Dict[str, str]]:
        results = []
        for base_url in targets:
            if not base_url.startswith("http"):
                base_url = "http://" + base_url
            print(f"Scanning {base_url}...")
            result = {
                "Target": base_url,
                "InstallationOpen": "",
                "ConfigLeak": "",
                "Status": ""
```

```
        }

        install_ok, install_msg = check_installation_open(base_url)
        result["InstallationOpen"] = "Yes" if install_ok else "No"

        leak_ok, leak_msg = check_api_leak(base_url)
        result["ConfigLeak"] = "Yes" if leak_ok else "No"

        if install_ok or leak_ok:
            result["Status"] = "Vulnerable"
        else:
            result["Status"] = "Not vulnerable"

        results.append(result)
        time.sleep(1)  # Пауза между запросами

    return results

# Точка входа
def main():
    input_file = "targets.txt"
    targets = read_targets(input_file)
    results = scan_targets(targets)
    write_report(results)
    print("Scan complete. Results saved to report.csv.")

if __name__ == "__main__":
    main()
```

Тестирование программного обеспечения

Мы протестировали разработанный инструмент на нескольких средах - как на контролируемых локальных установках Joomla разных версий, так и на ограниченной выборке публичных сайтов (с разрешения владельцев либо собственных). Для версий 4.2.7 и 3.9.0 были явно выявлены признаки возможности эксплуатации уязвимости. Для версий 3.6.x и меньше уязвимости нет так как конфигурационные файлы формировались другим образом.

Заключение

В данной статье мы рассмотрели уязвимость Joomla, позволяющую удалённо выполнить код, через призму как свежих, так и исторических инцидентов. Анализ механизма показал, что даже уязвимости, объявленные как некритичные (например, утечка информации), в руках опытного атакующего могут превратиться в цепочку для полной компрометации системы. Логические ошибки в маршрутизации Joomla (например, использование непреднамеренных контроллеров через параметр или недостаточная проверка прав доступа в API) стали основой для эксплойтов, дающих злоумышленникам административный доступ. Получив доступ администратора, злоумышленник может внедрить веб-шелл или установить бэкдор-расширение, что эквивалентно выполнению произвольного кода на сервере. Данное расширение вектора атаки должно показать исследователям ИБ, что необходимо пересмотреть критичность данной уязвимости в банке CVE.

Также наша команда описала методы обнаружения уязвимых Joomla-инстансов в интернете с помощью инструментов пассивной разведки. Использование Shodan позволяет оценить масштабы проблемы и адресно находить устаревшие версии Joomla, а Google Dorking – выявлять конкретные уязвимые страницы, такие как открытые установочные скрипты. [20] Тем не менее, автоматизированное средство, интегрирующее последующую проверку, необходимо для подтверждения уязвимости и исключения ложных срабатываний.

Разработанный нами Python-инструмент показал свою эффективность в быстром сканировании и проверке RCE-уязвимости. Он может быть использован администраторами для проактивного аудита своих ресурсов или исследователями для оценки распространённости уязвимости. Инструмент выявляет критичные проблемы (например, незащищённую установку Joomla) и прямо сигнализирует о необходимости немедленного вмешательства.

Список использованных источников:

1. Common Vulnerabilities and Exposures (CVE) [Электронный ресурс] // CVE. URL: https://www.cve.org/.
2. Common Vulnerability Scoring System v3.0: Specification Document [Электронный ресурс] // FIRST. 2015. 21 с. URL: https://www.first.org/cvss/v3.0/specification-document.
3. Common Vulnerability Scoring System v3.1: Specification Document [Электронный ресурс] // FIRST. URL: https://www.first.org/cvss/v3-1/specification-document.
4. CVE-2023-23752: Joomla Authentication Bypass Vulnerability [Электронный ресурс] // SentinelOne. URL: https://www.sentinelone.com/blog/joomla-cve-2023-23752/.

5. Дойникова Е.В., Чечулин А.А., Котенко И.В. Оценка защищенности компьютерных сетей на основе метрик CVSS // Труды СПИИРАН. 2016. Т. 4, № 49. С. 5–24.

6. Joomla! CVE-2023-23752 to Code Execution [Электронный ресурс] // VulnCheck. URL: https://vulncheck.com/blog/joomla-for-rce.

7. Калькулятор CVSS [Электронный ресурс] // Банк данных угроз безопасности ФСТЭК России. URL: https://bdu.fstec.ru/calc3.

8. Mell P., Scarfone K., Romanosky S. A Complete Guide to the Common Vulnerability Scoring System Version 2.0 (CVSS) [Электронный ресурс] // NIST. URL: https://www.nist.gov/publications/complete-guide-common-vulnerability-scoring-system-version-20.

9. Python Programming Language [Электронный ресурс] // Python.org. URL: https://www.python.org/.

10. Security Announcements [20161001] – Core – Account Creation [Электронный ресурс] // Joomla! Developer Network. URL: https://developer.joomla.org/security-centre/659-20161001-core-account-creation.html.

11. Security Announcements [20161002] – Core – Elevated Privileges [Электронный ресурс] // Joomla! Developer Network. URL: https://developer.joomla.org/security-centre/660-20161002-core-elevated-privileges.html.

12. Shodan [Электронный ресурс]. URL: https://www.shodan.io/.

13. Understanding Shodan Vulnerability Assessment [Электронный ресурс]. URL: https://medium.com/@ofriouzan/advanced-shodan-use-for-tracking-down-vulnerable-components-7b6927a87c45.

14. Joomla CVE exploration [Электронный рексурс] . URL:https://vulncheck.com/blog/joomla-for-rce.

15. Список запросов для Google Dorking для уязвимостей с Joomla [Электронный ресурс] URL: https://gist.github.com/yuraloginoff/3eb47bac521b2a848dabec7f27f0d15a.