КІБЕРНЕТИКА та КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ

УДК 004.981 DOI:10.34229/2707-451X.21.2.9

Т.О. НАУМЕНКО, В.С. ЧЕРНОМАЗ

АНАЛІЗ ЗАСТОСУВАННЯ ДЕТЕКТОРА SQL ІН'ЄКЦІЙ ПОБУДОВАНОГО НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ У БЕЗСЕРВЕРНІЙ АРХІТЕКТУРІ

Стаття є аналізом детектора SQL ін'єкцій побудованого на основі алгоритму машинного навчання у безсерверній архітектурі. Автор розглядає різні алгоритми класифікації даних та наводить результати їх роботи у хмарі, зокрема на платформі Google Cloud Platform.

Ключові слова: *машинне навчання, Google Cloud Platform, безпека, SQL ін'єкція.*

Вступ.

Аналіз останніх досліджень та публікацій

Коли йде мова про безсерверну архітектуру [1], щоб забезпечувати безпеку за допомогою таких засобів як антивіруси, системи моніторингу, системи превентивних дій тощо, перш за все потрібно звернути увагу на безпеку самого коду [2]. І це правда, якщо мається на увазі саме виключно безсерверна архітектура, тобто система, яка повністю адмініструється хмарним провайдером. Однак, якщо використовувати виключно сервіси, які пропонує провайдер, з'являється питання чи гарантує він достатній рівень безпеки. Наприклад, в Amazon Web Services неодноразово були знайдені вразливості [3]. Як зазначається в [4], лише близько 57 % компаній провели оцінку ризиків для безпеки даних у 2020 році. Одна з причин цього є те, що забезпечення безпеки системи – це складне завдання, яке потребує багато ресурсів. Саме тому, було проведено аналіз з використанням машинного навчання, яке може надати додаткову експертну оцінку для виявлення вразливостей та небезпечних звернень до системи.

Постановка проблеми і визначення мети статті

Згідно з OWASP [5] та SANS [6], однією із найбільш поширених загроз є SQL ін'єкція. Спеціалісти з усього світу пропонують багато нових рішень, які можуть допомогти вирішити дану проблему. У цій статті проаналізоване одне з них – класифікація за допомогою машинного навчання.

Ін'єкція вважається однією із найбільш небезпечних атак, оскільки несе в собі небезпеку для головних факторів системи: конфіденційності даних, керуванню рівнями доступу та процедурі встановлення справжності користувача. Вона є вставкою небезпечного запиту до бази даних у форми та запити, які надсилають на сервер користувачі. Оскільки більшість систем в Інтернет використовують бази даних для зберігання інформації користувачів, вони стають цілями зловмисників.

Дана робота націлена на аналіз застосування алгоритмів машинного навчання у роботі детекторів SQL ін'єкцій, зокрема їх роботі у безсерверній архітектурі. У роботі наведені основні риси такої архітектури, так само як і SQL ін'єкцій, та наведено аналіз роботи сервісу, який використовує алгоритми класифікації для знаходження шкідливих запитів.

Виклад основного матеріалу

1. Основні типи SQL ін'єкцій

Вразливості, які можна використати за допомогою SQL ін'єкцій можна знайти в будь-якій системі, яка використовує SQL запити. У роботі [7] авторами наведено чотири джерела можливих атак.

- *Ін'єкція через ввід даних користувача:* додатки часто отримують інформацію від користувачів через форми, наприклад реєстрація. Текстові поля в цих формах можуть бути використані зловмисниками для вставки небезпечного коду, за допомогою якого вони потім зможуть отримати доступ до секретних даних або отримати права привілейованого користувача.
- *Ін'єкція через кукі-файли:* кукі-файли використовуються розробниками додатків для зберігання кешу користувача на стороні користувача. Зловмисник може вставити небезпечний код у ці файли і таким чином змусити систему створити SQL запит, який є вразливим до атаки.
- *Ін'єкція через змінні сервера:* змінні сервера – це набір параметрів, які містять у собі мережеві заголовки, метадані, змінні середовища. Зазвичай, додатки використовують ці змінні для збору статистики та відбору трендів пошуку. Якщо ці змінні зберігаються у базі даних без валідації, то зловмисник може використати це поклавши SQL ін'єкцію напряму в змінні.
- *Ін'єкції, які зберігаються:* використовуючи ін'єкції, які зберігаються, зловмисник може помістити SQL запит напряму в базу даних та надалі неявно конструювати SQL ін'єкцію кожен раз, коли цей запит буде використовуватись.

Наприклад, якщо зловмисник використає рядок "admin'--" як значення імені користувача при реєстрації, то потім при зміні свого паролю може виникнути наступна ситуація:

```
UPDATE user_table SET password = 'my_password'
WHERE username = 'admin' -- AND password = 'old_password'
```

Оскільки в мові SQL знак "--" означає коментар, то все що йде після нього буде проігноровано.

Зловмисники можуть використовувати SQL ін'єкції для різних цілей: знаходження вразливих параметрів, визначення структури бази даних, збору даних, зміни даних, зміни прав доступу, виклику відмови служби.

Для розробників існують інструменти, які превентивно, на етапі написання коду, повідомляють їх, який код є вразливим до ін'єкцій. Однак, у даній статті річ піде про доповнення до цього – сервер, який захищає від ін'єкцій вже працюючу систему. Але перед цим потрібно також розглянути умови, в яких цей сервер буде існувати – безсерверну архітектуру.

2. Основні риси безсерверної архітектури

Як зазначає автор [8], немає чіткого уявлення про те, що із себе представляє безсерверна архітектура. Автор виділяє дві позиції:

- спочатку безсерверною архітектурою називали архітектуру систем, яка повністю або частково використовували послуги хмарних провайдерів, такі як аутентифікація, бази даних тощо. Їх назвали BaaS (Backend-as-a-Service);
- безсерверною також називають архітектуру, де логіка бекенда все ще написана розробником, але на відміну від традиційних архітектур, ця логіка перебуває у ізольованих середовищах, які спрацьовують за викликом, не мають стану та повністю адмініструються третьою стороною. Такий вид архітектури називається FaaS (Function-as-a-Service).

У даній статті йдеться про аналіз роботи серверу, який взаємодіє із FaaS, тобто припускається, що основна логіка системи створена за рахунок FaaS, а сервер працює як зворотний проксі-сервер, який фільтрує запити на основі того містять вони SQL ін'єкції чи ні.

Щоб краще зрозуміти суть FaaS потрібно більш детально розібрати визначення наведене вище:

- знаходяться в ізольованих середовищах – це означає, що різні виклики однієї функції несуть у собі різний контекст і ніяк не пов'язані між собою;
- продовженням характеристики вище є «не мають стану» – тобто, зміна контексту при першому виклику функції не гарантує того, що наступний виклик побачить цю зміну. Для таких змін необхідні інструменти на кшталт бази даних Redis;
- спрацьовують за викликом – функція починає виконання лише отримавши виклик, але при цьому вони не працюють як сервери, які постійно слухають на певному порту. При виклику функції у хмарі створюється середовище, в якому ця функція стартує своє виконання. Старт функції може бути як теплим, так і холодним: при холодному старті середовище створюється з нуля, що потребує часу, при теплому використовується середовище попереднього виклику;
- повністю адмініструються третьою стороною – у даному випадку хмарним провайдером, який надає послугу хмарних функцій, наприклад, Amazon, Google, Microsoft. Створивши функцію, розробнику не потрібно витрачати час на її розгортку та конфігурування середовища – провайдер все робить сам, у тому числі здійснення масштабування, що є невід'ємною частиною роботи будь-якої системи. Як тільки кількість запитів на функцію починає зростати, провайдер створює більше екземплярів цієї функції і розподіляє навантаження між ними.

Для проведення аналізу роботи сервісу використовувалась хмарна платформа Google Cloud Platform, яка надає FaaS послугу Cloud Functions. Cloud Functions дозволяє створювати функції на різних мовах програмування: Python, Go, Ruby, Node.js, Java, PHP, .Net.

3. Класифікація SQL ін'єкцій з використанням алгоритмів машинного навчання

Для виявлення небезпечних запитів, які містять SQL ін'єкції, використані наступні алгоритми машинного навчання: логістична регресія, наївний байєсів класифікатор, *к*-найближчих сусідів, згорткова нейронна мережа.

Як видно з рисунку, після того, як веб-сервер отримав запит, він посилає цей запит за допомогою HTTP-протокола алгоритму машинного навчання, який після підготовки отриманих даних вирішує є запит небезпечним чи ні. Якщо, запит є небезпечним – він відсіюється і нікуди не йде, якщо ж ні – веб-сервер перенаправляє запит на API Gateway, який є шлюзом для бек-енд сервісів та чудово поєднується з Cloud Functions. Оскільки запит до API має доходити швидко, то крім точності алгоритма важливо визначити його швидкість. Для розрахунків використовувалась машина Google Cloud Platform типу C2-standard, яка має 4 ядра процесору та 16 Гб оперативної пам'яті. Для тренування був використаний датасет, який містить приклади шкідливих і безпечних запитів обсягом у 33 тисячі записів.

Перед тренуванням моделі необхідно підготувати дані та очистити їх від інформації, яка може зашкодити точності алгоритму. Для цього із запитів були вилучені найбільш вживані англійські слова, залишені необхідні, наприклад "select", що є командою мови SQL. Після очистки дані векторизовані, в даному випадку кожному слову була співставлена кількість їх появи в тренувальних даних. Результати аналізу алгоритмів без урахування часу передачі запиту від веб-сервера до алгоритма наведено в таблиці.

Підготовка запиту до алгоритму зайняла в середньому ще 10 мс, що разом із середнім часом передачі запиту від веб-сервера до алгоритму і назад, який приблизно дорівнює 200 мс, на обробку одного запиту алгоритмом йде близько 215 мс. Звичайно можна помістити алгоритм і на сам веб-сервер, однак рішення вище зроблене із переконань, що в майбутньому систему доведеться масштабувати, тому для забезпечення більш гнучких рішень було вирішено розмістити алгоритм на окремій машині.

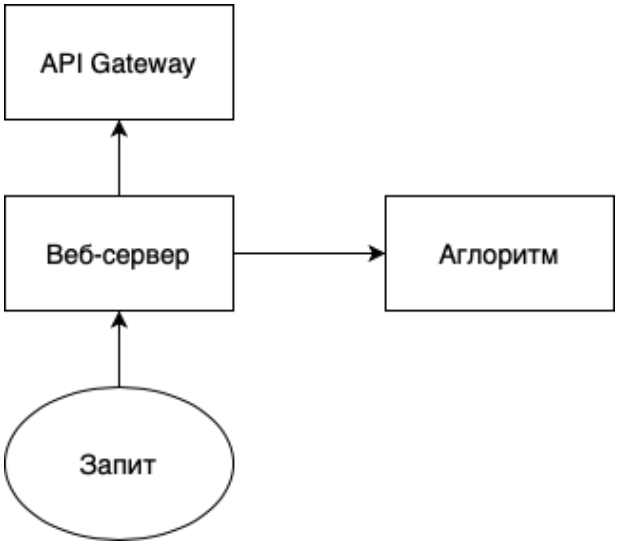

РИСУНОК. Архітектура системи для дослідження

ТАБЛИЦЯ

Назва алгоритму	Точність, %	Швидкість обробки одного запиту, мс
Логістична регресія	96	3.2
Наївний байєсів класифікатор	97	3.1
K-найближчих сусідів	96	2.6
Згорткова нейронна мережа	91	2.0

Висновки і перспективи подальших досліджень

Як видно з таблиці результати є доволі хорошими. Вони можуть бути покращені за рахунок більш детальної очистки даних та підбором більш точних параметрів. Варто зазначити, що дане рішення не є заміною людської експертизи, а є лише доповненням до неї. Все ще є поганою практикою писати вразливий до атак код: з цією проблемою допомагають впоратись такі інструменти, як SonarQube. Разом всі ці заходи безпеки допоможуть будувати безпечні системи, в яких не буде вразливих місць, або їх кількість буде мінімальною. Варто також зазначити, що дане рішення добре вбудовується у безсерверну архітектуру, добре масштабується і зручно розгортається. Але чи є доцільним його використання, слід визначати з фінансової точки зору, як і безсерверну архітектуру в цілому: GCP пропонує власне рішення GCP Firewalls, котре, однак, не використовує машинне навчання. GCP Firewalls коштує $0.02 за кожен ГБ трафіку в той час як оренда машини типу C2 - $0.03 за годину праці, тому рішення слід приймати в залежності від того, яка кількість навантаження приходиться на систему.

Подальші дослідження будуть направлені на використання більшої кількості прикладів нейронних мереж, які змогли б адаптуватись самостійно під отримані дані. Оскільки способи атак постійно змінюються, то така мережа змогла б істотно ускладнити зловмиснику задачу проникнення в систему, що гарантує підвищену безпеку.

Список літератури

1. Науменко Т.О. Безсерверна технологія (Functions as a Service) для створення хмарних мікросервісних додатків. *Комп'ютерно-інтегровані технології: освіта, наука, виробництво.* 2018. Вип. 33. С. 25–30.
2. Security Best Practices. https://docs.aws.amazon.com/whitepapers/latest/serverless-architectures-lambda/security-best-practices.html (звернення: 05.06.2021).
3. 7 Cloud Computing Security Vulnerabilities and What to Do About Them. https://towardsdatascience.com/7-cloud-computing-security-vulnerabilities-and-what-to-do-about-them-e061bbe0faee (звернення: 05.06.2021).
4. Stainer P. Alarming Cybersecurity Statistics for 2021 and the Future. https://www.retarus.com/blog/en/alarming-cybersecurity-statistics-for-2021-and-the-future/ (звернення: 05.06.2021).
5. OWASP Top Ten. https://owasp.org/www-project-top-ten/ (звернення: 28.05.2021).
6. CWE/SANS most dangerous software errors. https://www.sans.org/top25-software-errors/ (звернення: 28.05.2021).
7. Halfond W.G., Viegas J., Orso A. A classification of sql-injection attacks and countermeasures. *Proceedings of the IEEE International Symposium on Secure Software Engineering*. Vol. 1. IEEE. 2006. P. 13–15.
8. Roberts M. Serverless Architectures. https://martinfowler.com/articles/serverless.html (звернення: 28.05.2021).

Одержано 08.06.2021

Науменко Тетяна Олександрівна,
аспірантка Інституту прикладного системного аналізу НТУУ КПІ, Київ,
https://orcid.org/0000-0002-8660-597X

Черномаз Вадим Сергійович,
студент-бакалавр Інституту прикладного системного аналізу НТУУ КПІ, Київ.
https://orcid.org/0000-0003-2596-9734
chernomazv@gmail.com

UDC 004.981

Tetiana Naumenko, Vadym Chernomaz *

Analysis of Usage of SQL Detector Based on Artificial Intelligence in Serverless Architecture

Institute for Applied System Analysis NTUU KPI, Kyiv, Ukraine
* *Correspondence: chernomazv@gmail.com*

Introduction. The widespread use of the Internet leads to a fast increase of the quantity of data that goes into it. This generates interest in intruders which try different approaches to steal this data. One of the most popular approaches is SQL injection. There are a lot of measures which help to prevent and decrease the risk of being subjected to this attack: usage of code analysis tools, usage of firewalls which can filter dangerous traffic etc.

Usage of reverse proxy is analysed in this article, which with the help of machine learning algorithms checks requests for SQL injections and based on the result passes or forbids the request to go.

It is worth mentioning that such a solution is not a replacement of human expertise but addition to it, which with the help of big data can give an accurate result in most cases.

The purpose of the paper is to analyse and show effectiveness of usage of machine learning in information system security provisioning tasks with the system working in serverless architecture.

Results. A system is designed and developed which with the help of machine learning classifies received requests. The system is deployed to the cloud hosting Google Cloud Platform and integrated into an application which is designed according to the serverless architecture principles. Multiple algorithms were used to compare effectiveness of the system and percentage of successful results were calculated for each of them. Also, an average time of request execution is calculated for each algorithm.

Conclusions. Each algorithm's result of successful request classification is above 90% which is considered to be more than acceptable. The result can be improved using more data to train machine learning models. The system fits for work in serverless applications thanks to the simplicity of its integration but it should be considered if it fits from a hardware rent point of view.

Keywords: machine learning, Google Cloud Platform, security, SQL injection.