BÁO CÁO ĐỒ ÁN

**Môn học: Bảo mật Web và ứng dụng**

**Tên chủ đề: NoSQL Injection**

*GVHD: Nguyễn Công Danh*

1. **THÔNG TIN CHUNG:**

Lớp: NT213.P12ANTT

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** | **Email** |
| 1 | Lê Ngọc Hiếu | 22520435 | 22520435@gm.uit.edu.vn |
| 2 | Trần Minh Hiếu | 22520445 | 22520435@gm.uit.edu.vn |
| 3 | Trần Hữu Hiếu | 22520444 | 22520435@gm.uit.edu.vn |
| 4 | Hồ Vĩnh Nhật | 22521013 | 22520435@gm.uit.edu.vn |

1. **NỘI DUNG THỰC HIỆN:[[1]](#footnote-2)**

|  |  |  |
| --- | --- | --- |
| **STT** | **Công việc** | **Kết quả tự đánh giá** |
| 1 | Tổng quan đề tài | 100% |
| 2 | Triển khai kịch bản | 100% |
| 3 | Kết luận | 100% |

1. **PHÂN CÔNG THÀNH VIÊN:**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và tên** | **Phân công** | **Đóng góp** |
| 1 | Lê Ngọc Hiếu | Xây dựng kịch bản demo, thực hiện demo | 25% |
| 2 | Trần Minh Hiếu | Xây dựng kịch bản demo, thực hiện demo | 25% |
| 3 | Trần Hữu Hiếu | Soạn slide báo cáo, tìm nguồn demo | 25% |
| 4 | Hồ Vĩnh Nhật | Soạn slide báo cáo, tìm nguồn demo | 25% |

**MỤC LỤC**

[MỤC LỤC 2](#_Toc186492017)

[DANH MỤC HÌNH ẢNH 4](#_Toc186492018)

[Chương I: TỔNG QUAN ĐỀ TÀI 6](#_Toc186492019)

[1.1. Tổng quan về NoSQL 6](#_Toc186492020)

[1.1.1. NoSQL là gì 6](#_Toc186492021)

[1.1.2. Đặc điểm của NoSQL 7](#_Toc186492022)

[1.1.3. Ưu và nhược điểm của NoSQL 8](#_Toc186492023)

[1.1.4. Trường hợp nên sử dụng NoSQL 8](#_Toc186492024)

[1.2. NoSQL Injection 8](#_Toc186492025)

[1.2.1. NoSQL Injection là gì? 8](#_Toc186492026)

[1.2.2. Cách thức hoạt động của NoSQL Injection 9](#_Toc186492027)

[1.2.3. Hệ quả khi NoSQL Injection thành công 9](#_Toc186492028)

[1.2.4. Cách phòng chống NoSQL Injection 10](#_Toc186492029)

[1.3. Mục tiêu đề tài 10](#_Toc186492030)

[Chương II: TRIỂN KHAI KỊCH BẢN 11](#_Toc186492031)

[2.1 Công cụ sử dụng 11](#_Toc186492032)

[2.2 Kịch bản 1: Tấn công cơ chế xác thực (Web tự dựng) 11](#_Toc186492033)

[2.2.1 Mô tả lỗ hổng 11](#_Toc186492034)

[2.2.2 Nguyên nhân dẫn đến lỗ hổng 11](#_Toc186492035)

[2.2.3 Các bước tấn công 12](#_Toc186492036)

[2.2.4 Mức độ ảnh hưởng 13](#_Toc186492037)

[2.2.5 Khuyến cáo khắc phục 14](#_Toc186492038)

[2.3 Kịch bản 2: Tấn công thanh tìm kiếm (Web tự dựng) 14](#_Toc186492039)

[2.3.1 Mô tả lỗ hổng 14](#_Toc186492040)

[2.3.2 Nguyên nhân dẫn đến lỗ hổng 14](#_Toc186492041)

[2.3.3 Các bước tấn công 15](#_Toc186492042)

[2.3.4 Mức độ ảnh hưởng 16](#_Toc186492043)

[2.3.5 Khuyến cáo khắc phục 16](#_Toc186492044)

[2.4 Kịch bản 3 rootme NoSQL Injection - Blind: 16](#_Toc186492045)

[2.4.1 Mô tả lỗ hổng 17](#_Toc186492046)

[2.4.2 Nguyên nhân dẫn đến lỗ hổng 17](#_Toc186492047)

[2.4.3 Các bước tấn công 17](#_Toc186492048)

[2.4.4 Khuyển cáo khắc phục 18](#_Toc186492049)

[2.5 Kịch bản 4: CVE 2021-22910 Rocketchat 19](#_Toc186492050)

[2.5.1 Mô tả lỗ hổng 19](#_Toc186492051)

[2.5.2 Nguyên nhân dẫn đến lỗ hổng 19](#_Toc186492052)

[2.5.3 Các bước tấn công 20](#_Toc186492053)

[2.5.4 Mức độ ảnh hưởng 23](#_Toc186492054)

[2.5.5 Khuyển cáo khắc phục 24](#_Toc186492055)

[2.6 Kịch bản 5: CVE-2024-28192 YourSpotify 24](#_Toc186492056)

[2.6.1 Mô tả lỗ hổng 24](#_Toc186492057)

[2.6.2 Nguyên nhân dẫn đến lỗ hổng 24](#_Toc186492058)

[2.6.3 Các bước tấn công 26](#_Toc186492059)

[2.6.4 Mức độ ảnh hưởng 27](#_Toc186492060)

[2.6.5 Khuyển cáo khắc phục 27](#_Toc186492061)

[2.7 Kịch bản 6: CVE-2021-22911 Rocketchat 28](#_Toc186492062)

[2.7.1 Mô tả lỗ hổng 28](#_Toc186492063)

[2.7.2 Nguyên nhân dẫn đến lỗ hổng 28](#_Toc186492064)

[2.7.3 Các bước tấn công 28](#_Toc186492065)

[2.7.4 Mức độ ảnh hưởng 30](#_Toc186492066)

[2.7.5 Khuyển cáo khắc phục 30](#_Toc186492067)

[Chương III: KẾT LUẬN 31](#_Toc186492068)

[Chương IV: TÀI LIỆU THAM KHẢO 32](#_Toc186492069)

[Chương V: GIẢI TRÌNH CHỈNH SỬA 33](#_Toc186492070)

**DANH MỤC HÌNH ẢNH**

Hình 1 Ví dụ một file JSON 6

Hình 2 Ví dụ Database trong mongoDBCompass 7

Hình 3 Top 10 OWASP 2017 và 2021 9

Hình 4 Mã nguồn searchUser 11

Hình 5 Giao diện trang login 12

Hình 6 Chuỗi injection để đăng nhập 13

Hình 7 Injection đăng nhập thành công 13

Hình 8 Chuỗi injection để đăng nhập 15

Hình 9 Giao diện trang posts 15

Hình 10 Injection tìm bài viết ẩn thành công 16

Hình 11 Challenge trên rootme 17

Hình 12 Kiểm thử $regex để check flag 18

Hình 13 File python để bruteforce flag 18

Hình 14 API users.list không xử lý query 20

Hình 15 Giao diện đăng nhập/đăng kí 21

Hình 16 Injection để lấy mã 2FA 21

Hình 17 Request để injection 22

Hình 18 Response từ server có thông tin mật của người dùng 22

Hình 19 Reset password admin bằng token lấy được 23

Hình 20 Đăng nhập vào admin thành công 23

*Hình 21 File middleware sử dụng hàm getUserFromField* 25

*Hình 22 hàm getUserFromField không xử lý value từ User* 25

*Hình 23 Object dạng JSON* 26

*Hình 24 Đăng nhập bằng tài khoản Spotify* 26

*Hình 25 Các thông tin của người dùng bị lộ* 27

*Hình 26 ID của người dùng bị lộ* 27

*Hình 27 getPasswordPolicy không xử lý input* 28

*Hình 28 thực hiện bruteforce từng ký tự của token* 29

*Hình 29 Payload để gửi tới API* 29

*Hình 30 Bruteforce thành công, đổi mật khẩu người dùng* 29

1. **TỔNG QUAN ĐỀ TÀI**
   1. **Tổng quan về NoSQL**
      1. **NoSQL là gì**

NoSQL viết tắt của “Not only SQL” hoặc “Non-SQL”, là một loại cơ sở dữ liệu không sử dụng mô hình bảng quan hệ như các hệ thống quản lý cơ sở dữ liệu quan hệ (Relational Database Management System-RDBMS) truyền thống.

Thay vào đó, NoSQL lưu trữ và quản lý dữ liệu theo nhiều mô hình khác nhau, như cặp khóa-giá trị (JSON, BSON), tài liệu, cột, hoặc đồ thị, giúp tăng khả năng mở rộng và hiệu suất cho các ứng dụng hiện đại.

A screen shot of a computer

Description automatically generated

Hình 1 Ví dụ một file JSON

MongoDB là một hệ quản trị cơ sở dữ liệu NoSQL mã nguồn mở, được thiết kế để lưu trữ và quản lý dữ liệu dưới dạng tài liệu (document). Thay vì sử dụng các bảng và hàng như trong cơ sở dữ liệu quan hệ (SQL), MongoDB sử dụng các collection và document.

Một số toán tử (operator) mà MongoDB sử dụng:

* **Toán tử so sánh:** $eq, $gt, $lt,…
* **Toán tử logic:** $and, $or,…
* **Toán tử mảng:** $all, $elemMatch, $size.
* **Toán tử phần tử:** $exists, $type.
* **Toán tử câp nhật:** $set, $unset, $inc.
* **Toán tử tập hợp:** $group, $sort,…

A screenshot of a computer

Description automatically generated

Hình 2 Ví dụ Database trong mongoDBCompass

* + 1. **Đặc điểm của NoSQL**

Các đặc điểm chính của NoSQL:

* **Không có cấu trúc cố định:** Không yêu cầu một schema (lược đồ) cố định, cho phép lưu trữ dữ liệu không đồng nhất và linh hoạt.
* **Khả năng mở rộng:** NoSQL thường được thiết kế để dễ dàng mở rộng theo chiều ngang (horizontal scaling), tức là có thể thêm nhiều máy chủ hơn để xử lý dữ liệu lớn.
* **Hiệu suất cao:** NoSQL tối ưu hóa cho việc truy vấn dữ liệu lớn với tốc độ nhanh, thường phù hợp với các ứng dụng có yêu cầu về hiệu năng cao, như các hệ thống phân tán và ứng dụng thời gian thực.
* **Mô hình dữ liệu đa dạng:** Có thể lưu trữ dữ liệu dưới nhiều dạng khác nhau như: Key-Value, Document, Column-family, Graph.
  + 1. **Ưu và nhược điểm của NoSQL**

Ưu điểm của NoSQL:

* **Linh hoạt trong cấu trúc dữ liệu:** Có thể dễ dàng thay đổi cấu trúc mà không phải sửa đổi lược đồ như trong SQL.
* **Tốc độ truy vấn cao:** Thường truy vấn dữ liệu nhanh hơn trong trường hợp xử lý dữ liệu phi cấu trúc hoặc bán cấu trúc.
* **Khả năng mở rộng tốt**: Thích hợp cho các ứng dụng web lớn và các hệ thống phân tán. Nhược điểm của NoSQL:
* **Không hỗ trợ tốt tính nhất quán mạnh:** Nhiều cơ sở dữ liệu NoSQL tuân theo mô hình CAP Theorem, trong đó thường chọn khả năng sẵn sàng (availability) và phân vùng (partition tolerance) thay vì tính nhất quán (consistency).
* **Thiếu chuẩn hóa:** Không có chuẩn chung như SQL, mỗi hệ thống NoSQL có thể có cách thức sử dụng và truy vấn khác nhau.
* **Phức tạp khi xử lý giao dịch:** Không hỗ trợ tốt các giao dịch phức tạp như trong các hệ thống RDBM
  + 1. **Trường hợp nên sử dụng NoSQL**

Các cơ sở dữ liệu dạng NoSQL thích hợp cho các trường hợp sau đây:

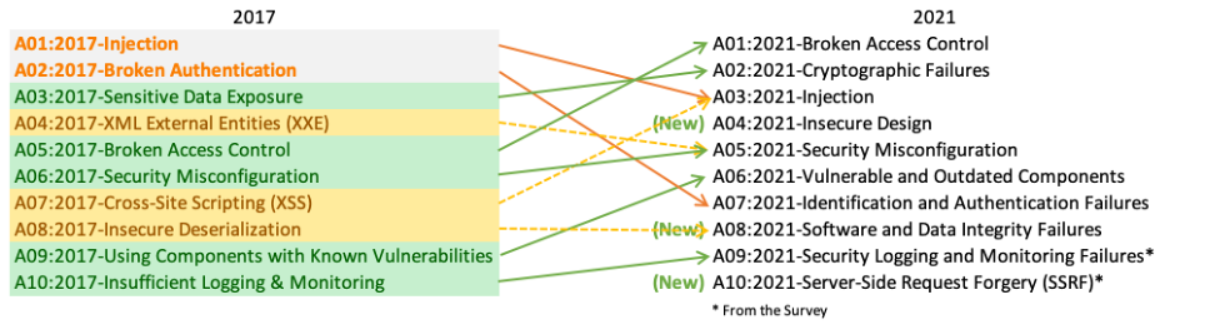
* Lưu trữ dữ liệu phi cấu trúc hoặc bán cấu trúc
* Yêu cầu khả năng mở rộng lớn
* Cần hiệu suất truy vấn cao
* Dữ liệu, cấu trúc dữ liệu thay đổi thường xuyên
* Các mô hình ứng dụng hiện đại như microservices và real-time streaming
  1. **NoSQL Injection**
     1. **NoSQL Injection là gì?**

NoSQL Injection là một dạng tấn công bảo mật, trong đó kẻ tấn công chèn mã độc vào các truy vấn NoSQL nhằm can thiệp vào hoạt động của cơ sở dữ liệu NoSQL.

Mục đích của tấn công này là lấy cắp dữ liệu, thay đổi dữ liệu trái phép hoặc phá hoại hệ thống.

NoSQL Injection tương tự như SQL Injection, nhưng nhắm vào các cơ sở dữ liệu NoSQL (như MongoDB, CouchDB, Cassandra) thay vì SQL.

NoSQL Injection nằm trong top 3 những lổ hổng bảo mật web nghiêm trọng nhất từ năm 2017 – 2021 theo thống kê của OWASP top 10.



Hình 3 Top 10 OWASP 2017 và 2021

* + 1. **Cách thức hoạt động của NoSQL Injection**

Nếu dữ liệu đầu vào từ người dùng không được kiểm tra và lọc kỹ, kẻ tấn công có thể chèn các câu truy vấn đã được chỉnh sửa nhằm truy vấn tới dữ liệu không mong muốn hoặc không được cho phép.

Câu truy vấn để thực hiện NoSQL Injection khác nhau tùy thuộc vào mỗi loại cơ sở dữ liệu khác nhau.

Đối với MongoDB, kẻ tấn công có thể thực hiện Injection bằng 2 cách sau:

* **Injection cú pháp (Syntax Injection):** Kẻ tấn công thực hiện thay đổi câu truy vấn bằng cách thêm/chỉnh sửa cú pháp khiến câu truy vấn của cơ sở dữ liệu thực hiện được, gọi tới các dữ liệu không mong muốn.
* **Injection các toán tử (Operator Injection):** Kẻ tấn công thực hiện thay đổi câu truy vấn bằng cách thêm/chỉnh sửa các toán tử được hỗ trợ khiến câu truy vấn của cơ sở dữ liệu thực hiện được, gọi tới các dữ liệu không mong muốn.
  + 1. **Hệ quả khi NoSQL Injection thành công**

Truy cập trái phép: Kẻ tấn công có thể bỏ qua xác thực để truy cập trái phép vào dữ liệu.

Rò rỉ dữ liệu: Các dữ liệu nhạy cảm của ứng dụng có thể bị lộ.

Thay đổi dữ liệu: Kẻ tấn công có thể chỉnh sửa hoặc xóa dữ liệu không mong muốn.

Phá hoại hệ thống: Thao tác không kiểm soát có thể gây lỗi hệ thống, ảnh hưởng đến tính toàn vẹn của dữ liệu*.*

* + 1. **Cách phòng chống NoSQL Injection**

**Kiểm tra và lọc đầu vào:** Lọc kỹ đầu vào của người dùng để ngăn chặn các ký tự đặc biệt hoặc toán tử.

**Sử dụng các thư viện an toàn:** Nhiều thư viện hoặc ORM hỗ trợ các truy vấn an toàn, giúp ngăn chặn các lệnh chèn mã độc.

**Hạn chế quyền truy cập:** Cấp quyền truy cập tối thiểu cho các tài khoản cơ sở dữ liệu để giảm thiểu thiệt hại khi bị tấn công. MongoDB có cơ chế IP Access List, chỉ cho phép một số IP có thể thực hiên truy vấn.

**Sử dụng mã hóa và xác thực mạnh:** Đảm bảo các yêu cầu xác thực được xử lý chặt chẽ và sử dụng cơ chế mã hóa để bảo vệ dữ liệu nhạy cảm.

* 1. **Mục tiêu đề tài**

Các thành viên trong nhóm hiểu cách thức hoạt động của NoSQL, hiểu được nguyên nhân, cách thức tấn công và cơ chế bảo mật đối với NoSQL Injection.

Nêu ra được 6 kịch bản tấn công lên cơ sở dữ liệu MongoDB với mức độ từ cơ bản đến nâng cao.

Nêu ra được các biện pháp khắc phục tương ứng, phù hợp với từng kịch bản tấn công.

1. **TRIỂN KHAI KỊCH BẢN**
   1. **Công cụ sử dụng**

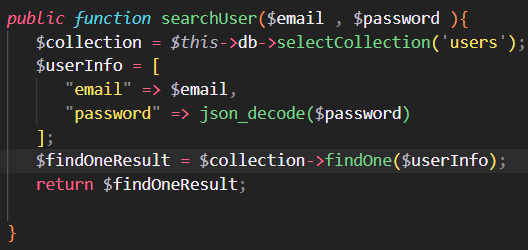
Các công cụ được sử dụng:

* VS Code
* BurpSuite Community
* MongoDBCompass 1.45.0
* XAMPP 3.3.0
* PHP 8.3.13
  1. **Kịch bản 1: Tấn công cơ chế xác thực (Web tự dựng)**
     1. **Mô tả lỗ hổng**

Tại trang login, các input của người dùng không được làm sạch nên kẻ tấn công có thể chèn các chuỗi injection để đăng nhập mà không cần biết mật khẩu người dùng.

* + 1. **Nguyên nhân dẫn đến lỗ hổng**

Mã nguồn trong file **/Lib/Dao.php** sử dụng hàm tìm kiếm không có cơ chế làm sạch input của người dùng, dẫn đến việc kẻ tấn công có thể chèn chuỗi injection để đăng nhập không cần sử dụng mật khẩu.



Hình 4 Mã nguồn searchUser

Ngoài ra ứng dụng còn sử dụng hàm json\_decode, gây mất an toàn nếu không kiểm tra đầu vào kĩ, giống với tấn công Deserialize.

* + 1. **Các bước tấn công**

1. Vào trang /login của trang web

A screenshot of a computer

Description automatically generated

Hình 5 Giao diện trang login

1. Nhập mail của user đã biết trước, sử dụng chuỗi “**{"$ne":"invalid"}**” làm mật khẩu

A screenshot of a login form

Description automatically generated

Hình 6 Chuỗi injection để đăng nhập

1. Đăng nhập thành công mà không cần biết mật khẩu

A close-up of a computer screen

Description automatically generated

Hình 7 Injection đăng nhập thành công

* + 1. **Mức độ ảnh hưởng**
* Mức độ nghiêm trọng: Cao
* Kẻ tấn công có thể đăng nhập bằng bất cứ tài khoản nào kể cả admin, dẫn đến việc ảnh hưởng đến sự bảo mật thông tin của toàn bộ người dùng.
  + 1. **Khuyến cáo khắc phục**
* **Sử dụng các framework an toàn**: không nên tự viết framework như trong kịch bản để tránh việc sử dụng cơ chế không an toàn hoặc các hàm không có cơ chế làm sạch đầu vào.
* **Làm sạch và xác thực dữ liệu đầu vào từ người dùng**: luôn kiểm tra đầu vào người dùng đã nhập, tránh các trường hợp sử dụng ký tự có thể tác động đến query của hệ thống.
* **Sử dụng tham số hóa trong truy vấn:** Tham số hóa các hàm liên quan đến các query nhằm tránh việc kẻ tấn công chèn các câu lệnh query độc hại vào trong câu truy vấn ban đầu.
* **Luôn nâng cấp lên phiên bản an toàn mới nhất:** thường xuyên cập nhật các bản vá liên quan đến bảo mật.
  1. **Kịch bản 2: Tấn công thanh tìm kiếm (Web tự dựng)**
     1. **Mô tả lỗ hổng**

Thanh tìm kiếm ở trang **/posts** bài viết trên trang web không làm sạch input của người dùng, dẫn đến việc kẻ tấn công có thể tìm kiếm các bài viết ẩn mà không cần biết tên bài viết.

* + 1. **Nguyên nhân dẫn đến lỗ hổng**

Mã nguồn tại **/Lib/Dao.php** chứa đoạn mã tìm kiếm không có cơ chế làm sạch input của người, từ đó kẻ tấn công có thể chèn chuỗi injection để tra toàn bộ thông tin tất cả các bài viết.

A screen shot of a computer code

Description automatically generated

Hình 8 Chuỗi injection để đăng nhập

Do sử dụng regex để tìm các bài viết và không có cơ chế lọc các kí tự đặc biệt từ người dùng nên người dùng có thể ghi các kí tự đặc biệt, ảnh hưởng đến query tìm kiếm.

* + 1. **Các bước tấn công**

1. Vào trang /posts sau khi đã đăng nhập.

A screenshot of a computer

Description automatically generated

Hình 9 Giao diện trang posts

1. Tìm kiếm chuỗi “.\*” vì trang web sử dụng regex để tìm kiếm thông tin. Tất cả các bài viết sẽ được in ra vì theo chuỗi Regex thì .\* sẽ trả kết quá đúng với mọi chuỗi

A screenshot of a computer

Description automatically generated

Hình 10 Injection tìm bài viết ẩn thành công

* + 1. **Mức độ ảnh hưởng**
* Mức độ nghiêm trọng: Trung bình
* Kẻ tấn công có thể lấy thông tin tất cả cá bài viết, kể cả các bài viết ẩn, từ đó có thể làm lộ thông tin bài viết sắp ra mắt, gây thiệt hại về tính bảo mật
  + 1. **Khuyến cáo khắc phục**
* **Sử dụng các framework an toàn**: không nên tự viết framework như trong kịch bản để tránh việc sử dụng cơ chế không an toàn hoặc các hàm không có cơ chế làm sạch đầu vào.
* **Làm sạch và xác thực dữ liệu đầu vào từ người dùng**: luôn kiểm tra đầu vào người dùng đã nhập, tránh các trường hợp sử dụng ký tự có thể tác động đến query của hệ thống.
* **Sử dụng tham số hóa trong truy vấn:** Tham số hóa các hàm liên quan đến các query nhằm tránh việc kẻ tấn công chèn các câu lệnh query độc hại vào trong câu truy vấn ban đầu.
* **Luôn nâng cấp lên phiên bản an toàn mới nhất:** thường xuyên cập nhật các bản vá liên quan đến bảo mật.
  1. **Kịch bản 3 rootme NoSQL Injection - Blind:** 
     1. **Mô tả lỗ hổng**
* Challenge check flag nhưng vẫn có thể sử dụng các toán tử như regex, kẻ tấn công có thể sử dụng để bruteforce từng ký tự của flag
  + 1. **Nguyên nhân dẫn đến lỗ hổng**
* Server không chặn việc sử dụng các operator của MongoDB, từ đó dẫn đến việc người dùng có thể bruteforce từng ký tự của flag
  + 1. **Các bước tấn công**

1. Truy cập vào challenge NoSQL Injection – Blind của rootme.

A screenshot of a computer

Description automatically generated

Hình 11 Challenge trên rootme

1. Thực hiện thay đổi thử tham số thành flag[$regex]=^3. Nhận thấy server trả về kết quả flag đúng, bây giờ cần bruteforce các ký tự còn lại bằng code python.

A screenshot of a computer

Description automatically generated

Hình 12 Kiểm thử $regex để check flag

1. Chạy code python sau

A computer screen with colorful text

Description automatically generated

Hình 13 File python để bruteforce flag

Vì flag thông thường có thêm 1 số ký tự đặc biệt nên ta thực hiện bruteforce tất cả các ký tự chữ in hoa và chữ thường, số và cộng thêm các kí tự đặc biệt, nếu có chữ Yeah ở kết quả trả về thì kí tự đúng, cộng thêm vào flag cho đến khi không còn tra thêm được ký tự nào

Kết quả flag: 3@sY\_n0\_5q7\_1nj3c710n

* + 1. **Khuy cáo khắc phục**
* **Sử dụng các framework an toàn**: không nên tự viết framework như trong kịch bản để tránh việc sử dụng cơ chế không an toàn hoặc các hàm không có cơ chế làm sạch đầu vào.
* **Làm sạch và xác thực dữ liệu đầu vào từ người dùng**: luôn kiểm tra đầu vào người dùng đã nhập, tránh các trường hợp sử dụng ký tự có thể tác động đến query của hệ thống.
* **Sử dụng tham số hóa trong truy vấn:** Tham số hóa các hàm liên quan đến các query nhằm tránh việc kẻ tấn công chèn các câu lệnh query độc hại vào trong câu truy vấn ban đầu.
* **Luôn nâng cấp lên phiên bản an toàn mới nhất:** thường xuyên cập nhật các bản vá liên quan đến bảo mật.
  1. **Kịch bản 4: CVE 2021-22910 Rocketchat**
     1. **Mô tả lỗ hổng**

**CVE-2021-22910**là một lỗ hổng bảo mật ứng dụng Rocket Chat với version 3.12.1, cho phép kẻ tấn công có thể khai thác Blind NoSQL Injection trên server mà không cần đăng nhập.

* + 1. **Nguyên nhân dẫn đến lỗ hổng**

Mã nguồn ở **/app/api/server/v1/users.js** không có cơ chế làm sạch query từ người dùng, dẫn đến việc kẻ tấn công có thể thực hiện lấy password reset token và mã của 2FA với điều kiện biết tên người dùng bằng toán tử $where.

A screen shot of a computer code

Description automatically generated

Hình 14 API users.list không xử lý query

Ngoài ra, các giá trị sort, fields, query được xử lý thông qua hàm parseJSONQuery() mà không kiểm tra đầu vào, gây mất an toàn nếu có injection xảy ra.

* + 1. **Các bước tấn công**

1. Đăng nhập vào một tài khoản vì các api chỉ cho phép tương tác với người dùng.

A screenshot of a login page

Description automatically generated

Hình 15 Giao diện đăng nhập/đăng kí

1. Thực hiện tạo một gói tin GET với body là **/api/v1/users.list?query={"$where":"this.username===admin+&&+(()=>{+throw+this.services.totp.secret+})()"}**, Server sẽ trả về mã 2FA tương ứng

A black background with colorful text

Description automatically generated

Hình 16 Injection để lấy mã 2FA

A screenshot of a computer program

Description automatically generated

Hình 17 Request để injection

A screenshot of a computer code

Description automatically generated

Hình 18 Response từ server có thông tin mật của người dùng

1. Đăng nhập với mật khẩu mới reset (thực hiện bằng code)

Screenshots of a computer screen

Description automatically generated

Hình 19 Reset password admin bằng token lấy được

A screenshot of a computer

Description automatically generated

Hình 20 Đăng nhập vào admin thành công

* + 1. **Mức độ ảnh hưởng**
* Mức độ nghiêm trọng: Cao
* Kẻ tấn công có thể dễ dàng lấy cả token reset password và mã 2FA của tất cả user, kể cả admin, từ đó gây mất an toàn với toàn hệ thống.
  + 1. **Khuyến cáo khắc phục**
* **Sử dụng các framework an toàn**: không nên tự viết framework như trong kịch bản để tránh việc sử dụng cơ chế không an toàn hoặc các hàm không có cơ chế làm sạch đầu vào.
* **Làm sạch và xác thực dữ liệu đầu vào từ người dùng**: luôn kiểm tra đầu vào người dùng đã nhập, tránh các trường hợp sử dụng ký tự có thể tác động đến query của hệ thống.
* **Sử dụng tham số hóa trong truy vấn:** Tham số hóa các hàm liên quan đến các query nhằm tránh việc kẻ tấn công chèn các câu lệnh query độc hại vào trong câu truy vấn ban đầu.
* **Luôn nâng cấp lên phiên bản an toàn mới nhất:** thường xuyên cập nhật các bản vá liên quan đến bảo mật
  1. **Kịch bản 5: CVE-2024-28192 YourSpotify**
     1. **Mô tả lỗ hổng**

**CVE-2024-28192** là một lỗ hổng bảo mật của trang YourSpotify version 1.8.0 trở xuống, tạo ra lỗ hổng NoSQL Injection trong logic xử lý public access token. Cho phép kẻ tấn công có thể vượt qua cơ chế xác thực public access token bất kể là token đó đã được tạo ra trước đó hay chưa.

* + 1. **Nguyên nhân dẫn đến lỗ hổng**

Mã nguồn tại **/server/src/tools/middleware.ts** sử dụng hàm “getUserFromField” không an toàn.

A screen shot of a computer program

Description automatically generated

*Hình 21 File middleware sử dụng hàm getUserFromField*

Hàm getUserFromField lấy value ( tức token của người dùng) từ request mà không làm sạch.

A computer screen shot of a program code

Description automatically generated

*Hình 22 hàm getUserFromField không xử lý value từ User*

Ngoài ra, Express.js cho phép tạo object bằng query ?param[key]=value. Từ đó sẽ tạo thành object có dạng sau:

A computer code with red text

Description automatically generated with medium confidence

*Hình 23 Object dạng JSON*

Điều này dẫn đến việc có thể thay chuỗi injection thành “/me?token[$ne] = DOESNOTEXIST“ để lấy token. Hoặc sử dụng lên đường dẫn /accounts để lấy id của toàn bộ người dùng, dẫn đến CVE-2024-28193

* + 1. **Các bước tấn công**

1. Đăng nhập vào một tài khoản.

A screenshot of a computer

Description automatically generated

*Hình 24 Đăng nhập bằng tài khoản Spotify*

1. Sau đó vào đường dẫn **/me?token[$ne]=DOESNOTEXIST** ở API endpoint (locahost:8080) để lấy access token

A screenshot of a computer

Description automatically generated

*Hình 25 Các thông tin của người dùng bị lộ*

1. Vào đường dẫn **/accounts?token[$ne]=DOESNOTEXIST** để lấy id toàn bộ người dùng

A screenshot of a computer

Description automatically generated

*Hình 26 ID của người dùng bị lộ*

* + 1. **Mức độ ảnh hưởng**
* Mức độ nghiêm trọng: Cao
* Kẻ tấn công có thể lấy được id của tất cả người dùng, dẫn đến CVE-2024-28193, từ đó ăn cắp session của người dùng thông qua jwt token.
* Từ session có thể đổi mật khẩu người dùng, đánh cắp tài khoản.
  + 1. **Khuyến cáo khắc phục**
* **Sử dụng các framework an toàn**: không nên tự viết framework như trong kịch bản để tránh việc sử dụng cơ chế không an toàn hoặc các hàm không có cơ chế làm sạch đầu vào.
* **Làm sạch và xác thực dữ liệu đầu vào từ người dùng**: luôn kiểm tra đầu vào người dùng đã nhập, tránh các trường hợp sử dụng ký tự có thể tác động đến query của hệ thống.
* **Sử dụng tham số hóa trong truy vấn:** Tham số hóa các hàm liên quan đến các query nhằm tránh việc kẻ tấn công chèn các câu lệnh query độc hại vào trong câu truy vấn ban đầu.
* **Luôn nâng cấp lên phiên bản an toàn mới nhất:** thường xuyên cập nhật các bản vá liên quan đến bảo mật
  1. **Kịch bản 6: CVE-2021-22911 Rocketchat**
     1. **Mô tả lỗ hổng**

**CVE-2021-22911**là một lỗ hổng bảo mật ứng dụng Rocket Chat với version 3.12.1, cho phép kẻ tấn công có thể khai thác Blind NoSQL Injection, bruteforce từng ký tự của mật khẩu trên server mà không cần đăng nhập.

* + 1. **Nguyên nhân dẫn đến lỗ hổng**

Mã nguồn ở /**server/methods/getPasswordPolicy.js** không sử dụng cơ chế làm sạch, dẫn đến việc kẻ tấn công có thể sử dụng {"$regex":"^A"} để tra từng ký tự của reset password token.

A computer screen with text

Description automatically generated

*Hình 27 getPasswordPolicy không xử lý input*

* + 1. **Các bước tấn công**

1. Tạo một file python tấn công như sau

A black background with colorful lines

Description automatically generated

*Hình 28 thực hiện bruteforce từng ký tự của token*

*A computer code with black text

Description automatically generated*

*Hình 29 Payload để gửi tới API*

1. Kết quả sau khi chạy code

A screen shot of a computer

Description automatically generated

*Hình 30 Bruteforce thành công, đổi mật khẩu người dùng*

* + 1. **Mức độ ảnh hưởng**
* Mức độ nghiêm trọng: Cao
* Kẻ tấn công có thể lấy được reset password token của một user nào đó với điều kiện ít user. Vì chuỗi tấn công so sánh với các token nên càng ít user, càng chắc chắn token đó thuộc về user nhất định.
  + 1. **Khuyến cáo khắc phục**
* **Sử dụng các framework an toàn**: không nên tự viết framework như trong kịch bản để tránh việc sử dụng cơ chế không an toàn hoặc các hàm không có cơ chế làm sạch đầu vào.
* **Làm sạch và xác thực dữ liệu đầu vào từ người dùng**: luôn kiểm tra đầu vào người dùng đã nhập, tránh các trường hợp sử dụng ký tự có thể tác động đến query của hệ thống.
* **Sử dụng tham số hóa trong truy vấn:** Tham số hóa các hàm liên quan đến các query nhằm tránh việc kẻ tấn công chèn các câu lệnh query độc hại vào trong câu truy vấn ban đầu.
* **Luôn nâng cấp lên phiên bản an toàn mới nhất:** thường xuyên cập nhật các bản vá liên quan đến bảo mật.

1. **KẾT LUẬN**

* Nhóm 06 đã thực hiện thành công 6 kịch bản NoSL injection, đánh giá mức độ nghiêm trọng và đề ra các giải pháp khắc phục.
* Thông qua quá trình thực hiện đồ án, các thành viên trong nhóm đã nắm được kiến thức cơ bản về NoSQL database và tấn công NoSQL Injection, từ đó nhận thấy các lỗ hổng vẫn còn tồn tại ở các dự án, chủ yếu do không xử lý input đúng cách, sử dụng các phiên bản dự án mã nguồn mở đã lỗi thời.
* Tuy gặp một số khó khăn nhất định, nhóm 6 đã thực hiện được muc tiêu ban đầu của đồ án

1. **TÀI LIỆU THAM KHẢO**

<https://portswigger.net/web-security/nosql-injection>

<https://cve.mitre.org/cve/search_cve_list.html>

<https://hackerone.com/>

https://www.root-me.org/

https://www.geeksforgeeks.org/introduction-to-nosql/

1. **GIẢI TRÌNH CHỈNH SỬA**

Sau buổi báo cáo cuối HK1 vào ngày 14/12/2024 với giảng viên Nguyễn Công Danh, nhóm 6 đã tiếp thu, tiến hành chỉnh sửa báo cáo theo ý kiến đề nghị của giảng viên và nhóm 6 xin giải trình như sau:

|  |  |  |
| --- | --- | --- |
| **STT** | **Góp ý của giảng viên** | **Vị trí được chỉnh sửa trong báo cáo** |
| **1** | **Bổ sung phần giải thích cho các kịch bản** | * 1. **Kịch bản 1**   Bổ sung nguyên nhân dẫn đến lỗ hổng và các bước tấn công  **Trang 10**   * 1. **Kịch bản 2**   Bổ sung nguyên nhân dẫn đến lỗ hổng và các bước tấn công  **Trang 13**   * 1. **Kịch bản 3**   Bổ sung nguyên nhân dẫn đến lỗ hổng và các bước tấn công  **Trang 16**   * 1. **Kịch bản 4**   Bổ sung nguyên nhân dẫn đến lỗ hổng và các bước tấn công  **Trang 18**   * 1. **Kịch bản 5**   Bổ sung nguyên nhân dẫn đến lỗ hổng và các bước tấn công  **Trang 23**   * 1. **Kịch bản 6**   Bổ sung nguyên nhân dẫn đến lỗ hổng và các bước tấn công  **Trang 27** |

***Nhóm 06 thực hiện***

---

**HẾT**

1. Ghi nội dung công việc [↑](#footnote-ref-2)