**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационной безопасности**

отчет

**по производственной практике**

**Тема: NOSQL injections. MongoDB.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7363 |  | Унщиков К.А. |
| Преподаватель |  | Племянников А.К. |

Санкт-Петербург

2020

1. **ЦЕЛЬ РАБОТЫ**

Ознакомиться с методами NOSQLинъекций. Создать веб сервис, который показывает, как эксплуатировать эту уязвимость. Научиться быстро находить и устранять эти уязвимости.

1. **ПОСТАНОВКА ЗАДАЧИ.**
   1. **Теоретическая часть**
      1. **Введение**

Инъекционные атаки считаются одними из самых опасных типов уязвимостей в существовании. Согласно списку уязвимостей веб-приложений OWASP Top десятка, инъекция является наиболее распространенным и опасным типом. В списке 259 наиболее опасных программных ошибок CWE 2019 года плохая санация входных данных (что позволяет использовать инъекционные атаки) заняла третье место.

Уязвимости инъекций существуют в различных контекстах, но все они сводятся к одной и той же проблеме. Несколько разных языков, будь то SQL, LDAP или оболочка Linux, смешивают код и данные вместе в командах.

Во многих случаях данные предоставляются ненадежными пользователями. Хотя язык обычно отличает код от данных, это различие обычно составляет пару одинарных или двойных кавычек. Если пользователь может закрыть набор кавычек (например, включив их в свои данные), он может получить часть своих данных, интерпретируемых как код и исполняемых программой.

Вот почему плохая очистка входных данных является основным фактором, способствующим инъекционным атакам. Если некоторые опасные символы (например, кавычки) запрещены или экранированы при вводе пользователем, их нельзя использовать для закрытия кавычек, которые очерчивают ввод пользователя в качестве данных. Однако многие разработчики не могут должным образом очистить ввод данных пользователем, оставляя свои приложения открытыми для атак.

* + 1. **Введение в NoSQL**

NoSQL (или нереляционные) базы данных не используют SQL для запросов. Существуют различные типы баз данных NoSQL, такие как:

* Key – Value
* Key – Document
* Column – Family
* Graph

NoSQL был разработан в ответ на переход к методологиям гибкого проектирования. Реляционные (SQL) базы данных требуют предварительного определения схем и не позволяют их обновлять, когда данные уже находятся в базе данных. Добавление новых полей требует переноса базы данных в новую схему, что может занять много времени.

NoSQL поддерживает определения динамических схем. Данные могут быть добавлены без четкого определения схемы, что позволяет базе данных работать в соответствии с циклами быстрой гибкой разработки. В отличие от реляционных баз данных, NoSQL также может легко масштабироваться по горизонтали с помощью встроенного автоматического шардинга. Это гарантирует, что команда разработчиков не будет ограничена доступным аппаратным обеспечением или вынуждена вручную внедрять шардинг.

* + 1. **Атака NoSQL**

Хотя базы данных NoSQL, такие как MongoDB, не используют SQL для запросов, они по-прежнему выполняют запросы на основе пользовательского ввода. Это означает, что они по-прежнему уязвимы для атак с использованием инъекций, если разработчик не выполняет надлежащую очистку входных данных.

Основное различие между внедрением SQL и NoSQL заключается в грамматике и синтаксисе запроса. Попытка выполнить NoSQL-инъекцию с помощью строки атаки SQL-внедрения вряд ли будет успешной. Базы данных NoSQL не имеют стандартизированного языка. Тем не менее, синтаксис их языков очень похож (так как они предназначены для того же).

Одним из приложений внедрения NoSQL является атака на веб-приложения, построенные на стеке MEAN (MongoDB, Express, Angular и Node). При передаче данных MEAN-приложения используют JSON, то же самое, что и MongoDB. Внедрение JSON-кода в MEAN-приложение может привести к внедрению атак на базу данных MongoDB.

Например, на рисунке 1 (от NullSweep ) показано приложение, уязвимое для внедрения NoSQL. Из запроса код извлекает имя пользователя и пароль и помещает их в запрос MongoDB без выполнения какой-либо очистки входных данных. Передавая JSON {«username»: «myaccount», «password»: {«$ ne»: 1}} , злоумышленник может пройти проверку подлинности в службе без пароля.

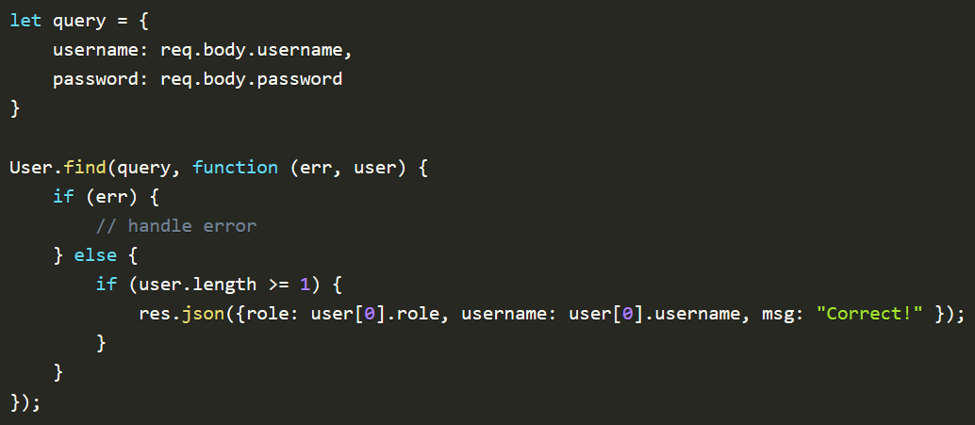


Рисунок 1 – Уязвимое приложение на стеке MEAN

* + 1. **Защита NoSQL**

Атаки с использованием NoSQL очень похожи на SQL-инъекции: они используют слабую очистку пользовательского ввода при построении запросов к базе данных. Это означает, что те же инструменты для защиты от атак внедрения SQL также работают для NoSQL, такие как:

* + - Никогда не строит запросы из строк
    - Использование типизированных входов
    - Использование входной библиотеки очистки для избежания опасных символов
    - Минимизация привилегий для общедоступных приложений

Однако при защите баз данных NoSQL необходимо принять дополнительные меры. Например, оценка JavaScript может быть отключена в MongoDB, уменьшая потенциальное влияние атаки с использованием инъекций. Чтение руководства и инструкций по безопасности для конкретной базы данных NoSQL - это всегда хорошая идея.

* 1. **NoSQL injection в MongoDB**

Да, MongoDB не поддерживает SQL, но СУБД не может обойтись без языка запросов. Разработчики MongoDB решили использовать вместо SQL популярнейший текстовый формат обмена данными JSON (BSON). Таким образом, можно попробовать осуществить разного рода атаки-инъекции (конечно, если входные данные не фильтруются). На рисунке 2 представлена инъекция.



Рисунок 2 – пример инъекции на языке Go

Здесь, вместо второго аргумента возможно внедрить nosql инъекция, например, если request у нас будет «{$ne:0}», тогда мы получим информацию не конкретного пользователя, а о всех.

В MongoDB запросах могут стать самыми опасными – служебные символы, такие как:

* $ne (не равно)
* $where (где)
* $regex (регулярное выражение)

Так, например, в RESTful приложении в url requests можно внедрять инъекции. Если запрос строится в виде: http://10.0.2.0:8090/nosql/database/users/?login=somelogin&pass=somepass.

Мало того, что вместо «somepass» мы можем ввести {$ne:1} и значение автоматически будет true, так мы еще можем ввести вместо «somelogin» - «admin#», т.к. данные после решетки не читаются, то мы войдем под админкой на сайт (Если конечно, данные не проходят проверку)

* 1. **Решение задачи.**

Мне необходимо создать веб-сервис, в котором будет NoSQl инъекция с последующим её устранением. Для этого задания я выбрал Golang+MongoDB+HTML+Gorilla Package(handlers, static files). Для начала я создал кластер на сайте mongoDB