Уметалиев Алишер

Go Review Code для Cloud.ru

\*\*• **Какие уязвимости присутствуют в этом фрагменте кода?\*\***

- В данном коде присутствует несколько уязвимостей, поэтому, я буду сообщать о них по убыванию их критичности:

1. **SQL-Injection**:

- При первом изучении кода заметил уязвимый процесс работы с query (стр. 38). На данной строчке видно, что searchQuery вставляется в чистом виде, без каких-либо фильтров и изменений.

2. Неправильная конфигурация и работа с исключениями:

- “\*\*log.fatal\*\*” в различных функциях. Это очень рисковый способ работы с исключениями, поскольку его срабатывание может прервать работу всего сервера. Для лучшей отладки рекомендуется использовать “log. PrintLn”, который позволит более детально логировать ошибку, а так же сообщить о ней клиенту сервиса.

\*\*• Указать строки, в которых присутствуют уязвимости.\*\*

- SQL-Injection – 38-ая строка

query := fmt.Sprintf("SELECT \* FROM products WHERE name LIKE '%%%s%%'", searchQuery)

rows, err := db.Query(query)

- Неправильная конфигурация и работа с исключениями – 17, 22, 52, 66.

 log.Fatal(err)

\*\*• **К каким последствиям может привести эксплуатация найденных уязвимостей злоумышленником?\*\***

- SQL-Injection является одной из самых базовых, однако одной из самых опасных с точки зрения конфиденциальности данных, согласно OWASP. При успешной эксплуатации этой уязвимости, у злоумышленника будет доступ ко всем корпоративным данным компании, включая личные данные сотрудников и клиентов. Однако, хакер может пойти дальше и пробросить RCE (Remote Code Execution) для доступа во внутреннюю инфраструктуру компании. А это уже влечет за собой еще более печальные последствия.

- При неправильном логировании и работе с исключениями и у разработчиков, и у клиентов возникают проблемы. Связаны они больше с тем, что ресурсы серверов будут распределяться неэффективно, из-за излишнего потребления трафика со стороны базы данных и ее функции «initDB», а это будет приводить к упадкам серверов и недовольным потребителям.

\*\*• **Описать способы исправления уязвимостей.\*\***

- Требуется интегрировать параметризованные запросы, чтобы база данных имела возможность различить входящий код и данные:

query := "SELECT \* FROM products WHERE name LIKE ?"

rows, err := db.Query(query, "%"+searchQuery+"%")

- Использование Whitelist списка особых символов способствует предотвратить символы, подобные ‘, ; #, --, которые могут использоваться как один из векторов атак на базы данных.

- Практика экранирования пользовательских данных изолирует символы друг от друга непосредственно перед их вставкой в запрос, что позволяет не допустить SQL атак.

query\_search := http.Request.URL.Query().Get("query")

query\_safe := fmt.Sprintf("SELECT \* FROM products WHERE name LIKE '%%%s%%'", sql.QueryEscape(searchQuery))

- Библиотеки с автоматической проверкой SQL-атак, такие как ORM & GORM или sqlX обеспечивают автоматическую проверку отправляемых запросов, посредством динамического сравнения.

- Рекомендуется избегать использования «\*\*log.fatal\*\*» , если нет необходимого случая. Это поможет сохранить стабильность сервисов, а также минимизировать риски утечки данных, когда initDB открыл соединение в более крупных проектах.

\*\*

• **Если уязвимость можно исправить несколькими способами, необходимо перечислить их, выбрать лучший по вашему мнению и аргументировать свой выбор**.\*\*

Каждый из описанных методов устранения уязвимостей представляет собой важный элемент в обеспечении безопасности, но в условиях постоянно развивающегося ландшафта киберугроз, опираться на единственный подход было бы недальновидно. Комплексное применение предложенных рекомендаций образует многоуровневую систему защиты, которая способна эффективно противостоять разнообразным векторам атак и обеспечить надежную защиту корпоративных данных и ресурсов.Тем не менее, в ситуации ограниченного бюджета или ресурсов, приоритетом должна стать имплементация параметризованных запросов. Данный подход не только обеспечивает значительное повышение безопасности против SQL-инъекций, но и может быть реализован без глубокой переработки существующей архитектуры системы и при минимальных затратах ресурсов. Это делает его идеальным первым шагом на пути к созданию защищенного цифрового пространства.