004.056.53

В.Г. Хомишин, аспірант
Тернопільський національний технічний університет імені Івана Пулюя, Україна

ЗАХИСТ ASP.NET CORE ВЕБ-ДОДАТКІВ ВІД ВПРОВАДЖЕННЯ SQL-КОДУ

V.G. Khomyshyn

PROTECTING ASP.NET CORE WEB APPLICATIONS FROM SQL CODE INJECTION

Атаки шляхом впровадження SQL коду є однією з найнебезпечніших загроз для застосування. Зловмисники створюють простий шкідливий код, який вони відправляють у наш додаток у вигляді традиційного введення на основі форм або шляхом налаштування URL-адрес і рядків запиту для виконання довільного коду у нашій базі даних. Ця вразливість може надати доступ зловмисникам до всієї нашої бази даних, тому дуже важливо знаходити та усувати будь-які подібні вразливості у своїх додатках. SQL має архітектурну проблему, характерну для багатьох мов запитів: дані та команди знаходяться в одному рядку. Якщо SQL-запит є результатом поєднання рядків, деякі з яких були введені користувачем, це може бути фактором ризику. Розглянемо два найпоширеніших шляхи вирішення проблеми впровадження SQL коду:

1. Використання підготовлених інструкцій. Впровадження SQL-коду працює, тому що зловмиснику вдається переключити контекст усередині SQL. Варто пам'ятати, що інструкція SQL – це рядок із командами та даними. Екранування введення може бути перспективною стратегією. Коли ми поміщаємо дані користувача в SQL-запит, це також вважається вводом, тому це правило можна застосовувати. Проте виявляється, що правильно екранувати спеціальні символи SQL не так просто, тому що тут дуже багато варіантів залежно від контексту. Усередині рядків на думку приходять одинарні лапки (і заміна їх на \' або ', залежно від бази даних, що використовується).

Щодо інших спеціальних символів: ( ) [ ] – групування, _ % – підстановочні знаки; ; – завершення запиту; -- # /* – коментар, – все залежить від конкретної системи баз даних.

Варіантів кілька, тож тут легко помилитися. Чому б не дозволити комусь іншому зробити всю тяжку роботу? Виявляється, що всі відповідні баз даних для платформи .NET підтримують підготовлені інструкції (які ще називають параметризованими запитами). У них SQL-запит відправляється до бази даних у два етапи. Спочатку створюється інструкція, яка може містити параметри. На другому етапі ці параметри набувають значень. Лише після цього інструкція виконується. Це простий, але ефективний підхід, тому що він нейтралізує проблему SQL архітектури: тепер зрозуміло, де команда, а де дані. Надаючи значення цим заповнювачам, база даних знає, що подана інформація повинна розглядатися лише як дані. Насправді, так ми відключили використання небажаного SQL-коду. Ось як може виглядати запит із параметрами:

```
SELECT * FROM users WHERE email=@email AND password=@password
```

Зверніть увагу, що перед параметрами стоїть символ @, а лапки навколо параметрів відсутні (інакше імена параметрів використовувалися буквально, як рядки). Існують різні варіанти синтаксису для підготовлених інструкцій. Той, що ми використовували досі (з префіксом @), є найбільш поширеним і зручним, оскільки кожен параметр може мати унікальне ім'я, що запам'ятовується. Також для позначення параметрів можна використовувати знак запитання (?), а потім надавати значення за розташуванням, а не по імені. Очевидно, що такий варіант більш схильний до помилок, оскільки розташування параметра буде змінюватися, як тільки десь перед ним буде доданий інший параметр.

Відомий виняток – СУБД Oracle, оскільки імена параметрів тут починаються з двокрапки, і інструкція SQL може виглядати так:

```
SELECT * FROM users WHERE email=:email AND password=:password
```

Протягом тривалого часу найпоширенішим способом надсилання SQL-запитів до бази даних із додатку ASP.NET була технологія ADO.NET (Advanced Data Objects для .NET). Цей підхід також працює з ASP.NET Core. Що стосується Microsoft SQL Server, пакет Microsoft.Data.SqlClient (www.nuget.org/packages/Microsoft.Data.SqlClient) містить постачальника даних. Є аналогічні пакети й інших основних баз даних, але їх способи реалізації підготовлених інструкцій переважно схожі. Якщо ми дійсно працюємо з SQL, то підготовлені інструкції – мабуть, найкращий засіб захисту від впровадження SQL-коду. Однак можна взагалі не писати SQL-код.

2. Використання Entity Framework Core. Щоб позбавити розробників писати SQL-команди, компанія Microsoft пропонує інструмент Entity Framework (EF) Core. Це проста, кросплатформова версія популярної технології доступу до даних, яка дозволяє розробникам .NET працювати з базою даних за допомогою об'єктів .NET та усуває потребу у більшій частині коду для доступу до даних, який, зазвичай, доводиться писати. У EF Core доступ до даних здійснюється за допомогою моделі. Модель складається з класів сутностей та об'єкта контексту, що представляє сеанс взаємодії з базою даних. Об'єкт контексту дозволяє виконувати запити та зберігати дані. Псевдокод, який буде запитувати всіх користувачів з бази даних, щоб знайти тих, хто має певну комбінацію адреси електронної пошти та пароля, має вигляд:

```
db.Users.Where(user => user.email == email && user.password == password)
```

Цей код робить майже те саме, що і SQL з попередніх прикладів, за винятком деяких особливих випадків, таких як бази даних з урахуванням регістру. Однак, тут немає конкатенації строк, а також відсутні параметри. Натомість є об'єкт, що позначає всіх користувачів в історії даних (db.Users) та синтаксис мови LINQ для вибору користувачів, які відповідають певним критеріям. У зловмисника немає можливості впровадити SQL, оскільки його немає. Ну, взагалі він є, але Entity Framework Core генерує його автоматично.

Але насправді це не зовсім повна картина. Entity Framework Core все ж таки дозволяє відправку SQL-запитів безпосередньо в базу даних. Це буде працювати аналогічно фрагменту коду, який ми щойно продемонстрували:

```
var users = db.Users.FromSqlRaw("SELECT * FROM users WHERE
  email='villain@example.com' AND password='noclue'").ToList();
```

Entity Framework Core не перевіряє SQL, він просто перенаправляється до бази даних. Якщо ми використовуємо тут конкатенацію рядків і дані користувача, то програма раптово знову стає вразливою для впровадження SQL-коду. Зручно, що є підтримка параметризованих запитів, де використовується синтаксис, аналогічний синтаксису методу платформи .NET, String.Format(). Але варто переконатись, що ми використовуємо його лише безпосередньо у виклику методу FromSqlRaw. Ця інструкція належним чином екранує значення для параметрів. В якості альтернативи можна використовувати інтерполяцію рядків.

Використання інструментів об'єктно-реляційного відображення, таких як Entity Framework Core, при всій своїй зручності, може створювати деякі проблеми. При їх застосуванні SQL-запити генеруються на основі наданої інформації (наприклад, об'єкта та його зв'язків). У деяких ситуаціях результат згубно впливає на продуктивність. Особливо це може статися, коли в одному запиті об'єднуються кілька таблиць. При роботі з таблицями без первинного ключа SQL може бути більш підходящим способом. Тому навіть найзатятіші прихильники Entity Framework Core завжди шукатимуть шляхи, в яких оригінальні SQL-запити є найкращим варіантом.