جلوگیری از حملات تزریق اس کیو ال

(SQL Injection Prevention)

تمرکز این مقاله بر ارائه راهنمایی عملی ، ساده و شفاف برای ممانعت از جریان‌های تزریق SQL در برنامه‌های کاربردی شما است. تزریق SQL متأسفانه بسیار رایج هستند و این دو دلیل دارد:

۱. رواج زیاد آسیب‌پذیری‌های تزریق SQL و
۲. جذابیت هدف(یعنی پایگاه داده معمولاً حاوی همه داده‌های مهم و جذاب برای برنامه کاربردی شما است)

این به‌نوعی شرم‌آور است که این‌همه حملات تزریق SQL موفق رخ می‌دهند، چراکه اجتناب از آسیب‌پذیری‌های SQL در کد شما، به‌شدت ساده است.

مشکل تزریق SQL این است که حملات تزریق SQL موفقی وجود دارند که پرس‌وجوهای پایگاه داده پویایی را ایجاد می‌کنند که شامل ورودی تأمین‌شده کاربر است. برای اجتناب از مشکل تزریق SQL، توسعه‌دهنده‌ها نیازمند: الف)نوشتن پرس‌وجوهای پویا را متوقف کنید؛ و/یا ب)از ورودی تأمین‌شده کاربر که حاوی SQL مخرب تأثیرگذار بر پرس‌وجوی اجراشده است، ممانعت به عمل می‌آورد.

این مقاله، مجموعه‌ای از تکنیک‌های ساده را برای ممانعت از آسیب‌پذیری‌های تزریق SQL با اجتناب از این دو مشکل، ارائه می‌کند. این تکنیک‌ها می‌توانند به‌ویژه توسط هر نوع زبان برنامه‌نویسی‌ای با هر نوع پایگاه داده‌ای استفاده شوند. انواع دیگر پایگاه داده‌ها نیز وجود دارند، ازجمله پایگاه داده‌های XML، که می‌توانند مشکلات مشابهی () داشته باشند و این تکنیک‌ها می‌توانند برای محافظت از آن‌ها نیز استفاده شوند.

دفاع‌های اولیه:

- گزینه ۱: استفاده از عبارات آماده(با پرس‌وجوهای پارامتر سازی شده)
- گزینه ۲: استفاده از روندهای ذخیره‌سازی شده
- گزینه ۳: اعتبار سنجی ورودی لیست سفید
- گزینه ۴: فرار[۱] از همه ورودی‌های تأمین‌شده کاربر

دفاع‌های اضافی:

- همچنین: تقویت حداقل امتیاز
- همچنین: اجرای اعتبارسنجی و ورودی لیست سفید به‌عنوان دفاع دوم

مثال غیر ایمن

نقایص تزریق SQL معمولاً به شکل زیر هستند:

مثال زیر(جاوا) UNSAFE است، و به مهاجم اجازه نخواهد داد تا کد را در پرس‌وجویی تزریق کند، که توسط پایگاه داده اجرا خواهد شد. پارامتر "customerName" غیر معتبر که به‌سادگی به پرس‌وجو اضافه‌شده بود و به مهاجم اجازه تزریق هر کد SQL که می‌خواهد را می‌دهد. متأسفانه، این روش برای ارزیابی پایگاه داده‌ها، بسیار رایج است.

```
String query = "SELECT account_balance FROM user_data WHERE user_name
= "
+ request.getParameter("customerName");


try {
Statement statement = connection.createStatement( ... );
ResultSet results = statement.executeQuery( query );
```

[۱] منظور همان Scape می باشد.

```
}
```

دفاع‌های اولیه

گزینه دفاع ۱: عبارات آماده‌شده (با پرس‌وجوهای پارامتر سازی شده)

استفاده از عبارات آماده‌شده با الزام متغیر(پرس‌وجوهای پارامتری سازی شده نیز نامیده می‌شود)عبارت است از اینکه همه توسعه‌دهنده‌ها باید اول بیاموزند که چگونه پرس‌وجوهای پایگاه داده را بنویسد. نوشتن آن‌ها ساده است و درک آن‌ها ساده‌تر از پرس‌وجوهای پویا. پرس‌وجوهای پارامتر سازی شده، توسعه‌دهنده را اول مجبور به تعریف کد SQL می‌کند، و سپس هر پارامتر را بعداً به پرس‌وجو ارسال می‌کند. این سبک کد نویسی، به پایگاه داده اجازه تشخیص بین کد و داده را صرف‌نظر از اینکه ورودی کاربر چه چیزی را تأمین کرده، می‌دهد.

عبارات آماده‌شده تضمین می‌کند که یک مهاجم نمی‌تواند هدف از پرس‌وجو را تغییر دهد، حتی اگر دستورات SQL توسط یک مهاجم درج‌شده باشند. در مثال ایمن زیر، اگر مهاجم نتواند userID تام یا "۱"=۱ را وارد کند، پرس‌وجوی پارامتر سازی شده آسیب‌پذیر نخواهد شد و در عوض به دنبال یک نام کاربری می‌گردد که به معنای واقعی کلمه، با رشته کلی تام یا "۱"=۱ تطبیق یافته است.

توصیه‌های خاص زبان:

- Java EE- از ()PreparedStatement با متغیرهای انقیاد استفاده می‌کند
- .NET- از پرس‌وجوهای پارامتر سازی شده‌ای همچون ()SqlCommand یا ()OleDbCommand با متغیرهای انقیاد استفاده می‌کند
- PHP- از PDO با پرس‌وجوهای پارامتر سازی شده پررنگ تایپ‌شده استفاده می‌کند (از ()bindParam استفاده می‌کند)

- به خواب رفتن[2] - از createQuery() با متغیرهای انقیاد استفاده می‌کند(پارامترهای Hibernate نامیده شده‌اند)
- SQLite- از sqlite3_prepare() برای ساخت یکشی عبارت استفاده کنید.

در شرایط نایاب، عبارات آماده‌سازی شده می‌توانند بر عملکرد صدمه بزنند. وقتی با این شرایط مواجه شد، بهترین راه این است که یا الف)همه داده‌ها را شدیداً اعتبار سنجی کنیم یا ب)از ورودی‌های تأمین‌شده کاربر را با استفاده از روال فرار ویژه برای فروشنده پایگاه داده‌تان به صورتی که در زیر توصیف‌شده فرار کنید، تا اینکه از یک عبارت آماده استفاده کنید.

مثال عبارت آماده‌سازی شده جاوای ایمن

مثال کد زیر از یک عبارت آماده‌سازی شده [3]استفاده می‌کند، پیاده‌سازی جاوا از یک پرس‌وجوی پارامتر سازی شده، برای اجرای پرس‌وجوی پایگاه داده مشابه.

```
String custname = request.getParameter("customerName"); // This should REALLY be validated too
// perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";

PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

مثال عبارت آماده‌سازی شده .NET. #C ایمن

[2] Hibernate
[3] PreparedStatement

با NET. حتی ساده‌تر نیز هست. ایجاد و اجرای پرس‌وجو، تغییر نمی‌کند. همه‌چیزی که باید انجام دهید، به‌سادگی عبور دادن پارامترها به پرس‌وجو با استفاده از فراخوانی the Parameters.Add() به شرح زیراست.

```
String query =
   "SELECT account_balance FROM user_data WHERE user_name = ?";
try {
OleDbCommand command = new OleDbCommand(query, connection);
command.Parameters.Add(new      OleDbParameter("customerName",
CustomerName Name.Text));
OleDbDataReader reader = command.ExecuteReader();
// …
} catch (OleDbException se) {
// error handling
}
```

ما مثال‌هایی را به جاوا و NET. نشان داده‌ایم، اما در عمل همه زبان‌های دیگر، ازجمله cold Fusion و Classic ASP از رابطه‌های پرس‌وجوی پارامتر سازی شده پشتیبانی می‌کنند. حتی لایه‌های انتزاع SQL، همچون زبان پرس‌وجوی خواب‌رفته(HQL) نیز نوع مشابهی از مشکلات تزریق را دارد (که ما آن را تزریق HQL می‌نامیم).

مثال‌های عبارت آماده‌سازی شده (پارامترها نامیده شده است) زبان پرس‌وجوی خواب‌رفته (HQL)

```
First is an unsafe HQL Statement

Query unsafeHQLQuery = session.createQuery("from Inventory where
productID='"+userSuppliedParameter+"'");
```

```
Here is a safe version of the same query using named parameters

Query safeHQLQuery = session.createQuery("from Inventory where
productID=:productid");
safeHQLQuery.setParameter("productid", userSuppliedParameter);
```

برای مثال‌های پرس‌وجوهای پارامتر سازی شده در سایر زبان‌ها، ازجمله Ruby, PHP, Cold Fusion و Perl، برگه تقلب پارامتر سازی شده پرس‌وجو یا http://bobby-tables.com را ببینید.

توسعه‌دهنده‌ها، تمایل به دوست داشتن رویکرد عبارت آماده‌سازی شده رادارند چراکه همه کدهای SQL در این کاربرد می‌مانند. این امر، برنامه کاربردی شمارا تقریباً مستقل از پایگاه داده می‌سازد.

گزینه دفاع۲: روندهای ذخیره‌سازی شده

روندهای ذخیره‌سازی شده همیشه از تزریق SQLدر امان نیستند. بااین‌حال، استاندارد مشخصی از ساختارهای برنامه‌ریزی روند ذخیره‌سازی شده، تأثیرات مشابهی همانند استفاده از پرس‌وجوهای پارامتر سازی شده را در زمان پیاده‌سازی ایمنی دارد، که یک‌روال عادی برای بیشتر زبان‌های روندی ذخیره‌شده می‌باشد. آن‌ها نیازمند این هستند که توسعه‌دهنده‌ها، فقط عبارات SQL را با پارامترهایی بسازند که به‌صورت خودکار پارامتر سازی شده‌اند مگر اینکه توسعه‌دهنده چیزی بزرگ‌تر از روال عادی انجام دهد. تفاوت بین عبارات آماده‌سازی شده و روندهای ذخیره‌سازی شده، این است که کد SQL برای روند ذخیره‌سازی شده، تعریف‌شده و در خود پایگاه داده ذخیره‌شده است و سپس از برنامه کاربردی فراخوانی شده است. هردوی این

تکنیک‌ها، اثربخشی مشابهی را در ممانعت از تزریق SQL دارند، بنابراین سازمان شما باید انتخاب کند که کدام رویکرد بیشترین معنا را برای شما دارد.

* نکته: ایمنی پیاده‌سازی شده، به معنای این است که روند ذخیره‌سازی شده شامل هیچ تولید SQL پویایی ناامنی نیست. توسعه‌دهنده‌ها، معمولاً SQL پویا را در روندهای ذخیره‌سازی شده تولید نمی‌کنند. بااین‌حال، می‌تواند انجام شود اما باید از آن اجتناب کرد. اگر امکان اجتناب از آن وجود نداشته باشد، روند ذخیره‌سازی شده باید از اعتبار سنجی ورودی یا فرار مناسب به صورتی که در این مقاله توصیف‌شده استفاده کند تا تضمین کند که همه ورودی‌های تأمین‌شده کاربر برای روند ذخیره‌سازی شده، نمی‌تواند برای تزریق کد SQL در پرس‌وجوی پویای تولیدشده، استفاده شود. حسابرس‌ها، باید همیشه به دنبال استفاده از sp_execute باشند، که اجرا یا اجرایی‌های درون روال‌های ذخیره‌سازی شده سرویس‌های SQL هستند. راهنماهای بازرسی مشابهی برای عملکردهای مشابه برای سایر فروشنده‌ها ضروری است.

همچنین موارد متعددی وجود دارند که در آن روندهای ذخیره‌سازی شده می‌توانند ریسک را افزایش دهند. برای مثال، در سرویس‌دهنده MS SQL، شما سه نقش پیش‌فرض اصلی دارید: db_datareader, db_datawriter و db_owner. قبل از آنکه روندهای ذخیره‌سازی شده مورداستفاده قرار گیرند، DBA حقوق db_datareader و db_datawriterحدودی را به سرویس‌دهنده وب برحسب نیازها، می‌دهند. بااین‌حال، روندهای ذخیره‌سازی شده نیازمند اجرای این حقوق هستند، نقشی که به‌صورت پیش‌فرض در دسترس نیست. برخی تنظیماتی که در آن مدیریت کاربر متمرکزشده است، بلکه به ۳ نقش محدودشده است، باعث می‌شود همه برنامه‌های کاربردی وب تحت حقوق db_owner به صورتی اجرا شود که روندهای ذخیره‌سازی شده بتوانند کار کنند. طبیعتاً، بدین معناست که اگر یک سرویس‌دهنده مهاجم را نقض کند، حقوق کامل در پایگاه داده را دارد، که در آن قبلاً ممکن است فقط دسترسی خواندن را داشته باشد. چیزهای بیشتری در مورد این عنوان را می‌توانید در این آدرس بخوانید. http://www.sqldbatips.com/showarticle.asp?ID=8

مثال روال ذخیره‌سازی شده جاوای ایمن

مثال کد زیر از یک CallableStatement استفاده می‌کند، پیاده‌سازی جاوای رابط روال ذخیره‌سازی شده، برای اجرای یک پرس‌وجوی پایگاه داده مشابه. روال ذخیره‌سازی شده "sp_getAccountBalance" می‌تواند در پایگاه داده از پیش تعریف‌شده باشد و عملکرد مشابهی را به‌صورت پرس‌وجوی تعریف‌شده بالا، پیاده‌سازی می‌کند.

```
String custname = request.getParameter("customerName"); // This should
                                                    REALLY be validated
                                                                try {
CallableStatement cs = connection.prepareCall("{call
                                                sp_getAccountBalance(?)}");
                                        cs.setString(1, custname);
                                ResultSet results = cs.executeQuery();
                                        // … result set handling
                                            } catch (SQLException se) {
                                    // … logging and error handling
                                                                }
```

مثال روال ذخیره‌سازی شده VB .NET ایمن

مثال کد زیر از یک SqlCommand استفاده می‌کند، پیاده‌سازی NET. از رابط روال ذخیره‌سازی شده، برای اجرای همان پرس‌وجوی پایگاه داده. روال ذخیره‌سازی شده "sp_getAccountBalance" باید در پایگاه داده از پیش تعریف‌شده باشد و عملکرد مشابهی را به‌صورت پرس‌وجوی تعریف‌شده بالا، پیاده‌سازی کند.

```
                                                                Try
Dim command As SqlCommand = new
                                        SqlCommand("sp_getAccountBalance", connection)
                        command.CommandType = CommandType.StoredProcedure
```

```
command.Parameters.Add(new      SqlParameter("@CustomerName",
                                                CustomerName.Text))
            Dim reader As SqlDataReader = command.ExecuteReader()
                                                ' ...
                                    Catch se As SqlException
                                        ' error handling
                                            End Try
```

گزینه دفاع ۳: اعتبارسنجی ورودی لیست سفید

بخش‌های پرس‌وجوهای SQL متعدد، موقعیت‌های قانونی برای استفاده از متغیرهای انقیاد ازجمله نام‌های جداول یا ستون‌ها و شاخص مرتبه مرتب‌سازی (ASC یا DESC) می‌باشد. در چنین شرایطی، اعتبارسنجی ورودی یا طراحی مجدد پرس‌وجو، مناسب‌ترین دفاع است. برای نام‌های جداول یا ستون‌ها، به‌صورت ایدئال، متغیرها از کد و از پارامترهای کاربر می‌آیند. اما اگر از مقادیر پارامتر کاربر برای ایجاد تمایز بین نام‌های جدول و نام‌های ستون استفاده شود، آنگاه مقادیر پارامتر بادی بر روی جدول قانونی/مورد انتظار یا نام‌های ستون‌ها برای ایجاد ورودی کاربر غیر معتبر استفاده شود که در پرس‌وجو تمام نمی‌شود. لطفاً توجه داشته باشید که، این‌یک نشانه از طراحی ضعیف است و یک بازنویسی کامل را باید در صورتی در نظر گرفت که زمان اجازه می‌دهد. این، مثالی از اعتبار سنجی نام جدول آورده شده است.

```
                                String tableName;
                                switch(PARAM):
                        case "Value1": tableName = "fooTable";
                                                break;
                        case "Value2": tableName = "barTable";
                                                break;
```

```
...
default     : throw new InputValidationException("unexpected value provided
for table name");
```

سپس tableName می‌تواند مستقیماً به پرس‌وجوی SQL افزوده شود، چراکه آن را حالا به‌عنوان یکی از مقادیر مورد انتظار یا قانونی برای نام جدول در این پرس‌وجو می‌شناسیم. به یاد داشته باشید که آن عملکردهای اعتبارسنجی جدول عمومی، می‌تواند منجر به از دست دادن داده‌ها شوند ، چراکه از نام‌های جدول در پرس‌وجوها استفاده می‌شود که در آن انتظار ندارند.

برای چیزی ساده همچون یک‌مرتبه مرتب‌سازی، بهترین چیز این است که ورودی تأمین‌شده توسط کاربر به یک Boolean تبدیل شود، و سپس از این‌Boolean برای انتخاب مقدار ایمن برای اضافه شدن به پرس‌وجو استفاده‌شده است. این‌یک نیاز بسیار استاندارد در ساخت پرس‌وجوی وب است. برای مثال:

```
public String someMethod(boolean sortOrder) {

String SQLquery = "some SQL ... order by Salary " + (sortOrder ? "ASC" :
"DESC");
...
```

هر ورودی کاربر زمانی‌ای می‌تواند به یک نوع بولین، عددی، تاریخ و غیره تبدیل شود. قبل از اینکه به یک پرس‌وجو افزوده شود، یا برای انتخاب یک مقدار برای افزوده شدن به پرس‌وجو استفاده‌شده باشد، تضمین‌کننده این است که انجام آن کار ایمن است یا نه.

اعتبارسنجی ورودی نیز به‌صورت دفاع ثانویه در همه موارد توصیه‌شده است، حتی وقتی‌که از متغیرهای انقیاد به صورتی استفاده می‌کنیم که بعدها در این مقاله موردبحث قرار می‌گیرد. تکنیک‌های بیشتری نیز در مورد نحوه پیاده‌سازی اعتبار سنجی ورودی لیست سفید قوی در برگه تقلب اعتبارسنجی ورودی، توصیف‌شده باشد.

گزینه دفاع ۴: فرار از همه ورودی‌های تأمین‌شده کاربر

این تکنیک نه‌تنها باید به‌صورت مرجع آخر استفاده شود، وقتی هیچ‌یک از موارد بالا امکان‌پذیر نیستند. اعتبارسنجی ورودی احتمالاً انتخاب بهتری است چراکه این فناوری در مقایسه با سایر دفاع‌ها شکننده است و ما نمی‌توانید تضمین کنیم که از همه تزریق‌های SQL در همه شرایط، ممانعت به عمل می‌آورد.

این تکنیک، فرار از ورودی کاربر، قبل از قرار دادن آن در یک پرس‌وجو است. پیاده‌سازی آن، بسیار ویژه پایگاه داده است. معمولاً وقتی اعتبارسنجی ورودی را پیاده‌سازی می‌کنیم، معمولاً فقط توصیه می‌شود که کدهای قدیمی را ترمیم کنید تا مقرون‌به‌صرفه باشد. برنامه‌های کاربردی از ابتدا ساخته‌شده، یا برنامه‌های کاربردی، نیازمند مقاومت ریسک پایین هستند که باید با استفاده از پرس‌وجوهای پارامتر سازی شده ، روندهای ذخیره‌سازی شده یا نگاشت‌های رابطه‌ای اشیای (ORM) بازنویسی یا ساخته شوند تا برایتان پرس‌وجو بسازند.

این تکنیک بدین‌صورت کار می‌کند. هر DBMS از الگوهای فرار کاراکتری ویژه انواع مشخصی از پرس‌وجوها پشتیبانی می‌کند. سپس اگر از ورودی تأمین‌شده کاربر با استفاده از الگوی فرار مناسب برای پایگاه داده‌ای که استفاده می‌کنید فرار کنید، DBMS آن ورودی را با کد SQL نوشته‌شده توسط توسعه‌دهنده اشتباه نمی‌گیرد، درنتیجه از هر آسیب‌پذیری تزریق SQL احتمالی‌ای اجتناب می‌کند.

API امنیت شرکتی OWASP (ESAPI) یک کتابخانه کنترل امنیت برنامه کاربردی وب است، که نوشتن کاربردهای با ریسک پایین را برای برنامه‌نویس‌ها ساده‌تر می‌کند. کتابخانه‌های ESAPI طراحی‌شده‌اند تا برای برنامه‌نویس‌ها، امنیت پیشرفته را در برنامه‌های کاربردی موجود، ساده‌تر کند. کتابخانه‌های ESAPI نیز به‌صورت بنیانی محکم برای توسعه جدید، خدمت‌رسانی می‌کنند.

- جزئیات کامل در ESAPI در اینجا در OWASP موجود است.
- Javadoc برای ESAPI 2.x (Legacy) در دسترس است. این کد، در نوامبر سال ۲۰۱۴ به GitHub منتقل‌شده بود.
- میراث ESAPI برای جاوا در GitHub به شناخت استفاده موجود از آن درزمانی که Javadoc به نظر ناکافی می‌رسد، کمک می‌کند.
- تلاش برای یک ESAPI دیگر برای Java GitHub رویکردهای دیگری دارد و آزمون یا کدگذاری‌های واقعی ندارد.

برای یافتن javadoc به‌ویژه برای رمزگذارهای پایگاه داده، بر روی کلاس Codec در سمت چپ کلیک کنید. کلی کدک پیاده‌سازی شده است. این دو کدک ویژه پایگاه داده عبارت‌اند از ,OracleCodec و .MySQLCodec. فقط بر روی نام‌های آن‌ها در "کلاس‌های پیاده‌سازی شناخته‌شده" : در بالای رابط صفحه کدک

در این زمان، ESAPI در حال حاضر کدگذارنده های پایگاه داده برای موارد زیر را دارد:

- اوراکل
- MySQL(ANSI و حالات بومی پشتیبانی می‌شوند)

رمزنگاران پایگاه داده در آینده برای موارد زیر خواهند بود:

- SQL Server
- PostgreSQL

اگر رمزنگار پایگاه داده شما ازدست‌رفته، لطفاً به ما خبر بدهید

جزئیات فرار ویژه پایگاه داده

اگر می‌خواهید روال‌های فرار خود را بسازید، اینجا جزئیات فرار برای هر یک از پایگاه داده‌ها وجود دارد که ما رمزنگارهای ESAPI را برای آن تدوین کرده‌ایم:

- Oracle
- SQL Server
- DB2

فرار از اوراکل

این اطلاعات مبتنی بر اطلاعات ویژگی فرار اوراکل است که می‌توانید در این آدرس بدان دست‌یابید:

http://www.orafaq.com/wiki/SQL_FAQ#How_does_one_escape_special_characters_when_writing_SQL_queries.3F

فرار از پرس‌وجوهای پویا

استفاده از کدک پایگاه داده ESAPI بسیار ساده است. یک مثال اوراکل به دنبال چیزی شبیه به مورد زیر می‌گردد:

```
ESAPI.encoder().encodeForSQL( new OracleCodec(), queryparam );
```

بنابراین اگر پرس‌وجوی پویای موجود خود که تولیدشده در کد خود رادارید، در اوراکل به شکل زیر خواهد بود:

```
String query = "SELECT user_id FROM user_data WHERE user_name = '" +
                                                    req.getParameter("userID")
                + "' and user_password = '" + req.getParameter("pwd") +"'";
                                                                    try {
                Statement statement = connection.createStatement( ... );
                ResultSet results = statement.executeQuery( query );
                                                                    }
```

شما خط اول را بازنویسی می‌کنید تا به شکل زیر باشد:

```
Codec ORACLE_CODEC = new OracleCodec();
String query = "SELECT user_id FROM user_data WHERE user_name = '" +
ESAPI.encoder().encodeForSQL( ORACLE_CODEC,
req.getParameter("userID")) + "' and user_password = '"
+ ESAPI.encoder().encodeForSQL( ORACLE_CODEC,
req.getParameter("pwd")) +"'";
```

و حالا صرف‌نظر از ورودی تأمین‌شده، از تزریق SQL ایمن خواهد بود.

برای قابلیت خواندن کد ماکسیممم، شما نیز می‌توانید OracleEncoder خود را بسازید.

```
Encoder oe = new OracleEncoder();
String query = "SELECT user_id FROM user_data WHERE user_name = '"
+ oe.encode( req.getParameter("userID")) + "' and user_password = '"
+ oe.encode( req.getParameter("pwd")) +"'";
```

با این نوعت راه‌حل، شما فقط نیازمند بسته‌بندی هر پارامتر تأمین‌شده کاربر است که به فراخوانی ESAPI.encoder().encodeForOracle( ) منتقل شود یا اینکه آیا شما فراخوانی نام‌گذاری کنید و آن را انجام خواهید داد.

خاموش کردن جایگزینی ویژگی

از Use SET DEFINE OFF و SET SCAN OFF برای تضمین این موضوع استفاده کنید که جایگزینی ویژگی اتوماتیک خاموش می‌شود. اگر این جایگزینی ویژگی روشن شود، با کاراکتر & همانند یک پیشوند متغیر رفتار خواهد شد که می‌تواند به مهاجم اجازه بازیابی داده‌های خصوصی را بدهد.

برای اطلاعات بیشتر، آدرس http://download.oracle.com/docs/cd/B19306_01/server.102/b14357/ch12040.htm#i2698854 and http://stackoverflow.com/questions/152837/how-to-insert-a-string-which-contains-an

فرار از شخصیت‌های اصلی در عبارت like

کلمه کلیدی LIKE امکان جستجوی بررسی متن را می‌دهد. در اوراکل، ویژگی زیرخط "_"فقط با یک کاراکتر تطبیق می‌یابد، درحالی‌که از "%" برای تطبیق صفر یا وقوع‌های هر کاراکتری استفاده‌شده است. این کاراکترها باید در معیار عبارت LIKE فرار کنند. برای مثال:

```
SELECT name FROM emp
WHERE id LIKE '%/_%' ESCAPE '/';
SELECT name FROM emp
WHERE id LIKE '%\%%' ESCAPE '\';
```

گریختن از اوراکل 10g

یک جایگزین برای اوراکل g۱۰ و بعدی، قرار دادن { و } حول این رشته برای فرار از کل رشته است. بااین‌حال، باید مراقب باشید که یک کاراکتر { در حال حاضر در رشته وجود نداشته باشد. باید به دنبال این‌ها بگردید و اگر یکی را پیدا کردید، آنگاه باید آن را با }} جایگزین کنید. در غیر این صورت، این کاراکتر اول به فاصله پایان می‌پذیرد و ممکن است یک آسیب‌پذیری را معرفی کند.

فرار MySQL

MySQL از دو حالت فرار پشتیبانی می‌کند:

۱. حالت ANSI_QUOTES SQL و یک حالت با خاموش شدن آن‌که آن را فراخوانی می‌کنیم
۲. حالت MySQL

حالت ANSI SQL: به‌سادگی همه کاراکترهای (تک تیک دار) را با (دو تیک جداگانه) رمزنگاری می‌کند

```
حالت MySQL، کارهای زیر را انجام می‌دهد:
NUL (0x00) --> \0  [This is a zero, not the letter O]
BS  (0x08) --> \b
```

```
TAB (0x09) --> \t
LF  (0x0a) --> \n
CR  (0x0d) --> \r
SUB (0x1a) --> \Z
"  (0x22) --> \"
%  (0x25) --> \%
'  (0x27) --> \'
\  (0x5c) --> \\
_  (0x5f) --> \_
all other non-alphanumeric characters with ASCII values less than 256  --> \c
where 'c' is the original non-alphanumeric character.
```

این اطلاعات، مبتنی بر اطلاعات ویژگی MySQL است که در اینجا یافت می‌شود: https://dev.mysql.com/doc/refman/5.7/en/string-literals.html

فرار از سرویس‌دهنده SQL

ما تاکنون روال فرار سرویس‌دهنده SQL را پیاده‌سازی نکرده‌ایم، اما در زیر اشاره‌گرهای خوب و ارتباطاتی به مقالاتی داریم که توصیف کنده این هستند که چگونه از حالت SQL بر سرویس‌دهنده SQL ممانعت به عمل‌آورند

- http://blogs.msdn.com/raulga/archive/2007/01/04/dynamic-sql-sql-injection.aspx

فرار DB2

این اطلاعات مبتنی بر کاراکترهای ویژه DB2 WebQuery هستند که در اینجا یافت می‌شوند: https://www-304.ibm.com/support/docview.wss?uid=nas14488c61e3223e8a78625744f00782983 و همچنین برخی اطلاعات در مورد محرک JDBC DB2 اوراکل را می‌توان در اینجا یافت: http://docs.oracle.com/cd/E12840_01/wls/docs103/jdbc_drivers/sqlescape.html

اطلاعات در رابطه با تفاوت‌های بین محرک‌های جهانی DB2 را می‌توانید در اینجا پیدا کنید:

رمزنگاری هگزای همه ورودی‌ها

یک مورد ویژه از فرار، عبارت است از فرایند کدگذاری هگزای کل رشته‌های دریافت شده از کاربر(این را می‌توانید به‌صورت فرار هر کاراکتر ببینید).برنامه کاربردی وب، باید ورودی کاربر را قبل از قرار دادن آن در عبارت SQL، به‌صورت هگزا کدگذاری کند. این عبارت SQL باید این واقعیت را در نظر بگیرد، و بر اساس آن داده‌ها را مقایسه کند. برای مثال، اگر ما به رکوردی منطبق با sessionID نگاه کنیم، و کاربر رشته abc123 را به‌عنوان IDنشست منتقل کند، عبارت انتخاب می‌تواند به شرح زیر باشد:

```
SELECT ... FROM session
WHERE hex_encode (sessionID) = '616263313233'
```

(کد هگزا باید توسط تسهیلات ویژه‌ای برای پایگاه داده مورداستفاده، جایگزین شود) رشته 606162313233، نسخه کدگذاری شده هگزای رشته دریافتی از کاربر است (دنباله مقادیر هگزای کدهای ASCII/UTF-8 از داده‌های کاربر است).

اگر مهاجم مجبور به منتقل نمودن رشته‌ای شود که حاوی یک کاراکتر تک نقل‌قول پس از تلاش آن‌ها برای تزریق کد SQL است شود، عبارت SQL ساخته‌شده به‌صورت زیر خواهد شد:

```
WHERE hex_encode ( ... ) = '2720 ... '
```

۲۷، کد اسکی تک نقل‌قول است، که به‌سادگی همانند هر کاراکتر دیگری در این رشته، کدگذاری هگزا شده است. SQL حاصل، می‌تواند فقط حاوی ارقام عددی و حروف a تا f شود، و هیچ‌وقت یک کاراکتر ویژه وجود ندارد که بتواند یک تزریق SQL شود.

فرار SQLi در PHP

استفاده از عبارات آماده‌سازی شده و پارامتر سازی شده. این‌ها، عبارات SQL هستند که ارسال‌شده‌اند و توسط سرویس‌دهنده پایگاه داده به‌صورت جداگانه از هر پارامتری، تفسیر شده‌اند. بدین‌صورت، برای مهاجم، تزریق SQL مخرب امکان‌پذیر است.

شما در اصل برای رسیدن به این، دو گزینه دارید:

۱. استفاده از PDO (برای هر محرک پایگاه داده پشتیبانی شده‌ای):

```
$stmt = $pdo->prepare('SELECT * FROM employees WHERE name = :name');

$stmt->execute(array('name' => $name));

foreach ($stmt as $row) {
// do something with $row
}
```

۲. استفاده از MySQLi (برای MySQL):

```
$stmt = $dbConnection->prepare('SELECT * FROM employees WHERE name = ?');
$stmt->bind_param('s', $name);

$stmt->execute();

$result = $stmt->get_result();
while ($row = $result->fetch_assoc()) {
// do something with $row
}
```

PDO یک گزینه جهانی است. اگر شما به پایگاه داده‌ای جز MySQL متصل می‌شوید، می‌توانید به یک گزینه دوم ویژه محرک نیز اشاره‌کنید(مثلاً pg_prepare() و pg_execute() برای PostgreSQL).

دفاع‌های اضافی

فراتر از پذیرش یکی از چهار دفاع اصلی، ما نیز تطبیق همه این دفاع‌های اضافی را برای فراهم آوری دفاع عمقی، پیشنهاد می‌کنیم. این دفاع‌های اضافی عبارت‌اند از:

- حداقل اولویت
- اعتبار سنجی ورودی لیست سفید

حداقل اولویت

برای حداقل سازی آسیب احتمالی یک حمله تزریق SQL موفق، باشد اولویت‌های تخصیص داده‌شده به هر حساب پایگاه داده را در محیط خود، حداقل سازی کنید. DBA یا دسترسی نوع admin را به حساب‌های کاربردی خود ندهید. ما می‌فهمیم که این ساده است، و همه‌چیز فقط وقتی درست‌کار می‌کنند که بدین‌صورت کار را انجام دهید، اما بسیار خطرناک است. از ابتدا آغاز کنید تا تعیین کنید که حساب‌های کاربردی شما نیازمند جه حقوق دسترسی‌ای هستند، به‌جای اینکه تلاش کنید تا دریابید که نیاز به چه حقوق دسترسی‌ای برای بردن هستید. مطمئن شوید که این حساب‌ها فقط نیازمند دسترسی خواندن هستند که فقط دسترسی خواندن را به جداولی می‌دهند که نیازمند دسترسی به آن هستید. اگر یک حساب فقط نیازمند دسترسی به بخش‌های جدول باشد، ایجاد یک دیدگاه را در نظر بگیرید که دسترسی به آن بخش از داده‌ها را محدود می‌کند و دسترسی حساب را به دیدگاه تخصیص می‌دهد تا به جدول زیربنایی. به‌ندرت، تاکنون، ساخت امتیاز یا حذف دسترسی به حساب‌های پایگاه داده وجود دارد.

اگر سیاستی را به کار ببرید که در آن از روندهای ذخیره‌سازی در هرجایی استفاده کنید و اجازه اجرای مستقیم پرس‌وجوهای خود را به برنامه کاربردی ندهید، آنگاه

آن حساب‌ها را به مواردی محدود کنید که فقط بتوانند روندهای موردنیاز را اجرا کنند. به آن‌ها حقوق مستقیم به جداول را در پایگاه داده، ندهید.

تزریقSQL، تنها تهدید برای داده‌های پایگاه داده شما نیست. مهاجمان می‌توانند به‌سادگی مقادیر پارامترها را از یکی از مقادیر قانونی‌ای که با آن نمایش داده‌شده‌اند، به مقداری تغییر دهند که برای آن‌ها غیرمجاز است، اما این برنامه کاربردی خودش ممکن است مجاز به دسترسی باشد. همچنین حداقل سازی اولویت‌های داده‌شده به برنامه کاربردی شما، احتمال چنین تلاش‌های دسترسی غیرمجازی را کاهش خواهد داد، حتی وقتی‌که یک مهاجم تلاش نمی‌کند تا از تزریق SQL به‌عنوان بخشی از سوءاستفاده آن‌ها بهره ببرد.

درحالی‌که در آن هستید، باید اولویت‌های حساس سیستم عملیاتی‌ای را حداقل سازی کنید که DBMS تحت آن‌ها اجرا می‌شود. DBMS خود را به‌عنوان ریشه یا سیستم اجرا نکنید!بیشترDBMS -ها با حساب سیستم بسیار قدرتمند، جعبه‌هایشان تمام‌شده. برای مثال، MySQL به‌صورت سیستمی بر روی ویندوز به‌صورت پیش‌فرض اجرا می‌شود! حساب DBMS OS را به چیزی مناسب‌تر با اولویت‌های محدودشده، تغییر دهید.

چند کاربر DB

طراح برنامه‌های وب، نباید از استفاده از یک حساب مالک/ادمین در برنامه‌های کاربردی اجتناب کنند تا به پایگاه داده وصل شوند. کاربران DB متعددی، می‌توانند باری برنامه‌های کاربردی وب متفاوت، استفاده شوند. به‌صورت کلی، هر برنامه کاربردی وب جداگانه‌ای که نیازمند دسترسی به پایگاه داده است، می‌تواند یک حساب کاربردی پایگاه داده اختصاصی داشته باشد که برنامه کاربردی وب از آن برای اتصال به DB استفاده خواهد کرد. بدین‌صورت، طراح برنامه کاربردی می‌تواند سطح مناسبی برای کنترل دسترسی داشته باشد، درنتیجه اولویت‌ها را تاجای ممکن، کاهش دهد. هر کاربر DB سپس دسترسی انتخابی به چیزی را خواهد داشت که نیاز دارد و در صورت نیاز، دسترسی نوشتن خواهد داشت.

به‌عنوان‌مثال، یک صفحه ورود، نیازمند دسترسی خواندن برای حوزه‌های کلمه عبور و نام کاربر یک جدول است، اما دسترسی نوشتاری به هر شکلی را ندارد (درج، به‌روزرسانی یا حذف نیست). بااین‌حال، صحه ثبت‌نام، قطعاً نیازمند اولویت درج در

آن جدول است؛ این محدودیت می‌تواند در صورتی تقویت شود که این برنامه‌های کاربردی وب، از کاربران DBبرای اتصال به پایگاه داده استفاده کنند.

دیدگاه‌ها

شما می‌توانید از دیدگاه‌های SQL برای افزایش اندازه دسترسی با محدودیت سازی دسترسی خواندن به حوزه‌های مشخصی از جدول یا الحاق‌های جداول، استفاده کنید. می‌تواند احتمالاً مزایای اضافی‌ای داشته باشد برای مثال، فرض کنید که این سیستم نیازمند (شاید به دلیل برخی نیازهای قانونی ویژه) بازیابی کلمات عبور کاربران به‌جای کلمات عبور درهم سازی شده دانه‌ای[4] باشد. این طراح می‌تواند از دیدگاه‌ها برای جبران این محدودیت استفاده کنید؛ همه دسترسی‌ها به جدول را لغو کند(از همه کاربران DB به‌جز ادمین/مالک) و دیدگاهی بسازد که درهم سازی کلمه عبور را تولید می‌کند و نه خود فیلد را. هر حمله تزریق SQL که در دزدیدن اطلاعات DB موفق باشد، محدود به دزدیدن درهم سازی کلمات عبور خواهد شد(نمی‌تواند حتی یک‌درهم سازی کلیدسازی شده باشد)، چراکه هیچ کاربر DB برای هیچ برنامه کاربردی وب به خود جدول دسترسی ندارد.

اعتبار سنجی ورودی لیست سفید

علاوه بر اینکه یک دفاع ابتدایی درزمانی است که هیچ‌چیز دیگری ممکن نیست (مثلاً وقتی یک متغیر انقیاد قانونی نباشد)، اعتبار سنجی ورودی همچنین می‌تواند دفاع ثانویه استفاده‌شده برای تشخیص ورودی غیرمجاز قبل از ارسال آن به پرس‌وجوی SQL باشد. برای کسب اطلاعات بیشتر لطفاً برگه تقلب اعتبار ورودی را ببینید. لذا به احتیاط ادامه دهید. داده‌های اعتبار سنجی شده برای درج در پرس‌وجوهای SQL از طریق ساخت رشته، ضروری نیست.

ترجمه و تدوین توسط: نیما اخلاقی

منبع: مقاله OWASP

[4] salted-hashed