کاهش آسیب‌پذیری پایگاه‌های اطلاعاتی وب از طریق تشخیص حملات تزریق کور SQL بر اساس درخواست مشتری

مجتبی مصطفوی قهفرخی[1]، احمد براآنی دستجردی[2]

تاریخ دریافت: ۹۰/۰۳/۲۵
تاریخ پذیرش: ۹۰/۰۶/۰۷

چکیده

طی سال‌های اخیر، «حملات تزریق SQL» یکی از تهدیدهای جدی برای برنامه‌های کاربردی وب که به نوعی پایگاه داده‌ای را به خدمت می‌گیرند، به حساب می‌آیند. این نوع از حملات اینترنتی، SQL ناخواسته را از طریق یک پارامتر ورودی، به پایگاه داده تزریق می‌کنند. راه‌های مختلفی برای جلوگیری از نفوذ حملات تزریق SQL به پایگاه داده معرفی شده است که همگی در سمت سرویس‌دهنده قابل انجام است و با کدهای کاربرد وب و یا کدهای پرس‌وجوی پایگاه داده سر و کار دارند. در این مقاله یک روش جدید برای جلوگیری از حملات تزریق معرفی شده است که با درخواست‌هایی که کاربر به سمت سرویس‌دهنده ارسال می‌کند سر و کار دارد. به این صورت که با تشخیص این که کاربر در حال ارسال یک SQL ناخواسته و تزریق آن به پایگاه داده است از ارسال آن به برنامه‌های کاربرد وب جلوگیری می‌شود. از آنجا که درخواست مشتری به سمت سرویس‌دهنده، در نقطه خروجی از سمت مشتری و در نقطه ورودی از سمت سرویس‌دهنده قابل دسترسی است، امکان پیاده‌سازی این روش در هر یک از این نقاط بررسی می‌گردد. نوع خاصی از این حملات، «حملات تزریق کور SQL» نامیده می‌شوند، که به دلیل کم‌تر کار شدن بر روی آنها، روش معرفی شده در این مقاله ویژه این نوع از حملات است.

کلیدواژه‌ها: تزریق SQL، تزریق کور SQL، برنامه‌های کاربردی وب، درخواست کاربر، پرس‌وجو، پدافند غیرعامل

۱- کارشناس ارشد گروه مهندسی کامپیوتر دانشگاه اصفهان، Email: mmg_ui@eng.ui.ac.ir
۲- استادیار گروه مهندسی کامپیوتر دانشگاه اصفهان، Email: ahmadb@eng.ui.ac.ir

۱- مقدمه

پدافند غیرعامل[1]، یکی از روش‌های نوین مقابله پیشگیرانه در برابر حملات دشمن است. پدافند غیر عامل به مجموعه اقداماتی اطلاق می‌گردد که مستلزم به‌کارگیری جنگ‌افزار و تسلیحات نبوده و با اجرای آن می‌توان از وارد شدن خسارات مالی به تجهیزات و تأسیسات حیاتی، حساس و مهم نظامی و غیرنظامی و تلفات انسانی جلوگیری نموده و یا میزان خسارت و تلفات ناشی از حملات دشمن را به حداقل ممکن کاهش داد[۱]. یکی از موضوعاتی که در بحث پدافند غیرعامل مطرح است، امنیت شبکه‌های اینترنتی و دفاع در برابر حملات اینترنتی است. اهمیت این حوزه به اهمیت امنیت در شبکه‌های اینترنتی گره خورده است. با توجه به اینکه روزانه شاهد ده‌ها مورد حملات اینترنتی مخرب در جهان هستیم، از آنجا که بیش از ۱۴ درصد این حملات، حملات تزریق SQL به پایگاه داده‌های تحت وب هستند[۳]، لذا تا زمان برقراری یک امنیت نسبی برای این نوع از داده‌ها، تلاش برای کاهش آسیب‌پذیری پایگاه داده‌های اطلاعاتی ضروری است، چرا که ارزش داده‌هایی که روی شبکه قرار می‌دهیم ممکن است بسیار مهم و حیاتی باشد.

این دسته از حملات ممکن است بر اثر اضافه کردن کدهای SQL دلخواه در ادامه یک آدرس مشابه زیر باشد:

http://www.site.com/id=1

و پس از ارسال به سرویس دهنده کدی مانند زیر به پایگاه داده ارسال شود:

select * from table where id='1'

می توان پس از جایگزینی مقدار id با عبارت (1' and 't'='t) کد SQL زیر را به پایگاه داده تزریق کرد:

select * from table where id='1' and 't'='t'

این کد تزریقی ساده ممکن است آن‌قدر پیچیده شود که بتواند داده‌های حساس تجاری، علمی یا شخصی مانند نام و کلمه عبور کاربران، آدرس و تلفن، موجودی یک حساب، جزئیات کارت اعتبار و... را از پایگاه داده‌ها بیرون کشیده و آنها را برای هکر هویدا سازد.

اولین بار Rain Forest Puppy در سال ۱۹۹۸ در [۴] با معرفی این نوع حملات به معرفی کشف هیجان برانگیز خود پرداخت.

در سال ۲۰۰۰ این ایده خود را در [۵] تکمیل کرده و به این موضوع پرداخت که چگونه می‌توان یک وب سایت عمومی را از طریق پیغام‌های خطایی که توسط پایگاه داده آن اعلام می‌شود، توسط این حملات به خطر انداخت و با این کار باعث شد تا هزاران مدیر سایت از افشا شدن اطلاعات سایت خود به ترس و واهمه بیفتند.

طراحان پایگاه داده که هنوز مفهوم عمیق‌تر نفوذ SQL را درک نکرده بودند « پیشنهاد دادند که سیستم پیغام خطاهای درازگو[2] را قطع کنند و این به خاطر این فهم اشتباه بود که حمله کننده به پایگاه داده بدون مشاهده پیغام خطاهای متعدد نمی‌تواند داده‌ها را از پایگاه داده به دست آورد» [۶].

Chris Anley، در سال ۲۰۰۲، مجموعه ای از حملات را روی یک صفحه آسیب‌پذیر معرفی کرد [۳] که بدون اینکه نیاز به پیغام خطایی از سوی سیستم باشد، به راحتی می‌توان توسط آنها به پایگاه داده نفوذ کرد. از آن پس این نوع حملات به حملات تزریق کور SQL[3] نامگذاری شدند [۶].

برای مثال با نوشتن عبارت‌هایی که کد SQL زیر را تولید کند، در صورتی که محتوای آن صفحه همراه با داده‌های استخراج شده از پایگاه داده نشان داده شود می‌توان احتمال داد پایگاه داده شرط ۱=۱ را چک کرده و خروجی دستور به برنامه کاربردی و از آنجا به همراه کدهای HTML به سمت مشتری ارسال شده و در واقع تزریق انجام پذیرفته است:

select * from table where id=10 and 1=1

اگر نفوذگر احتمال دهد که آشکار شدن داده‌های استخراج شده از پایگاه داده دلیل دیگری دارد با تولید کد زیر و برگرداندن محتوای صفحه‌ای بدون این داده‌ها از تزریق اطمینان پیدا می‌کند. چرا که شرط نادرست ۱=۲ در صورت تزریق، شرط پرس‌وجو را نادرست می‌نماید.

select * from table where id=10 and 1=2

برای جلوگیری از حملات تزریق SQL راه‌هایی بیان شده است که شرح آن در قسمت ۷ خواهد آمد. از لحاظ تولید نرم افزار، بیشتر ابزارهای تدافعی برای جلوگیری از حملات معمول تزریق SQL ساخته شده است. مثلاً از معروفترین این ابزارها scrawlr است که به دنبال درخواست شرکت مایکروسافت از شرکت اچ پی ساخته شده است، که در توضیحات آن نوشته شده است:

1- Passive Defense

2- verbose

3- blind SQL injection

«این ابزار حملات تزریق کور SQL را حمایت نمی‌کند» [۷].
روشی که در این مقاله ارائه شده است، برای تشخیص حملات تزریق کور SQL در درخواست مشتری به کار می‌آید. یعنی در مواقعی که تشخیص داده شد قرار است از سمت مشتری با ارسال یک درخواست، حمله‌ای صورت پذیرد، پس از تشخیص، از همان ابتدای ورود به سمت سرویس‌دهنده از پاسخ به آن جلوگیری شود. این روش با کدهای برنامه‌های کاربرد وب و یا کدهای پرس و جویی که برای پایگاه داده ارسال می‌شود کاری ندارد و فقط درخواستی که از طرف مشتری قرار است ارسال شود را بررسی می‌کند.
برای رسیدن به این هدف، ابتدا در بخش ۲ پیشینه تحقیق آورده می‌شود. در بخش ۳، تکنیک‌های تزریق کور SQL معرفی و در بخش ۴ به ارائه ساختار آنها پرداخته می‌شود، بخش‌های ۵ و ۶ به پیاده‌سازی ابزار و تست و ارزیابی آن می‌پردازد. در نهایت در بخش ۷، نتیجه‌گیری این مقاله آورده شده است.
از آنجا که روش ارائه شده در این مقاله با ساختار دقیق حملات سر و کار دارد و تاحدی وارد جزئیات محتوای آنها می‌شود، پس برای هر پایگاه داده خاص، نیاز به بررسی جداگانه دارد. پایگاه داده‌ای که در این مقاله روی آن کار شده است پایگاه داده پرکاربرد و متن باز MySQL و زبان مورد نظر php است.

۲- کارهای انجام شده و مقایسه آنها با راهکار ارائه شده

Boyd and Keromytis در سال ۲۰۰۴ توانستند ابزاری بسازند که بتواند با استفاده از یک پروکسی مابین برنامه‌های کاربردی وب و پایگاه داده، ابتدا لغات کلیدی SQL را با لغاتی تصادفی و برپایه یک کلید تصادفی جایگزین کند. پس از این که درخواست از سمت مشتری ارسال شد، کدها با همان لغت تصادفی رمز گشایی شده و در صورت وجود کلمات کلیدی SQL، نقش اصلی آنها از بین می‌رود. [۸].
Dysart and Sherriff در سال ۲۰۰۸ موفق به ساخت ابزاری شدند که مابین سرویس‌دهنده و پایگاه داده قرار می‌گیرد و با استفاده از گزاره‌های آماده و ساختار ثابت پرس و جو مقادیر ورودی را از یک مسیر امن و ساخت‌یافته دریافت می‌کند. آنها این ابزار را بر محیط php سوار کردند. این ساخته آنها برای یک پرس و جوی پویا نیز کاربرد داشت .[۹]. در همان سال Thomas and Williams کاری مشابه آنها انجام دادند با این تفاوت که در کاربردهای جاوا و در محیط‌های ایستا به‌کار می‌رفت [۱۰].
Martin and Lam در سال ۲۰۰۸ ابزاری تولید کردند که با بررسی کدهای کاربرد وب در یک سرویس‌دهنده آنهایی را که احتمال تزریق SQL می‌رود با کدهای امن جایگزینی کند. [۱۱].
Spinellis and Mitropoulos در سال ۲۰۰۹ متدی ارائه کردند که در آن یک پروکسی بین برنامه‌های کاربردی وب و سیستم مدیریت پایگاه داده قرار می‌گیرد. این پروکسی پس از طی یک فاز آموزشی، یک مدل از پرس و جوها به دست آورده که در زمان اجرا چک می‌کند که آیا پرس و جوها با مدل تعلیم دیده مطابقت دارند یا نه. در صورتی‌که این تطبیق پیدا نشد، از اجرای آن جلوگیری می‌کند [۱۲].
اعرابی و برنجکوب در سال ۱۳۸۸ روشی ارایه کرده‌اند که در آن بر اساس راهکاری مستقل از فناوری وب و بدون تفسیر و یا تغییر در کدهای برنامه، حملات تزریق SQL شناسایی و از بروز آنها جلوگیری شده است [۲].
تمامی راه‌حل‌هایی که بیان شد در سمت سرویس‌دهنده انجام می‌پذیرد و با کد SQL نهایی که برای پایگاه داده ارسال می‌شود کار می‌کند. روشی که در این مقاله ارائه شده است به دستور پرس‌وجویی که برای سیستم مدیریت پایگاه داده ارسال می‌شود کار ندارد. بلکه درخواست‌هایی که از سمت مشتری ارسال می‌شود را بررسی می‌کند. به‌طوری‌که برای جلوگیری از حملات تزریق کور SQL می‌توان این روش را در هر نقطه قبل از ورود به کدهای کاربرد وب به کار برد.

۳- انواع تکنیک‌های تزریق کور SQL به یک پایگاه داده

انواع روش‌های تزریق کور عبارات SQL در پایگاه داده‌ها که مطابقت با پایگاه داده MySQL دارند عبارتند از روش جستجوی ساده، جستجوی دودویی، روش مبتنی بر زمان و روش مبتنی بر پاسخ [۶] که همگی این روش‌ها از تکنیک‌های استنتاج محسوب می‌شوند. این تکنیک به نفوذگر اجازه می‌دهد که با تزریق یک عبارت SQL به یک سؤال شرطی مشابه زیر پاسخ دهد:

IF x THEN y ELSE z

که در اینجا شرط x یک SQL مناسب مانند این عبارت است که «آیا مقدار بیت ۲ از بایت ۱ از ستون ۱ از سطر ۱ برابر ۱ است؟». خروجی‌های y و zهم مقادیر یا رفتار متفاوتی را بستگی به انتخاب نفوذگر، از خود نشان می‌دهند. از آنجا که در حملات تزریق کور SQL فرض بر این است که پیغام‌های درازگوی سیستم بسته شده‌اند و هکرها نمی‌توانند از طریق حملات معمول به آن نفوذ کنند پس خروجی ما یک رفتار و مفهوم است که تقریباً همیشه مبتنی است بر زمان پاسخ، محتوای صفحه، خطای صفحه یا ترکیبی از اینها. در اینجا برای اختصار فقط تکنیک مبتنی بر زمان شرح داده می‌شود:

۳-۱- روش مبتنی بر زمان

این روش بر اساس زمان پاسخ سرویس‌دهنده به مشتری عمل می‌کند. به‌گونه‌ای که اگر تأخیر در برگرداندن جواب بود، شرط درست، و در غیر این صورت شرط نادرست است و یا بالعکس. برای مثال در تولید کد زیر از دستور if برای شرط و از دستور sleep برای تأخیر استفاده شده است:

```
select count(*) from table where id=10 union select if(
substring( system_user, 1, 1)= 'a' , sleep(10),1)
```

یعنی در صورتی که حرف اول نام کاربر سیستم a بود صفحه را با تأخیر نمایش می‌دهد.

۴- ارائه ساختار برای تکنیک استنتاج

با توجه به تکنیک‌های ذکر شده در مورد حملات تزریق کور SQL، از آنجا که از میان آنها، روش‌های مختلف تکنیک استنتاج برای پایگاه داده MySQL قابل استفاده است، در این بخش به ارائه ساختارهایی برای یکی از آنها پرداخته می‌شود:

۴-۱- تکنیک‌های مبتنی بر زمان

برای به دست آوردن ساختار حملات مبتنی بر زمان باید گفت منطق این نوع حملات تأخیر داشتن یا تأخیر نداشتن در برگرداندن جواب است، که یکی برای شرط درست و دیگری برای شرط نادرست در نظر گرفته می‌شود.
برای انجام این حملات، دو شرط اساسی لازم است:

- اول اینکه نیاز به دستوراتی است که در آنها یک شرط چک شود و بتواند در صورت درست بودن، عملی و در صورت نادرست بودن عمل دیگری را انجام دهد.
- دوم اینکه این دستورات بتوانند به نحوی در نقاط مستعد تزریق به‌همراه پارامترهای ورودی مورد استفاده قرار گیرند، که این هم مستلزم آن است که یک خروجی داشته باشند.

مثلاً دستور while فقط شرط اول را دارد و هیچ خروجی‌ای را بر نمی‌گرداند و نمی‌توان از آن استفاده نمود. از میان دستورات SQL که در آنها یک شرط چک می‌شود، دستور if و نوع خاصی از دستور case هستند که دو شرط مذکور را دارند (شکل ۱): همان‌طور که ملاحظه می‌شود دستورات در این شکل علاوه بر چک کردن یک شرط، دارای خروجی output1 یا output2 هستند که یک نفوذگر می‌تواند با استفاده از عملگرهای منطقی یا مجموعه‌ای عبارات مناسبی را جایگزین این خروجی‌ها نماید. با توجه به اینکه شرط (condition)، در دستورات فوق همان ساختار تکنیک جستجو(structure1) است می‌توان ساختار این نوع حملات را مانند شکل (۲) بیان نمود (در این شکل نشان ستاره به معنی هر تعداد کاراکتر است).

```
IF (condition , output1 if condition is true,
output2 if condition is false)


CASE
 WHEN condition THEN SELECT output1;
 ELSE SELECT output2;
END CASE;
```

شکل ۱- ساختار دو دستور if و case دارای شرایط تزریق

به همین ترتیب می‌توان برای تمامی روش‌های تکنیک‌های تزریق کور، ساختارهایی به دست آورد که در این صورت پس از رفع ابهام و استانداردسازی ساختارهای به دست آمده مطابق [۱۴]، یک گرامر همانند شکل (۳) برای آنها تولید می‌شود.

<structure2>:<logic_opr>,<cond_stmt>
<cond_stmt>:<if_stmt> || <case_stmt>
<if_stmt>:<if_stmt1> || <if_stmt2>
<if_stmt1> :<IF>, <(>, <structure1>, <,>, <delay_function>, <,> ,<*> ,<)>
<if_stmt2> :<IF>,<(>,<structure1>, <,>, <*>, <,> ,<delay_function> ,<)>
<case_stmt>:< case_stmt1> || < case_stmt2>
<case stmt1>:<CASE><WHEN><structure1>

شکل ۲- ساختار تزریق کور با استفاده از تکنیک مبتنی بر زمان

۵- پیاده‌سازی

با توجه به اینکه منطق ابزار این است که بر روی درخواست مشتری کار می‌کند، بنابر این می‌توان آن را از نقطه انتهایی سمت مشتری تا نقطه ورودی کدهای کاربردی وب پیاده‌سازی نمود.

از آنجا که در سمت مشتری، همه چیز در دست کاربر است و او می‌تواند به دلخواه خود هر تغییری در درخواست نهایی اعمال کند، باید برای این نوع پیاده‌سازی - بسته به انتخاب روش نهایی برای پیاده‌سازی - فرضیات و محدودیت‌هایی قائل شد.

اگر روش انتخاب شده برای پیاده‌سازی استفاده از یک پروکسی در سمت مشتری باشد یک نفوذگر نیز می‌تواند با تولید یک پروکسی دیگر به دلخواه خود در خواست ارسالی را تغییر دهد.

اگر روش استفاده شده برای پیاده‌سازی بر روی مرورگرها به کار گرفته شود، مسلماً می‌تواند تنها جلوی حملاتی را بگیرد که از طریق مرورگرها قابل انجام است. البته بسیاری از حملاتی که بر روی یک وب سایت انجام می‌شود، از طریق مرورگرهای وب است و معمولاً کاربرانی که مهارت چندانی در نفوذ به یک پایگاه‌داده ندارند و تعدادشان بیشتر از نفوذگرهای ماهر است، حملات را از طریق مرورگرها انجام می‌دهند. اما یک نفوذگر ماهر می‌تواند با نصب مرورگرهایی غیر استاندارد که اسکریپت را حمایت نکنند یا استفاده از یک پروکسی قبل از دریافت پاسخ توسط مرورگر (با گوش دادن به پورت ۸۰)، مدیریت نمایش صفحه را به دست بگیرد. باید گفت که در مورد پیاده‌سازی ابزار در سمت مشتری، می‌توان با اعمال فرض‌هایی، کاربردهایی تعریف کرد که در آنها هیچ کاربری نتواند به پایگاه داده نفوذ پیدا کند. این کاربردها باید یک خصوصیت مشترک داشته باشند و آن اینکه «کاربران نتوانند در هیچ یک از سیستم‌های سمت مشتری نرم‌افزار دلخواه خود را نصب نمایند».

این محدودیت برای کاربران می‌تواند به دو شکل باشد: محدودیت سخت‌افزاری یا محدودیت نرم‌افزاری. در محدودیت سخت‌افزاری به دلیل نبود سخت‌افزارهایی مانند CD-ROM، DVD-ROM یا USB کاربر امکان تبادل داده با سیستم را ندارد. برای مثال می‌توان به دستگاه‌هایی که یک سازمان یا اداره برای کاربران خود در مکان‌های مختلفی برای اتصال به شبکه قرار داده است، اشاره نمود. بنابر این، پیاده‌سازی در سمت مشتری تنها در کاربردهایی خاص و یا به طور موازی با روش‌های دیگر، قابل انجام است و در صورتی که کاربر بتواند داده‌ها را از طریق نرم‌افزارهایی غیر از مرورگرهای استاندارد دریافت کند، عملیات تزریق کور SQL به پایگاه داده قابل انجام است و این بزرگترین ایراد این روش در کاربردهای عمومی است.

ما برای استفاده از ابزار آن را در قالب یک پروکسی توسط زبان برنامه نویسی #c پیاده‌سازی کرده‌ایم که در سمت سرویس‌دهنده و مابین مشتری و برنامه‌های کاربردی وب قرار می‌گیرد. تنها کاری که باید انجام شود این است که مقادیر پارامترهای متدهای GET و POST که در بسته ارسالی از سمت مشتری وجود دارد باید به ورودی این کدها داده شود تا ابزار قوانین به دست آمده در گرامر شکل (۳) را در مورد هر یک از آنها بررسی نماید.

با توجه به اینکه سرویس‌دهنده وب[1] به پورتی مانند پورت شماره ۸۰ گوش می‌دهد و منتظر است تا از یک مشتری تقاضای ارتباطی را دریافت نماید [۱۵]، کافی است پروکسی این نقش را بازی کند تا به ازای هر درخواستی ابتدا مقادیر پارامترها را از لحاظ سلامت چک نماید.

در واقع کاری که ابزار باید انجام دهد این است که بتواند با پیدا کردن نقاط مستعد تزریق، گرامر نهایی به دست آمده در بخش ۴ را در آنها جستجو نماید و در صورت یافتن آنها عکس‌العمل مناسبی انجام دهد.

این عملیات برای متد GET توسط تابع safe1 و برای متد POST توسط تابع safe2 انجام می‌شود.

1- web server

<Structure>:<Logic_opr><T> || <Union_select><Z>
<T>:<Logic_expr> || <Cond_stmt>
<Z>:<If_stmt> ||<Case_stmt>
<Logic_opr>:<and> || <or>
<Logic_expr>:<not><Logic_expr><X> ||
<Expr><X> || <(><Logic_expr><)>
<X>:<Logic_opr><Logic_expr><X> ||<Null>
<Opr>:<Oprator1><Logic_expr>
||<Expr><Oprator2><Expr> || <(><Opr><)>
<Expr>:<String>||<Number>||<Function> || <Opr>
<Function>:<Func_name><Y> || <Main_func> ||
<Logic_func> || <(><Function><)>
<Y>:<(><Func_name><)> || <(><Expr><)>
<Cond_stmt>:<If_stmt> || <Case_stmt>
<If_stmt>:<if><(><String><,><K>
<K>:<Delay_function><,><*><)> ||
<*><,><Delay_function><)>
<Case_stmt>:<case><when><*><then><P>
<P>:<Delay_functions><;><Else_stmt>||<*><;><else
><Delay_functions><;>
<Else_stmt>:<Null> || <else><*><;>

شکل ۳- گرامر استاندارد تکنیک های تزریق کور SQL

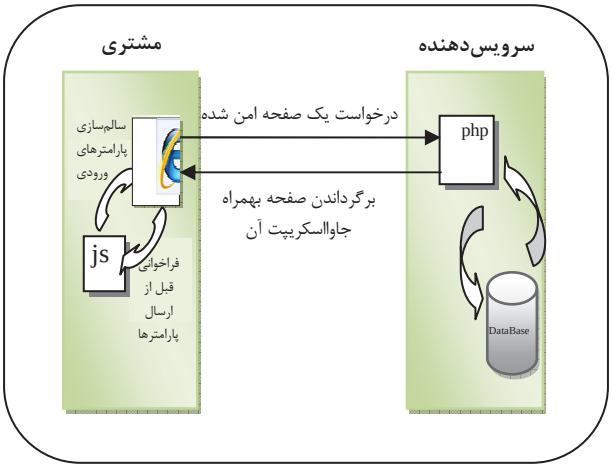
هر دو تابع safe1 و safe2 تمامی پارامترهای ورودی را که مستعد تزریق هستند جدا نموده و برای بررسی وجود ساختارهای تزریق کور در آنها به تابع structure می‌فرستند. تابع structure که هم‌نام اولین غیرپایانه گرامر شکل (۳) است، در واقع شروع کننده گرامر است. این تابع در پی جستجوی سمت راست قانون اول گرامر است:

1- structure(input: global data, output: true or false)
2- begin
3- if ((logic_opr() && T()) || ((union_select()) && Z())): return true;
4- else return false;
5- end structure

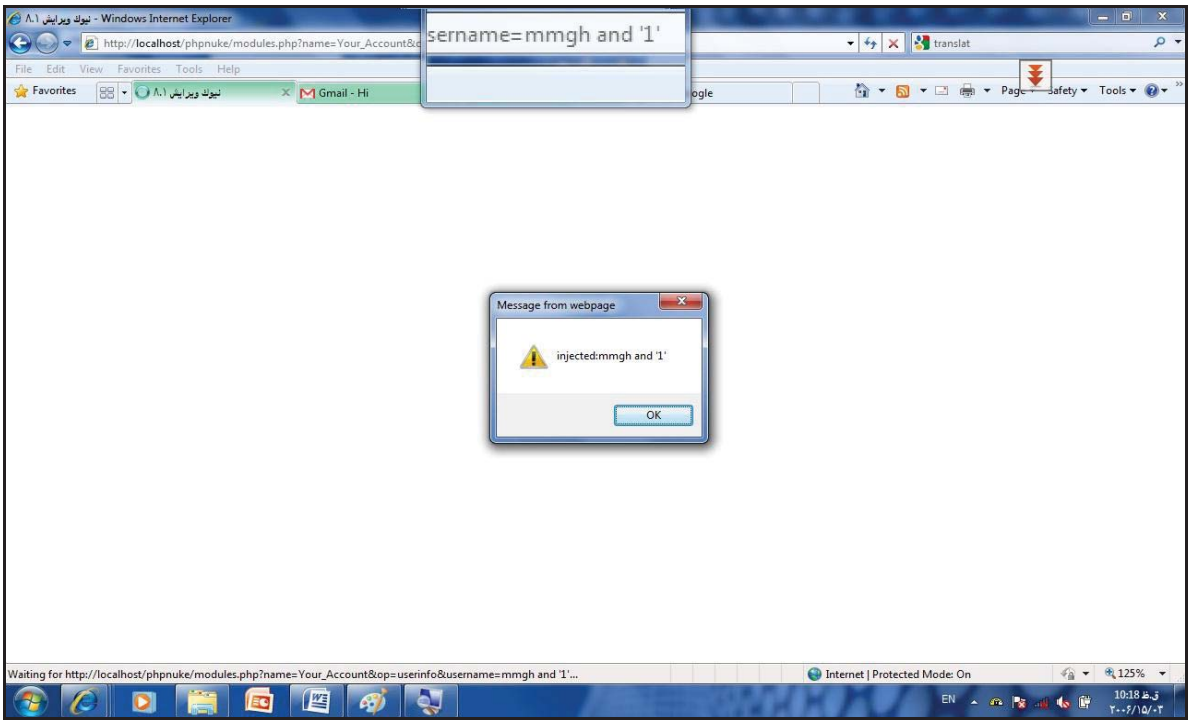
به همین شکل توابع نظیر غیرپایانه‌های دیگر نیز همانند قوانین نظیر خود عمل می‌کنند. برای قوانینی که دارای پایانه هستند، توابعی با کدهای متفاوت‌تر نسبت به کدهای مذکور نوشته شده است. برای مثال پیاده‌سازی غیرپایانه if_stmt توسط تابعی به همین نام انجام می‌شود. این تابع، پس از عبور از کلمه کلیدی if باید پرانتز باز مشاهده کند و سپس با گذر از یک رشته و یک علامت کاما به غیر پایانه K برسد.

۶- مورد مطالعه و آزمایش

مورد مطالعه این مقاله، کدهای متن باز phpnuke هستند. ابتدا رویکرد پیاده‌سازی در سمت مشتری انتخاب شد. در این روش، کدهای safe1() و safe2() در قالب یک فایل جاوااسکریپت با نام safe.js ذخیره شده‌اند که در ابتدای صفحاتی که امکان تزریق در آنها وجود دارد شناسانده می‌شود. از آنجا که ممکن است درخواست به گونه‌ای باشد که بتواند داده‌هایی ناخواسته را به پایگاه داده تزریق کند، قبل از ارسال توسط فایل جاوااسکریپت، یک تست سلامت انجام می‌شود (مانند شکل ۴). هرگاه در مقابل متغیرهای نوار آدرس یا فرم‌های یک صفحه، عبارتی با ساختاری از گرامر شکل (۳) نوشته شود قبل از بار شدن، نتایج حمله کشف شده و عکس العمل مناسب نشان داده می‌شود. مثلاً در شکل (۵)، کدهای جاوااسکریپت عبارت تزریقی جلوی نوار آدرس را تشخیص داده و آن را با صدور یک پیام و تغییر مسیر به صفحه اول سایت اعلام می‌کند. اگر چه در این روش در مورد ترفندها و تک‌های نفوذگر مانند از کار انداختن جاوااسکریپت و ساخت فرم جعلی، پاتک‌های مناسبی تدارک دیده شده است، اما به دلیل نقاط ضعف و محدودیت‌های آن که تا حدی در بخش قبل شرح داده شد، از توضیح جزئیات و ادامه دادن پیاده‌سازی صرفنظر می‌شود.

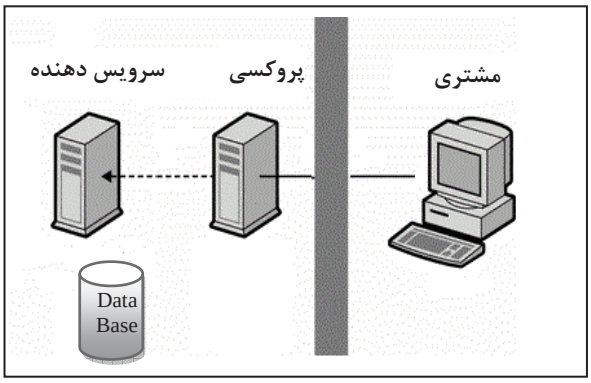


شکل ۴- سالم سازی پارامترهای ورودی توسط فایل جاوااسکریپت

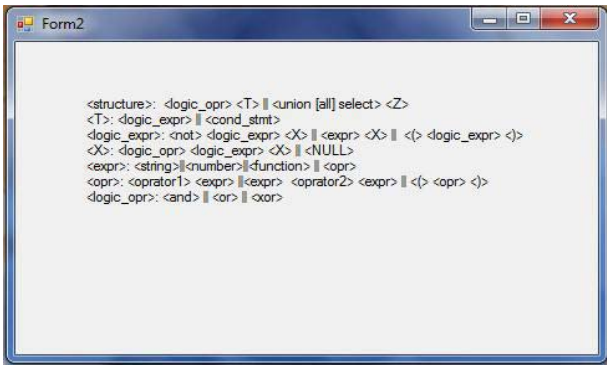


شکل ۵- شناسایی و جلوگیری از حمله در سمت مشتری

نوع دیگری از پیاده‌سازی ابزار، قرار دادن آن در اولین نقطه در سمت سرویس‌دهنده است. در این روش پیاده‌سازی، کاری که انجام می‌شود دریافت پارامترهای ارسالی از سمت مشتری (اعم از پارامترهای متد GET یا POST) و ارسال آنها به تابع structure است که همانند پیاده‌سازی در سمت مشتری از دو تابع safe1 و safe2 استفاده می‌شود. مطابق شکل (۶)، کلیه این توابع در یک پروکسی قرار دارد که قبل از سرویس‌دهنده، درخواست کاربر را دریافت نموه و آن را از لحاظ سالم بودن بررسی می‌نماید. کدهای پروکسی توسط زبان #c نوشته شده است و خروجی موقت آن، قوانینی از گرامر شکل (۳) است که منجر به کشف حمله شده‌اند. برای مثال شکل (۷) نتایج ارسال یک کد تزریقی به همراه درخواست کاربر را نشان می‌دهد. در این مثال ساده که کاربر عبارت ‘1’=’1 and را به متغیر id با مقدار ۱ اضافه کرده است، یک تزریق اتفاق افتاده که توسط نرم‌افزار تشخیص داده شده است. همانطور که مشاهده می‌شود برای آگاهی از نحوه تشخیص حمله، گرامرهایی که این ساختار حمله را کشف نموده‌اند در پیغامی از سوی نرم‌افزار نشان داده شده‌اند.



شکل ۶- پروکسی سمت سرویس دهنده



شکل ۷- شناسایی حمله در سمت سرویس دهنده

۷- تست و ارزیابی

در این بخش به ارزیابی پیاده‌سازی در سمت سرویس‌دهنده پرداخته می‌شود. همان‌طور که ملاحظه شد، پروکسی ساخته شده قبل از سرویس‌دهنده وب درخواست مشتری را دریافت نموده و آن را از لحاظ سلامت در برابر حملات تزریق کور SQL بررسی می‌نماید. از آنجا که حملات به صورت ساختاری بررسی شده است هر نوع حمله‌ای از نوع حملات تزریق کور SQL، پوشش داده می‌شود. همچنین از آنجا که در کدنویسی سمت سرویس‌دهنده از دستور foreach واقع در کدهای #C برای پیداکردن متغیرهای دو متد استفاده شده است، تمامی پارامترهای دریافتی بررسی می‌شوند. همچنین از آنجا که محل اجرای کدها در سمت سرویس‌دهنده است، کاربر همانند روش پیاده‌سازی در سمت مشتری امکان از کار انداختن این کدها یا جایگزین کردن کدهای دیگر به جای آن را ندارد.

چندین نمونه از کدهای تزریق کور بر روی فرم‌ها و متغیرهای نوار آدرس این سایت تست شد که برخی از نتایج تست در جدول (۱) آورده شده است.

همان‌طور که در این جدول مشخص شده است نمونه‌هایی از انواع حملات تزریق کور SQL جهت تست به ورودی نرم‌افزار داده شده است که تمامی آنها تشخیص داده شده و از انجام آن جلوگیری شده است. همچنین عباراتی سالم ولی مشابه با عبارات حملات تزریقی به نرم‌افزار داده شده است که هیچ یک از آنها به عنوان حمله شناسایی نشده‌اند.

جدول ۱- نتایج برخی از نمونه های تزریق کور بر روی کدهای phpnuke

نمونه حمله	متد GET	متد POST
' and '1'='1'	تشخیص و جلوگیری	تشخیص و جلوگیری
' and t=t	تشخیص و جلوگیری	تشخیص و جلوگیری
union select if(substring(system_user, 1, 1)= 'a' , sleep(10),1)	تشخیص و جلوگیری	تشخیص و جلوگیری
union%20%20 %20all%20 select if(substring(system_user, 1, 1)= 'a' , sleep(10),1)	تشخیص و جلوگیری	تشخیص و جلوگیری
and substring(system_user,1,1)='a'	تشخیص و جلوگیری	تشخیص و جلوگیری
or%20%20 substring (system_user , 1 , %201)='a'	تشخیص و جلوگیری	تشخیص و جلوگیری
' select count(*) from table where id=10 union select if(substring(system_user, 1, 1)='a', select table_name from information_schema.columns), 1)	تشخیص و جلوگیری	تشخیص و جلوگیری
' and ascii(substring(system_user,1,1))>63	تشخیص و جلوگیری	تشخیص و جلوگیری
Union all select case when ‘t’ then sleep else 0 end case;	تشخیص و جلوگیری	تشخیص و جلوگیری
" Union %20 x %20 select 1	حمله نیست	حمله نیست
) Random 1	حمله نیست	حمله نیست
%20Random 2* random 3 and 1	تشخیص و جلوگیری	تشخیص و جلوگیری
') Union xyz union select pqr	تشخیص و جلوگیری	تشخیص و جلوگیری
' Union xyz union selectpqr	حمله نیست	حمله نیست
+union select 1	حمله نیست	حمله نیست
Or %20%20 substring(system_user, 1, 1)='a'	تشخیص و جلوگیری	تشخیص و جلوگیری

۸- نتیجه‌گیری

این مقاله روشی ارائه کرده است که بدون استفاده از کدهای برنامه‌های کاربردی وب و کدهای دستورات پایگاه داده، فقط از طریق درخواست کاربر، حملات تزریق کور SQL را شناسایی می‌نماید. مهم‌ترین هدفی که دنبال می‌شود، بالا بردن ضریب امنیتی پایگاه داده‌های وب و کم کردن بار سرویس‌دهنده در برخی کاربردها از طریق جلوگیری از نوع خاصی از حملات تزریق SQL از همان ابتدای شکل‌گیری در سمت مشتری یا بررسی درخواست‌ها در سمت سرویس‌دهنده بدون تغییر اساسی روی کدهای کاربرد وب و یا کدهای پرس‌وجوی پایگاه داده است.

در این روش ابتدا نقاط مستعد تزریق SQL شناسایی شدند. از طرفی ساختارهایی برای تزریق کور SQL در پایگاه داده MySQL ارائه شد. اگر یکی از ساختارهای تزریق کور در نقاطی از یک فرم یا نوار آدرس که مستعد تزریق هستند به دست آید، حمله تزریق کور تشخیص و عکس‌العمل مناسبی در برابر آن نشان داده می‌شود.

برای پیاده‌سازی ابزار دو روش کلی بیان شد: پیاده‌سازی ابزار در سمت مشتری و پیاده‌سازی آن در سمت سرویس‌دهنده. پس از پیاده‌سازی دو روش، ضمن بیان کاربردهایی برای پیاده‌سازی در سمت مشتری به بیان نقاط ضعف و محدودیت‌های این روش پرداخته شد.

این پیاده‌سازی‌ها بر روی کدهای متن باز phpnuke صورت پذیرفت. نتایج در هر دو متد اصلی ارسال پارامتر بررسی شد و مطابق آنچه در بخش تست ابزار آمده است، ابزار توانست امتحان خود را به خوبی پس دهد.

در آینده سعی می‌شود این روش برای حملات گسترده‌تری از تزریق SQL طراحی و پیاده‌سازی شود.

سپاسگزاری

با تشکر از پژوهشکده مستقل پدافند غیر عامل دانشگاه جامع امام حسین(ع) که تمام هزینه‌های مصرفی این پایان‌نامه را تأمین نمودند.

مراجع

۱. ح نمازی و م فکوری، "پدافند غیر عامل"، معاونت حفظ و توسعه منابع و سرمایه‌های اداره کل آموزش ضمن خدمت، (**۱۳۸۷**).

۲. اعرابی، احسان؛ مهدی برنجکوب؛ محمدعلی منتظری؛ "تشخیص آسیب‌پذیری تزریق SQL بر اساس راهکاری مستقل از فناوری توسعه وب"، ششمین کنفرانس بین‌المللی انجمن رمز ایران، دانشگاه صنعتی اصفهان، (**۱۳۸۸**).

3. Nist, National Vulnerability Database, http://Nvd. Nist.Gov/, Accessed January (**2007**).
4. R. F. Puppy, Phrack, http://www.Phrack. Com/ Issues. HTML?Issue= 54&Id=8#Article. [Accessed May 12, (**2010**)].
5. R. F. Puppy, Rfp.Labs, http://www.Wiretrip. Net/Rfp/Txt/ Rfp2k01.Txt. [Accessed May 7, (**2010**)]
6. C. Anley, "Advanced SQL Injection In SQLserver Applications" An NGS Software Insight Security Research (NISR) Publication, 2002. http://www. Ngssoftwa- Re.Com/ Papers/ Advanced_SQL_ Injection.pdf. [Accessed May 17, (**2010**)].
7. Peterson, "Finding SQL Injection With Scrawlr", Hp , Http://Www.Communities. Hp.Com, June (**2008**).
8. W. Boyd And A. D. Keromytis, "SQLrand: Preventing SQL Injection Attacks", In International Conference On Applied Cryptography And Network Security (Acns), (**2004**).
9. F. Dysart And M. Sherriff, "Automated Fix Generator For SQL Injection Attacks", Ieee Computer Society, (**2008**).
10. S. Thomas And L. Williams, "On Automated Prepared Statement Generation To Remove SQL Injection Vulnerabilities", Elsevier Computer Communications Journal, March (**2009**).
11. M. Martin And M. S. Lam, "Automatic Generation Of Xss And SQL Injection Attacks With Goal-Directed Model Checking", Proceedings Of The 17th Conference On Security Symposium, July 28-August 01, (**2008**).
12. D. Mitropoulos And D. Spinellis , "Location-Specific Signatures Prevent SQL Injection Attack"S, Computer And Security (**2009**).
13. J. Clarke, "SQL Injection Attacks And Defense", 1st Edition, Syngress Publishing , Elsevier, (**2009**).
14. P. Linz, "An Introduction To Formal Languages And Automata" , Fourth Edition, Jones And Bartlett, (**2006**).
15. The Mentalis.Org Team, "Proxy Server In C#", http://www.Mentalis.Org, [Accessed Feb. (**2011**)]

Vulnerability Mitigation of Web Data Bases by Identification of SQL Blind Injection Based on Customer Request

Mojtaba Mostafavi Ghahfarkhi[1]

Ahmad BaraAni Dastjerdi[2]

Abstract

In recent years, the "SQL Injection Attacks" are considered one of the serious threats for web application programs which somehow take advantage of data bases. These internet based attacks, inject the unwanted SQL into the data base through an input parameter. Various methods to prevent the intrusion of SQL injection attacks have been introduced, all of which are practical in the favor of the service provider and are involved with web application codes or data base inquiry codes. In this article, a new method to prevent the injection attacks, has been presented which involves the requests sent to the service provider in such a way as to prevent the web application programs by identifying that the user is sending an unwanted SQL and injecting it to the data base. Since the customer request towards the service provider at the exit point from the customer and the input from the service provider are accessible, the possibility of implementing this method
at both of these points will be reviewed. A special kind of these attacks is called " SQL Blind Injection Attacks" which due to their being less reviewed, is especially condidered in this essay.

Key Words: *SQL Injection, Blind SQL Injection, Web Application Programs, Customer Request, Inquiry, Passive Defense*

1- MS. Computer Engineering of Isfahan University (Email: mmg_ui@eng.ui.ac.ir)
2- Associate Professor of Computer Engineering of Isfahan University (Email: ahmadb@eng.ui.ac.ir)