نشریه علمی "پدافند الکترونیکی و سایبری"

سال هشتم، شماره ۴، زمستان ۱۳۹۹، ص ۹۳- ۷۵

علمی– پژوهشی

تشخیص و پیشگیری از حملات تزریق SQL در زمان اجرا با استفاده از طبقه‌بندی درخت تصمیم

علی شهیدی‌نژاد[1*]، میترا ترابی[2]

۱- استادیار، ۲- کارشناس ارشد، گروه مهندسی کامپیوتر، واحد قم، دانشگاه آزاد اسلامی، قم، ایران

(دریافت: ۱۳۹۸/۰۷/۲۸، پذیرش: ۱۳۹۸/۱۱/۱۲)

چکیده

استفاده از برنامه‌های کاربردی وب به‌طور فزاینده‌ای در فعالیت روزمره ما، مانند خواندن اخبار، پرداخت صورت حساب و خرید آنلاین محبوب شده است. با افزایش در دسترس بودن این خدمات، شاهد افزایش تعداد و پیچیدگی حملاتی هستیم که برنامه‌های کاربردی وب را هدف قرار می‌دهند. تزریق SQL، یکی از جدی‌ترین تهدیدها برای برنامه‌های کاربردی وب در فضای سایبری محسوب می‌شود. حملات تزریق SQL، به مهاجمان اجازه می‌دهند تا دسترسی نامحدود به پایگاه داده‌ای که برنامه کاربردی و اطلاعات بالقوه حساس را شامل می‌شوند، به‌دست آورند. اگرچه محققان و متخصصان، روش‌های مختلفی برای حل مسئله تزریق SQL، پیشنهاد کرده‌اند، اما رویکردهای فعلی یا به‌طور کامل برای حل محدوده‌ای از مشکل شکست خورده‌اند، یا محدودیت‌هایی دارند که از استفاده و پذیرش آن‌ها جلوگیری می‌کند. این تحقیق بر آن است که یک روش، برای تشخیص و پیشگیری از حملات تزریق SQL در زمان اجرا ارائه دهد، که می‌تواند حملات موجود و جدید را تشخیص دهد، به‌علاوه بر حملات به‌طور مداوم نظارت کند. روش تشخیص و پیشگیری پیشنهادی، با نظارت زمان اجرا و به‌کارگیری طبقه‌بندی درخت تصمیم بر روی پایگاه داده تزریق SQL، حملات تزریق SQL موجود را مسدود می‌کند همچنین با استفاده از ناظر پایگاه داده حملات جدید را تشخیص می‌دهد. در پایان، روش پیشنهادی، با دیگر روش‌های تشخیص و پیشگیری از حملات تزریق SQL موجود، مقایسه می‌شود، نتایج به‌دست‌آمده نشان می‌دهد، که روش پیشنهادی، به‌طور قابل توجهی در تشخیص و پیشگیری از حملات تزریق SQL موفق است. دقت روش پیشنهادی در مقایسه با دو روش مورد مقایسه مقاله به ترتیب ٪۱۲ و ۱۶٪ افزایش یافته است.

کلیدواژه‌ها: برنامه‌های کاربردی وب، امنیت پایگاه داده، حملات تزریق SQL، تشخیص، پیشگیری.

۱- مقدمه

بسیاری از سازمان‌ها نیاز به ذخیره اطلاعات حساس مانند سوابق مشتری یا اسناد خصوصی دارند و این اطلاعات را روی شبکه در دسترس قرار می‌دهند. به همین دلیل، برنامه‌های کاربردی وب مبتنی بر پایگاه داده به‌طور گسترده‌ای در سامانه‌های سازمانی و اینترنت مستقر شده‌اند. هم‌زمان با گسترش رو به رشد آن‌ها، حملات فراوانی این برنامه‌ها را هدف قرار داده‌اند. نوعی از حمله به‌طور خاص، حملات تزریق SQL است که بسیار زیان‌آور است. حملات تزریق SQL می‌توانند به مهاجمان دسترسی مستقیم به پایگاه داده تحت یک برنامه کاربردی را بدهند و به آن‌ها اجازه می‌دهد اطلاعات محرمانه یا حتی حساس را نابود کنند. نمونه‌های بسیاری از حملات تزریق SQL با عواقب جدی وجود دارد حتی نگران‌کننده‌تر، مطالعه‌ای است که توسط گروه گارتنر در بیش از ۳۰۰ وب‌سایت انجام شده است، که نشان می‌دهد ۹۷٪ از سایت‌های مورد بازرسی در برابر این نوع حملات وب آسیب‌پذیر هستند [۱].

تزریق SQL، به یک کلاس از حملات تزریق کد اشاره دارد که در آن، داده‌های ارائه‌شده توسط کاربر، در یک پرس‌وجوی SQL به‌گونه‌ای قرار می‌گیرند که بخشی از ورودی کاربر به‌عنوان کد SQL رفتار شود. حملات تزریق SQL نوعی آسیب‌پذیری است که در نهایت ناشی از اعتبارسنجی ناکافی ورودی کاربر می‌باشد و زمانی رخ می‌دهند که داده‌های ارائه‌شده توسط کاربر به‌درستی اعتبارسنجی نشده‌اند و مستقیماً در یک پرس‌وجوی SQL وارد شده باشد. با استفاده از این آسیب‌پذیری‌ها، مهاجم می‌تواند دستورات SQL را مستقیماً به پایگاه داده ارسال کند. آسیب‌پذیری‌های برنامه کاربردی وب شامل یک نقص سامانه یا ضعف در یک برنامه کاربردی وب است و عمدتاً به‌دلیل عدم اعتبارسنجی و عدم پاک‌سازی ورودی‌های فرم، وب‌سرورهایی با پیکربندی اشتباه و نقص‌های طراحی برنامه کاربردی می‌باشند [۲]. تزریق SQL یک آسیب‌پذیری رایج است که برای نفوذ در پایگاه داده‌های برنامه کاربردی وب توسط اجرای یک کد SQL مخرب تزریق‌شده توسط مهاجم به‌کار گرفته می‌شود. همچنین تزریق SQL به‌عنوان اولین آسیب‌پذیری خطرناک با توجه به

* رایانامه نویسنده مسئول: a.shahidinejad@qom-iau.ac.ir

آمارهای OWASP [۳] طبقه‌بندی شده است. این نوع آسیب‌پذیری یک تهدید جدی برای هر برنامه کاربردی وب را نشان می‌دهد که ورودی را از کاربران (از طریق فرم‌های وب یا APIهای وب) می‌خواند و از آن برای ساختن پرس‌وجوهای SQL به یک پایگاه داده زیربنایی استفاده می‌کند. اکثر برنامه‌های کاربردی وب که در اینترنت یا درون شرکت‌ها استفاده می‌شوند، به این طریق کار می‌کنند و بنابراین می‌توانند به تزریق SQL آسیب‌پذیر باشند. اگرچه آسیب‌پذیری‌هایی که به حملات تزریق SQL منجر می‌شوند، به‌خوبی درک می‌شوند، اما به‌دلیل کمبود روش‌های مؤثر برای تشخیص و پیشگیری از آن، همچنان ادامه دارند. شیوه‌های برنامه‌نویسی مانند برنامه‌نویسی دفاعی و روش‌های اعتبارسنجی ورودی پیشرفته می‌توانند برخی از آسیب‌پذیری‌ها را از بین ببرند. با این حال، مهاجمان همچنان به پیدا کردن سوءاستفاده‌های جدید ادامه می‌دهند تا بتوانند برنامه‌نویسان را کنترل کنند. علاوه بر این، برنامه‌نویسی تدافعی، کاری فشرده است و این باعث می‌شود که یک روش غیرعملی برای حفاظت از سامانه‌های بزرگ باشد با این حال، در عمل استفاده از چنین روش‌هایی مبتنی بر انسان است و بنابراین مستعد اشتباهات است.

ابزارهای عمومی مانند فایروال‌ها و سامانه‌های تشخیص نفوذ (IDSs) نیز معمولاً در برابر حملات تزریق SQL ناکارآمد هستند. حملات تزریق SQL از طریق پورت‌هایی که برای ترافیک وب معمولی (معمولاً در فایروال‌ها باز هستند) انجام می‌شوند و در سطح برنامه کاربردی کار می‌کنند (بر خلاف اکثر IDS ها). در نهایت، بسیاری از روش‌های مبتنی بر تجزیه و تحلیل برای تشخیص آسیب‌پذیری، ویژگی‌های خاص از حملات تزریق SQL را مورد خطاب قرار نمی‌دهند و در نتیجه در این زمینه بی‌اثر هستند. چند روش تجزیه و تحلیل که به‌طور خاص برای هدف‌گیری حملات تزریق SQL طراحی شده‌اند، تنها راه‌حل جزئی برای حل این مشکل را ارائه می‌دهند. به‌طور خاص روش‌های پویا مانند آزمودن نفوذ، مسائل کامل را معرفی می‌کنند و اغلب منجر به تولید منفی کاذب می‌شوند، درحالی‌که روش‌های مبتنی بر تجزیه و تحلیل ایستا یا بیش از حد غیردقیق هستند و یا فقط روی یک جنبه خاص از مشکل تمرکز می‌کنند. این مقاله بر آن است که روشی برای تشخیص و پیشگیری از حملات تزریق SQL در زمان اجرا ارائه دهد، که می‌تواند حملات موجود و جدید را تشخیص دهد، به‌علاوه بر حملات به‌طور مداوم نظارت کند. روش تشخیص و پیشگیری پیشنهادی، با نظارت زمان اجرا و به‌کارگیری طبقه‌بندی درخت تصمیم بر روی پایگاه داده تزریق SQL، حملات تزریق SQL موجود را مسدود می‌کند همچنین با استفاده از ناظر پایگاه داده حملات جدید را تشخیص می‌دهد. در این مقاله، همچنین یک ارزیابی تجربی از روش ارائه می‌شود. موضوعات با شمار زیادی از دسترسی‌های قانونی و حملات تزریق SQL هدف قرار گرفته است و توانایی روش پیشنهادی برای تشخیص و پیشگیری از تزریق SQL بررسی می‌شود. نتایج ارزیابی بسیار امیدوارکننده است.

دستاوردهای اصلی این پژوهش به شرح زیر می‌باشد:

- ارائه روشی برای تشخیص و پیشگیری از حملات تزریق SQL
- تشخیص زمان اجرا، برای تشخیص انواع مختلف حملات تزریق SQL
- ناظر پایگاه داده، برای شناسایی حملات جدید تزریق SQL ممکن با نظارت بر تراکنش‌های پایگاه داده
- تشکیل پایگاه داده تزریق SQL و به‌کارگیری طبقه‌بندی درخت تصمیم برای تشخیص حملات تزریق SQL

ادامه ساختار مقاله بدین شرح است: در بخش دوم مقاله نمونه‌ای از تزریق SQL مثال زده می‌شود و نیز در ادامه بخش دوم انواع مختلف حملات تزریق SQL شرح داده می‌شود. در بخش سوم کارهای مرتبط انجام‌شده در زمینه دفاع در برابر حملات تزریق SQL بررسی می‌شود. در بخش چهارم، روش تشخیص و پیشگیری از حملات تزریق SQL پیشنهادی مقاله، ارائه می‌گردد. در بخش پنجم به ارزیابی روش پیشنهادی مقاله می‌پردازد و در نهایت در بخش ششم نتیجه‌گیری و پیشنهادها مطرح می‌گردد.

۲- نمونه‌ای از تزریق SQL

برای خدمت به کاربر در یک وب‌سایت، اطلاعات کاربر، مورد نیاز است. بر این اساس، برنامه‌های کاربردی وب معمولاً یک صفحه ورود به سایت (Login) شامل دو فیلد متنی را فراهم می‌کنند تا به کاربر اجازه دهد نام و کلمه عبورش را وارد کند. بعد از ورودی کاربر، داده ارسال خواهد شد و اطلاعات کاربر به پایگاه داده برنامه کاربردی وب برای بررسی، فرستاده خواهد شد. با ارسال داده کاربر، این داده به پایگاه داده برنامه کاربردی وب با استفاده از دستور SQL زیر فرستاده خواهد شد:

```
SELECT * FROM UserTable WHERE username=
“user_entry_name” and userpassword
=”user_entry_password”
```

هنگامی‌که این دستور SQL اجرا می‌شود، سامانه نتیجه‌ای از پرس‌وجو را باز خواهد گرداند. اگر داده کاربر معتبر باشد، سپس برنامه کاربردی وب به کاربر مجوز دسترسی به دیگر صفحات

وب‌سایت را می‌دهد یا اینکه ورودی کاربر پذیرفته نخواهد شد و صفحه ورود مجدداً بارگذاری می‌شود.

با این حال، سناریوی دیگری وجود دارد، اگر کاربر این کد را در فیلد نام کاربری (username or ‘1’=’1’ --) وارد کند سپس دستور SQL به شرح زیر خواهد بود:

SELECT * FROM UserTable WHERE username = “username or ‘1’=’1’ -- “

در این مرحله موتور پایگاه داده هر کدی بعد از WHERE را به‌عنوان یک بیان شرطی در نظر می‌گیرد و هنگامی‌که مفسر پایگاه داده عبارت “ -- 1=1 or” را می‌یابد، شرط بررسی همیشه صحیح است. علاوه بر این، هر کد یا شرطی بعد از دابل دش[1] (--) نادیده گرفته خواهد شد. در نتیجه مهاجم دسترسی غیرمجاز به این برنامه کاربردی وب خواهد داشت. این مثال، نمونه برجسته از یک پرس‌وجوی درست‌نما[2] مبتنی بر حمله تزریق SQL است [۴-۶]. این نوع از حمله توسط تزریق به برنامه کاربردی وب، با بیان یک دستور که معمولاً صحیح برمی‌گرداند، انجام شده است. انواع حملات تزریق SQL بسیاری وجود دارند که در بخش بعدی توضیح داده می‌شود.

۲-۱- طبقه‌بندی انواع مختلف حملات تزریق SQL

مطابق با [۷]، انواع مختلفی از روش‌های تزریق SQL وجود دارد. هر نوع می‌تواند در انزوا یا به‌صورت ترکیبی انجام شوند. این به تجربیات، اهداف و رفتار مهاجم بستگی دارد. در این بخش، انواع مختلفی از حملات تزریق SQL مورد بحث قرار خواهد گرفت.

۲-۱-۱- پرس‌وجوی درست‌نما

در این روش اظهارات شرطی معمولاً صحیح[3] را برمی‌گردانند، یا صحیح ارزیابی می‌شوند. هنگامی‌که مهاجم، اظهارات شرطی پرس‌وجوی برنامه کاربردی وب را با کد مخرب تزریق می‌کند، مهاجم قصد دارد تا مقدار اظهارات شرطی برابر با صحیح را حفظ کند. این روش را معمولاً در صفحه ورود به سایت برای تزریق در فیلد ورود با ”or 1=1“ به‌کار می‌گیرد [۷-۱۰].

۲-۱-۲- پرس‌وجوی Piggy-Backed

هدف از این نوع حمله، تزریق پرس‌وجوی اصلی با یک پرس‌وجوی اضافی می‌باشد. همه پرس‌وجوها به ترتیب شروع با یکی از پرس‌وجوی اصلی اجرا خواهند شد. این حمله از دیگر حملات متفاوت است زیرا مهاجمان پرس‌وجوی اصلی را ویرایش یا تغییر نمی‌دهند، آن‌ها تنها سعی دارند که پرس‌وجوهای جدید اضافه و آن‌ها را به پرس‌وجوی اصلی پیوست کنند. بر این اساس، موتور پایگاه داده بیشتر از یک پرس‌وجو دریافت خواهد کرد، اولین پرس‌وجو به‌طور نرمال اجرا خواهد شد سپس دومی یا دیگر پرس‌وجوها اجرا خواهند شد. در نتیجه، اگر پرس‌وجوی دوم به‌طور موفقیت‌آمیز اجرا شده باشد، مهاجم می‌تواند هر دستور SQL مانند روال‌های ذخیره شده یا هر دستور دیگر را اجرا و تزریق کند [۷-۱۰].

۲-۱-۳- پرس‌وجوی اجتماع[4]

ایده‌ی پشت این حمله پرس‌وجوی اجتماع، به انواع دیگر تزریق SQL شبیه است. مهاجمان به دنبال یک پارامتر آسیب‌پذیر هستند و سعی دارند به بهره‌برداری از آن با استفاده از تغییر مجموعه داده‌ای که برای یک پرس‌وجوی ارسال شده برگردانده می‌شود، بپردازند. علاوه بر این، با استفاده از این روش برنامه کاربردی نتایج مختلفی از پایگاه داده به‌جای یک برنامه‌نویسی توسط توسعه‌دهنده دریافت خواهد کرد. این روش با تزریق پارامتر آسیب‌پذیر با استفاده از کلمه کلیدی UNION SELECT آغاز می‌شود، بنابراین مهاجم می‌تواند پرس‌وجوی دوم را برای به‌دست‌آوردن اطلاعات پایگاه داده کنترل کند. علاوه بر این، داده از هر جدولی در دسترس خواهد بود و مهاجم تنها باید انتخاب کند که چه داده‌ای یا از هر جدول خاص می‌خواهد [۷-۱۰].

۲-۱-۴- پرس‌وجوی نادرست از لحاظ منطقی یا پرس‌وجوی غیرقانونی[5]

پرس‌وجوی نادرست از لحاظ منطقی یا پرس‌وجوی غیرقانونی، یک نوع حمله تزریق SQL است که در مراحل اولیه از یک حمله برای جمع‌آوری اطلاعات درباره پایگاه داده مانند نوع پایگاه داده، ستون‌های جدول و نوع ستون یا دیگر اطلاعات استفاده می‌شود. در این نوع از حمله، ورودی یک جمله نادرست از لحاظ منطقی می‌باشد، که موجب یک پاسخ خطای پایگاه داده می‌شود. مانند اضافه کردن ۱=۲ به جمله شرط. بنابراین، این روش معمولاً، با تزریق پارامتر آسیب‌پذیر از صفحه وب با یک دستور نادرست (به لحاظ منطقی) برای ایجاد یک خطا از موتور پایگاه داده آغاز می‌شود [۷-۱۰].

۲-۱-۵- پرس‌وجوی رویه‌های ذخیره‌شده[6]

این روش حمله تزریق SQL، برای اجرا یا ایجاد رویه‌های ذخیره‌شده به‌کار می‌رود، که توسط موتور پایگاه داده استفاده شده‌اند. رویه ذخیره شده، معمولاً توسط توسعه‌دهنده یا مدیر پایگاه داده برای کنترل پایگاه داده و استفاده از مزایای امکانات

[1] Double dash
[2] Tautology Query
[3] True
[4] Union Query
[5] Logically Incorrect Query
[6] Stored Procedures

پایگاه داده استفاده شده است. مانند دسترسی به پایگاه داده یا سرویس‌های پایگاه داده [۷-۱۰].

۲-۱-۶- پرس‌وجوی استنتاجی یا استنباطی[1]

این روش حمله، هنگامی‌که مهاجم قادر نیست هرگونه پیام تعاملی را از طریق یک دستور تزریق SQL به‌دست آورد، استفاده می‌شود. بنابراین، مهاجمان به دنبال پیدا کردن روش‌های دیگر برای افشای آسیب‌پذیری وب‌سایت هستند. مهاجم اینجا یک پاسخ برنامه کاربردی وب را توسط تزریق آن با کلمه‌های کلیدی SQL مختلف تخمین می‌زند، تا اینکه او اطلاعات مورد نیاز از پایگاه داده را برای شروع حمله خود به‌دست می‌آورد [۷-۱۰]. این نوع از حمله به‌طور کلی به انواع زیرمجموعه‌هایی به شرح زیر تقسیم می‌شود:

- **پرس‌وجوی استنتاج تزریق کور[2]**

در این روش، مهاجمان به صفحه وب با یک جمله شرطی تزریق می‌کنند که به آن‌ها کمک می‌کند تا به آرایش پایگاه داده از طریق ارزیابی کردن پاسخ موتور پایگاه داده با جمله شرطی تزریق پی‌برند، خواه جمله درست یا غلط باشد. در این مرحله، اگر جمله درست ارزیابی شود، سامانه کار را به‌صورت نرمال ادامه خواهد داد. در نتیجه، اگر جمله تزریق‌شده غلط ارزیابی شود، صفحه وب یک پیغام خطا باز نخواهد گرداند. همان‌طورکه، صفحه وب به‌طور نرمال کار نخواهد کرد، برای مثال، تفاوت‌هایی بین رفتار صفحه قبل از جمله تزریق و بعد از آن وجود دارد. بنابراین، مهاجم اینجا اطلاعات را با استفاده از مقایسه نتایج پاسخ از پرس‌وجوها با تزریق دستور تزریق‌شده‌ی true یا false جمع‌آوری خواهد کرد [۱۱].

- **پرس‌وجوی زمان‌بندی استنتاج[3]**

در این روش، مهاجم پرس‌وجوهایی با یک دستور مخرب تزریق می‌کند، که یک تأخیر سامانه ایجاد کند. سپس مهاجم، واکنش برنامه کاربردی وب را با نظارت بر زمان پاسخ و جمع‌آوری اطلاعات درباره پایگاه داده طبق این پاسخ مشاهده خواهد کرد. درصورتی‌که تأخیری وجود داشت سپس دستور تزریق‌شده با موفقیت اجرا می‌شود، در غیر این‌صورت، اجرای دستور مشکل دارد و مهاجم نیازمند تغییر دستور تزریق‌شده می‌باشد.

۲-۱-۷- پرس‌وجوی رمزگذاری جایگزین[4]

روش‌های حمله نرمال، به دنبال دانستن کاراکترها یا کلمات کلیدی هستند که معمولاً کاراکترهای بد، نامیده می‌شوند. در این روش، مهاجمان از روش‌های تشخیص نرمال با استفاده از متن تزریق‌شده که رمزگذاری جایگزین به‌کار می‌گیرد، می‌گریزند. رمزگذاری جایگزین، متن تزریق شده رمزگذاری شده در اسکی، یونیکد یا هگزادسیمال را استفاده می‌کند. به‌این‌ترتیب، اهداف مهاجم نمی‌تواند تعریف شده باشد، بنابراین، مهاجم می‌تواند بیشتر از یک روش رمزگذاری استفاده کند. در نتیجه، در طول توسعه برنامه کاربردی، توسعه‌دهنده باید امنیت برنامه کاربردی تحت وب را در برابر این نوع از حمله با استفاده از روش مؤثری که احتمالات مختلف از متن رمزگذاری مخرب را در نظر می‌گیرد، برای پیشگیری این نوع از حمله تأمین کند.

۲-۱-۸- توضیحات درون‌خطی[5]

این حمله تزریق SQL می‌تواند با همه روش‌های حمله قبلی استفاده شود، زیرا مهاجم می‌تواند دستور تزریق را با استفاده از قابلیت‌های برنامه‌نویسی از توضیحات درون‌خطی تقسیم کند. این روش، می‌تواند مهاجم را برای دور کردن از روش‌های تشخیص و پیشگیری اولیه که به دنبال یک کاراکتر خاص هستند، پشتیبانی کند.

این طبقه‌بندی از تزریق SQL، مفید است زیرا آن به توسعه‌دهنده کمک می‌کند، تا آسیب‌پذیری‌های تزریق SQL را در طول مرحله توسعه برنامه کاربردی، تشخیص و تعمیر کند.

۳- کارهای مرتبط انجام‌شده

گولد و همکاران [۱۲]، یک بررسی‌کننده JDBC پیشنهاد داده‌اند، یک ابزاری است که می‌تواند به‌طور ایستا برای بررسی نوع پرس‌وجوهای صحیح در دستور SQL مورد استفاده قرار گیرد و به‌طور پویا در Java ایجاد شده است. این روش تنها آسیب‌پذیری‌های تزریق SQL را که بر اساس عدم تطابق نوع، مانند حملات پرس‌وجوی نادرست منطقی هستند را تشخیص می‌دهد. زیرا آن‌ها تنها ساختار نحوی از دستورات SQL را برای خطاها بررسی می‌کنند. اما حملات تزریق SQL می‌تواند از لحاظ نحوی درست باشد و خطاهای پایگاه داده را بر نگرداند.

در مدل ADMIRE [۱۳] نویسندگان، مدل ریسک تهدید جامع و مرحله‌ای ADMIRE را، برای شناسایی و مقابله با حملات تزریق SQL، توسط محافظت از پایگاه داده زیرین پیشنهاد کرده‌اند. مدل ADMIRE، ارزیابی خطر را تجزیه و تحلیل می‌کند. این مدل، شش مرحله را برای مقابله با حملات تزریق SQL، دنبال می‌کند: ۱- اهداف امنیتی را تجزیه و تحلیل می‌کند. ۲- برنامه کاربردی را تقسیم می‌کند. ۳- آسیب‌پذیری‌ها را

[1] Inference Query
[2] Blind Injection Inference
[3] Timing Inference Query
[4] Alternate Encoding
[5] Inline Comments

علامت‌گذاری می‌کند. ۴-تهدیدات را شناسایی می‌کند. ۵- تهدید را رتبه‌بندی می‌کند. ۶- تهدید را از بین می‌برد. مدل تهدید ADMIRE، به توسعه‌دهندگان و طراحان، یک درک بهتری از برنامه کاربردی ارائه می‌دهد و کمک به پیدا کردن اشکالات می‌کند. این یک رویکرد پیشگیرانه و یک بخش حیاتی از فرآیند طراحی می‌باشد.

روش X-log authentication [۱۴] برای پیشگیری از حملات تزریق SQL، در برنامه کاربردی وب پیشنهاد شده است. ایده اصلی این روش، استفاده از روش‌های سه‌گانه فیلتراسیون برای پیشگیری از تزریق SQL است، که در زیر شرح داده می‌شود:

- محافظ آسیب‌پذیری: برای شناسایی فراکاراکترها، کاراکترهایی که نشان‌دهنده مجموعه‌ای از کاراکترها می‌باشند مانند txt.*، برای پیشگیری از حملات تزریق SQL می‌باشد.
- احراز هویت ورودی (X-log authentication): برای بررسی ورودی کاربر، از X- Log Generator ، که در آن یک پایگاه داده معتبر به‌طور جداگانه ذخیره شده است و سپس، فیلد ورودی کاربر اعتبارسنجی شده، مجاز به ادامه دادن می‌شود.
- رویه ذخیره‌شده: برای بررسی اندازه و نوع داده ورودی کاربر و انجام اعتبارسنجی سمت سرور می‌باشد. نویسندگان، این روش را در بسیاری از برنامه‌های کاربردی وب، آزمایش کرده‌اند و استدلال می‌کنند که از انواع حملات تزریق SQL پیشگیری می‌کند.

دو رویکرد مرتبط SQLGuard [۱۵] و SQLCheck[۱۶]، نیز پرس‌وجوها را در زمان اجرا بررسی می‌کنند تا ببینند که آیا آن‌ها مطابق با یک مدل از درخواست‌های مورد انتظار است. در این رویکردها، مدل به‌عنوان یک گرامر بیان می‌شود که تنها پرس‌وجوهای قانونی را می‌پذیرد. در SQLGuard، مدل در زمان اجرا، با بررسی ساختار پرس‌وجو قبل و پس از اضافه کردن ورودی کاربر محاسبه می‌شود. در SQLCheck، مدل به‌طور مستقل توسط توسعه‌دهنده مشخص شده است. هر دو روش، از یک کلید مخفی، برای تعیین ورودی کاربر، در هنگام تجزیه، توسط کنترل‌کننده زمان اجرا، استفاده می‌کنند. بنابراین امنیت این رویکرد، بستگی به مهاجمانی دارد که قادر به کشف کلید نیستند. علاوه بر این، استفاده از این دو رویکرد، نیاز به توسعه‌دهنده دارد که یا کد را برای استفاده از یک کتابخانه واسط خاص، بازنویسی کند، یا به‌طور دستی نشانگرهای خاصی را به کد ورودی کاربر، به یک پرس‌وجوی ایجاد شده، که به‌طور پویا اضافه شده است، وارد کند.

در روش Header Sanitization [۱۷] متغیرهای دریافت شده را در روش‌های درخواست سرآیند HTTP پاک‌سازی می‌کند. محتوای پاک‌سازی‌شده داخل رشته سرآیند اصلی جایگزین می‌شود.

روش Network Analyzer [۱۸]، یک سامانه تشخیص بین مهاجم و سرور وب ایجاد می‌کند. این سامانه سرآیندها و حمل بار (محموله) را از طریق بازرسی عمیق بسته تجزیه و تحلیل می‌کند.

در روش Web Application Firewall [۱۹]، برنامه کاربردی وب، متغیرهای دریافت شده در داخل روش‌های درخواست سرآیند HTTP را تجزیه و تحلیل می‌کند. محتوای پاک‌سازی‌شده یا رد و یا درصورتی‌که تزریق SQL مشخص نشود به موتور SQL منتقل می‌شود.

هوانگ و همکاران [۲۰]، ابزار WebSSARI را ارائه داده‌اند، که یک الگوریتم تشخیص بر اساس روش تجزیه و تحلیل جریان اطلاعات برنامه کاربردی برای تشخیص تابع حساس به‌کار می‌گیرد، که می‌تواند در یک برنامه کاربردی PHP، آلوده شده باشد. این ابزار توسط یک محافظ زمان اجرا پشتیبانی شده است، که می‌تواند یک بررسی اضافی برای توابع حساس که با تجزیه و تحلیل ایستا یافت شده است، را اجرا کند. علاوه بر تجزیه و تحلیل ایستا، یک محافظ زمان اجرا اضافه شده است که به تفسیراتی که توسط کاربر آماده شده وابسته است. محافظ زمان اجرا ورودی ارسال شده کاربر را از هر کلمه کلیدی SQL، که در این ورودی تزریق شده فیلتر می‌کند. با این حال، اولین مرحله تجزیه و تحلیل ایستا یک تعداد بالایی از منفی کاذب و مثبت کاذب را گزارش می‌دهد.

هالفوند و همکاران [۴]، ابزار AMNESIA[1] (ابزار تجزیه و تحلیل و نظارت برای خنثی کردن حملات تزریق SQL) که برای تشخیص و پیشگیری از حملات تزریق SQL مورد استفاده قرار گرفته را ارائه دادند. این ابزار دو روش تجزیه و تحلیل ایستا و نظارت زمان اجرا را ترکیب می‌کند. روش تجزیه و تحلیل ایستا، یک مدل پرس‌وجوی SQL با استفاده از JSA[2] (تجزیه و تحلیل رشته جاوا) می‌سازد [۲۱]، که ساختار نقاط پرس‌وجویی که دسترسی مستقیم به پایگاه داده دارد و دنباله‌ای از نشانه‌هایی که

[1] Analysis and Monitoring for Neutralizing SQL Injection Attacks
[2] Java String Analysis

پرس‌وجو مشخص می‌کند را تعیین می‌کند. در پی آن، مرحله دیگر نظارت زمان اجرا است که تمام پرس‌وجوها را قبل از اینکه آن‌ها به پایگاه داده ارسال شوند بررسی می‌کند. این بررسی ساختار پرس‌وجوها را، در زمان اجرا بررسی می‌کند و آن‌ها را در برابر حملات موجود مقایسه می‌کند. نظارت زمان اجرا، به‌طور خاص دنباله‌ای از نشانه‌هایی که توسط مدل پرس‌وجوی SQL مشخص شده است، بررسی می‌کند، بنابراین اگر مرحله نظارت کشف کند، که پرس‌وجو منطبق با هیچ دنباله قبلی نیست، پرس‌وجو برای دسترسی به پایگاه داده ممنوع خواهد شد. این روش متشکل از دو مرحله است و محدودیت مرحله نظارت این است، که به نتیجه‌ای از مرحله تجزیه و تحلیل وابسته است. برای مثال، در یک رشته سخت رمز شده (مانند کاراکتر null 00%) یک عدم تطابق بین مدل پرس‌وجوی SQL و نظارت زمان اجرا به‌عنوان یکی از آخرین جستجوی کلمات کلیدی اصلی وجود دارد، نمی‌تواند یک رشته سخت رمزی که توسط مدل پرس‌وجوی SQL تشخیص داده شده را به‌دست آورد. شرما و همکاران [۲۲] نیز از رویکرد AMNESIA برای امنیت برنامه‌های کاربردی وب و پایگاه داده در برابر حملات تزریق SQL استفاده شده است.

همچنین کمالیس و همکاران [۲۳]، یک روش نظارت بر اساس یک الگوریتم تشخیص که ساختار نحوی برای همه دستورات SQL برنامه کاربردی را از طریق چندین مرحله مشخص می‌کند، را ارائه دادند. این مراحل هر دستور SQL برنامه کاربردی را با استفاده از تحلیل‌گر لغوی [۲۴]، برای تعیین دنباله‌ای از کلمات کلیدی SQL در این دستورات توصیف می‌کند. مرحله نظارت، بررسی می‌کند که اگر هر کد SQL تزریق‌شده در یک دستور SQL خاص بر اساس مشخصاتی از این دستور SQL وجود دارد، سپس دستورات SQL ناامن از اجرا روی پایگاه داده مسدود شود.

لم و همکاران [۲۵]، رویکرد قبلی خود [۲۶]، را که از یک روش تجزیه و تحلیل ایستا بر اساس جریان اطلاعات استفاده می‌کرد، را بهبود دادند. در بهبود خود، آن‌ها یک بازیابی خطای پویایی را اضافه کردند، که یک روش نظارت زمان اجرا بر اساس مشخصات PQL است، که در مرحله تجزیه و تحلیل ایستا توصیف شده است. این نظارت برای بازیابی بعضی موارد که خطاهایی را در طول تجزیه و تحلیل ایستا ایجاد می‌کنند، اضافه شده است. نظارت دنباله‌ای از محتویات پرس‌وجو از یک پرس‌وجوی خاص با مشخصات PQL آن را مقایسه می‌کند، اگر یک تفاوت بین آن‌ها وجود داشت این پرس‌وجو از اجرا روی پایگاه داده ممنوع خواهد شد.

مانیکانتا و همکاران [۲۷]، یک روش مشابه‌ای را پیشنهاد کردند، که با تجزیه و تحلیل همه لینک‌های آدرس برنامه کاربردی برای مشخص کردن پارامترهای آسیب‌پذیر و نقاط تزریق برنامه کاربردی با استفاده از w3af، آغاز می‌شود، که یک ابزار تجزیه و تحلیل ایستا است. مرحله بعدی ایجاد پرس‌وجوهای SQL قانونی بر اساس نتایج مرحله قبل می‌باشد. پرس‌وجوهای SQL قانونی، تمام پرس‌وجوهای معتبر برنامه کاربردی می‌باشد، که می‌تواند اجرا شود. مرحله نظارت با استفاده از GreenSQL به‌عنوان یک فایروال پایگاه داده یا جبهه پایانی پایگاه داده‌ای، که می‌تواند از پایگاه داده برنامه کاربردی در برابر تزریق SQL محافظت کند، به‌کار گرفته می‌شود. GreenSQL پرس‌وجوهای قانونی را نظارت می‌کند و همه حملات را رد می‌کند و تلاش‌های حمله را گزارش می‌کند. نویسنده در اینجا ترکیبی بین دو راه‌حل موجود برای رسیدن به بهترین نتایج از سامانه حفاظتی انجام می‌دهد. با این حال، GreenSQL حفاظت برای پایگاه داده اوراکل را پشتیبانی نمی‌کند.

Inyong Lee [۲۸] روشی بر اساس تحلیل ایستا و پویا برای مقابله با حملات تزریق SQL پیشنهاد کردند که از یک ایده ساده استفاده می‌کند. این روش بر اساس ادعای نویسندگان آن، قادر است انواع حملات تزریق SQL را شناسایی کند و وابستگی به پایگاه داده ندارد. اما جدا از همه مزایا، یکی از مشکلاتی که این روش از آن رنج می‌برد، نرخ بالای مثبت کاذب آن است، به‌طوری‌که تنها پرس‌وجوهایی مجوز دسترسی به پایگاه داده را پیدا می‌کنند که مطابق یکی از عناصر موجود در لیست پرس‌وجوهای ایستا باشد. این بازبینی بسیار سخت‌گیرانه است و باعث به وجود آمدن تعداد زیادی نتایج مثبت و منفی کاذب خواهد شد. همچنین این روش همه پرس‌وجوهای پویا را پوشش نمی‌دهد و فقط پرس‌وجوهای موجود در کد منبع را پشتیبانی می‌کند.

روش [۲۹] نیز از ترکیب تجزیه و تحلیل ایستا و پویا استفاده می‌کند و از دو فاز تشکیل شده است که به‌صورت ترکیبی با هم کار می‌کنند. فاز اول آن بر مبنای روش Inyong Lee است. در فاز اول، پرس‌وجوهایی که پایگاه داده ارجاع داده می‌شوند، با حذف مقادیر صفات خاصه آن، در برابر لیست پرس‌وجوهای ایستا مطابقت داده می‌شوند و در صورت مطابق بودن پرس‌وجو با لیست ایستا، مجاز اعلام می‌شود ولی در صورت شناسایی هرگونه مغایرتی، شانس دیگری به پرس‌وجو داده می‌شود و پرس‌وجو وارد مرحله دوم می‌شود. در مرحله دوم اجزای پرس‌وجو بر اساس فاکتورهای حمله بررسی می‌شود و الگوها و ارتباطات معنایی بین اجزای پرس‌وجو مورد بررسی قرار می‌گیرد.

الخالف و همکاران [۳۰] توابع اعتبارسنجی و پاک‌سازی ورودی را پیشنهاد دادند. هدف اصلی از بین بردن افزونگی در سمت سرور و سایر موارد در حال بررسی است. الگوریتم اصلاح کارکردهای اعتبارسنجی و پاک‌سازی را به‌عنوان ورودی درنظرگرفته و قصد دارد که تفاوت معنایی را اصلاح کند.

فراجتک و همکاران [۳۱] در کاهش کد اعتبارسنجی ورودی کاربر در برنامه‌های کاربردی وب با استفاده از افزونه Pex تمرکز کردند. Pex یک تولیدکننده ورودی آزمودن جعبه سفید برای برنامه‌های Net. است. این روش میزان کدی را که توسط توسعه‌دهندگان ایجاد شده است را کاهش می‌دهد. کدی که مقادیر پارامترهای ورودی روش را تائید می‌کند لازم نیست در جاوا اسکریپت کپی شده باشد. این کد هر زمان که یک تغییر در کد روش که درخواست مشتری را رسیدگی می‌کند ایجاد شود، بروز می‌شود.

لی و همکاران [۳۲] تمام آسیب‌پذیری‌ها و حملات شناخته‌شده را خلاصه می‌کنند آن‌ها یک بررسی از فنون و رویکردهای اخیر برای امن کردن سمت سرور برنامه‌های کاربردی وب ارائه می‌دهند.

چو و همکاران [۳۳] یک روش را پیشنهاد کردند که مقادیر ورودی برنامه‌های کاربردی وب مبتنی بر جاوا را با استفاده از ابزار ایستای کدبایت و اعتبارسنجی زمان اجرای ورودی تائید می‌کند. این روش روش‌های هدف یا سازنده‌های شی در فایل‌های کلاس جاوای کامپایل‌شده را جستجو می‌کند و ماژول‌های کدبایت را به‌صورت ایستا درج می‌کند.

مدیروس و همکاران [۳۴] در مورد استفاده از تجزیه و تحلیل ایستا برای تشخیص آسیب‌پذیری در برنامه‌های وب بحث کردند. آن‌ها سپس از خروجی استفاده می‌کنند و از روش‌های داده‌کاوی برای تشخیص استفاده می‌کنند و تعداد مثبت‌های کاذب را کاهش دادند.

سلمن و همکاران [۳۵] تجزیه و تحلیل پیشگویانه یادگیری ماشین را برای پیش‌بینی و جلوگیری از حملات تزریق SQL در برنامه کاربردی وب میزبان ابری را ارائه دادند. یک رابط برنامه‌نویسی برنامه کاربردی پراکسی وب برای پیش‌بینی دقیق حملات تزریق SQL مخرب در درخواست وب به‌عنوان یک سرویس وب برای محافظت از پایگاه داده نهایی ارائه شد.

جینگلینگ و همکاران [۳۶] برای کشف آسیب‌پذیری‌های تزریق SQL و XSS در برنامه‌های کاربردی وب، یک روش پویای ردیابی آسیب را پیشنهاد کردند.

WOVSQLI [۳۷]، یکی از ابزارهای حمله تزریق SQL که برای شناسایی حمله ایجاد شده است. این ابزار با بردار کلمه SQL و شبکه‌های عصبی LSTM برای شناسایی حمله تزریق SQL در برنامه کاربردی وب استفاده شد و یک مجموعه داده بزرگ برای نشان دادن دقت ابزار به‌کار گرفته شد.

آرومگام و همکاران [۳۸] سامانهی را پیشنهاد کردند که با هدف پیش‌بینی وقوع حمله تزریق SQL در یک سرور معین، با برنامه مستقرشده بر روی آن، از یک منبع داده شده در یک زمان خاص کار می‌کند. این پیش‌بینی با کمک ابزار JMeter قابل انجام است. Apache JMeter برای شبیه‌سازی داده‌های ثبت وقایع استفاده می‌شود. از این طریق آن می‌تواند پیش‌پردازش، استخراج ویژگی‌ها و طبقه‌بندی انجام دهد که سپس به یک مدل برای پیش‌بینی حملات تزریق SQL تبدیل می‌شود.

میتروپولوس و همکاران [۳۹] یک تحلیل در مورد سازوکارهای دفاعی مختلف در برابر حملات تزریق SQL ارائه دادند و مدلی پیشنهاد کردند که نقاط ضعف اصلی در ایجاد این حملات را نشان می‌دهد، سپس براساس ویژگی‌های دقت، کارایی، استقرار، امنیت و در دسترس بودن آن‌ها مجموعه‌ای از ۴۱ سازوکارهای دفاعی که قبلاً پیشنهاد شده بود را طبقه‌بندی و تجزیه و تحلیل کردند.

تانگ و همکاران [۴۰] طرحی را پیشنهاد کردند که در آن ویژگی‌های ترافیک HTTP در مجموعه‌های آموزشی استخراج می‌شود و از شبکه عصبی عمیق LSTM و مجموعه داده‌های آموزشی MLP استفاده می‌کند.

[۴۱] یک بررسی گسترده از انواع روش‌های مختلف تشخیص و پیشگیری از حملات تزریق SQL، از ابعاد مختلف مانند ماهیت دفاع، روش تجزیه و تحلیل، زمان تشخیص، مکان تشخیص و مثبت و منفی کاذب ارائه داده است.

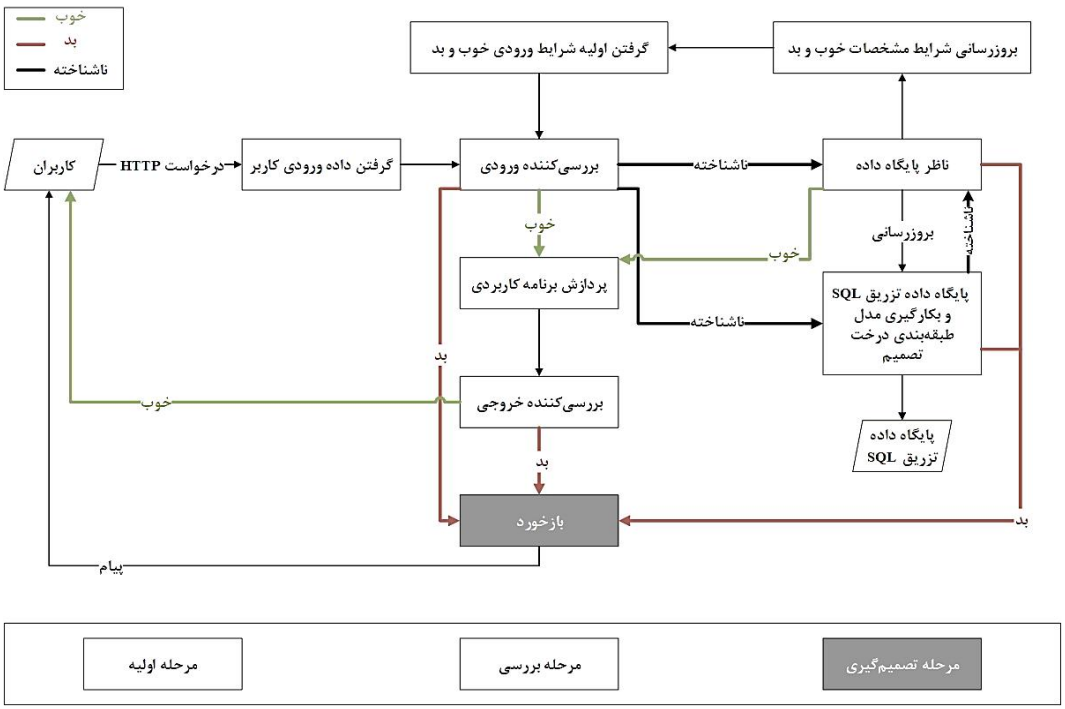
[۴۲] روشی برای شناسایی حملات تزریق SQL از طرق مدل کردن پرس‌وجوها به گرافی از توکن‌ها و نیز استفاده از معیار مرکزیت گره‌ها در گراف، جهت آموزش دسته‌بندی‌کننده ماشین بردار پشتیبان (SVM) را ارائه دادند و با به‌کارگیری هسته‌های مختلف آن را آزمایش کردند.

[۴۳] روشی برای شناسایی حملات تزریق SQL در سطح پایگاه داده با استفاده از خوشه‌بندی فازی پیشنهاد کردند که ابتدا مؤثرترین ویژگی‌هایی که یک پرس‌وجوی نرمال را از نوع مخرب متمایز می‌کند، استخراج شوند. این ویژگی‌ها مبتنی بر تکرار کاراکترهای خطرناک در یک پرس‌وجوی SQL تعریف شده‌اند.

این پژوهش از تجزیه و تحلیل ایستا و پویا استفاده می‌کند و یک روش، برای تشخیص و پیشگیری از حملات تزریق SQL در زمان اجرا ارائه می‌دهد، که می‌تواند حملات موجود و جدید را تشخیص دهد، به‌علاوه بر حملات به‌طور مداوم نظارت کند. روش تشخیص و پیشگیری پیشنهادی، با نظارت زمان اجرا و به‌کارگیری طبقه‌بندی درخت تصمیم بر روی پایگاه داده تزریق SQL، حملات تزریق SQL موجود را مسدود می‌کند همچنین با استفاده از ناظر پایگاه داده حملات جدید را تشخیص می‌دهد. این روش، مزیت یادگیری ماشین را نیز دارد. این روش در مقایسه با روش تحلیل ساختاری و معنایی پرس‌وجو [۲۹] و روش‌های دیگر از نتایج مثبت کاذب کمتری برخوردار است و به‌دلیل جزء ناظر پایگاه داده قادر به تشخیص انواع حملات تزریق SQL جدید می‌باشد.

۴- روش تشخیص و پیشگیری پیشنهادی

اهداف اصلی از روش تشخیص و پیشگیری پیشنهادی، نظارت و مسدود کردن حملات تزریق SQL می‌باشد، که به‌منظور به‌دست آوردن دسترسی غیرمجاز به برنامه کاربردی وب و پایگاه داده آن‌ها به‌کار گرفته می‌شوند. روش تشخیص و پیشگیری پیشنهادی، به‌عنوان ابزار نظارت و تائید زمان اجرا، به‌منظور مسدود کردن ورودی‌های کاربران مخرب استفاده می‌شود. روش پیشنهادی، به‌وسیله مشخص کردن الگوهای حمله موجود آغاز می‌شود. شکل (۱) معماری اصلی از روش پیشنهادی را نشان می‌دهد. همان‌طور که شکل (۱) نشان می‌دهد، روش پیشنهادی هنگامی‌که کاربر داده‌ای را وارد و آن را به صفحه برنامه کاربردی وب ارسال می‌کند، آغاز به کار می‌کند. داده از ماشین سرویس‌گیرنده به سرور وب، توسط پروتکل HTTP فرستاده خواهد شد. علاوه بر این، روش پیشنهادی داده ارسال‌شده را در جزء گرفتن داده ورودی کاربر، استخراج می‌کند. استخراج داده، به نقاط اتصال یک برنامه کاربردی وب، که داده ارسالی را به متغیرهای آن در برنامه کاربردی وب آدرس‌دهی می‌کند، وابسته است. نقاط اتصال با استفاده از یکی از ابزارهای تجزیه و تحلیل ایستای موجود می‌توانند مشخص شوند، مانند ابزار pixy، برای برنامه‌های کاربردی که با PHP توسعه یافته‌اند. داده، در برابر الگوهای حمله موجود با استفاده از جزء بررسی‌کننده ورودی، بررسی می‌شود. نتیجه بررسی، تعیین می‌کند که آیا ورودی خوب یا بد و یا ناشناخته است و این اطلاعات برای بررسی حملات مربوطه استفاده خواهد شد.



شکل (۱): معماری روش تشخیص و پیشگیری پیشنهادی

روش پیشنهادی، چهار جزء بررسی دارد که عبارت‌اند از: بررسی‌کننده ورودی، بررسی‌کننده خروجی، به‌کارگیری مدل طبقه‌بندی درخت تصمیم بر روی پایگاه داده تزریق SQL و ناظر پایگاه داده. هریک از این چهار جزء، روش‌های مختلفی برای بررسی ورودی کاربر دارند. داده ورودی کاربر، دارای سه وضعیت خوب یا بد و یا ناشناخته است. اولین مرحله بررسی، بررسی‌کننده ورودی، برای تجزیه و تحلیل داده ورودی کاربر با استفاده از دو گام صورت می‌گیرد. اولین گام، ورودی کاربر را برای بررسی

این‌که آیا آن، ورودی خوبی است تجزیه و تحلیل خواهد کرد، که در اینجا منظور از ورودی خوب، داده ورودی غیر مخرب کاربر می‌باشد و منظور از ورودی بد، داده ورودی مخرب یا کد تزریق‌شده کاربر می‌باشد. اگر اولین گام تعیین کند که ورودی خوب نیست، سپس گام دوم، آن ورودی‌ها را با الگوهای حمله موجود، مقایسه خواهد کرد. بنابراین، اگر گام اول در نظر بگیرد که ورودی کاربر خوب است، سپس آن برای پردازش توسط جزء پردازش برنامه کاربردی فرستاده خواهد شد. اگر ورودی کاربر، با یک الگوی حمله موجود، مطابقت داشته باشد، ورودی برای پردازش، پذیرفته نخواهد شد و به کاربر، توسط جزء بازخورد روش پیشنهادی، به‌عنوان مرحله تصمیم‌گیری، اطلاع داده می‌شود. اگر بررسی‌کننده ورودی، نتواند خوب یا بد بودن ورودی کاربر را تعیین کند، یعنی داده ورودی کاربر، برای الگوریتم بررسی‌کننده ورودی، ناشناخته و جدید باشد، سپس روش پیشنهادی، ناظر پایگاه داده را برای تعیین اینکه ورودی کاربر روی موتور پایگاه داده چه تأثیری دارد، اجرا خواهد کرد. اگر ورودی کاربر توسط ناظر پایگاه داده پذیرفته شود، سپس روش پیشنهادی، ورودی کاربر را به جزء پردازش برنامه کاربردی می‌فرستد، اگر ورودی کاربر پذیرفته نشود، سپس روش پیشنهادی، ورودی کاربر را رد خواهد کرد و با استفاده از جزء بازخورد، به کاربر اطلاع می‌دهد. روش پیشنهادی، همچنین پایگاه داده تزریق SQL و الگوی حملات موجود را با اطلاعاتی که یک حمله تزریق SQL جدید، روی برنامه کاربردی تحت وب استفاده کرده است، به‌روزرسانی می‌کند. پایگاه داده تزریق SQL توسط ناظر پایگاه داده برای یک ورودی بد جدید به‌روزرسانی می‌شود، با به‌روزرسانی و اضافه کردن امضای حملات ناشی از ورودی بد کاربر در پایگاه داده تزریق SQL و به‌کارگیری طبقه‌بندی درخت تصمیم‌گیری مبتنی بر امضای حملات، ورودی‌های بد کاربر با ویژگی‌های مشابه ویژگی‌های ورودی بد کاربر که قبلاً توسط ناظر پایگاه داده به‌عنوان ورودی بد شناخته شده بودند، فیلتر می‌شوند، به‌این‌ترتیب زمانی که پایگاه داده تزریق SQL به‌روزرسانی شود، داده ورودی‌ای که در جزء بررسی‌کننده ورودی ناشناخته باشد ابتدا در پایگاه داده تزریق SQL با استفاده از الگوریتم درخت تصمیم، در امضای حملات بررسی می‌شود، اگر در پایگاه داده تزریق SQL وجود داشت فیلتر می‌شود و اگر در این پایگاه داده وجود نداشت برای تشخیص به ناظر پایگاه داده فرستاده می‌شود. یک درخت تصمیم، یک روش داده‌کاوی است که از یک ساختار درختی برای تصمیم‌گیری‌ها و هزینه احتمالی آن‌ها استفاده می‌کند.

آخرین جزء از مرحله بررسی، بررسی‌کننده خروجی می‌باشد که برای تعیین اینکه، پیامی که از موتور پایگاه داده با کاربر ارتباط برقرار می‌کند، شامل هیچ اطلاعاتی درباره نوع پایگاه داده یا ساختار آن نباشد، مورد استفاده قرار می‌گیرد، زیرا این انواع از پیام‌ها ناامن هستند. علاوه بر این، بررسی‌کننده خروجی، پیام‌های ناامن را با استفاده از فراخوانی‌های کتابخانه‌ای در زبان برنامه‌نویسی که در طول برنامه کاربردی وب به‌کار گرفته شده است را مسدود خواهد کرد.

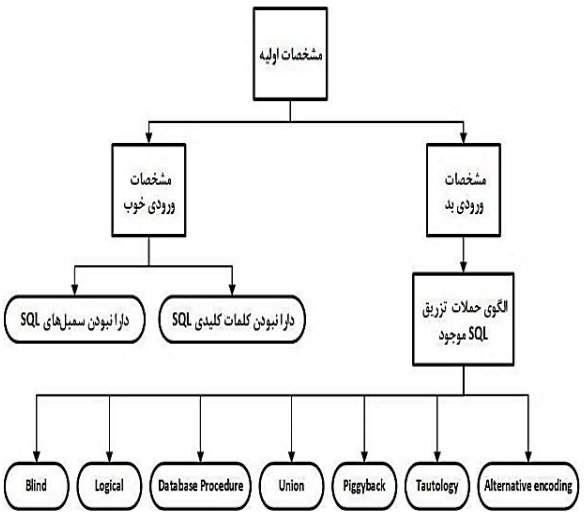
روش پیشنهادی از سه مرحله تشکیل می‌شود که شامل مرحله اولیه، مرحله بررسی و مرحله تصمیم‌گیری می‌باشد. برخی از اجزای این مراحل در زیر بحث خواهد شد.

۴-۱- مرحله اولیه (دریافت داده)

مرحله اولیه شامل چندین گام است که نیازمند این است قبل از اینکه داده بتواند در جزء بررسی‌کننده ورودی بررسی شود، انجام گیرد. در زیر مرحله اولیه و برخی از اجزای آن شرح داده می‌شود.

۴-۱-۱- جزء گرفتن اولیه شرایط ورودی خوب و بد

جزء گرفتن اولیه شرایط ورودی کاربر، نیازمند تعیین ورودی خوب و بد در سطح کاراکتر می‌باشد، همان‌طور که در شکل (۲) نشان داده شده است. ورودی خوب، نباید شامل هر سمبل بد، شبیه سینگل کوتیشن[1] و دابل کوتیشن یا ستاره[2] باشد، یا ورودی خوب نباید شامل کلمات کلیدی SQL باشد که برای حمله به پایگاه داده برنامه کاربردی وب بتوان از آن‌ها استفاده کرد. بنابراین ورودی بد، توسط توصیف بعضی از الگوهای حمله مانند پرس‌وجوی درست‌نما، پرس‌وجوی اجتماع، پرس‌وجوی Piggy-Back و غیره مشخص خواهد شد. مشخصات حملات موجود، برای هر حمله که کاراکتر تزریق یکسان به‌کار می‌برند، ادغام خواهد شد. بدین معناست که فرمول مشخصات هر نوع را به‌طور جداگانه توصیف نمی‌کند.



شکل (۲): مشخصات اولیه ورودی خوب و بد

برای مثال حملات تزریق کور می‌تواند با استفاده از یک سینگل کوتیشن شبیه به یک حمله Piggy-Back، انجام شود، بنابراین مشخصاتی از حملات، که یک سینگل کوتیشن را شامل

[1] Single quotation
[2] Star

می‌شوند، نیازی نیست که در فرمول تشخیص جداگانه باشد و هر دو حمله با استفاده از یک مشخصات حملاتی که سینگل کوتیشن را پوشش می‌دهند، می‌توانند تشخیص داده شوند. بنابراین مشخصات اولیه خوب و بد برای آغاز روش پیشنهادی استفاده خواهد شد.

۴-۱-۲- جزء گرفتن داده

این جزء، داده را برای تجزیه و تحلیل توسط جزء بررسی‌کننده ورودی، آماده می‌کند، بنابراین، اطلاعات ارائه شده (آدرس IP کاربر، زمان ارسال درخواست و داده ارسال‌شده کاربر) از طریق درخواست‌های HTTP به سرور ارسال می‌شود.

جزء گرفتن داده، متشکل از دو گام است، که گام اول داده را استخراج می‌کند و گام دوم داده را نرمال‌سازی می‌کند. دو روال نرمال‌سازی وجود دارد که عبارت‌اند از: ۱) روال تبدیل به حروف کوچک که برای تبدیل ورودی کاربر به حروف کوچک استفاده می‌شود. ۲) روال کاهش فضاهای خالی که برای حذف هر فضای اضافی در ورودی کاربر استفاده می‌شود. این روال‌ها به‌صورت پیوسته برای هر تراکنش اجرا می‌شوند، الگوریتم (۱) شبه کدی از نرمال‌سازی داده در جزء گرفتن داده را نشان می‌دهد. بنابراین، داده استخراج شده می‌تواند با استفاده از بررسی‌کننده ورودی مورد تجزیه و تحلیل قرار گیرد. داده کاربر، به سبک آرایه دریافت می‌شود و تنها داده ارائه‌شده مورد تجزیه و تحلیل قرار می‌گیرد و وضعیتی مانند، امن یا ناامن بودن داده تعیین می‌شود.

Algorithm 1

Function Normalization(Req*uest*)
for $i \leftarrow 1$ *to n do*
$R = LowerCase(\text{Req}_i)$
$\text{Req}_i = R$
end for
for $i \leftarrow 1$ *to n do*
$D = DecreaseSpaces(\text{Req}_i)$
$\text{Req}_i = D$
end for
return Req*uest*

الگوریتم (۱): شبه کدی از جزء گرفتن داده

۴-۲- مرحله بررسی

در این مرحله، سامانه داده‌ای را که از جزء گرفتن داده می‌گیرد، تجزیه و تحلیل خواهد کرد. این مرحله چهار جزء بررسی دارد که برخی از اجزای آن در زیر توضیح داده می‌شوند.

۴-۲-۱- جزء بررسی‌کننده ورودی

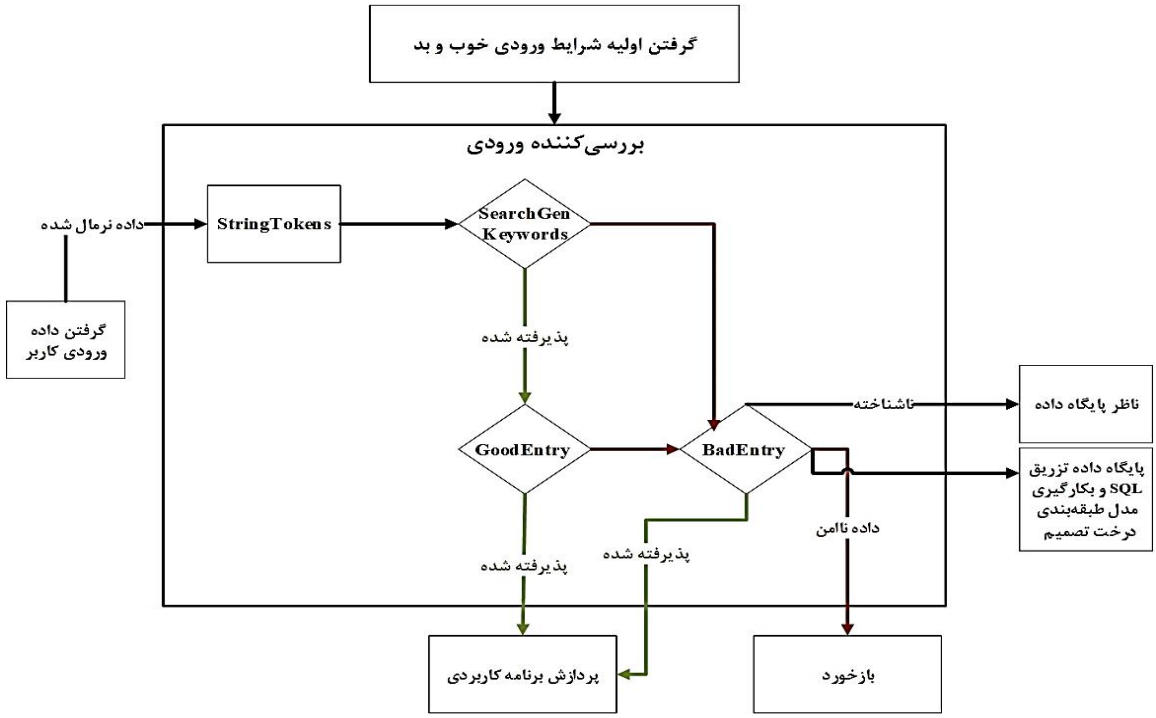
این جزء قلب روش پیشنهادی می‌باشد، همچنین آن مرحله بعدی از روش پیشنهادی، که آیا ورودی کاربر، برای جزء پردازش نرمال برنامه کاربردی اقدام شود یا در جزء ناظر پایگاه داده ادامه داده شود، را تعیین می‌کند. علاوه بر این، جزء بررسی‌کننده ورودی، الگوهای حمله موجود که قبلاً توسط جزء گرفتن اولیه شرایط خوب و بد ورودی کاربر، آماده شده است را استفاده خواهد کرد. شکل (۳) ساختار بررسی‌کننده ورودی را نشان می‌دهد. اولین گام از بررسی‌کننده ورودی، جدا کردن رشته ورودی کاربر به نشانه‌ها می‌باشد (روال StringTokens) و سپس آن توسط روال‌های دیگری مانند SearchGenKeywords، SearchGoodEntry و SearchBadEntry دنبال می‌شود.

الگوریتم (۲) بخشی از شبه کد روش تشخیص و پیشگیری پیشنهادی را نشان می‌دهد که در آن ابتدا ورودی یا درخواست کاربر استخراج می‌شود سپس آدرس IP و زمان ارسال درخواست استخراج می‌شود سپس تابع نرمال‌سازی که در الگوریتم (۱) بیان شده است، فراخوانی می‌شود و پس از آن توابع بررسی‌کننده ورودی فراخوانی می‌شوند. هر یک از این توابع به‌طور جداگانه در الگوریتم (۳) نشان داده شده است. اولین تابع در الگوریتم (۳)، تابع StringTokens، برای جداسازی رشته ورودی کاربر به نشانه‌ها می‌باشد. برای تجزیه و تحلیل ورودی باید اول نشانه‌های ورودی را جداسازی کنیم. تابع StringTokens دارای دو مرحله است: مرحله اول، یک کاراکتر غیر فاصله‌ای از رشته ورودی را تعیین می‌کند و مرحله دوم، رشته ورودی را در یک آرایه‌ای از نشانه‌ها که می‌تواند برای جستجوی کلمات کلیدی خاص SQL استفاده شود، جداسازی می‌کند.

Algorithm 2

for $i \leftarrow 1$ *to n do*
Put Req_i *to* Req*uest_queue*
end for
while(Req*uest_queue is not empty*)
foreach req in Req*uest_queue do*
$Date = Date\_\text{Req}_i(\ )$
$ip = ip\_\text{Req}_i(\ )$
$Normalization\_\text{Req}uest(\text{Req}_i)$
StringTokens(Req*uest*, *A*)
SearchGenKeywords(*S*, *X*)
SearchGoodEntry(*S*, *X*)
SearchBadEntry(*S*, *X*)
end for
end while

الگوریتم (۲): بخشی از شبه کد روش تشخیص و پیشگیری پیشنهادی



شکل (۳): بررسی‌کننده ورودی

```
Function SearchGenKeywords(S, X)
 StringTokens(Request, A)
SQLKeys = ["select","drop","update","delete","alter"
,"create","union","declare","bigen","exec","ascii"]
 i = 0
 j = 0
 R = 'g'
 while i < |SQLKeys| do
   while j < |A| do
    if (SQLKeys[i] == A[ j])
          R = 'n'
     else
          R = R
return(X = R)
Function GoodEntry(S, X)
 R = 'g'
 i = 0
while i < |S| do
 if (S[0..2] = "0x" or S[i..i + 3] = " 0x")
   then  {i = |S|
          R = 'n'}
   else if (ascii(S[i]) >= 97 and ascii(S[i]) <= 122)
   then  {R = 'R'
          i = i + 1}
```

Algorithm 3

```
Function StringTokens(Request, A)
 for i ← 1 to n do
 S = Req_i
exists A1, A2, E : { List(A2, |S|) and A2 = [ ] }
 for i ⟨ |S| do
  if S[i] ! = " "
  then A_2 = A_2 + [i]
 else
  A_2 = A_2
end for
List(A1, |S|) and A1 = [ ] and E = A2[0]
 for i ⟨ |A2 − 1| do
 if (A2[i + 1] − A2[i] == 1)
 then {A1 = A1
          E = E}
 else {A1 = A1 + [S[E..A2[i] + 1] ]
          E = A2[i + 1] }
end for
return(A = A1 + [S[E..A2[|A2| − 1] + 1] ])
```

```
else if (ascii(S[i]) >= 48 and ascii(S[i]) <= 58)
then   {R = 'R'
        i = i + 1}
else if (ascii(S[i]) == 32 or ascii(S[i]) == 33 or
 ascii(S[i]) == 63 or ascii(S[i]) == 95 or
 ascii(S[i]) == 43 or ascii(S[i]) == 61)
then   {R = 'R'
        i = i + 1}
else{R = 'n'
 i = |S|
 return 'n' } /*return n means not good entry*/
Function SearchBadEntry(S, X)
R = 'u'
i = 0
while i < |S| do
 if ((S[i..i+2]= "--" and (i+2 <= |S|)) or
  (S[i..i+3]= "- -" and (i+3 <= |S|)))
        then{i = |S|
         R = 'b'}
 else    { R = R
        i = i + 1 }
```

الگوریتم (۳): بخشی از شبه کد بررسی‌کننده ورودی

هنگامی‌که تابع SearchGenKeywords، فراخوانی می‌شود، این تابع برای شایع‌ترین کلمات کلیدی تزریق SQL که بعد از یک علامت تزریق SQL، مانند سمی‌کالن (;) و سینگل کوتیشن (') قرار می‌گیرند، جستجو انجام می‌دهد. تابع با تبدیل ورودی به آرایه‌ای از نشانه‌ها با استفاده از فراخوانی تابع StringTokens شروع می‌شود. آرایه بازگشتی از نشانه‌های رشته‌ای با کلمات کلیدی SQL رایج که می‌توانند در یک حمله تزریق SQL مورد استفاده قرار گیرند، به‌عنوان بخشی از تجزیه و تحلیل ورودی کاربر مقایسه می‌شود. این تابع برای مشخص کردن اینکه رشته ورودی شامل هر کلمه کلیدی SQL می‌شود یا نه، مورد بررسی قرار می‌گیرد.

تابع SearchGoodEntry، یکی از توابع اصلی برای تجزیه و تحلیل ورودی کاربر است. تابع به‌عنوان ورودی، یک رشته نرمال‌شده دارد و ابتدا برای یک روش تزریق SQL شبه کدگذاری شده، با مقایسه ورودی کاربر در سطح کاراکتر با یک نوع از این تزریق مانند '0X...' بررسی می‌کند، اگر منطبق نباشد روند تجزیه و تحلیل به دنبال بررسی کاراکترهای ورودی در محدوده '0' تا '9' و 'a' تا 'z' با استفاده از کد اسکی خواهد بود. علاوه بر این، برخی از نمادها وجود دارند که امن در نظر گرفته می‌شوند، زیرا آن‌ها به‌عنوان یک کلید در حمله تزریق SQL استفاده نمی‌شوند و می‌توانند هنگام استفاده از یک برنامه کاربردی وب به‌عنوان بخشی از ورودی کاربر استفاده شوند. مراحل تجزیه و تحلیل به‌طور پیوسته برای هر کاراکتر اجرا می‌شوند، بنابراین اگر هر یک از مراحل ذکرشده دارای یک تطبیق مثبت باشد تابع 'n' را بازمی‌گرداند که به این معناست ورودی امن نیست.

SearchBadEntry، یکی از توابع اصلی برای تجزیه و تحلیل ورودی کاربر است که ورودی‌های کاربر را بررسی می‌کند و آن را در برابر الگوهای حملات موجود مقایسه می‌کند. این تابع دارای چند بخش است: بخش اول، برای کلیدهای تزریق SQL مانند یک سینگل کوتیشن به‌عنوان یک کاراکتر جداکننده، سمی‌کالن به‌عنوان جداکننده پرس‌وجو، نماد (#) و (ـ ـ) به‌عنوان کاراکتر جداکننده توضیح و یک /* */ به‌عنوان یک جداکننده توضیح، بررسی انجام می‌دهد. به‌عنوان مثال نحوه این بررسی در مورد دابل‌دش (ـ ـ) در تابع SearchBadEntry نشان داده شده است. بررسی این بخش، بر اساس مقایسه بین توالی کاراکترها در رشته S با کاراکتر(ـ ـ) با فاصله یا بدون فاصله می‌باشد. بخش دوم نشانه یا کلمه‌ای که بعد از یک کلید تزریق قرار می‌گیرد را بررسی می‌کند و آخرین بخش بررسی آنچه که بعد از این نشانه یا کلمه تزریق قرار می‌گیرد، می‌باشد. تابع SearchBadEntry، دارای مراحل بسیاری برای تجزیه و تحلیل ورودی کاربر است. آوردن تمامی این مراحل در قالب الگوریتم، از حوصله این مقاله خارج است. جزء بررسی‌کننده ورودی، ورودی کاربر را در برابر الگوهای حملات موجود تجزیه و تحلیل خواهد کرد و نتیجه یکی از سه احتمال زیر می‌باشد:

داده ورودی کاربر خوب است، که بدین معناست که ورودی کاربر، هیچ سمبل یا کلمات کلیدی که در حملات تزریق SQL موجود استفاده می‌شود، شامل نشود. سپس داده را به سرور برنامه کاربردی وب، برای پردازش نرمال، عبور می‌دهد. دیگر احتمال این است، که داده ورودی بد است، بنابراین داده پذیرفته نخواهد شد و یک پیام برای ارسال به کاربر توسط جزء بازخورد آماده خواهد شد. آخرین احتمال، این است که داده ورودی ناشناخته باشد، در این مورد ناظر پایگاه داده، وضعیت خوب یا بد بودن داده ورودی کاربر را بر طبق تأثیری که آن داده، روی پایگاه داده می‌گذارد، تعیین می‌کند و پایگاه داده تزریق SQL به‌روزرسانی می‌شود. جزء ناظر پایگاه داده، داده ورودی را به‌وسیله اعتبارسنجی چهار شرط بررسی می‌کند که تعیین کند داده ورودی کاربر، امن یا ناامن است. جزء ناظر پایگاه داده، در بخش بعدی توضیح داده می‌شود.

۴-۲-۲- جزء ناظر پایگاه داده

این جزء، موارد ورودی ناشناخته که به‌وسیله جزء بررسی‌کننده ورودی، ناشناخته است، را بررسی می‌کند. هدف از جزء ناظر پایگاه داده، این است که تعیین شود برای پایگاه داده، روی تراکنشی از ورودی کاربر، دقیقاً چه اتفاقی خواهد افتاد و این توسط نظارت بر نتیجه‌ای از هر تراکنش پایگاه داده انجام خواهد شد.

نظارت بر تراکنش پایگاه داده به توسعه‌دهنده برنامه کاربردی وب، احتیاج دارد زیرا جزء ناظر پایگاه داده، به توسعه‌دهنده برای مشخص کردن نتیجه مورد انتظار از هر تراکنش که توسط برنامه کاربردی وب اجرا می‌شود، نیاز دارد.

این جزء، با موارد ورودی ناشناخته، به‌منظور تعیین پاسخ درست موتور پایگاه داده، مربوط به این نوع ورودی کاربر می‌پردازد. ناظر پایگاه داده برای تعیین حملات جدید، با استفاده از زبان توسعه برنامه کاربردی به‌منظور مشاهده موارد ناشناخته، توسعه یافته است. ناظر پایگاه داده تراکنش را نظارت می‌کند تا چهار شرط زیر را بررسی کند:

- نوع تراکنش در زمان اجرا، با نوع مورد انتظار که توسط توسعه‌دهنده تعیین شده است، یکسان باشد.
- نام جدول تراکنش در زمان اجرا، با نام جدول مورد انتظار که توسط توسعه‌دهنده تعیین شده است، یکسان باشد.
- تعداد رکوردهای تراکنش در زمان اجرا، با تعداد رکوردهای مورد انتظار که توسط توسعه‌دهنده تعیین شده است، یکسان باشد.
- نوع کاربری تراکنش در زمان اجرا، با نوع کاربری مورد انتظار که توسط توسعه‌دهنده تعیین شده است، یکسان باشد.

باید توجه داشت که ناظر پایگاه داده فقط می‌تواند با تراکنش‌های قابل بازیابی و همچنین فرمان‌های بدون DLL[1] (زبان‌های تعریف داده) مانند create، drop و alter table مقابله کند، زیرا فرمان‌های DLL تزریق‌شده نمی‌توانند توسط فرمان عقب‌گرد دوباره بازیابی شوند، بنابراین این تراکنش‌ها باید قبلاً توسط جزء بررسی‌کننده ورودی رد شده باشد یا پذیرفته نشده باشد.

۴-۲-۳- جزء پایگاه داده تزریق SQL و به‌کارگیری مدل طبقه‌بندی درخت تصمیم

هنگامی‌که داده ورودی کاربر برای جزء بررسی‌کننده ورودی ناشناخته بود، این داده برای جزء ناظر پایگاه داده برای بررسی اینکه رشته ورودی کاربر چه تأثیری روی پایگاه داده دارد، ارسال می‌شود، اگر جزء ناظر پایگاه داده تشخیص دهد که ورودی کاربر بد است، امضای حمله در پایگاه داده تزریق SQL ذخیره می‌شود. به‌تدریج پایگاه داده تزریق SQL توسط امضای حملات جدید به‌روزرسانی می‌شود، با به‌روزرسانی امضای حملات ناشی از ورودی بد کاربر در پایگاه داده تزریق SQL و به‌کارگیری طبقه‌بندی درخت تصمیم‌گیری مبتنی بر امضای حملات، ورودی‌های بد کاربر با ویژگی‌های مشابه ویژگی‌های ورودی بد کاربر که قبلاً توسط ناظر پایگاه داده به‌عنوان ورودی بد شناخته شده بودند و در پایگاه داده تزریق SQL ذخیره شده بودند، فیلتر می‌شوند و لازم نیست بار دیگر به ناظر پایگاه داده برای بررسی ارسال شود. روش پیشنهادی، پایگاه داده تزریق SQL را با اطلاعاتی که یک حمله تزریق SQL جدید، روی برنامه کاربردی تحت وب استفاده کرده است، به‌روزرسانی می‌کند. به‌این‌ترتیب زمانی که پایگاه داده تزریق SQL، به‌روزرسانی شود، داده ورودی‌ای که در جزء بررسی‌کننده ورودی ناشناخته باشد ابتدا در پایگاه داده تزریق SQL، با استفاده از الگوریتم درخت تصمیم، در امضای حملات بررسی می‌شود، اگر در پایگاه داده تزریق SQL وجود داشت فیلتر می‌شود و اگر در این پایگاه داده وجود نداشت برای تشخیص به ناظر پایگاه داده فرستاده می‌شود. یک درخت تصمیم، یک روش داده‌کاوی است که از یک ساختار درختی برای تصمیم‌گیری‌ها و هزینه احتمالی آن‌ها استفاده می‌کند. یک درخت تصمیم‌گیری یک گراف و ساختار داده‌ی ترکیبی درختی است که گره داخلی یک آماده‌سازی روی یک ویژگی را نشان می‌دهد، هر انشعاب گره، نتایجی از آزمون و هر گره برگ یک برچسب کلاس را نشان می‌دهد. مسیرهایی از مبدأ به برگ، قوانین طبقه‌بندی را نشان می‌دهد. مدل درخت تصمیم تزریق SQL که در آن شانس آسیب‌پذیری‌ها به‌دست می‌آید و پایگاه داده تزریق SQL نهایی برای استفاده به‌عنوان داده طبقه‌بندی ساخته می‌شود. برای بررسی و یافتن تزریق SQL، یک پایگاه داده تعریف می‌شود که این پایگاه داده همه خطای پایگاه داده مربوط به حملات تزریق SQL مختلف را ذخیره می‌کند. درخواست سرویس‌گیرنده به سرور با استفاده از درخت تصمیم فیلتر می‌شود که در آن پرس‌وجوی SQL به‌عنوان کلاس حمله و کلاس غیر حمله بر اساس اطلاعات قبلی طبقه‌بندی می‌شود. الگوریتم (۴) بر این اساس اول داده وابسته به حمله تزریق SQL از منابع و با استفاده از الگوریتم طبقه‌بندی درخت تصمیم برای

[1] Data Definition Languages

طبقه‌بندی تزریق SQL را به‌دست خواهد آورد. قوانین کلاس برای طبقه‌بندی پرس‌وجوهای SQL به‌عنوان حملات و غیر حملات ساخته می‌شود. زمانی‌که یک درخواست به سرویس‌دهنده وب ارسال می‌شود سرویس‌دهنده درخواست را به الگوریتم تشخیص حمله تزریق SQL ارسال می‌کند که در الگوریتم (۵) نشان داده شده است و در آن استفاده از تحلیل‌گر درخت تصمیم را مهیا می‌کند که در الگوریتم (۶) نشان داده شده است.

Algorithm 4

*Step*1: *Check the SQL injection database*
*Step*2: *For each SQL query I*
*Step*3: *Find the* inf *ormation gain ratio*
*Step*4: *Based up on* inf *ormation gain*
*Step*5: *Create a decision node that splits with* inf *ormation gain*
*Step*6: *Based up on nodes construct the tree for classifying SQL queries as attack and non attacks.*

الگوریتم (۴): الگوریتم ساختار درخت تصمیم بر روی پایگاه داده تزریق SQL .

Algorithm 5

*Step*1: *Create a Queue Data Structure to store inco* min *g requests*
*Step*2: *Decide whether given requests is to access database or not*
*Step*3: *If the request is to t* arg *et query then send it to Decision tree Analayzer and record response.*
*Step*4: *If the response is positive direct to database.*
*Step*5: *Else report failure message to user.*
*Step*6: *Finish.*

الگوریتم (۵): الگوریتم روال تشخیص حمله تزریق SQL .

Algorithm 6

*Step*1: *Get the SQL request from web server*
*Step*2: *Map to the SQL injection Decision tree and find class of SQL query*
*Step*3: *If the resulted class is attack class give the negative result*
*Step*4: *Else give the positive response.*

الگوریتم (۶): الگوریتم تحلیل‌گر درخت تصمیم.

۴-۳- مرحله تصمیم‌گیری

این مرحله از روش پیشنهادی، به نتایجی از مرحله بررسی وابسته است و شامل جزء بازخورد می‌باشد. جزء بازخورد، پیامی که به کاربر با توجه به مواردی از داده ورودی بد فرستاده خواهد شد را مهیا می‌کند. اگر ورودی کاربر بد باشد و ورودی به‌عنوان ناامن گرفتار شده باشد، سپس روش پیشنهادی به این ورودی توسط یک پیام آماده شده بسته به نوع ورودی بد، پاسخ خواهد داد.

۵- ارزیابی روش پیشنهادی

روش پیشنهادی با نظارت مداوم می‌تواند در برابر انواع حملات تزریق SQL بیان‌شده در بخش ۲-۱- مقابله کند. در جدول (۱) روش‌های پیشین با روش پیشنهادی از نظر پوشش انواع حملات مقایسه شده است. همان‌طور که ملاحظه می‌شود روش ارائه‌شده انواع حملات تزریق SQL را پوشش می‌دهد.

جدول (۱): ارزیابی روش پیشنهادی از نظر میزان مقابله با انواع حملات تزریق SQL .

روش	پرس‌وجوی درست‌نما	نادرست منطقی	Piggy-back	اجتماع	روال ذخیره‌شده	استنتاج	کدگذاری جایگزین
AMNESIA	√	√	√	√	×	√	√
SQLCHECK	√	√	√	√	×	√	√
SQLGuard	√	√	√	√	×	√	√
SQLrand	√	×	√	√	×	√	×
Tautology checker	√	×	×	×	×	×	×
SQL DOM	√	√	√	√	×	√	√
WebSSARI	√	√	√	√	√	√	√
Inyong Lee	√	√	√	√	√	√	√
Proposed Method	√	√	√	√	√	√	√

مطالعات و ابزارهای پویش آسیب‌پذیری‌های برنامه‌های کاربردی تحت وب بسیاری وجود دارد، که با مشکل تزریق SQL مقابله کرده‌اند. روش پیشنهادی، با ابزارهای پویش برنامه‌های کاربردی وب مانند Nikto و Acunetix مقایسه نخواهد شد، زیرا آن‌ها از روش‌های آزمودن جعبه سیاه استفاده می‌کنند و با آسیب‌پذیری‌های مختلف برنامه‌های کاربردی وب مقابله می‌کنند. در این بخش، روش پیشنهادی و نتایج بررسی آن مورد بحث قرار خواهد گرفت و با سایر مطالعات پیشنهادشده برای مسدود کردن حملات تزریق SQL مقایسه شده است. مقایسه بر اساس معیارهای زیر است:

- مسدود کردن تمام نوع حملات
- استفاده از تحلیل ایستا

- اصلاح کد
- سطح مشخصات توسعه‌دهنده
- تولید مثبت کاذب و منفی کاذب
- منطق پایه زمان اجرا

علاوه بر معیارهای مقایسه، روش پیشنهادی از روش‌های موجود متفاوت است، زیرا با استفاده از الگوریتم بررسی‌کننده ورودی و طبقه‌بندی درخت تصمیم بر روی پایگاه داده حملات تزریق SQL، حملات را فیلتر کند و همچنین می‌تواند حملات جدید را توسط تراکنش‌های پایگاه داده، با استفاده از ناظر پایگاه داده مسدود کند. مقایسه در دو جدول تقسیم خواهد شد، زیرا اطلاعات مربوط به معیارهای مقایسه در برخی مطالعات موجود نیست. جدول (۲) نتیجه مقایسه برخی از معیارهای ذکرشده را نشان می‌دهد.

جدول (۲): مقایسه رویکردهای موجود با روش پیشنهادی (۱)

مسیریابی حملات	مسدود کردن حملات موجود	مشخصات حملات	استفاده از تحلیل ایستا	رویکردها
خیر	همه حملات به جزء رویه‌های ذخیره‌شده	خودکار	کاملاً	Halfond, Orso [4]
خیر	همه	خودکار	کاملاً	Wassermann , Su [44]
خیر	همه	فیلتر دستی	خیر	Shrivastava, Bhattacharyji [45]
خیر	برخی	خودکار	بله	Natarajan, Subramani [46]
خیر	همه	خودکار	کاملاً	Manikanta, Sardana [27]
خیر	همه	خودکار	کاملاً	Inyong Lee [28]
بله	همه	خودکار دستی	تا حدی	Proposed Method

جدول (۲) برخی رویکردهای موجود و اطلاعات مقایسه‌ای بر اساس استفاده از تحلیل ایستا، مشخصات حملات، مسدود کردن حملات موجود و مسیریابی حملات را نشان می‌دهد. برخی از رویکردهای موجود، کد را تجزیه و تحلیل و شبیه‌سازی می‌کنند، تا محتویات آسیب‌پذیر را پیدا کنند، برخی دیگر به مرحله تجزیه و تحلیل ایستا نیاز ندارند، زیرا بر اساس فیلتر کردن ورودی‌ها هستند. روش پیشنهادی فرض می‌کند، که تجزیه و تحلیل ایستا برای تعیین نقاط حساس از برنامه کاربردی استفاده می‌شود. مشخصات حملات روش پیشنهادی به‌صورت خودکار- دستی انجام خواهد شد. اطلاعات مقایسه دوم در جدول (۳) نشان داده شده است.

جدول(۳): مقایسه رویکردهای موجود با روش پیشنهادی (۲)

ناظر پایگاه داده	نظارت زمان اجرا	منفی کاذب	مثبت کاذب	اصلاح کد	رویکردها
خیر	خیر	خیر	خیر	بله	Boyd, Keromytis [47]
خیر	بله، جاوا بر اساس NDFA	خیر	کم	خیر	Halfond, Orso [4]
خیر	خیر	خیر	کم	خیر	Wassermann, Su[44]
خیر	خیر	N/A	N/A	خیر	Shrivastava, Bhattacharyji [45]
خیر	بله، نظارت جاوا	بله	N/A	خیر	Natarajan, Subramani [46]
خیر	بله، با استفاده از فایروال DB	خیر	خیر	خیر	Manikanta, Sardana [27]
بله	بله، نظارت PHP	خیر	کم	بله	Proposed Method

جدول (۳) یک مقایسه دیگر، که بر اساس معیارهای اصلاح کد، مثبت کاذب، منفی کاذب، نظارت زمان اجرا و ناظر پایگاه داده است، را نشان می‌دهد. برخی از رویکردهای موجود، کد برنامه را تغییر داده تا رویکرد خود را به‌کار گیرند، مانند [۴۷]، زیرا آن‌ها نیازمند یک نرم‌افزار مجتمع هستند که می‌توانند مقداردهی اولیه کنند و عدد تصادفی هر کلمه کلیدی SQL را به خاطر آورد. روش پیشنهادی، اصلاح کد کمی نیاز دارد، زیرا نقاط تأکید برای هر نقطه حساس از نظارت زمان اجرا، به یک کد برنامه کاربردی وب اضافه شده است. خطرناک‌ترین نوع از نتیجه بررسی، مثبت کاذب و منفی کاذب است. در روش پیشنهادی مثبت کاذب‌ها محدود هستند، چنانچه در ارزیابی دقت روش پیشنهادی در بخش بعدی بحث شده است.

۵-۱- ارزیابی دقت روش پیشنهادی

برای ارزیابی دقت روش پیشنهادی، چند پایگاه داده آزمایشی را در نظر گرفته و از روش‌های پیشین، روش AMNESIA[4] و Inyong Lee[28] انتخاب شد. الگوریتم‌های به‌کاررفته در این دو روش به‌همراه روش پیشنهادی بر روی پرس‌وجوهای ایجاد شده اجرا شد. در جدول (۴)، نتیجه یک نمونه پایگاه داده آزمایشی دیده می‌شود، که ارزیابی آن در سه مرحله شامل ۵۰، ۷۵۰ و

۱۵۰۰ پرس‌وجو انجام شد. در مرحله اول ۵۰ پرس‌وجو در نظر گرفته شده و در مرحله دوم ۷۵۰ پرس‌وجو که ۴۰۰ تای آن از نوع حمله تزریق می‌باشد و در این مرحله به‌طور متوسط از هر نوع حمله ۵۷ مورد تعریف شده است و ۳۵۰ تای آن پرس‌وجوی نرمال است. فرض بر این است که هر چه میزان استفاده از پایگاه داده و حجم درخواست‌ها از طرف کاربران بالاتر می‌رود، به تعداد پرس‌وجوهای پویای خارج از کد منبع افزوده می‌شود. تعدادی از پرس‌وجوها نیز جزء رویه‌های ذخیره شده هستند.

همان‌طور که ملاحظه می‌شود، با افزایش تعداد پرس‌وجوها، به علت افزایش روال ذخیره‌شده و پرس‌وجوهای پویای خارج از کد منبع، مقدار مثبت کاذب افزایش می‌یابد، که در روش‌های AMNESIA و Inyong Lee این مقدار، افزایش زیادی یافته است. در روش پیشنهادی به علت وجود جزء ناظر پایگاه داده و نیز به‌کارگیری طبقه‌بندی درخت تصمیم، مقدار مثبت کاذب بسیار کمتر است. روش‌های AMNESIA و Inyong Lee فقط قادر به شناسایی پرس‌وجوهای منطبق بر کد منبع هستند و پرس‌وجوهای پویایی را که توسط کاربر تولید شده و در کد منبع وجود ندارد، را شناسایی نمی‌کنند.

با توجه به جدول (۴)، تعداد مثبت کاذب در روش پیشنهادی و نیز روش‌های AMNESIA و Inyong Lee را با افزایش تعداد پرس‌وجوها نشان می‌دهد، بر اساس این نمودار، همان‌طور که در شکل دیده می‌شود، با افزایش تعداد پرس‌وجوها، تعداد مثبت کاذب در دو روش AMNESIA و Inyong Lee افزایش می‌یابد.

روش مربوط به Inyong Lee از روش AMNESIA بهتر عمل می‌کند، زیرا رویه‌های ذخیره‌شده را پوشش می‌دهد. روش پیشنهادی بهتر از دو روش دیگر عمل می‌کند، زیرا وابسته به کد منبع نیست و همه حملات را شناسایی می‌کند و پرس‌وجوهای پویا را پوشش می‌دهد.

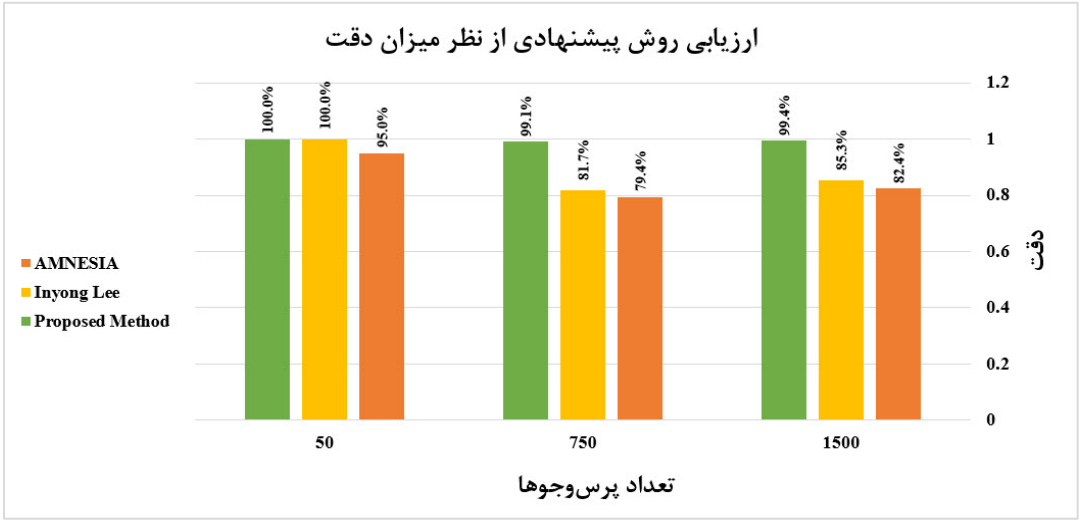
با توجه به جدول (۴)، ارزیابی میزان دقت با استفاده از رابطه (۱) صورت می‌گیرد که در آن TP مثبت صحیح و FP مثبت کاذب می‌باشد:

$$\text{دقت} = \frac{TP}{TP+FP} \qquad (۱)$$

در نمودار شکل (۴) ارزیابی دقت روش پیشنهادی در مقایسه با دو روش AMNESIA و Inyong Lee نمایش داده شده است. میانگین دقت روش پیشنهادی در مقایسه با میانگین دقت روش‌های Inyong Lee و AMNESIA به‌ترتیب ۱۲٪ و ۱۶٪ بهبود یافته است. همان‌طور که ملاحظه می‌شود در روش‌های پیشین جزء ناظر پایگاه داده و نیز پایگاه داده تزریق SQL و طبقه‌بندی درخت تصمیم برای تشخیص و پیشگیری از حملات تزریق SQL در نظر گرفته نشده است، همچنین هیچ کدام از آن‌ها از پرس‌وجوهای پویای خارج از کد منبع که توسط کاربر تولید می‌شود، پشتیبانی نمی‌کنند. دقت روش پیشنهادی بالاتر از دو روش دیگر می‌باشد، زیرا وابسته به کد منبع نیست و همه حملات را شناسایی می‌کند و پرس‌وجوهای پویا را پوشش می‌دهد و به‌دلیل جزء ناظر پایگاه داده قادر است که حملات جدید را نیز شناسایی کند.

جدول (۴): ارزیابی روش پیشنهادی از نظر میزان مثبت کاذب

روش	کل پرس‌وجو	پرس‌وجوی نرمال	روال ذخیره‌شده نرمال داخل کد منبع	پرس‌وجوی نرمال خارج از کد منبع	پرس‌وجوی نرمال شناسایی‌شده	مثبت کاذب
Proposed Method	۵۰	۲۰	۱	۰	۲۰	۰
	۷۵۰	۳۵۰	۲۱	۶۵	۳۴۷	۳
	۱۵۰۰	۸۰۰	۵۳	۱۲۴	۷۹۵	۵
AMNESIA	۵۰	۲۰	۱	۰	۱۹	۱
	۷۵۰	۳۵۰	۲۱	۶۵	۲۷۸	۷۲
	۱۵۰۰	۸۰۰	۵۳	۱۲۴	۶۵۹	۱۴۱
Inyong Lee	۵۰	۲۰	۱	۰	۲۰	۰
	۷۵۰	۳۵۰	۲۱	۶۵	۲۸۶	۶۴
	۱۵۰۰	۸۰۰	۵۳	۱۲۴	۶۸۲	۱۱۸



شکل (۴): ارزیابی دقت روش پیشنهادی

۶- نتیجه‌گیری

تزریق SQL، یکی از جدی‌ترین تهدیدها برای برنامه‌های کاربردی وب در فضای سایبری محسوب می‌شود. در این پژوهش یک روش برای تشخیص و پیشگیری از حملات تزریق SQL پیشنهاد گردید. روش پیشنهادی، با نظارت زمان اجرا و به‌کارگیری طبقه‌بندی درخت تصمیم بر روی پایگاه داده تزریق SQL، حملات تزریق SQL موجود را مسدود می‌کند و همچنین با استفاده از ناظر پایگاه داده حملات جدید را تشخیص می‌دهد. روش پیشنهادی، با دیگر روش‌های تشخیص و پیشگیری از حملات تزریق SQL موجود، مقایسه گردید، نتایج به‌دست‌آمده نشان داد، که روش پیشنهادی، به‌طور قابل توجهی در تشخیص و پیشگیری از حملات تزریق SQL موفق است و نسبت به روش‌های دیگر از نتایج مثبت کاذب کمتری و نیز دقت بالایی برخوردار است، زیرا روش پیشنهادی از پرس‌وجوهای پویای خارج از کد منبع که توسط کاربر تولید می‌شود، پشتیبانی می‌کند و نیز در روش‌های پیشین، ترکیب جزء ناظر پایگاه داده و نیز پایگاه داده تزریق SQL و طبقه‌بندی درخت تصمیم برای تشخیص و پیشگیری از حملات تزریق SQL در نظر گرفته نشده است.

۷- منابع

[1] W. G. Halfond and A. Orso, “AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks,” in Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, 2005, pp. 174-183.

[2] S. Kumar, R. Mahajan, N. Kumar, and S. K. Khatri, “A study on web application security and detecting security vulnerabilities,” in 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO), 2017, pp. 451-455.

[3] OWASP Top 10 Application Security Risks – 2017. Available: https://www.owasp.org/index.php/Top_10_2017-Top_10

[4] W. G. Halfond and A. Orso, “Preventing SQL injection attacks using AMNESIA,” in Proceedings of the 28th international conference on Software engineering, pp. 795-798, 2006.

[5] K. Beaver, Hacking for dummies: John Wiley & Sons, 2007.

[6] M. T. Simpson, K. Backman, and J. Corley, Hands-on ethical hacking and network defense: Cengage Learning, 2010.

[7] W. G. Halfond, J. Viegas, and A. Orso, “A classification of SQL-injection attacks and countermeasures,” in Proceedings of the IEEE International Symposium on Secure Software Engineering, pp. 13-15, 2006.

[8] E. R. Indrani Balasundaram, “An approach to detect and prevent SQL injection attacks in database using web service,” IJCSNS International Journal of Computer Science and Network Security, vol. 11, pp. 95-100, 2011.

[9] R. Yeole, S. Ninawe, P. Dhore, and P. Tembhare, “A Study on Detection and Prevention of SQL Injection Attack,” 2017.

[10] U. Agarwal, M. Saxena, and K. S. Rana, “A Survey of SQL Injection Attacks,” International Journal of Advanced Research in Computer Science and Software Engineering, vol. 5, pp. 286-289, 2015.

[11] A. Tajpour, M. Z. Heydari, M. Masrom, and S. Ibrahim, “SQL injection detection and prevention tools assessment,” in Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on, pp. 518-522, 2010.

2008 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation, pp. 3-12, 2008.

[26] V. B. Livshits and M. S. Lam, "Finding Security Vulnerabilities in Java Applications with Static Analysis," in USENIX Security Symposium, pp. 18-18, 2005.

[27] Y. V. N. Manikanta and A. Sardana, "Protecting web applications from SQL injection attacks by using framework and database firewall," in Proceedings of the International Conference on Advances in Computing, Communications and Informatics, pp. 609-613, 2012.

[28] I. Lee, S. Jeong, S. Yeo, and J. Moon, "A novel method for SQL injection attack detection based on removing SQL query attribute values," Mathematical and Computer Modelling, vol. 55, pp. 58-68, 2012.

[29] B. Tajalipour and A. Safaie, "Structural and semantic analysis of query to detect SQL injection attacks," Journal of Electronical & Cyber Defence, vol. 2, pp. 83-97, 2014. (In Persian)

[30] M. Alkhalaf, A. Aydin, and T. Bultan, "Semantic differential repair for input validation and sanitization," in Proceedings of the 2014 International Symposium on Software Testing and Analysis, pp. 225-236, 2014.

[31] K. Frajták, M. Bureš, and I. Jelínek, "Reducing user input validation code in web applications using Pex extension," in Proceedings of the 15th International Conference on Computer Systems and Technologies, pp. 302-308, 2014.

[32] X. Li and Y. Xue, "A survey on server-side approaches to securing web applications," ACM Computing Surveys (CSUR), vol. 46, p. 54, 2014.

[33] S. Cho, G. Kim, S.-j. Cho, J. Choi, M. Park, and S. Han, "Runtime input validation for Java web applications using static bytecode instrumentation," in Proceedings of the International Conference on Research in Adaptive and Convergent Systems, pp. 148-152, 2016.

[34] I. Medeiros, N. F. Neves, and M. Correia, "Automatic detection and correction of web application vulnerabilities using data mining to predict false positives," in Proceedings of the 23rd international conference on World wide web, pp. 63-74, 2014.

[35] S. O. Uwagbole, W. J. Buchanan, and L. Fan, "Applied machine learning predictive analytics to SQL injection attack detection and prevention," in 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 1087-1090, 2017.

[36] J. Zhao, J. Qi, L. Zhou, and B. Cui, "Dynamic taint tracking of web application based on static code analysis," in 2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), pp. 96-101, 2016.

[12] C. Gould, Z. Su and P. Devanbu, "JDBC checker: A static analysis tool for SQL/JDBC applications," in Proceedings of the 26th International Conference on Software Engineering, pp. 697-698, 2004.

[13] S. Madan and S. Madan, "Shielding against sql injection attacks using admire model," in Computational Intelligence, Communication Systems and Networks, 2009. CICSYN'09. First International Conference on, pp. 314-320, 2009,

[14] B. Indrani and E. Ramaraj, "X–LOG Authentication Techniqe to Prevent SQL Injection Attacks," International Journal of Information Technology and Knowledge Management, vol. 4, pp. 323-328, 2011.

[15] G. Buehrer, B. W. Weide, and P. A. Sivilotti, "Using parse tree validation to prevent SQL injection attacks," in Proceedings of the 5th international workshop on Software engineering and middleware, pp. 106-113, 2005.

[16] Z. Su and G. Wassermann, "The essence of command injection attacks in web applications," in ACM SIGPLAN Notices, pp. 372-382, 2006.

[17] A. Sadeghian, M. Zamani, and A. A. Manaf, "SQL injection vulnerability general patch using header sanitization," in Computer, Communications and Control Technology (I4CT), 2014 International Conference on, pp. 239-242, 2014.

[18] A. Pramod, A. Ghosh, A. Mohan, M. Shrivastava, and R. Shettar, "SQLI detection system for a safer web application," in Advance Computing Conference (IACC), 2015 IEEE International, pp. 237-240, 2015.

[19] A. Makiou, Y. Begriche, and A. Serhrouchni, "Improving Web Application Firewalls to detect advanced SQL injection attacks," in Information Assurance and Security (IAS), 2014 10th International Conference on, pp. 35-40, 2014.

[20] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo, "Securing web application code by static analysis and runtime protection," in Proceedings of the 13th international conference on World Wide Web, pp. 40-52, 2004.

[21] A. S. Christensen, A. Møller, and M. I. Schwartzbach, "Precise analysis of string expressions," in International Static Analysis Symposium, pp. 1-18, 2003.

[22] D. Sharma, K. Kale, C. Date, and D. Bhave, "Using AMNESIA to secure web applications and database against SQL injection attack," 2017.

[23] K. Kemalis and T. Tzouramanis, "SQL-IDS: a specification-based approach for SQL-injection detection," in Proceedings of the 2008 ACM symposium on Applied computing, pp. 2153-2158, 2008.

[24] V. Kodaganallur, "Incorporating language processing into java applications: A javacc tutorial," IEEE software, vol. 21, pp. 70-77, 2004.

[25] M. S. Lam, M. Martin, B. Livshits, and J. Whaley, "Securing web applications with static and dynamic information flow tracking," in Proceedings of the

[37] Y. Fang, J. Peng, L. Liu, and C. Huang, "WOVSQLI: Detection of SQL injection behaviors using word vector and LSTM," in Proceedings of the 2nd International Conference on Cryptography, Security and Privacy, pp. 170-174, 2018.

[38] C. Arumugam, V. B. Dwarakanathan, S. Gnanamary, V. N. Neyveli, R. K. Ramesh, Y. Kandhavel, et al., "Prediction of SQL Injection Attacks in Web Applications," in International Conference on Computational Science and Its Applications, pp. 496-505, 2019.

[39] D. Mitropoulos, P. Louridas, M. Polychronakis, and A. D. Keromytis, "Defending against web application attacks: approaches, challenges and implications," IEEE Transactions on Dependable and Secure Computing, vol. 16, pp. 188-203, 2017.

[40] P. Tang, W. Qiu, Z. Huang, H. Lian, and G. Liu, "SQL Injection Behavior Mining Based Deep Learning," in International Conference on Advanced Data Mining and Applications, pp. 445-454, 2018.

[41] M. Torabi and A. Shahidinejad, "A Classification of SQL Injection Attacks and Techniques to Defend These Attacks in the Passive Defense," Passive Defense Quarterly, vol. 9, pp. 101-117, 2018. (In Persian)

[42] M. Talaiezavareh and A. Shahidinejad, "Identification of SQL Injection Attacks using Support Vector Machine and Token Graph," in National Conference on Modern Research in Computer Engineering, Electrical, Information Technology, 2018. (In Persian)

[43] F. Kiasat, M. Fathi, and H. Golbaghi, "Detection of SQL Injection Attacks using Fuzzy Clustering, " in Third Conference on Security of information exchange space Accidents and Vulnerabilities, 2017. (In Persian)

[44] G. Wassermann and Z. Su, "Sound and precise analysis of web applications for injection vulnerabilities," in ACM Sigplan Notices, pp. 32-41, 2007.

[45] R. Shrivastava, J. Bhattacharyji, and R. Soni, "SQL injection attacks in database using web service: detection and prevention–review," Asian Journal of Computer Science & Information Technology, vol. 2, 2013.

[46] K. Natarajan and S. Subramani, "Generation of SQL-injection free secure algorithm to detect and prevent SQL-injection attacks," Procedia Technology, vol. 4, pp. 790-796, 2012.

[47] S. Boyd and A. Keromytis, "SQLrand: Preventing SQL injection attacks," in Applied Cryptography and Network Security, pp. 292-302, 2004.

Journal of Electronical & Cyber Defence
Vol. 8, No. 4, 2021, Serial No. 32

Detection and Prevention of SQL Injection Attacks at Runtime Using Decision Tree Classification

A. Shahidinejad*, M. Torabi

*Department of Computer, Faculty of Engineering, Islamic Azad University, Qom branch, Qom, Iran

(Received: 20/10/2019, Accepted: 01/02/2020)

ABSTRACT

The use of web applications has become increasingly popular in our routine activities, such as reading the news, paying bills, and shopping on-line. As the availability of these services grows, we are witnessing an increase in the number and sophistication of attacks that target web applications. SQL injection attacks are a serious security threat to web applications in the cyberspace. SQL injection attacks allow attackers to gain unlimited access to a database that includes applications and potentially sensitive information. Although researchers and practitioners have proposed different methods to solve the SQL injection problem, current approaches either fail to solve the full scope of the problem or have limitations that prevent their use and adoption. This study is designed to provide a method for detecting and preventing SQL injection attacks at runtime, which can detect and continuously monitor the existing and new attacks. The proposed detection and prevention method by runtime monitoring and implementation of the decision tree classification on the SQL injection database, blocks existing SQL injection attacks and also detects new attacks using the database supervisor. In the end, the proposed method is compared with other methods for detecting and preventing existing SQL injection attacks, the results showing that the proposed method is significantly successful in detecting and preventing SQL injection attacks. Compared to the two methods explored in this article, the presented method increases the accuracy by 12% for one method and 16% for the other.

Keywords: Web Applications, Database Security, SQL Injection Attacks, Detection, Prevention

* Corresponding Author Email: a.shahidinejad@qom-iau.ac.ir