دانشکده مهندسی

گروه کامپیوتر

کارشناسی نرم افزار

طراحی و ساخت ابزار پویش آسیب‌پذیری‌های SQL Injection و XSS

استاد راهنما:

جناب آقای دکتر اصغر تاج‌الدین

دانشجویان:

آیدا علیمحمدی و نگار رضائی

تابستان ۱۴۰۳

تقدیر و تشکر

مراتب قدردانی و سپاس خود را از استاد راهنمای محترم جناب آقای دکتر اصغر تاج‌الدین که با رهنمودها و حمایت‌های خود در تمامی مراحل این پروژه همراهی کردند، به استحظار می‌رسانیم. دانش و حضور ایشان در هر قدم از کار الهام‌بخش و راهگشای این مسیر بود و بدون کمک‌های ایشان، این موفقیت میسر نمی‌شد. از خداوند برای ایشان سلامتی و موفقیت روزافزون خواستاریم.

چکیده

در دنیای دیجیتال امروزی، امنیت سامانه‌های تحت‌وب به یکی از مسائل حیاتی برای سازمان‌ها و کسب‌وکارها تبدیل شده‌است. آسیب‌پذیری‌هایی نظیر SQL Injection و XSS [1]می‌توانند به سوءاستفاده‌های امنیتی و نفوذ به داده‌های حساس منجر شوند. پروژه VulneraXSQL به منظور ارائه یک راهکار جامع برای شناسایی و مدیریت این آسیب‌پذیری‌ها طراحی شده‌است. این ابزار با استفاده از روش‌های پیشرفته و الگوریتم‌های مدرن، آسیب‌پذیری‌های سامانه‌های تحت‌وب را شناسایی کرده و گزارش‌های عملیاتی در مورد نقاط ضعف امنیتی ارائه می‌دهد. ابزار VulneraXSQL شامل طراحی و پیاده‌سازی API[2] های کارآمد، ایجاد رابط‌کاربری [3]ساده و کاربرپسند با استفاده از React ، و طراحی پایگاه داده[4] با SQLite است. این ابزار به سازمان‌ها و توسعه‌دهندگان کمک می‌کند تا با تحلیل و رفع آسیب‌پذیری‌های شناسایی شده، امنیت سامانه‌های تحت‌وب خود را تقویت کنند و از حملات احتمالی جلوگیری نمایند. همچنین، VulneraXSQL به عنوان یک وسیله آموزشی برای آشنایی با آسیب‌پذیری‌های امنیتی و روش‌های مقابله با آن‌ها عمل می‌کند. این پروژه زیرساختی است که امکان افزودن سایر حملات را نیز دارد. از چالش‌های اصلی پروژه می‌توان به شناسایی سامانه‌های تحت‌وب مخرب، مدیریت حجم داده‌ها و اطمینان از دقت نتایج اشاره کرد. هدف نهایی این پروژه، ارتقاء سطح امنیت سامانه‌های تحت‌وب و تسهیل فرآیند شناسایی آسیب‌پذیری‌ها به گونه‌ای مؤثر و سریع است.

کلیدواژه‌ها: آسیب‌پذیری سامانه‌تحت‌وب، SQL Injection، XSS، امنیت وب، ابزار شناسایی آسیب‌پذیری، SQLite

[1] Cross-Site Scripting
[2] Application Programming Interface
[3] Frontend
[4] Database

فهرست مطالب

فهرست اشکال

فهرست جداول

فصل اول: مقدمه

۱-۱ مقدمه

در عصر دیجیتال امروزی، سامانه‌های تحت‌وب به عنوان یکی از اصلی‌ترین ابزارهای ارتباطی و تجاری برای کسب‌وکارها و سازمان‌ها شناخته می‌شوند. از طریق این سامانه‌های تحت‌وب، شرکت‌ها می‌توانند محصولات و خدمات خود را به گستره وسیعی از مخاطبان ارائه دهند، اطلاعات حساس را مدیریت کنند و تعاملات روزانه خود را بهینه سازند. با این حال، با افزایش استفاده از سامانه‌های تحت‌وب، همزمان تهدیدات امنیتی نیز به طور قابل توجهی رشد یافته‌اند.

این تهدیدات می‌توانند از نقص‌های امنیتی ساده‌ای که در طراحی یا پیاده‌سازی سامانه‌های تحت‌وب وجود دارند، سوءاستفاده کنند و منجر به دسترسی غیرمجاز به داده‌های حساس، تغییر و تخریب اطلاعات یا حتی از بین بردن داده‌ها شوند. برای مثال، حملات SQL Injection و XSS از جمله رایج‌ترین و خطرناک‌ترین حملات سایبری هستند که هکرها از آن‌ها برای نفوذ به سیستم‌های وبی و دسترسی به اطلاعات محرمانه استفاده می‌کنند.

با توجه به این تهدیدات، سازمان‌ها و کسب‌وکارها نیازمند ابزارهایی هستند که بتوانند به طور مداوم و خودکار آسیب‌پذیری‌های سامانه‌های تحت‌وبیشان را شناسایی کرده و اقدامات لازم برای رفع آن‌ها را انجام دهند. ابزارهای امنیتی مانند VulneraXSQL نه تنها به شناسایی این آسیب‌پذیری‌ها کمک می‌کنند، بلکه با ارائه گزارش‌های دقیق و راهکارهای عملی، به تیم‌های امنیتی و توسعه‌دهندگان امکان می‌دهند تا به سرعت وارد عمل شده و مشکلات را قبل از تبدیل شدن به یک بحران جدی حل کنند.

علاوه بر این، چنین ابزارهایی می‌توانند به عنوان بخشی از فرآیند توسعه نرم‌افزار و طراحی وب، از همان مراحل اولیه امنیت سایبری سامانه‌ها را بهبود دهند. با استفاده از VulneraXSQL ، توسعه‌دهندگان می‌توانند برخی از آسیب‌پذیری‌ها را شناسایی نموده و راهکاری برای حل آن‌ها اتخاذ کنند.

هدف اصلی پروژه VulneraXSQL، توسعه یک ابزار جامع و کارآمد برای شناسایی و مدیریت آسیب‌پذیری‌های امنیتی سامانه‌های تحت‌وب است. این ابزار با ارائه راه‌حلی آسان و موثر، به کاربران امکان می‌دهد تا به سرعت نقاط ضعف سیستم‌های خود را شناسایی و رفع کنند، به گونه‌ای که از نفوذهای احتمالی و حملات مخرب جلوگیری شود. اهداف مشخص این پروژه عبارتند از:

- بهبود امنیت سامانه‌های تحت‌وب: فراهم آوردن ابزاری که به سازمان‌ها و کاربران امکان می‌دهد تا به طور مستمر وضعیت امنیتی سامانه‌های تحت‌وبی خود را پایش و ارزیابی کنند و در صورت شناسایی آسیب‌پذیری‌ها، اقدامات اصلاحی لازم را انجام دهند.

- سادگی و کاربرپسندی: طراحی واسط کاربری ساده و کاربرپسند برای VulneraXSQL به گونه‌ای که کاربران با هر سطح دانش فنی بتوانند به راحتی از امکانات ابزار بهره‌مند شوند.

- کارایی و سرعت بالا: بهینه‌سازی فرآیند پویش و تحلیل آسیب‌پذیری‌ها به منظور کاهش زمان لازم برای شناسایی نقاط ضعف و افزایش کارایی ابزار در محیط‌های با بار کاری بالا.

- افزایش آگاهی امنیتی: از طریق ارائه گزارش‌های دقیق و جامع، به کاربران کمک می‌کند تا دانش و آگاهی خود را در زمینه امنیت وب افزایش دهند و به این ترتیب بتوانند تصمیمات آگاهانه‌تری در خصوص مدیریت امنیت سیستم‌های خود اتخاذ کنند.

۱-۲ بیان مسئله

با توسعه سریع فناوری‌های تحت‌وب و افزایش تعاملات آنلاین، سازمان‌ها و کسب‌وکارها بیشتر از همیشه به سامانه‌های وب‌محور متکی هستند. این رشد سریع در فضای آنلاین همزمان با افزایش پیچیدگی و تعداد حملات سایبری بوده است. یکی از مهم‌ترین مشکلاتی که در این فضا بروز می‌کند، آسیب‌پذیری‌های امنیتی همچون

SQL Injection و XSS است که به مهاجمان امکان سوءاستفاده از داده‌های حساس و حتی نفوذ به ساختارهای داخلی سیستم‌ها را می‌دهد.

مشکل اصلی در این میان، عدم وجود ابزارهایی است که به طور مداوم و دقیق بتوانند این نوع آسیب‌پذیری‌ها را شناسایی و به توسعه‌دهندگان و مدیران سامانه‌ها گزارش‌های عملیاتی و قابل‌فهم ارائه دهند. بسیاری از ابزارهای موجود یا بسیار پیچیده‌اند یا فاقد دقت و سرعت کافی برای تحلیل محیط‌های پیچیده وب هستند. این امر باعث می‌شود که حتی شرکت‌های بزرگ نیز در برابر حملات سایبری آسیب‌پذیر باقی بمانند.

علاوه بر این، بسیاری از سازمان‌ها با چالش‌هایی نظیر حجم بالای داده‌ها، زمان‌بر بودن فرآیند شناسایی آسیب‌پذیری‌ها و نیاز به دانش فنی تخصصی برای کار با ابزارهای امنیتی مواجه هستند. عدم وجود راهکارهای کاربرپسند و قابل‌اتکا باعث می‌شود تا شناسایی و رفع آسیب‌پذیری‌ها به تعویق افتاده و احتمال وقوع حملات بیشتر شود. این وضعیت می‌تواند منجر به خسارات مالی، از دست رفتن اعتبار سازمان و حتی از بین رفتن اعتماد کاربران شود.

بنابراین، وجود یک ابزار جامع و کارآمد مانند VulneraXSQL که بتواند به صورت خودکار و با دقت بالا آسیب‌پذیری‌های امنیتی وب را شناسایی کند، به یک ضرورت تبدیل شده‌است. این ابزار می‌تواند به سازمان‌ها کمک کند تا به جای اتکا به فرآیندهای دستی و پیچیده، به سرعت نقاط ضعف خود را شناسایی کنند.

۱-۳ کاربردها

ابزار VulneraXSQL زیرساخت اولیه برای شناسایی آسیب‌پذیری‌ها فراهم می‌کند از جمله کاربردهای این ابزار عبارتند از:

- آزمون نفوذ (Penetration Testing): می‌تواند به عنوان یک ابزار موثر در فرآیند آزمون نفوذ مورد استفاده قرار گیرد تا نقاط ضعف امنیتی سیستم‌ها و سامانه‌های تحت‌وب شناسایی شوند.

- ارزیابی‌های امنیتی دوره‌ای: سازمان‌ها می‌توانند به صورت منظم از VulneraXSQL برای ارزیابی امنیتی سامانه‌های تحت‌وب و سیستم‌های خود استفاده کنند تا از به‌روز بودن و کارآمدی اقدامات امنیتی خود اطمینان حاصل کنند.

- پایش مستمر امنیت: با استفاده ازVulneraXSQL ، سازمان‌ها می‌توانند به صورت پیوسته وضعیت امنیتی سامانه‌های تحت‌وب را پایش کنند و در صورت شناسایی آسیب‌پذیری‌های جدید، به سرعت واکنش نشان دهند.

- ادغام با فرآیند توسعه نرم‌افزار: VulneraXSQL می‌تواند به عنوان بخشی از چرخه توسعه نرم‌افزار (SDLC) استفاده شود تا در مراحل اولیه توسعه، آسیب‌پذیری‌های احتمالی شناسایی و رفع شوند، به طوری که نرم‌افزارها قبل از انتشار نهایی، از نظر امنیتی بهینه شده باشند.

- آموزش و آگاهی‌بخشی: VulneraXSQL می‌تواند به عنوان ابزاری آموزشی برای توسعه‌دهندگان و مدیران امنیتی مورد استفاده قرار گیرد تا با انواع آسیب‌پذیری‌ها و روش‌های شناسایی آن‌ها آشنا شوند و دانش خود را در زمینه امنیت وب افزایش دهند.

۱-۴ چالش‌ها و پیچیدگی‌ها

توسعه و پیاده‌سازی ابزار VulneraXSQL با چالش‌ها و پیچیدگی‌های متعددی مواجه بوده است که برخی از آن‌ها به شرح زیر می‌باشند:

- شناسایی سامانه‌های تحت‌وبی مخرب: یکی از بزرگ‌ترین چالش‌ها در این پروژه، تشخیص و شناسایی سامانه‌های تحت‌وب مخرب و تحلیل دقیق آن‌ها برای انجام تست‌های امنیتی بهینه است. این فرآیند نیازمند الگوریتم‌های پیشرفته و به‌روز برای تشخیص تهدیدات جدید و پیچیده می‌باشد.

- مدیریت حجم بالا از بارهای‌آسیب‌پذیر[1]: آسیب‌پذیری‌های مختلف نیازمند ارسال بارهای‌آسیب‌پذیر متنوع و پیچیده برای شناسایی نقاط ضعف هستند. مدیریت و پردازش این حجم بالا از داده‌ها می‌تواند زمان‌بر و منابع‌بر باشد، که نیازمند بهینه‌سازی دقیق فرآیند پویش و تحلیل است.

- تضمین دقت و صحت نتایج: یکی دیگر از پیچیدگی‌های پروژه، اطمینان از دقت و صحت نتایج پویش است. ابزار باید قادر باشد تا به درستی آسیب‌پذیری‌ها را شناسایی کرده و گزارش‌های دقیقی ارائه دهد تا کاربران بتوانند اقدامات لازم را به طور مؤثر انجام دهند.

- امنیت خود ابزار: همان‌طور که VulneraXSQL برای شناسایی آسیب‌پذیری‌های دیگران طراحی شده‌است، خود ابزار نیز باید از نظر امنیتی مقاوم باشد تا توسط هکرها مورد سوء استفاده قرار نگیرد. این امر نیازمند پیاده‌سازی فرآیندهای امنیتی قوی و به‌روز است.

- مقیاس‌پذیری: ابزار باید بتواند با افزایش تعداد کاربران و حجم داده‌ها به خوبی مقیاس‌پذیر باشد. این مسئله مستلزم استفاده از معماری‌های انعطاف‌پذیر و فناوری‌های مناسب برای مدیریت بار کاری بالا است.

[1] Payload

فصل دوم: تاریخچه و تکامل ابزارهای پویش آسیب‌پذیری

۲-۱ مروری بر روند پویش آسیب‌پذیری

نخستین ابزارهای پویش آسیب‌پذیری به صورت دستی و با استفاده از اسکریپت‌های ساده‌ای که توسط متخصصان امنیت نوشته می‌شد، ایجاد شدند. این اسکریپت‌ها عمدتاً برای شناسایی آسیب‌پذیری‌های خاص مانند SQL Injection یا Cross-Site Scripting (XSS) طراحی شده بودند. در آن زمان، این اسکریپت‌ها کارآمد بودند، اما استفاده از آن‌ها نیاز به دانش فنی بالایی داشت و فرآیند پویش به صورت دستی انجام می‌شد که زمان‌بر و مستعد خطا بود.

با گذشت زمان و افزایش پیچیدگی حملات سایبری، نیاز به ابزارهای خودکار برای پویش و شناسایی آسیب‌پذیری‌ها افزایش یافت. این نیاز منجر به توسعه اولین نسل از ابزارهای پویش خودکار شد که توانستند به طور خودکار آسیب‌پذیری‌های متداول را در سامانه‌های تحت‌وب شناسایی کنند. این ابزارها به توسعه‌دهندگان و متخصصان امنیت کمک کردند تا پیش از آنکه حمله‌ای واقعی صورت گیرد، نقاط ضعف سیستم‌های خود را شناسایی و رفع کنند.

یکی از اولین ابزارهای برجسته در این زمینه، Whisker بود که در اواخر دهه ۱۹۹۰ معرفی شد. این ابزار به صورت خودکار به دنبال آسیب‌پذیری‌های متداول در سامانه‌های تحت‌وب می‌گشت و توانست به عنوان یکی از اولین ابزارهای پویش خودکار جای خود را باز کند. با وجود محدودیت‌هایی که در آن زمان وجود داشت، Whisker نقطه شروعی برای توسعه ابزارهای پیشرفته‌تر بود.

همچنین، در اوایل دهه ۲۰۰۰، ابزارهایی مانند Nessus معرفی شدند که توانستند با بهره‌گیری از یک پایگاه داده بزرگ از آسیب‌پذیری‌ها، به صورت خودکار و دقیق‌تر به شناسایی آسیب‌پذیری‌ها بپردازند. Nessus یکی از اولین ابزارهایی بود که به صورت گسترده در سازمان‌ها مورد استفاده قرار گرفت و به دلیل قابلیت‌های پیشرفته‌ای که داشت، به یکی از محبوب‌ترین ابزارهای پویش آسیب‌پذیری تبدیل شد.

۲-۲ دو نمونه از ابزارها

ابزارهای پویش آسیب‌پذیری طی سال‌ها پیشرفت کرده‌اند و از روش‌های ابتدایی به ابزارهای پیشرفته و چندمنظوره رسیده‌اند. در این بخش، به بررسی برخی از ابزارهای برجسته در این زمینه می‌پردازیم و به نحوه تکامل آن‌ها اشاره می‌کنیم.

۲-۲-۱ ابزار Havij:

ابزار Havij یک ابزار شناخته‌شده در زمینه پویش SQL Injection بود که در سال ۲۰۱۰ معرفی شد. این ابزار با هدف آسان‌سازی کشف آسیب‌پذیری‌های SQL Injection طراحی شد و به کاربران غیر فنی نیز اجازه می‌داد تا با چند کلیک ساده، این نوع آسیب‌پذیری‌ها را شناسایی کنند. هرچند Havij در ابتدا محبوبیت زیادی داشت، اما به مرور زمان به دلیل تک‌منظوره بودن و تمرکز فقط بر روی SQL Injection، جای خود را به ابزارهای جامع‌تری داد.

۲-۲-۲ ابزار Acunetix:

یکی از ابزارهای پیشرفته و چندمنظوره که به طور گسترده در پروژه ما نیز مورد استفاده قرار گرفت، Acunetix بود. این ابزار توانسته است در طول زمان به یکی از برترین پویش‌های امنیتی تبدیل شود و علاوه بر SQL Injection، آسیب‌پذیری‌های دیگری مانند XSS، RFI[1]، LFI [2]و موارد دیگر را شناسایی کند.

در پروژه، Acunetix به عنوان یک ابزار حیاتی برای شناسایی آسیب‌پذیری‌ها و تحلیل امنیتی استفاده شد. قابلیت‌های چندگانه این ابزار، مانند پویش دقیق و خودکار برای انواع آسیب‌پذیری‌های وب و ارائه گزارش‌های جامع، به ما در ارزیابی دقیق امنیت سامانه‌های تحت‌وبی هدف کمک شایانی کرد. Acunetix همچنین به دلیل امکان ارائه راه‌حل‌های عملی برای برطرف کردن آسیب‌پذیری‌ها، نقش مهمی در بهبود امنیت پروژه ما ایفا کرد.

[1] Remote File Inclusion
[2] Local File Inclusion

این ابزار با موتور پویش پیشرفته خود، سرعت و دقت بالایی را در شناسایی تهدیدات امنیتی ارائه می‌دهد و با پشتیبانی از استانداردهای مختلف و پروتکل‌های وب، به ما اجازه داد که به راحتی سیستم‌های مختلف را ارزیابی کنیم. توانایی Acunetix در یکپارچه‌سازی با سایر ابزارها و سیستم‌ها، نیز یکی دیگر از ویژگی‌های برجسته آن بود که باعث شد در توسعه پروژه ما نقش کلیدی داشته باشد.

فصل سوم: ضرورت امنیت و شناسایی آسیب‌پذیری‌ها در سامانه‌های تحت‌وب

۳-۱ مفهوم امنیت در فناوری اطلاعات

امنیت در حوزه فناوری اطلاعات شامل مجموعه‌ای از سیاست‌ها، پروتکل‌ها، اقدامات و فناوری‌هایی است که برای حفاظت از سیستم‌ها، شبکه‌ها و داده‌ها در برابر تهدیدات و حملات سایبری طراحی شده‌اند. این اقدامات شامل کنترل دسترسی، رمزنگاری داده‌ها، مدیریت رخنه‌ها، و پیاده‌سازی تدابیر دفاعی است که همگی با هدف حفظ محرمانگی[۱]، صحت[۲] و در دسترس بودن[۳] اطلاعات انجام می‌شوند. این سه اصل، که به اختصار به عنوان مدل CIA شناخته می‌شوند، اساس امنیت اطلاعات را تشکیل می‌دهند:

۳-۱-۱ محرمانگی:

اطمینان از اینکه فقط افراد مجاز به اطلاعات حساس دسترسی دارند. به عنوان مثال، اطلاعات کارت اعتباری مشتریان باید به گونه‌ای ذخیره و منتقل شود که تنها خود مشتریان و سیستم‌های پرداخت معتبر به آن دسترسی داشته باشند.

۳-۱-۲ صحت:

تضمین می‌کند که اطلاعات دقیق و قابل اعتماد هستند و در طول زمان تغییر یا تخریب نمی‌شوند. به عنوان مثال، سیستم‌های مالی باید به گونه‌ای طراحی شوند که از دستکاری یا تغییر غیرمجاز در تراکنش‌ها جلوگیری کنند.

۳-۱-۳ در دسترس بودن:

اطمینان از اینکه اطلاعات و سیستم‌ها در مواقع نیاز در دسترس کاربران مجاز قرار دارند. برای مثال، در یک سامانه بانکی آنلاین، سرویس‌ها باید به‌صورت ۲۴ ساعته و بدون وقفه در دسترس باشند تا مشتریان بتوانند تراکنش‌های خود را انجام دهند. اهمیت امنیت در دنیای امروز به وضوح در هر جنبه‌ای از زندگی دیجیتال ما قابل

[1] Confidentiality
[2] Integrity
[3] Availability

مشاهده است. به عنوان مثال، تصور کنید یک فروشگاه آنلاین با استفاده از سیستم‌های امنیتی ضعیف، اطلاعات کارت اعتباری مشتریان خود را به‌درستی محافظت نکند. در این شرایط، احتمال وقوع یک نفوذ سایبری بالا می‌رود که می‌تواند به سرقت اطلاعات حساس و ضرر مالی برای مشتریان و شرکت منجر شود. نمونه‌های واقعی از چنین حملاتی شامل نفوذهای بزرگ به شرکت‌های معتبری مثل Target و Equifax است که باعث از دست رفتن اطلاعات میلیون‌ها کاربر و خسارات گسترده مالی و اعتباری شد.

۳-۲ ابعاد امنیت سامانه‌های تحت‌وب:

در دنیای الکترونیکی امروز، امنیت نه تنها یک نیاز بلکه یک ضرورت است. با گسترش استفاده از اینترنت و فناوری‌های دیجیتال، تهدیدات امنیتی به طور قابل توجهی افزایش یافته‌اند. از دست رفتن امنیت می‌تواند عواقب فاجعه‌باری برای سازمان‌ها، کسب‌وکارها و کاربران داشته باشد. در زیر به برخی از دلایل اصلی اهمیت امنیت اشاره می‌شود:

۳-۲-۱ محافظت از داده‌های حساس:

اطلاعات شخصی کاربران، مانند شماره‌های ملی، اطلاعات مالی، و سوابق پزشکی، جزو دارایی‌های بسیار حساس به شمار می‌روند. بدون امنیت مناسب، این اطلاعات می‌توانند در معرض سرقت یا سوءاستفاده قرار گیرند. به عنوان مثال، یک بیمارستان که اطلاعات بیماران را به درستی محافظت نمی‌کند، ممکن است قربانی حملات سایبری شود که می‌تواند به افشای اطلاعات پزشکی محرمانه منجر شود.

۳-۲-۲ جلوگیری از حملات سایبری:

حملات سایبری می‌توانند طیف گسترده‌ای از آسیب‌ها را به سازمان‌ها وارد کنند، از سرقت داده‌ها گرفته تا ایجاد اختلال در خدمات حیاتی. به عنوان مثال، در سال ۲۰۱۷ حمله سایبری "NotPetya" سیستم‌های کامپیوتری بسیاری از شرکت‌ها و سازمان‌های بزرگ را در سراسر جهان تحت تأثیر قرار داد و باعث خسارات

میلیارد دلاری شد. این حمله نشان داد که حتی بزرگترین و پیشرفته‌ترین شرکت‌ها نیز در برابر تهدیدات سایبری آسیب‌پذیر هستند.

۳-۲-۳ حفظ اعتماد مشتریان و اعتبار برند:

مشتریان به سازمان‌ها اعتماد می‌کنند تا اطلاعاتشان را امن نگه دارند. اگر این اعتماد نقض شود، بازگشت به حالت عادی می‌تواند دشوار و پرهزینه باشد. به عنوان مثال، نفوذ به سیستم‌های شرکت Yahoo در سال ۲۰۱۳ و افشای اطلاعات بیش از ۳ میلیارد حساب کاربری منجر به افت شدید اعتماد کاربران به این شرکت شد و تأثیرات منفی طولانی‌مدتی بر اعتبار Yahoo گذاشت.

۳-۲-۴ مقابله با جرایم سایبری:

جرایم سایبری شامل فعالیت‌های غیرقانونی است که از کامپیوترها یا شبکه‌ها برای ارتکاب جرم استفاده می‌کنند. این جرایم شامل کلاهبرداری، سرقت هویت، و جاسوسی صنعتی هستند. با استفاده از راهکارهای امنیتی مناسب، سازمان‌ها می‌توانند از خود در برابر این نوع تهدیدات محافظت کنند و مانع از دست رفتن داده‌ها یا آسیب به اعتبار خود شوند.

۳-۲-۵ رعایت مقررات و قوانین:

بسیاری از صنایع و سازمان‌ها موظف به رعایت قوانین و مقرراتی هستند که برای محافظت از داده‌ها و اطلاعات شخصی وضع شده‌اند. به عنوان مثال، مقررات GDPR در اتحادیه اروپا الزاماتی سخت‌گیرانه برای حفاظت از داده‌های شخصی کاربران وضع کرده است. عدم رعایت این مقررات می‌تواند منجر به جریمه‌های سنگین و از دست رفتن اعتبار سازمان شود.

۳-۳ انواع آسیب‌پذیری‌ها:

آسیب‌پذیری‌ها نقاط ضعف یا نقص‌هایی در سیستم‌ها، نرم‌افزارها، شبکه‌ها، یا حتی فرآیندهای مدیریتی هستند که می‌توانند توسط مهاجمان سایبری مورد سوءاستفاده قرار گیرند. این نقاط ضعف ممکن است ناشی از دلایلی نظیر طراحی نادرست، پیکربندی اشتباه، یا حتی عدم به‌روزرسانی مناسب سیستم‌ها باشد. آسیب‌پذیری‌ها می‌توانند به روش‌های مختلفی آشکار شوند، از جمله کدهای آسیب‌پذیر، نقص در پروتکل‌های ارتباطی، یا حتی در نقص‌های فیزیکی در زیرساخت‌ها.

آسیب‌پذیری‌ها می‌توانند در دسته‌های مختلفی قرار گیرند، که در زیر به برخی از آن‌ها اشاره می‌شود:

۳-۳-۱ آسیب‌پذیری‌های نرم‌افزاری:

این نوع آسیب‌پذیری‌ها ناشی از خطاهای برنامه‌نویسی یا طراحی نادرست نرم‌افزارها هستند. به عنوان مثال، عدم اعتبارسنجی صحیح ورودی‌های کاربر می‌تواند منجر به حملات SQL Injection شود. در این حمله، مهاجم از طریق تزریق کدهای مخرب در ورودی‌های برنامه، می‌تواند به پایگاه داده دسترسی پیدا کرده و داده‌ها را استخراج یا تغییر دهد.

۳-۳-۲ آسیب‌پذیری‌های شبکه‌ای:

این آسیب‌پذیری‌ها در سطح شبکه رخ می‌دهند و ممکن است ناشی از پیکربندی نادرست دستگاه‌های شبکه‌ای، استفاده از پروتکل‌های ناامن، یا نقص‌های موجود در نرم‌افزارهای شبکه‌ای باشد. به عنوان مثال، یک شبکه بی‌سیم بدون رمزنگاری مناسب (مانندWPA2) ممکن است مورد حملات دسترسی غیرمجاز قرار گیرد و مهاجمان بتوانند ترافیک شبکه را شنود کنند.

۳-۳-۳ آسیب‌پذیری‌های فیزیکی:

این نوع آسیب‌پذیری‌ها مربوط به جنبه‌های فیزیکی سیستم‌ها و زیرساخت‌ها می‌شوند. به عنوان مثال، دسترسی فیزیکی غیرمجاز به سرورها می‌تواند منجر به سرقت یا تخریب داده‌ها شود. همچنین، عدم تأمین امنیت فیزیکی مراکز داده می‌تواند باعث بروز حوادثی نظیر آتش‌سوزی یا سرقت تجهیزات شود.

۳-۳-۴ آسیب‌پذیری‌های انسانی:

یکی از رایج‌ترین آسیب‌پذیری‌ها، خطاهای انسانی است. به عنوان مثال، یک کارمند ممکن است به‌طور تصادفی یک ایمیل فیشینگ را باز کند و دسترسی مهاجمان به سیستم‌های سازمان را فراهم کند. همچنین، استفاده از گذرواژه‌های ضعیف یا به اشتراک‌گذاری آن‌ها با دیگران می‌تواند به عنوان یک نقطه ضعف جدی در نظر گرفته شود.

۳-۴ مثال‌های واقعی از آسیب‌پذیری‌ها

برای درک بهتر مفهوم آسیب‌پذیری‌ها، چند مثال واقعی از حملات سایبری را بررسی می‌کنیم.

۳-۴-۱ حمله SQL Injection به LinkedIn (۲۰۱۲):

در سال ۲۰۱۲، یک آسیب‌پذیری SQL Injection در یکی از زیرسیستم‌های LinkedIn کشف شد که به مهاجمان امکان داد تا به داده‌های کاربران دسترسی پیدا کنند. این حمله منجر به افشای اطلاعات حساس میلیون‌ها کاربر شد و نشان داد که حتی پلتفرم‌های بزرگ و محبوب نیز ممکن است دچار چنین آسیب‌پذیری‌هایی باشند.

۳-۴-۲ حمله WannaCry (۲۰۱۷):

WannaCryیک حمله‌ی باج‌افزاری بود که از یک آسیب‌پذیری در سیستم‌عامل ویندوز به نام "EternalBlue" بهره‌برداری کرد. این حمله هزاران کامپیوتر در سراسر جهان را آلوده کرد و موجب شد که اطلاعات کاربران قفل

شود و تنها در صورت پرداخت باج به مهاجمان، دوباره به اطلاعات خود دسترسی پیدا کنند. این حمله نشان داد که چگونه عدم بروزرسانی سیستم‌ها می‌تواند به یک تهدید جهانی تبدیل شود.

فصل چهارم: بررسی آسیب پذیری‌های مرتبط با ابزار VulneraXSQL

۱-۴ آسیب پذیری SQL Injection

SQL Injection یکی از مهم‌ترین و خطرناک‌ترین آسیب‌پذیری‌های امنیتی است که زمانی رخ می‌دهد که یک مهاجم، کدهای مخرب SQL را به یک درخواست یا کوئری وارد می‌کند و باعث می‌شود که این کدها به طور نادرست در پایگاه داده اجرا شوند. این نوع حمله می‌تواند به مهاجمان اجازه دهد تا به اطلاعات حساس دسترسی پیدا کنند، داده‌ها را تغییر دهند، یا حتی پایگاه داده را به کلی خراب کنند. SQL Injection زمانی رخ می‌دهد که ورودی‌های کاربر به درستی اعتبارسنجی یا پاکسازی نشوند و همین امر باعث می‌شود که کدهای SQL ناخواسته اجرا شوند.

ابزار ما، VulneraXSQL، به گونه‌ای طراحی شده‌است که با دادن ورودی مجموعه‌ای ازبارهای‌آسیب‌پذی های مخصوص به قالب‌ها، نشانی‌های اینترنتی[1]، و دیگر ورودی‌ها، به تست آسیب‌پذیری SQL Injection می‌پردازد. این بارهای‌آسیب‌پذیر به‌گونه‌ای طراحی شده‌اند که نقاط ضعف در نحوه تعامل برنامه وب با پایگاه داده را آشکار کنند. اگر برنامه وب به درستی با ورودی‌ها برخورد نکند، این کدهای SQL تزریق شده ممکن است اجرا شوند و آسیب‌پذیری را آشکار کنند.

در این ابزار، ما به طور خلاصه به این موضوع پرداخته‌ایم که چگونه VulneraXSQL می‌تواند این نوع آسیب‌پذیری‌ها را شناسایی کند. جزئیات دقیق‌تر و روش‌های خاص استفاده شده در این فرآیند در فصل‌های بعدی به طور کامل توضیح داده خواهند شد. هدف اصلی این ابزار این است که فرآیند شناسایی این آسیب‌پذیری‌ها را به صورت خودکار انجام دهد و به سازمان‌ها کمک کند تا قبل از اینکه مهاجمان بتوانند از این نقاط ضعف سوءاستفاده کنند، آن‌ها را شناسایی و برطرف کنند.

[1] URL

۲-۴ آسیب پذیری XSS

یکی دیگر از آسیب‌پذیری‌های رایج و خطرناک، XSS است که در آن مهاجم کدهای مخرب جاوااسکریپت[1] را به یک سامانه‌های تحت‌وب تزریق می‌کند. این کدها به‌طور نادرست توسط مرورگر کاربران دیگر اجرا می‌شوند و به مهاجم اجازه می‌دهند تا داده‌های حساس کاربران را سرقت کند، محتوا[2]ی سامانه‌های تحت‌وب را تغییر دهد، یا حملات فیشینگ را اجرا کند. XSS زمانی رخ می‌دهد که یک سامانه‌های تحت‌وب داده‌های ورودی کاربر را بدون اعتبارسنجی یا پاک‌سازی مناسب به خروجی HTML یا جاوااسکریپت اضافه می‌کند. سه نوع اصلی XSS وجود دارد:

Stored XSS: در این نوع از حمله، کدهای مخرب در سرور ذخیره می‌شوند و با هر بار باز شدن صفحه، این کدها توسط مرورگر کاربران اجرا می‌شوند. این نوع حمله به دلیل باقی ماندن کدهای مخرب در سرور خطرناک‌تر است.

Reflected XSS: در این نوع حمله، کدهای مخرب در نشانی اینترنتی یا متغیرهای ورودی تزریق می‌شوند و بلافاصله در پاسخ سرور به کاربر ارسال می‌شوند. این نوع XSS به طریقی به نشانی اینترنتی وابسته است و مهاجم از کاربران می‌خواهد که روی لینک‌های مخرب کلیک کنند.

DOM-Based XSS: این نوع از حمله به طور مستقیم در سمت کاربر (مرورگر) رخ می‌دهد و بدون دخالت سرور اجرا می‌شود. مهاجم کدهای مخرب را به جاوااسکریپت صفحه تزریق می‌کند و این کدها در مرورگر اجرا می‌شوند.

[1] JavaScript
[2] context

ابزار ما، VulneraXSQL، برای شناسایی آسیب‌پذیری‌های XSS نیز از روش‌های مشابهی استفاده می‌کند. این ابزار مجموعه‌ای از بارهای‌آسیب‌پذیر خاص را به ورودی‌های مختلف سامانه‌های تحت‌وب تزریق می‌کند تا بررسی کند آیا این داده‌ها بدون پاک‌سازی مناسب به خروجی HTML یا جاوااسکریپت اضافه می‌شوند یا خیر. اگر برنامه وب به درستی این ورودی‌ها را کنترل نکند، کدهای مخرب جاوااسکریپت اجرا می‌شوند و نشان می‌دهند که سامانه‌های تحت‌وب در برابر XSS آسیب‌پذیر است.

هدف اصلی این ابزار، شناسایی خودکار این نوع آسیب‌پذیری‌ها و کمک به سازمان‌ها برای جلوگیری از حملات XSS است، پیش از اینکه مهاجمان بتوانند از آن‌ها سوءاستفاده کنند. در این فصل به صورت مختصر به نحوه عملکرد VulneraXSQL در شناسایی آسیب‌پذیری XSS پرداخته‌ایم. جزئیات بیشتری از روش‌های استفاده‌شده و نحوه عملکرد دقیق این ابزار در فصول بعدی توضیح داده خواهند شد.

فصل پنجم : ساختار و طرح کلی پروژه

۵-۱ مقدمه

پروژه‌ای که تحت عنوان VulneraXSQL پیاده‌سازی شده‌است، ابزاری جامع برای بررسی امنیت سامانه‌های تحت‌وب در برابر آسیب‌پذیری‌های مختلفی چون SQL Injection و XSS است. در دنیای امروز، امنیت سامانه‌های تحت و برنامه‌های آنلاین از اهمیت بالایی برخوردار است. حملات سایبری مانند XSS و SQL Injection می‌تواند به راحتی موجب نفوذ به سیستم‌های حساس و دسترسی غیرمجاز به اطلاعات شوند. این ابزار با بهره‌گیری از ترکیب Django به عنوان چارچوب[1] سمت سرور[2] و React به عنوان رابط‌کاربری، به کاربران امکان می‌دهد که به سادگی سامانه‌های تحت‌وب خود را برای یافتن این نوع آسیب‌پذیری‌ها آزمایش کنند. پایگاه داده SQLite به منظور ذخیره‌سازی نتایج پویش‌ها استفاده شده‌است. در این پروژه همچنین از JWT[3] برای احراز هویت کاربران و مدیریت نشانه‌های دسترسی[4] استفاده شده‌است. RabbitMQ به عنوان واسطه ارتباطی برای ارسال و دریافت پیام‌ها بین اجزای سیستم استفاده شده و Celery برای مدیریت وظایف[5] پس‌زمینه‌ای و غیرهمزمان مانند پویش کردن و توقف پویش استفاده می‌شود.

هدف این پروژه، ایجاد ابزاری ساده و کارآمد برای کمک به کاربران، از جمله سازمان‌ها و افراد است تا به راحتی بتوانند امنیت سامانه‌های تحت‌وب خود را بررسی و تقویت کنند. این ابزار به آن‌ها امکان می‌دهد که آسیب‌پذیری‌های موجود را شناسایی کرده و اقدامات مناسب برای محافظت از داده‌های حساس خود انجام دهند.

۵-۲ شرح کلی پروژه

این پروژه به گونه‌ای طراحی شده‌است که شامل بخش‌های مختلفی باشد تا عملکردی کارآمد و منعطف داشته باشد. ساختار کلی آن شامل موارد زیر است:

[1] Framework
[2] Backend
[3] JSON Web Token
[4] Access Token
[5] Tasks

۱. سمت سرور: از جنگو[1] به‌عنوان چارچوب اصلی استفاده شده‌است که به‌خوبی با نیازهای پروژه در زمینه امنیت و مدیریت داده‌ها سازگاری دارد. در قسمت بک اند از انواع تابع‌ها استفاده شده‌است تا پویش آسیب‌پذیری به خوبی انجام بگیرد. برای مدیریت APIها، از Django REST Framework بهره گرفته شده‌است. احراز هویت کاربران از طریق JWT[2] انجام می‌شود، که شامل نشانه دسترسی برای مدیریت جلسات کاربر است و همینطور از RabbitMQ و Celery در جهت مدیریت پویش‌های پیچیده و طولانی مدت استفاده شده‌است.

۲. سمت کلاینت: بخش کاربری و واسط گرافیکی این پروژه با استفاده از React پیاده‌سازی شده‌است. طراحی واسط کاربری به شکلی است که کاربران می‌توانند به سادگی وارد سیستم شوند، ثبت‌نام کنند و فرآیند پویش سامانه‌های تحت‌وب را انجام دهند.

۳. پایگاه داده: برای ذخیره‌سازی اطلاعات و نتایج پویش‌ها، از پایگاه داده سبک SQLite استفاده شده‌است که به‌خاطر سازگاری بالا با پروژه انتخاب شده‌است.

در این پروژه انجام پویش به دو حالت اتفاق می‌افتد که کاربر براساس هدف خود می‌تواند یکی از گزینه‌های زیر را استفاده کند و سامانه‌های تحت‌وب مدنظر خود را پویش کنند:

[1] Django
[2] JSON Web Tokens

- Get Demo: در این حالت، پویش با زمان کوتاه‌تر و استفاده از پیلودهای محدود انجام می‌شود. این حالت برای کاربرانی مناسب است که می‌خواهند به سرعت وضعیت امنیتی سامانه‌تحت‌وب خود را بررسی کنند. بعد از انجام پویش کاربر می‌تواند به صورت سند[1]، گزارش مربوط به پویش خود را دانلود کند، جزئیات پویش را مطالعه کند و اطلاعاتی را که لازم دارد استفاده کند. برخی از بارهای‌آسیب‌پذیر که در این بخش برای پیدا کردن آسیب‌پذیری استفاده می‌شود:

```
scan_results = []
payloads = [
    "' OR '1'='1",
    "' OR '1'='1' --",
    "' OR '1'='1' /*",
    "' OR ''='",
    "' OR 1=1--",
    "' OR 1=1#",
    "' OR 1=1/*",
    "admin' --",
    "' UNION SELECT NULL, NULL, NULL --",
    "' UNION SELECT 1, 'text', NULL --",
    "' AND '1'='1",
    "' AND '1'='1' --",
    "' AND 1=1",
    "' AND 1=1--",
    "' AND 1=1#",
    "' AND 1=1/*"
]
```

شکل ۱-۵: بارهای‌آسیب‌پذیر مربوط به SQL Injection

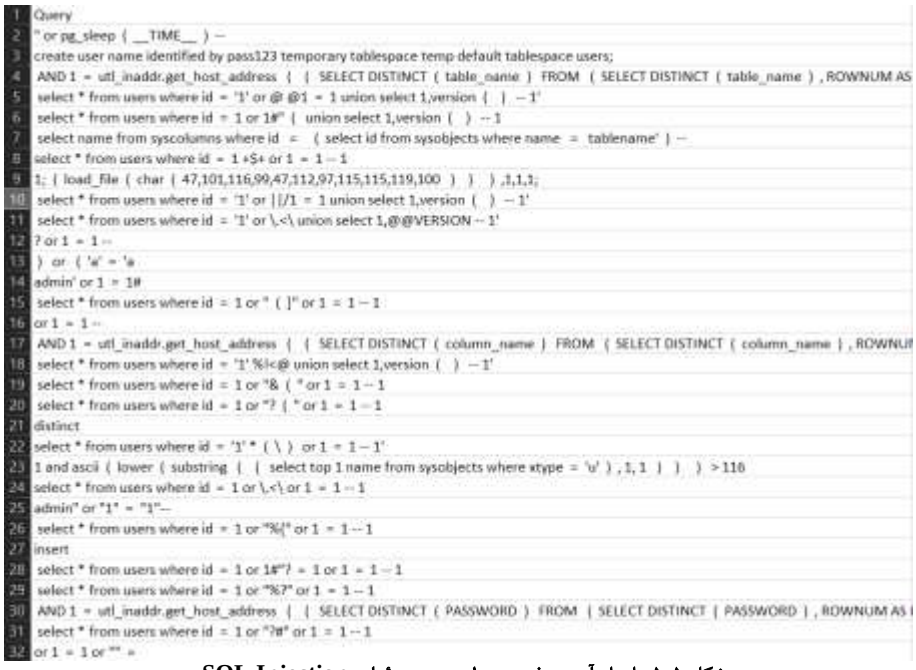
```
vulnerabilities = []
XSS_PAYLOADS = [
    '<script>alert(1)</script>',
    '"><script>alert(1)</script>',
    '"><img src=x onerror=alert(1)>',
    '<img src=x onerror=alert(1)>',
    '<svg/onload=alert(1)>',
    '<iframe src="javascript:alert(1)"></iframe>',
]
```

شکل ۲-۵: بارهای‌آسیب‌پذیر مربوط به XSS

[1] PDF

- Deep Scan: در این حالت، پویش به‌صورت دقیق‌تر و با استفاده از یک دیتاست گسترده‌تر برای هر کدام از آسیب‌پذیری‌ها انجام می‌شود. این گزینه برای کاربران حرفه‌ای و سازمان‌هایی که نیاز به تحلیل عمیق‌تری دارند مناسب است. منطقا با توجه به اینکه این بخش به دیتاست برای هر نوع آسیب‌پذیری متصل است، زمان پویش سامانه‌های تحت‌وب طولانی تر خواهد بود. برخی از بارهای‌آسیب‌پذیر که در دیتاست‌ها قرارداده شده‌اند به شرح زیر است:

شکل ۵-۵: بارهای‌آسیب‌پذیر مربوط به مجموعه‌داده XSS



شکل ۵-۵: بارهای‌آسیب‌پذیر مربوط به مجموعه‌داده SQL Injection

۳-۵ قابلیت ابزار طراحی شده

ابزار VulneraXSQL با ویژگی‌های منحصر به فرد خود به کاربران این امکان را می‌دهد تا آسیب‌پذیری‌های XSS و SQL Injection را به‌طور مؤثر شناسایی و مدیریت کنند. این دو آسیب‌پذیری از مهم‌ترین تهدیدات امنیتی در سامانه‌های‌تحت‌وب به شمار می‌آیند و شناسایی و رفع آن‌ها برای حفظ امنیت سامانه‌های‌تحت‌وب از اهمیت بالایی برخوردار است. طراحی این ابزار به گونه‌ای است که امکان افزودن آسیب‌پذیری‌های جدید به آن در آینده وجود دارد، که به این ترتیب امکان ارتقای ابزار و پاسخگویی به تهدیدات جدید فراهم می‌شود.

برای استفاده از VulneraXSQL، کاربران ابتدا باید ثبت‌نام کنند که این فرآیند از طریق سیستم احراز هویت Django انجام می‌شود. این سیستم به کاربران اجازه می‌دهد تا اطلاعات خود را ثبت، مدیریت و تأیید کنند. Django شامل مدل‌های کاربری پیش‌فرض و قالب‌های نمایشی است و برای افزایش امنیت، از توکن‌های JWT و نشانه‌های دسترسی استفاده می‌شود. پس از ثبت‌نام، کاربر وارد صفحه ورود می‌شود و نام کاربری و رمز عبور خود را وارد می‌کند. سرور پس از تأیید اعتبار، یک نشانه دسترسی و یک نشانه بازیابی تولید و به کاربر ارسال می‌کند. نشانه دسترسی برای ورود به منابع محافظت‌شده استفاده می‌شود و در صورت منقضی شدن، نشانه بازیابی برای دریافت نشانه جدید استفاده می‌شود. زمان اعتبار این توکن‌ها بسته به نیاز و کاربرد آن‌ها قابل تنظیم است.

پس از ورود به سیستم، کاربران به صفحه اصلی منتقل می‌شوند که شامل بخش‌های مختلفی برای ارائه اطلاعات مفید درباره ابزار و آسیب‌پذیری‌ها است. این صفحه شامل تصاویری از صفحه کاربری، گزارش‌های پویش و آرم ابزار است. در نوار ناوبری صفحه، گزینه‌های مختلفی به نمایش درمی‌آید که به کاربران این امکان را می‌دهد تا با توجه به نیاز خود، یکی از گزینه‌های موجود را انتخاب کنند. این انعطاف‌پذیری به کاربران اجازه می‌دهد تا به تناسب نیاز و زمان خود، یکی از گزینه‌های پویش را انتخاب کرده و گزارش کاملی از پویش‌های خود دریافت کنند.

برای پویش سریع‌تر و با سرعت بالاتر، کاربران می‌توانند از بخش Demo استفاده کنند. در این بخش، بارهای آسیب‌پذیر کمتری مورد بررسی قرار می‌گیرند و فقط بخشی از بارهای آسیب‌پذیر مهم بررسی می‌شود. این گزینه برای کاربرانی مناسب است که نیاز به بررسی سریع و کلی دارند. برای بررسی جامع‌تر و دقیق‌تر، گزینه Deep Scan فراهم شده است که با استفاده از بارهای آسیب‌پذیر بیشتر، پویش عمیق‌تری انجام می‌دهد و تمامی جوانب آسیب‌پذیری‌ها را به‌طور کامل مورد بررسی قرار می‌دهد.

پس از انتخاب گزینه پویش، کاربران به صفحه‌ای به نام Dashboard منتقل می‌شوند. در این صفحه، نوار کناری[1] در سمت چپ طراحی شده است که شامل گزینه‌هایی مانند Scan، Target و Report است. این نوار به کاربران امکان می‌دهد تا به بخش‌های مختلف ابزار دسترسی پیدا کنند.

در بخش Scan، کاربران می‌توانند لیست آدرس‌های اینترنتی که قبلاً برای پویش وارد کرده‌اند را مشاهده کنند. اگر کاربر تازه‌وارد باشد، این جدول خالی خواهد بود. این بخش به کاربران این امکان را می‌دهد تا وضعیت پویش‌های قبلی خود را مشاهده کنند و به راحتی به آدرس‌های مورد نظر دسترسی داشته باشند.

در بخش Target، کاربران می‌توانند آدرس اینترنتی مدنظر خود را وارد کنند و با انتخاب یکی از گزینه‌های آسیب‌پذیری (All Vulnerabilities، XSS یا SQL Injection)، پویش را آغاز کنند. با فشردن دکمه Start Scan، پویش آغاز شده و در صورت نیاز، با استفاده از دکمه Stop، می‌توان پویش را متوقف کرد. هنگامی که دکمه Stop فشرده می‌شود، پویش در همان لحظه متوقف شده و کاربر می‌تواند گزارشی از وضعیت پویش تا آن زمان دریافت کند. برای ادامه پویش از نقطه‌ای که متوقف شده، از دکمه Continue استفاده می‌شود. برای مدیریت صحیح فرآیند توقف و از سرگیری پویش، از Celery و RabbitMQ استفاده شده است. Celery وظیفه مدیریت و توقف پویش را بر عهده دارد و RabbitMQ به عنوان واسط پیام، پیام‌های مربوط به وظایف را مدیریت و به کارگران[2]

[1] Sidebar
[2] Workers

Celery ارسال می‌کند. این فناوری‌ها تضمین می‌کنند که فرآیند توقف و ادامه پویش به‌طور مؤثر و بدون مشکل انجام شود.

در صورتی که کاربر تصمیم به توقف پویش نگیرد و اجازه دهد پویش به‌طور کامل انجام شود، پس از اتمام پویش، صفحه‌ای به او نمایش داده می‌شود که امکان دریافت گزارش کامل را فراهم می‌آورد. این گزارش شامل گزینه‌هایی برای دانلود به صورت سند است و کاربران می‌توانند گزارش‌هایی از تمامی آدرس‌های اینترنتی پویش‌شده دریافت کنند، به‌طور کلی و یا به‌صورت جزئی.

در بخش Report، کاربران می‌توانند به تمام گزارش‌های پویش‌های انجام‌شده دسترسی پیدا کرده و آن‌ها را دانلود کنند. این صفحه شامل اطلاعاتی مانند امتیاز، زمان پویش و آدرس اینترنتی اصلی است. این بخش به کاربران این امکان را می‌دهد تا گزارش‌های کامل و مفصلی از پویش‌های قبلی خود دریافت کنند و وضعیت آسیب‌پذیری‌های مختلف را بررسی کنند.

رابط‌کاربری این ابزار به‌گونه‌ای طراحی شده است که بسیار کاربرپسند و آسان است. حتی کاربران بدون دانش فنی زیاد نیز می‌توانند به‌راحتی از این ابزار استفاده کرده و نتیجه مطلوب را دریافت کنند. برای راهنمایی بیشتر، دستورالعمل‌هایی برای استفاده از VulneraXSQL در نظر گرفته شده است که به کاربران کمک می‌کند تا از تمام امکانات ابزار به‌طور مؤثر بهره‌برداری کنند.

رابط کاربری این ابزار به‌طور خاص برای سهولت استفاده طراحی شده است. با استفاده از طراحی مینیمالیستی و قابل فهم، کاربران به راحتی می‌توانند به تمامی امکانات مورد نیاز دسترسی پیدا کنند. اجزای مختلف صفحه، از جمله قالب‌های ورود، نوار کناری، و گزینه‌های پویش به گونه‌ای مرتب شده‌اند که کاربر بتواند به‌سرعت با محیط ابزار آشنا شده و بدون نیاز به آموزش ویژه‌ای از آن استفاده کند.

۵-۴ نتیجه گیری

پروژه ابزار پویش آسیب‌پذیری XSS و SQL Injection یک گام مهم در جهت افزایش امنیت سامانه‌های‌تحت‌وب است. با ترکیب فناوری‌های مدرن مانند Django، React و SQLite، این ابزار به کاربران این امکان را می‌دهد که به راحتی و با دقت بالا وضعیت امنیتی سامانه‌های‌تحت‌وب خود را بررسی کنند. حالت‌های متنوع پویش و احراز هویت قوی از جمله ویژگی‌های این ابزار هستند که آن را به انتخاب مناسبی برای سازمان‌ها و افراد علاقه‌مند به امنیت وب تبدیل می‌کنند. گزارش‌هایی که در اختیار کاربران قرار می‌دهد می‌تواند آن‌ها را خیلی خوب راهنمایی کند. این پروژه با موفقیت توانسته است زیرساختی ساده و کارآمد برای شناسایی و کاهش آسیب‌پذیری‌ها ارائه دهد، که می‌تواند نقش مهمی در افزایش امنیت دیجیتال ایفا کند.

فصل ششم: فناوری‌های مورد استفاده

۶-۱ مقدمه

در دنیای مدرن فناوری اطلاعات، انتخاب فناوری‌های مناسب برای توسعه یک پروژه نرم‌افزاری نقش حیاتی دارد. این انتخاب‌ها می‌توانند تاثیرات عمیقی بر عملکرد، مقیاس‌پذیری، و قابلیت نگهداری سیستم داشته باشند. در این فصل، به بررسی فناوری‌های مختلفی که در پروژه ما مورد استفاده قرار گرفته‌اند خواهیم پرداخت. پروژه ما به‌طور خاص از فناوری‌های متنوعی بهره می‌برد که شامل پایگاه داده‌ها، کتابخانه‌ها و فریم‌ورک‌های مختلف، و ابزارهای ارتباطی است.

۶-۲ فناوری‌های سمت سرور

در توسعه نرم‌افزار، بخش سمت سرور، مسئول پردازش داده‌ها، مدیریت پایگاه داده‌ها، و انجام منطق‌های تجاری است. برای پیاده‌سازی این بخش، از فناوری‌ها و ابزارهای مختلفی استفاده شده‌است که در ادامه به تفصیل معرفی می‌شوند.

۶-۲-۱ چارچوب Django:

یکی از چارچوب‌های محبوب و قدرتمند توسعه وب در زبان برنامه‌نویسی پایتون، Django است. این چارچوب به دلیل ویژگی‌های قدرتمند و ساده‌سازی فرآیند توسعه وب، انتخاب مناسبی برای ساخت برنامه‌های کاربردی تحت وب محسوب می‌شود. به عنوان یک چارچوب تمام‌عیار (Full-Stack) طراحی شده‌است، که به معنای ارائه ابزارها و قابلیت‌های مورد نیاز برای توسعه برنامه‌های وب از جمله مدیریت پایگاه داده، مدل‌های داده، و ایجاد رابط‌کاربری است.

در این ابزار از کتابخانه‌های مفید و زیادی استفاده شده که یکی از پرکاربردترین آن‌ها، BeautifulSoup است. این کتابخانه اجازه می‌دهد تا هنگام پویش برای بررسی آسیب پذیری‌های XSS و SQL Injection، به اجزای مختلف یک سند HTML دسترسی پیدا کند و اطلاعات لازم را استخراج کند. می‌تواند قالب‌های ورودی یک صفحه را استخراج کند به طور خودکار بارهای‌آسیب‌پذیر امنیتی را در آن‌ها وارد کند.

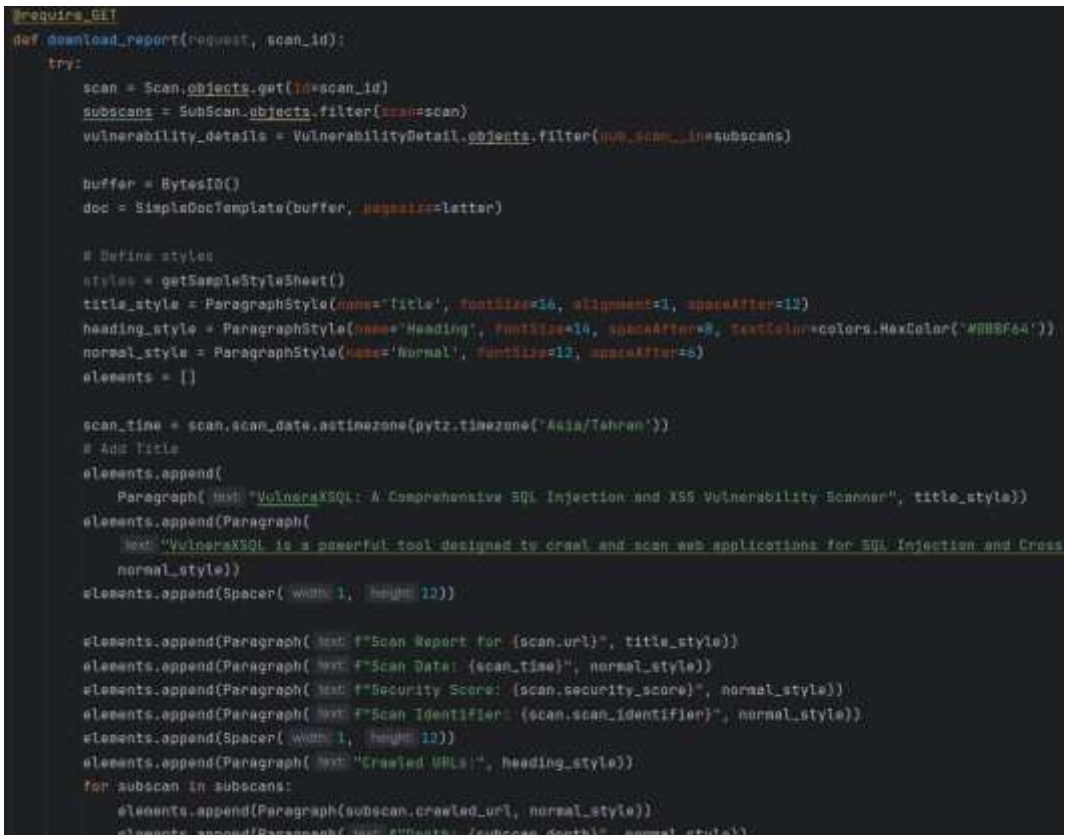
از کتابخانه‌های دیگری که باعث افزایش کیفیت گزارش ارائه شده، Matplotlib است. ابزار قدرتمندی برای رسم نمودار است. نتایج پویش‌های امنیتی (مانند درصد آسیب‌پذیری‌های XSS و SQL Injection) را با استفاده از نمودارهای دایره‌ای، به کاربر نمایش می‌دهد. این کار می‌تواند درک بصری بهتری از وضعیت امیتاز امنیتی سامانه‌های تحت‌وب به کاربر بدهد. در این قسمت به برخی از امکانات استفاده شده در پروژه می‌پردازیم:

- چارچوب جنگو به توسعه‌دهندگان این امکان را می‌دهد تا مدل‌های داده را به‌سادگی تعریف کنند و از امکانات (Object-Relational Mapping) ORM برای تعامل با پایگاه داده استفاده کنند. در پروژه ما، مدل‌هایی مثل scan و vulnerabilityDetail به صورت مستقیم با جداول پایگاه داده مرتبط هستند. ORM به ما این امکان را می‌دهد که این مدل‌ها را به راحتی مدیریت کرده و عملیات‌هایی مثل ایجاد، خواندن، بروزرسانی و حذف (CRUD) را انجام دهیم. این ویژگی به سادگی مدیریت و تعامل با داده‌ها را تسهیل می‌کند.از مدل‌هایی که در پروژه استفاده کردیم:

```python
from datetime import timedelta
from django.conf import settings
from django.db import models
from django.utils import timezone
from django.contrib.auth import get_user_model

User = get_user_model()
class Scan(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    url = models.URLField()
    scan_date = models.DateTimeField(auto_now_add=True)
    last_scan_date = models.DateTimeField(default=timezone.now)
    is_vulnerable = models.BooleanField(default=False)
    vulnerabilities = models.TextField(blank=True)
    security_score = models.IntegerField(default=100)
    main_url = models.URLField(max_length=200, null=True, blank=True)
    scan_identifier = models.CharField(max_length=36, blank=True, null=True)

    def __str__(self):
        return self.url

    3 usages (3 dynamic)
    def can_rescan(self):
        return timezone.now() - self.last_scan_date > timedelta(hours=1)

    def update_last_scan_date(self):
        self.last_scan_date = timezone.now()
        self.save()
```

شکل۶-۱: کد مرتبط با Model در جنگو

- مسئول پردازش درخواست‌ها و تولید پاسخ‌ها، View هستند. آنها بخش‌هایی از کد هستند که تصمیم می‌گیرند چه داده‌هایی باید به کاربر نمایش داده شود و چه اقداماتی باید در پاسخ به درخواست‌های HTTP[1] انجام شود. در Django، Viewها می‌توانند به‌صورت توابع ساده یا کلاس‌های مبتنی بر API تعریف شوند. به عنوان مثال بعد از اینکه پویش انجام شود، با استفاده از تابع هایی که در قسمت view تعریف شده‌است، کاربران گزارش‌ها را می‌توانند به راحتی به صورت سند دانلود کنند.



```
@require_GET
def download_report(request, scan_id):
    try:
        scan = Scan.objects.get(id=scan_id)
        subscans = SubScan.objects.filter(scan=scan)
        vulnerability_details = VulnerabilityDetail.objects.filter(sub_scan__in=subscans)

        buffer = BytesIO()
        doc = SimpleDocTemplate(buffer, pagesize=letter)

        # Define styles
        styles = getSampleStyleSheet()
        title_style = ParagraphStyle(name='Title', fontSize=16, alignment=1, spaceAfter=12)
        heading_style = ParagraphStyle(name='Heading', fontSize=14, spaceAfter=8, textColor=colors.HexColor('#0B0F64'))
        normal_style = ParagraphStyle(name='Normal', fontSize=12, spaceAfter=6)
        elements = []

        scan_time = scan.scan_date.astimezone(pytz.timezone('Asia/Tehran'))
        # Add Title
        elements.append(
            Paragraph( text "VulnoraXSQL: A Comprehensive SQL Injection and XSS Vulnerability Scanner", title_style))
        elements.append(Paragraph(
            text "VulnoraXSQL is a powerful tool designed to crawl and scan web applications for SQL injection and Cross
            normal_style))
        elements.append(Spacer( width: 1,  height: 12))

        elements.append(Paragraph( text f"Scan Report for {scan.url}", title_style))
        elements.append(Paragraph( text f"Scan Date: {scan_time}", normal_style))
        elements.append(Paragraph( text f"Security Score: {scan.security_score}", normal_style))
        elements.append(Paragraph( text f"Scan Identifier: {scan.scan_identifier}", normal_style))
        elements.append(Spacer( width: 1,  height: 12))
        elements.append(Paragraph( text "Crawled URLs:", heading_style))
        for subscan in subscans:
            elements.append(Paragraph(subscan.crawled_url, normal_style))
```

شکل۶-۲: نمونه ای از بخش View مربوط به report

[1] HyperText Transfer Protocol

```python
@require_GET
def index(request):
    form = ScanForm()
    return render(request, template_name: 'server1/index.html', context: {'form': form})

2 usages
class StartScanView(APIView):
    permission_classes = [IsAuthenticated]

    def post(self, request, *args, **kwargs):
        if not request.user.is_authenticated:
            return JsonResponse( data: {'error': 'User not authenticated'}, status=401)

        try:
            scan_identifier = str(uuid.uuid4())
            print(f"Raw request body: {request.body}")
            data = json.loads(request.body)
            print(f"Received data: {data}")
            form = ScanForm( *args: data, current_user=request.user)
```

شکل۶-۳: نمونه ای از بخش View مربوط به scan

- از امکانات خوبی که در جنگو وجود دارد و می‌توان به راحتی از آن‌ها استفاده کرد، Formها هستند. Formها برای دریافت ورودی از کاربران و اعتبارسنجی آن استفاده می‌شوند. Formها به ما این امکان را می‌دهند که داده‌هایی را از کاربران جمع‌آوری کنیم، آن‌ها را بررسی کنیم و در نهایت به یک مدل در پایگاه داده متصل کنیم. در این پروژه، ScanForm برای ثبت یا تغییر یک نشانی اینترنتی در مدل Scan استفاده می‌شود. با استفاده از ModelForm، Form به طور خودکار به مدل Scan متصل می‌شود و داده‌های ارسال شده توسط کاربر به طور مستقیم می‌توانند به این مدل مرتبط شوند. برای اعتبارسنجی از روش[1] Clean_url استفاده کرده‌ایم. این روش یک مثال از اعتبارسنجی سفارشی در جنگو است که به ما این امکان را می‌دهد که شرایط خاصی را برای ورودی‌های فرم تعیین کنیم. این روش در پروژه ما تعریف شده‌است تا بررسی کند نشانی اینترنتی وارد شده توسط کاربر، قبلا ثبت

[1] Method

شده‌است یا خیر. در حقیقت تکراری بودن نشانی اینترنتی را بررسی می‌کند و از پویش مجدد یک نشانی اینترنتی در یک بازه زمانی کوتاه جلوگیری می‌کند. این موضوع می‌تواند به جلوگیری از هدررفت منابع و بهینه‌سازی فرآیند پویش کمک کند.

```python
from django import forms
from .models import Scan

class ScanForm(forms.ModelForm):
    class Meta:
        model = Scan
        fields = ['url']

    def clean_url(self):
        url = self.cleaned_data.get('url')
        if not url:
            raise forms.ValidationError("URL cannot be empty.")

        recent_scan = Scan.objects.filter(url=url).order_by('-scan_date').first()

        if recent_scan and not recent_scan.can_rescan():
            raise forms.ValidationError(
                "This URL was scanned recently. Please wait for at least an hour or request a new scan."
            )
        return url

    def __init__(self, *args, **kwargs):
        self.current_user = kwargs.pop('current_user', None)
        super().__init__(*args, **kwargs)
```

شکل۶-۴: نمونه ای از بخش Form مربوط به scan

- در قسمت Backend ، بخشی به نام خزش[1] وجود دارد. این بخش، صفحات وب را به‌طور خودکار خزیده و آسیب‌پذیری‌های SQL Injection و XSS را بررسی می‌کند. نشانی‌های اینترنتی معتبر به‌صورت تودرتو خزیده شده و اطلاعات مربوط به آسیب‌پذیری‌ها در پایگاه داده ذخیره می‌شود. این فرآیند به صورت ساختاریافته با استفاده از مدل‌های Scan, SubScan, VulnerabilityDetail انجام

[1] Crawler

می‌گیرد. خزش شامل فرآیندهایی برای مدیریت عمق خزیدن و جلوگیری از پویش مجدد نشانی‌های اینترنتی تکراری است.

```python
def is_valid_url(url, base_url):
    parsed_url = urlparse(url)
    return bool(parsed_url.scheme and parsed_url.netloc and url.startswith(base_url))

4 usages
def crawl(url, scan_id,max_depth = 2):
    visited_urls = set()
    urls_to_visit = [(url, 0)]
    parent_scan = Scan.objects.get(id=scan_id)

    while urls_to_visit:
        current_url, depth = urls_to_visit.pop(0)
        if current_url in visited_urls or depth > max_depth:
            continue

        visited_urls.add(current_url)
        try:
            response = requests.get(current_url)
            response.raise_for_status()

            print(f"Crawling URL: {current_url} at depth {depth}")
            soup = BeautifulSoup(response.text, features: 'html.parser')

            sql_injection_results = check_sql_injection(current_url)
            xss_results = check_xss(current_url)

            vulnerabilities = sql_injection_results + xss_results

            sub_scan = SubScan.objects.create(
                crawled_url=current_url,
                scan_date=parent_scan.scan_date,
                is_vulnerable=bool(vulnerabilities),
                vulnerabilities=json.dumps(vulnerabilities),
```

شکل۶-۵: کد مربوط به بخش خزش ابزار

- بخش نرم‌افزارهای کمکی[1]، یک ابزار پایه برای بررسی آسیب‌پذیری‌های امنیتی SQL Injection و XSSدر یک نشانی اینترنتی است. تابع check_sql_injection با تزریق مجموعه‌ای از بارهای‌آسیب‌پذیر مخرب به متغیرهای نشانی اینترنتی، تلاش می‌کند تا با ارسال درخواست‌های HTTP و تحلیل پاسخ‌ها، نشانه‌هایی از وجود آسیب‌پذیری SQL Injection را شناسایی کند. اگر در پاسخ به درخواستHTTP ، خطایی مرتبط با پایگاه داده مشاهده شود (به عنوان مثال پیام" خطا[2] " در پاسخ مشاهده شود)، این تابع نتیجه‌ای حاکی از وجود آسیب‌پذیری را برمی‌گرداند. تابع check_xss نیز مشابه SQL Injection ، با ارسال بارهای‌آسیب‌پذیر مخصوص XSS به متغیرهای نشانی اینترنتی، تلاش می‌کند تا وجود این نوع آسیب‌پذیری را تشخیص دهد. در صورت شناسایی هر گونه واکنش نامطلوب از سمت سرور، این تابع نتیجه‌ای مبنی بر وجود آسیب‌پذیری XSS ارائه می‌دهد.

```python
def check_sql_injection(url):
    logger.info(f"URL received in check_sql_injection: {url}")
    if not url or not isinstance(url, str):
        raise ValueError("URL must be a non-empty string")

    if not validate_url(url):
        raise ValueError("Invalid URL: URL must be a valid and non-empty URL")

    scan_results = []
    payloads = [
        "' OR '1'='1",
        "' OR '1'='1' --",
        "' OR '1'='1' /*",
        "' OR ''='",
        "' OR 1=1--",
        "' OR 1=1#",
        "' OR 1=1/*",
        "admin' --",
        "' UNION SELECT NULL, NULL, NULL --",
        "' UNION SELECT 1, 'text', NULL --",
        "' AND '1'='1",
        "' AND '1'='1' --",
        "' AND 1=1",
        "' AND 1=1--",
        "' AND 1=1#",
        "' AND 1=1/*"
    ]
```

شکل۶-۶: تابع و Paylaodهای مربوط به SQL Injection

[1] Utils
[2] error

```python
def check_xss(url):
    logger.info(f"URL received in check_xss: {url}")

    vulnerabilities = []
    XSS_PAYLOADS = [
        '<script>alert(1)</script>',
        '"><script>alert(1)</script>',
        '"><img src=x onerror=alert(1)>',
        '<img src=x onerror=alert(1)>',
        '<svg/onload=alert(1)>',
        '<iframe src="javascript:alert(1)"></iframe>',
    ]

    for payload in XSS_PAYLOADS:
        try:
            response = requests.get(url, params={'q': payload}, timeout=5)
            logger.info(f"Testing with payload: {payload}, Status Code: {response.status_code}")
            logger.info(f"Response Text: {response.text[:500]}")

            if "error" in response.text.lower() or response.status_code == 500:
                vulnerabilities.append({
                    'payload': payload,
                    'response': 'XSS vulnerability found',
                    "description": "Possible XSS Injection vulnerability detected"
                })
            else:
                logger.info(f"No XSS detected with payload: {payload}")
```

شکل۶-۷: تابع و Paylaodهای مربوط به XSS

- بخش Deep در این پروژه به منظور بررسی و تشخیص حملات و آسیب‌پذیری‌های امنیتی پیشرفته طراحی شده‌است. این بخش با استفاده از روش‌های مبتنی بر داده‌ها، تهدیداتی مانند SQL Injection و Cross-Site Scripting (XSS) را شناسایی می‌کند. هدف اصلی این قسمت، تحلیل دقیق درخواست‌ها و پاسخ‌ها به همراه داده‌های مخرب در دنیای واقعی است. برای این منظور، از مجموعۀداده‌ها استفاده شده‌است که شامل نمونه‌هایی از انواع حملات متداول هستند. دو مجموعۀداده اصلی برای بررسی امنیتی در نظر گرفته شده‌است. اولین مجموعۀداده شامل مجموعه‌ای از درخواست‌هایSQL است که هدف آنها بهره‌برداری از ضعف‌های امنیتی موجود در پایگاه داده‌هاست.

این درخواست‌ها شامل حملاتی هستند که می‌توانند به مهاجم امکان دسترسی غیرمجاز به اطلاعات، تغییر داده‌ها یا اجرای دستورات دلخواه را بدهند. مجموعۀداده بعدی شامل مجموعه‌ای از حملات XSSاست که به مهاجم اجازه می‌دهد کدهای مخرب JavaScript را در مرورگر قربانی اجرا کند. این حملات می‌توانند منجر به دزدیدن کوکی‌ها، اطلاعات کاربران، یا تغییرات مخرب در صفحات وب شوند. بخش Deep با استفاده از این مجموعۀداده‌ها، درخواست‌های ورودی را به صورت دقیق تجزیه و تحلیل کرده و تلاش می‌کند تا هرگونه رفتار غیرعادی یا مخرب را شناسایی کند. الگوریتم‌های به کار رفته در این بخش از اطلاعات مجموعۀداده‌ها برای شناسایی الگوهای حمله استفاده می‌کنند. تمام ورودی‌های کاربر بررسی شده و با داده‌های موجود در مجموعۀداده مقایسه می‌شوند. سیستم تلاش می‌کند تا الگوهای مشابه با حملات SQL Injection و XSS را شناسایی کند. در صورتی که الگوهای مخرب تشخیص داده شوند، هشدارهای امنیتی به مسئولان سیستم ارسال می‌شود.

- در فصل قبل به این موضوع اشاره شد که برای بهبود عملکرد بخش‌های مختلف سیستم و مدیریت وظایف زمان‌بر، از Celery به عنوان یک سیستم صف‌بندی وظایف توزیع‌شده استفاده شده است . Celery به‌طور خاص برای این منظور طراحی شده است که اجازه می‌دهد عملیات‌های زمان‌بر به پس‌زمینه منتقل شوند تا برنامه اصلی بتواند همچنان پاسخگو بماند و از بروز مشکلاتی مانند کندی عملکرد جلوگیری شود. این ویژگی به ویژه در پروژه‌هایی که شامل پویش‌های امنیتی یا پردازش‌های سنگین هستند، کاربردی است. این موضوع برای سیستم‌های توزیع‌شده که وظایف می‌توانند روی سرورها یا فرآیندهای مختلف تقسیم شوند، بسیار حیاتی است. در این بخش، نحوه استفاده از Celery برای متوقف کردن پویش‌ها، به همراه فناوری RabbitMQ به‌عنوان پیام‌رسان، به تفصیل توضیح داده می‌شود. در این بین استفاده کردن از فناوری RabbitMQ بسیار مفید و سودمند بود. در اینجا از

RabbitMQ به عنوان واسط پیام[1] برای Celery استفاده می‌شود. وقتی کاربر یک وظیفه(مثل شروع، توقف یا ادامه پویش) صادر می‌کند، Celery یک پیام به RabbitMQ ارسال می‌کند. RabbitMQ این پیام را صف می‌کند و مطمئن می‌شود که به یکی از کارگران Celery تحویل داده شود. سپس کارگر Celery وظیفه را اجرا می‌کند. در این پروژه، Celery برای مدیریت وظایف مرتبط با پویش‌ها به‌کار می‌رود، از جمله متوقف کردن. در پروژه، تابعی به نامStop_scan_task به‌عنوان یک وظیفه Celery تعریف شده است. این وظیفه به کاربران این امکان را می‌دهد که یک پویش در حال اجرا را متوقف کنند. هنگامی که کاربر تصمیم به توقف پویش می‌گیرد، این تابع فعال می‌شود. در این وظیفه، ابتدا شناسه پویش[2] را دریافت می‌کند، سپس از میان پویش‌هایی که در حال اجرا هستند و وضعیت آن "درحال اجرا[3]" است، انتخاب می‌کند و وضعیت آن‌ها را به ناموفق[4] تغییر می‌دهد. این تغییر وضعیت، به معنی متوقف شدن پویش است. این روند به‌ویژه زمانی که چندین پویش به‌طور هم‌زمان در حال اجرا هستند، بسیار حائز اهمیت است زیرا می‌توان به‌طور دقیق مشخص کرد که کدام پویش باید متوقف شود.

- برای ارسال و دریافت پیام‌ها میان Celery و کارگران، از RabbitMQ به‌عنوان پیام‌رسان استفاده برای ارسال و دریافت پیام‌ها میان Celery و کارگران، از RabbitMQ به‌عنوان پیام‌رسان استفاده می‌شود که نقش کلیدی در مدیریت و هماهنگی ارتباطات میان Celery و وظایف آن دارد. RabbitMQ به‌عنوان یک سیستم صف‌بندی پیام، این امکان را فراهم می‌آورد که پیام‌های مربوط به وظایف به‌طور مؤثر و منظم منتقل شوند. زمانی که کاربر یک وظیفه مانند شروع، توقف، یا ادامه پویش را صادر می‌کند، Celery یک پیام حاوی جزئیات این وظیفه به RabbitMQ ارسال می‌کند. RabbitMQ این

[1] Message broker
[2] Scan Identifier
[3] in_progress
[4] Failed

پیام را در صف خود قرار داده و آن را به یکی از کارگران Celery تحویل می‌دهد. کارگران Celery وظایف دریافت شده را به نوبت و با دقت اجرا می‌کنند. این سیستم صف‌بندی، اطمینان حاصل می‌کند که پیام‌ها به‌طور منظم و بدون افت یا گم‌شدگی به کارگران منتقل می‌شوند و هر وظیفه در زمان مناسب اجرا می‌شود. یکی از مزایای استفاده از RabbitMQ، توانایی آن در مدیریت و هماهنگی بین چندین سرور و فرآیند است. این سیستم به‌ویژه در محیط‌های توزیع‌شده که در آن وظایف ممکن است بر روی سرورها یا فرآیندهای مختلف تقسیم شوند، حیاتی است. RabbitMQ تضمین می‌کند که پیام‌ها به‌صورت قابل اعتماد و منظم به مقصد برسند و وظایف به‌طور صحیح پردازش شوند، حتی در شرایطی که بار ترافیکی بالا باشد یا سیستم تحت فشار قرار گیرد. علاوه‌بر این، RabbitMQ با ارائه امکاناتی مانند پایش وضعیت صف‌ها، تنظیم اولویت‌ها و مدیریت زمان‌بندی پیام‌ها، به بهینه‌سازی عملکرد سیستم کمک می‌کند. این ویژگی‌ها باعث می‌شود که ارتباطات میان Celery و کارگران در مقیاس‌های بزرگتر و در شرایط پیچیده به‌طور مؤثر و کارآمد مدیریت شوند.

۳-۶ فناوری‌های سمت کلاینت

در برای ایجاد ظاهری زیبا و تعاملی در پروژه، از کتابخانه React استفاده شده است. React یک کتابخانه متن‌باز است که برای ساخت رابط‌های کاربری توسعه داده شده و توسط فیس‌بوک نگهداری می‌شود. این کتابخانه به توسعه‌دهندگان این امکان را می‌دهد تا رابط‌های کاربری پیچیده و غنی را با استفاده از مؤلفه [۱] مستقل و قابل استفاده مجدد طراحی کنند.

React بر اساس مفهوم " مدل شیء مستند مجازی [۲]" عمل می‌کند، که به‌طور خلاصه به معنای استفاده از یک نمای داخلی و بهینه‌شده از مدل شیء مستند[۳] واقعی برای بهبود کارایی و سرعت برنامه‌های وب است. مدل شیء مستند مجازی به این صورت کار می‌کند که تغییرات در رابط کاربری به‌جای اعمال مستقیم بر روی مدل شیء مستند واقعی، ابتدا به مدل شیء مستند مجازی اعمال می‌شود. سپس، React تفاوت‌ها را میان مدل شیء مستند مجازی و واقعی مقایسه کرده و تنها تغییرات لازم را به مدل شیء مستند واقعی اعمال می‌کند. این فرآیند باعث کاهش تعداد عملیات‌های مستقیم بر روی مدل شیء مستند می‌شود و در نتیجه، عملکرد و سرعت برنامه افزایش می‌یابد.

React به دلیل سادگی و کارایی بالا، یکی از پرطرفدارترین ابزارهای توسعه وب در دنیا است و توسط بسیاری از شرکت‌ها و توسعه‌دهندگان برای ساخت برنامه‌های کاربردی وب و به‌ویژه برنامه‌های تک صفحه‌ای[۴] استفاده می‌شود. این کتابخانه به توسعه‌دهندگان این امکان را می‌دهد تا تجربه کاربری روان و تعاملی را با بارگذاری سریع‌تر صفحات و به‌روزرسانی‌های آنی فراهم کنند.

[1] Component
[2] Virtual DOM
[3] Document Object Model
[4] SPA

علاوه بر این، نسخه‌های جدید React شامل ویژگی‌ای به نام "Hook" هستند که به توسعه‌دهندگان اجازه می‌دهند تا قابلیت‌های مختلف React، مانند مدیریت حالت[1] و روش‌های چرخه حیات[2]، بدون نیاز به استفاده از کلاس‌ها پیاده‌سازی کنند. این ویژگی به ساده‌سازی کد و بهبود قابلیت استفاده مجدد از کد کمک می‌کند.

برای بخش رابط کاربری در پروژه از مولفۀها و ویژگی‌هایی استفاده شده‌است که در این فصل به آن‌ها می‌پردازیم:

- مؤلفه APP در پروژه VulneraXSQL به عنوان نقطه ورودی برای برنامه React عمل می‌کند و ساختار اصلی مسیرها و مؤلفه‌ها را تنظیم می‌کند. این مؤلفه شامل تنظیمات مربوط به مسیریابی، مدیریت وضعیت کاربر و ارائه محتواهای لازم برای بخش‌های مختلف برنامه است.

```
  }, []);
  return (
    <Router>
      <AuthProvider>
      <Navbar user={user} />
      <Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="/login" element={<Login setUser={setUser}/>} />
        <Route path="/signup" element={<Signup />} />
        <Route path="/dashboard" element={<Dashboard user={user}/>}>
          <Route path="scan" element={<ScanPage />} />
          <Route path="target" element={<TargetPage />} />
          <Route path="report" element={<UserScans />} />
        </Route>
        <Route path="/report-page/:scanId" element={<ReportPage />} />
      </Routes>
      </AuthProvider>
    </Router>
  );
};
```

شکل۶-۸: مسیرهای مربوط به پروژه در App.js

[1] State
[2] lifecycle methods

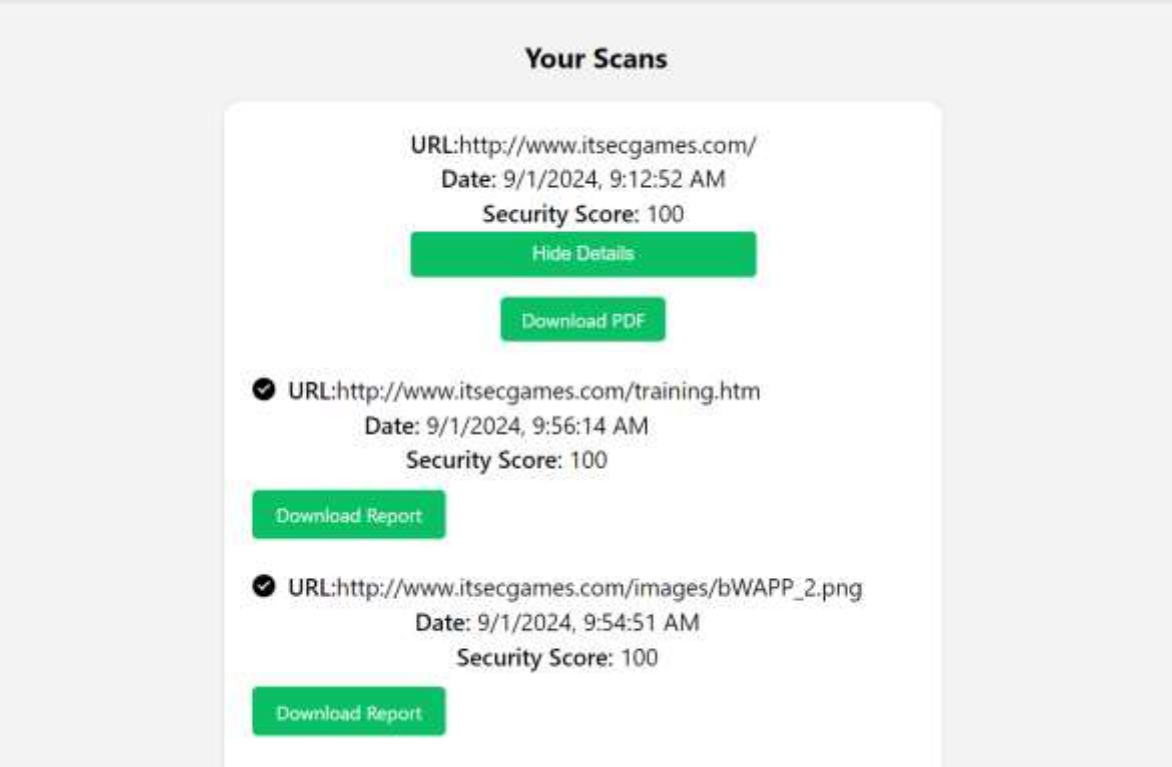
- صفحه اصلی، به عنوان اولین صفحه‌ای که کاربران مشاهده می‌کنند، طراحی شده‌است. با استفاده ازuseState و useEffect، وضعیت ورود کاربر بررسی می‌شود و اگر کاربر وارد نشده باشد، دکمه‌های ورود به سیستم فعال می‌شوند. اگر نشانه[1] ورود در ذخیره‌سازی محلی[2] موجود باشد، درخواست به سرور ارسال می‌شود تا وضعیت کاربر تأیید شود. اگر وضعیت احراز هویت کاربر مناسب نباشد یا خطایی در دریافت داده‌ها به وجود بیاید، نشانه حذف شده و کاربر به صفحه ورود هدایت می‌شود.

شکل۶-۹: مؤلفه **های استفاده شده در HomePage**

[1] Token
[2] localStorage

- مؤلفه TargetPage وUserScans هر دو بخش‌های کلیدی رابط‌کاربری برنامه را مدیریت می‌کنند. مؤلفه TargetPage برای شروع پویش‌های امنیتی طراحی شده و شامل قالب ورودی نشانی اینترنتی و انتخاب نوع پویش است. این مؤلفه با استفاده از وضعیت‌های مختلف مانند scanType، url، loading عملیات اعتبارسنجی نشانی اینترنتی و ارسال درخواست به سرور را مدیریت می‌کند. در صورت موفقیت‌آمیز بودن ارسال، کاربر به صفحه‌ای جدید هدایت می‌شود و در صورت بروز خطا، پیام خطا به کاربر نمایش داده می‌شود. در سوی دیگر، مؤلفه UserScans به نمایش و مدیریت پویش‌های امنیتی قبلی پرداخته و از قابلیت‌های محتوا برای مدیریت وضعیت فهرست کناری استفاده می‌کند. این مؤلفه داده‌های مربوط به پویش‌های کاربر را از سرور دریافت کرده و به کاربر این امکان را می‌دهد تا جزئیات پویش‌ها را مشاهده کرده و گزارش‌های سند را دانلود کند. در سندی که کاربران مشاهده می‌کنند، اطلاعات مفیدی وجود دارد. زمان پویش انجام شده، امتیاز امنیتی، نشانی‌های اینترنتی خزیده شده و وضعیت[1] برای نشانی اینترنتی هم نمایش داده می‌شود. کاربران می‌توانند گزارش نشانی‌های اینترنتی خزیده شده را نیز به صورت جداگانه دانلود کنند و گزارشی با جزئیات بیشتر داشته باشند.

[1] status



شکل۶-۱۰: بخش رابط کاربری مربوط به دانلود گزارش‌های پویش‌های انجام شده

- از Context API برای مدیریت وضعیت سراسری در برنامه استفاده می‌شود. این API به ما کمک می‌کند تا وضعیت‌هایی مانند احراز هویت کاربر یا وضعیت نوار کناری را به‌صورت متمرکز مدیریت کنیم و نیازی به ارسال ویژگی‌ها[1] به‌صورت دستی از یک مؤلفه به دیگری نداشته باشیم. برای مدیریت وضعیت احراز هویت کاربران از کتابخانه AuthContext استفاده شد. این محتوا نشانه JWT را نگه‌داری می‌کند و از طریق آن، وضعیت ورود یا خروج کاربر را مدیریت می‌کند. همچنین برای مدیریت وضعیت نوار کناری در برنامه از SidebarContext استفاده شده‌است. این محتوا وضعیت باز یا بسته بودن نوار کناری را در خود نگه‌داری می‌کند.

[1] props

- برای ارسال درخواست‌های HTTP، از کتابخانه ای به نام Axois استفاده شده‌است. مدیریت زمان‌ها، ایجاد درخواست‌های پیچیده و برخورداری از دسترسی آسان به داده‌های پاسخ را فراهم می‌کند. ما در این پروژه از Axios برای ارتباط با API سرور استفاده می‌کنیم.

۶-۴ فناوری امنیتی

معمولاً برای احراز هویت و امنیت در برنامه‌های وب از JWT استفاده می‌شود. در پروژه ما، JWT در هر دو قسمت رابط‌کاربری و سمت سرور استفاده می‌شود، اما نقش آن‌ها در هر قسمت متفاوت است. به دلیل استفاده از JWT برای احراز هویت، دو نشانه به نام‌های نشانه دسترسی و نشانه بازیابی کاربرد دارند تا تجربه کاربری بهتری را ایجاد کنند و همینطور امنیت سیستم را بهبود ببخشند.

۶-۴-۱ JWT

یک استاندارد باز برای تبادل امن اطلاعات بین مشتری[1] و سرور مورد استفاده قرار می‌گیرد. این تبادل به صورت ایمن انجام می‌شود زیرا اطلاعات داخل JWT به صورت الکترونیکی امضا می‌شود. این امضا معمولاً با استفاده از یک کلید مخفی[2] (که فقط بین دو طرف شناخته شده‌است) یا یک کلید عمومی[3]/خصوصی[4] (در موارد استفاده از امضای RSA یا ECDSA) انجام می‌شود که از سه بخش اصلی تشکیل شده‌است که با "." از دیگری جدا می‌شود.

- سرصفحه[5]: این بخش شامل اطلاعاتی درباره‌ی نوع نشانه (که در اینجا JWT است) و الگوریتم امضای استفاده‌شده (مانند HMAC SHA۲۵۶ یا RSA) است.

[1] Client
[2] Secret Key
[3] Public Key
[4] Private Key
[5] Header

- بارآسیب‌پذیر: این بخش حاوی داده‌های مورد نظر برای تبادل است. داده‌ها به صورت یک شیء JSON قرار می‌گیرند. در این بخش می‌توان اطلاعاتی مانند شناسه کاربر[1]، نقش کاربر[2] و دیگر موارد را قرار داد.

- امضا[3]: این بخش به منظور تأیید اعتبار نشانه استفاده می‌شود و از ترکیب سرصفحه، محتوا، و یک کلید مخفی یا عمومی ایجاد می‌شود. این امضا تضمین می‌کند که محتوا در طول تبادل تغییری نکرده است.

معمولا در نقشه‌های زیر می‌توان از JWT استفاده کرد:

- احراز هویت[4]: بعد از اینکه کاربر با موفقیت وارد سیستم شد، سرور یک JWT تولید کرده و به مشتری ارسال می‌کند. مشتری از این نشانه در سرصفحه درخواست‌های بعدی استفاده می‌کند تا هویت خود را تأیید کند. این روش به عنوان "Stateless Authentication" شناخته می‌شود، زیرا نیازی به ذخیره کردن اطلاعات جلسه[5] در سرور نیست.

- مجوزدهی[6]: از JWT می‌توان برای تعیین سطح دسترسی کاربران استفاده کرد. به عنوان مثال، در محتوای JWT می‌توان نقش‌های کاربر (مثل مدیر[7] یا کاربر عادی) را قرار داد و بر اساس این نقش‌ها دسترسی به منابع مختلف را کنترل کرد.

[1] user ID
[2] user roles
[3] Signature
[4] Authentication
[5] session
[6] Authorization
[7] Admin

- انتقال داده‌های امن: به دلیل اینکه اطلاعات داخل JWT به صورت الکترونیکی امضا می‌شود، می‌توان از آن برای تبادل داده‌هایی که باید از تغییرات محافظت شوند، استفاده کرد.

۶-۴-۲ Access Token

یک نشانه JWT است که برای دسترسی به منابع محافظت‌شده[1] استفاده می‌شود. این نشانه پس از ورود موفقیت‌آمیز کاربر ایجاد می‌شود و در هر درخواست به سرور ارسال می‌شود تا اعتبار سنجی کاربر انجام شود. نشانه دسترسی معمولاً عمر کوتاهی دارد تا در صورت دزدیده‌شدن، به سرعت منقضی شود و جلوی سوءاستفاده گرفته شود. در پروژه VulneraXSQL ، به دلیل طولانی بودن پویش‌هایی که انجام می‌شود، زمان این نشانه را به مدت یک روز قرار دادیم تا در حین انجام پویش هیچ مشکلی ایجاد نشود و کاربر بتواند خیلی راحت از ابزار استفاده کند. به دلیل عمر کوتاه آن‌ها، در صورت منقضی شدن نیاز به تمدید توسط نشانه بازیابی دارند. این تنظیمات به بهبود امنیت سیستم و تجربه کاربری کمک می‌کنند.

```
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(days=1),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
    'ROTATE_REFRESH_TOKENS': True,
    'BLACKLIST_AFTER_ROTATION': True,
}
```

شکل۶-۱۱: تنظیمات مربوط به JWT

۶-۵ پایگاه داده

در این پروژه، از پایگاه داده SQLite استفاده شده‌است. SQLite یک پایگاه داده‌ی رابطه‌ای است که به دلیل سبک بودن در تنظیمات، به‌خصوص در مراحل توسعه و آزمایش، بسیار محبوب است. این پایگاه داده تمام اطلاعات پروژه از جمله اطلاعات پویش‌ها، جزئیات آسیب‌پذیری‌ها، و کاربرانی که این پویش‌ها را اجرا می‌کند، ذخیره

[1] Protected Resources

می‌کند. SQLite نیازی به سرور جداگانه ندارد و تمام داده‌ها در یک فایل منفرد ذخیره می‌شوند. این ویژگی باعث شده‌است که SQLite برای پروژه‌هایی با مقیاس متوسط یا پروژه‌هایی که به استقرار سریع نیاز دارند، انتخاب مناسبی باشد.

تنظیمات مربوط به پایگاه داده در فایل settings.py قرار دارد. در این پروژه، پایگاه داده به صورت پیش‌فرض SQLite است اما با استفاده از کتابخانه dj-database-url می‌توان آن را به پایگاه داده دیگری تغییر داد. این تنظیمات به گونه‌ای است که اگر متغیر محیطی DATABASE_URL تنظیم شده باشد، از آن استفاده می‌شود و در غیر این صورت از SQLite به عنوان پایگاه داده پیش‌فرض استفاده خواهد شد.

```
DATABASES = {
    'default': dj_database_url.parse(
        os.environ.get('DATABASE_URL', 'sqlite:///' + str(BASE_DIR / 'db.sqlite3'))
    )
}
```

شکل۶-۱۲: تنظیمات مربوط به database

۶-۵-۱ مدل‌ها

در این پروژه، مدل‌های مختلفی برای ذخیره داده‌های مربوط به پویش‌ها، زیر-پویش‌ها و جزئیات آسیب‌پذیری‌ها تعریف شده‌است. هر مدل به عنوان یک جدول در پایگاه داده SQLite عمل می‌کند. در اینجا به چند نمونه از این مدل‌ها اشاره می‌شود:

- مدل Scan: این مدل، اطلاعات مربوط به یک پویش کامل را ذخیره می‌کند که شامل کاربر اجراکننده پویش، نشانی اینترنتی پویش شده، تاریخ‌های پویش و امتیاز امنیتی است. همچنین وضعیت پویش (ناموفق، کامل، درحال اجرا) نیز در این مدل ذخیره می‌شود. در جدول زیر مقدارهای بکار رفته در این مدل بررسی می‌شود.

- مدل SubScan: این مدل جزئیات مربوط به زیر-پویش‌های انجام شده در طی یک پویش را ذخیره می‌کند. برای هر نشانی اینترنتی جستجو شده، یک رکورد در این مدل ایجاد می‌شود که شامل عمق جستجو، امتیاز امنیتی و اطلاعات آسیب‌پذیری‌هاست.

فیلد	نوع داده	توضیحات
user	ForeignKey (User)	کاربری که پویش را انجام داده است
url	URLField	آدرس سایت برای پویش
scan_date	DateTimeField	تاریخ و زمان اولین پویش
last_scan_date	DateTimeField	آخرین زمانی که پویش انجام شده‌است
is_vulnerable	BooleanField	مشخص می‌کند که آیا سایت آسیب‌پذیری دارد یا خیر
vulnerabilities	TextField	لیست آسیب‌پذیری‌ها پیدا شده در پویش
security_score	IntegerField	امتیاز امنیتی سایت
main_url	URLField	نشانی اینترنتی اصلی سایت در صورت پویش شدن نشانی‌های اینترنتی جانبی.
scan_identifier	CharField	شناسه منحصر به فرد برای پویش.
status	CharField	وضعیت پویش (در حال انجام، کامل شده، یا شکست‌خورده).
Is_stoping	BooleanField	فرآیند توقف را مدیریت می‌کند

جدول۶-۱: جدول مربوط به مدل Scan

- مدل VulnerabilityDetail: این مدل اطلاعات دقیقی از آسیب‌پذیری‌های کشف شده در یک SubScan را ذخیره می‌کند، شامل پارامتر آسیب‌پذیر، نوع آسیب‌پذیری و توضیحات مربوطه.

فیلد	نوع داده	توضیحات
scan	ForeignKey (Scan)	پویش اصلی مرتبط با این زیر پویش
crawled_url	URLField	نشانی اینترنتی که در این زیر پویش خاص پویش شده‌است
depth	IntegerField	عمق پویش در ارتباط با نشانی‌های اینترنتی وابسته
scan_date	DateTimeField	تاریخ و زمان انجام این زیر پویش.
is_vulnerable	BooleanField	آیا این نشانی اینترنتی خاص آسیب‌پذیر است یا خیر
vulnerabilities	TextField	لیست آسیب‌پذیری‌ها پیدا شده دراین زیر پویش
security_score	IntegerField	امتیاز امنیتی این زیر پویش.

جدول۶-۲: جدول مربوط به مدل SubScan

فیلد	نوع داده	توضیحات
sub_scan	ForeignKey (SubScan)	زیر پویش مرتبط با این جزئیات آسیب‌پذیری
parameter	CharField	پارامتری که در آن آسیب‌پذیری پیدا شده‌است (مثلاً متغیرهای نشانی اینترنتی)
payload	TextField	داده‌هایی که مهاجم برای بهره‌برداری از آسیب‌پذیری استفاده کرده است
description	TextField	توضیح در مورد نوع آسیب‌پذیری و نحوه بروز آن
vuln_type	CharField	نوع آسیب‌پذیری (XSS و SQL Injection)
type	CharField	نوع دقیق‌تر آسیب‌پذیری (در صورت مشخص بودن)

جدول۶-۳: جدول مربوط به مدل VulnerabilityDetail

۶-۵-۲ استفاده از عملیات CRUD

هر مدل در جنگو، به عنوان یک رابط مستقیم برای جداول پایگاه داده عمل می‌کند و این امکان را فراهم می‌کند تا عملیات ایجاد[۱]، خواندن[۲]، بروزرسانی[۳] و حذف [۴]را به سادگی بر روی رکوردهای پایگاه داده انجام شود.

۶-۶ طراحی و پیاده‌سازیAPI ها

یک واسط است که به نرم‌افزارهای مختلف اجازه می‌دهد با همدیگر ارتباط برقرار کنند. این ارتباط می‌تواند بین دو سیستم متفاوت، دو سرویس، یا حتی بخش‌های مختلف یک نرم‌افزار باشد. API به توسعه‌دهندگان این امکان را می‌دهند که بدون نیاز به دانستن جزئیات داخلی یک سیستم، از قابلیت‌های آن استفاده کنند. در پروژه‌های مدرن، APIها نقش کلیدی در یکپارچه‌سازی سرویس‌ها و ارائه داده‌ها به صورت RESTful یا GraphQL ایفا می‌کنند. توسعه‌دهندگان به راحتی می‌توانند بخش‌های مختلف نرم‌افزار را جداگانه توسعه بدهند و سپس به صورت متمرکز از آن‌ها استفاده کنند. APIها انواع مختلفی دارند:

- RESTful API: یک معماری است که برای طراحی APIهای ساده و قابل توسعه استفاده می‌شود. APIهای RESTful از پروتکل HTTP برای ارسال و دریافت داده‌ها استفاده می‌کنند. عملیات‌های اصلی در REST شامل دریافت[۵]، ارسال[۶]، به‌روزرسانی[۷] و حذف[۸] است که به ترتیب برای خواندن، ایجاد، بروزرسانی و حذف داده‌ها استفاده می‌شوند.

- GraphQL API: یک زبان پرس‌وجوی داده است که به کلاینت‌ها اجازه می‌دهد دقیقا داده‌های مورد نیاز خود را درخواست کنند. این روش در مقایسه با REST انعطاف‌پذیری بیشتری دارد. در پروژه‌های

[1] Create
[2] Read
[3] Update
[4] Delete
[5] Get
[6] Post
[7] Put
[8] Delete

پیچیده که نیاز به تعاملات گسترده بین مشتری و سرور وجود دارد، GraphQL می‌تواند گزینه مناسبی باشد.

- WebSocket API: یک پروتکل ارتباطی است که امکان برقراری ارتباط دوطرفه و هم‌زمان بین مشتری و سرور را فراهم می‌کند. این نوع API بیشتر در برنامه‌های بلادرنگ[1] مثل گفتگو یا بازی‌های برخط استفاده می‌شود.

در این پروژه، از DRF[2] برای پیاده‌سازی APIها استفاده شده‌است. سریال‌سازها[3] نقش مهمی در این فرآیند دارند، چرا که آن‌ها داده‌های ورودی و خروجی API را به قالب‌های قابل استفاده (مانند JSON) تبدیل می‌کنند. در DRF، سریال‌سازها به طور خودکار اعتبارسنجی داده‌های ورودی را انجام می‌دهند. می‌توانید قوانین اعتبارسنجی سفارشی نیز تعریف کنید. با استفاده از DRF می‌توان آزمون‌های واحد برای APIها را نوشت. این آزمون‌ها باید شامل آزمون عملکرد صحیح هر نقطه پایانی[4] و همچنین تست‌های مربوط به احراز هویت و اعتبارسنجی داده‌ها باشند.

در بالا به نقطه پایانی‌ها اشاره شد. نقاط پایانی در API به نشانی اینترنتی‌هایی اشاره دارند که کاربران (یا سیستم‌های دیگر) برای ارتباط با سرور به آن‌ها درخواست ارسال می‌کنند. این نقاط پایانی معمولاً عملیات خاصی را انجام می‌دهند، مانند ایجاد، خواندن، به‌روزرسانی یا حذف داده‌ها. هر نقطه پایانی یک نشانی اینترنتی منحصر به فرد دارد که به یک منبع خاص در سیستم مرتبط است و معمولاً با یکی از روش‌های HTTP (مانند دریافت،

[1] Real-Time
[2] Django REST Framework
[3] Serializers
[4] Endpoint

ارسال، به‌روزرسانی، حذف) ترکیب می‌شود. در پروژه، چندین نقطه پایانی تعریف شده‌است که به کاربران امکان انجام عملیات مختلف بر روی داده‌ها را می‌دهد:

- URL: /api/scan: این نقطه پایانی که با روش ارسال کار می‌کند، به کاربران اجازه می‌دهد تا یک نشانی اینترنتی جدید برای پویش ثبت کنند. سریال‌سازها ScanSerializer اطلاعات مربوط به نشانی اینترنتی و سایر متغیرها را دریافت می‌کند و یک پویش جدید ایجاد می‌کند یا اگر قبلاً نشانی اینترنتی ثبت شده‌است، اطلاعات آن را به کاربر برمی‌گرداند.

- URL: /api/scan/<id>: این نقطه پایانی که با روش دریافت کار می‌کند، به کاربران اجازه می‌دهد تا نتایج یک پویش خاص را بر اساس شناسه[1] آن دریافت کنند. داده‌هایی مانند نشانی اینترنتی ، تاریخ پویش، آسیب‌پذیری‌های کشف‌شده و امتیاز امنیتی از طریق این نقطه پایانی به کاربر برگردانده می‌شود.

[1] ID

منابع و مراجع

1. https://www.sonarsource.com/solutions/security/owasp
2. https://owasp.org/www-community/attacks/xss/
3. https://owasp.org/www-community/attacks/SQL_Injection/
4. https://icons۸.com/icons
5. https://www.acunetix.com
6. https://www.acunetix.com/websitesecurity/csrf-attacks/
7. https://www.w3schools.com/REACT/DEFAULT.ASP
8. https://www.django-rest-framework.org/
9. https://www.kaggle.com/datasets/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning
10. https://www.kaggle.com/datasets/sajid576/sql-injection-dataset
11. https://www.w3schools.com/sql/sql_injection.asp
12. https://www.logoai.com/
13. https://www.w3schools.com/w3css/defaulT.asp

فهرست واژگان

English	فارسی
Vulnerability	آسیب‌پذیری
API	رابط برنامه‌نویسی کاربردی
Frontend	رابط‌کاربری (سمت کاربر)
Backend	سمت سرور
Database	پایگاه داده
Penetration Testing	آزمون نفوذ
Django	جنگو (چارچوب وب برای پایتون)
Confidentiality	محرمانگی
Integrity	صحت
Availability	در دسترس بودن
Payload	بارگذاری داده
Access Token	نشانه دسترسی
Authentication	احراز هویت
React	یک کتابخانه جاوااسکریپت برای ساخت رابط‌کاربری
Security Score	امتیاز امنیتی
SQLite	اس‌کیوال‌لایت(یکی از انواع پایگاه داده)
Celery	سیستم صف‌بندی وظایف توزیع‌شده

Abstract

In today's digital world, the security of web-based systems has become one of the critical issues for organizations and businesses. Vulnerabilities such as SQL Injection and XSS can lead to security breaches and unauthorized access to sensitive data. The VulneraXSQL project is designed to provide a comprehensive solution for identifying and managing these vulnerabilities. This tool utilizes advanced methods and modern algorithms to detect vulnerabilities in web-based systems and provides detailed reports and actionable insights about security weaknesses. VulneraXSQL includes the design and implementation of efficient APIs, the creation of a simple and user-friendly interface using React, and the design of a database with SQLite. This tool helps organizations and developers analyze and address the identified vulnerabilities to enhance the security of their web systems and prevent potential attacks. Additionally, VulneraXSQL serves as an educational tool to familiarize users with security vulnerabilities and the methods for addressing them. The project also provides a foundation for adding other types of attacks in the future. Some of the key challenges of the project include identifying malicious web systems, managing data volume, and ensuring the accuracy of results. The ultimate goal of this project is to improve the security level of web systems and facilitate the vulnerability detection process in an effective and fast manner.

Keywords: Website vulnerability, SQL Injection, XSS, Web security, Vulnerability detection tool, SQLite

Faculty of Engineering

Department of Computer Science

Bachelor’s in Software Engineering

Design and Development of a Vulnerability Scanning Tool

for

SQL Injection and XSS

Supervisor:

Dr. Asghar Tajoddin

Prepared by:

Ayda Alimohammadi & Negar Rezaei

Summer 2024