کدنویسی مطمئن، تکنیک‌های امنیت داده برای برنامه‌نویسان وب

این کتاب شما را با تکنیک‌های اساسی امنیتی آشنا می‌کند که هر برنامه‌نویسی برای محافظت از برنامه‌های خود نیاز دارد. با مطالعه آن، نه‌تنها امنیت دیجیتالی خود را تقویت می‌کنید، بلکه اعتماد کاربران را نیز جلب می‌نمایید.

نویسنده احسان بابائی

برای دسترسی به مطالب آموزشی بیشتر، مخصوص برنامه نویسان .NET و C# وب‌سایت و شبکه های اجتماعی ما را دنبال کنید

https://bugeto.net

https://t.me/bugeto

http://instagram.com/bugeto_net

زمستان 1402

فهرست مطالب

مقدمه

در دنیای فناوری و برنامه‌نویسی امروز، که اطلاعات به سرعت در حال جابجایی هستند، امنیت باید در صدر هر تصمیمی قرار بگیرد. برای یک برنامه‌نویس، دانش امنیت داده‌ها فراتر از یک مهارت است؛ یک ضرورت است. اما، به راستی چگونه می‌توانیم این امنیت را در اپلیکیشن‌هایی که می‌سازیم، تضمین کنیم؟

من، در این میکرو کتاب به شما نشان می‌دهم که چگونه می‌توانید با اجرای چهار روش اولیه و کلیدی، اپلیکیشن‌های خود را ایمن‌تر کنید. این کتاب برای برنامه‌نویسانی نوشته شده که می‌خواهند امنیت داده‌ها را در کار خود به سطح بالاتری ببرند.

امنیت یک مبحث گسترده است، اما در این کتاب تمرکز من بر روی ارائه‌ی حداقل‌هایی است که هر برنامه‌نویس وب باید بداند و از آنها در پروژه های خود استفاده کند. شگفت‌زده خواهید شد اگر بدانید بسیاری از برنامه‌نویسان، حتی آن‌هایی که سابقه‌ی کاری طولانی دارند، هنوز با این روش‌ها آشنا نیستند.

هدف من از نگارش این کتاب، آشنا کردن شما با روش‌هایی است که به طور موثری امنیت اطلاعات کاربران را در برابر تهدیدات سایبری محافظت می‌کند. با مطالعه‌ی این کتاب، شما با روش‌هایی برای رمزگذاری داده‌ها، هش کردن پسوردها، تقویت آن‌ها با استفاده ازSalt ، و به کارگیری توکن‌های احراز هویت مدرن آشنا می‌شوید.

پس از خواندن این کتاب، دعوتتان می‌کنم به وب‌سایت ما به نشانی Bugeto.net سری زده و از مطالب آموزشی بیشتری که به صورت مرتب به‌روز می‌شوند، بهره‌مند شوید.

با احترام، احسان بابائی

Data Encryption

رمزگذاری چگونه میتواند داده‌ها را در برابر دزدان دیجیتالی محافظت کند؟

فرض کنید که در حال توسعه یک وبسایت هستید که اطلاعات متنوعی از کاربران نگهداری می‌کند؛ از جمله اطلاعات شخصی مانند نام، نام خانوادگی، آدرس‌ها، تاریخچه سفارش‌ها و اطلاعات دستگاه‌های کاربران. در صورتی که دیتابیس شما توسط هکرها مورد حمله قرار بگیرد، چه تهدیداتی ایجاد می‌شود؟

تهدیدها

۱. **سرقت هویت** :هکرها می‌توانند از اطلاعات شخصی کاربران سوءاستفاده های انجام دهند.

۲. **کلاهبرداری مالی** :دسترسی به اطلاعات مالی کاربران می‌تواند به کلاهبرداری‌های مالی منجر شود.

۳. **نقض حریم خصوصی** :فاش شدن آدرس‌ها و تاریخچه سفارش‌ها حریم خصوصی کاربران را زیر پا می‌گذارد.

۴. **نفوذ به دستگاه‌ها** :دسترسی به اطلاعات دستگاه‌های کاربران می‌تواند به هکرها اجازه دهد تا به دستگاه‌های کاربران نفوذ کنند.

راه حل، رمزگذاری داده‌ها

برای مقابله با این تهدیدات، رمزگذاری داده‌های حساس یک راه حل موثر است. رمزگذاری به این معنی است که حتی اگر هکرها به داده‌ها دسترسی پیدا کنند، نمی‌توانند آنها را بخوانند یا از آنها سوءاستفاده کنند. این امر به دلیل رمزگذاری شدن داده‌ها و نیاز به کلید رمزگشایی برای خواندن آنها است. بنابراین، حتی در صورت نفوذ به دیتابیس، اطلاعات کاربران در امان باقی می‌مانند.

فرایند رمزگذاری داده‌ها

مرحله ۱: ثبت اطلاعات

فرض کنید کاربری در اپلیکیشن شما آدرس خود را وارد می‌کند. شما این اطلاعات را دریافت می‌کنید و برای تضمین امنیت آن‌ها، تصمیم به رمزگذاری می‌گیرید.

مرحله ۲: رمزگذاری

قبل از ذخیره‌سازی داده‌ها در دیتابیس، آن‌ها را با استفاده از یک کلید رمزگذاری تبدیل به رشته‌های رمزگذاری شده می‌کنید. این کلید مانند یک رمز مخفی است که فقط شما به آن دسترسی دارید.

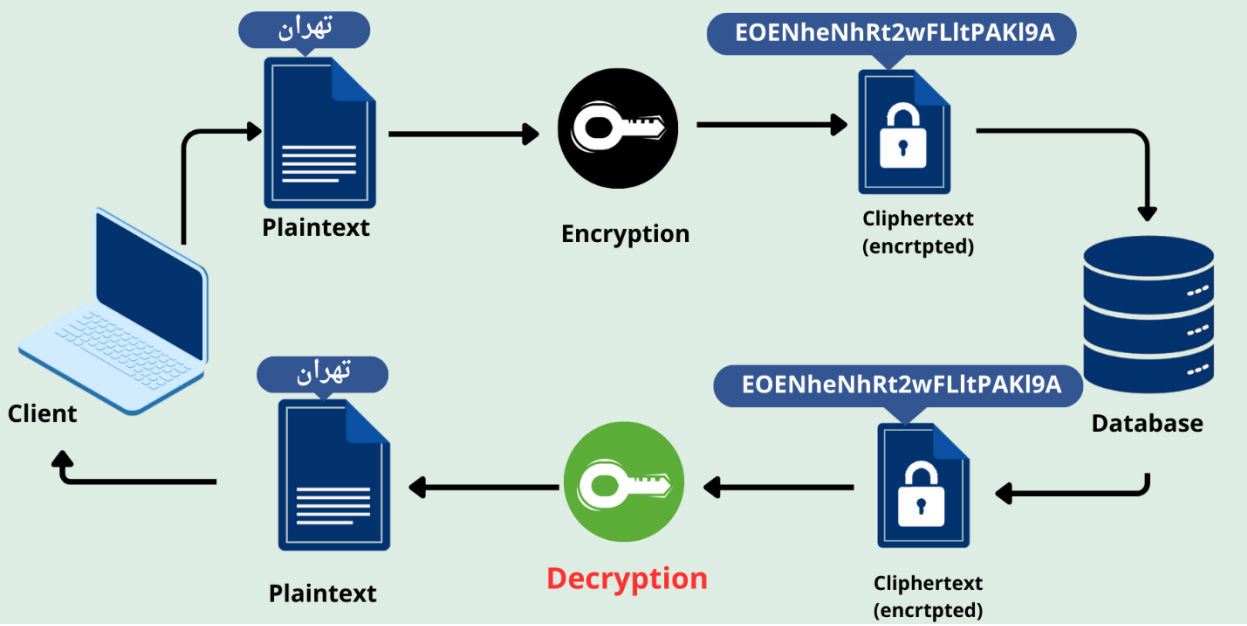
مرحله ۳: ذخیره‌سازی داده‌های رمزگذاری شده

شما نسخه رمزگذاری شده آدرس‌ها را در دیتابیس ذخیره می‌کنید، تا از خوانده شدن آن‌ها توسط دسترسی‌های غیرمجاز جلوگیری کنید.و حالا اگر کس جدول آدرس‌ها در دیتابیس شمارا مشاهده کند فقط داده های رمزگذاری شده را مشاهده می کند.

مرحله ۴: رمزگشایی

حالا هروقت نیاز به اصل اطلاعات داشتید، مانند مشاهده آدرس توسط کاربر در اپلیکیشن، شما با استفاده از همان کلید رمزگذاری، داده‌ها را به فرمت اصلی بازمی‌گردانید و کاربران می توانند اطلاعات اصلی را مشاهده کنند.

این روش اطمینان می‌دهد که حتی در صورت دسترسی غیرمجاز به داده‌ها، محتوای واقعی آن‌ها خوانده نشود و امنیت اطلاعات کاربران حفظ گردد.



تصویر ۱: فرایند رمز نگاری و رمزگشایی داده ها

حفاظت از داده‌ها در برابر هکرها

حالا اگر یک گروه هکری دسترسی به دیتابیش شما داشته باشد برای مشاده محتوای اصلی داده ها چیکار می تواند انجام دهد؟ هکر برای خواندن داده‌ها نیاز به کلید رمزگشایی دارد. بدون این کلید، دسترسی به محتوای رمزگذاری شده تقریباً غیرممکن است. اما با این حال هکر میتواند با انجام موارد زیر به داده ها دسترسی داشته باشد که کار بسیار دشواری است و در بسیاری از موارد غیر ممکن.

۱. تلاش برای شکستن رمزگذاری

هکرها ممکن است سعی کنند با استفاده از حملات **Brute Force** کلید را حدس بزنند. این روش زمان‌بر و اغلب در مقابل رمزگذاری‌های پیچیده ناموفق است.

کشف الگوهای احتمالی در داده‌های رمزگذاری شده که بسته به قدرت رمزگذاری، دشوار است.

۲ .بهره‌گیری از آسیب‌پذیری‌ها

وجود آسیب‌پذیری‌ها در سیستم رمزگذاری، مانند نقص در نرم‌افزار یا الگوریتم‌های رمزگذاری، می‌تواند راهی برای دسترسی به داده‌ها باشد.

۳ .دسترسی به اطلاعات جانبی

گاهی هکرها می‌توانند از طریق دسترسی به اطلاعات یا سیستم‌های دیگر به کلیدهای رمزگذاری دست یابند.

پس در نتیجه می توان گفت که حتی اگر هکر بتواند به داده های شما دسترسی داشته باشد،بدون دسترسی به کلید رمزگذاری یا بدون وجود آسیب‌پذیری‌های خاص، نمی تواند داده های شما را بخواند. این اهمیت حفظ امنیت کلیدهای رمزگذاری و استفاده از الگوریتم‌های رمزگذاری قوی و به‌روز را نشان می‌دهد.

انتخاب الگوریتم‌های رمزگذاری برای افزایش امنیت:

برای محافظت از داده‌ها در برابر دسترسی‌های غیرمجاز و تقویت امنیت داده‌های رمزگذاری شده، استفاده از الگوریتم‌های رمزگذاری قوی و به‌روز ضروری است. در اینجا چند مثال از الگوریتم‌های مطرح و معتبر رمزگذاری آورده شده است:

AES (Advanced Encryption Standard)

- AESیکی از پرکاربردترین استانداردهای رمزگذاری است که به دلیل قدرت و سرعت بالا در بسیاری از برنامه‌ها و سیستم‌های امنیتی استفاده می‌شود.
- این الگوریتم با طول کلیدهای 128، 192، و 256 بیتی ارائه شده و قابلیت مقابله با بیشتر انواع حملات را دارد.

RSA (Rivest–Shamir–Adleman)

- RSAیک الگوریتم رمزگذاری کلید عمومی است که برای امنیت در انتقال داده‌ها و امضای دیجیتال استفاده می‌شود.
- این الگوریتم بر مبنای مسئله سخت ریاضی فاکتورگیری از اعداد بزرگ استوار است و با افزایش طول کلید، امنیت آن افزایش می‌یابد.

ECC (Elliptic Curve Cryptography)

- ECCیک روش رمزگذاری کلید عمومی است که در آن از خواص ریاضی منحنی‌های بیضوی استفاده می‌شود.
- این الگوریتم با استفاده از کلیدهای کوچکتر، امنیت برابر یا بالاتری نسبت به الگوریتم‌های دیگر مانند RSA ارائه می‌دهد و کارآمدتر است.

با انتخاب الگوریتم‌های رمزگذاری مناسب و به‌روز، می‌توانید امنیت داده‌های رمزگذاری شده را به طور چشمگیری افزایش دهید و از آن‌ها در برابر دسترسی‌های غیرمجاز محافظت کنید.

نکات مهم برای ایجاد و نگهداری کلید ها

در الگوریتم‌های رمزگذاری، کلید نقش مهمی دارد. کلید همان عنصری است که برای رمزگذاری و رمزگشایی داده‌ها به کار می‌رود. به عبارت دیگر، کلید مانند رمز یا کدی است که امنیت داده‌های رمزگذاری شده را تضمین می‌کند. حالا به نکات مهم در مورد ایجاد و نگهداری این کلیدها می‌پردازیم:

ایجاد کلیدها

۱. **استفاده از الگوریتم‌های معتبر** :برای تولید کلیدها، باید از الگوریتم‌های رمزگذاری معتبر و قابل اعتماد استفاده کنید. این کار تضمین می‌کند که کلیدهای شما قوی و امن هستند.

۲. **طول کلید** :اطمینان حاصل کنید که طول کلید کافی باشد تا از امنیت بالایی برخوردار باشد. برای مثال، کلیدهای 128 بیتی یا بلندتر معمولاً برای اکثر کاربردها مناسب هستند.

نگهداری کلیدها

۱. **ذخیره‌سازی امن** :کلیدهای رمزگذاری باید در مکان‌های امن و قابل اعتماد ذخیره شوند. این می‌تواند شامل سیستم‌های مدیریت کلید امنیتی (Key Management Systems) یا دستگاه‌های سخت‌افزاری امن باشد.

۲. **جداسازی و کنترل دسترسی** :کلیدها نباید در همان مکانی که داده‌های رمزگذاری شده ذخیره می‌شوند، نگهداری شوند. همچنین، دسترسی به کلیدها باید محدود و تحت کنترل باشد.

گم کردن کلید

- اگر کلید رمزگذاری گم شود، دسترسی به داده‌های رمزگذاری شده بدونه آن کلید برای خودتان هم امکان پذیر نیست. بنابراین، بسیار مهم است که از کلیدها به درستی نگهداری شود و پشتیبان‌گیری امن از آنها انجام گیرد.

نکته مهم

- **پشتیبان‌گیری کلیدها** :حتماً باید از کلیدهای خود پشتیبان‌گیری کنید و این پشتیبان‌ها را در مکان‌هایی متفاوت و امن نگهداری کنید تا در صورت گم شدن یا آسیب دیدن کلید اصلی، بتوانید به داده‌های رمزگذاری شده دسترسی پیدا کنید.

به خاطر داشته باشید که مدیریت کلیدهای رمزگذاری یکی از مهم‌ترین جنبه‌های تأمین امنیت داده‌ها است و باید با دقت و احتیاط انجام شود.

امکانات .NET. برای رمزگذاری داده‌ها

دات نت (.NET) امکانات گسترده‌ای را برای رمزگذاری داده‌ها در اختیار برنامه‌نویسان قرار می‌دهد. این امکانات شامل کتابخانه‌ها و ابزارهایی برای ایجاد کلیدها، استفاده از الگوریتم‌های رمزگذاری، مدیریت کلیدها و موارد دیگر است:

ایجاد کلید

- **کلاس های System.Security.Cryptography :**

.NET فضای نام System.Security.Cryptography را با کلاس‌های متنوع برای ایجاد کلیدهای رمزگذاری ارائه می‌دهد. این کلاس‌ها امکان ساخت کلیدهایی امن و قدرتمند را فراهم می‌کنند.

الگوریتم‌های رمزگذاری

- **AES (Advanced Encryption Standard) :** کلاس‌هایی مانند Aes و AesManaged برای پیاده‌سازی رمزگذاری AES در .NET وجود دارند.
- **RSA (Rivest–Shamir–Adleman) :** کلاس RSACryptoServiceProvider برای پیاده‌سازی الگوریتم RSA در .NET استفاده می‌شود.
- ECC (Elliptic Curve Cryptography) : دات نت از رمزگذاری منحنی بیضوی با کلاس‌هایی مانند ECDsaCng پشتیبانی می‌کند.

مدیریت کلیدها

- **Secure Storage** : دات نت امکاناتی برای ذخیره‌سازی امن کلیدها ارائه می‌دهد. برای مثال، می‌توانید از Windows Data Protection API (DPAPI) برای ذخیره‌سازی امن کلیدها استفاده کنید.
- **Cryptography Next Generation (CNG)** : این API به شما امکان می‌دهد تا کلیدهای رمزگذاری را به طور امن مدیریت کنید و از قابلیت‌های پیشرفته‌تر رمزگذاری استفاده نمایید.

ابزارها

- **کلاس ProtectedData** : این کلاس برای رمزگذاری داده‌ها در حافظه یا فایل‌های دیسک استفاده می‌شوند و به شما امکان می‌دهد تا داده‌ها را به صورت امن در حافظه یا دیسک ذخیره کنید.
- **کلاس‌های HashAlgorithm** : برای ایجاد هش‌های امن، دات نت کلاس‌های متعددی را ارائه می‌دهد.

نکته مهم

- **به‌روزرسانی و امنیت** :همیشه مهم است که از آخرین نسخه‌های دات نت استفاده کنید و کتابخانه‌ها و ابزارهای رمزگذاری خود را به‌روز نگه دارید تا از قابلیت‌های امنیتی جدید و رفع اشکالات امنیتی بهره‌مند شوید.

با استفاده از این ابزارها و کلاس‌ها، برنامه‌نویسان دات نت می‌توانند امنیت داده‌ها را در برنامه‌های خود به طور مؤثری تقویت کنند.

چه داده های نیاز به رمزگذاری دارند؟

حالا فرض کنید در حال توسعه یک فروشگاه اینترنتی هستید! چطور باید تصمیم بگیرید کدام داده‌ها را رمزگذاری کنید؟ در یک فروشگاه اینترنتی، و یا هر اپلیکیشن دیگری، تصمیم‌گیری در مورد اینکه کدام داده‌ها باید رمزگذاری شوند و کدام نه، به حساسیت و اهمیت اطلاعات بستگی دارد.

رمزگذاری داده‌های حساس مانند اطلاعات شخصی کاربران یا جزئیات حساب‌های بانکی به حفاظت از حریم خصوصی کاربران و جلوگیری از سوءاستفاده‌های احتمالی کمک می‌کند.

اما برای داده‌های با حساسیت کمتر مانند اطلاعات عمومی محصولات یا داده‌های بازاریابی که برای عملکرد روزمره فروشگاه حیاتی هستند، رمزگذاری نه‌تنها ضروری نیست بلکه ممکن است فرایندهای کسب‌وکار را کند نیز کند.

در ادامه به بررسی داده‌هایی که نیاز به رمزگذاری دارند و داده‌هایی که نیاز به رمزگذاری ندارند می‌پردازیم:

داده‌هایی که نیاز به رمزگذاری دارند:

۱. **اطلاعات شخصی کاربران :**مانند نام کامل، آدرس، شماره تلفن و...

۲. **جزئیات حساب‌های بانکی :**برای حفاظت از اطلاعات مالی کاربران.

داده‌هایی که نیاز به رمزگذاری ندارند:

۱. **اطلاعات عمومی :**مانند نام محصول، توضیحات، قیمت،آمار فروش و ایمیل های بازاریابی و هر اطلاعاتی که ارزشی برای هک شدن ندارند.

خلاصه

رمزگذاری داده‌ها یکی از راهکارهای اصلی برای حفاظت از اطلاعات حساس در برابر هکرها است. در توسعه وب‌سایت‌ها و اپلیکیشن‌ها، تعیین اینکه چه داده‌هایی باید رمزگذاری شوند، براساس میزان حساسیت و اهمیت آن‌ها صورت می‌گیرد. داده‌هایی مانند اطلاعات شخصی و مالی کاربران باید رمزگذاری شوند تا از سرقت هویت، کلاهبرداری‌های مالی، و نقض حریم خصوصی جلوگیری کنند. در مقابل، داده‌های عمومی مانند اطلاعات محصولات نیازی به رمزگذاری ندارند. با استفاده از ابزارها و کتابخانه‌هایNET. ، برنامه‌نویسان می‌توانند از الگوریتم‌های رمزگذاری پیشرفته و مدیریت امن کلیدها برای تقویت امنیت داده‌ها استفاده کنند. این روش‌ها اطمینان می‌دهند که حتی در صورت دسترسی غیرمجاز، داده‌های حساس قابل خواندن نباشند.

Password Hashing

فرض کنید پسوردها در دیتابیس به همان صورتی که کاربر در صفحه ثبت نام وارد کرده است ذخیره شوند؛ هر کسی که به آنها نگاه کند، می‌تواند ببیند و بخواند. هیچ رمزی، هیچ پوششی. این دقیقاً همان چیزی است که به آن **Plain Text** یا **متن ساده** می‌گوییم.

مشکل ذخیره‌سازی پسوردها به صورت Plain Text

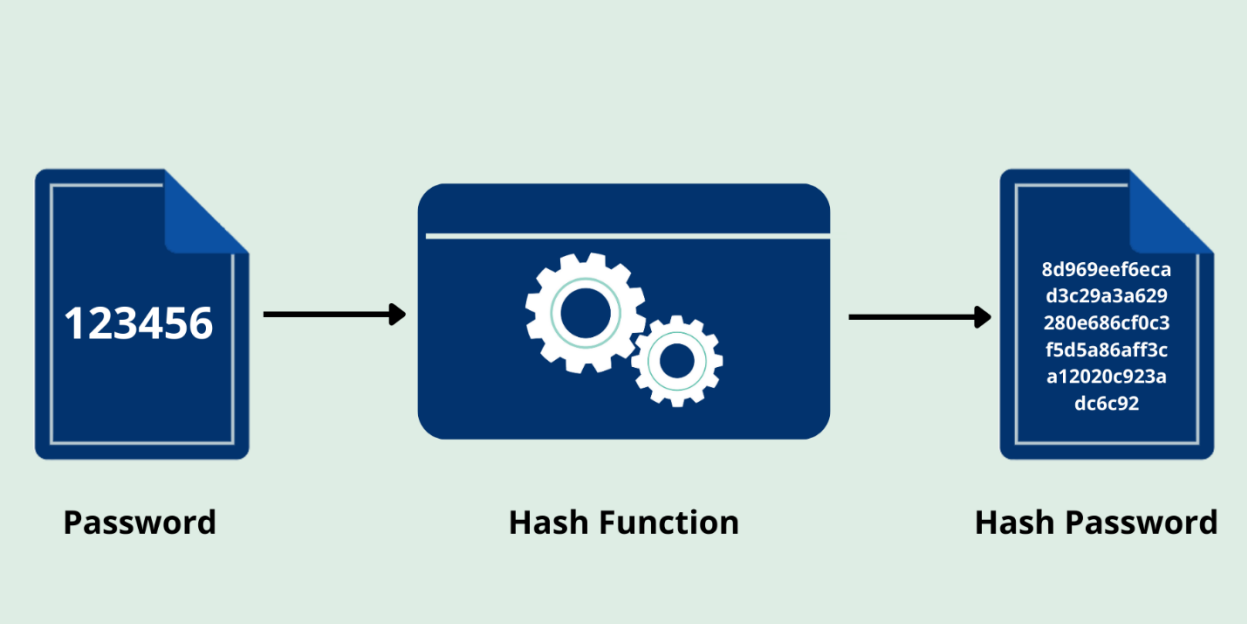
حالا تصور کنید شما یک برنامه‌نویس هستید و تصمیم گرفته‌اید که پسوردهای کاربران‌تان را به این شکل ساده ذخیره کنید. اما یک روز، بدشانسی آورده و دیتابیس شما هک می‌شود!

هکرها می‌توانند به راحتی پسوردهای کاربران شما را ببینند و از آنها برای ورود به حساب‌های دیگر استفاده کنند، مخصوصاً اگر کاربران از یک پسورد برای چندین سایت مهم دیگر هم استفاده کرده باشند.

خبر هک شدن دیتابیس و ذخیره‌سازی پسوردها به صورت **Plain Text** می‌تواند شهرت شما را به عنوان یک توسعه‌دهنده یا یک شرکت نابود کند. کاربران احساس می‌کنند که به آنها خیانت شده است.

راه حل: استفاده از هش

خوب، حالا وقت آن رسیده که به جای **Plain Text**، از **Hash** استفاده کنید. هش یعنی تبدیل پسورد به یک رشته پیچیده و غیرقابل خواندن که حتی اگر کسی به دیتابیس شما دسترسی پیدا کند، نمی‌تواند پسورد اصلی را بفهمد.



تصویر۲ : فرایند هش

هش چطور کار می‌کند؟

هنگامی که کاربر پسورد خود را وارد می‌کند، سیستم به جای ذخیره‌سازی همان پسورد، آن را از طریق یک الگوریتم هش می‌کند و نتیجه هش شده را ذخیره می‌کند. حتی اگر هکرها به این داده‌ها دسترسی پیدا کنند، نمی‌توانند پسورد اصلی را بازیابی کنند.

بنابراین، بیایید با استفاده از هش کردن پسوردها، مطمئن شویم که هکر نمی‌تواند به گنجینه اطلاعات حساس ما دسترسی پیدا کند. این یک قدم بزرگ برای حفاظت از اطلاعات کاربران و حفظ اعتبار بیزینس شما است.

آشنایی با Hashing

بیایید مفهوم هش را با یک مثال ساده از دنیای واقعی بررسی کنیم، فرض کنید شما تعدادی هویج دارید و آنها را درون یک دستگاه آبمیوه‌گیری می‌ریزید تا تبدیل به آب هویج شوند. پس از این فرآیند، دیگر به هیچ وجه نمی‌توانید آب هویج را به هویج‌های اولیه‌ای که وارد دستگاه کرده بودید، تبدیل کنید.

این مثال دقیقا نشان‌دهنده فرآیند هش کردن در دنیای کامپیوتر است. وقتی یک داده، مثلاً یک پسورد، را هش می‌کنیم، این داده تبدیل به یک رشته پیچیده و غیرقابل بازگشت می‌شود، دقیقا مانند تبدیل هویج به آب هویج. این رشته هش شده نمی‌تواند به شکل اصلی خود بازگردد.

این موضوع، دقیقاً برعکس رمزگذاری است. در رمزگذاری، وقتی از یک کلید برای رمزگذاری داده‌ها استفاده می‌کنیم، می‌توانیم با همان کلید، داده‌های رمزگذاری شده را به حالت اولیه‌شان برگردانیم. اما در هش کردن، چنین امکانی وجود ندارد. همین ویژگی هش کردن را به یک ابزار امنیتی قدرتمند در محافظت از پسوردها تبدیل می‌کند.

پسورد اولیه	مقدار هش شده
1234	81dc9bdb52d04dc20036dbd8313ed055
password	5f4dcc3b5aa765d61d8327deb882cf99
hello	5d41402abc4b2a76b9719d911017c592
test123	cc03e747a6afbbcbf8be7668acfebee5
user	ee11cbb19052e40b07aac0ca060c23ee

جدول 1 : نمونه‌هایی از پسوردهای ساده و نتایج هش شده آن‌ها با استفاده از الگوریتم MD5 را نشان می‌دهد.

فرآیند هش کردن چگونه کار می‌کند؟

هش کردن فرآیندی است که در آن داده‌ها، مثلاً یک پسورد، از طریق یک الگوریتم هش به یک رشته پیچیده و طولانی از کاراکترها تبدیل می‌شوند. فکر کنید مانند ریختن میوه‌ها در دستگاه خردکن و خرد کردن آن‌ها به تکه‌های ریز است؛ بعد از آن نمی‌توانید میوه‌ها را به حالت اولیه بازگردانید. همین اتفاق در هش کردن می‌افتد، نمی‌توانید از هش به پسورد اصلی برگردید.

علاوه بر این، اگر دو پسورد حتی کمی با هم متفاوت باشند، هش‌های کاملاً متفاوتی تولید می‌شوند. بنابراین، هر پسورد یک هش منحصربه‌فرد دارد که همچون اثر انگشت منحصر به فرد آن است.

چرا از هش استفاده می‌کنیم؟

استفاده از هش به این دلیل است که حتی اگر داده‌های هک شوند، کسی نمی‌تواند به راحتی به پسوردهای اصلی دسترسی پیدا کند. این روش یک سد دفاعی قوی در برابر حملات سایبری و سرقت اطلاعات فراهم می‌آورد.

هکرها باهوش‌تر از آنند که فکر می‌کنید

حتی اگر از هش برای ذخیره‌سازی پسوردها استفاده کنید، هکرها هنوز روش‌هایی دارند تا به پسوردهای اصلی دست یابند. در اینجا روش‌هایی که هکرها ممکن است استفاده کنند و راه‌های مقابله با آنها را بررسی می‌کنیم.

روش‌های دستیابی هکرها به پسوردهای هش شده

۱. **(Brute Force Attacks)**هکرها ممکن است تلاش کنند تا با امتحان کردن ترکیبات مختلف پسورد، هش متناظر را پیدا کنند.

- **روش کار :**در این حمله، هکر تلاش می‌کند همه ترکیبات ممکن پسورد را امتحان کند. مثلاً اگر پسورد شما سه حرفی است، هکر از aaa شروع کرده و تا zzz ادامه می‌دهد تا هش متناظر با هر ترکیب را با هش‌های ذخیره شده در دیتابیس مقایسه کند.
- **نکته:** در حملاتBrute Force ، هکرها تمام ترکیبات ممکن را برای یافتن پسورد امتحان می‌کنند.برای این کار باید زمان زیادی صرف شود و هرچه پسورد طولانی‌تر و پیچیده‌تر باشد، شانس موفقیت هکرها کمتر می‌شود.

۲. **Dictionary Attacks** : استفاده از لیست‌های بزرگی از پسوردهای رایج و هش آنها برای یافتن تطابقات در دیتابیس هش شده.

- **روش کار :**در این روش، به جای امتحان کردن هر ترکیب ممکن، هکر از یک لیست بزرگ از پسوردهای رایج و هش‌های آنها استفاده می‌کند. این لیست ممکن است شامل هزاران یا حتی میلیون‌ها پسورد رایج باشد. هکر هش هر یک از این پسوردها را با هش‌های موجود در دیتابیس مقایسه می‌کند.
- **نکته :**این روش برای پسوردهای ساده و رایج مثل 123456 و امثال آن موثر است اما در برابر پسوردهای پیچیده و منحصربه‌فرد کارآمد نیست.

پسورد	پسورد	پسورد	پسورد
123456	1234	password	carmen
12345	qwerty	password1	admin
password	money	123456789	secret
password1	carmen	12345678	summer
123456789	mickey	1234567890	internet

جدول ۲ – مثالی پسوردهای رایج

در لینک زیر میتوانید لیست پسوردهای رایج را دانلود نمایید.

https://github.com/CTzatzakis/Wordlists/blob/master/password.list

۳. **Rainbow Tables** : جداول رنگین کمان، جداول بزرگی که ترکیبات مختلف پسورد و هش‌های مربوط به آنها را ذخیره کرده‌اند.

- **روش کار :**رینبو تیبلز جداول پیچیده‌ای هستند که ترکیبات مختلف پسورد و هش‌های مربوط به آنها را ذخیره کرده‌اند. در این روش، هکر به جای اینکه خودش هش‌ها را محاسبه کند، از این جداول برای پیدا کردن هش متناظر با پسورد در دیتابیس استفاده می‌کند.
- **نکته :**رینبو تیبلز برای پسوردهای بدون Salt اثربخش هستند. استفاده از Salt در هش‌ها این روش را غیرموثر می‌کند.

به خاطر داشته باشید که هر چقدر پسوردها طولانی‌تر و پیچیده‌تر باشند، این حملات کم‌اثرتر خواهند بود. همچنین، استفاده از Salt در هش کردن پسوردها می‌تواند به طور چشمگیری از اثربخشی این حملات بکاهد.

برای آشنایی بیشتر با رینبو تیبلز (Rainbow Tables) و دانلود نمونه‌های آن، می‌توانید از لینک زیر استفاده کنید.https://freerainbowtables.com/

راه‌های مقابله با این حملات

۱. **استفاده ازSalt** : نمک زدن یا همان Salt یک ترفند ساده اما مؤثر برای افزایش امنیت پسوردهاست. تصور کنید که قبل از هش کردن یک پسورد، یک رشته تصادفی به آن اضافه می‌کنیم. این رشته تصادفی، که Salt نامیده می‌شود، هر بار منحصر به فرد است و باعث می‌شود که حتی اگر دو نفر یک پسورد یکسان داشته باشند، هش نهایی آن‌ها متفاوت باشد. به این ترتیب، حتی اگر هکرها به هش‌ها دسترسی پیدا کنند، نمی‌توانند به راحتی پسوردهای اصلی را حدس بزنند، چرا که Salt به هر پسورد یک ویژگی منحصر به فرد اضافه می‌کند.

۲. **به‌روز نگه داشتن الگوریتم‌های هش** :استفاده از آخرین و قوی‌ترین الگوریتم‌های هش می‌تواند در برابر حملات **Brute Force Attacks** و **Dictionary Attacks** مقاومت بیشتری داشته باشد.

۳. **محدودیت تعداد تلاش‌های ورود** :اعمال محدودیت برای تعداد دفعات تلاش برای ورود ناموفق می‌تواند از **Brute Force Attacks** جلوگیری کند.

۴. **استفاده از پیچیدگی‌های پسورد** :ترغیب کاربران به استفاده از پسوردهای پیچیده و طولانی که شامل ترکیبی از حروف بزرگ و کوچک، اعداد و نمادها هستند.

با این روش‌ها، می‌توانید از اطلاعات کاربران در برابر هکرهای باهوش و حملات آنها محافظت بهتری انجام دهید.

معرفی الگوریتم‌های هش

در این بخش به معرفی چند الگوریتم هش پرکاربرد می‌پردازیم:

SHA-256 (Secure Hash Algorithm 256 bit)

این الگوریتم یکی از محبوب‌ترین‌ها در دنیای امنیت است SHA-256 .بخشی از خانواده SHA-2 است و یک هش 256 بیتی ایجاد می‌کند. این الگوریتم برای امنیت بالا و تولید هش‌های یکتا شناخته شده است.

MD5 (Message Digest Algorithm 5)

MD5یکی از قدیمی‌ترین الگوریتم‌های هش است که هش‌های 128 بیتی تولید می‌کند. با اینکه سرعت و عملکرد بالایی دارد، اما به دلیل آسیب‌پذیری‌های امنیتی در برابر حملات مختلف، امروزه کمتر توصیه می‌شود.

SHA-1 (Secure Hash Algorithm 1)

SHA-1هش‌های 160 بیتی ایجاد می‌کند و برای مدت‌ها استفاده می شد. با این حال، مانندMD5 ، SHA-1نیز به دلیل آسیب‌پذیری‌های امنیتی در برابر حملات خاص، دیگر به عنوان یک الگوریتم امن توصیه نمی‌شود.

SHA-3 (Secure Hash Algorithm 3)

جدیدترین عضو خانوادهSHA ، SHA-3 هش‌هایی با اندازه‌های مختلف از 224 تا 512 بیت ایجاد می‌کند. این الگوریتم برای پاسخ به نیازهای امنیتی مدرن و مقاومت در برابر حملات متداول طراحی شده است.

bcrypt

برای امنیت بیشتر، bcryptاز Salt استفاده می‌کند تا از حملات رینبو تیبل و فرهنگ لغت جلوگیری کند. این الگوریتم برای ذخیره امن پسوردها در دیتابیس‌ها مناسب است.

هر کدام از این الگوریتم‌ها ویژگی‌ها و کاربردهای خاص خود را دارند و انتخاب بین آنها باید بر اساس نیازهای خاص امنیتی و محیط کاربردی انجام شود.

نمونه کد:

در زبان برنامه‌نویسی سی شارپ، با استفاده از الگوریتم‌های SHA512 وSHA256 ، رشته "Bugeto" را هش می‌کنیم و خروجی زیر را دریافت می‌کنیم:


```
using System.Security.Cryptography;
using System.Text;
string original = "Bugeto"; // اولیه
using (SHA512 sha512Hash = SHA512.Create())
{
    // تبدیل رشته به بایت و هش کردن آن
    byte[] sourceBytes = Encoding.UTF8.GetBytes(original);
    byte[] hashBytes = sha512Hash.ComputeHash(sourceBytes);

    //  تبدیل بایتهای هش شده به رشته هگزادسیمالتبدیل
    string hash = BitConverter.ToString(hashBytes).Replace("-", String.Empty);


    Console.WriteLine($"The SHA512 hash of {original} is: {hash}");
}
Console.WriteLine("---------------");
using (SHA256 sha256Hash = SHA256.Create())
{
    // رشته به بایت و هش کردن آن
    byte[] sourceBytes = Encoding.UTF8.GetBytes(original);
    byte[] hashBytes = sha256Hash.ComputeHash(sourceBytes);

    //  تبدیل بایتهای هش شده به رشته هگزادسیمالرشته
    string hash = BitConverter.ToString(hashBytes).Replace("-", String.Empty);


    Console.WriteLine($"The SHA256 hash of {original} is: {hash}");
}
```

برای درک بهتر فرایند هش، پیشنهاد می‌کنم از لینک زیر استفاده کنید. در این وب‌سایت می‌توانید متنی را انتخاب کرده و با استفاده از الگوریتم‌های مختلف هش، آن را هش کنید. مشاهده نتایج به شما کمک می‌کند تا این فرایند را بهتر درک کنید. https://www.tools4noobs.com/online_tools/hash/

خلاصه

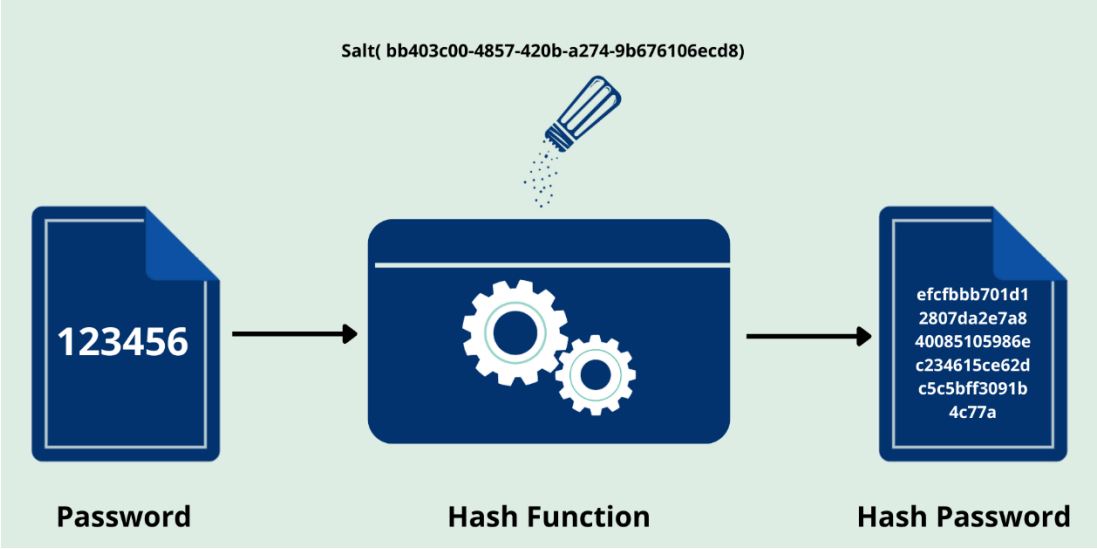
هش کردن پسوردها، تبدیل آن‌ها به رشته‌های پیچیده و غیرقابل بازگشت است، که از اطلاعات کاربران در برابر دسترسی‌های غیرمجاز حفاظت می‌کند. استفاده از هش به جای ذخیره‌سازی پسوردها به صورت متن ساده، امنیت دیتابیس را تضمین می‌کند. با استفاده از الگوریتم‌های قوی مانندSHA-256 ، SHA-3 وbcrypt ، و اضافه کردن Salt به پسوردها قبل از هش کردن، می‌توان از حملات هکرها محافظت کرد و امنیت پسوردها را افزایش داد.

استفاده از Salt برای هش کردن رمزعبورها

Salt چیست؟

تصور کنید پسورد شما مانند یک غذای ساده است. وقتی به این غذا نمک اضافه می‌کنید، طعم آن تغییر می‌کند و دیگر مانند قبل نیست. در دنیای رمزنگاری، Saltهم مثل همین نمک عمل می‌کند. وقتی Salt را به پسورد اضافه می‌کنیم، قبل از اینکه آن را هش کنیم، پسورد تغییر می‌کند و دیگر مانند قبل نیست. این کار باعث می‌شود که پسوردها حتی اگر یکسان باشند، هش‌های متفاوتی داشته باشند. بنابراین، اضافه کردن Salt به پسوردها مثل پاشیدن نمک روی غذا، امنیت آنها را بیشتر می‌کند.

وقتی می‌خواهیم پسورد یک کاربر را هش کنیم، برای او یک Salt منحصر به فرد ایجاد می‌کنیم. این Salt که برای هر کاربر متفاوت است، به پسورد اضافه می‌شود. این کار باعث می‌شود که حتی کاربرانی که از پسوردهای یکسان استفاده می‌کنند، هش‌های کاملاً متفاوتی داشته باشند. به این ترتیب، با تولید Salt اختصاصی برای هر کاربر، امنیت بیشتری در هش کردن پسوردها ایجاد می‌کنیم.



تصویر۳ : فرایند هش با استفاده از Salt

Salt چگونه امنیت هش را افزایش می‌دهد؟

با اضافه کردن Salt به پسورد، هر هش نهایی منحصربه‌فرد خواهد بود. حتی اگر دو کاربر از همان پسورد استفاده کنند، Saltمتفاوت باعث می‌شود هش‌های نهایی متفاوت باشند.

استفاده از Salt هش‌ها را در برابر حملات رینبو تیبل و Dictionary Attacks مقاوم‌تر می‌کند، زیرا هکر نمی‌تواند به سادگی از یک فرهنگ لغت پیش‌ساخته برای یافتن پسوردها استفاده کند.

Salt چگونه ایجاد و ذخیره می‌شود؟

- Salt معمولاً به صورت تصادفی توسط الگوریتم‌های تولید اعداد تصادفی ایجاد می‌شود. این بدان معنی است که هر بار یک پسورد هش می‌شود، یک Salt جدید و منحصربه‌فرد تولید می‌گردد.
- Saltکه به پسوردها اضافه می‌شود، همراه با هش پسورد در دیتابیس ذخیره می‌گردد. اگرچه Salt به تنهایی مخفی نیست، اما بدون داشتن پسورد اصلی، برای هکر کاربردی ندارد. به عبارت دیگر، Saltبه خودی خود اطلاعات محرمانه‌ای نیست، اما در ترکیب با پسورد، امنیت را بالا می‌برد.

اهمیت دسترسی هکر بهSalt

در حالی که Salt معمولاً در کنار هش ذخیره می‌شود و از لحاظ تئوری قابل دسترسی است، دانستن Salt به تنهایی برای هکر کافی نیست. ترکیب Salt با پسورد قبل از هش کردن باعث می‌شود که حملات برای یافتن پسورد اصلی بسیار دشوارتر و زمان‌بر شود. بنابراین، حتی اگر هکر به Salt دسترسی داشته باشد، بدون داشتن پسورد اصلی، نمی‌تواند هش را به پسورد اصلی تبدیل کند.

توجه کنید که به دلیل وجود یک Salt منحصربه‌فرد برای هر پسورد، حتی اگر هکرها بخواهند از لیست پسوردهای رایج استفاده کنند، موفقیت آنها بسیار سخت خواهد بود. به این دلیل که برای هر پسورد، یک ترکیب جدید از پسورد و Salt باید هش شود. این فرآیند نیاز به محاسبات سنگین و زمان‌بر دارد، به خصوص اگر پسوردها پیچیده باشند. در نتیجه، وجود Salt عملاً شناسایی پسوردها را توسط هکرها با استفاده از روش‌های معمول، غیرممکن یا بسیار دشوار می‌کند. این ویژگیSalt ، لایه امنیتی اضافی ایجاد کرده و از پسوردها در برابر دسترسی‌های غیرمجاز به طور موثری محافظت می‌کند.

استفاده از Hash و Salt درASP.NET Core Identity

ASP.NET Core Identityیکی از ابزارهای قدرتمند مایکروسافت برای مدیریت کاربران و احراز هویت در برنامه‌های وب است. یکی از مزایای بزرگ استفاده ازIdentity ، رویکرد امنیتی پیشرفته‌ای است که برای ذخیره‌سازی پسوردها به کار می‌برد، یعنی استفاده از تکنیک‌های Hash و Salt .

چگونگی عملکرد

ASP.NET Core Identity، پسوردها را به صورت خودکار هش می‌کند و به آنها یک Salt تصادفی اضافه می‌کند تا امنیت آنها را به شکل چشمگیری افزایش دهد. پسوردهای هش شده به همراه Salt در قسمت PasswordHash جدول کاربران، ذخیره می‌شوند. و برای اطمینان از اعتبار پسوردها در صورت تغییرات، از فیلد SecurityStamp استفاده می‌شود.

درASP.NET Core Identity ، فرآیند هش کردن و استفاده از Salt برای پسوردها به این صورت انجام می‌شود:

۱. **هش کردن و اضافه کردنSalt :** وقتی کاربر یک پسورد را تعیین می‌کند، ASP.NET Core Identity ابتدا به آن پسورد یک Salt تصادفی اضافه می‌کند.

۲. **ذخیره‌سازی در دیتابیس** :هش نهایی، که حاصل ترکیب پسورد و Salt است، به همراه Salt در یک فیلد به نام PasswordHash در جدول کاربران ذخیره می‌شود. این فیلد هم پسورد هش شده و هم Saltرا در خود نگه می‌دارد.

۳. **بازیابی Salt هنگام ورود** :زمانی که کاربر تلاش می‌کند وارد سیستم شود، ASP.NET Core Identity ابتدا Salt را از PasswordHash استخراج می‌کند. سپس، با استفاده از همین Salt و پسوردی که کاربر وارد کرده است، هش جدیدی تولید می‌کند.

۴. **تطبیق هش‌ها برای تأیید هویت** :اگر هش تولید شده از پسورد وارد شده توسط کاربر با هش ذخیره‌شده در PasswordHash مطابقت داشته باشد، پسورد صحیح است و کاربر می‌تواند وارد سیستم شود.

PasswordHash	SecurityStamp
AQAAAAEAACcQAAAAEHdxqw0O...	TJZJFPIXHUDMPXSGHYZHLHJENOVQB...

تصویر۳ – ذخیره اطلاعات توسط Identity

این روش باعث می‌شود که حتی اگر دیتابیس هک شود، هکرها نتوانند به راحتی پسوردهای اصلی را بازیابی کنند، زیرا آن‌ها فقط به هش پسوردها دسترسی دارند، نه خود پسوردها. همچنین، استفاده از Salt منحصربه‌فرد برای هر کاربر به این معناست که حتی اگر دو کاربر پسورد یکسانی داشته باشند، هش‌های متفاوتی تولید می‌شوند. این فرآیند یک لایه امنیتی اضافی به سیستم اضافه می‌کند.

چرا باید از ASP.NET Core Identity استفاده کرد؟

- **امنیت بالا** :با استفاده از الگوریتم‌های هش مدرن و افزودنSalt ، ASP.NET Core Identity اطمینان می دهد که پسوردهای کاربران به بهترین شکل محافظت می‌شوند.
- **سهولت در استفاده** :تمام این فرآیندهای پیچیده به صورت پشت‌پرده و خودکار انجام می‌گیرند، بنابراین برنامه‌نویسان نیازی به نگرانی در مورد جزئیات فنی ندارند.
- **استانداردهای امنیتی** :با استفاده ازASP.NET Core Identity ، می‌توانید از رعایت استانداردهای امنیتی مطمئن باشید و از این طریق اعتبار برنامه وب خود را افزایش دهید.

به عنوان یک برنامه‌نویس دات نت، استفاده از ASP.NET Core Identity می‌تواند تجربه کاربری امن‌تر و راحت‌تری را برای کاربران شما فراهم کند، در حالی که امنیت داده‌های حساس آنها را تضمین می‌کند. این ابزار از پیچیدگی‌های مدیریت دستی پسوردها جلوگیری کرده و امنیت سیستم شما را به سطح بالاتری می‌برد.

خلاصه

درASP.NET Core Identity ، پسوردها با استفاده از Hash و Salt امن می‌شوند. Saltیک داده تصادفی است که به پسورد اضافه می‌شود و باعث می‌شود هر پسورد یک هش منحصربه‌فرد تولید کند. این Salt به همراه هش پسورد در دیتابیس ذخیره می‌شود. وقتی کاربر وارد سیستم می‌شود، Identityابتدا Salt را بازیابی می‌کند و با ترکیب آن با پسورد وارد شده، هش جدیدی تولید می‌کند. اگر این هش با هش ذخیره‌شده در دیتابیس

مطابقت داشته باشد، پسورد صحیح است. استفاده از این روش امنیت داده‌های کاربران را تضمین کرده و به سادگی قابل استفاده است.

Authentication Tokens

توکن احراز هویت چیست؟

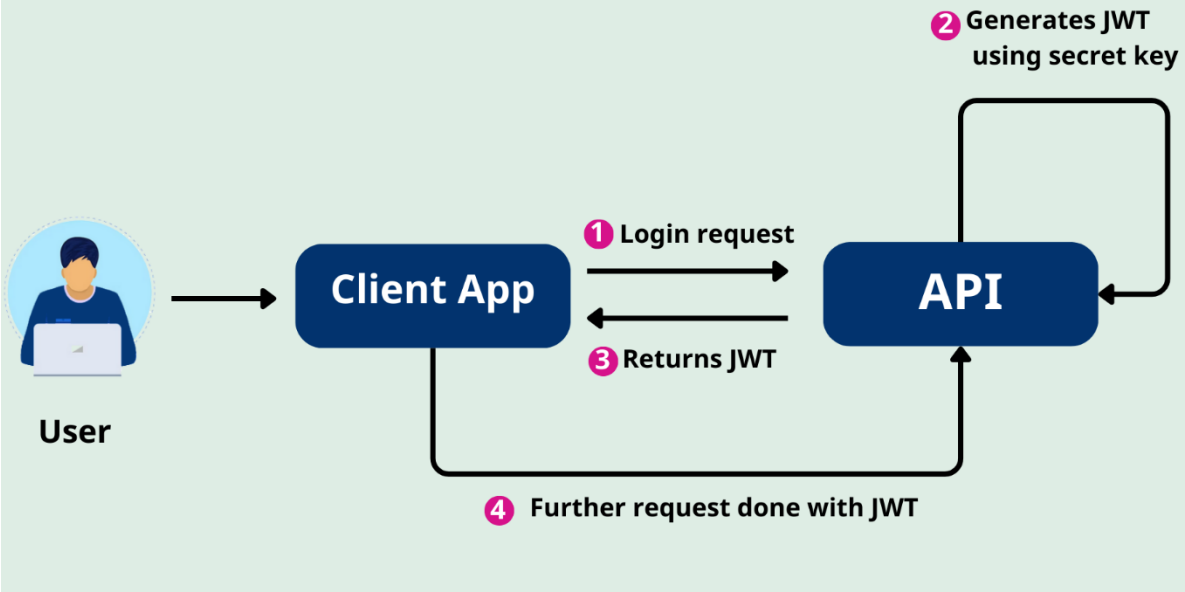
توکن‌های احراز هویت نوعی از کلیدهای دیجیتالی هستند که در فرآیندهای احراز هویت به کار می‌روند. این توکن‌ها، که می‌توانند به صورت رشته‌های رمزگذاری شده باشند، اطلاعات کاربر را به شکل امن نگهداری می‌کنند و برای تایید هویت کاربر در هر بار تعامل با سرور استفاده می‌شوند.

چه مشکلی را برطرف می‌کنند؟

توکن‌های احراز هویت مشکلات مربوط به امنیت ورود و نگهداری اطلاعات حساس را حل می‌کنند:

1. **کاهش خطر نفوذ** :به جای ذخیره‌سازی اطلاعات ورود یا ارسال مکرر آنها بین کلاینت و سرور، توکن احراز هویت یک بار صادر شده و برای تایید هویت‌های بعدی استفاده می‌شود. این امر از خطر نفوذ و دزدیده شدن اطلاعات کاربر در هر بار ارتباط با سرور می‌کاهد.
2. **فرآیند ورود امن‌تر** :با استفاده از توکن‌ها، اطلاعات کاربر تنها در هنگام لاگین اولیه ارسال می‌شوند. پس از آن، توکن صادر شده برای تایید هویت استفاده می‌شود که به معنای کاهش تبادل اطلاعات حساس است.
3. **دسترسی مدیریت شده** :توکن‌ها می‌توانند محدودیت‌های زمانی داشته باشند و اطلاعاتی در مورد دسترسی‌های کاربر به سیستم ارائه دهند. این امکان مدیریت دسترسی‌ها را بهبود می‌بخشد و امنیت کلی سیستم را تقویت می‌کند.

به طور خلاصه، استفاده از توکن‌های احراز هویت به عنوان یک رویکرد امن‌تر برای مدیریت ورود و دسترسی کاربران، نقش مهمی در افزایش امنیت و کارآمدی سیستم‌های مبتنی بر وب دارد. این تکنیک به کاربران اجازه می‌دهد تا به شکلی ایمن‌تر و راحت‌تر با سیستم‌ها تعامل داشته باشند، بدون اینکه نیاز باشد اطلاعات حساس خود را به دفعات وارد کنند.



تصویر ۴ : احراز هویت با استفاده از JWT

فرآیند ایجاد توکن‌های احراز هویت

توکن‌های احراز هویت نقش کلیدی در تأمین امنیت و تسهیل دسترسی کاربران در سیستم‌های مبتنی بر وب دارند. این فرآیند شامل مراحل زیر است:

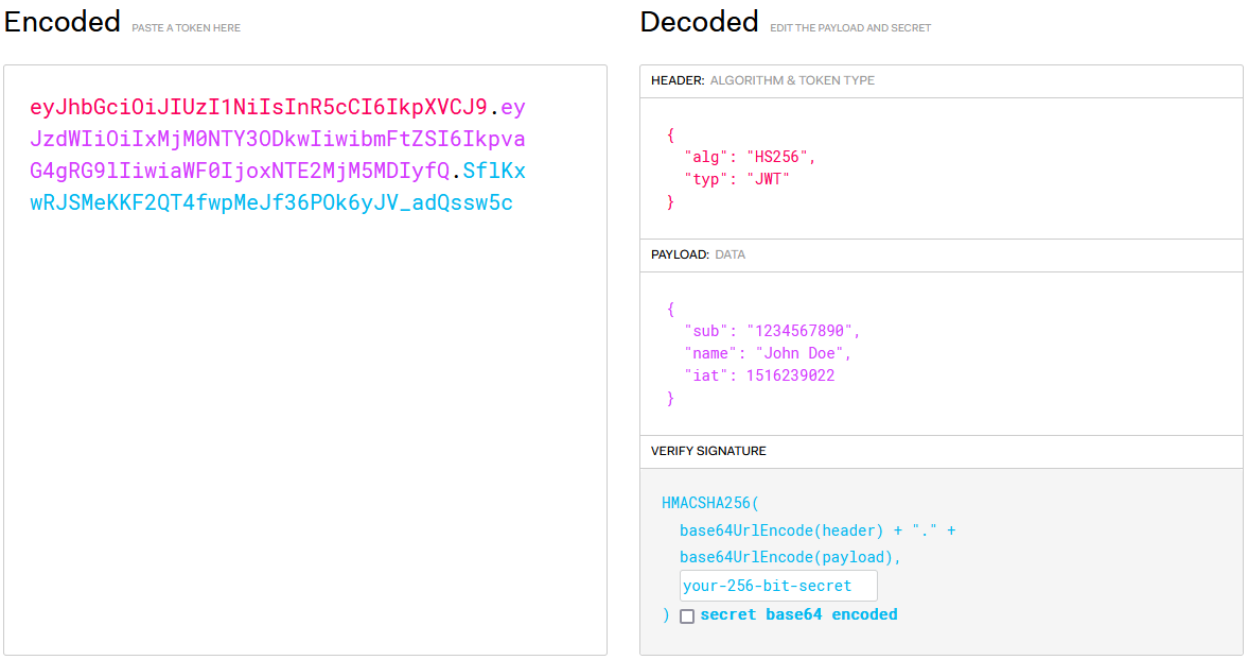
1. **درخواست کاربر** :فرآیند با یک درخواست ورود از سمت کاربر شروع می‌شود. کاربر نام کاربری و رمزعبور خود را به سرور ارسال می‌کند.
2. **احراز هویت** :سرور اطلاعات دریافتی را بررسی می‌کند. اگر اطلاعات کاربر صحیح باشند، فرآیند احراز هویت موفقیت‌آمیز است.
3. **ایجاد توکن** :پس از احراز هویت موفق، سرور یک توکن احراز هویت ایجاد می‌کند. این توکن معمولاً شامل اطلاعات کاربری و دیگر داده‌های مربوط به جلسه کاربر است و از طریق رمزگذاری امن محافظت می‌شود.
4. **ارسال توکن به کاربر** :توکن ایجاد شده به کاربر بازگردانده می‌شود و معمولاً در کلاینت (مثلاً مرورگر یا اپلیکیشن موبایل) ذخیره می‌شود.
5. **استفاده از توکن در درخواست‌های بعدی** :کاربر برای هر درخواست بعدی به سرور، این توکن را ارسال می‌کند. سرور توکن را بررسی می‌کند و اگر معتبر باشد، به درخواست کاربر پاسخ می‌دهد.
6. **انقضای توکن** : توکن‌ها معمولاً مدت زمان معینی اعتبار دارند. پس از اتمام این مدت، به جای اینکه کاربر مجبور باشد دوباره با نام کاربری و رمزعبور خود وارد شود، می‌توان از فرآیندی به نام **رفرش توکن** استفاده کرد. این فرآیند به کاربر اجازه می‌دهد که بدون نیاز به ورود مجدد اطلاعات حساس خود، یک توکن جدید دریافت کند.

.در این فرآیند، هنگام صدور توکن اولیه، یک **رفرش توکن** نیز به کاربر داده می‌شود. این رفرش توکن به کاربر امکان می‌دهد که در صورت انقضای توکن اصلی، از طریق ارسال رفرش توکن به سرور، بدون نیاز به وارد کردن مجدد نام کاربری و رمزعبور، یک توکن جدید دریافت کند. این روش باعث می‌شود تجربه کاربری روان‌تر و کارآمدتر شود، در حالی که امنیت را حفظ می‌کند.

فرآیند ایجاد توکن‌های احراز هویت به این ترتیب اطمینان حاصل می‌کند که هر تعامل بین کاربر و سرور به شکلی ایمن و کارآمد انجام شود. این روش نه‌تنها از اطلاعات کاربران محافظت می‌کند بلکه تجربه کاربری را نیز با کاهش نیاز به ورود مکرر اطلاعات ورود، بهبود می‌بخشد.

JWT (JSON Web Tokens)

JWT یک استاندارد (RFC 7519) برای ایجاد توکن‌های احراز هویت است که به طور گسترده در ارتباطات برنامه‌های وب استفاده می‌شود. این توکن‌ها، که به صورت رشته‌های رمزنگاری شده‌ای هستند، اطلاعات مفیدی در مورد کاربر و claims مربوط به آن کاربر می باشد.



تصویر۵ : نمونه ای از توکن احراز هویت

ساختارJWT

یک توکن JWT از سه قسمت تشکیل شده است که توسط نقطه (.) از یکدیگر جدا می‌شوند:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

این یک نمونه توکن JWT می باشد

۱. **Header**

- بخش اول یکJWT ، به نامHeader ، اطلاعاتی درباره نوع توکن که معمولاً JWT است و روشی که برای محافظت از توکن استفاده شده مثل HMAC SHA256 یا RSA را در خود دارد. به زبان ساده، این قسمت به ما می‌گوید توکن چه نوعی است و چطور امن شده است.
- به صورت JSON نوشته شده و سپس به Base64Url رمزگذاری می‌شود.

۲. **Payload**

- این بخش حاوی claimsاست که اطلاعاتی در مورد کاربر و دیگر داده‌های موردنیاز برنامه را در بر می‌گیرد.
- Claim ها می‌توانند هم عمومی مانند نام کاربری و هم خصوصی مانند اطلاعات کاربری خاص باشند.
- Payload نیز مشابه Header به صورت JSON نوشته شده و به Base64Url کدگذاری می‌شود.

۳. **Signature**

- قسمت امضا در یک JWT برای تأیید اینکه توکن توسط سرور معتبر صادر شده و بدون تغییر باقی مانده، طراحی شده است.
- سرور این بخش را با گرفتن بخش‌های Header و Payload که به صورت کدگذاری شده هستند، ترکیب می‌کند و سپس با استفاده از یک کلید خصوصی که فقط سمت سرور موجود است، آن‌ها را هش می‌کند.
- وقتی کاربر توکن را برای سرور ارسال می‌کند، سرور با استفاده از همان کلید خصوصی، امضای توکن را بررسی می‌کند تا اطمینان حاصل کند که توکن توسط کاربر تغییر نیافته است.

نحوه کار JWT

۱. **احراز هویت کاربر**

کاربر با استفاده از اطلاعات احراز هویت خود مانند نام کاربری و رمز عبور وارد سیستم می‌شود.

۲. **صدور توکنJWT**

سرور پس از تأیید اطلاعات کاربر، یک توکن JWT صادر کرده و آن را به کاربر ارسال می‌کند.

۳. **استفاده از توکن**

کاربر برای دسترسی به منابع محافظت شده در سمت سرور، توکن JWT را در هدر درخواست‌های HTTP خود قرار می‌دهد.

۴. **تأیید توکن**

سرور توکن را بررسی کرده و در صورت اعتبار، به درخواست کاربر پاسخ می‌دهد.

مزایا و کاربردها

- **کم حجم و سریع** :به دلیل ساختار کم حجم، در ارتباطات شبکه سربار کمی ایجاد می‌کند.
- **خودکفا** :تمام اطلاعات مورد نیاز در خود توکن وجود دارد، بنابراین به دیتابیس برای تأیید هویت نیاز نیست.
- **امن** :امکان تغییر توکن توسط کاربران غیرمجاز وجود ندارد، مگر اینکه آن‌ها به کلید رمزنگاری دسترسی داشته باشند.
- **مقیاس‌پذیر و قابل استفاده در چندین سیستم** :مناسب برای سناریوهای Single Sign-On و ارتباطات میکروسرویس‌ها.

نکات امنیتی

- باید از الگوریتم‌های رمزنگاری قوی استفاده شود.
- کلیدهای رمزنگاری باید به دقت مدیریت و محافظت شوند.
- برای جلوگیری از حملات تکراری، باید از مکانیزم‌هایی مانند زمان انقضا استفاده کرد.

JWTیک روش امن و کارآمد برای مدیریت احراز هویت و ارتباطات در برنامه‌های وب و موبایل محسوب می‌شود و به خصوص در معماری‌های مبتنی بر API محبوبیت بالایی دارد.

خلاصه

JWT (JSON Web Tokens)یک استاندارد مدرن برای تولید توکن‌های احراز هویت است که در ارتباطات وب کاربرد دارد. این توکن‌ها با استفاده از رمزگذاری، اطلاعات کاربری و دسترسی‌ها را به صورت امن نگهداری می‌کنند.

ساختار JWT شامل سه قسمت است:

۱. **Header**: اطلاعات درباره نوع توکن و الگوریتم رمزنگاری.

۲. **Payload**: حاوی اطلاعات کاربری و claims مربوطه.

۳. **Signature**: بخشی که توسط سرور امضا می‌شود تا اطمینان حاصل کند که توکن تغییر نکرده است.

فرآیند کار با JWT به این صورت است:

- کاربر با ورود اطلاعات خود، احراز هویت می‌شود.
- سرور پس از تایید اطلاعات کاربر، یک توکن JWT صادر می‌کند.
- کاربر این توکن را برای دسترسی به منابع محافظت شده به سرور ارسال می‌کند.
- سرور توکن را بررسی کرده و در صورت اعتبار، به درخواست‌های کاربر پاسخ می‌دهد.

مزایای استفاده از JWT شامل سرعت بالا، خودکفایی، امنیت قوی، و قابلیت استفاده در سناریوهای متنوع مانند Single Sign-On و میکروسرویس‌ها است. امنیت در JWT با استفاده از الگوریتم‌های رمزنگاری قوی و مدیریت دقیق کلیدهای رمزنگاری تضمین می‌شود JWT .به خصوص در معماری‌های مبتنی بر API بسیار محبوب است.

یادداشت پایانی

همانطور که در این کتاب مشاهده کردید، امنیت در دنیای دیجیتال یک مقوله پیچیده و دائماً در حال تغییر است. در حالی که چهار روش ارائه شده در این کتاب - رمزگذاری داده‌ها، هش کردن پسوردها، استفاده ازSalt ، و توکن‌های احراز هویت - بنیادی و ضروری هستند، باید تأکید کنم که این‌ها تنها نقطه شروعی برای ایجاد یک محیط امن هستند.

به خاطر داشته باشید که امنیت کامل تنها با انجام این حداقل‌ها تضمین نمی‌شود. با این حال، پیاده‌سازی این تکنیک‌ها قطعاً سطح امنیتی اپلیکیشن‌های شما را ارتقا خواهد داد و به شما کمک می‌کند تا در برابر تهدیدات دیجیتالی متداول قوی‌تر باشید.

در وب‌سایت باگتو، ما متعهد به ارائه آموزش‌های تخصصی در زمینه‌های دات نت و سی شارپ هستیم. برای رشد و پیشرفت در این حوزه‌ها، شما را دعوت می‌کنیم تا از مقالات و ویدیوهای آموزشی رایگان موجود در سایت ما بهره‌مند شوید.

علاوه بر این، دوره‌های ما با عنوان **ستارگان** طراحی شده‌اند تا به شما در کسب مهارت‌های پیشرفته و اعتلای دانش فنی‌تان کمک کنند. این دوره‌ها فرصتی عالی برای یادگیری عمیق‌تر و توسعه تخصصی‌تر در زمینه‌های مورد علاقه‌تان هستند.

پس نگاهی به وب‌سایت ما بیندازید و از فرصت یادگیری بی‌نظیری که در انتظار شماست، استفاده کنید.

احسان بابائی