از فرم تا پایگاه داده: پیشگیری از XSS ، sql injection

و اعتبارسنجی ناکافی

نویسنده: محمد مهدی عبدی

دانشجوی رشته‌ی مهندسی برق – دانشگاه علم و صنعت ایران

فهرست:

چکیده:

امنیت نرم‌افزارهای وب به‌طور مستقیم با نحوه‌ی مدیریت ورودی‌های کاربر در ارتباط است. آسیب پذیری هایی مانند: sql-injection و Cross-Site Scripting (xss) نشان داده‌اند که حتی یک خط کد اشتباه میتواند کل سامانه را در معرض نفوذ قرار دهد. در این مقاله، با تکیه بر یک سناریوی عملی (فرم ارسال کامنت) نشان می‌دهیم که چگونه بی‌توجهی به اعتبارسنجی داده‌ها در سمت کاربر و سرور، می‌تواند زمینه بروز حملات جدی را فراهم کند. در ادامه، نمونه‌های واقعی از کد ناامن و نسخه‌ی اصلاح‌شده‌ی آن‌ها ارائه می‌شود و راهکارهایی عملی برای جلوگیری از این دسته حملات پیشنهاد خواهد شد. هدف این نوشتار، ایجاد پلی میان دانش فنی توسعه‌دهندگان فرانت‌اند و الزامات امنیتی بک‌اند است.

۱. فصل اول: اهمیت امنیت ورودی‌های کاربر در توسعه وب

۱،۱. چشم انداز کلی:

امنیت برنامه‌های بستر وب یکی از مهم‌ترین چالش‌های دنیای فناوری اطلاعات محسوب می‌شود. طی دو دهه‌ی اخیر، با گسترش استفاده از اینترنت و افزایش وابستگی سازمان‌ها و کاربران به سرویس‌های آنلاین، بسترهای وب به هدف اصلی مهاجمان سایبری تبدیل شده‌اند. گزارش‌های امنیتی متعدد، از جمله گزارش سالانه OWASP نشان می‌دهند که بخش قابل توجهی از حملات موفق، ناشی از **مدیریت نادرست ورودی کاربر** است.

دو نمونه‌ی بارز این مدل حملات، sql injection و cross-site scripting هستند. در حملات SQLi[1] مهاجم با واردکردن کدهای مخرب در ورودی‌های سمت کلاینت می‌تواند ساختار دستورات پایگاه[2] داده را تغییر دهد و به داده‌های حساس دسترسی پیدا کند. در مقابل، حملات XSS با تزریق اسکریپت‌های مخرب به صفحات وب، زمینه‌ی اجرای کد دلخواه مهاجم در مرورگر سایر کاربران را فراهم می‌سازند. هر دو آسیب‌پذیری، در صورت بی‌توجهی به اعتبارسنجی ورودی‌ها و استفاده‌ی نادرست از ابزارهای توسعه، می‌توانند خسارات گسترده‌ای ایجاد کنند.

این مقاله بر پایه‌ی یک سناریوی ساده اما پرکاربرد طراحی شده است: **فرم ارسال کامنت**

این مثال، فرصتی فراهم می‌کند تا هر سه لایه‌ی اصلی مسیر داده یعنی فرانت‌اند، بک‌اند و پایگاه داده مورد بررسی قرار گیرند.

---

[1] Sql injection

[2] Change query

ابتدا نحوهٔ وقوع حملات SQLi[3] و XSS[4] در این سناریو شرح داده می‌شود. سپس با ارائه‌ی نمونه‌کدهای ناامن و امن، نشان داده خواهد شد که چگونه می‌توان از بروز چنین تهدیداتی جلوگیری کرد. در کنار این موارد، به اهمیت اعتبارسنجی داده‌ها در هر دو سمت کلاینت و سرور نیز پرداخته می‌شود.

هدف نهایی این نوشتار، ارائه‌ی یک نگاه کاربردی و عملیاتی به امنیت ورودی‌های کاربر است؛ نگاهی که نه‌تنها برای توسعه‌دهندگان بک‌اند بلکه برای برنامه‌نویسان فرانت‌اند نیز قابل درک و استفاده باشد. بدین ترتیب، مقاله می‌کوشد شکاف میان مباحث تئوری امنیت و نیازهای واقعی پروژه‌های نرم‌افزاری را پر کند.

۱،۲. معرفی سناریو (فرم ارسال کامنت)

برای بررسی عملی آسیب‌پذیری‌های امنیتی مورد نظر، یک فرم ساده‌ی ارسال کامنت به‌عنوان سناریوی اصلی انتخاب شده است. این مثال به دلیل سادگی و در عین حال کاربرد گسترده، بستری مناسب برای تحلیل تهدیدات امنیتی فراهم می‌کند. در معماری چنین سیستمی، کاربر محتوای متنی را در یک فیلد ورودی ثبت می‌کند و سپس با فشردن دکمه‌ی ارسال، داده به سرور منتقل شده و در پایگاه داده ذخیره می‌شود. در مرحله‌ی بعد، این داده‌ها دوباره به سمت کلاینت ارسال شده و در صفحه‌ی وب به نمایش در می‌آیند.

ویژگی این مثال این است که در عین سادگی، از سه بخش فرانت-اند، بک-اند و پایگاه داده استفاده میشود. و امکان آسیب پذیری در هر سه بخش بررسی میشود. برای نمونه، اگر ورودی کاربر در سمت فرانت‌اند بدون اعتبارسنجی رها شود، مهاجم می‌تواند داده‌ای فراتر از انتظار وارد کند. اگر در بک‌اند، داده بدون استفاده از پرس‌وجوهای امن به پایگاه داده ارسال شود خطر SQLi وجود دارد افزون برین امکان تغییر در کد های سمت فرانت-اند برای کاربری که قصد نفوذ به سایت را دارد بسیار ساده‌تر نسبت به تغییر در کد های سمت بک اند هست. همچنین اگر نمایش دیتا ها در صفحات وب بدون پاکسازی و دقت انجام بگیرد امکان حملات XSS بوجود می‌آید.

نمونه‌ی بسیار ساده‌ی چنین فرمی در HTML به شکل رویه‌رو است:

```html
<form id="commentForm">
    <input id="commentInput" name="comment" type="text" placeholder="write your comment">
    <button type="submit">ارسال || send</button>
</form>

<section id="commentsList">
    <!-- نمایش کامنت‌ها -->
     <!-- show comments -->
</section>

<script src="app.js"></script>
```

در کد بالا، المان‌ها با شناسه مشخص شده‌اند تا در سمت کلاینت با جاوااسکریپت مدیریت شوند.

[3] Sql injection
[4] Cross-site scripting

در نسخه‌ی ناامن، اسکریپت جاوااسکریپت داده‌ی ورودی کاربر را بدون هیچ‌گونه اعتبارسنجی به سرور ارسال کرده و هنگام نمایش، از ویژگی innerHTML استفاده میشود:

```
// app.js (نسخه ناامن) || (Insecure version)

// Getting element from html ⇓
const form = document.querySelector('#commentForm');
const input = document.querySelector('#commentInput');
const list = document.querySelector('#commentsList');

form.addEventListener('submit', async (e) => {
  e.preventDefault();
  const value = input.value;

  // ارسال بدون اعتبارسنجی || fetch without Validation
  await fetch('/api/comments', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({ comment: value })
  });

  // رندر ناامن: innerHTML مستقیم || innerHTML
  const res = await fetch('/api/comments');
  const comments = await res.json();
  list.innerHTML = comments.map(c => `<div>${c.comment}</div>`).join('');
});
```

این رویکرد زمینه‌ساز حملات است؛ چرا که ورودی کاربر مستقیماً در HTML تفسیر می‌شود. در نتیجه، اگر مهاجم در بخش کامنت خود قطعه‌کدی مانند: <script>alert(“XSS”)<script> وارد کند، مرورگر سایر کاربران آن را اجرا خواهد کرد. در مقابل، نسخه‌ی ایمن‌تر همین سناریو از اعتبارسنجی اولیه و رندر امن استفاده می‌کند. برای مثال، با خاصیت textContent میتوان تضمین کرد که داده به صورت متن خام نمایش داده شود و مرورگر آن را به کد تبدیل نکند.

۲. فصل دوم: SQL Injection

۲،۱. معرفی و نحوه وقوع

منظور از SQL Injection دسته‌ای از حملات است که مهاجم با ارسال ورودی ساختگی به برنامه، ساختار پرس و جوی SQL را تغییر میدهد(باعث تغییر query مورد نظر میشود). که این تغییر در ساختار میتواند باعث دسترسی غیرمجاز به دیتابیس که شامل خواندن، حذف، تغییر یا حتی ساخت دیتای جدید، میشود. این آسیب‌پذیری ناشی از قرار گیری مستقیم ورودی کاربر در رشته‌ی SQL میباشد.

چرا خطرناک است؟

یک SQLi موفق می‌تواند به نشت داده‌های حساس (یوزرها، پسوردها، توکن‌ها)، تغییر یا حذف داده‌ها، و در برخی موارد به اجرای دستورات مدیریتی روی دیتابیس منجر شود. از آنجا که پایگاه‌داده معمولاً منبع حقیقت (source of truth) سیستم است، تأثیر این حمله بسیار بالا است.

چطور اتفاق می‌افتد؟

۱) مهاجم یک مقدار را از طریق فرم، پارامتر URL یا درخواست API ارسال می‌کند.
۲) سرور آن مقدار را بدون پارامترایز یا آماده‌سازی در یک رشته SQL (string concatenation) قرار می‌دهد.
۳) رشته‌ی اصلی به رشته‌ای جدید تبدیل میشود. و به عنوان رشته‌ی اصلی پایگاه داده شناخته می‌شود.
۴) دیتابیس آن رشته را به‌عنوان بخشی از پرس‌وجو اجرا می‌کند و ساختار منطقی پرس‌وجو تغییر می‌یابد.
۵) نتیجه اجرای پرس‌وجوی تغییر یافته همان نتایجی را می‌دهد که مهاجم می‌خواهد. (میتواند هر کدام از عملیات CRUD[5] باشد.)

مثالی از ساختار ناایمن:

```
// مثال بد – نمونهٔ نمادین (DO NOT use)
const q = "SELECT id, content FROM comments WHERE author = '" + req.body.author + "'";
const rows = await pool.query(q);
```

در این مثال، ورودیِ req.body.author ستقیماً در رشته‌ی SQL جای‌گذاری شده است؛ اگر محتوای ورودی شکل ساختار پرس‌وجو را تغییر دهد، نتیجه‌ی ناخواسته رخ می‌دهد.

[5]CRUD: Create, Read, Update, Delete

مثال ایمن(پارامترایز):

```
// مثال امن با pg (parameterized)
const q = 'SELECT id, content FROM comments WHERE author = $1';
const { rows } = await pool.query(q, [req.body.author]);
```

استفاده از پارامترها باعث می‌شود موتور پایگاه‌داده مقدار کاربر را به‌عنوان داده (نه بخشِ دستوری) پردازش کند و از تغییر ساختار رشته ی SQL جلوگیری شود.

الگوهای دفاعی تکمیلی:

۱) همیشه از پارامترایز[۶]، ORM یا Query Builder استفاده شود.

۲) اعتبارسنجی ورودی در سرور: نوع، طول، الگو[۷] مقدار بررسی شود.

۳) استفاده از WAF و محدودیت نرخ[۸] برای کاهش حملات خودکار.

۴) تست منظم و تست‌های نفوذ برای بخش های حساس برنامه انجام شود.

۲٫۲. جمع بندی فصل دوم

SQLi یک آسیب‌پذیری ساختاری است که ریشه‌اش در قرار دادن ورودی کاربر در قالب رشته‌ی دستوری SQL بدون جداسازی مناسب داده و دستور است. پیامدهای یک SQLi موفق می‌تواند از افشای داده‌های حساس تا تغییر یا حذف داده‌ها و در موارد نادر، ایجاد دسترسی مدیریتی به سامانه را شامل شود.

برای پیشگیری مؤثر، تاکید اصلی بر دو اصل است:

۱) هرگز ورودی کاربر بصورت مستقیم در رشته‌ی SQL قرار نگیرد. همواره از پارامترایز امن استفاده شود.

۲) اعتبارسنجی و سیاست‌های بررسی (type, length, format) باید در سمت سرور اعمال شوند، چون اعتبارسنجی فرانت‌اند به‌راحتی قابل دورزدن است.

---

[6] parameterized queries
[7] regex
[8] rate limiting

۳. فصل سوم: Cross-Site Scripting

۳،۱. معرفی و انواع رایج:

XSS[9] مجموعه‌ای از آسیب‌پذیری هاست که در آن ورودی که در آن ورودیِ کنترل‌شده توسط کاربر به نحوی وارد خروجی DOM می‌شود که مرورگر آن را به‌عنوان کد اجرایی تفسیر می‌کند. نتیجه‌ی XSS اجرای اسکریپت دلخواه مهاجم در زمینه‌ی دامنه‌ی قربانی است؛ عملیاتی مانند دزدیدنsession ، انجام درخواست‌های ناخواسته به نام کاربر، تغییر ظاهر، محتوای صفحه وب یا نمایش محتوای فیشینگ ممکن می‌شود.

سه نوع رایج XSS بصورت خلاصه:

۱) ذخیره‌شده[۱۰]:

- payload مخرب در سرور یا پایگاه‌داده ذخیره می‌شود (مثلاً در جدول کامنت‌ها) و هر بار که صفحه واکشی می‌شود، برای همه‌ی کاربران رندر و اجرا می‌گردد.
- معمولاً همه‌ی بازدیدکنندگان صفحه در معرض قرار می‌گیرند.
- نمونه‌برداری فنی: فرم‌های کامنت، پروفایل‌کاربری، فیلدهای قابل ویرایش

۲) بازتابی[۱۱]:

- payloadدر پارامتر درخواست(URL یا فرم)ارسال می‌شود و سرور آن را بدون پاک‌سازی در پاسخ قرار می‌دهد؛ داده ذخیره نمی‌شود و اجرا تنها در پاسخ همان درخواست رخ می‌دهد.
- محدودتر) نیاز به فریبِ کاربر برای کلیک روی لینکِ (crafted ولی در حملات فیشینگ/مهندسی اجتماعی کاربردی است.
- نمونه‌برداری فنی: پارامترهای جستجو، پیام خطا که پارامتر ورودی را نمایش می‌دهد.

۳) مبتنی برDOM[۱۲]

- هیچ دخالتی از سمت سرور نیست؛ کد جاوااسکریپتِ سمت کلاینت ورودی را از URL/DOM می‌خواند و مستقیم در DOM می‌گذارد. برای مثال ورودی یا استفاده از innerHTML وارد صفحه وب میشود.
- وابسته به منطق کلاینت و پیچیدگیSPA[13] ها؛ در اپلیکیشن‌های مدرن با مصرف زیاد JS شایع است.
- نمونه‌برداری فنی: manipulation مستقیم یا استفاده از innerHTML بصورت نا ایمن.

---

[9] Cross-Site Scripting
[10] Stored XSS
[11] Reflected XSS
[12] DOM-based XSS
[13] Single Page Aplication

چرا باید جدی گرفته شود؟

XSSمی‌تواند منجر به سرقت نشست‌کاربری) اگر کوکی‌ها HttpOnly نباشند(، جعل درخواست‌ها از طرف کاربر، تغییر محتوای صفحات و اجرای حملات فیشینگ درون‌سایتی شود. از آنجا که اجرا در مرورگر قربانی رخ می‌دهد، اثبات آسیب‌پذیری و اثر آن معمولاً ساده و سریع است.

نکاتِ سریعِ دفاعی (چکیده):

- خروجی را escape کن: هنگام قرار دادن داده‌ی کاربر در HTML از escaping مناسب استفاده کن. (در فرانت اند از textContent)
- از innerHTML و dangerouslySetInnerHTML اجتناب کن.
- کوکی‌های حساس را با HttpOnly ست کن تا JS نتواند آن‌ها را بخواند.
- سیاست CSP[14] مناسب اعمال کن تا اجرای اسکریپت‌های inline و منابع ناشناس محدود شود.

۳/۲. نمونه‌ی ناامن و نسخه‌ی اصلاح‌شده:

در سناریوی «فرم ارسال کامنت» رایج‌ترین اشتباهِ فرانت این است که داده‌ٔ کاربر را بدون پاک‌سازی دوباره در DOM قرار دهند. برای مثال از innerHTML استفاده شود. همچنین بعضی تیم‌ها فقط به اعتبارسنجی فرانت اکتفا می‌کنند و روی سرور هیچ‌چیز را بررسی نمی‌کنند. ترکیب این دو شرط، زمینه‌ی XSS ذخیره‌شده را فراهم می‌کند.

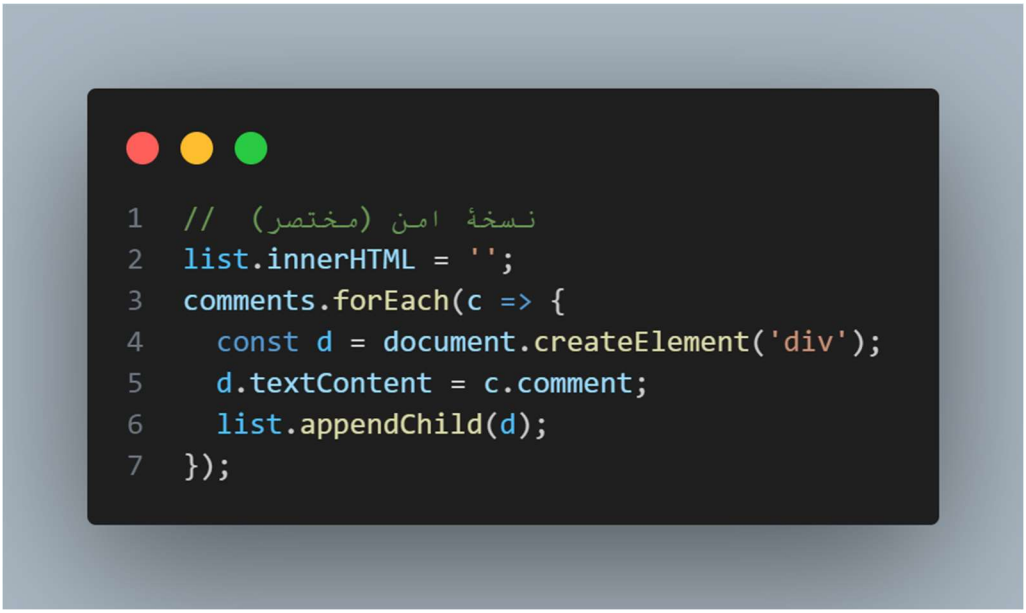
نمونه‌ی بسیار مختصر ناامن

```
// نسخهٔ نا امن (مختصر)
list.innerHTML = comments.map(c => `<div>${c.comment}</div>`).join('');
```

در این حالت هر تگی که در c.commentباشد، در DOM تفسیر و احتمالاً اجرا می‌شود. و احتمال بروز آسیب‌پذیری وجود دارد.

[14] Content Security Policy

نسخه‌ی اصلاح‌شده(ایمن):



```
// نسخهٔ امن (مختصر)
list.innerHTML = '';
comments.forEach(c => {
  const d = document.createElement('div');
  d.textContent = c.comment;
  list.appendChild(d);
});
```

استفاده از textContentتضمین می‌کند که محتوای کاربر به‌عنوان متن معمولی نمایش داده شود و مرورگر آن را اجرا نکند.

۳٫۳. جمع‌بندی فصل سوم

XSSزمانی رخ می‌دهد که داده‌ی کنترل‌شده توسط کاربر به گونه‌ای وارد HTML/DOM شود که مرورگر آن را به‌عنوان کد اجرا کند. مهم‌ترین نکته این است که دفاعِ مؤثر، **چندلایه** است: فقط یک تغییر در فرانت کافی نیست و فقط پچ کردن یک endpointهم تضمین‌کننده‌ی امنیت کل سیستم نیست.

- هر جا کاربر داده‌ای وارد می‌کند، در سمت سرور هم validate و sanitize کن.
- در نمایش داده‌ها در فرانت ازtextContent یا نودهای متنی استفاده کن.
- CSPمناسب تنظیم کن و از unsafe-inline پرهیز کن.
- لاگ‌گذاری کن و رفتارهای مشکوک را مانیتور کن.

٤. فصل چهارم: اعتبارسنجی داده‌ها

٤،١. نقش فرانت‌اند و بک‌اند

اعتبارسنجی[15] داده‌ها یکی از مؤثرترین راه‌ها برای پیشگیری از ورود داده‌های مخرب یا نامعتبر به سیستم است. این فرآیند تضمین می‌کند که ورودی کاربر با معیارهای مورد انتظار (طول، نوع داده، قالب، کاراکترهای مجاز و ...) هم‌خوانی دارد.

- **در سمت فرانت‌اند**

هدف اصلی بهبود تجربه کاربری (UX) و کاهش بار سرور است. به‌عنوان مثال، بررسی اینکه فیلد ایمیل خالی نباشد یا طول کامنت از مقدار مشخصی بیشتر نشود. اما باید در نظر داشت که فرانت‌اند در کنترل کاربر است و به‌راحتی قابل دور زدن است.

- **در سمت بک‌اند**

این‌جا مهم‌ترین خط دفاعی قرار دارد. حتی اگر کاربر مرورگر را تغییر دهد یا درخواست ساختگی بفرستد، سرور باید قبل از ذخیره یا پردازش داده، همه ورودی‌ها را بررسی کند. نبود این لایه مساوی است با باز بودن در برای حملاتSQLi ، XSSو سایر سوءاستفاده‌ها.

به بیان ساده فرانت نمیتواند نیاز های امنیتی یک وبسایت را مهیا کند. از طرف دیگر اگر اعتبارسنجی در آن بخش وجود نداشته باشد. فشار و بار بر روی سرور بالا میرود و همینطور تجربه کاربری را ضعیف میکند. در نتیجه هر دو باید مکمل هم باشند، نه جایگزین یکدیگر.

٤،٢. چالش‌ها و الگوهای عملی در اعتبارسنجی داده‌ها

اعتبارسنجی داده‌ها صرفاً به بررسی طول رشته یا خالی نبودن فیلدها خلاصه نمی‌شود. در عمل، توسعه‌دهندگان با چالش‌های متنوعی مواجه هستند که باید برای هرکدام راهکار مشخص داشته باشند.

- تفاوت محیط‌ها و زبان‌ها

اعتبارسنجی در فرانت‌اند معمولا با زبان جاوااسکریپت انجام مشود در حالی که ممکن است زبان استفاده شده در بک‌اند متفاوت باشد. هماهنگ نگه داشتن قوانین اعتبارسنجی بین این دو محیط یک چالش جدی است. عدم هماهنگی باعث می‌شود کاربر چیزی را در فرانت معتبر ببیند ولی سرور آن را رد کند، یا بالعکس. اهکارهای مدرن مثل «اشتراک‌گذاری Schema»

- مرز Validation وSanitization

بسیاری از برنامه‌نویسان این دو مفهوم را اشتباه به‌کار می‌گیرند Validation بررسی می‌کند داده با قواعد از پیش تعیین‌شده همخوانی دارد یا نه، در حالی که Sanitization داده را به‌گونه‌ای تغییر می‌دهد که امن شود. مثلاً حذف

[15] Validation

تگ‌های HTML یا کاراکترهای خاص. رویه آن است که ابتدا داده اعتبارسنجی شود و سپس اگر مجاز شناخته شد، قبل از ذخیره یا نمایش Sanitization هم اعمال گردد.

- هزینه‌های کارایی[۱۶]

اعتبارسنجی سنگین در بک‌اند ممکن است باعث کندی پاسخ‌گویی شود، به‌ویژه در سامانه‌های با کاربر زیاد. در این حالت لازم است تعادلی بین «امنیت» و «کارایی» برقرار شود: قوانین ساده و پرکاربرد می‌توانند در سمت فرانت بررسی شوند، ولی قوانین حیاتی و حساس باید در بک‌اند اجرا شوند حتی اگر هزینه‌ی بیشتری داشته باشند.

- اهمیت تست و مانیتورینگ

هیچ مکانیزم اعتبارسنجی بدون آزمون ارزش ندارد. وجود تست‌های واحد و مانیتورینگ ورودی‌های غیرعادی، به تیم توسعه اطمینان می‌دهد که تغییرات بعدی در کد باعث از بین رفتن لایه‌های دفاعی نشود.

٤,٣. جمع‌بندی

اعتبارسنجی داده‌ها تنها یک فیلتر ساده نیست، بلکه بخشی جدایی‌ناپذیر از معماری امن وب‌اپلیکیشن محسوب می‌شود. این فرآیند اگر درست طراحی و پیاده‌سازی شود، هم امنیت سیستم را تضمین می‌کند و هم تجربه کاربری را بهبود می‌بخشد. با این حال، باید به چالش‌های همگام‌سازی قوانین بین فرانت و بک‌اند، تفاوت Validation و Sanitization ، و موازنه‌ی امنیت و کارایی توجه ویژه داشت. در نهایت، همان‌طور که OWASP توصیه می‌کند، هیچ‌گاه نباید اعتبارسنجی را به یک نقطه‌ی خاص محدود کرد؛ بلکه باید آن را بخشی از زنجیره‌ی دفاعی چندلایه در نظر گرفت.

[16] Performance Costs

۵. فصل پنجم: دفاع چندلایه در برابر حملات وب

۵،۱. چرا یک لایه کافی نیست؟

یکی از مهم‌ترین درس‌هایی که از سه فصل پیشین به دست می‌آید این است که هیچ مکانیزم امنیتی به‌تنهایی قادر به پوشش تمام تهدیدات نیست.

- اعتبارسنجی در فرانت‌اند تنها برای راحتی کاربر است و به راحتی دور زده میشود.
- پارامترایز و ORM در برابر حملات SQLi مقاوم هستند اما اگر داده بدون پاک‌سازی در HTML قرار گیرد، همچنان می‌تواند منجر به XSS شود.
- CSP جلوی اجرای اسکریپت‌های ناخواسته را می‌گیرد، اما مانع ورود داده‌ی مخرب به دیتابیس نمی‌شود.

به همین دلیل، امنیت وب باید به صورت **چندلایه** طراحی شود: هر لایه جلوی بخشی از تهدیدات را می‌گیرد و در صورت شکست یکی، لایه‌های دیگر همچنان مانع می‌شوند. این رویکرد «دفاع در عمق»[17] نامیده می‌شود.

۵،۲. ترکیب دفاع‌ها

۱) اعتبارسنجی(Validation & Sanitization)

- اولین سد دفاعی است؛ داده را بررسی می‌کند که از نوع، طول و قالب درست باشد.
- Sanitization کاراکترهای خطرناک مانند <script> را حذف یا بی‌اثر می‌کند.
- باید در **فرانت** برای UX و در **بک‌اند** برای امنیت جدی انجام شود.

۲) Prepared Statements / ORM

- برای همهٔ کوئری‌های پایگاه داده باید از پرس‌وجوهای پارامتری یا ORM استفاده شود.
- این لایه جلوی هرگونه تغییر ساختاری در کوئری را می‌گیرد، حتی اگر Validation فرانت دور زده شود.

۳) CSP[18]

- مرورگر را مجبور می‌کند فقط منابع اسکریپت مشخص‌شده را اجرا کند.
- باعث می‌شود حتی اگر یک اسکریپت به‌طور تصادفی در DOM وارد شود، مرورگر آن را اجرا نکند مگر اینکه در لیست سفید باشد.

[17] Defense in Depth
[18] Content Security Policy

ترکیب هر سه لایه:

۵/۳. سناریوی ترکیبی: حمله‌ی چندمرحله‌ای و نحوه‌ی جلوگیری

شرح سناریو:

فرض کنید مهاجم می‌خواهد در فرم کامنت، اسکریپت مخربی وارد کند تا هر بار کاربر دیگری صفحه را باز می‌کند، نشست[19] او را بدزدد.

اگر فقط یک لایه باشد:

- تنها Validation در فرانت؟ مهاجم با ابزار مانند Postman یا تغییر مرورگر، دادهی خام ارسال می‌کند.
- تنها Prepared Statement؟ جلوی SQLi گرفته می‌شود، اما کد جاوااسکریپت ذخیره‌شده در دیتابیس همچنان اجرا می‌شود.
- تنها CSP؟ جلوی بعضی اجرای اسکریپت را می‌گیرد، اما دادهٔ مخرب همچنان در سیستم باقی مانده و می‌تواند با دور زدن CSP اجرا شود.

با دفاع چندلایه:

- Validationدر بک‌اند جلوی ورود رشته‌های غیرمجاز را می‌گیرد.
- Prepared Statementمانع تزریق SQL می‌شود.
- Escapeو Sanitization خروجی جلوی اجرای HTML/JS ناخواسته را می‌گیرد.
- CSPجلوی اجرای هر اسکریپت احتمالی باقی‌مانده را می‌گیرد.

به این ترتیب، حتی اگر مهاجم یک لایه را شکست دهد، لایه‌های دیگر زنجیره را حفظ می‌کنند.

۵،٤. چک‌لیست عملی پیش از انتشار اپلیکیشن

برای اینکه مقاله خروجی کاربردی داشته باشد، چک‌لیست زیر می‌تواند راهنمای توسعه‌دهندگان باشد:

[19] session

A. اعتبارسنجی و پاک‌سازی داده‌ها
- در سمت فرانت: قوانین ساده (طول، فرمت ایمیل، خالی نبودن فیلدها).
- در سمت بک‌اند: قوانین قطعی مانند نوع، طول، لیست کاراکترهای مجاز و regex

B. استفاده از ORM یا Prepared Statement
- عدم استفاده از String Concatenation در کوئری‌ها.
- تست تمام ورودی‌های دیتابیس با داده‌های غیرمعمول.

C. پاک‌سازی خروجی
- به‌جای innerHTML از textContent یا safe rendering استفاده شود.

D. CSP و هدرهای امنیتی
- تعریف script-src بدون unsafe-inline.
- استفاده از HttpOnly، Secure و SameSite در کوکی‌ها

E. مانیتورینگ و تست مداوم
- ثبت همه‌ی خطاها و تلاش‌های ناموفق در ورود داده.
- بررسی دوره‌ای با ابزارهای OWASP ZAP یا Burp Suite.

۵،۵. جمع‌بندی فصل پنجم

امنیت وب با یک تکنیک حل نمی‌شود. همان‌طور که در این فصل نشان داده شد، تنها ترکیب چندین لایه‌ی دفاعی است که می‌تواند جلوی سناریوهای پیچیده‌ی حمله را بگیرد. این رویکرد «دفاع در عمق» تضمین می‌کند که شکست یک لایه به معنی سقوط کل سیستم نباشد. با اجرای Validation در دو سمت، استفاده از Prepared Statements ، Sanitizationدر خروجی و اعمال CSP در مرورگر، می‌توان سطح امنیت اپلیکیشن‌های وب را به‌طور چشمگیری ارتقا داد.

۶. فصل ششم: نتیجه‌گیری و مسیرهای آینده

٦،١. مرور دستاوردهای مقاله

در این مقاله سه تهدید رایج امنیت وب یعنی XSS[20]، SQLi[21] و ضعف در **اعتبارسنجی داده‌ها** بررسی شدند. ابتدا سازوکار هر حمله و نمونه‌های ساده‌ی وقوع آن در سناریوی «فرم ارسال کامنت» تشریح شد. سپس نسخه‌های ناامن و اصلاح‌شده‌ی کد، به‌همراه توضیح متنی ارائه گردید تا خواننده تفاوت‌ها را درک کند.

در ادامه نشان داده شد که امنیت یک وب‌اپلیکیشن تنها با یک تکنیک حاصل نمی‌شود و باید از **دفاع چندلایه** بهره گرفت: اعتبارسنجی ورودی‌ها، استفاده از کوئری‌های پارامتری، پاک‌سازی خروجی‌ها و اعمال سیاست‌های امنیتی مرورگر. این ترکیب،

[20] Cross-Site Scripting
[21] SQL Injection

احتمال موفقیت مهاجم را به حداقل می‌رساند و حتی در صورت شکست یک لایه، سایر لایه‌ها از سقوط کل سیستم جلوگیری می‌کنند.

۶،۲. اهمیت مستندسازی امنیت برای تیم‌های توسعه

یکی از نقاط ضعف بسیاری از پروژه‌های نرم‌افزاری، نبود مستندات روشن در حوزه‌ی امنیت است. کدی که بدون راهنمای امنیتی نوشته شود، دیر یا زود دچار آسیب‌پذیری می‌شود. بنابراین مستندسازی باید بخشی جدایی‌ناپذیر از چرخه‌ی توسعه باشد؛ به‌گونه‌ای که توسعه‌دهندگان تازه‌وارد بتوانند سیاست‌های امنیتی تیم را به‌سرعت یاد بگیرند و رعایت کنند.

مستندات امنیتی نه‌تنها در سطح فنی اهمیت دارند، بلکه در سطح مدیریتی نیز ارزشمندند. سازمان‌ها با داشتن چنین مستنداتی می‌توانند در برابر بازرسی‌های امنیتی و استانداردهای بین‌المللی مانند ISO 27001 یا NIST آمادگی بیشتری داشته باشند.

۶،۳. پیشنهاد برای پژوهش و پیاده‌سازی‌های آینده

این تحقیق به‌صورت هدفمند روی دو نوع حمله‌ی کلاسیک(SQLi و XSS) و یک لایه‌ی دفاعی عمومی (Validation) متمرکز شد. اما امنیت وب حوزه‌ای بسیار گسترده‌تر دارد. برای کارهای آینده می‌توان به موارد زیر پرداخت:

- **بررسی سایر حملات رایج وب** مانند CSRF (جعل درخواست میان‌وب‌گاهی) و Clickjacking
- مقایسه‌ی ابزارها و کتابخانه‌های Validation مانند Yup، Joi و Zod از نظر امنیت، کارایی و سهولت استفاده.
- تحلیل عملکرد CSP در اپلیکیشن‌های تک‌صفحه‌ای[۲۲] و تأثیر آن بر تجربه‌ی کاربر.
- **تست کارایی دفاع چندلایه** در پروژه‌های واقعی و اندازه‌گیری سربار پردازشی.
- **یکپارچه‌سازی امنیت در CI/CD** (خط تولید نرم‌افزار) به‌گونه‌ای که هر بار دیپلوی با اسکن امنیتی خودکار همراه باشد.

۶،۴. جمع‌بندی نهایی

امنیت وب یک موضوع ایستا و تمام‌شده نیست؛ بلکه فرآیندی پویا و مستمر است که باید همزمان با رشد فناوری و پیچیده‌تر شدن تهدیدات تکامل یابد. هدف این مقاله ایجاد پلی میان مفاهیم تئوری و نیازهای عملی توسعه‌دهندگان بود. با درک ریسک‌های SQL Injection وXSS ، و با یادگیری نقش اعتبارسنجی و دفاع چندلایه، توسعه‌دهندگان می‌توانند اپلیکیشن‌هایی بسازند که هم از نظر کاربرپسندی و هم از نظر امنیت قابل اتکا باشند.

۷. چک لیست:

با توجه به مباحث مطرح‌شده در فصل‌های قبل، می‌توان نتیجه گرفت که امنیت ورودی‌های کاربر تنها از طریق یک تکنیک خاص حاصل نمی‌شود، بلکه نیازمند مجموعه‌ای از اقدامات هماهنگ و چندلایه است. برای سهولت استفاده‌ی عملی از این یافته‌ها، در

[22] SPA

ادامه یک **چک‌لیست کاربردی** ارائه می‌شود. این چک‌لیست می‌تواند به‌عنوان راهنمای نهایی توسعه‌دهندگان در مراحل پیاده‌سازی، بازبینی کد و انتشار وب‌اپلیکیشن مورد استفاده قرار گیرد.

مرحله	اقدام اصلی	توضیح
۱	اعتبارسنجی در فرانت‌اند	بررسی طول، فرمت و نوع داده برای تجربه‌ی کاربری بهتر (قابل دور زدن، پس کافی نیست).
۲	اعتبارسنجی در بک‌اند	بررسی قطعی نوع داده، طول، الگو(Regex) ، و محدودسازی ورودی‌ها؛ تنها منبع مطمئن اعتبارسنجی.
۳	Sanitization (پاک‌سازی)	حذف یا بی‌اثرکردن کاراکترهای خطرناک قبل از ذخیره/نمایش مثلاً <script>
۴	کوئری های امن	استفاده از پارامتر به‌جای رشته‌سازی در SQL برای جلوگیری ازSQLi
۵	خروجی امن	به‌جای innerHTML از textContent استفاده شود.
۶	Content Security Policy	تعریف منابع مجاز برای اسکریپت‌ها و جلوگیری از اجرای inline scripts
۷	هدرهای امنیتی	Secure, SameSite HttpOnly برای کوکی‌ها؛ فعال‌سازی X-Content-Type-Options و X-Frame-Options.
۸	Principle of Least Privilege	دسترسی دیتابیس محدود: کاربر اپلیکیشن فقط مجوز SELECT/INSERT موردنیاز داشته باشد
۹	تست و مانیتورینگ	Unit Test برای ورودی‌ها، لاگ‌گذاری تلاش‌های مشکوک، اسکن دوره‌ای با OWASP
۱۰	مرور و مستند سازی	بررسی دوره‌ای کدها، نگارش راهنمای امنیتی تیم، و به‌روزرسانی سیاست‌ها.

در نهایت هدف این پژوهش ایجاد درکی عملی از تهدیدات امنیتی وب در میان توسعه‌دهندگان تازه‌کار است تا امنیت به بخشی جدایی‌ناپذیر از چرخه توسعه نرم‌افزار تبدیل شود.

پایان.

منابع:

[1]: OWASP Foundation - SQL Injection Prevention Guide

[2]: Mozilla Developer Network - Cross-Site Scripting (XSS)

[3]: Testing for SQL Injection - WSTG - Stable | OWASP Foundation
[4] MDN Web Docs - Input Validation and Sanitization Best Practices

ایمیل:
mhmdmhdybdy21@gmail.com
آدرس گیت‌هاب:
https://github.com/mehdiAB21