



# OWASP API Security Top 10 2019

۱۰ ریسک بحرانی امنیت API از منظر OWASP - ۲۰۱۹

## درباره OWASP

پروژه بازمتن امنیت وب اپلیکیشن‌ها (OWASP) جامعه ای باز و آزاد است که اختصاصا در حوزه توانمندسازی سازمان‌ها در حوزه توسعه، تهیه و ایجاد اپلیکیشن‌ها و API‌های قابل اعتماد فعالیت دارد. در OWASP، موارد زیر را بصورت رایگان و آزاد خواهید یافت:

- استانداردها و ابزارهای امنیت اپلیکیشن.
- کتاب‌هایی درباره تست امنیت اپلیکیشن‌ها، توسعه ایمن کد و بازیابی امنیت کد.
- ارائه‌ها و ویدئوها.
- راهنما و برگه تقلب برای بسیاری از موضوعات رایج.
- کنترل‌ها و کتابخانه‌های استاندارد در حوزه امنیت.
- شعب محلی در سرتاسر جهان.
- تحقیقات به روز و پیشرو در حوزه امنیت.
- کنفرانس‌های تخصصی در سرتاسر جهان.
- لیست‌های پست الکترونیک برای ارسال اخبار.

اطلاعات بیشتر در: <https://owasp.org>

تمامی ابزارها، مستندات، ویدئوها، ارائه‌ها و شعب OWASP رایگان بوده و استفاده از یا مشارکت در آنها برای کلیه افرادی که تمایل به بهبود امنیت اپلیکیشن‌ها دارند، آزاد است.

در OWASP امنیت اپلیکیشن بعنوان مساله‌ای مهم از منظر افراد، فرایندها و فناوری‌ها در نظر گرفته می‌شود چرا که موثرترین رویکردها در امنیت اطلاعات نیز به بهبود در این حوزه‌ها نیاز دارند.

OWASP تعریف جدیدی از سازمان ارائه می‌دهد. رهایی از بند فشار مسائل مالی امکان فراهم آوردن اطلاعات بیطرفانه، عملی و مقرون به صرفه در حوزه امنیت اپلیکیشن‌ها را به ما داده است.

OWASP به هیچ کمپانی فناوری وابستگی ندارد اگرچه از استفاده آگاهانه از فناوری‌های تجاری در حوزه امنیت نیز حمایت می‌کنیم. OWASP انواع مختلفی از اطلاعات را به گونه‌ای همکارانه، شفاف و باز ارائه می‌دهد.

بنیاد OWASP موجودیتی غیرانتفاعی و عام المنفعه است که توفیق بلند مدت پروژه OWASP را تضمین می‌نماید. تقریبا تمامی کسانی که با OWASP پیوند دارند، از قبیل اعضای هیئت مدیره، روسای شعبه‌ها، راهبران پروژه‌ها و اعضای پروژه‌ها داوطلبانه این همکاری را انجام می‌دهند.

همچنین ما از تحقیقات نوآورانه در حوزه امنیت با ارائه کمک‌های مالی و زیرساختی حمایت می‌کنیم. به ما بپیوندید!

۲	فهرست مطالب
۳	پیشگفتار
۴	I مقدمه
۵	RN یادداشت
۶	RISK ریسک های امنیتی API
۷	T10 ده آسیب پذیری بحرانی امنیت API از منظر OWASP – ۲۰۱۹
۸	API1:2019 مجوزدهی نادرست در سطح اشیاء
۱۰	API2:2019 احراز هویت نادرست کاربر
۱۳	API3:2019 افشای مفروض داده
۱۵	API4:2019 کمبود منابع و نبود محدودیت بر نرخ ارسال
۱۸	API5:2019 مجوزدهی نادرست در سطح توابع
۲۱	API6:2019 تخصیص جمعی
۲۴	API7:2019 پیکربندی امنیتی نادرست
۲۷	API8:2019 تزریق ورودی‌های مخرب
۳۰	API9:2019 مدیریت نادرست دارایی‌ها
۳۳	API10:2019 پایش و نظارت ناکافی
۳۵	+D گام بعدی برای توسعه دهندگان
۳۶	+DSO گام بعدی برای DevSecOps
۳۷	+DAT متدلوژی و داده
۳۸	+ACK سپاسگزاری‌ها

در دنیای مبتنی بر App امروز، یکی از ابعاد بنیادین نوآوری واسط برنامه نویسی اپلیکیشن<sup>1</sup> یا همان API ها هستند. از بانک ها گرفته تا خرده فروشی ها، حوزه حمل نقل، اینترنت اشیا، وسائل نقلیه خودران و شهرهای هوشمند، API ها بخشی حیاتی از اپلیکیشن های موبایل، وب و SaaS به شمار می آیند.

API ها ذاتا منطق اپلیکیشن و داده های حساسی از قبیل PII<sup>2</sup> (داده هایی که به تنهایی و بدون نیاز به داده اضافی دیگر، هویت یک کاربر را عیان می کنند نظیر شماره ملی) را در معرض دید قرار داده و در نتیجه، به طور روزافزون توجه بخش بیشتری از مهاجمین را به خود جلب می نمایند. بدون داشتن API هایی ایمن، توسعه سریع نوآوری های فناورانه، امکان پذیر نخواهد بود.

اگر چه کماکان می توان از لیست ده آسیب پذیری امنیتی بحرانی وب اپلیکیشن ها نیز برای امنیت API ها بهره برد، اما با توجه به ماهیت خاص API ها نیاز به لیستی از تهدیدات امنیتی مختص آنها احساس می شود. مقوله امنیت API بر راهکارها و استراتژی های لازم برای فهم و رفع آسیب پذیری ها و تهدیدات امنیتی خاص و منحصر به API ها تمرکز دارد.

اگر با پروژه [OWASP Top 10](#) آشنایی داشته باشید، شباهت هایی بین آن و مستند پیش رو خواهید یافت: هر دو با نیت فهم آسان توسط مخاطب و قابلیت بکارگیری و انطباق در سازمان تهیه شده اند. در صورتی که با مجموعه های OWASP Top 10 آشنایی ندارید، بهتر است پیش از رفتن به سراغ لیست اصلی، بخش های [ریسک های امنیتی API](#) و [متدلوژی و داده](#) از همین مستند را مطالعه نمایید.

با پرسش ها، نظرات و ایده های خود در [GitHub](#) پروژه می توانید در توسعه OWASP API Security Top 10 مشارکت کنید:

- <https://github.com/OWASP/API-Security/issues>
- <https://github.com/OWASP/API-Security/blob/master/CONTRIBUTING.md>

در اینجا می توانید OWASP API Security Top 10 را بیابید:

- [https://www.owasp.org/index.php/OWASP\\_API\\_Security\\_Project](https://www.owasp.org/index.php/OWASP_API_Security_Project)
- <https://github.com/OWASP/API-Security>

بدین وسیله از تمامی مشارکت کنندگان در این پروژه که با تلاش های خود در بوجود آمدن آن نقش داشته اند سپاسگزاریم. لیست تمامی آنها در قسمت [سپاسگزاری ها](#) قابل مشاهده است. متشکریم!

<sup>1</sup> Application Programming Interface

<sup>2</sup> Personally Identifiable Information

## به 2019 – OWASP API Security Top 10 خوش آمدید!

به اولین ویراست ده آسیب‌پذیری برتر امنیت API خوش آمدید. اگر با پروژه OWASP Top 10 آشنایی داشته باشید، شباهت‌هایی بین آن و مستند پیش رو خواهید یافت: هر دو با نیت فهم آسان توسط مخاطب و قابلیت بکارگیری و انطباق در سازمان تهیه شده‌اند. در غیر این صورت، پیش از مطالعه عمیق‌تر ریسک‌های بحرانی امنیت API بهتر است [صفحه ویکی پروژه امنیت API](#) را مطالعه نمایید.

API نقش خیلی مهمی در معماری اپلیکیشن‌های مدرن امروزی دارد. از آنجا که آگاهی بخشی امنیتی و نوآوری در این حوزه گام‌های مختلفی دارد، تمرکز بر نقاط ضعف رایج API‌ها اهمیت زیادی خواهد داشت.

هدف اصلی مستند و پروژه ده آسیب‌پذیری بحرانی امنیت API آموزش افراد دخیل در توسعه و نگهداری API‌ها از قبیل توسعه دهندگان، طراحان، معماران، مدیران و سازمان‌ها است.

در بخش [متدلوژی و داده](#)، اطلاعات بیشتری درباره نحوه ایجاد اولین نسخه از مستند حاضر خواهید یافت. در نسخه‌های آتی، جامعه امنیت را نیز دخیل نموده و به منظور دریافت داده‌های مرتبط، فراخوان عمومی خواهیم داد. در حال حاضر همگان را به مشارکت در [انبار داده Github](#) یا [لیست پست الکترونیک](#) ما از طریق ارسال سوال، نظر و پیشنهاد تشویق می‌کنیم.

مستند پیش رو اولین ویراست ده آسیب‌پذیری بحرانی امنیت API است و ما بنا بر آن داریم که بصورت دوره‌ای، هر سه یا چهار سال یکبار، آن را بروزرسانی نماییم.

بر خلاف نسخه حاضر، در نسخه‌های آتی برای دریافت داده‌های عمومی فراخوان داده و صنعت امنیت سایبری را نیز در تلاش خود سهیم خواهیم کرد. برای آشنایی بیشتر با نحوه آماده سازی این مستند می‌توانید به بخش [متدولوژی و داده](#) مراجعه نمایید. همچنین جزئیات ریسک‌های امنیتی مرتبط در بخش [ریسک‌های امنیتی API](#) قابل مطالعه هستند.

فهم تغییرات اساسی در معماری اپلیکیشن‌ها در سالیان گذشته از اهمیت زیادی برخوردار است. امروزه API‌ها نقشی کلیدی در معماری ریزسرویس‌ها<sup>1</sup>، اپلیکیشن‌های تک صفحه ای (SPA<sup>2</sup>)، اپلیکیشن‌های موبایل، اینترنت اشیا و ... دارند.

پروژه ده آسیب‌پذیری بحرانی امنیت API تلاشی ضروری برای آگاهی بخشی در حوزه مسائل امنیتی API‌های مدرن به شمار می‌رود که بدون تلاش‌های داوطلبانه افراد متعدد، که در بخش [سپاسگزاری‌ها](#) از تمامی آنان نام برده شده، به سرانجام رساندن آن امکان پذیر نبود. متشکریم!

<sup>1</sup> Microservices

<sup>2</sup> Single Page Application

به منظور تحلیل ریسک، از متدولوژی رتبه بندی ریسک OWASP استفاده شده است.

جدول زیر، واژگان مرتبط با رتبه ریسک را مختصراً نشان می‌دهد.

عوامل تهدید	قابلیت بهره برداری	میزان شیوع آسیب پذیری	قابلیت شناسایی آسیب پذیری	پیامد فنی	تأثیر بر کسب و کار
خاص API	آسان: ۳	گسترده: ۳	آسان: ۳	شدید: ۳	خاص کسب و کار
	متوسط: ۲	متداول: ۲	متوسط: ۲	متوسط: ۲	
	سخت: ۱	سخت: ۱	سخت: ۱	جزئی: ۱	

در این رویکرد، نوع فناوری مورد استفاده و احتمال وقوع آسیب پذیری در رتبه ریسک تأثیر ندارند؛ عبارت دیگر در این روش رتبه بندی ریسک، راهکار مورد استفاده برای پیاده سازی API، با رویکردی مستقل از جزئیات فناوری به ارزیابی ریسک می‌پردازد. هرکدام از عوامل یاد شده می‌تواند در پیداکردن و سواستفاده از یک آسیب پذیری به مهاجم کمک بسزایی کند. این رتبه بندی تأثیر واقعی بر کسب و کارها را نشان نداده و این سازمان‌ها هستند که با توجه به نوع کسب و کار و فرهنگ سازمانی خود، در میزان پذیرش خطر امنیتی استفاده از اپلیکیشن‌ها و API‌ها تصمیم گیرنده هستند. هدف از مستند ده آسیب پذیری بحرانی امنیت API، تحلیل ریسک نیست.

## مراجع

### OWASP

- [OWASP Risk Rating Methodology](#)
- [Article on Threat/Risk Modeling](#)

## خارجی

- [ISO 31000: Risk Management Std](#)
- [ISO 27001: ISMS](#)
- [NIST Cyber Framework \(US\)](#)
- [ASD Strategic Mitigations \(AU\)](#)
- [NIST CVSS 3.0](#)
- [Microsoft Threat Modeling Tool](#)



<b>API1: مجوزدهی نادرست در سطح اشیا</b>	APIها معمولا توابع مدیریت کننده شناسه های اشیا را در معرض دید قرار داده و سطح حمله <sup>۱</sup> گسترده ای را برای نقض کنترل دسترسی ایجاد می نمایند. کنترل های مجوزدهی در سطح اشیا بایستی در کلیه توابعی که با گرفتن ورودی از کاربر به منابع داده دسترسی دارند پیاده سازی شود.
<b>API2: احراز هویت نادرست کاربر</b>	مکانیزم های احراز هویت غالبا به درستی پیاده سازی نشده و سبب دسترسی مهاجمین به توکن های احراز هویت و ربایش موقت یا دائمی هویت سایر کاربران با استفاده از نقایص این مکانیزم ها می شوند. نقض توانایی سیستم در شناسایی کلاینت یا کاربر، منجر به نقض امنیت API خواهد شد.
<b>API3: افشای مفرط داده</b>	با بکارگیری سرویس های عمومی API، توسعه دهندگان عملا تمامی ویژگی های اشیا را بدون در نظر گرفتن حساسیت تک تک آنها و صرفا با تکیه بر فیلترینگ داده پیش از نمایش به کاربر، توسط کلاینت، در معرض دید عموم قرار می دهند.
<b>API4: کمبود منابع و نبود محدودیت نرخ در ارسال درخواست</b>	معمولا APIها هیچ محدودیتی بر اندازه یا تعداد منابع درخواستی توسط کلاینت یا کاربر اعمال نمی نمایند. این موضوع نه تنها با تاثیرگذاری منفی بر عملکرد سرور API می تواند منجر به حمله رد سرویس (DoS) شود، بلکه در را برای نقض احراز هویت از طریق حملاتی نظیر Brute Force نیز باز می گذارد.
<b>API5: مجوزدهی نادرست در سطح توابع</b>	مکانیزم های پیچیده کنترل دسترسی با سلسله مراتب، گروه ها و نقش های متفاوت و مرز نامشخص بین توابع عادی و مدیریتی سبب بروز نقایص مجوزدهی می شوند. با بهره برداری از این آسیب پذیری ها مهاجمین به منابع سایر کاربران و یا توابع مدیریتی دست خواهند یافت.
<b>API6: تخصیص جمعی</b>	پیوند دادن داده ارائه شده توسط کلاینت (نظیر اشیا JSON) با مدل های داده بدون فیلترکردن مناسب آنها بر مبنای یک لیست سفید می تواند منجر به تخصیص جمعی شود. با تشخیص ویژگی های اشیا، کاوش سایر توابع، خواندن مستندات یا ارائه ویژگی های اضافی برای اشیا در بدنه درخواست ها، مهاجم می تواند ویژگی هایی از اشیا که برای وی مجاز نیست را دستکاری نماید.
<b>API7: پیکربندی امنیتی نادرست پیامدی از بکارگیری پیکربندی نایمن پیشفرض، پیکربندی ناقص یا غیرمتمرکز، فضای ذخیره سازی ابری باز و محافظت نشده، سرایندهای HTTP با پیکربندی نادرست، متدهای غیرضروری HTTP، خط مشی های سهل انگارانه برای اشتراک گذاری منابع متقابل (CORS) و پیامهای خطای تفصیلی و مشروح می باشد.</b>	پیکربندی امنیتی نادرست پیامدی از بکارگیری پیکربندی نایمن پیشفرض، پیکربندی ناقص یا غیرمتمرکز، فضای ذخیره سازی ابری باز و محافظت نشده، سرایندهای HTTP با پیکربندی نادرست، متدهای غیرضروری HTTP، خط مشی های سهل انگارانه برای اشتراک گذاری منابع متقابل (CORS) و پیامهای خطای تفصیلی و مشروح می باشد.
<b>API8: آسیب پذیری های تزریق</b>	آسیب پذیری های مبتنی بر تزریق نظیر SQL، NoSQL، تزریق دستور و ... زمانی رخ می دهند که داده های نامطمئن بعنوان بخشی از یک دستور یا پرس و جو به مفسر تحویل داده شود. این داده مخرب می تواند مفسر را وادار به اجرای دستوری ناخواسته یا دسترسی غیرمجاز به داده ها نماید.
<b>API9: مدیریت نادرست دارایی ها</b>	APIها معمولا توابع بیشتری را نسبت به وب اپلیکیشن های سنتی در معرض دید قرار می دهند که این موضوع اهمیت مستندسازی مناسب و بروز را دوچندان می نماید. داشتن فهرستی از میزبان ها و نسخه های بکار گرفته شده API نقش مهمی در رفع آسیب پذیری های مرتبط با نسخ قدیمی API و توابع مرتبط با debugging ایفا می کند.
<b>API10: پایش و نظارت ناکافی</b>	پایش و نظارت ناکافی در کنار عدم وجود فرایند پاسخ دهی به وقایع <sup>۲</sup> یا پیاده سازی ناقص آن به مهاجم امکان تثبیت دسترسی، حمله به سایر سیستم ها و استخراج/نابودسازی داده ها را می دهد. مطالعات انجام شده بیانگر آن است که زمان آگاهی یافتن از نفوذ انجام شده به طور میانگین بیش از ۲۰۰ روز پس از انجام نفوذ بوده و تشخیص آن نیز بجای آنکه توسط فرایندهای درونی پایش و نظارت باشد توسط شرکت های ثالث صورت می پذیرد.

<sup>1</sup> Attack Surface

<sup>2</sup> Incident Response

پیامد		ضعف امنیتی		مسیر حمله		عوامل تهدید
خاص کسب و کار	پیامد فنی: ۳	قابلیت تشخیص: ۲	میزان شیوع: ۳	قابلیت بهره‌برداری: ۳	خاص API	
دسترسی غیرمجاز می‌تواند منجر به افشای اطلاعات به طرف‌های غیرمجاز، از دست رفتن داده یا دستکاری آن شود. همچنین دسترسی غیرمجاز به اشیاء می‌تواند سبب تحت کنترل گرفتن کامل حساب کاربری توسط مهاجم گردد.		این حمله رایج‌ترین آسیب‌پذیری API ها بوده و بیشترین پیامدها را نیز در پی دارد. مکانیزم‌های مجوزدهی و کنترل دسترسی در اپلیکیشن‌های مدرن، پیچیده و گسترده هستند. حتی اگر اپلیکیشن زیرساخت مناسب را برای کنترل‌های مجوزدهی پیاده‌سازی نماید، ممکن است توسعه دهندگان پیش از دسترسی به اشیاء حساس، استفاده از این کنترل‌ها را فراموش نمایند. تشخیص نقایص مربوط به کنترل دسترسی از طریق تست‌های ایستا یا پویا به صورت خودکار غالباً امکان پذیر نیست.		مهاجمین می‌توانند از نقاط و توابع آسیب‌پذیر (از منظر مجوزدهی نادرست در سطح اشیاء) با دستکاری شناسه شیء <sup>۱</sup> ارسالی درون درخواست سوءاستفاده و بهره‌برداری نمایند. این امر می‌تواند منجر به دسترسی غیرمجاز به داده حساس شود. دسترسی غیرمجاز به داده حساس، مساله‌ای رایج در اپلیکیشن‌های مبتنی بر API است چرا که مولفه سرور غالباً به طور کامل وضعیت کلاینت را رهگیری نمی‌کند و در عوض برای تصمیم‌گیری درباره دسترسی کلاینت به اشیاء از پارامترهایی نظیر شناسه شی که از سوی خود کلاینت ارسال می‌شوند، تکیه دارند.		

## آیا API از نظر مجوزدهی نادرست در سطح اشیاء<sup>۲</sup> آسیب‌پذیر است؟

مجوزدهی در سطح اشیاء مکانیزمی برای کنترل دسترسی است که غالباً در سطح کد پیاده‌سازی شده و دسترسی کاربر به اشیایی که بایستی به آنها دسترسی داشته باشد را تضمین می‌نماید.

هر تابعی در API که یک شناسه شی دریافت نموده و نوعی عملیات بر روی آن شی انجام می‌دهد، بایستی کنترل‌های مجوزدهی در سطح اشیاء را بکار گیرد. این کنترل‌ها باید دسترسی کاربر<sup>۳</sup> وارد شده به انجام عمل درخواستی بر روی شی درخواستی را اعتبارسنجی نمایند.

وجود ایراد و نقصان در این مکانیزم منجر به افشای اطلاعات غیرمجاز، تغییر یا از بین رفتن تمامی داده خواهد شد.

<sup>1</sup> Object ID

<sup>2</sup> Broken Object Level Authorization

<sup>3</sup> Logged-in User



## مثال‌هایی از سناریوهای حمله

### سناریو ۱#

یک پلتفرم تجارت الکترونیک، برای فروشگاه‌های آنلاین نمودارهای سود فروشگاه‌های میزبانی شده را در قالب یک لیست چندصفحه‌ای ارائه می‌دهد. مهاجم با بررسی درخواست‌های مرورگر، توابعی از API که نقش منبع داده برای نمودارهای مذکور را دارند و الگوی آنها به صورت `/shops/{shopName}/revenue_data.json` می‌باشد را شناسایی می‌کند. با استفاده از یک تابع دیگر API، مهاجم می‌تواند لیست نام کلیه فروشگاه‌های میزبانی شده را استخراج نماید. همچنین مهاجم با استفاده از یک اسکریپت ساده و جایگزین کردن `{shopName}` در URL خواهد توانست به داده‌ی فروش هزاران فروشگاه دسترسی یابد.

### سناریو ۲#

با پایش ترافیک شبکه‌ی یک گجت پوشیدنی<sup>۱</sup> درخواست HTTP PATCH زیر توجه مهاجم را به وجود سرآیند HTTP سفارشی `X-User-Id: 54796` جلب می‌نماید. با جایگزین کردن مقدار `X-User-Id` با `54795`، مهاجم پاسخ HTTP موفقیت آمیز گرفته و قادر به تغییر اطلاعات حساب سایر کاربران خواهد بود.

## چگونه از آسیب‌پذیری مجوزدهی نادرست در سطح اشیاء پیشگیری کنیم؟

- بکارگیری یک مکانیزم مجوزدهی که بر خط مشی و سلسله مراتب کاربری تمرکز دارد.
- استفاده از یک مکانیزم مجوزدهی برای بررسی اینکه آیا کاربر وارد شده مجوز لازم برای انجام عملیات درخواستی بر روی رکورد در تمامی توابعی که از کلاینت، ورودی می‌گیرند تا به رکورد مذکور در پایگاه داده دسترسی داشته باشند را دارا است یا خیر؟
- ارجحیت استفاده از مقادیر تصادفی و غیرقابل پیش بینی بعنوان GUID<sup>۲</sup> برای شناسه رکوردها.
- طراحی آزمون‌هایی برای ارزیابی صحت عملکرد مکانیزم‌های مجوزدهی.

## مراجع

## خارجی

- [CWE-284: Improper Access Control](#)
- [CWE-285: Improper Authorization](#)
- [CWE-639: Authorization Bypass Through User-Controlled Key](#)

<sup>1</sup> Wearable Device

<sup>2</sup> Globally Unique Identifier

پیامد		ضعف امنیتی		مسیر حمله		عوامل تهدید
خاص کسب و کار	پیامد فنی: ۳	قابلیت تشخیص: ۲	میزان شیوع: ۲	قابلیت بهره‌برداری: ۳	خاص API	
<p>مهاجمین می‌توانند به حساب‌های کاربری سایر کاربران دسترسی یافته، اطلاعات شخصی آنها را خوانده و عملیات حساس (نظیر نقل و انتقالات مالی و ارسال پیام‌های شخصی) را از طرف آنها انجام دهد.</p>		<p>در اینجا دو مساله وجود دارد:</p> <ol style="list-style-type: none"> <li>۱. نبود مکانیزم‌های حفاظتی: رفتار با نقاط و توابع مسئول احراز هویت در API بایستی متفاوت از سایر نقاط و توابع بوده و لایه‌های حفاظتی بیشتری داشته باشد.</li> <li>۲. پیاده‌سازی نادرست مکانیزم حفاظتی: مکانیزم حفاظتی بدون لحاظ کردن بردارهای حمله<sup>۱</sup> یا با موارد استفاده<sup>۲</sup> نادرست (مثلا بکارگیری مکانیزم احراز هویتی که برای IoT طراحی شده در وب اپلیکیشن‌ها) پیاده‌سازی یا استفاده شده‌اند.</li> </ol>		<p>احراز هویت در API ها مکانیزمی پیچیده و سردرگم کننده است. در نتیجه امکان دارد مهندسین نرم افزار و امنیت تصورات غلطی درباره حد و مرز احراز هویت و نحوه پیاده‌سازی آن داشته باشند. بعلاوه، مکانیزم احراز هویت هدفی بدیهی و آسان برای مهاجمان خواهد بود چرا که در معرض دید عموم قرار دارد. این دو نکته، مولفه احراز هویت را در مقابل بهره برداری‌ها و اکسپلویت‌های متعدد آسیب‌پذیر می‌سازد.</p>		

## آیا API از نظر احراز هویت نادرست کاربر<sup>۳</sup> آسیب‌پذیر است؟

نقاط، توابع و جریان‌های احراز هویت API دارایی‌هایی هستند که بایستی محافظت شوند. همچنین توابع «فراموشی گذرواژه یا بازیابی گذرواژه» نیز بایستی در زمره مکانیزم‌های احراز هویت در نظر گرفته شوند.

یک API از منظر احراز هویت نادرست کاربر آسیب‌پذیر است اگر:

- اجازه حمله [درج هویت](#)<sup>۴</sup> را بدهد که در آن مهاجم از لیستی از نام‌های کاربری و گذرواژه‌های معتبر استفاده می‌نماید.
- بدون استفاده از مکانیزم‌های CAPTCHA یا قفل کردن حساب کاربری<sup>۵</sup> اجازه حمله Brute Force روی یک حساب کاربری را بدهد.
- اجازه استفاده از گذرواژه‌های ضعیف را بدهد.
- جزئیات و داده‌های حساس مرتبط با احراز هویت از قبیل توکن‌های اصالت سنجی و گذرواژه‌ها را از طریق URL ارسال نماید.
- اصالت توکن‌ها را به‌بوته آزمون نگذارد.
- توکن‌ها JWT ضعیف یا بدون امضا ("alg": "none") را بپذیرد یا تاریخ انقضای آنها را اعتبارسنجی ننماید.
- از گذرواژه‌های آشکار<sup>۶</sup>، رمزگذاری نشده یا درهم سازی شده بصورت ضعیف<sup>۷</sup> استفاده نماید.
- از کلیدهای رمزگذاری ضعیف بهره‌برد.

<sup>1</sup> Attack Vectors

<sup>2</sup> Use Case

<sup>3</sup> Broken User Authentication

<sup>4</sup> Credential Stuffing

<sup>5</sup> Account Lockout

<sup>6</sup> Plaintext

<sup>7</sup> Weakly Hashed

## مثال‌هایی از سناریوهای حمله

## سناریو ۱#

درج هویت (استفاده از لیستی از نام‌های کاربری یا گذرواژه‌های شناخته شده) حمله‌ای رایج است. اگر اپلیکیشن از مکانیزم‌های حفاظتی خودکار در مقابل تهدیداتی نظیر درج هویت بهره نبرده باشد، آنگاه اپلیکیشن می‌تواند بعنوان یک پیشگوی گذرواژه<sup>۱</sup> یا آزمونگر جهت بررسی صحت اطلاعات هویتی جهت عبور از مکانیزم احراز هویت بکار رود.

## سناریو ۲#

مهاجم جریان بازیابی گذرواژه را با ارسال یک درخواست POST به `/api/system/verification-codes` و ارائه نام کاربری در بدنه پیام آغاز می‌کند. سپس یک توکن پیامک ۶ رقمی به تلفن قربانی ارسال می‌گردد. از آنجا که API خط مشی محدودیت سازی نرخ ارسال درخواست را بکار نگرفته، مهاجم می‌تواند تمامی جایگشت‌ها و ترکیبات محتمل را با استفاده از یک اسکریپت چنددخی<sup>۲</sup> با تابع زیر برای یافتن توکن صحیح ظرف چند دقیقه بیازماید.

`/api/system/verification-codes/{smsToken}`

## چگونه از آسیب‌پذیری احراز هویت نادرست کاربر پیشگیری کنیم؟

- حصول اطمینان از آنکه تمامی جریان‌های ممکن برای احراز هویت API (موبایل یا وب، سایر لینک‌هایی که از مکانیزم احراز هویت با یک کلیک و غیره) شناسایی شده است.
- مشورت با توسعه دهندگان و مهندسين در ارتباط با جریان‌های احراز هویتی که ممکن است از نظر دور مانده باشند.
- مطالعه و فهم کامل مکانیزم‌های احراز هویت استفاده شده در اپلیکیشن؛ بایستی در نظر داشت که OAuth و کلیدهای API نمی‌توانند بعنوان مکانیزمی برای احراز هویت به شمار آیند.
- در مساله احراز هویت، تولید توکن و ذخیره‌سازی گذرواژه، نباید چرخ را از ابتدا اختراع کرد بلکه بایستی از استانداردها استفاده نمود.
- توابع بازیابی فراموشی گذرواژه بایستی از منظر محافظت در مقابل Brute Force، محدودسازی نرخ و قفل شدن حساب کاربری هم ارز با توابع و نقاط ورود<sup>۳</sup> در نظر گرفته شود.
- از راهنمای احراز هویت OWASP استفاده شود.
- بکارگیری احراز هویت چندعاملی<sup>۴</sup>، در هر جا که امکان داشت.

<sup>۱</sup> Password Oracle

<sup>۲</sup> Multi-threaded

<sup>۳</sup> Login

<sup>۴</sup> Multi-factor Authentication

- بکارگیری مکانیزم‌های ضد Brute Force برای جلوگیری از حملات درج هویت، Dictionary و Brute Force بر روی توابع و نقاط احراز هویت در API. این مکانیزم بایستی سختگیرانه‌تر از مکانیزم محدودسازی نرخ معمول پیاده‌سازی شود.
- بکارگیری مکانیزم‌های [قفل کردن حساب کاربری](#) / CAPTCHA برای جلوگیری از حمله Brute Force علیه کاربران خاص.
- بکارگیری کنترل‌های مقابله با گذرواژه‌های ضعیف.
- کلیدهای API نبایستی برای احراز هویت کاربران بکار برده شود اما در عوض می‌تواند برای احراز هویت اپلیکیشن/پروژه [کلاینت](#) استفاده گردد.

### مراجع

#### OWASP

- [OWASP Key Management Cheat Sheet](#)
- [OWASP Authentication Cheatsheet](#)
- [Credential Stuffing](#)

### خارجی

- [CWE-798: Use of Hard-coded Credentials](#)

پیامد		ضعف امنیتی		مسیر حمله		عوامل تهدید
خاص کسب و کار	پیامد فنی: ۲	قابلیت تشخیص: ۲	میزان شیوع: ۲	قابلیت بهره برداری: ۳	خاص API	
	افشای مفرط و بیش از حد داده معمولاً منجر به افشای اطلاعات حساس می شود.	API ها برای فیلتر کردن داده به کلاینت ها اتکا می کنند. از آنجا که API ها به عنوان منابع داده استفاده می شوند، توسعه دهندگان گاهی آنها را بدون توجه به حساسیت اطلاعاتی که افشا می شود بکار می گیرند. ابزارهای خودکار غالباً نمی توانند این آسیب پذیری را کشف کنند چرا که تمایز دادن بین داده مجازی که توسط API بازگردانده می شود با داده حساسی که نباید توسط API بازگردانده شود بدون داشتن فهمی عمیق از اپلیکیشن امکان پذیر نیست.			بهره برداری از این آسیب پذیری آسان بوده و غالباً با شنود ترافیک به منظور تحلیل پاسخ های API برای یافتن داده حساسی که نباید به کاربر بازگردانده شود امکان پذیر است.	

## آیا API از نظر افشای مفرط داده<sup>۱</sup> آسیب پذیر است؟

طراحی API به گونه ای است که داده حساس را به کلاینت باز می گرداند. این داده غالباً پیش از ارائه و نمایش به کاربر در سمت کلاینت فیلتر می شود. در نتیجه مهاجم می تواند براحتمالی و با شنود ترافیک، این داده حساس را مشاهده نماید.

## مثال هایی از سناریوهای حمله

### سناریو ۱#

تیم توسعه موبایل از `/api/articles/{articleId}/comments/{commentId}` برای مشاهده و پردازش فراداده<sup>۲</sup> کامنت ها بهره می برد. با شنود ترافیک اپلیکیشن موبایل، مهاجم در می یابد که داده مرتبط با نویسنده کامنت نیز بازگردانده می شود. این موضوع به این دلیل است که پیاده سازی API از یک متد عمومی `toJSON()` برای سریالیزه کردن شیء `User` بهره می برد که این شی حاوی داده حساس PII<sup>۳</sup> می باشد.

### سناریو ۲#

یک سیستم نظارتی مبتنی بر IoT به مدیران خود اجازه ایجاد کاربرانی با سطوح مجوز مختلف می نماید. یکی از مدیران یک حساب کاربری برای یک نیروی حفاظت فیزیکی (نگهبان) جدید می سازد که بر مبنای آن تنها امکان دسترسی به ساختمان های مشخصی بایستی وجود داشته باشد. به محض استفاده نگهبان مذکور از اپلیکیشن موبایل خود، یک فراخوانی API به سوی `/api/sites/111/cameras` روانه می شود تا اطلاعات مرتبط با دوربین های موجود را دریافت نموده و آنها را در داشبورد خود نمایش دهد. پاسخ، لیستی از جزئیات دوربین ها با فرمت زیر را در بردارد.

```
{“id”:”xxx”,”live_access_token”:”xxxx-bbbbb”,”building_id”:”yyy”}
```

<sup>1</sup> Excessive Data Exposure

<sup>2</sup> Metadata

<sup>3</sup> Personally Identifiable Information

در حالیکه رابط گرافیکی کلاینت فقط دوربین‌هایی که نگهبان مذکور بایستی به آنها دسترسی داشته باشد را نشان می‌دهد، اما لیست کامل این دوربین‌ها در پاسخ API وجود دارد.

## چگونه از آسیب‌پذیری افشای مفرط داده پیشگیری کنیم؟

- عدم تکیه بر کلاینت در مساله فیلتر کردن داده حساس.
- بازیابی پاسخ دریافتی از API به منظور حصول اطمینان از آنکه فقط داده لازم و اصلی در آن نمایش داده می‌شود.
- پیش از افشا و در معرض دید عموم قرار دادن یک API، مهندسین توسعه دهندگان Back-End بایستی از خود بپرسند: مصرف‌کننده و مخاطب این داده چه کسی است؟
- اجتناب از استفاده از متدهای عمومی `to_json()` و `to_string()` و در عوض دستچین کردن تک‌تک ویژگی‌ها و مشخصه‌هایی که برای پاسخ ضروری هستند.
- طبقه‌بندی اطلاعات حساس و شخصی<sup>1</sup> ذخیره شده در API‌ها و بازیابی تمامی فراخوانی‌های API‌هایی که این اطلاعات را باز می‌گردانند به منظور کشف و شناسایی مواردی که ضعف امنیتی در پی دارند.
- بکارگیری یک مکانیزم اعتبارسنجی الگومحور برای بررسی اعتبار پاسخ‌ها بعنوان یک لایه امنیتی دیگر و همچنین تعریف و اعمال این مکانیزم بر روی داده بازگردانده شده تمامی API‌ها از جمله خطاها.

مراجع

خارجی

- [CWE-213: Intentional Information Exposure](#)

<sup>1</sup> Personally Identifiable Information



پیامد		ضعف امنیتی		مسیر حمله		عوامل تهدید
خاص کسب و کار	پیامد فنی: ۲	قابلیت تشخیص: ۳	میزان شیوع: ۳	قابلیت بهره‌برداری: ۲	خاص API	
	بهره‌برداری از این آسیب‌پذیری می‌تواند منجر به بروز DoS شده، در نتیجه API را از پاسخ به درخواست‌ها باز دارد و یا حتی آن را از دسترس خارج نماید.	یافتن APIهایی که محدودسازی نرخ ارسال را بکار نگرفته یا محدودیت‌های اعمال شده آنها ناکافی است، کار دشواری نیست.			بهره‌برداری از این آسیب‌پذیری نیاز به ارسال درخواست‌های ساده‌ای به سوی API دارد و به احراز هویت هم نیازی نیست. کافی است تعدادی درخواست هم‌زمان از یک ماشین و یا با استفاده از منابع رایانش ابری به سوی API ارسال گردد تا بتوان از این آسیب‌پذیری بهره برد.	

## آیا API از نظر کمبود منابع و نبود محدودیت بر نرخ ارسال<sup>۱</sup> آسیب‌پذیر است؟

درخواست‌های ارسال شده به سوی API منابعی از قبیل پهنای باند شبکه، پردازنده، حافظه و فضای ذخیره‌سازی را مصرف می‌کنند. مقدار منابعی که برای پاسخگویی به یک درخواست صرف می‌شود عمدتاً به ورودی‌های کاربر و منطق تجاری<sup>۲</sup> توابع API بستگی دارد. همچنین باید این موضوع را نیز در نظر داشت که درخواست‌های کلاینت‌های API مختلف برای دریافت منابع رقابت می‌کنند.

اگر دست‌کم یکی از محدودیت‌های زیر در سمت API به کلی اعمال نشده یا بطور نادرست (مثلاً بیش از حد زیاد یا بیش از حد کم) پیاده‌سازی شده باشد آنگاه API از منظر محدودیت یا کمبود نرخ ارسال، آسیب‌پذیر خواهد بود:

- Time Out اجرا<sup>۳</sup>
- حداکثر میزان حافظه قابل تخصیص
- تعداد توصیف‌گر<sup>۴</sup> فایل‌ها
- تعداد پردازنده‌ها
- اندازه محموله<sup>۵</sup> در درخواست‌ها (مثلاً در هنگام آپلود)
- تعداد درخواست‌ها به ازای کلاینت یا منبع
- تعداد رکوردهایی که به ازای یک درخواست در یک صفحه نمایش داده می‌شوند.

<sup>۱</sup> Lack of Resources and Rate Limiting

<sup>۲</sup> Business Logic

<sup>۳</sup> Execution Timeout

<sup>۴</sup> Descriptor

<sup>۵</sup> Payload

## مثال‌هایی از سناریوهای حمله

### سناریو ۱#

مهاجم از طریق ارسال یک درخواست POST به `/api/v1/images` اقدام به آپلود یک تصویر بزرگ می‌نماید. بعد از اتمام آپلود، API از روی تصویر آپلود شده تصاویر انگشتی<sup>۱</sup> متعددی با اندازه‌های مختلف ایجاد می‌نماید. به دلیل اندازه تصویر آپلودشده، حافظه‌ی در دسترس در خلال فرایند ایجاد تصاویر انگشتی تحت فشار قرار گرفته و API به وضعیت غیر پاسخگو<sup>۲</sup> می‌رسد.

### سناریو ۲#

اپلیکیشنی لیست کاربران را در UI با محدودیت ۲۰۰ کاربر در صفحه نمایش می‌دهد. لیست این کاربران از طریق ارسال پرس و جوی زیر از سرور دریافت می‌گردد: `/api/users?page=1&size=100`. در اینجا مهاجم می‌تواند با تغییر پارامتر `size` به ۲۰۰۰۰۰، مشکلاتی در عملکرد پایگاه داده پدید آورده و API را به وضعیت غیر پاسخگو برساند. در این حالت API قادر به پاسخگویی به هیچ درخواستی نخواهد بود (همان DoS).

همین سناریو را می‌توان به طریق مشابه برای ایجاد حملات سرریز Integer و سرریز Buffer استفاده نمود.

## چگونه از آسیب‌پذیری کمبود منابع و نبود محدودیت بر نرخ ارسال پیشگیری کنیم؟

- محدودسازی حافظه، پردازنده، تعداد دفعات راه اندازی مجدد، توصیف‌گرهای فایل و پردازنده‌ها با استفاده از Docker.
- اعمال محدودیت بر تعداد دفعاتی که در یک زمان مشخص امکان فراخوانی API وجود دارد.
- پس از رد شدن کلاینت از آستانه مجاز، این موضوع به همراه زمان رفع محدودیت به کلاینت اطلاع داده شود.
- افزودن اعتبارسنجی سمت سرور برای بررسی پارامترهای موجود در بدنه درخواست‌ها و رشته‌های پرس و جو، خصوصا مواردی که به نحوی با تعداد رکوردهای نمایش داده شده در پاسخ ارتباط دارند.
- تعریف و اعمال بیشینه اندازه داده (نظیر بیشینه طول برای رشته‌ها یا بیشینه تعداد عناصر در آرایه‌ها) در درخواست‌ها و محموله‌های ورودی.

<sup>1</sup> Thumbnail

<sup>2</sup> Unresponsive

## مراجع

### OWASP

- [Blocking Brute Force Attacks](#)
- [Docker Cheat Sheet - Limit resources \(memory, CPU, file descriptors, processes, restarts\)](#)
- [REST Assessment Cheat Sheet](#)

## خارجی

- [CWE-307: Improper Restriction of Excessive Authentication Attempts](#)
- [CWE-770: Allocation of Resources Without Limits or Throttling](#)
- *“Rate Limiting (Throttling)”* - [Security Strategies for Microservices-based Application Systems](#),

NIST

پیامد		ضعف امنیتی		مسیر حمله		عوامل تهدید
خاص کسب و کار	پیامد فنی: ۲	قابلیت تشخیص: ۱	میزان شیوع: ۲	قابلیت بهره‌برداری: ۲	خاص API	
چنین مشکلاتی منجر به دسترسی مهاجم به توابع غیرمجاز می‌شود. در این صورت توابع مدیریتی <sup>۲</sup> از جمله اهداف کلیدی مهاجم خواهند بود.		کنترل‌های مجوزدهی برای توابع یا منابع غالباً در سطح پیکربندی یا کد مدیریت می‌شوند. بکارگیری کنترل‌های مناسب می‌تواند گیج‌کننده باشد چرا که اپلیکیشن‌های مدرن امروزی غالباً دارای انواع مختلفی از نقش‌ها و گروه‌ها و سلسله مراتب کاربری هستند (مثلاً کاربران دارای بیش از یک نقش).		بهره‌برداری از این آسیب‌پذیری یعنی ارسال فراخوانی‌های API درست <sup>۱</sup> توسط مهاجم به سوی API Endpoint در ارتباط با فراخوانی‌هایی که مهاجم مجوز آنها را ندارد. این Endpointها ممکن است در معرض دید کاربران ناشناس، بدون مجوز یا عادی قرار داشته باشند. برای مهاجم تشخیص وجود چنین نواقصی در API آسان‌تر است چرا که ساختارمندتر بوده و نحوه دسترسی آنها به توابع، قابل پیش‌بینی‌تر است (مثلاً تغییر متد HTTP از GET به PUT یا تغییر رشته "users" در URL به "admins").		

## آیا API از نظر مجوزدهی نادرست در سطح توابع<sup>۳</sup> آسیب‌پذیر است؟

بهترین راه یافتن مشکلات مجوزدهی در سطح توابع، تحلیل عمیق مکانیزم مجوزدهی با لحاظ کردن سلسله مراتب کاربران، نقش‌ها و گروه‌های متفاوت موجود در اپلیکیشن و پرسیدن پرسش‌های زیر است:

- آیا کاربر عادی می‌تواند به توابع و نقاط مدیریتی در API دسترسی داشته باشد؟
- آیا کاربری می‌تواند عمل حساسی که مجوز انجام آن را ندارد (نظیر ایجاد، تغییر یا حذف) را صرفاً با تغییر متد HTTP (مثلاً از GET به DELETE) انجام دهد؟
- آیا کاربری از گروه X می‌تواند صرفاً با حدس زدن URLهای توابع و پارامترهای آن به مسیری (نظیر `/api/v1/users/export_all`) که فقط باید برای کاربران گروه Y قابل مشاهده باشد دسترسی یابد؟

بایستی در نظر داشت که عادی یا مدیریتی بودن یک تابع در API (همان API Endpoint) صرفاً بر مبنای مسیر URL تعیین نمی‌شود.

در حالیکه توسعه دهندگان بیشتر تمایل دارند که توابع مدیریتی را ذیل یک مسیر نسبی<sup>۴</sup> معین مانند `api/admin` قرار دهند، اما بسیار دیده می‌شود که این توابع مدیریتی در کنار توابع عادی در مسیرهایی نظیر `api/users` قرار داده شده‌اند.

<sup>1</sup> Legitimate

<sup>2</sup> Administrative Functions

<sup>3</sup> Broken Function Level Authentication

<sup>4</sup> Relative Path

## مثال‌هایی از سناریوهای حمله

### سناریو ۱#

در خلال فرایند ثبت نام در یک اپلیکیشن که فقط به کاربران دعوت شده اجازه عضویت می‌دهد، اپلیکیشن موبایل، یک فراخوانی API به `GET /api/invites/{invite_guid}` می‌فرستد. پاسخ دریافتی فایل JSON را دارا است که درون آن اطلاعات دعوتنامه‌ها شامل نقش کاربر و آدرس ایمیل وی دیده می‌شود.

مهاجم درخواست مذکور را ضبط کرده و متد HTTP را به `POST /api/invites/new` تغییر می‌دهد. این تابع تنها بایستی از طریق کنسول مدیریت و برای ادمین‌ها قابل دسترسی باشد که بعلت عدم بکارگیری کنترل‌های صحیح مجازدهی در سطح توابع اینگونه نیست.

در گام بعد مهاجم از این مساله بهره برداری کرده و برای خود دعوتنامه‌ای جهت ساخت یک اکانت ادمین می‌فرستد:

```
POST /api/invites/new
{"email":"hugo@malicious.com","role":"admin"}
```

### سناریو ۲#

یک API دارای تابعی است که فقط ادمین‌ها بایستی آن را ببینند:

```
GET /api/admin/v1/users/all
```

این تابع در پاسخ جزئیات تمامی کاربران اپلیکیشن را برگردانده و کنترل‌های مجازدهی در سطح توابع را نیز به درستی پیاده‌سازی نکرده است. مهاجمی که با ساختار API آشنایی پیدا کرده، این مسیر را حدس زده و اطلاعات حساس تمامی کاربران اپلیکیشن را می‌رباید.

## چگونه از آسیب‌پذیری مجازدهی نادرست در سطح توابع پیشگیری کنیم؟

ماژول مجازدهی اپلیکیشن بایستی بطور یکپارچه توسط تمامی توابع اپلیکیشن فراخوانی شده و تحلیل آن نیز آسان باشد. همچنین در بیشتر مواقع، این روش حفاظتی توسط یک یا چند مولفه بیرونی و خارج از کد اصلی اپلیکیشن فراهم می‌شود.

- مکانیزم (های) اعمال شده بایستی بطور پیشفرض کلیه دسترسی‌ها را Deny (رد) نموده و برای دسترسی به هر یک از توابع، مجوز خاص دسترسی نقش مربوطه را طلب نمایند.
- توابع API از منظر عیوب مجازدهی در سطح تابع با درنظر گرفتن منطق اپلیکیشن و سلسله مراتب گروه‌های کاربری مورد بازبینی قرار گیرد.

- تمامی کنترلگرهای مدیریتی از یک کنترلگر مدیریتی انتزاعی که مجوزها را بر حسب نقش کاربر یا گروه پیاده‌سازی نموده، ارث بری داشته باشند.
- تمامی توابع مدیریتی درون یک کنترلگر عادی (غیرمدیریتی)، کنترل‌های مجوز مبتنی بر نقش کاربر یا گروه را بکارگیرند.

## مراجع

### OWASP

- [OWASP Article on Forced Browsing](#)
- [OWASP Top 10 2013-A7-Missing Function Level Access Control](#)
- [OWASP Development Guide: Chapter on Authorization](#)

## خارجی

- [CWE-285: Improper Authorization](#)



پیامد		ضعف امنیتی		مسیر حمله		عوامل تهدید
خاص کسب و کار	پیامد فنی: ۲	قابلیت تشخیص: ۲	میزان شیوع: ۲	قابلیت بهره‌برداری: ۲	خاص API	
بهره برداری از این آسیب‌پذیری می‌تواند منجر به افزایش سطح دسترسی، دستکاری داده، عبور از مکانیزم‌های امنیتی و ... شود.		چارچوبهای جدید غالباً توسعه دهندگان را به استفاده از توابعی تشویق می‌کنند که بطور خودکار، ورودی‌های دریافتی از کلاینت را به متغیرهای کد و اشیاء داخلی آن پیوند می‌دهند. مهاجمین با سواستفاده از این متدلوژی می‌توانند به گونه ای اقدام به بروزرسانی یا بازنویسی ویژگی‌های اشیاء (داده) حساس نمایند که توسعه دهنده هیچگاه قصد افشای آن ویژگی‌ها را نداشته است.		بهره برداری از این آسیب‌پذیری غالباً نیاز به فهم منطق تجاری، روابط مابین اشیاء و ساختار API از سوی مهاجم دارد. بهره برداری از مقوله تخصیص جمعی در APIها ساده تر است چرا که در مرحله طراحی، پیاده‌سازی زیرین اپلیکیشن به همراه نام ویژگی‌های اشیاء افشا می‌شود و در معرض دید عموم قرار می‌گیرد.		

## آیا API از نظر تخصیص جمعی<sup>۱</sup> آسیب‌پذیر است؟

اشیا در اپلیکیشن‌های مدرن می‌توانند ویژگی‌های<sup>۲</sup> متعددی داشته باشند. برخی از این ویژگی‌ها بایستی مستقیماً توسط کلاینت قابل بروزرسانی باشند (مثلاً `user.first_name` یا `user.address`) در حالی که کلاینت نباید بتواند سایر ویژگی‌ها را دستکاری نماید (مثلاً پرچم `user.is_vip`).

یک تابع در API اگر بطور خودکار پارامترهای کلاینت را بدون لحاظ کردن حساسیت و سطح افشای<sup>۳</sup> ویژگی‌های آن، مستقیماً تبدیل به ویژگی‌های اشیاء داخلی نماید، از منظر تخصیص جمعی آسیب‌پذیر خواهد بود. این آسیب‌پذیری به مهاجم اجازه می‌دهد تا بتواند ویژگی‌هایی از اشیاء را که نباید به آنها دسترسی داشته باشد، بروزرسانی نماید.

نمونه‌هایی از «ویژگی‌های حساس» عبارتند از:

- ویژگی‌های مرتبط با مجوزها<sup>۴</sup>: پرچم‌هایی نظیر `user.is_admin` و `user.is_vip` فقط بایستی توسط ادمین‌ها تنظیم شوند.
- ویژگی‌های وابسته به فرایند<sup>۵</sup>: `user.cash` فقط باید بصورت داخلی و پس از تایید پرداخت بروزرسانی شود.
- ویژگی‌های داخلی: `article.created_time` فقط باید بصورت داخلی و توسط اپلیکیشن تنظیم گردد.

<sup>1</sup> Mass Assignment

<sup>2</sup> Properties

<sup>3</sup> Level of Exposure

<sup>4</sup> Permission-Related

<sup>5</sup> Process-Related

## مثال‌هایی از سناریوهای حمله

### سناریو #۱

یک اپلیکیشن هم‌سفری<sup>۱</sup> به کاربر امکان ویرایش اطلاعات پایه‌ای پروفایل خود را می‌دهد. در خلال این فرایند، یک فراخوانی API به `PUT /api/v1/users/me` با شی مجاز JSON زیر فرستاده می‌شود:

```
{"user_name":"inons","age":24}
```

اما درخواست `GET /api/v1/users/me` ویژگی اضافی `credit_balance` را نیز در خود دارد:

```
{"user_name":"inons","age":24,"credit_balance":10}
```

در اینجا مهاجم درخواست اول را با محتوای مخرب زیر بازارسال<sup>۲</sup> می‌نماید:

```
{"user_name":"attacker","age":60,"credit_balance":99999}
```

### سناریو #۲

یک پلتفرم اشتراک‌گذاری ویدئو به کاربران خود اجازه دانلود محتوا با فرمت‌های مختلفی را می‌دهد. مهاجم که در حال بررسی API است، در می‌یابد که `GET /api/v1/videos/{video_id}/meta_data` یک شیء JSON با ویژگی‌های ویدئو را باز می‌گرداند. یکی از این ویژگی‌ها، `"mp4_conversion_params":"-v codec h264"` است که نشان می‌دهد اپلیکیشن از یک دستور Shell برای تبدیل ویدئو بهره می‌برد.

همچنین مهاجم متوجه می‌شود که `POST /api/v1/videos/new` در برابر تخصیص جمعی آسیب‌پذیر بوده و به کلاینت اجازه می‌دهد که هریک از ویژگی‌های شیء ویدئو را با به صورت دلخواه تنظیم نماید. در نتیجه مهاجم مقدار مخربی را به صورت زیر در قست ویژگی ویدئو وارد می‌کند که در نتیجه آن با دانلود ویدئو با فرمت MP4 توسط مهاجم حمله تزریق دستور Shell<sup>۳</sup> اجرا خواهد شد.

```
"mp4_conversion_params":"-v codec h264 && format C:/"
```

## چگونه از آسیب‌پذیری تخصیص جمعی پیشگیری کنیم؟

- در صورت امکان، از بکارگیری توابعی که ورودی کلاینت را بصورت خودکار تبدیل به متغیرهای کد یا اشیای داخلی اپلیکیشن می‌کنند خودداری شود.
- تنها ویژگی‌های ضروری که باید توسط کلاینت بروزرسانی شوند در لیست سفید<sup>۴</sup> قرار گیرد.

<sup>۱</sup> Ride Sharing

<sup>۲</sup> Replay

<sup>۳</sup> Shell Command Injection

<sup>۴</sup> Whitelist

- از قابلیت‌های تعبیه شده در اپلیکیشن‌ها برای قراردادن ویژگی‌هایی که تغییر آنها برای کلاینت غیرمجاز است در لیست سیاه<sup>۱</sup> استفاده شود.
- در صورت امکان، الگوهای واضح و بدون ابهام برای داده ورودی تعریف و اعمال شود.

مراجع

خارجی

- [CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes](#)

---

<sup>1</sup> Blacklist

پیامد		ضعف امنیتی		مسیر حمله		عوامل تهدید
خاص کسب و کار	پیامد فنی: ۲	قابلیت تشخیص: ۳	میزان شیوع: ۳	قابلیت بهره‌برداری: ۳	خاص API	
	پیکربندی امنیتی نادرست نه تنها می‌تواند اطلاعات حساس کاربر را افشا کند بلکه جزئیاتی از سیستم که ممکن است به از دست رفتن کامل سرور منجر شود را نیز در معرض خطر قرار می‌دهد.	پیکربندی امنیتی نادرست می‌تواند در هر سطحی از API، از سطح شبکه تا سطح اپلیکشن روی دهد. ابزارهای خودکاری وجود دارند که فرایند تشخیص و بهره‌برداری از پیکربندی‌های نادرست نظیر تشخیص سرویس‌های غیرضروری را انجام می‌دهند.			مهاجمین غالباً در تلاش برای یافتن حفره‌های وصله نشده، توابع رایج یا فایل‌ها و مسیرهای محافظت نشده به منظور دسترسی غیرمجاز به سیستم هستند.	

## آیا API از نظر پیکربندی امنیتی نادرست<sup>۱</sup> آسیب‌پذیر است؟

API از منظر پیکربندی امنیتی نادرست آسیب‌پذیر است اگر:

- ایمن‌سازی امنیتی مناسب<sup>۲</sup> در هر قسمت از پشته اپلیکیشن رعایت نشده یا اپلیکیشن مجوزهای با پیکربندی نادرست روی سرویس‌های ابری داشته باشد.
- جدیدترین وصله‌های امنیتی نصب نشده و سیستم‌ها کاملاً بروز نباشند.
- ویژگی غیرضروری (نظیر افعال اضافی HTTP) فعال باشند.
- امنیت لایه انتقال (TLS) غیرفعال باشد.
- دستورات و الزامات امنیتی (نظیر [سرایندهای امنیتی](#)) به سوی کلاینت ارسال نشوند.
- خط مشی اشتراک متقابل منابع (CORS<sup>۳</sup>) وجود نداشته یا به درستی پیاده‌سازی نشده باشد.
- پیام‌های خطا ردپای پشته<sup>۴</sup> یا اطلاعات حساس دیگر را افشا نمایند.

## مثال‌هایی از سناریوهای حمله

### سناریو ۱#

مهاجم فایل `bash_history` را (که دستورات مورد استفاده تیم DevOps برای دسترسی به API را در خود دارد) در مسیر `root` سرور می‌یابد:

```
$ curl -X GET 'https://api.server/endpoint/' -H 'authorization: Basic Zm9vbmJhcG=='
```

همچنین مهاجم خواهد توانست توابعی از API که تنها توسط تیم DevOps مورد استفاده قرار گرفته و مستند نشده‌اند را نیز بیابد.

<sup>1</sup> Security Misconfiguration

<sup>2</sup> Hardening

<sup>3</sup> Cross-Origin Resource Sharing

<sup>4</sup> Stack Trace

## سناریو #۲

برای هدف قراردادن یک سرویس مشخص، مهاجم از موتورهای جستجوی رایج برای یافتن کامپیوترهایی که مستقیماً توسط اینترنت قابل دسترسی هستند بهره می‌برد. در نتیجه، مهاجم میزبانی را می‌یابد که از یک سیستم مدیریت پایگاه داده محبوب استفاده نموده و اقدام به شنود روی پورت پیشفرض آن dbms می‌کند. از آنجا که میزبان از پیکربندی پیشفرض (که احراز هویت را بطور پیشفرض غیرفعال نموده) استفاده کرده، مهاجم می‌تواند به میلیون‌ها رکورد حاوی PII، داده‌های احراز هویت و ... دست یابد.

## سناریو #۳

مهاجم با بررسی ترافیک یک اپلیکیشن موبایل متوجه می‌شود که تمامی ترافیک HTTP بر بستر یک پروتکل ایمن (نظیر TLS) منتقل نمی‌شود و این موضوع خصوصاً در زمان دالود تصاویر پروفایل صدق می‌کند. از آنجا که تعامل کاربر با اپلیکیشن دودویی<sup>۱</sup> است، علیرغم انتقال ترافیک API بر بستر پروتکلی ایمن، مهاجم خواهد توانست الگویی را در اندازه پاسخ API شناسایی نموده و از آن برای رهگیری ترجیحات کاربر<sup>۲</sup> در خصوص محتوای ارائه شده به اپلیکیشن (مثلاً تصاویر پروفایل) بهره ببرد.

## چگونه از آسیب‌پذیری پیکربندی امنیتی نادرست پیشگیری کنیم؟

چرخه حیات API بایستی شامل موارد زیر باشد:

- فرایندی تکرار شونده برای ایمن سازی API که منجر به پیاده‌سازی سریع و آسان یک محیط ایمن شود.
- فرایندی برای بازبینی و بروزرسانی پیکربندی‌ها در سراسر پشته API؛ این بازبینی بایستی موارد از جمله بازبینی هماهنگی بین فایل‌ها، مولفه‌های API و سرویس‌های ابری (نظیر مجوزهای باکتهای S3) را دربرگیرد.
- برقراری یک کانال ارتباطی ایمن برای دسترسی تمامی تعاملات API به دارایی‌های ایستا (نظیر تصاویر).
- فرایندی خودکار جهت ارزیابی پیوسته و مداوم اثربخشی پیکربندی و تنظیمات اعمال شده در سراسر محیط API و اپلیکیشن.

بعلاوه:

- برای جلوگیری از ارسال رهگیری رویدادهای استثنا و سایر داده‌های ارزشمند به مهاجم، در صورت امکان برای تمامی پاسخ‌های API (از جمله خطاها) الگوهای محموله<sup>۳</sup> مشخص تعریف و اعمال گردد.
- حصول اطمینان از اینکه API فقط به افعال HTTP مدنظر توسعه دهنده پاسخ می‌دهد و غیرفعال کردن سایر افعال (نظیر HEAD).

<sup>۱</sup> Binary

<sup>۲</sup> User Preferences

<sup>۳</sup> Payload Schema

- APIهایی که انتظار می‌رود دسترسی به آنها از طریق کلاینت‌های مبتنی بر مرورگر (مثلا فرانت WebApp) باشد، بایستی خط مشی CORS مناسب را بکار گیرند.

## مراجع

### OWASP

- [OWASP Secure Headers Project](#)
- [OWASP Testing Guide: Configuration Management](#)
- [OWASP Testing Guide: Testing for Error Codes](#)
- [OWASP Testing Guide: Test Cross Origin Resource Sharing](#)

## خارجی

- [CWE-2: Environmental Security Flaws](#)
- [CWE-16: Configuration](#)
- [CWE-388: Error Handling](#)
- [Guide to General Server Security](#) , NIST
- [Let's Encrypt: a free, automated, and open Certificate Authority](#)



پیامد		ضعف امنیتی		مسیر حمله	عوامل تهدید
خاص کسب‌وکار	پیامد فنی: ۳	قابلیت تشخیص: ۳	میزان شیوع: ۲	قابلیت بهره‌برداری: ۳	خاص API
آسیب‌پذیری تزریق ورودی‌های مخرب می‌تواند منجر به افشای اطلاعات و یا از دست رفتن اطلاعات شود. همچنین ممکن است این ضعف منجر به اختلال در سرویس‌دهی شده و یا حتی باعث از دست رفتن کامل دسترسی میزبان شود.		وجود آسیب‌پذیری تزریق ورودی‌های مخرب، امری متداول بوده و معمولاً در پرس و جوی‌های SQL، LDAP یا NoSQL، دستورات سیستم عامل، تجزیه کنندگان XML <sup>۱</sup> و ORM یافت می‌شود. این نقص‌ها به سادگی در زمان بازبینی کد منبع قابل کشف می‌باشند. مهاجمین نیز برای این منظور از اسکرپرها و Fuzzerها استفاده می‌کنند.		مهاجمین تلاش می‌کنند تا هرچه مسیر برای تزریق (از جمله، ورودی‌های مستقیم، متغیرها و سرویس‌های یکپارچه) وجود دارد را با داده‌های مخرب پر کنند و انتظار دارند این اطلاعات بدست لایه مفسر برسد.	

## آیا API از نظر نقص تزریق<sup>۲</sup> ورودی‌های مخرب آسیب‌پذیر است؟

API از منظر نقص تزریق ورودی‌های مخرب آسیب‌پذیر است اگر:

- API اطلاعات وارد شده توسط کاربر را اعتبار سنجی، فیلتر یا پاکسازی نکند.
- اطلاعات وارد شده توسط کاربر به صورت مستقیم استفاده شده و یا به انواع دستورات پرس و جو (SQL یا NoSQL یا LDAP) یا دستورات سیستم عامل، تجزیه‌کنندگان XML، ORM<sup>۳</sup> یا ORM<sup>۴</sup> افزوده شود.
- API اطلاعات دریافت شده از سیستم‌های خارجی را اعتبار سنجی، فیلتر یا پاکسازی نکند.

## مثال‌هایی از سناریوهای حمله

### سناریو ۱#

میان‌افزار یک دستگاه کنترل (فرزند) توسط والدین تابعی را در مسیر `/api/CONFIG/restore` ارائه می‌کند، که انتظار دارد مقدار `appId` ارسال شده به سمت آن، دارای مقادیر چند متغیره باشد. با استفاده از یک دیکامپایلر، مهاجم در می‌یابد مقدار `appId` بدون هیچ‌گونه تغییر یا پاکسازی به یک فراخوانی سیستمی<sup>۵</sup> ارسال می‌شود.

```
snprintf(cmd, 128, "%srestore_backup.sh /tmp/postfile.bin %s %d",
          "/mnt/shares/usr/bin/scripts/", appId, 66);
system(cmd);
```

دستور زیر به مهاجم این امکان را می‌دهد تا هر دستگاهی با این میان‌افزار آسیب‌پذیر را خاموش کند.

```
$ curl -k "https://${deviceIP}:4567/api/CONFIG/restore" -F
'appid=$(/etc/pod/power_down.sh)'
```

<sup>۱</sup> Parser

<sup>۲</sup> Injection

<sup>۳</sup> Object Relational Mapping

<sup>۴</sup> Object Document Mapper

<sup>۵</sup> System Call

## سناریو #۲

سامانه تحت وب ساده‌ای با عملکردهای اولیه 'CRUD'، برای انجام عملیات‌های رزرو وجود دارد. مهاجم موفق به شناسایی توزیع NoSQL از طریق متغیر bookingId در رشته پرس و جو و در درخواست حذف رزرو شده است. درخواست مذکور شبیه به DELETE /api/booking?bookingId=678 می‌باشد.

سرور API از تابع زیر برای رسیدگی کردن به درخواست‌های حذف استفاده می‌کند:

```
router.delete('/bookings', async function (req, res, next) {
  try {
    const deletedBooking = await Bookings.findOneAndRemove({_id':
:req.query.bookingId});

    res.status(200);
  } catch (err) {
    res.status(400).json({
      error: 'Unexpected error occurred while processing a request'
    });
  }
});
```

مانند آنچه در زیر مشاهده می‌کنید، مهاجم پس از رهگیری درخواست و تغییر مقدار bookingId استفاده شده در رشته پرس و جو، می‌تواند رزرو انجام شده توسط کاربر دیگری را حذف نماید.

DELETE /api/bookings?bookingId[\$ne]=678

## چگونه از آسیب‌پذیری توزیع ورودی‌های مخرب پیشگیری کنیم؟

جلوگیری از آسیب‌پذیری توزیع ورودی‌های مخرب، نیازمند جداسازی اطلاعات از دستورات و پرس و جوها می‌باشد.

- داده‌ها باید توسط یک کتابخانه پایدار، فعال و قابل اطمینان اعتبار سنجی شود.
- تمامی اطلاعات وارد شده توسط کاربران و دیگر سیستم‌های یکپارچه باید اعتبار سنجی، فیلتر یا پاکسازی شود.
- کاراکترهای خاص باید توسط قوانین مشخص برای مفسران نهایی تغییر داده شود.
- همواره تعداد رکوردهای بازگردانده شده باید محدود شود تا در صورت وجود نقص توزیع ورودی‌های مخرب، از افشای انبوه اطلاعات جلوگیری شود.
- داده‌های ورودی باید توسط فیلترهای مناسب اعتبار سنجی شود تا تنها مقادیر معتبر برای هر پارامتر ورودی مجاز به وارد شدن باشند.
- برای تمامی متغیرهای رشته‌ای، نوع داده و الگوی سخت‌گیرانه‌ای تعریف شود.

<sup>1</sup> Create, Read, Update, Delete

## مراجع

### OWASP

- [OWASP Injection Flaws](#)
- [SQL Injection](#)
- [NoSQL Injection Fun with Objects and Arrays](#)
- [Command Injection](#)

## خارجی

- [CWE-77: Command Injection](#)
- [CWE-89: SQL Injection](#)

پیامد		ضعف امنیتی		مسیر حمله		عوامل تهدید
خاص کسب‌وکار	پیامد فنی: ۲	قابلیت تشخیص: ۲	میزان شیوع: ۳	قابلیت بهره‌برداری: ۳	خاص API	
	مهاجم می‌تواند از طریق نسخه‌های قدیمی API که کماکان به پایگاه داده‌ی اصلی متصل هستند، به داده‌ی حساس و یا حتی سرور دسترسی یابد.	مستندات قدیمی و بروزرسانی نشده، امکان یافتن و یا رفع آسیب‌پذیری‌ها را دشوار می‌سازند. همچنین نبود فهرستی از دارایی‌ها و فقدان یک استراتژی مدون برای از دور خارج کردن نسخه‌های قدیمی منجر به وجود سیستم‌های وصله یا تعمیر نشده و نهایتاً نشت اطلاعات خواهد شد. امروزه با کمک مفاهیم نوینی نظیر میکروسرویس‌ها که امکان بکارگیری اپلیکیشن‌ها بصورت مستقل را تسهیل نموده‌اند (نظیر رایانش ابری، k8s یا کوبرنیتس و ...)، یافتن API‌هایی که به صورت غیرضروری در معرض دید همگان قرار دارند تبدیل به امری رایج و آسان شده است.			نسخه‌های قدیمی API غالباً اصلاح و بروزرسانی نشده‌اند و از آنجا که از مکانیزم‌های دفاعی نوین موجود در API‌های جدید بهره نمی‌برند، راهی آسان برای دسترسی به سیستم‌ها برای مهاجمین فراهم می‌سازند.	

## آیا API از نظر مدیریت نادرست دارایی‌ها<sup>۱</sup> آسیب‌پذیر است؟

در صورتی که یکی از شرایط زیر وجود داشته باشد، API آسیب‌پذیر خواهد بود:

- هدف از وجود API نامشخص بوده و پاسخی برای سوال‌های زیر وجود نداشته باشد:
- API در چه محیطی در حال اجرا است (مثلاً محیط تست، توسعه، اجرا<sup>۲</sup> یا عملیات<sup>۳</sup>)؟
- چه کسانی بایستی دسترسی شبکه‌ای به API داشته باشند (همه، افراد دخیل یا شرکا)؟
- چه نسخه‌ای از API در حال اجرا است؟
- چه داده‌ای (نظیر PII) توسط API در حال جمع‌آوری و پردازش است؟
- جریان داده به چه صورت است؟
- مستندی برای API وجود ندارد یا بروز نیست.
- برنامه‌ای برای بازنشستگی و از دور خارج شدن هریک از نسخه‌های API وجود ندارد.
- فهرست میزبان‌ها<sup>۴</sup> وجود ندارد یا قدیمی است.
- فهرست سرویس‌های یکپارچه<sup>۵</sup>، چه سرویس‌های متعلق به خود سازمان و چه سرویس‌های شرکت‌های ثالث، وجود ندارد یا قدیمی است.
- نسخه‌های قدیمی یا پیشین API بدون اصلاح و وصله شدن<sup>۶</sup> کماکان در حال اجرا هستند.

<sup>1</sup> Improper Asset Management

<sup>2</sup> Stage

<sup>3</sup> Production

<sup>4</sup> Host Inventory

<sup>5</sup> Integrated Service Inventory

<sup>6</sup> Patch

## مثال‌هایی از سناریوهای حمله

### سناریو ۱#

پس از بازطراحی یک اپلیکیشن، یک سرویس جستجوی Local وجود دارد که از یک نسخه قدیمی API (`api.someservice.com/v1`) به صورت محافظت نشده بهره می‌برد که در عین حال این API قدیمی به پایگاه داده کاربران دسترسی دارد. مهاجم که جدیدترین نسخه اپلیکیشن را به عنوان هدف در نظر گرفته، آدرس API (`api.someservice.com/v2`) را می‌یابد. جایگزینی v2 با v1 در URL سبب دسترسی مهاجم به API محافظت نشده و قدیمی می‌شود که در نتیجه‌ی آن، اطلاعات شناسایی شخصی (PII) بیش از ۱۰۰ میلیون کاربر افشا گردیده است.

### سناریو ۲#

یک شبکه اجتماعی از مکانیزم محدودسازی نرخ ارسال درخواست<sup>۱</sup> برای جلوگیری از انجام حملات Brute Force توسط مهاجمین جهت حدس توکن‌های تغییر گذرواژه بهره می‌برد. این مکانیزم نه به عنوان بخشی از کد API، بلکه به عنوان مولفه ای مابین کلاینت و API اصلی (در `www.socialnetwork.com`) پیاده‌سازی شده است. مهاجم یک نسخه بتا از میزبان API (`www.mbasic.beta.socialnetwork.com`) می‌یابد که از API یکسانی بهره می‌برد و رویه تغییر گذرواژه یکسانی دارد با این تفاوت که در آن هیچ مکانیزمی جهت محدودسازی نرخ درخواست تعبیه نشده است؛ در نتیجه مهاجم قادر خواهد بود که گذرواژه هر یک از کاربران را طی یک عملیات Brute Force ساده با حدس زدن یک توکن ۶ رقمی تغییر دهد.

## چگونه از آسیب‌پذیری مدیریت نادرست دارایی‌ها پیشگیری کنیم؟

- فهرستی از تمامی میزبان‌های API تهیه شده و جنبه‌های مهم هر کدام با تمرکز بر محیط API (محیط تست، توسعه، اجرا یا عملیات)، افراد مجاز به دسترسی شبکه‌ای به میزبان (همه، افراد دخیل یا شرکا) و نسخه API مستند شود.
- فهرستی از سرویس‌های یکپارچه تهیه شده و جنبه‌های مهم این سرویس‌ها نظیر نقش آنها، داده‌ی مبادله شده (جریان داده) و میزان حساسیت آنها مستند شود.
- تمامی جنبه‌های API نظیر نحوه احراز هویت، خطاها، ریدایرکت‌ها، محدودسازی نرخ درخواست، خط مشی‌های اشتراک گذاری منابع متقابل (CORS) و نقاط پایانی یا توابع (Endpointها) شامل پارامترها، درخواست‌ها و پاسخ‌ها مستند شوند.
- با بکارگیری و انطباق با استانداردهای باز، فرایند تولید مستند بطور خودکار انجام شده و این فرایند در CI/CD Pipeline تعبیه گردد.

<sup>1</sup> Rate-Limiting

- مستندات API در اختیار افرادی که مجاز به دسترسی به API هستند قرار گیرد.
- از مکانیزم‌های محافظتی خارجی از جمله فایروال‌های امنیت API برای محافظت از تمامی نسخه‌های در معرض دید API (نه فقط نسخه فعلی) استفاده گردد.
- از استفاده همزمان نسخه‌های عملیاتی شده<sup>۱</sup> و عملیاتی نشده<sup>۲</sup> API اجتناب شود. اگر این همزمانی اجتناب ناپذیر است، برای نسخه‌های عملیاتی نشده API نیز باید همان حفاظت‌های امنیتی نسخه‌های عملیاتی شده برقرار باشد.
- هنگامی که در نسخه‌های جدیدتر API بهبودهای امنیتی اعمال می‌شود، بایستی فرایند تحلیل ریسک نیز صورت پذیرد تا بتوان تصمیمات لازم در خصوص اقدامات جبرانی برای رفع مشکلات امنیتی نسخه‌های قدیمی‌تر را اتخاذ نمود. بعنوان نمونه، آیا می‌توان بدون تحت‌الشعاع قراردادن انطباق‌پذیری<sup>۳</sup> API بهبودهای امنیتی را در نسخه‌های قدیمی نیز وارد نمود یا اینکه بایستی تمامی نسخه‌های قدیمی به سرعت از دسترس خارج شده و تمامی کلاینت‌های مجبور به استفاده از آخرین نسخه شوند؟

## مراجع

## خارجی

- [CWE-1059: Incomplete Documentation](#)
- [OpenAPI Initiative](#)

---

<sup>1</sup> Production

<sup>2</sup> Non-Production

<sup>3</sup> Compatibility



پیامد		ضعف امنیتی		مسیر حمله		عوامل تهدید
خاص کسب و کار	پیامد فنی: ۲	قابلیت تشخیص: ۱	میزان شیوع: ۳	قابلیت بهره برداری: ۲	خاص API	
	بدون پایش فعالیت‌های مخربی که در حال انجام است، مهاجمین زمان زیادی برای نفوذ به سیستم‌ها خواهند داشت.	بدون ثبت وقایع و پایش آنها یا با ثبت و پایش ناکافی، رهگیری فعالیت‌های مخرب و پاسخ آنها در زمان مناسب تقریباً غیرممکن خواهد بود.		مهاجمین می‌توانند از فقدان فرایند ثبت وقایع و پایش برای سوءاستفاده پنهانی از سیستم‌ها بهره ببرند.		

## آیا API از نظر پایش و نظارت ناکافی<sup>۱</sup> آسیب پذیر است؟

در صورتی که یکی از شرایط زیر وجود داشته باشد، API آسیب پذیر خواهد بود:

- API هیچگونه Log ای تولید نکند، سطح ثبت وقایع<sup>۲</sup> به درستی تنظیم نشده باشد یا پیام‌های Log، حاوی جزئیات کافی نباشند.
- جامعیت<sup>۳</sup> Log ها تضمین نشده باشد (مثلاً [Log Injection](#) رخ دهد).
- Log ها به طور پیوسته پایش نشوند.
- زیرساخت API به طور پیوسته پایش نشود.

## مثال‌هایی از سناریوهای حمله

### سناریو ۱#

کلیدهای دسترسی به یک API مدیریتی در یک انبار<sup>۴</sup> عمومی افشا شده و در اختیار همگان قرار گرفته است. مالک انبار از طریق یک ایمیل از این افشای احتمالی مطلع می‌شود اما بیش از ۴۸ ساعت طول می‌کشد تا اقدام مقتضی انجام شود که در این حداصل، افشای کلید دسترسی ممکن سبب دسترسی غیرمجاز به داده حساس شده باشد. از آنجا که پایش کافی وجود نداشته است، سازمان نخواهد توانست بفهمد عوامل مخرب به چه داده‌ای دسترسی پیدا کرده‌اند.

### سناریو ۲#

یک پلتفرم اشتراک گذاری ویدئو با یک حمله درج هویت<sup>۵</sup> در مقیاسی بزرگ مواجه می‌شود. علیرغم آنکه تلاش‌های ناموفق ورود ثبت می‌شوند، اما هیچگونه هشداری در طول زمان حمله اعلام نشده است؛ بلکه تنها در واکنش به شکایت‌های کاربران، Log های API تحلیل و حمله کشف شده است. در نتیجه سازمان مجبور به صدور اعلامیه‌ای رسمی شده و از تمامی کاربران می‌خواهد که گذرواژه‌های خود را تغییر دهند. همچنین سازمان بایستی به مراجع نظارتی در خصوص این حادثه گزارش داده و پاسخگوی آنها باشد.

<sup>1</sup> Insufficient Monitoring

<sup>2</sup> Logging Level

<sup>3</sup> Integrity

<sup>4</sup> Repository

<sup>5</sup> Credential Stuffing

## چگونه از آسیب پذیری پایش و نظارت ناکافی پیشگیری کنیم؟

- تمامی تلاش های ناموفق احراز هویت، دسترسی های غیرمجاز و خطاهای اعتبارستجی ورودی<sup>1</sup> بایستی ثبت<sup>2</sup> شوند.
- Log ها باید به گونه ای تهیه شوند که توسط راهکارهای مدیریت Log قابل استفاده بوده و همچنین جزئیات کافی جهت شناسایی عامل مخرب را در خود داشته باشند.
- با Log ها بایستی به عنوان داده حساس رفتار شده و جامعیت آنها هم در زمان ذخیره سازی و هم در زمان انتقال تضمین شود.
- یک سیستم پایش پیکربندی و راه اندازی شود تا بتوان بطور مداوم و پیوسته عملکرد زیرساخت، شبکه و API را پایش نمود.
- از یک سیستم مدیریت رویدادها و اطلاعات امنیتی (SIEM) برای جمع و مدیریت Log های دریافتی از تمامی مولفه های پشته API و میزبان های آن استفاده شود.
- از Dashboard ها و هشدارها یا اعلان های سفارشی سازی شده به منظور تشخیص و پاسخ سریع به فعالیت های مشکوک استفاده شود.

## مراجع

### OWASP

- [OWASP Logging Cheat Sheet](#)
- [OWASP Proactive Controls: Implement Logging](#)
- [OWASP Application Security Verification Standard: V7: Error Handling and Logging Verification Requirements](#)

## خارجی

- [CWE-1059: Incomplete Documentation](#)
- [OpenAPI Initiative](#)

<sup>1</sup> Input Validation

<sup>2</sup> Log

وظایف مرتبط با ایجاد و نگهداری ایمن از نرم افزارها یا تعمیر نرم افزارهای موجود می تواند دشوار باشد و API ها نیز از قضیه مستثنی نیستند.

بر این باوریم که آموزش و آگاه سازی، گامی کلیدی در راستای نوشتن و توسعه نرم افزارهای ایمن هستند. تمامی الزامات دیگر در راستای نیل به هدف فوق به ایجاد و استفاده از فرایندهای امنیتی تکرارپذیر و کنترل های امنیتی استاندارد بستگی دارد.

OWASP منابع آزاد و رایگان متعددی برای پاسخ به مسائل امنیتی از ابتدای پروژه ایجاد نموده است. به منظور آشنایی با لیست جامع پروژه های در دسترس، [صفحه پروژه های OWASP](#) را ملاحظه نمایید.

آموزش	برای شروع می توان از <a href="#">پروژه مطالب آموزشی OWASP</a> بسته به علاقه و نوع حرفه آغاز نمود. برای آموزش عملیاتی، <sup>1</sup> crAPI را نیز به <a href="#">نقشه راه</a> خود افزوده ایم. تست های مربوط به WebAppSec را می توان با <a href="#">OWASP DevSlop Pixi Module</a> که یک WebApp و سرویس API آزمایشگاهی آسیب پذیر است، انجام داد. استفاده از چنین ابزارهایی سبب یادگیری نحوه تست وب اپلیکیشن ها و API های مدرن از منظر مسائل امنیتی و چگونگی توسعه API های مدرن در آینده خواهد شد. همچنین امکان شرکت در جلسات آموزشی <a href="#">کنفرانس AppSec</a> و عضویت در <a href="#">شعب محلی OWASP</a> نیز برای علاقه مندان وجود دارد.
الزامات امنیتی	امنیت باید بعنوان بخشی تفکیک ناپذیر در تمامی پروژه ها از ابتدا در نظر گرفته شود. در هنگام استخراج الزامات امنیتی، باید معنی واژه «ایمن» برای هر پروژه مشخصا تعریف شود. OWASP استفاده از <a href="#">استاندارد امنیت سنجی اپلیکیشن (ASVS)</a> را بعنوان راهنمایی برای تعیین الزامات امنیتی توصیه می کند. در صورت برون سپاری نیز، استفاده از <a href="#">ضمیمه قرارداد نرم افزار ایمن OWASP</a> (که بایستی با قوانین و رگولاتوری های محلی انطباق یابد) می تواند انتخاب مناسبی باشد.
معماری امنیتی	امنیت بایستی در تمامی مراحل توسعه پروژه ها اهمیت داشته باشد. <a href="#">برگه های راهنمای پیشگیری OWASP</a> نقطه شروع مناسبی برای چگونگی طراحی ایمن در خلال فاز طراحی معماری به شمار آید. همچنین <a href="#">برگه راهنمای امنیت REST</a> و <a href="#">برگه راهنمای ارزیابی REST</a> نیز گزینه های مناسبی در این راستا هستند.
کنترل های امنیتی استاندارد	بکارگیری و انطباق با کنترل های امنیتی استاندارد ریسک ایجاد ضعف های امنیتی در خلال ایجاد برنامه ها با منطق سازمانی را کاهش می دهد. علیرغم اینکه بسیاری از چارچوب های مدرن امروزی با استانداردهای توکار و موثر امنیتی توزیع می شوند، اما <a href="#">کنترل های پیشگیرانه و فعال OWASP</a> دید خوبی از کنترل هایی که باید در پروژه ها لحاظ شوند بدست می دهد. OWASP کتابخانه و ابزارهای متعددی از جمله در حوزه کنترل های اعتبارسنجی در اختیار عموم قرار می دهد که می توانند مفید باشند.
چرخه حیات توسعه نرم افزار ایمن	به منظور بهبود فرایندها در هنگام ایجاد و ساخت API ها می توان از <a href="#">مدل ضمانت کمال نرم افزار OWASP (SAMM)</a> بهره برد. همچنین پروژه های متعدد دیگری نیز در OWASP وجود دارند که می توانند در فازهای مختلف توسعه API مفید باشند که از جمله آنها می توان، <a href="#">پروژه بازبینی کد OWASP</a> را نام برد.

<sup>1</sup> Completely Ridiculous API

با توجه به اهمیت API ها در معماری اپلیکیشن های جدید، ایجاد API های ایمن امری حیاتی می باشد. مقوله امنیت را نمی توان نادیده گرفت و باید آن را جزئی از کل چرخه توسعه اپلیکیشن در نظر گرفت. انجام اسکن و تست نفوذ، آن هم به صورت سالیانه به هیچ عنوان کافی نمی باشد.

DevSecOps باید به فرایند توسعه افزوده شده و در تمام زمان های توسعه نرم افزار، انجام تست های امنیتی مداوم را تسهیل کند. هدف آنها بهره گیری از خودکارسازی فرایندهای امنیتی در جهت بهبود فرایند تولید نرم افزار بوده به شکلی که تاثیری بر سرعت توسعه نداشته باشد. اگر شک دارید، [مانیفست DevSecOps](#) را بررسی کنید تا در جریان باشید.

اولویت تست ها از مدل تهدیدات بدست می آید. اگر شما مدل تهدیدات ندارید می توانید از <a href="#">OWASP Application Security Verification Standard (ASVS)</a> و <a href="#">OWASP Testing Guide</a> به عنوان ورودی استفاده کنید. همچنین مشارکت دادن تیم توسعه می تواند باعث شود آنها نسبت به موضوعات امنیتی آگاه تر شوند.	درک مدل تهدیدات <sup>1</sup>
تیم توسعه را به فرایند اضافه کنید تا آنها نیز درک بهتری از چرخه توسعه نرم افزار پیدا کنند. مشارکت شما در انجام تست های مداوم امنیتی باید همراستا با افراد، فرایندها و ابزارها باشد. همه باید با فرایند موافق باشند تا هیچ گونه اصطکاک و مقاومتی وجود نداشته باشد.	درک چرخه توسعه نرم افزار
با توجه به اینکه کار شما نباید تاثیری بر سرعت توسعه داشته باشد. بنابراین باید خیلی آگاهانه بهترین تکنیک (ساده، سریع ترین و دقیق ترین) را برای تایید الزامات امنیتی انتخاب کنید. <a href="#">OWASP Security Knowledge Framework</a> و <a href="#">OWASP Application Security Verification Standard</a> می توانند منابع خوبی برای الزامات عملکردی و غیر عملکردی باشند. منابع خوب دیگری از <a href="#">پروژه ها</a> و <a href="#">ابزارها</a> مشابه با مواردی که توسط <a href="#">DevSecOps community</a> پیشنهاد می شود، وجود دارد.	راہبرد انجام تست
شما پلی هستید بین تیم توسعه دهنده و پیاده سازی، برای اینکه به این مهم دست یابید نه تنها باید بر روی عملکرد و قابلیت ها تمرکز کنید بلکه باید به هماهنگی نیز توجه کنید. از ابتدا به صورت نزدیک با هر دو تیم توسعه و پیاده سازی کار کنید تا بتوانید زمان و تلاش تان را بهینه نمایید. شما باید برای حالتی که الزامات امنیتی به صورت مداوم بررسی شوند، هدف گذاری کنید.	دستیابی به جامعیت و دقت
با کمترین اصطکاک یا بدون اصطکاک مشارکت داشته باشید. یافته ها را در بازه زمانی مشخص و در قالب ابزارهای مورد استفاده توسط تیم توسعه (نه فایل های PDF) تحویل دهید. به تیم توسعه اضافه شوید تا یافته ها را به آنها نشان دهید. از این فرصت برای آموزش آنها استفاده کنید، به صورت شفاف در مورد نقطه ضعف و روش های سوء استفاده از آن (که شامل سناریوهای حملات می باشند) توضیح دهید تا واقعی به نظر برسد.	به وضوح یافته ها را به اشتراک بگذارید

<sup>1</sup> Threat Model

## بررسی اجمالی

از آنجا که صنعت AppSec مشخصاً بر امنیت اپلیکیشن‌های معماری نوین که در آنها API نقشی حیاتی دارد، تمرکز ننموده، ایجاد لیستی از ده ریسک امنیتی بحرانی امنیت API بر مبنای فراخوان عمومی کاری سخت خواهد بود. علیرغم اینکه فراخوانی برای داده‌های عمومی داده نشده، اما لیست فعلی بر مبنای داده‌های در دسترس عموم، مشارکت کارشناسان امنیتی و نظرات متخصصان حوزه امنیت، تهیه گردیده است.

## متدلوژی و داده

در فاز اول، داده‌های در دسترس عموم در حوزه رخدادهای مرتبط با امنیت API توسط گروهی از متخصصین امنیت جمع آوری، بازبینی و دسته بندی شدند. این داده‌ها از پلتفرم‌های شکار باگ<sup>1</sup> و پایگاه‌های داده آسیب‌پذیری در یک چارچوب زمانی یک ساله به منظور تحلیل آماری جمع آوری شده اند.

در فاز بعد، از متخصصین امنیت با سویه عملیاتی و تجربه تست نفوذ خواسته شد تا آنان نیز لیست ده ریسک امنیتی بحرانی API از منظر خود را با گروه به اشتراک گذارند.

به منظور انجام فرایند تحلیل ریسک از [متدلوژی رتبه بندی ریسک OWASP](#) استفاده و نتایج آن نیز توسط متخصصین امنیتی بازبینی قرار گرفت. برای مطالعه بیشتر در این حوزه به بخش [ریسک‌های امنیتی API](#) مراجعه نمایید.

پیش نویس اولیه ده ریسک امنیتی بحرانی API ها در ۲۰۱۹ از منظر OWASP از اجماع بین نتایج آماری فاز اول و لیست مدنظر متخصصین بدست آمده است و سپس به منظور بازبینی مجدد در اختیار گروه دیگری از متخصصین (با تجربه مرتبط در حوزه امنیت API) قرار گرفته است.

مستند ده ریسک امنیتی بحرانی API ها در ۲۰۱۹ از منظر OWASP اولین بار در رویداد جهانی OWASP AppSec در تل‌آویو (می ۲۰۱۹) ارائه شده و پس از آن برای بحث و مشارکت عموم در GitHub قرار گرفت.

لیست مشارکت کنندگان در بخش [سپاسگزاری‌ها](#) قابل مشاهده است.

---

<sup>1</sup> Bug Bounty

## سیاسگزاری از مشارکت کنندگان

بدینوسیله از تمامی مشارکت کنندگانی که به طور عمومی در GitHub و به سایر طرق در توسعه این مستند نقش داشته‌اند تشکر می‌نماییم.

- |                    |                    |
|--------------------|--------------------|
| • 007divyachawla   | • Keith Casey      |
| • Abid Khan        | • kozmic           |
| • Adam Fisher      | • LauraRosePorter  |
| • anotherik        | • Matthieu Estrade |
| • bkimminich       | • Mr-Listener      |
| • caseysoftware    | • nathanawmk       |
| • Chris Westphal   | • PauloASilva      |
| • dsopas           | • pentagramz       |
| • DSotnikov        | • philippederyck   |
| • emilva           | • pleothaud        |
| • ErezYalon        | • r00ter           |
| • flascelles       | • Raj kumar        |
| • Guillaume Benats | • RNPG             |
| • IgorSasovets     | • Sagar Popat      |
| • Inonshk          | • Stephen Gates    |
| • JonnySchnittger  | • This-is-neo      |
| • jmanico          | • Thomaskonrad     |
| • jmdx             | • xycloops123      |

## ترجمه فارسی (Farsi Translation)

این ترجمه با حمایت شرکت راسپینا نت پارس تهیه شده است. استاندارد OWASP Security API Top 10 می‌تواند به عنوان مرجع راهنما در توسعه ایمن API و همچنین مرجع بررسی در فرایند آزمون نفوذ پذیری مورد استفاده قرار گیرد.



## مترجمین (Translators)

محمد رضا اسمعیلی طبا (Mohammad Reza Ismaeli Taba)

علیرضا مستمع (Alireza Mostame)

## ویراستار (Editor)

امیرمهدی نوبخت (Amirmahdi Nowbakht)