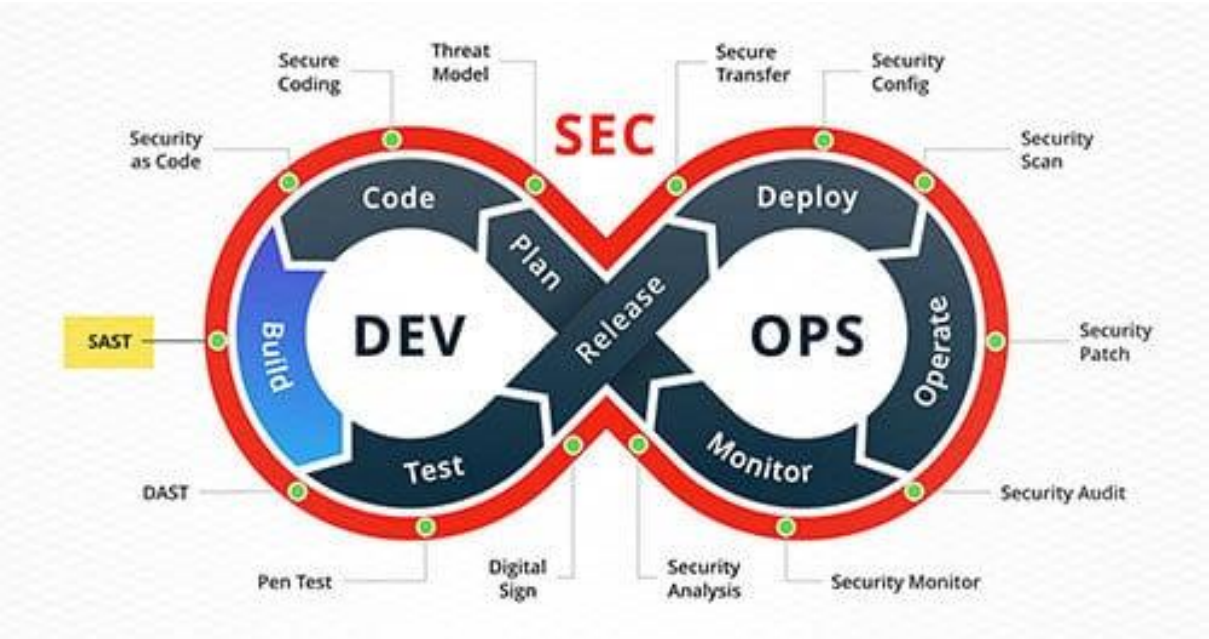
Araştırma Konusu

SAST (Static Application Security Testing)

Statik Kod Analizi ve Güvenlik Testleri



Sunum Ekibi

200290010- Muhammed ATMACA

200290032- Yunus Emre ERTÜRK

200290018- Ekrem KAVAK

210290018- Fatih ÖZVEREN

210290045 - Furkan KELEŞ

1. SAST Nedir? Amacı Nedir? Diğer Güvenlik Testlerinden Farkları Nelerdir? Tarihçesi Ve Evrimi.

1. SAST'in Önemi

Günümüzde yazılım güvenliği giderek daha kritik hale gelmektedir. SAST, yazılım geliştirme sürecinin erken aşamalarında güvenlik açıklarını tespit ederek, kötü niyetli saldırılara karşı önlem alınmasına yardımcı olur. Bu da hem kullanıcıların verilerinin korunmasını sağlar hem de şirketler için itibar ve finansal zararların önüne geçer.

2. SAST Nedir?

- **Kavramsal Tanım:** Statik Kod Analizi ve Güvenlik Testleri (SAST), yazılımın kaynak kodunu analiz ederek potansiyel güvenlik açıklarını tespit etmeye odaklanan bir güvenlik testi yöntemidir. Kodun derleme zamanında incelenmesi, kodun çalışma zamanındaki davranışı değil, kodun kendisindeki hataları belirlemeyi amaçlar.
- **Temel İlkeleri:** SAST'in temel prensipleri arasında otomatik analiz, statik tarama, kod analizi ve kod karmaşıklığının değerlendirilmesi gibi unsurlar bulunur. Bu prensipler, SAST'in etkinliğini sağlamak için temel yapı taşlarıdır.

3. SAST'in Amacı

- **Yazılım Güvenliği ve SAST İlişkisi:** Yazılım güvenliği, modern bilgi teknolojisinin temel taşlarından biridir. SAST, yazılım geliştirme sürecinin erken aşamalarında güvenlik açıklarını tespit ederek, yazılımın güvenilirliğini artırmayı hedefler. Böylece, kötü niyetli saldırıları önlemek ve kullanıcıların güvenliğini sağlamak mümkün olur.

4. Diğer Güvenlik Testlerinden Farkları

- **Dinamik Testler ile Karşılaştırma:** SAST, kodun statik yapısını analiz ederken dinamik testler, kodun çalışma zamanı davranışını inceler. SAST, kodun doğası gereği hata yapma potansiyelini değerlendirirken, dinamik testler, kodun gerçek dünya koşullarında nasıl davranacağını test eder.
- **Manuel Kod İncelemesi ile Karşılaştırma:** Manuel kod incelemesi, insan denetimi gerektiren ve zaman alıcı bir süreçtir. SAST ise bu süreci otomatikleştirerek daha hızlı ve daha kapsamlı bir analiz sağlar. Ayrıca, SAST'in tekrarlanabilirliği ve ölçeklenebilirliği, manuel incelemelerde sağlanamayan avantajlardır.

5. SAST'in Tarihçesi ve Evrimi

- **Başlangıç Noktası ve İlk Uygulamalar:** SAST'in kökenleri, bilgisayar güvenliği alanındaki ihtiyaçlar doğrultusunda şekillenmiştir. İlk uygulamalar genellikle manuel kod incelemesi şeklindeydi, ancak teknolojik ilerlemelerle birlikte otomatik analiz araçları geliştirilmiştir.
- **Gelişim Süreci ve Teknolojik İlerlemeler:** SAST'in gelişimi, yazılım geliştirme süreçlerinin karmaşıklığının artmasıyla paralel olarak hızlandı. Yapay zeka ve makine öğrenimi gibi teknolojik ilerlemeler, SAST araçlarının etkinliğini artırdı ve daha akıllı analizlerin yapılmasını sağladı.
- **Bugünkü Durumu ve Gelecek Beklentileri:** Bugün, SAST endüstrisi hızla büyümekte ve yeni teknolojik gelişmelerle desteklenmektedir. Gelecekte, daha da gelişmiş analiz teknikleri ve entegrasyon seçenekleri beklenmektedir. Bu da, yazılım güvenliği alanında daha etkili ve kapsamlı çözümlerin geliştirilmesine olanak tanır.

2. SAST (Static Application Security Testing) Araçları ve Teknolojileri

SAST (Static Application Security Testing) araçları, yazılım geliştirme sürecinde kaynak kodu statik olarak analiz ederek güvenlik açıklarını tespit eden teknolojilerdir. Bu araçlar, kod derlenmeden veya çalıştırılmadan önce potansiyel güvenlik zafiyetlerini belirlemeye yardımcı olur. İşte popüler SAST araçları ve bu araçların özellikleri, çalışma prensipleri ve entegrasyon yöntemleri hakkında detaylı bir inceleme:

1. SonarQube

Özellikleri:

- **Kod Kalitesi ve Güvenlik:** SonarQube, hem kod kalitesi hem de güvenlik açıklarını tespit eden kapsamlı bir platformdur.
- **Çoklu Dil Desteği:** Java, C#, JavaScript, Python, PHP ve daha birçok dili destekler.
- **Sürekli Entegrasyon:** Jenkins, GitLab CI/CD gibi sürekli entegrasyon sistemleriyle kolayca entegre olur.
- **Kod Kokuları:** Sadece güvenlik açıklarını değil, aynı zamanda kod kokularını (kodun kötü kokması anlamına gelen kod kalitesi sorunları) da tespit eder.
- **Raporlama:** Detaylı raporlar ve metrikler sunarak, geliştiricilerin sorunları hızlıca çözmelerine yardımcı olur.

Nasıl Çalışır: SonarQube, kaynak kodu statik olarak analiz eder ve kodu önceden belirlenmiş kurallara göre tarar. Güvenlik açıklarını, kod kokularını ve kod kalitesi sorunlarını belirleyerek geliştiricilere raporlar sunar.

Entegrasyon Yöntemleri:

- **Sürekli Entegrasyon Araçları:** Jenkins, Bamboo, GitLab CI/CD ile entegre edilebilir.
- **IDE Entegrasyonu:** Eclipse, IntelliJ IDEA gibi IDE'lere eklentiler aracılığıyla entegre edilebilir.
- **API Desteği:** REST API'leri aracılığıyla diğer sistemlerle entegre edilebilir.

2. Checkmarx

Özellikleri:

- **Geniş Güvenlik Kapsamı:** OWASP Top 10, SANS Top 25 ve diğer güvenlik standartlarına göre analiz yapar.
- **Çoklu Dil Desteği:** Java, C#, JavaScript, Python, PHP, Ruby, vb. gibi birçok dili destekler.
- **Güvenlik Analizi:** Derinlemesine kod analizi yaparak, kod tabanında mevcut olan güvenlik açıklarını tespit eder.
- **Yönetim ve İzleme:** Kullanıcı dostu bir arayüzle güvenlik açıklarını yönetir ve izler.
- **Eğitim ve Farkındalık:** Geliştiricilere güvenlik açıklarını nasıl düzeltecekleri konusunda rehberlik eder.

Nasıl Çalışır: Checkmarx, kaynak kodu tarar ve güvenlik açıklarını bulmak için çeşitli statik analiz tekniklerini kullanır. Güvenlik açıklarını ve kod zafiyetlerini belirleyerek, geliştiricilere bu açıkların nasıl düzeltileceği konusunda öneriler sunar.

Entegrasyon Yöntemleri:

- **CI/CD Entegrasyonu:** Jenkins, GitLab CI/CD, Azure DevOps gibi CI/CD araçlarıyla entegre edilebilir.
- **IDE Entegrasyonu:** Visual Studio, Eclipse, IntelliJ IDEA gibi popüler IDE'lere eklentiler sunar.
- **API Desteği:** RESTful API'leri aracılığıyla diğer uygulamalar ve sistemlerle entegrasyon sağlar.

3. Fortify (Micro Focus Fortify)

Özellikleri:

- **Kapsamlı Güvenlik Testi:** Uygulama güvenliği için geniş kapsamlı bir çözüm sunar.
- **Çoklu Dil Desteği:** Java, C/C++, .NET, PHP, Python ve diğer birçok dili destekler.
- **Derinlemesine Analiz:** Kodun derinlemesine analizini yaparak, olası güvenlik açıklarını belirler.
- **Gelişmiş Raporlama:** Detaylı raporlar ve analizler sunarak, güvenlik açıklarını anlamayı ve düzeltmeyi kolaylaştırır.
- **Bulut Desteği:** Hem on-premise hem de bulut çözümleri sunar.

Nasıl Çalışır: Fortify, statik kod analiz araçları ile kaynak kodu tarar ve güvenlik açıklarını tespit eder. Bu analiz sırasında, kodun tüm katmanlarını inceleyerek potansiyel zafiyetleri ortaya çıkarır.

Entegrasyon Yöntemleri:

- **CI/CD Entegrasyonu:** Jenkins, Bamboo, GitLab gibi CI/CD sistemleriyle entegrasyon sağlar.
- **IDE Entegrasyonu:** Eclipse, IntelliJ IDEA, Visual Studio gibi IDE'lerle entegre edilebilir.

- **API Desteği:** REST API'leri aracılığıyla diğer sistemlerle entegrasyon imkanları sunar.

Sonuç

SAST araçları, yazılım geliştirme sürecinde güvenlik açıklarını erken aşamalarda tespit etmek için kritik öneme sahiptir. SonarQube, Checkmarx ve Fortify gibi popüler SAST araçları, farklı özellikleri ve entegrasyon yöntemleriyle geliştiricilere kapsamlı güvenlik çözümleri sunar. Bu araçların doğru kullanımı, yazılım projelerinde güvenliğin sağlanmasına ve kod kalitesinin artırılmasına büyük katkı sağlar.

3. SAST'in Yazılım Geliştirme Yaşam Döngüsünde (SDLC) Yeri

SAST (Statik Uygulama Güvenlik Testi), yazılım geliştirme yaşam döngüsü (SDLC) içinde önemli bir role sahiptir.

SAST, kaynak kodun taranarak uygulamalardaki güvenlik açıklarının tespit edilmesi için kullanılan bir güvenlik aracıdır ve beyaz kutu testi olarak da bilinir.

Planlama ve Gereksinim Analizi:
1. **Tasarım**
2. **Geliştirme**
3. **Test**
4. **Dağıtım**
5. **Bakım ve Güncellemeler**

1. Planlama ve Gereksinim Analizi:

Planlama aşaması, projenin temellerinin atıldığı ve güvenlik gereksinimlerinin tanımlandığı kritik bir dönemdir. SAST, bu erken aşamada, güvenlik gereksinimlerinin doğru bir şekilde tanımlanmasına yardımcı olur.

Örneğin, geliştiriciler ve güvenlik uzmanları, SAST araçlarını kullanarak, uygulamanın tasarımında dikkate alınması gereken potansiyel güvenlik zafiyetlerini ve riskleri belirleyebilir.

Tasarım:

Tasarım sürecinde, SAST araçları, uygulamanın mimarisinin güvenlik açısından sağlam olup olmadığını değerlendirmek için kullanılır. Bu aşamada, güvenlik açıklarını önlemek için alınacak önlemler ve güvenlik kontrolleri belirlenir.
SAST, tasarım kararlarının güvenlik üzerindeki etkilerini analiz ederek, daha güvenli yazılım tasarımları oluşturulmasına katkıda bulunur.

Geliştirme:

Geliştirme aşamasında, SAST, kodun yazıldığı andan itibaren devreye girer. Geliştiriciler, yazdıkları kod üzerinde SAST taramaları yaparak, güvenlik açıklarını erken bir noktada tespit edebilir ve düzeltebilir.

Bu, “Shift Left” yaklaşımının bir parçasıdır ve güvenlik kontrollerinin geliştirme sürecinin daha erken aşamalarına taşınmasını ifade eder. Bu sayede, güvenlik sorunları daha büyük ve maliyetli hale gelmeden önce ele alınabilir.

Shift Left yaklaşımı nedir?
"Shift Left" yaklaşımı, yazılım geliştirme sürecinde kalite ve güvenlik testlerini erken aşamalara taşıma stratejisidir. Bu yaklaşım, yazılım yaşam döngüsünün erken safhalarında potansiyel hataların, güvenlik açıklarının ve diğer sorunların tespit edilip düzeltilmesini sağlar. Böylece, hataların daha büyük ve maliyetli hale gelmeden çözülmesi mümkün olur, genel yazılım kalitesi ve güvenliği artırılır

Test:
Test aşamasında, SAST, uygulamanın test ortamına yerleştirilmesinden önce kaynak kod üzerinde kapsamlı bir güvenlik taraması yapar. Bu taramalar, daha önce gözden kaçmış olabilecek güvenlik açıklarını ortaya çıkarır.

Test sürecinde bulunan güvenlik açıklarının düzeltilmesi, uygulamanın canlı ortama güvenli bir şekilde sürülmesini sağlar.

Dağıtım:
Dağıtım aşamasında, SAST, son kez güvenlik taraması yaparak, uygulamanın canlı ortama sürülmeden önce herhangi bir güvenlik açığı olmadığından emin olur. Bu, uygulamanın güvenli bir şekilde yayınlanmasını ve kullanıcıların güvenliğini sağlar.

Bakım ve Güncellemeler:
Bakım ve güncelleme aşamasında, SAST, düzenli aralıklarla güvenlik taramaları yaparak, uygulamanın güncel güvenlik standartlarına uygun kalmasını sağlar. Yeni tehditler ve güvenlik açıkları ortaya çıktıkça, SAST bu açıkların tespit edilmesine ve düzeltilmesine yardımcı olur.

SCA *Yazılım Kompozisyon Analizi nedir:*

Yazılım Kompozisyon Analizi, yazılım geliştirme sürecinin ayrılmaz bir parçasıdır ve açık kaynak bileşenlerinin ve potansiyel güvenlik açıklarının kapsamlı bir şekilde tanımlanmasını, izlenmesini ve yönetilmesini sağlayarak güvenliği artırır.

Shift-Everywhere yaklaşımı:
Benzer şekilde, SAST, kod yazma ve derlemeye odaklanarak, geliştirmenin ilk aşamalarında yoğun bir şekilde kullanılmaktadır. Bu aşamalarda SAST, olası güvenlik açıklarını belirleyerek geliştiricilere müdahale için anında içgörüler sunar.

"Her yere kaydırma" bakış açısını benimseyen SAST, geliştirmenin ötesine geçerek test, entegrasyon ve operasyonel aşamalara aktif olarak katılır. Amacı, uygulamanın tüm ömrü boyunca güvenlik açıklarının sürekli izlenmesini ve çözülmesini sağlamaktır.

SAST ve SCA yı SDLC de birlikte kullanmanın önemi

Kapsamlı Güvenlik Analizi:

- SCA, üçüncü taraf bileşenlerdeki güvenlik açıklarını tanımlar.
- SAST, kod analizi yoluyla kaynak koddaki güvenlik açıklarını ortaya çıkarır.
- Birlikte kullanıldıklarında, hem dış bağımlılıkların hem de orijinal kodun güvenlik yönlerinin kapsamlı bir analizini sağlarlar.

Gelişmiş Tehdit Algılama:

SCA, bilinen üçüncü taraf bileşenlerdeki güvenlik tehditlerini tanımlar.

SAST, kaynak kodu düzeyinde bilinmeyen güvenlik açıklarını tespit eder.

Birlikte kullanıldıklarında, hem bilinen hem de bilinmeyen güvenlik tehditlerine karşı daha geniş bir kapsam sağlarlar.

İyileştirme ve Uygulama Hızı:

- SCA, güvenlik açıkları tespit edilen kütüphanelerin hızlı bir şekilde güncellenmesini kolaylaştırır.
- Kaynak kodu düzeyinde hataları tespit eden SAST, geliştiricilere erken uyarılar sağlar.
- Bu iki yöntem birlikte kullanıldığında, hataların erken tespit edilmesini ve düzeltilmesini sağlayarak daha hızlı ve daha etkili bir güvenlik yanıtı sağlar.

Elbette! Statik Uygulama Güvenlik Testi (SAST), uygulama güvenliğini sağlamak için kullanılan bir test yaklaşımıdır. İşte SAST'ın avantajları ve dezavantajları:

Avantajları:

1. Kapsamlı Analiz: SAST, uygulamanın kaynak kodunu istatistiksel olarak inceleyerek tüm güvenlik açığı kaynaklarını tespit eder. Bu, kusurları tespit etmek için uygulamanın dahili bileşenlerini kapsamlı bir şekilde analiz ettiği için bazen “beyaz kutu” güvenlik testi olarak bilinir.

2. Belirli Konum Tespiti: SAST aracı, sorunları dosya adı ve satır numarası dahil olmak üzere belirli konumlarını belirleme kapasitesine sahiptir. Bu, hataları daha hızlı ve hassas bir şekilde çözmenizi sağlar.

3. Erken Aşamada Uygulanabilir: SAST, uygulama geliştirmenin ilk aşamalarında, derlemenin tamamlanmasından önce kod düzeyinde yapılabilir.

Dezavantajları:

1. Yanlış Pozitif Sonuçlar: SAST araçları, aslında sorunlu olmayabilecek konulara dikkat çekebilir. Bu, geliştiricilerin zamanlarını yanlış pozitif sonuçları ele alarak harcamalarına neden olabilir12.

2. Sadece Kaynak Kodunu İnceleme: SAST, sadece kaynak kodunu analiz eder. Bu nedenle, çalışma zamanı davranışları veya dışa açık bileşenlerle ilgili güvenlik açıklarını tespit etmez.

Uygulama güvenliği stratejinizi belirlerken, SAST’ın avantajlarını ve dezavantajlarını göz önünde bulundurmalısınız. İyi bir uygulama güvenliği için SAST’ı diğer test yöntemleriyle birlikte kullanmanızı öneririm.

Daha fazla bilgi edinin

4. SAST'in Avantajları ve Dezavantajları

SAST Avantajları

1. Daha Hızlı ve Daha Hassas

- Açıklama: SAST araçları, uygulamanızın ve kaynak kodunuzun hızlı ve hassas bir şekilde taranmasını sağlar. Manuel kod incelemelerine göre çok daha hızlıdırlar ve milyonlarca satır kodu kısa sürede analiz edebilirler. Bu sayede altta yatan sorunları hızlıca tespit edebilir ve çözebilirsiniz.

- Örnek: Bir banka uygulaması geliştirdiğinizi düşünün. Manuel kod incelemesi ile tüm güvenlik açıklarını tespit etmek haftalar sürebilir. Ancak, bir SAST aracı kullanarak kodunuzu dakikalar içinde tarayabilir ve potansiyel güvenlik açıklarını hızlıca belirleyebilirsiniz.

2. Erken Gelişimsel Güvenlik Sağlar

- Açıklama: SAST, uygulama geliştirme sürecinin erken aşamalarında güvenliği sağlamak için idealdir. Kodlama veya tasarım süreci sırasında kaynak kodunuzdaki zayıflıkları belirlemenize yardımcı olur. Erken tespit edilen sorunlar, daha kolay ve daha az maliyetle çözülebilir.

- Örnek: Yeni bir e-ticaret sitesi geliştirdiğinizi varsayalım. SAST kullanarak geliştirme sürecinin başında SQL enjeksiyon açıklarını tespit edebilirsiniz. Bu şekilde, kodlama sürecinde bu açıkları düzelterek daha güvenli bir uygulama oluşturabilirsiniz.

3. Dahil Edilmesi Basit

- Açıklama: SAST araçları, mevcut uygulama geliştirme yaşam döngüsüne kolayca entegre edilebilir. Diğer güvenlik test araçları ve kaynak kod depoları ile uyumlu çalışırlar. Kullanıcı dostu arayüzleri sayesinde, yüksek bir öğrenme eğrisi olmadan kullanımı mümkündür.

- Örnek: Bir yazılım geliştirme ekibiniz olduğunu düşünün. Ekip, Git gibi bir kaynak kod yönetim sistemi kullanıyor. SAST aracını Git ile entegre ederek, her kod değişikliği yapıldığında otomatik olarak güvenlik taraması yapabilir ve potansiyel sorunları anında tespit edebilirsiniz.

4. Güvenli Kodlama

- Açıklama: SAST araçları, masaüstü bilgisayarlar, mobil cihazlar, gömülü sistemler veya web siteleri için güvenli kod yazmanıza yardımcı olur. Güvenli kodlama uygulamaları oluşturarak, uygulamanızın saldırılara karşı daha dayanıklı olmasını sağlar.

- Örnek: Bir mobil uygulama geliştirdiğinizi düşünün. SAST kullanarak, kullanıcı verilerini şifreleyen kodun güvenliğini kontrol edebilir ve güvenli kodlama uygulamalarını takip edebilirsiniz. Bu, kullanıcı verilerinin çalınma riskini azaltır ve müşteri güvenini artırır.

5. Yüksek Riskli Güvenlik Açıklarının Tespiti

- Açıklama: SAST araçları, uygulamanızı ciddi şekilde etkileyebilecek yüksek riskli güvenlik açıklarını tespit eder. Arabellek taşmaları, SQL enjeksiyon kusurları ve siteler arası komut dosyası çalıştırma (XSS) gibi güvenlik açıklarını belirler.

- Örnek: Bir sağlık uygulaması geliştirdiğinizi düşünün. SAST aracı, uygulamanızda bir arabellek taşması açığı tespit edebilir. Bu açık, kötü niyetli bir saldırganın sisteme zararlı kod enjekte etmesine neden olabilir. Bu tür yüksek riskli açıkları erken tespit ederek, uygulamanızın güvenliğini artırabilirsiniz.

Sonuç

SAST, uygulama geliştirme sürecinde güvenliği sağlamak için güçlü bir araçtır. Hızlı ve hassas analizler yaparak, geliştirme sürecinin erken aşamalarında güvenlik açıklarını tespit eder ve

çözmenizi sağlar. SAST araçları, kolay entegrasyon ve kullanıcı dostu arayüzleri sayesinde geliştiricilere büyük kolaylık sağlar. Güvenli kodlama uygulamalarını teşvik eder ve yüksek riskli güvenlik açıklarını tespit ederek, uygulamanızın güvenliğini artırır.

Dezavantajlar

1. Çoğu Parametre Değeri veya Çağrı Onun Tarafından Kontrol Edilemez

• Açıklama: SAST araçları, statik analiz yaparken dinamik girdileri ve parametre değerlerini göz önünde bulunduramaz. Bu, özellikle dinamik veri akışının önemli olduğu durumlarda sınırlamalar yaratır. SAST, sadece kodun statik yapısını inceleyebildiği için çalışma zamanı verilerini analiz edemez.

• Örnek: Bir web uygulamasında kullanıcıdan alınan giriş verilerinin dinamik olarak işlendiği bir senaryoyu düşünün. SAST aracı, bu giriş verilerinin nasıl işlendiğini veya hangi parametre değerlerinin kullanıldığını tam olarak tespit edemez. Bu da potansiyel güvenlik açıklarının gözden kaçmasına neden olabilir.

2. Kodu Test Etmek ve Yanlış Pozitifleri Önlemek için Verileri Birleştirmesi Gerekir

• Açıklama: SAST araçları, bazen yanlış pozitif sonuçlar üretebilir. Yanlış pozitif, aslında güvenli olan kod parçalarının riskli olarak işaretlenmesi durumudur. Bu, geliştiricilerin zamanlarını yanlış pozitif sonuçları analiz etmekle harcamalarına neden olabilir. Bu sorunu azaltmak için, kodun farklı parçalarını ve verilerini birleştirerek daha bütüncül bir analiz yapmak gerekir.

• Örnek: Bir e-ticaret platformunda, kullanıcı oturumlarını yöneten bir kod bloğunu düşünün. SAST aracı, oturum yönetimi ile ilgili potansiyel bir güvenlik açığı tespit ettiğinde, bu açığın gerçekten var olup olmadığını belirlemek için geliştiricilerin kodun diğer parçalarıyla ilişkilendirme yapması gerekebilir. Bu da ekstra zaman ve çaba gerektirebilir.

3. Belirli Bir Dile Bağlı Olan Araçlar, Kullanılan Her Dil İçin Farklı Şekilde Geliştirilmeli ve Sürdürülmelidir

• Açıklama: SAST araçları genellikle belirli programlama dillerine özgüdür ve her dil için ayrı ayrı geliştirilmeleri ve sürdürülmeleri gerekir. Bu da çok dilli projelerde SAST araçlarının kullanımını zorlaştırabilir ve ekstra maliyet yaratabilir.

• Örnek: Bir yazılım projesi, hem Java hem de Python kullanıyorsa, bu projeyi tamamen taramak için iki ayrı SAST aracı kullanmanız gerekebilir. Bu, ek maliyet ve yönetim zorlukları yaratabilir.

4. Kütüphaneleri veya Çerçeveleri Anlamakta Zorlanır

• Açıklama: SAST araçları, bazen kullanılan üçüncü parti kütüphane ve çerçeveleri doğru bir şekilde analiz edemez. Özellikle karmaşık ve büyük projelerde, bu durum önemli güvenlik açıklarının gözden kaçmasına neden olabilir.

• Örnek: Bir web uygulaması geliştirdiğinizi düşünün ve bu uygulama Spring Framework gibi karmaşık bir çerçeve kullanıyor. SAST aracı, bu çerçevenin iç işleyişini tam olarak anlayamayabilir ve bu nedenle çerçevenin içinde saklı olan güvenlik açıklarını tespit edemeyebilir.

5. API veya REST Uç Noktaları Anlamakta Zorlanır

• Açıklama: SAST araçları, API ve REST uç noktalarının dinamik doğasını tam olarak analiz edemez. Bu, özellikle mikro hizmet mimarilerinde ve API tabanlı uygulamalarda büyük bir dezavantaj olabilir.

• Örnek: Bir mobil uygulama, çeşitli RESTful API'ler ile etkileşimde bulunuyorsa, SAST aracı bu API çağrılarının dinamik doğasını ve potansiyel güvenlik açıklarını tam olarak anlamayabilir. Bu da API güvenliğiyle ilgili risklerin gözden kaçmasına neden olabilir.

SAST, uygulama güvenliğini sağlamak için güçlü bir araç olsa da, bazı sınırlamaları vardır. Çoğu parametre değeri veya çağrıyı kontrol edememesi, kodu test etmek ve yanlış pozitifleri önlemek için verilerin birleştirilmesi gerekliliği, belirli dillere bağımlılık, kütüphane ve çerçeveleri anlamada zorluklar ve API veya REST uç noktalarını analiz etmede yetersizlik gibi dezavantajlar, SAST'ın kullanımını etkileyebilir. Bu nedenle, SAST araçlarını diğer güvenlik test yöntemleriyle birlikte kullanmak, daha kapsamlı ve güvenli bir uygulama geliştirme süreci sağlayacaktır.

5. SAST Raporlama ve Sonuçları Yönetme

SAST En İyi Uygulamaları, Stratejiler ve Gelecek Trendleri

En İyi Uygulamalar

Otomasyon

Sürekli Entegrasyon (CI) ve Sürekli Teslimat (CD): SAST araçlarını CI/CD süreçlerine entegre ederek, kodun her değişiklikte otomatik olarak taranmasını sağlamak. Jenkins gibi bir CI/CD aracı kullanarak, her kod değişikliği sonrası SAST taraması yapılabilir ve bu taramaların sonuçları geliştirici ekibe anında bildirilir.

Otomatik Bildirimler: Tespit edilen güvenlik açıkları için otomatik bildirim sistemleri kurarak, geliştiricilerin anında bilgilendirilmesini sağlamak. Örneğin, GitHub Actions kullanarak, pull

request oluşturulduğunda otomatik SAST taraması yapılıp sonuçlar e-posta veya Slack üzerinden ekibe bildirilebilir.

Entegrasyon

IDE Entegrasyonu: SAST araçlarını geliştiricilerin kullandığı Entegre Geliştirme Ortamlarına (IDE) entegre ederek, hataların kod yazımı sırasında tespit edilmesini sağlamak. Visual Studio Code kullanarak, SonarLint gibi bir eklenti ile yazım sırasında güvenlik açıklarının belirlenmesi sağlanabilir.

Diğer Güvenlik Araçlarıyla Entegrasyon: Dinamik Uygulama Güvenlik Testi (DAST) ve Güvenlik Bilgi ve Olay Yönetimi (SIEM) sistemleri ile entegre çalışarak, kapsamlı bir güvenlik stratejisi oluşturmak. SAST araçları tarafından tespit edilen güvenlik açıkları, Splunk gibi bir SIEM platformuna gönderilerek merkezi olarak yönetilebilir.

Düzenli Taramalar

Sürekli Tarama ve Denetim: Yazılım geliştirme sürecinin her aşamasında düzenli olarak SAST taramaları yapmak, güvenlik açıklarının erken tespit edilmesini sağlar.

Sık Güncellemeler ve Bakım: SAST araçlarının ve kurallarının düzenli olarak güncellenmesi, yeni güvenlik tehditlerine karşı koruma sağlar. Bu, yazılımın sürekli olarak güvenli kalmasını sağlar.

3. Gelecek Trendleri

Yapay Zeka ve Makine Öğrenimi

Gelişmiş Analiz: Yapay zeka ve makine öğrenimi kullanarak, daha hassas ve hızlı güvenlik analizi sağlamak. Bu teknolojiler, yanlış pozitif oranlarını azaltabilir ve güvenlik açıklarının daha etkili bir şekilde tespit edilmesini sağlayabilir. Örneğin, Codacy gibi araçlar, yapay zeka destekli analizler ile kodun daha derinlemesine incelenmesini sağlar.

b. Bulut Tabanlı Çözümler

Esneklik ve Ölçeklenebilirlik: Bulut tabanlı SAST çözümleri, esnek ve ölçeklenebilir bir yapı sunarak, farklı büyüklükteki projelerde etkili kullanım sağlar. SonarCloud, bulut tabanlı bir SAST aracı olarak geniş kapsamlı ve esnek tarama hizmetleri sunar.

Entegrasyon Kolaylığı: Bulut tabanlı çözümler, farklı platformlar ve araçlarla daha kolay entegrasyon sağlayarak, iş süreçlerini optimize eder. AWS CodePipeline ile entegre çalışan bir bulut SAST aracı, sürekli entegrasyon süreçlerinde büyük kolaylık sağlar.

DevSecOps

Güvenli Yazılım Geliştirme: DevSecOps yaklaşımı ile güvenlik, yazılım geliştirme sürecinin başından itibaren entegre edilerek, güvenliğin tüm süreç boyunca sağlanması hedeflenir. Örneğin, DevSecOps kültürüne sahip bir ekip, her yazılım geliştirme adımında güvenlik kontrollerini devreye alarak çalışır.

Kültürel Değişim: Güvenliğin birincil öncelik haline gelmesi ve tüm ekiplerin bu anlayışla çalışması, daha güvenli yazılımlar geliştirilmesini sağlar. Örneğin, şirket içi hackathon etkinlikleri düzenleyerek, güvenlik bilincini artırmak ve güvenlik konusunu eğlenceli hale getirmek mümkündür.

Regülasyon ve Standartlar

Uyum ve Sertifikasyon: SAST kullanımının yasal ve endüstri standartlarına uyumu artıracak yönde gelişmesi. GDPR, HIPAA gibi düzenlemelere uyumlu çözümler geliştirilmesi. Örneğin, sağlık sektöründe faaliyet gösteren bir yazılım şirketi, HIPAA uyumluluğu için düzenli SAST taramaları yapar ve sonuçları belgeler.

SAST Raporlama ve Sonuçları Yönetme

SAST Raporlarının Analizi:

SAST araçları, yazılım kodunu inceleyerek potansiyel güvenlik açıklarını ve kod zayıflıklarını raporlar. Bu analiz süreci aşağıdaki adımları içerir:

1-Kod Analizi: SAST araçları, yazılım kodunu tarar ve güvenlik açıklarını belirler.

2-Kategorize Etme: Açıklar türlerine göre sınıflandırılır.

3-Raporlama: Detaylı açıklar ve çözüm önerileri raporlanır.

Güvenlik Açıklarının Önceliklendirilmesi ve Giderilmesi

SAST raporlarında tespit edilen güvenlik açıkları, etkilerine ve önem derecelerine göre önceliklendirilmelidir. Bu süreç şu adımları kapsar:

1-Sınıflandırma: Açıklar CVSS gibi standartlarla sınıflandırılır.

2-Risk Değerlendirmesi: Potansiyel etkiler değerlendirilir.

3-Önceliklendirme: Açıklar öncelik sırasına göre (yüksek, orta, düşük) sıralanır.

Raporların Yönetimi ve Takibi:

SAST raporlarının etkin yönetimi ve takibi, güvenlik açıklarının zamanında ve doğru bir şekilde giderilmesini sağlar. Bu süreç şu adımları içerir:

1-Depolama: Raporlar merkezi bir depoda saklanır.

2-Takip Sistemi: Güvenlik açıkları takip edilir.

3-Periyodik İncelemeler: Açıklar düzenli olarak gözden geçirilir.

4-Raporlama ve İletişim: Düzenli raporlar hazırlanır ve paylaşılır.

KAYNAKÇA

OWASP (Open Web Application Security Project)
Microsoft Security Development Lifecycle (SDL)
ISO/IEC 27001:2013 - Information Security Management
NIST (National Institute of Standards and Technology)

OWASP. "OWASP SAMM (Software Assurance Maturity Model)." OWASP SAMM.

Gartner. "Market Guide for Application Security Testing." Gartner.

Veracode. "State of Software Security." Veracode.

Synopsys. "Building Security In Maturity Model (BSIMM)." BSIMM.

NIST. "NIST Special Publication 800-53: Security and Privacy Controls for Information Systems and Organizations." NIST SP 800-53.
https://www.acunetix.com/
https://medium.com/@eliftutar007/software-development-life-cycle-sast-and-sca-combined-225f94749a5f
https://scand.com/company/blog/main-roles-and-responsibilities-in-the-software-development-cycle/

https://www.guardrails.io/blog/what-is-static-application-security-testing-sast/

https://checkmarx.com/appsec-knowledge-hub/incorporate-sast-sca-dast-in-sdlc/

https://www.sonarsource.com/learn/shift-left/

https://hashdork.com/tr/sast-vs-dast/

https://snyk.io/learn/application-security/static-application-security-testing/

https://csrc.nist.gov/pubs/sp/800/53/r5/upd1/final