



Yazılım Güvenliđi

Öğr. Gör. Gözde Mihran ALTINSOY
gozdemihranaltinsoy@beykoz.edu.tr

Savaş Sanatı

«Kendini ve düşmanını tanırsan, tehlikeye girmeden sayısız savaş kazanabilirsin...» *Sun Tzu, Savaş Sanatı ~ 2500 yıl önce*

'Savaşmadan kazanmak, en büyük başarıdır.'

'Uzak Doğu dövüş yöntemleri de benzer felsefe içinde minimum güç kullanımı ile hasmına maksimum zarar vermeye, gereğinde geri çekilir gibi manevra yaparak düşmanı gafil avlamaya yöneliktir.'

'Taoizm ; yaşamın birbiriyle sürekli çelişen güçlerin karışımı olduğunu önerir.'

'Tao düşüncesini yansıtmakta olan Savaş Sanatı, sadece savaşın değil aynı zamanda barışın da kitabıdır.'

Savaş Sanatı – Taoizm, Taoist

'Savaş Sanatı'na göre, usta savaşçı çatışma psikolojisi ve mekanizmalarını öylesine iyi bilir ki, düşmanın her hareketini derhal algılayıp, her olasılığa uygun en doğal manevrayı en az güç kullanımı ile uygular.'

'Duruş belirsiz, hamleler öngörülemez olunca. hamleye hazırlık yapmak imkansızdır.'

'Savaş Sanatı'nda Sun Tzu, « Olabildiğince gizlen, öyle ki görünmez ol. Olabildiğince gizemli ol, öyle ki sesin bile işitilmesin . O zaman düşmanın kaderi senin elindedir.» der.'

SAVAŞ SANATI = SAVAŞMADAN KAZANMAK



Bir komutan kendi başına görmeli, bilmelidir.

Bunun anlamı, komutanın başkalarının göremediğini görmesi, başkalarının bilemediğini bilmesidir.

Başkalarının göremediğini görmek parlak zeka, başkalarının bilemediğini bilmek üstün zekadır.

İlk kazanan parlak, üstün zekalılardır. Çünkü sadece onlar saldırılması olanaksız yerlerde savunma yapabilir, direnilmesi imkansız yerlere saldırabilirler.

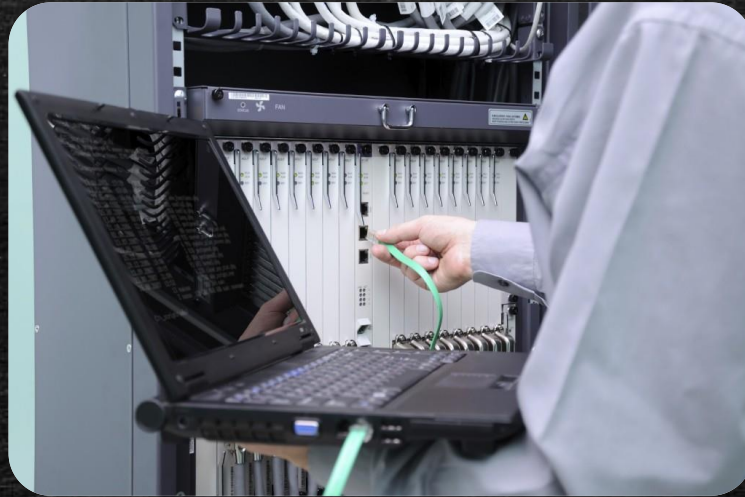
Temel Farklar

Uygulama Geliştiricileri



Hedef: Uygulama nasıl çalışacak?
Uygulamanın güvenliği tam olarak sağlanabildi mi?
Baz aldıkları: Gereksinimler ve kullanım senaryoları neler?
Sonuç: Hedefleri gerçekleştirecekleri dizaynları ve uygulamaları geliştirmek, gerekli özelliklere sahip olmasını sağlamak

Sistem Yöneticisi



Hedef: Sistemin güvenliği tam olarak sağlanabildi mi?
Amaç: Sistemlerinin çalışma devamlılığını sağlamak.

Saldırganlar



Hedef:

- Bir uygulama veya bir sistem ne için ve nasıl çalışıyor?
- Var olan kontrolleri nasıl atlatabilir?
- Sistemin muhtemel açıkları neler?

Amaç: Sistemi kötüye kullanabilmek.

Temel Sorular?

- ! ? Yazılım geliştirici ve sistem yöneticisi saldırgan kadar bilgi birikimine sahip mi?
- ! ? Saldırganların kullandıkları teknikler ve buna karşın alınabilecek önlemler tam olarak biliniyor mu?
- ! ? Uygulamalar ve sistemler saldırganlardan korunabilecek mi?

Bilgi GüvenliĐinin Özellikleri

↓

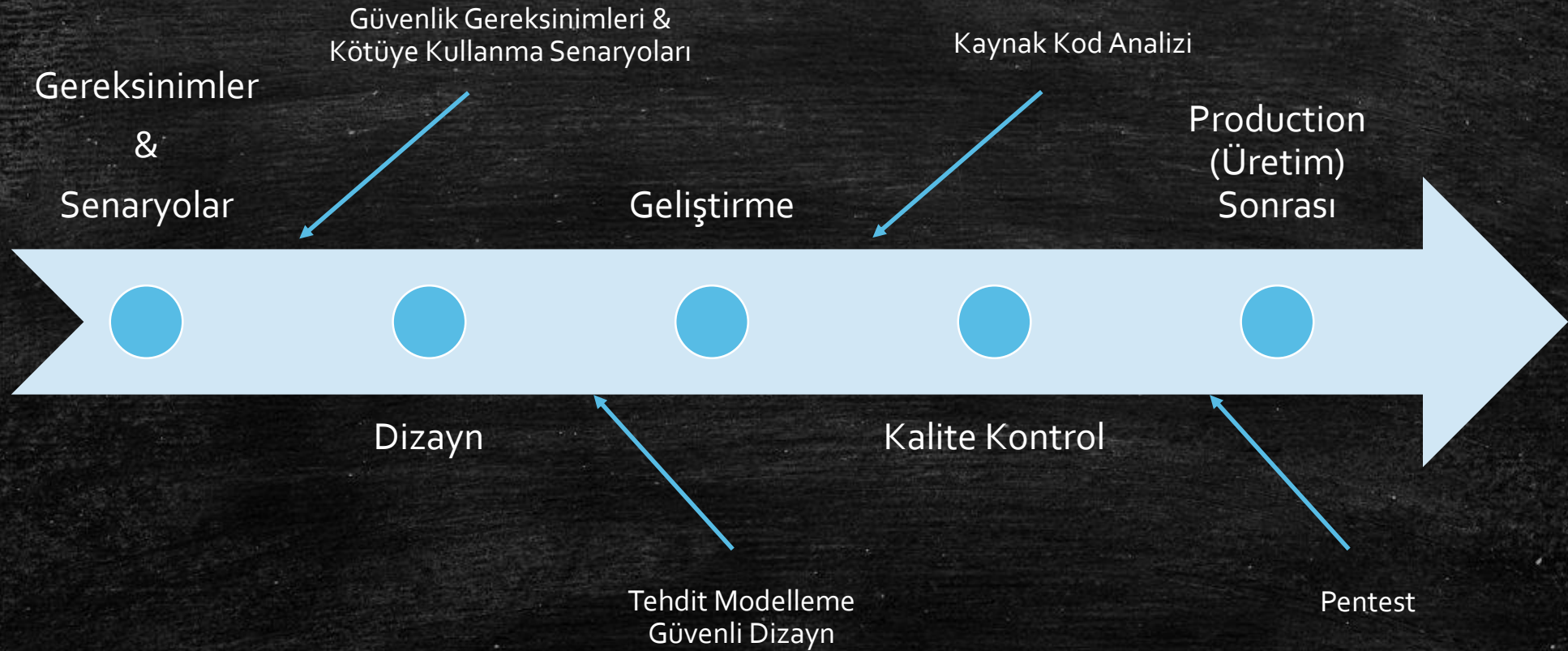
Güvenli Bir Yazılımın Prensipleri



Güvenli Yazılım

- İşlediği ve ulaşabildiği verinin bütünlüğünü, gizliliğini korumalıdır.
- Bilgiye erişimin devamlılığını sağlamalı, yani doğru çalışmaya devam etmelidir.
- Güvenlik önlemlerinin veya denetimlerin uygulama geliştirme safhasının uygulanabilecek en erken dönemlerinde gerçekleştirilmesi gerekmektedir.

Uygulama Güvenliği



Fuzzing (Sızma Testi)

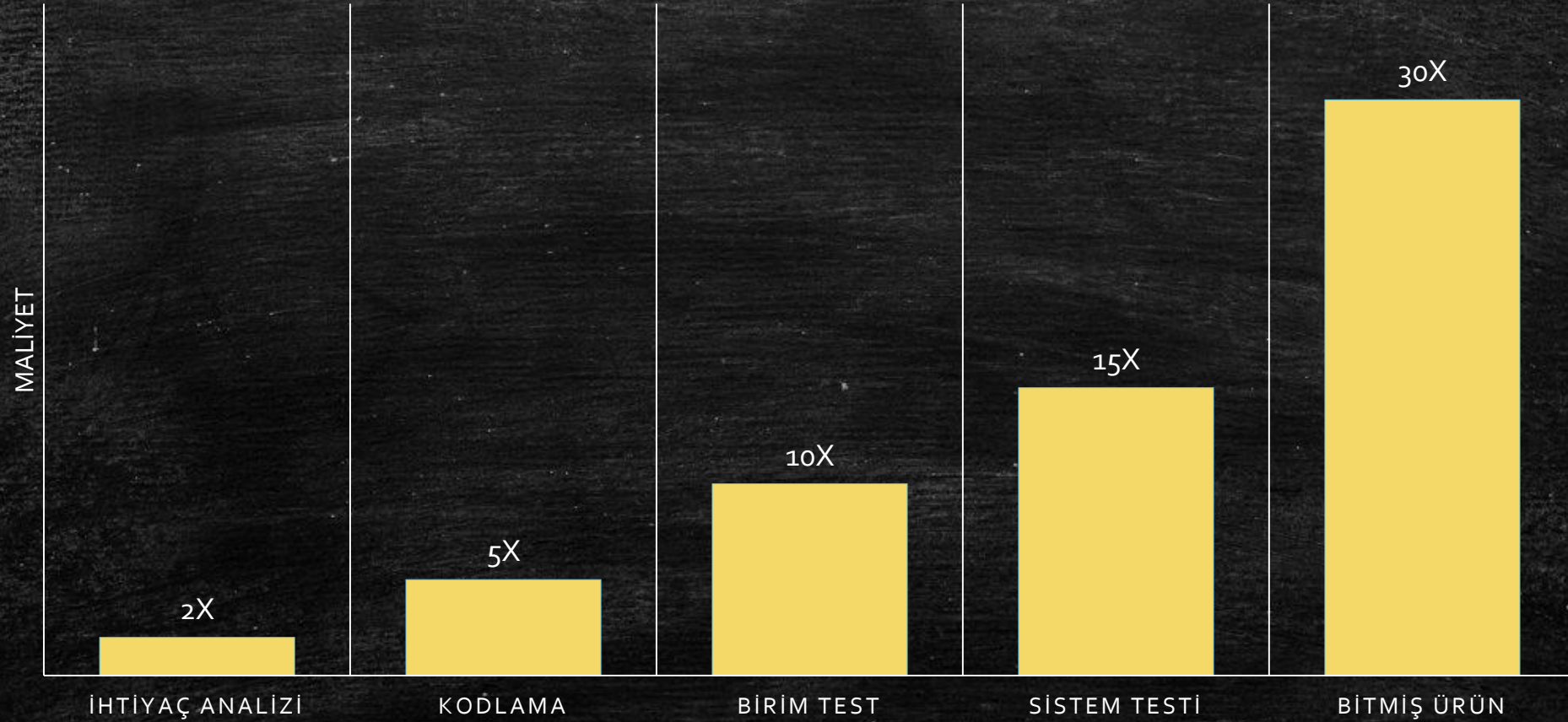
Fuzzing, sisteme beklenmedik, yarı geçerli, sıralı verilerin gönderimi gibi yöntemlerle sistemin iç yapısındaki hataları bulmayı hedefleyen Kapalı-Kutu yazılım test etme yöntemi.

Fuzzing için çeşitli test yöntemleri isimlendirilmektedir:

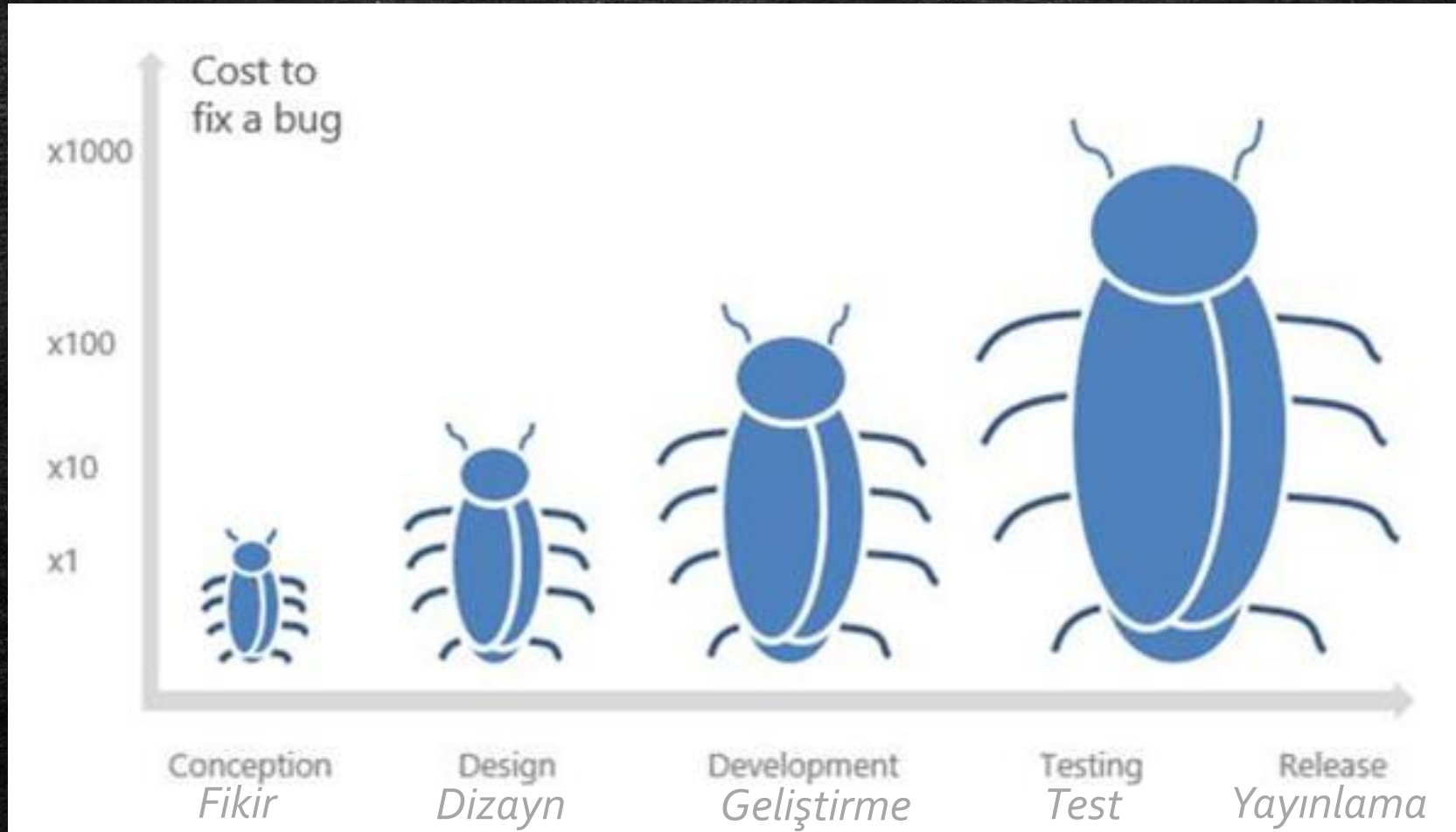
- Olumsuz test yöntemi;
- Protokol mutasyon;
- Güvenilirlik test yöntemi;
- Sözdizimi test yöntemi;
- Hata enjeksiyonu;
- Yağmurlu-gün test yöntemi;
- Kirli test yöntemi.

Fuzzing yöntemlerinin uygulanmasını sağlamak için geliştirilen programlara ise Fuzzer denilmektedir.

IBM Araştırma Sonucu Maliyet Analizi



Hataların Zamana Göre Maliyete Etkisi



Yazılım Açıklıklarının Verebileceği Zararlar



- Açıklıkların Saldırganlar Tarafından Kullanılması
- Sosyal Medyada Yer Bulması
- Açıklıkların Maliyeti
- İtibar Kaybı
- Finanslar Kayıplar
- Lisans Kayıpları vb.

Olgunluk Modelleri;

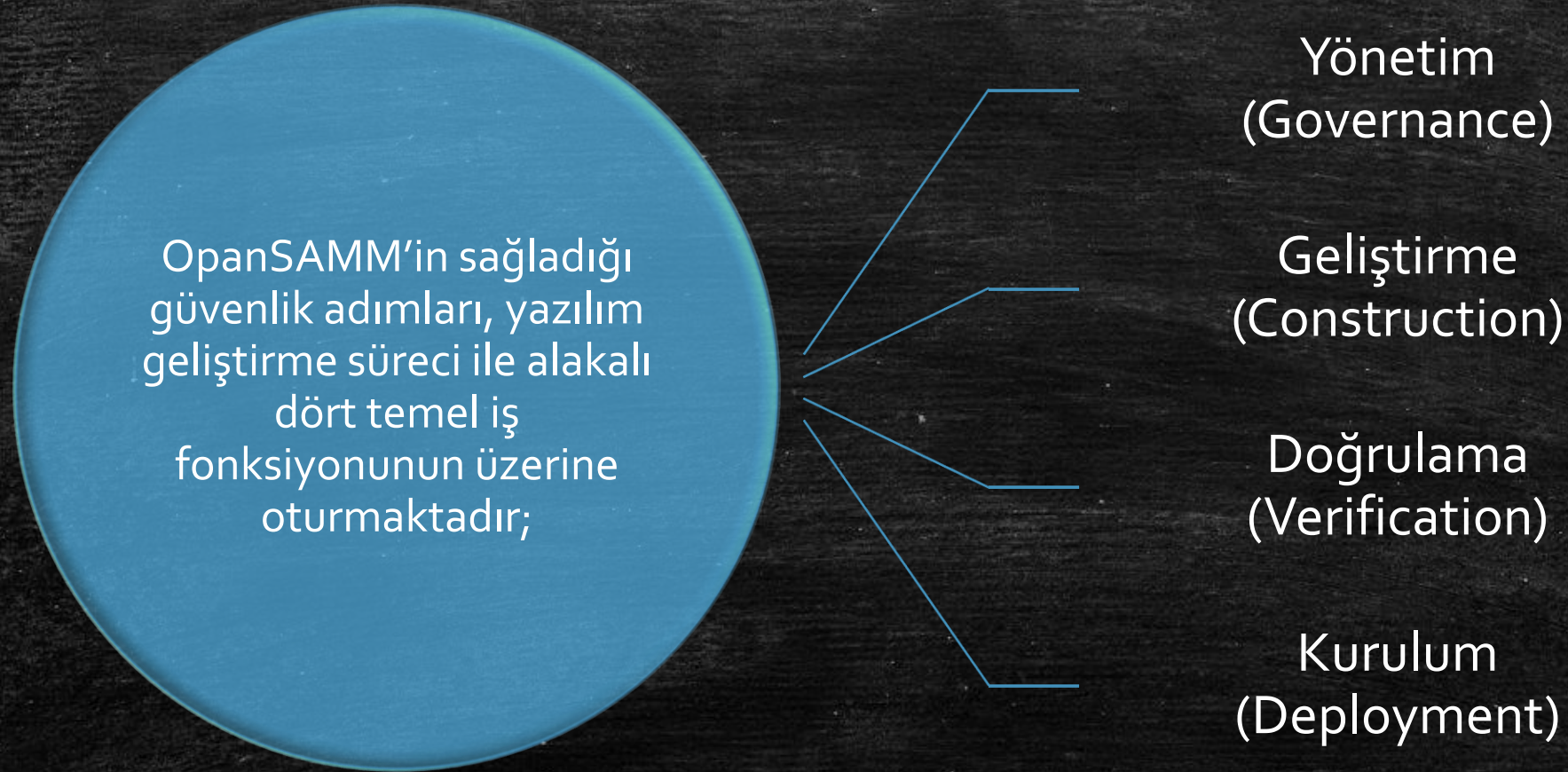
- Bir organizasyonun diğer organizasyonlara göre hangi seviyede olduğunu gösterir.
- Organizasyon içi ve organizasyonlar arası ortak bir dil oluşmasına yardımcı olur.
- Organizasyon içi muhtemel eksikliklerin bulunması ve bu eksikliklerin anlaşılmasını sağlar.
- Alınması gereken aksiyonlarda, hedef organizasyona özel önem derecelendirmesini ortaya çıkarır.
 - 90'lı yılların sonlarına doğru yazılım güvenliğine yönelik aksiyonlar alınmaya başlanmıştır.
 - 2008 yılında yazılım güvenliğine yönelik olgunluk modelleri BSIMM, OpenSAMM dokümanları ile ortaya çıkmıştır.

BSIMM, OpenSAMM Olgunluk Modelleri

- Bu modellerin uygulanması ile beraber organizasyon;
 - Uygulanan veya uygulanmayan yazılım güvenlik süreçleri ile tanımlanan modelde nereye oturduklarını öğrenebileceklerdir.
 - Uygulanan veya uygulanmayan süreçlerde muhtemel eksikliklerini bulabileceklerdir.
 - Bulunan eksikliklere göre alınacak aksiyonların listesini çıkarabilecek ve önemlerine göre bu aksiyonları seviyelendirebileceklerdir.



OpenSAMM (Software Assurance Maturity Model) Olgunluk Modeli (OWASP Projesi)



OpenSMM Güvenlik Adımları (Alınması Gereken Aktiviteler)



Yönetim	Geliştirme	Doğrulama	Kurulum
<ul style="list-style-type: none">• Strateji ve Metrikler (Strategy & Metrics)• Politika ve Uyumluluk (Policy & Compliance)• Eğitim ve Kılavuzluk (Education & Guidance)	<ul style="list-style-type: none">• Tehdit Değerlendirme (Threat Assessment)• Güvenlik Gereksinimleri (Security Requirements)• Güvenli Mimari (Secure Architecture)	<ul style="list-style-type: none">• Tasarım Denetimi (Design Review)• Kod Denetimi (Code Review)• Güvenlik Testi (Security Testing)	<ul style="list-style-type: none">• Açıklık Yönetimi (Vulnerability Management)• Sistem Sıkılaştırma (Environment Hardening)• Operasyonel Strateji (Operational Enablement)

Örnek Bir OpenSAMP fonksiyonu olan Doğrulama İncelemesi



- **Tasarım Denetimi (Design Review) :** Tehdit Modelleme
 - Güvenlik risklerinin sistematik bir şekilde bulunması ve önem sırasına konulmasıdır.
 - **TEHDİT:**
 - İstenmeyen olay. Bu istenmeyen olay sistem kaynaklarına zarar verir.
 - Saldırgan aktör (kötü niyetli bir kullanıcı veya virüs gibi).
- **Kod Denetimi (Code Review) :** Kod Gözden Geçirme (İnsanların Tarafından Analiz Edilmesi) veya Statik Kaynak Kod Analizi (Özel Programlar Tarafından Analiz Edilmesi)
 - Bir yazılımdaki güvenlik problemlerinin bulunması için gerçekleştirilen analizdir.
- **Güvenlik Testi (Security Testing) :** Sızma Testi

1. Tehdit Modelleme

Genel anlamda beş ana tehdit modelleme adımı vardır;

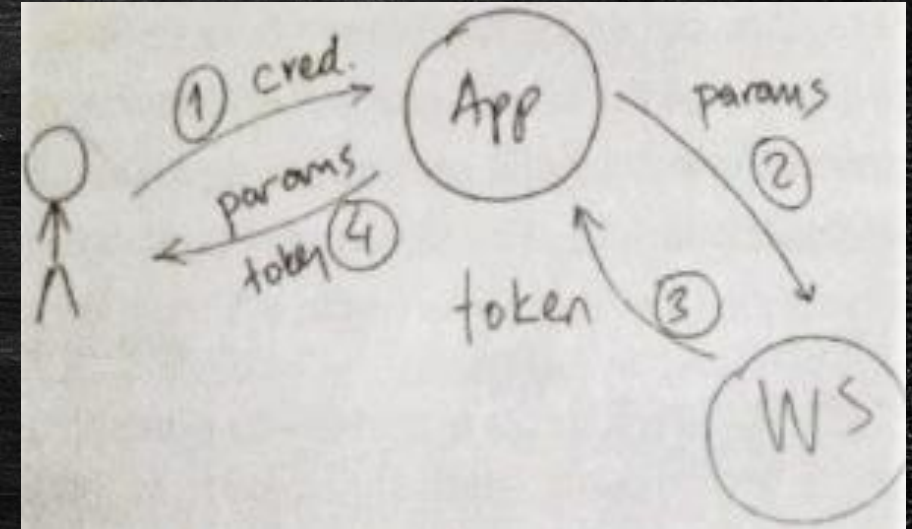
- 🔒 Güvenlik hedeflerinin ortaya çıkarılması; tehdit model aktivitesinin nedeninin açık bir şekilde belirtilmesi ve sürecin zamanının belirlenmesi sağlanır.
- 🔒 Genel bir uygulama şemasının çizilmesi; uygulamanın en önemli parçalarının ve aktörlerin belirlenmesi gerçekleştirilir.
- 🔒 Uygulamanın parçalara bölünmesi; uygulama mekaniğinin daha detaylı parçalara bölünmesi daha detaylı ve ilişkili tehditlerin belirlenmesini sağlar.
- 🔒 Tehditlerin belirlenmesi; yukarıdaki iki adım kullanılarak uygulamaya ve kapsamına has tehditler belirlenir.
- 🔒 Açıklıkların belirlenmesi; belirlenen tehditler ile alakalı zafiyetlerin bulunması gerçekleştirilir.

1. Tehdit Modelleme

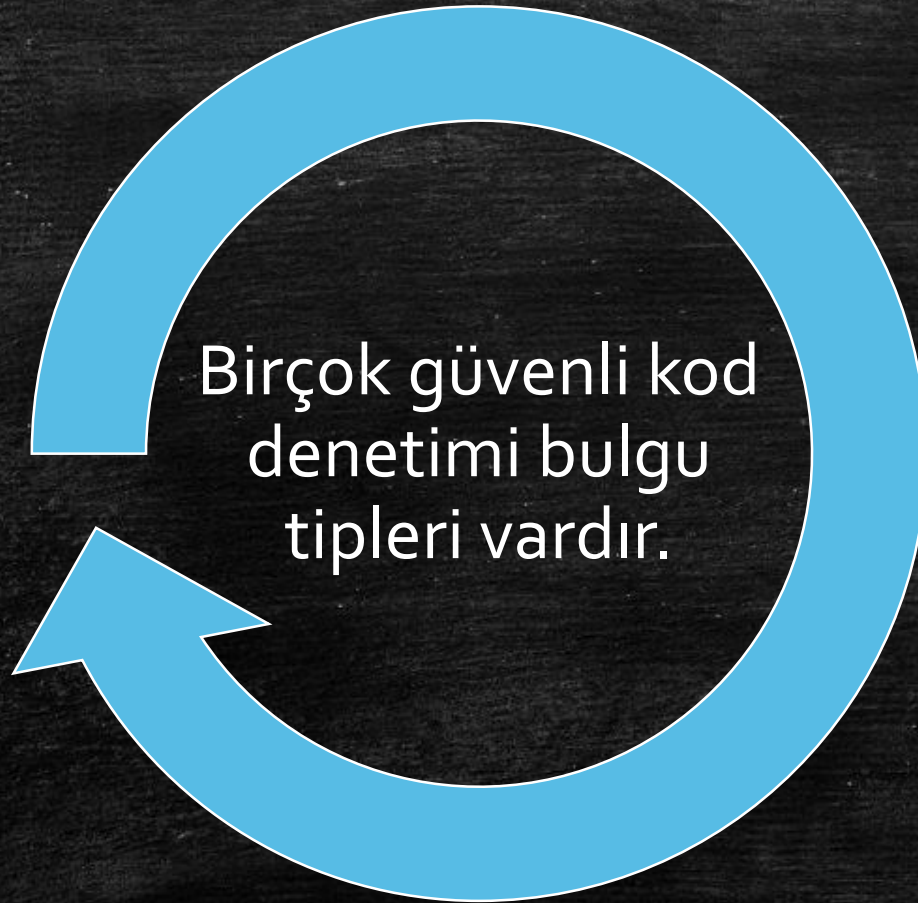
Tehdit modelleme genellikle **uygulama analistleri, operasyon liderleri ve/veya geliştiricilerle** bir toplantı şeklinde gerçekleştirilir.

Uygulamanın **ne yaptığı, hangi kaynakları nasıl kullandığı, önemli kullanım senaryoları** gibi konular konuşuldukça tek sayfalık bir **aktörler, kaynaklar, bileşenler, iletişim trafiği** bilgisi ortaya çıkar. Bu bilgiler yardımı ile tehditler ortaya çıkarılabilir ve sonrasında tek sayfa çiziminde daha da detaya inilebilir.

Yandaki resim, beyaz bir tahtaya çizilmiş örnek bir uygulama modellemesini göstermektedir.



2. Güvenli Kod Denetimi



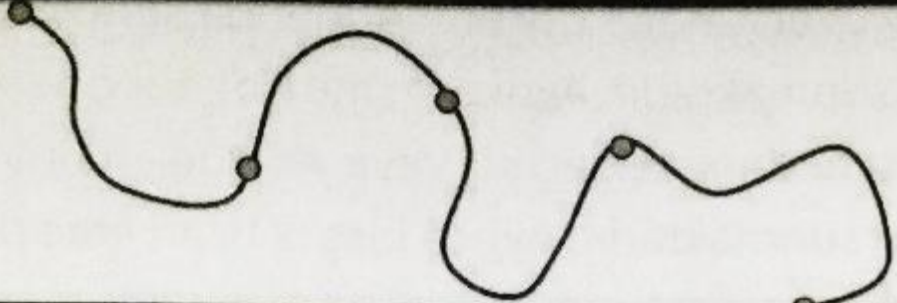
- Yapılandırma problemleri
- Yapısal problemler
- Akış kontrol problemleri
- Veri akış problemleri

2. Güvenli Kod Denetimi

Veri akış problemlerine örnek verecek olursak;

Resimde temel bir Java servlet kod bölümlerini göstermektedir. Kullanıcıdan alınan *name* HTTP parametre isimli değişken değerinin, yazılım içerisinde belirli noktalarda değişkenler veya veri yapıları (Arrays, Collections, Model nesneleri) yardımı ile aktarılarak denetim yapılmadan bir dinamik SQL sorgusuna eklendiğini göstermektedir. Bu da **SQL Enjeksiyonu** güvenlik açığı demektir.

Otomatik olarak kaynak kod üzerinde çalışan bir aracın, bu tür veri akışlarını anlayabilmesi ve güvenilmeyen girdilerin tehlikeli API'lara ulaştığını fark edebilmesi gerekmektedir.



```
String userval = request.getParameter("name");  
  
qry = "select * from users where name=" + par + "";  
conn.RunQuery(qry);
```


Statik Test vs. Statik Analiz

Statik Test,
yazılımın
çalıştırılmadan
(execute)
kontrol
edilmesidir.

- Gözden Geçirme (Review)
 - Statik Analiz
 - Yazılımı oluşturan parçaların (kod veya veri), çoğunlukla çeşitli araçlar kullanılarak test edilmesidir.
 - Statik Kod Analizi, temiz (clean) kod yazmanın en önemli bacağı olarak değerlendirilmelidir.
 - Bu araçları kullanarak clean kod yazmak için gerekli olan,
 - 👍 Kodlama standartlarına uygunluk
 - 👍 Tekrardan kaçınma
 - 👍 Birim testlerinin yeterliliği
 - 👍 Karmaşıklığın doğru dağılımı
 - 👍 Spagetti olmayan tasarım
 - 👍 Yeterli yorum frekansı
- gibi kriterleri sağlamak çok daha kolay olmaktadır.

Statik Analizde Kullanılabilir Araçlar

Bu araçların bir kısmı halihazırda kullanılan Visual Studio, IntelliJ Idea gibi popüler IDE'ler ile entegre iken, bir kısmı da bağımsız olarak yer almaktadır.

PMD, Checkstyle ve Findbugs bu araçlardan Java ekosistemi içinde en fazla kullanılanlardandır.



Statik Analizde Kullanılabilir Araçlar

Checkstyle

- Yazılan kodun, kodlama standartlarına uygunluğunu kontrol eden bir araçtır. Belirlediğiniz kod konvansiyonuna göre, isimlendirme, kütüphane kullanımı, boşluklandırma ve yorum frekansı gibi kodun okunabilirliğini ve bakım yapılabilirliği arttıracak nitelikleri kontrol eder.

FindBugs

- Bu araç diğerlerinden farklı olarak direkt bytecode üzerinden analiz işlemi yapmaktadır. PMD'ye benzer özelliklere sahip olmasının yanında concurrency (eşzamanlılık, ortak zamanlılık yani bağımsız olarak yürütülen görevlerin kompozisyonu) sorunları ve bazı zafiyetlerin bulunması yardımcı olmaktadır.

PMD

- Yazılan koddan üretilen Abstract Syntax Tree (AST) üzerinden kodu çalıştırmadan analiz işlemi yapan bir araçtır. Erişilemeyen veya tekrar eden kod parçacıklarını, aşırı karmaşıklığı, olası hataları ve optimize edilmemiş alanları belirleyen etkili bir **open-source** yazılımdır.

Dinamik Kod Analizi

Yazılımın çalıştırılarak (execute edilerek) method, kod satırı gibi seviyelerde kontrol edilmesi aktivitesidir.

Dinamik analiz ,özellikle run-time esnasında oluşabilecek hataların tespitini hedeflemektedir.

Yazılım üzerinde, bulunduğu donanımın göstereceği performans veya belirli girdiler alması sonrasında oluşacak güvenlik açıkları bu konudaki iyi örneklerdendir.

Dinamik Kod Analizinin Gerçekleştirilmesi

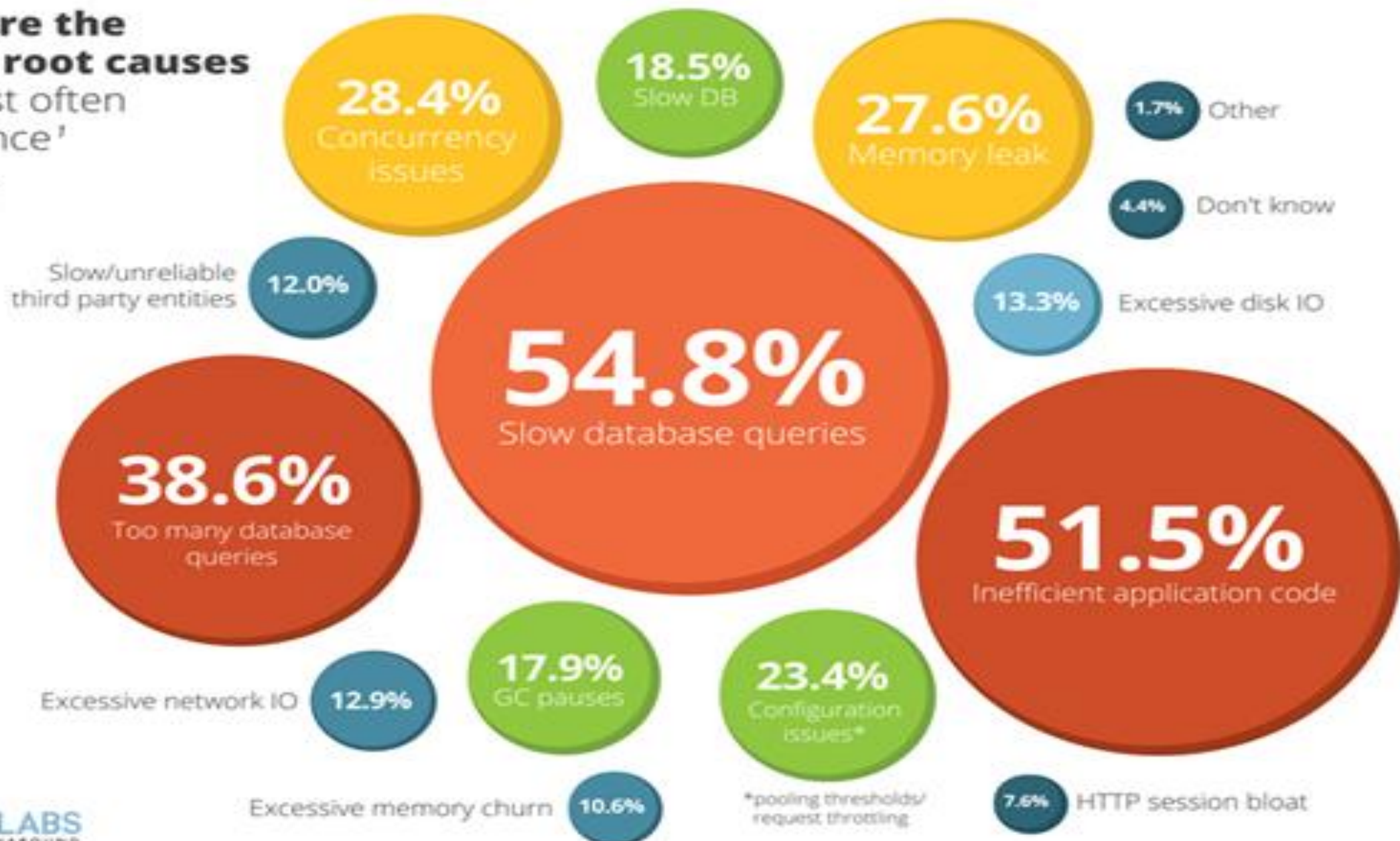


Dinamik analiz aktiviteleri, kodun geliştirilmesinin hemen sonrasında statik analiz kontrollerinin yapılması ve derlenmesi ardından gerçekleştirilmelidir. Zira, statik analizler ile dinamik analizde tespit edilebilecek sorunların daha basit bir şekilde çözülmesi mümkün olabilir.

Örneğin, yazılmış yüksek karmaşıklığa sahip bir kod bloğu çalıştırılmadan önce olası performans problemleri tespit edilerek çözümlenebilir. Özetle, çalıştırıldıktan sonra yapılacak, görece olarak daha zorlu profiling / load generation işlemlerine gerek kalmadan hatalar ortadan kaldırılabilir. Ancak bu durum, tüm hataların statik analiz ile tespit edilmesinin mümkün olabildiği anlamına gelmemektedir. Bu aşamada dinamik analiz ile beraber kullanımı oldukça faydalı olacaktır.

What are the typical root causes you most often experience¹

Figure 1.16



Dinamik Analiz Araçları = Performans

Performans amaçlı dinamik kod analizi için Java ekosisteminde kullanılabilecek, önde gelen araçlar aşağıdaki gibidir.

- Visual VM
- JProfiler
- Java Mission Control
- YourKit
- NetBeans Profiler
- JProbe
- Xrebel

Bunlar arasından VisualVM ve JProfiler en sık kullanılan iki uygulama olarak öne çıkmaktadır.



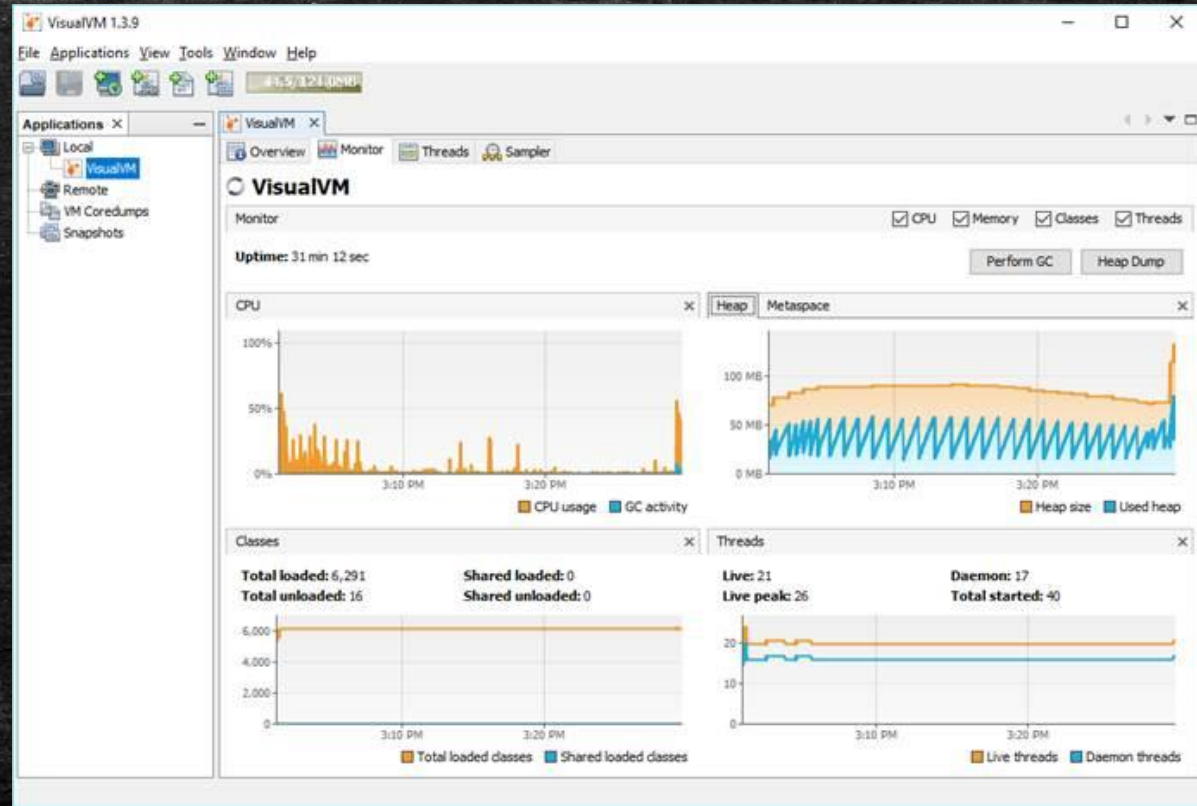
JProfiler

Dinamik Analiz Araçları - Visual VM

Bir Java uygulamasını çalıştırıp, ayağa kaldırdıktan sonra projemiz Visual VM'de gözükmektedir.

Monitör bölümünde anlık olarak değişen grafikler bulunmaktadır.

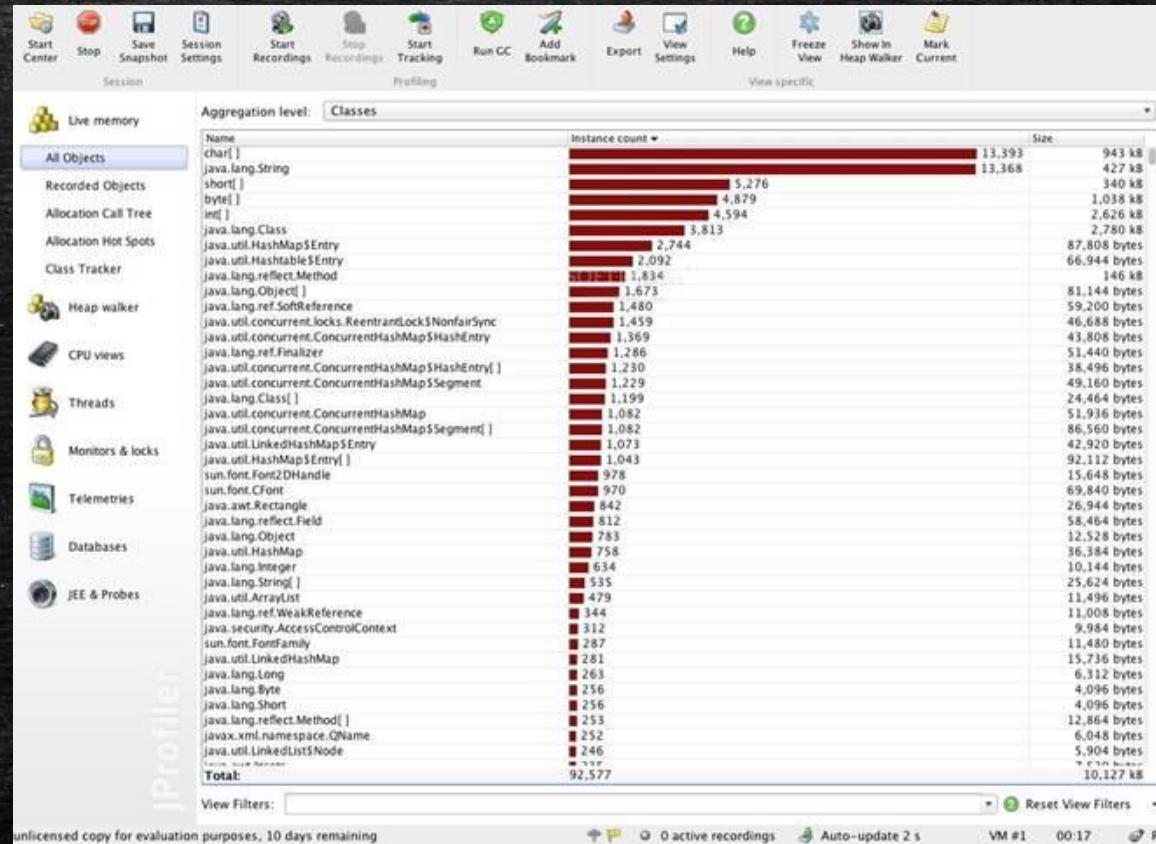
CPU, Classes, Threads, Heap ve Metaspase gibi bilgilendirmeler verilmektedir.



Dinamik Analiz Araçları - JProfiler

Bu analiz aracı genel itibariyle Visual VM aracıyla benzerlik göstermektedir.

Görselde görüldüğü üzere Memory, Heap, CPU, Threads ve Databases gibi alanlarda kodları analiz edip, sonuçları göstermektedir.



3. Sızma Testleri - PENTEST

- Sızma testleri, çalışan uygulamalar üzerinde dinamik olarak yapılan güvenlik zafiyet bulma testleridir. Hedef bir sistemin, uygulamanın güvenlik açısından genel bir resmini çekmeye yönelik uygulamalardır.
- Bu testler saldırgan bakış açısıyla gerçekleştirilirken bir çok farklı yöntem kullanılabilir.
- Bu yöntemler genel olarak ikiye ayrılır:
 - Kara kutu (blackbox)
 - Hedef sistemin alt yapısı hakkında ön bir bilgi olmadığını kabul eder. Bu nedenle testi gerçekleştiren kişiler öncelikle hedef hakkında elde edebildikleri kadar bilgi edinirler.
 - Beyaz kutu (whitebox)
 - Hedef sistem hakkında birçok ön bilgiye sahiptir; IP adresleri, kaynak kodu, alt yapı bilgileri, süreçler vb. gibi.

İş Mantığı Problemleri

İnsan gözü ile tespit edilebilecek zafiyetler bulunmaktadır. Örneğin; bir kullanıcının başka bir kullanıcıya ait fatura detayını görüntüleyebilmesi gibi.

Bu tür problemleri otomatize araçlar ile tespit etmek maalesef güçtür. Bu yüzden insan gözü ile sızma testleri bu tür problemlerin tespiti için yapılmalıdır.



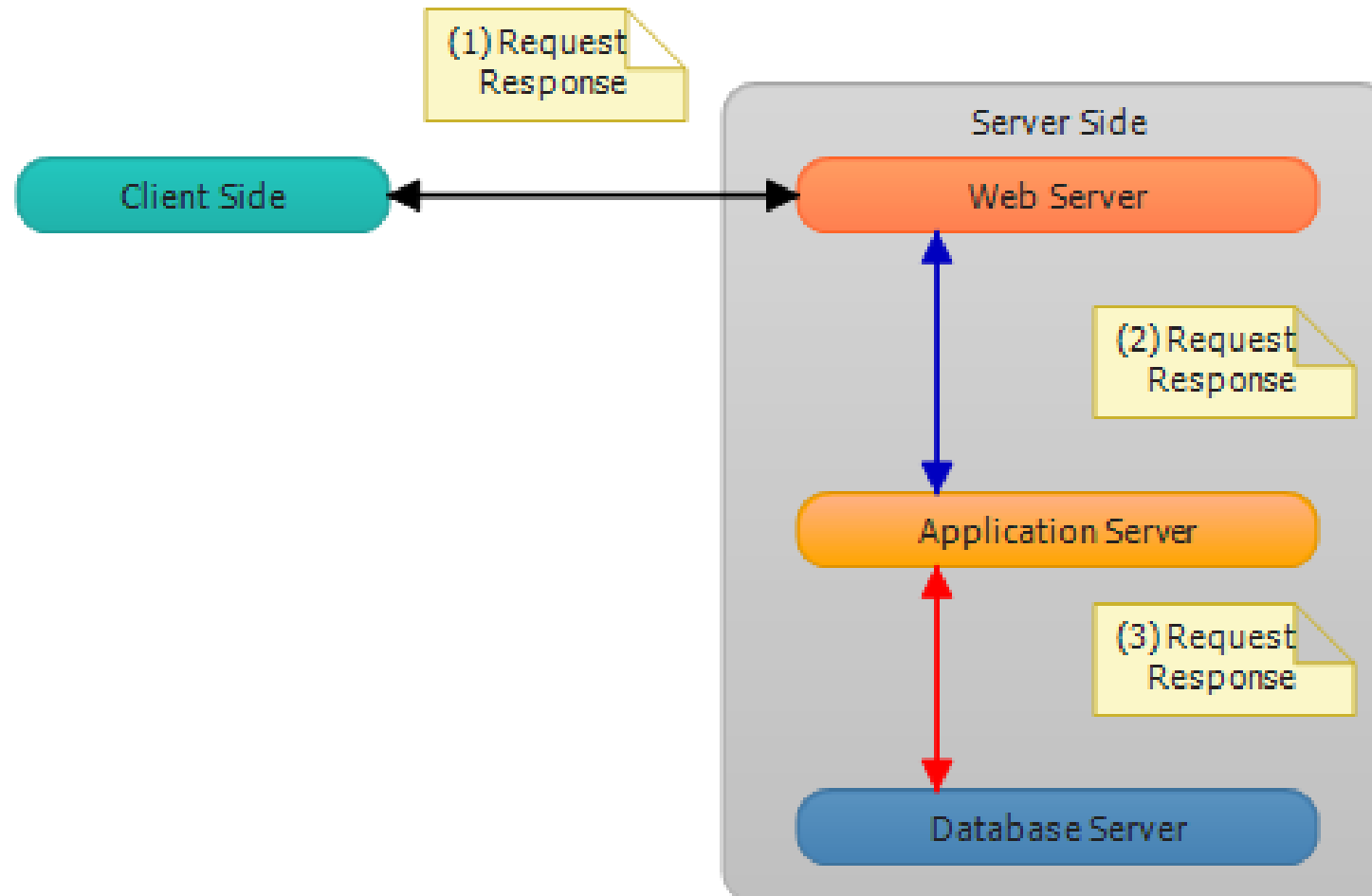
Web Standardı

Öğr. Gör. Gözde Mihran ALTINSOY
gozdemihranaltinsoy@beykoz.edu.tr

Web Uygulama Katmanları

- İstemci Tarafı
 - Son kullanıcının olduğu kısımdır.
- Sunucu Katmanı
 - Uygulamalarımızın barındığı web sunucularıdır. Güvenli uygulama geliştirme ve güvenlik test süreçlerinde dile hakim olmak önemlidir.
- Orta Katman
 - Uygulama sunucularının bulunduğu katmandır.
- Arka Uç
 - Uygulamaların kullandığı veritabanı yönetim sistemleridir.

Web Uygulama Katmanları



HTML Standardı

HTML (HyperText Markup Language – Zengin Metin İşaretleme Dili)

- Web tarayıcılarının yazı, link ve görsel verilerini sunması ve bunların ilişkilerini sağlaması için kullanılmaktadır. Programlama dili değildir, bir işaretleme dilidir.

HTML

Yanda HTML ile yazılmış örnek bir kaynak kod bulunmaktadır.

`<!DOCTYPE html>` satırı, diğer satırların tarayıcılar tarafından hangi kurallar dahilinde yorumlanacağını belirtir. Buna Document Type Declaration (DocType) denir. Bu tanımlama ile tarayıcı tarafından yorumlanan HTML kaynak kodunun geçerli olup olmadığının kontrolü amaçlanmaktadır.

HTML, iç içe etiketlerden oluşan bir dildir. HTML etiketiyle başlar ve HTML etiketiyle biter.

```
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="utf-8">
    <meta name="author" content="Gözde Mihran Altınsoy">
    <title>Gözde Mihran ALTINSOY</title>
  </head>

  <body>
    Web Siteme Hoşgeldiniz.
  </body>

</html>
```


HTML Örnek Kod ve Tarayıcıdaki Görünümü

HTML Örnek Kod

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>

    <h1>Benim Sayfam</h1>
    <p>Bu bir paragraftır.</p>
    <ul>
      <li>1.Madde</li>
      <li>2.Madde</li>
    </ul>
  </body>
</html>
```

HTML Örnek Çıktı

Benim Sayfam

Bu bir paragraftır.

- 1.Madde
- 2.Madde

HTML Kodlama (HTML Encoding)

Tarayıcıların <, >, & gibi özel karakterleri gösterebilmesi için kullanılan bir yöntemdir.

Örnek;

En çok kullanılan karakterler:

<

<

>

>

"

"

&

&

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <p>&quot;Web Yapıları &amp; Web Güvenliği&quot;</p>
    <p>HTML yapıları &lt;HTML&gt; etiketiyle başlar.
  </body>
</html>
```

"Web Yapıları & Web Güvenliği"

HTML yapıları <HTML> etiketiyle başlar.

Önemli HTML Etiketleri

- ``
 - Bu imajın görüntülenmesi için kullanılan etikettir.
- `Beykoz Üniversitesi`
 - Bir linkin görüntülenmesi için kullanılan etikettir.
- `<iframe src="deneme.asp" width="300px" height="200px"></iframe>`
 - İçeriğinde deneme.asp sayfasını gösterecek olan ve yüksekliği 200px, genişliği 300px olan bir iframe (çerçeve) etiketidir.
- `<input type="text" name="isim"/>`
 - Web sayfamıza girdi noktası (text nesnesi) sağlayan etikettir.

Kaynakça

- Yazılım Güvenliği Saldırı ve Savunma – Bünyamin Demir
- <http://www.wikizero.biz/index.php?q=aHRocHM6Ly9oci53aWtpcGVkaWEub3JnL3dpa2kvRnV6emluZw>
- https://www.keytorc.com/blog/kod-kalitesi-ve-kod-analizi-serisi-3-static-code-analysis_8767/
- https://www.keytorc.com/blog/kod-kalitesi-ve-kod-analizi-serisi-4-dynamic-code-analysis_8847/