Web Programlama 2

Ders 10: Güvenlik

Kimliklendirme/Yetkilendirme

- **Kimliklendirme**:
  - X kullanıcısının kim olduğunu biliyor muyum?
  - X ve Y kullanıcısının farklı olduğunu nasıl ayırabilirim?
- **Yetkilendirme**:
  - X kullanıcısını biliyorum ancak X kullanıcısının ne yapmaya izni var?
  - Veri silebilir mi?
  - Diğer kullanıcıların verilerini görebilir mi?
  - vs.

Kimliklendirme/Yetkilendirme hataları

- Eğer kimliklendirme gerçekleşmezse, sunucu:
  - Login sayfasına yönlendirebilir, HTTP durum kodu: 3xx
  - Hata sayfası, HTTP durum kodu 401 *Unauthorized*
- Kimliklendirilmiş ancak yetkilendirilmediyse
  - Ör X kullanıcısı Y'nin verilerine erişmeye çalışıyorsa
  - 3xx yönlendirmesi
  - HTTP durum kodu: 403 *Forbidden*

Blacklist vs Whitelist

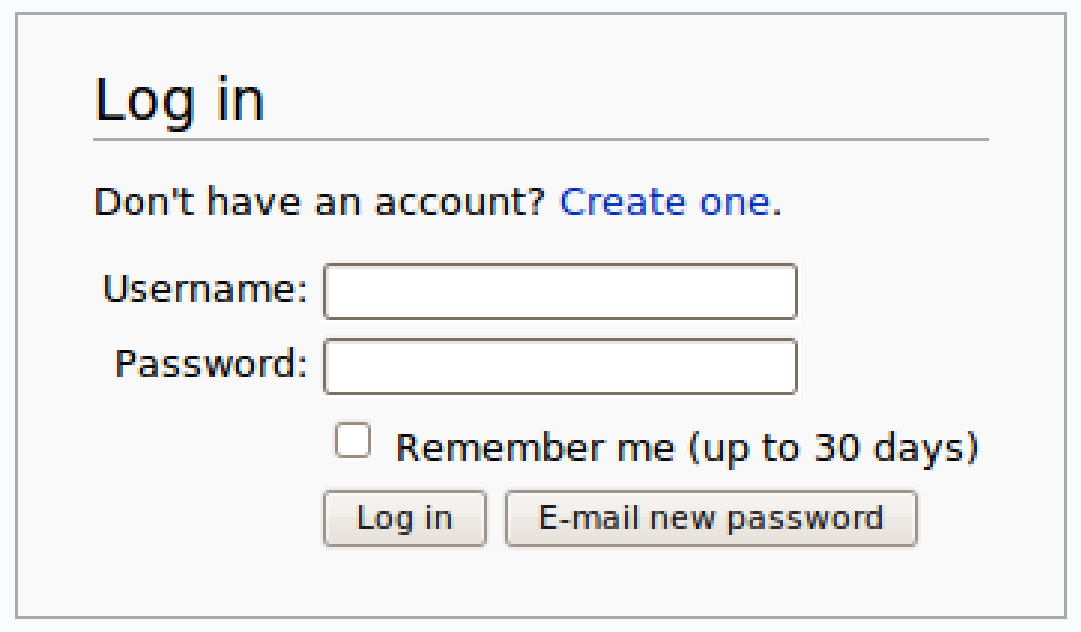
- Kimliklendirme sunucu tarafında gerçekleştirilir ve dil/framework bağımlıdır
  - JEE, Spring, .Net, NodeJS, vs.
  - Kullanıcı yalnızca 3xx veya 403 cevabı alır
- *Blacklist*: varsayılan olarak her şeye izin verilir. Kullanıcı/grup neye yetkili değilse açıkça belirtilir
  - Genellikle iyi bir fikir değildir, oldukça kritik operasyonları engellemek rahatlıkla unutulabilir
- *Whitelist*: varsayılan olarak hiçbir şeye izin verilmez. Neye izin verildiği açıkça belirtilir
  - “bir şeye izin vermeyi unutmak” (fonksiyonelliği azaltır), “bir şeyi engellemeyi unutmaktan” (güvenlik problemi) çok çok çok çok daha iyidir

Kimliklendirme: ilk adım

- Sunucu kullanıcının kim olduğunu bilemez
- Sunucu yalnızca gelen HTTP/S mesajlarını görür
  - Bir tarayıcıya ihtiyaç bulunmaz, kullanıcı TCp bağlantısını script aracılığı ile de kurabilir
- HTTP/S durumsuzdur
- HTTP/S çağrılarının aynı kullanıcıdan geldiğini söylemek gerekmektedir
- Kullanıcı her bir HTTP/S isteğinde kim olduğuna dair bir bilgi göndermelidir
- Ancak kullanıcılar **yalan** söyleyebilir (ör, hackerlar)

Id ve Password

- Bir kullanıcı tekil bir id ile kayıt olur
- Ayrıca giriş için gizli bir şifresi bulunur
  - Aksi takdirde diğer kullanıcılar da aynı id ile giriş yapabilir
- HTTP/S diğer kullanıcıların giriş denemelerini engellemez



Login mekanizması nasıl uygulanır?

- Güvenliğin sunucuda nasıl konuşulacağı düşünüldüğünde mutlaka tarayıcıdan gelmeyen HTTP/S mesajlar da düşünülmelidir
- Verilen id/password ile bir token dönen endpoint kullanılabilir
  - Burada gelen token sonraki isteklerde parametre olarak kullanılır
- GET /login?**userId=x**&**password=y**
  - /login endpointine userId/password parametreleri URL parametresi olarak gönderilebilir
  - Bir kullanıcıya bağlı Z token'ı da bir HTTP/S response body değeri olarak dönebilir
- GET /somePageIWantToBrowse**?token=z**
  - Her bir HTTP/S isteğinde de "token=z" değeri parametre olarak gönderilebilir

Ancak Bu Rezalet Bir Çözümdür

- Bu çözüm HTTPS ile çalışabilir ancak…
- “*/login?userId=x&password=y*” adresi çıkış yapsanız da tarayıcı geçmişinde cache’lenebilir
- Bir HTML sayfasındaki bütün <a> taglarına nasıl “*?token=z*” değeri ekleyeceksiniz?
  - Yapılabilir ancak oldukça maliyetlidir
- Tarayıcı imlerini nasıl çözeceksiniz?
  - Token’lar da imlere eklenir ancak çıkış yaptıktan sonra bu token geçersiz olacaktır

POST ve Cookie

- Kullanıcı id ve şifresi asla GET ile gönderilmemelidir
  - GET özellikleri isteğin body'sinde veri gönderilmesine izin vermez
- POST metodunun HTTP body'sinde gönderilmelidir
  - Ayrıca bu HTML'deki <input> tagının varsayılan davranışıdır
- Authentication "token" URL içinde değil HTTP Header içerisinde gönderilmelidir
- **Cookie**: kullanıcıyı tanımlamak için kullanılan özelleşmiş bir header'dır
- Kullanıcı cookie değerini belirleyemez, sunucu kullanıcıya atma yapar
- Hatırlatma: kullanıcı kendi HTTP mesajını oluşturabilir bu yüzden sunucu cookie değeri geçerli bir değer midir kontrol etmesi gerekir

Cookie ile Login

- Tarayıcı: POST /login
  - Kullanıcı adı X ve şifre HTTP body içerisinde
- Sunucu: eğer giriş başarılı ise POST cevabı olarak tekil bir tanımlayıcı Y ile «Set-Cookie» header'ı döner
  - Sunucunun bu Y cookie'sinin X ile ilişkili olduğunu hatırlaması gerekir
  - *Set-Cookie: <cookie-name>=<cookie-value>*
- Tarayıcı: Bundan sonra her bir HTTP isteğinde "*Cookie: Y*" header bilgisi gönderilir
- Logout: Y cookie'si ile X kullanıcısının ilişkisi sonlandırılır
- Sunucu: HTTP isteği cookie bulunmadığı/geçersiz olduğu için 3xx durum kodu ile login sayfasına yönlendirir

Cookie ve Session

- Sunucular genellikle login işleminden bağımsız olarak «Set-Cookie» headerını gönderirler
  - Kimliklendirmeden bağımsız olarak isteklerin aynı kullanıcıdan mı geldiği bilinmek istenebilir
  - Cookie’ler session tanımlamak için kullanılabilir
- Giriş yaptıktan sonra session oluşturulabileceği gibi (eski cookie geçersiz kılınır ve yenisi oluşturulur) var olan session cookie'si (örnek: ilk GET ile çekilen login sayfasıyla birlikte gelen cookie bilgisi) de kullanılabilir
- Session cookie bilgisi tekrar kullanılıyor ise bütün sayfaların HTTPS olarak kullanılması gerekir
  - Login sayfası da dahil olmak üzere hepsinde HTTPS kullanın
  - Önce HTTP ile giriş yapıp daha sonra HTPPS ile değiştirmeyin

Cookie’lerin saklanması

- Tarayıcılar yerel olarak cookie'leri saklayabilir
- Her bir HTTP/S isteğinde bu cookie HTTP header içerisinde gönderilir
- Cookie'ler yalnızca onları ayarlamasını isteyen sunucuya gönderilir
  - Örnek: foo.com tarafından ayarlanmış bir cookie bar.org adresine gönderilmez
- JavaScript cookie değerlerini okuyabilir
- Peki neden problem yaratır?
  - JS ile bütün cookie'leri okuyan bir web sayfası oluşturabilir ve kullanıcının Google/Facebook/Banka hesabı gibi bilgilere erişim sağlayabilirsiniz
- Çerezler isteğe bağlı diziler olduğundan veri depolamak için kullanılabilirler.
  - genellikle domain başına 4K bayta kadar bir veri tarayıcıda saklanabilir

Expires / Secure / HttpOnly

- **Set-Cookie: <name>=<value>; Expires=<date>; Secure; HttpOnly**
- *Expires*: Cookie ne kadar süreliğine saklanacak
- *Secure*: Cookie yalnızca HTTPS üzerinden gönderilir
  - Bazı saldırı türlerinde aynı sunucuya HTTPS yerine HTTP istek yapılması vardır. Böylelikle cookie bilgileri plain text olarak okunabilir
- *HttpOnly*: JS'in cookie değerlerini okumasına izin vermez
  - Kimliklendirme cookie'leri için oldukça önemlidir

Cookie Takibi

- Sunucu tarafından session/login cookieleri hariç başka cookieler de ayarlanabilir
- Çerez kullanımı ile ilgili çeşitli yasalar bulunmaktadır
- Peki neden? Çünkü takip edilme ve gizlilik endişeleri sebebiyle…

Tracking (Takip)

- Pek çok site diğer sitelerdeki kaynaklara ihtiyaç duymaktadır
  - Resim, JavaScript, CSS dosyaları, vs.
  - Facebook'taki Like butonu
- Y adresindeki kaynakları kullanan X adresindeki HTML sayfası indirildiğinde HTTP Y adresine, Y'ye ait önceki cookieleri de içeren de GET isteği yapar
- Facebook'tan çıkış yapsanız bile hangi sayfaları gezdiğini bilebilir
- Daha da kötüsü hiç FB kullanmasanız bile FB tarayıcınızı takip edebilir!!!
- Bu yalnızca X sayfasını açmanızla birlikte gerçekleşecektir. Herhangi bir şeye tıklamanıza gerek yoktur
- *referrer* HTTP header: Y'den gelmeyen ancak Y'ye yapılan isteklerin alan adı orijini
  - Örnek: Referer:X X'den Y'ye istek atıldığında eklenir

AddToCart-standart.js
format:webp
format:webp
osd.js?cb=%2Fr20100101
format:webp
format:webp
8390192428486553754
ext.js
track
format:webp
format:webp
rx_lidar.js?cache=r20110914
?id=891502007900202&ev=Microdata&d
view?xai=AKAOjsvvWRhmvxk9WOOaUKR
container.html
container.html
view?xai=AKAOjsvhUrlVHE_FpVMCtGedm
data:image/png;base...
webtrekk.js
sodar?sv=200&tid=gpt&tv=20210420018
pixel?d=KAE
track
appboy.min.js
sodar2.js
7f85a56ba4.css
data/

Request Method: GET

Status Code: 200

Remote Address: 185.60.218.35:443

Referrer Policy: no-referrer-when-downgrade

▶ **Response Headers (12)**

▼ **Request Headers**

:authority: www.facebook.com

:method: GET

:path: /tr/?id=[redacted]v=Microdata&dl=https%3A%2F%2Fwww.hepsiburada.com%2F&rl=&if=false&te=[redacted]cd[DataLayer]=%5B%5D&cd[Meta]=%7B%22title%22%3A%22T%C3%BCrkiye%27nin%20En%20B%C3%BCy%C3%BCk%20Online%20Al%C4%B1%C5%9Fveri%C5%9F%20Sitesi%20Hepsiburada.com%22%7D&cd[OpenGraph]=%7B%7D&cd[Schema.org]=%5B%5D&cd[JSON-LD]=%5B%5D&sw=1920&sh=1080&v=2.9.39&r=stable&ec=1&o=30&fbp=fb.1.1619296653045.144861492&it=1619296652670&coo=false&es=automatic&tm=3&rqm=GET

:scheme: https

accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8

accept-encoding: gzip, deflate, br

accept-language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7

cache-control: no-cache

cookie: datr=tyRHXRP_Fs4FEoG8xpBhIL90; sb=PW2BX5_0_k79XV8h-0fHZzSY; fr=0dli8sfPUcde5oS61..Bc3qoO.UA.GBp.0.0.Bggt3Q.

pragma: no-cache

referer: https://www.hepsiburada.com/

Passwords

- Kullanıcının doğrulanmasına ihtiyaç vardır
- Çok kısa ve basit olmamalıdır. Aksi takdirde brute-force ile rahatlıkla kırılabilir
- *Güvenlik vs Kullanılabilirlik:* İyi bir dengenin yakalanması zordur
  - Örnek: İdealde her web sayfası için farklı bir şifre olmalıdır ve bunlar da sıklıkla değiştirilmelidir. Ancak bunu kim, neden yapar ki? :)

Şifre Saklama

- Kullanıcı oluşturulduğunda şifreyi saklamak için bir alana ihtiyaç bulunmaktadır (genellikle veritabanı)
- **ASLA ŞİFRELERİ PLAIN TEXT OLARAK SAKLAMAYIN**
- Şifreler hashlenmelidir
- Hacker veritabanına tam erişimde bulunsa bile şifreyi elde edemeyecektir
  - SQL Injection ataklarındaki tipik örnektir
  - Ancak pek çok durumda karşımıza çıkabilir örneğin mutsuz bir çalışan veya çöpe atılmış bir hard disk :)
- Bir hacker aynı şifreyi farklı sitelere girişte de kullanmaya çalışabilir (Amazon/Facebook/Instagram vs.)

Hash Fonksiyonu

- *h(x) = y*
- Yalnızca x'den y'ye matematiksel bir fonksiyondur
  - Bizim senaryomuzda, *x* şifre, y ise hashlenmiş değerdir
- Deterministik: her zaman aynı x ile aynı y elde edilir
- h() öğesinin nasıl uygulandığı hakkında tam bilginiz olsa bile, x'i y ile bulamamalısınız.
- x'den x' e yapılan küçük değişiklik y ile y' arasındaki farkı oldukça büyük yapmalıdır
  - ör, *y ve y' arasında bir korelasyon görülmemelidir böylelikle x ile x' birbirine benzerdir denilememelidir*
- Çakışma olmamalı: iki farklı değer aynı hash değerine sahip olmamalıdır, ör: *h(x) = y = h(z)*

Hash’lenmiş Şifre ile Giriş

- X şifresi ile giriş yapan A kullanıcısının başarılı bir şekilde giriş yaptığını sunucu şifreyi değil yalnızca Y=h(x) hash değerini biliyorsa nasıl anlayabilir?
- Sunucunun veritabanından A kullanıcısına ait Y hash değerini alması, daha sonra verilen X şifresi için Y hash değerini tekrardan hesaplaması bu hash değerlerini karşılaştırması (Y==h(X)) gerekir

Salted Şifreler

- Kullanıcının oldukça uzun şifrelere sahip olması beklenemez
- Bir hacker veritabanına erişim sağlarsa belli bir uzunluğa sahip (ör N=8) bütün karakterler için hash değerini hesaplayıp veritabanında ar olup olmadığını kontrol edebilir
- Küçük N için bu yapılabilirdir. Hatta bu değerler önceden hesaplı olduğu için (ör Rainbow Table) h()'i çalıştırmaya bile gerek yoktur
- Bir diğer problem: aynı şifreye sahip iki kullanıcı aynı Y hash değerine sahip olacaktır
- Çözüm: Şifreyi hashlemeden önce rastgele S salt (tuz) değeri (rastgele uzun bir string gibi) ekle ve bu salt değeriyle birlikte şifreyi bir arada veritabanında tut
- $h(X+S)=Y$
- Her bir kullanıcının kendine ait rastgele salt değeri olacaktır

Pepper

- Eğer hacker veritabanına erişirse salt değerini okuyabilir
- Hala hash kodunu kırmak kolay değildir ancak yapılabilirdir
- *Pepper*: hash değeri hesaplanmadan önce eklenen bir diğer rastgele random değer
- Veritabanı DIŞINDA bir yerde tutulur
  - dosya, uzak sunucu veya kaynak kod içinde hard coded gibi
- Bütün uygulama için tek bir pepper değeri bulunur (kullanıcı için değil)
- Hacker veritabanına erişse (SQL injection gibi) bile pepper değerini okuyamayacağı için işi çok daha zor olacaktır

Hash Fonksiyon Hızı

- Kırılmasının zorlaşması için hesaplama maliyeti yüksek düşük hızlı fonksiyonlar isteyebilirsiniz
  - Ancak yine de kimliklendirmede kullanılabilmesi için yönetilebilir bir sürede olması gerekir
- *BCrypt* şifreler için en ünlü hash fonksiyonudur
- Ancak, herhangi bir hash fonksiyonunu (ör SHA256) döngüler aracılığı ile N sefer tekrarlayarak yavaşlatabilirsiniz
  - *ör, N=6 -> h(h(h(h(h(h(x))))))) = y*

Spring Security

Slaytlar Hakkında

- Bu slaytlar oldukça yüksek seviyede genel bir bakış açısı sunmaktadır
- Detaylar Git repository’sindeki yorum satırlarında bulunmaktadır

Güvenlik oldukça zorlu bir süreçtir

- Güvenlik için kendi çözümlerinizi sunmamalısınız
  - Hata yapmaya oldukça açık bir alandır
- Var olan ve test edilmiş frameworkleri kullanmalısınız
- *Spring Security*: Spring'in güvenlik süreçlerini kontrol etmenizi sağlayan modülüdür
- Ancak iç yapısının nasıl işlediğini anlamanız oldukça önemlidir

Ayarlamalar

- *WebSecurityConfigurerAdapter'*den extend edilmiş *@Configuration* bean'ine ihtiyaç bulunmaktadır
- Ayrıca *@EnableWebSecurity* annotation'ı gerekmektedir
- Daha sonra böyle bir sınıfla metotlar override edilebilir:
  - Kimliklendirme için: *configure(AuthenticationManagerBuilder auth)*, ör kullanıcı ve şifre veritabanında bulunuyor mu
  - Yetkilendirme için: *configure(HttpSecurity http*, ör erişim kuralları için

OWASP

Open Web Application Security Project (OWASP)

- www.owasp.org
- Yazılım güvenliği için kar amacı bulunmayan bir organizasyondur
- Yazılım güvenliğini öğreneceğiniz temel kaynaklardan biridir
- Ayrıca bazı açık kaynak araçlar sunar (ör penetrasyon testi için *ZAP*)
- Maven plugin: *dependency-check-maven*
  - Bilinen açıklar için 3. parti bağımlılıklarınızı tarar
  - Güncellenen veritabanına otomatik olarak bağlanır
  - Not: bu günlerde GitHub gibi servisler bağımlılıkları otomatik olarak kontrol etmektedir

Git Repository Modülü

- *NOT: açıklamaların büyük bir çoğunluğu kod içerisinde yorum satırı olarak bulunmaktadır, burada slaytlarda bulunmamaktadır*
- **intro/spring/security/manual**
- **intro/spring/security/framework**
- **intro/spring/security/dependencies**
- Ders 10 alıştırması (dokümantasyona bakınız)