Kod Uygulamaları

Gerekli tahmini süre: 20 dakika

Giriş

Kod Uygulamaları için uygulamalı laboratuvara hoş geldiniz. Daha güvenlik odaklı bir geliştirici olmak için benimsemeniz gereken birkaç kod uygulaması vardır. GitHub'da gizli bilgileri güvenli bir şekilde saklamak ve Hashicorp Vault veya Linux pass komutu gibi bir gizli yönetici kullanmak iyi bir başlangıçtır. Ancak, kodunuzu daha güvenli hale getirme pratiği de yapmalısınız.

Bu laboratuvar, kodunuzu nasıl daha güvenli hale getirebileceğinizi inceleyecek ve öğrendiklerinizi uygulayacaksınız. Flask çerçevesiyle yazılmış Python web uygulamalarında daha güvenli kod yazma pratiği yapacaksınız.

Öğrenme Hedefleri

Bu laboratuvarı tamamladıktan sonra şunları yapabileceksiniz:

- `flask-talisman` kullanarak HTTP güvenlik başlıkları oluşturmak
- Python Flask uygulamasının kod uygulamalarını değerlendirmek
- Bir arka uç uygulama kullanıcı arayüzü kurmak ve başlatmak
- `flask-cors` kullanarak Cross Origin Resource Sharing (CORS) politikalarını belirlemek

Güvenlik Kodu Uygulamaları

Güvenli uygulamalar geliştirmek için, geliştiricilerin DevSecOps modelini takip etmeleri gerekir. Bu model, güvenlik endişelerini daha yüksek bir önceliğe kaydırır, böylece yazılım geliştirme yaşam döngüsünün erken aşamalarında ele alınır.

Güvenlik Kodu Uygulamaları, geliştiricilerin yazılım geliştirme yaşam döngüsünün başlangıcından itibaren uygulamalarının güvenli olmasını sağlamak için takip ettikleri bir dizi uygulamadır. Takip edeceğimiz bazı uygulamalar şunlardır:

- HTTP başlıklarını ayarlamak
- Cross Origin Resource Sourcing (CORS) politikalarını uygulamak
- Kimlik bilgileri ve GitHub ile çalışmak
- Vault'u kullanmak

HTTP Başlıklarıyla Güvenliği Artırma

Uygulamanızın güvenliğini artırmak için yapabileceğiniz ilk şey güvenli HTTP başlıkları ayarlamaktır. Bunu, Flask uygulamanızı `Flask-Talisman` adlı bir Python paketi ile sarmalayarak basitçe yapabilirsiniz.

Örnek

Aşağıda, güvenlik uygulamayan çok temel bir Flask uygulaması bulunmaktadır.

```
from flask import Flask
app = Flask(__name__)
@app.route('/', methods=['GET'])
def index():
    """Base URL for our service"""
    return app.send_static_file("index.html")
```

Bu uygulama, kök `/` URL'sinden `index.html` adlı statik bir HTML sayfasını geri gönderir.

Güvenlik İyileştirmesi

Uygulamanın bu versiyonu, içeriğin diğer sitelerden yüklenmesini reddeden güvenlik başlıkları eklemek için `Flask-Talisman` kullanır, böylece uygulamanızı daha güvenli hale getirir.

```
from flask import Flask
from flask_talisman import Talisman
app = Flask(__name__)
# Create a content security policy and apply it
csp = {
    'default-src': '\'self\''
}
talisman = Talisman(app, content_security_policy=csp)
@app.route('/', methods=['GET'])
def index():
    """Base URL for our service"""
    return app.send_static_file("index.html")
```

Dikkat ettiniz mi, sadece iki satır kod eklendi? Bu iki basit satır, her yanıta güvenlik başlıkları ekleyerek bu uygulamayı daha güvenli hale getirdi. Eğer biri, orijinal Web sitesinin dışından içerik yüklemeye çalışan bir script enjeksiyonu saldırısı yapmaya çalışırsa, güvenlik politikası sayesinde hiçbir şey yapmanıza gerek kalmadan engellenecektir.

Bu uygulamanın daha güvenli hale gelmesine neden olanlara bir göz atalım:

1. Satır **2**, `flask_talisman` paketinden `Talisman` sınıfını içe aktarıyor; bunu `pip install flask-talisman` ile kurabilirsiniz.
2. Satırlar **6 - 8**, Talisman için bir içerik güvenlik politikası oluşturuyor. Bu varsayılan politikadır. Bir Web sitesi yöneticisi, tüm içeriğin sitenin kendi kaynağından gelmesini ister (bu alt alanları dışlar). Bu, Web sitenizde diğer sitelerden yüklenen kütüphaneler, resimler ve yazı tipleri gibi şeylerin olamayacağı anlamına gelir. Eğer buna ihtiyacınız varsa, bunu içerik güvenlik politikasında belirtmelisiniz.
3. Satır **9**, satır **4**'teki Flask uygulamasını ve satır **6**'daki içerik güvenlik politikasını geçirerek bir Talisman nesnesi oluşturur.

Sonuçlar

```
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'self'
Referrer-Policy: strict-origin-when-cross-origin
```

Yukarıdaki çıktı, `X-Frame-Options`, `X-XSS-Protection` ve `X-Content-Type-Options` gibi Talisman kullanılmadan önce mevcut olmayan ek güvenlik başlıklarını göstermektedir. Ayrıca, isteklerin yalnızca aynı kökenden yapılmasını sağlamak için bir `Content-Security-Policy` ve `Referrer-Policy` eklenmiştir.

Talisman ayrıca, alan adınızın HTTP yerine HTTPS kullanmasını sağlar. Cloud IDE ortamında, uygulamayı dahili olarak proxy'liyoruz, bu nedenle zaten HTTPS kullanıyor, ancak kendi uygulamalarınızda farkı göreceksiniz.

Gerekirse, güvenlik üzerinde daha fazla kontrol sağlamak için `@talisman` dekoratörünü de kullanabilirsiniz.

Şimdi Cross Origin Resource Sharing (CORS) konusuna daha derinlemesine bakalım.

Çapraz Kaynak Paylaşımı (CORS)

CORS'u açıklamak için bir örnekle başlayalım.

Diyelim ki, belirli bir müşteriye veri sağladığınız bir web uygulaması geliştirdiniz. Veriyi almak için müşterinizin kendi uygulaması, sizin uygulama uç noktanıza bir GET isteği gönderiyor. Ama ya rakibi sizi keşfederse ve verilerinizi isterse? Müşterinize gönderdiğiniz uygulama uç noktasını kullanabilirler!

Bu, **kaynak** olarak sizin uç noktanızdaki verinin, **kaynağın** ise uygulamanızın uç noktasını arayan uygulamanın adresi olduğu, çapraz köken kaynak paylaşımı olarak bilinir (müşterinizin uygulaması ve rakibinin uygulaması). Önemli olan, **farklı** bir kökenin başka bir köken üzerindeki kaynaklara erişmesidir.

Adım 1: Arka Uç Servisini Kurun

Arka uç servisi, Flask framework'ü kullanılarak Python'da yazılmıştır. Bu adımda, kodun bulunduğu GitHub deposunu `git` kullanarak klonlayacak, gerekli Python paketlerini `pip` Python Paket Yöneticisi ile yükleyecek ve uygulamayı `flask` kullanarak başlatacaksınız.

Göreviniz

1. Cloud IDE editörünün üst menü çubuğundan bir terminal açın `Terminal -> Yeni Terminal` veya aşağıdaki kodu çalıştırın:

```
cd /home/project
```

2. Aşağıdaki `git clone` komutunu çalıştırarak kodu depodan indirin ve ardından oluşturulan `DevSecOps-HTTP-app` klasörüne `cd` ile geçin:

```
git clone https://github.com/ibm-developer-skills-network/DevSecOps-HTTP-app.git
cd DevSecOps-HTTP-app/
```

3. Bu laboratuvar için yalnızca gerekli Python paket bağımlılıklarını yüklemek üzere aşağıdaki `pip install` komutunu çalıştırın.

```
pip install --user -r requirements.txt
```

4. Uygulama arka ucunu başlatmak için aşağıdaki `flask` komutunu çalıştırın.

```
flask run --reload
```

Not: `--reload` seçeneği, Flask'a kaynak kodda herhangi bir değişiklik yapıldığında uygulamayı yeniden yüklemesini söyler. Bu, geliştirme sırasında özellikle faydalıdır.

Sonuçlar

Aşağıdakine benzer bir çıktı görmelisiniz:

Tebrikler! Flask uygulamasını başarıyla başlattınız. Uygulama artık `localhost` üzerinde çalışıyor ve `5000` portunu (Flask için varsayılan port) dinliyor olmalıdır.

Adım 2: Arka uç uygulama UI'sını başlat

Artık arka uç Flask uygulaması çalıştığına göre, düzgün çalıştığından emin olmak için UI'yi başlatabilirsiniz.

Göreviniz

1. Skills Network eklentisini açın ve `Launch Application`'ı `5000` portunu ve `/posts` yolunu kullanarak seçin veya aşağıdaki `[Launch Application]` butonuna basın.

Launch Application

Web sayfası şöyle görünmelidir:

2. Uygulama URL'sine `/posts` yolunu ekleyin ve istemci uygulamasına dönecek verileri görmek için ENTER'a basın.

Sonuçlar

İstemciye geri gönderilen ham JSON verilerini görmelisiniz. Verilerinizin farklı olacağını unutmayın çünkü arka uç her çağırdığınızda rastgele gönderiler oluşturur. Önemli olan, JSON verilerinin geri döndüğünü görmenizdir.

Bu, `/posts` uç noktasının istemci uygulamasına döndürdüğü ham `json` verisidir.

Veri sağlayacak arka uç API'sini başarıyla kurdunuz!

Adım 3: İstemci uygulamasını ayarlayın

Artık verilerimizi satın alan istemci uygulamasını ayarlayacağız. Hadi bunu ayarlayalım.

Öncelikle, ön uç uygulamasını yeni dağıttığınız arka ucu çağıracak şekilde değiştirmeniz gerekiyor. Bunu yapmak için, arka ucunuzun URL'si ile ön uç kodunu değiştirmeniz gerekecek; bu, bir Cloud IDE kullanıcısı olarak size özeldir.

Göreviniz

1. Öncelikle, uygulama sekmesine gidin ve Flask uygulamasının URL'sini panoya kopyalayın.

   (Bu, `https://user-5000.theia-0-labs-prod-misc-tools-us-east-0.proxy.cognitiveclass.ai/posts` gibi görünmelidir; burada `user`, Theia kullanıcı kimliğinizdir)

2. Ardından, `/home/project/DevSecOps-HTTP-app/frontend/src/components/` dizinine gidin ve IDE editöründe `data.jsx` dosyasını açın veya aşağıdaki düğmeye basın:

Open **data.jsx** in IDE

3. Şimdi, 10. satırdaki `http://127.0.0.1:5000/posts` adresini kopyaladığınız URL ile değiştirin.

   URL'nin sonuna `/posts` eklediğinizden emin olun!

   IDE, değişikliklerinizi otomatik olarak kaydetmelidir.

4. Son olarak, `Terminal -> Yeni Terminal` ile yeni bir terminal penceresi açın ve ön uç uygulamasını başlatmak için aşağıdaki komutları çalıştırın:

```
cd /home/project/DevSecOps-HTTP-app/frontend/
npm i
npm start
```

   Not: Uygulamanın başlaması için tüm JavaScript kütüphanelerinin indirilmesi uzun sürecektir.

Sonuçlar

Bu, herhangi bir hata olmadan ön uç uygulamasını başlatmalıdır, ancak bazı uyarılar olabilir.

Adım 4: İstemci uygulaması arayüzünü başlat

Önceki adımda istemci uygulamasını başlattıktan sonra, web arayüzünü başlatma ve çalışıp çalışmadığını görme zamanı.

Göreviniz

1. Skills Network uzantısını açın ve `Uygulamayı Başlat` seçeneğini `3000` portunu kullanarak seçin veya aşağıdaki `[Uygulamayı Başlat]` düğmesine basın.

Uygulamayı Başlat

Sonuçlar

Aşağıdakine benzer bir şey görmelisiniz:

Ve sayfaya sağ tıklayıp `İncele` veya `Cmd + Shift + C` seçeneğini seçerek açabileceğiniz konsolda, aşağıdaki örneğe benzer bir şey görmelisiniz. Alan adlarının farklı olacağını unutmayın.

Arka uca yapılan istek, yalnızca orijinden erişime izin veren bir CORS politikası tarafından engellenmiştir. Bunu bir sonraki adımda düzelteceksiniz.

Adım 5: CORS politikasını güncelle

İstemci ön uç uygulamanızın arka uç ile iletişim kurabilmesi için, istemci uygulamasının URL'sini arka uçtaki CORS politikasına eklemeniz gerekmektedir. Bunu `app.py` dosyasını düzenleyerek ve varsayılan `http://localhost:3000` adresini uygulamanızın Cloud IDE ortamında çalıştığı URL ile değiştirerek yapabilirsiniz.

Göreviniz

1. Öncelikle, istemci uygulamasının URL'sini panoya kopyalayın.

   (Bu, `user` kısmı sizin Theia kullanıcı ID'niz olacak şekilde `https://user-3000.theia-0-labs-prod-misc-tools-us-east-0.proxy.cognitiveclass.ai` gibi görünmelidir)

2. Ardından, `DevSecOps-HTTP-app` klasöründeki `app.py` dosyasını açın veya aşağıdaki `[Open App.py in IDE]` butonuna basın:

Open **app.py** in IDE

3. Son olarak, **14. satırda**, `http://localhost:3000` adresini kopyaladığınız URL ile değiştirin.

   URL'nin sonunda `/` olmadığından emin olun

Artık uygulamayı yeniden yüklemeye ve bu değişikliğin sonuçlarını görmeye hazırsınız.

Adım 6: Uygulama Arayüzünü Yenile

Artık istemci uygulamanızı yenilemeniz gerekiyor. Web sayfasını yenilemek için iç tarayıcıdaki yenile butonuna basmanız yeterli olmalı. Eğer yüklenmezse, bağlantıyı kopyalayıp tarayıcınızdaki Özel/İnce Modda yeniden açmanız gerekebilir.

Sonuçlar

Ön uç uygulamanızda arka uçtan alınan bir veri tablosu görmelisiniz. Verileriniz farklılık gösterecektir çünkü arka uç her çağrıda yeni veriler üretir. Önemli olan, artık bir hata mesajı değil, bir veri tablosu görmenizdir.

Artık verilerinize erişimi kısıtlamayı öğrendiniz, böylece yalnızca belirttiğiniz kişiler erişebilir. İstemciniz aşağıdakine benzer bir şey görmelidir:

Bu, uygulamamızın bildiğimiz istekler tarafından güvenli bir şekilde erişildiği ve bilmediğimiz istekleri reddedeceği anlamına geliyor! Üretim ortamında, geliştiriciler izin vermek istedikleri kaynakları/URL'leri ortam değişkenlerinde saklar, çünkü bir istemci uygulamasının kaynağı sahneleme veya test ortamlarında farklı olabilir.

Sonuç

Bu laboratuvar çalışmasında, CORS'un bir tarayıcıda nasıl çalıştığını, belirli kaynaklara erişim izni verilmeyen kökenlerden gelen tüm istekleri engelleyerek öğrendiniz. Ayrıca HTTP güvenlik başlıkları oluşturdunuz ve arka uç uygulama UI'ları ile istemci uygulama UI'ları kurarak, verilerinizi özel olarak bir istemciyle paylaşmanıza yardımcı olmak için CORS'u etkinleştirdiniz.

Ayrıca `flask-talisman` uygulayarak, flask uygulamanızı anında daha güvenli hale getirebileceğinizi öğrendiniz.

Sonraki adımlar

Belirli flask uygulamanızı daha güvenli hale getirmek için HTTP başlıklarını nasıl uygulayabileceğinizi öğrenmek için flask-talisman sitesini ziyaret edin. Ayrıca, Python Flask uygulamalarınıza `Flask-CORS` ve `Flask-Talisman` ekleyerek güvenliklerini artırmayı deneyin.

Author(s)

Richard Ye
John J. Rofrano