Tytuł szkolenia: Bezpieczeństwo aplikacji Spring, Angular, Node.js

Kod szkolenia: W-BEZP-SYS-ROZPR

Wprowadzenie

Bezpieczeństwo aplikacji rozproszonych to temat, którego zgłębienie wymaga dużego wysiłku nie wspominając już o pracy jaką trzeba włożyć w **implementację zabezpieczeń**. Powyższe stwierdzenie to oczywiście stereotyp... Stosując odpowiednie techniki i narzędzia można stosunkowo niewielkim nakładem pracy, zwinnie wdrożyć **politykę bezpieczeństwa** na poziomie wystarczającym i zadowalającym.

W takim razie, dlaczego tak wiele systemów ma rażące luki w bezpieczeństwie, które mogą być wykorzystane w celu nieuprawnionego pozyskania danych lub destabilizacji pracy aplikacji? Z jakiego powodu częstą, złą praktyką, jest ciągłe odkładanie zadań związanych z implementacją zabezpieczeń "na później"? Dlaczego zbyt często zespołom brakuje wspólnej wizji sposobu implementacji systemu zabezpieczeń? Z jakiego powodu w zespołach projektowych począwszy od kadry zarządzającej przez programistów i (już rzadziej) testerów powielany jest stereotyp, że wdrożenie polityki bezpieczeństwa jest czasochłonne i bardzo trudne? Czy naprawdę potrzeba aż tak bardzo specjalistycznej wiedzy i dodatkowego budżetu aby aplikację uczynić bezpieczną? I w końcu co to znaczy, że **aplikacja jest bezpieczna**? Kiedy jest bezpieczna?

W sprawie bezpieczeństwa, jak zresztą w każdej innej, **czynnik ludzki** jest bardzo ważny, jeśli nie najważniejszy. Z tego powodu nie można pozostawiać na marginesie zagadnień związanych z zarządzaniem zespołem, harmonogramem i codziennymi praktykami stosowanymi w zespole i warto mieć świadomość jak to wszystko wpływa na poziom bezpieczeństwa produktu końcowego.

Diabeł jak zwykle tkwi w szczegółach. Obok czynnika ludzkiego i dobrych praktyk w zespole mamy **inżynieryjne techniki implementacji zabezpieczeń**. Tu pojawia się wiele pytań: Jaki system autentykacji będzie odpowiedni w budowanym systemie? Jaka metoda autoryzacji będzie łatwa w modyfikacji przy rozbudowie systemu? Jak zaimplementować OAuth, SSO, JAAS, LDAP? Jak wyeliminować problem kodowania "na sztywno" systemu ról w kodzie aplikacji. Jak zabezpieczyć kanał komunikacyjny? Czy SSL wystarczy? Czym różni się podejście w implementacji bezpieczeństwa w przypadku aplikacji MVC, usługach opartych o RESTful, WebSocket, mikroserwisach oraz tzw. grubych klientach w postaci aplikacji SPA.

To tylko część pytań na jakie musi znać odpowiedź każdy, kto pracuje z aplikacją na poziomie projektu i kodu. **Zabezpieczenia** zawsze można podzielić na te znajdujące się po stronie usług i te na poziomie danych, choć nie zawsze ta granica jest wyraźna. Zabezpieczając warstwę dostępu do danych tu również pojawią się pytania: Jak zastosowany system ORM wpływa na poziom bezpieczeństwa? Jak implementować bezpieczny dostęp do danych gdy projekt pozbawiony jest warstwy ORM? Co z bezpieczeństwem w przypadku silników NoSQL?

Faza testów to ta część cyklu życia aplikacji, w której powinny na bierząco wypływać luki bezpieczeństwa. Ale w wielu projektach nie wypływają, ponieważ testy bezpieczeństwa nie są implementowane w odpowiednim zakresie lub z odpowiednią starannością. Koniec końców te luki prędzej czy później będą musiały zostać naprawione przez programistów. W takim razie pojawia się pytanie, czy już na poziomie pojedynczego stanowiska developerskiego można, w sposób nie powodujący istotnego odciągania programisty od zadań implementacyjnych kolejnych funkcjonalności, uzbroić go w narzędzia pozwalające wstępnie przetestować aplikację pod kontem luk w zabezpieczeniach w kodzie, który pisze? Przenosząc zagadnienie na wyższy poziom automatyzacji procesu, powstaje pytanie: Jak można, niewielkim nakładem pracy, wdrożyć testy bezpieczeństwa do systemu CI?

Każda **faza procesu**, każda warstwa budowanego systemu i poszczególnej jego poziomy to miejsca w których można i trzeba stawiać pytania o bezpieczeństwo i sposób jego wdrożenia. Pytań jest dużo, część z nich dotyczy aspektów miękkich, czynnika ludzkiego a część jest czysto inżynieryjna - techniczna, architektoniczna. Fakt, że tych pytań jest wiele i pojawiają się wszędzie, nie oznacza, że zadbanie o bezpieczeństwo budowanego systemu musi być zagadnieniem trudnym.

O tym jak wdrożyć politykę bezpieczeństwa w proces wytwórczy - jak zminimalizować czynnik ludzki i niewielkim kosztym zautomatyzować proces wykrywania luk i wreszcie jak zaimplementować zabezpieczenia w systemie dowiesz się na szkoleniu z bezpieczeństwa systemów rozproszonych.

Adresaci szkolenia

Szkolenie kierowane jest do **programistów** pracujących w **technologiach Java** oraz **JS** na platformie **Node.js** przy projektach aplikacji klasy enterprise.

Cel szkolenia

Pozyskanie, uzupełnienie i usystematyzowanie wiedzy z dziedziny **bezpieczeństwa aplikacji**.
Nabycie praktycznych umiejętności konfigurowania i implementacji polityki bezpieczeństwa w projekcie opartym o **Spring** oraz **Node.js** z uwzględnieniem aplikacji typu **SPA**.
Zrozumienie jak na bezpieczeństwo budowanego systemu wpływa przebieg procesu realizacji i zarządzanie projektem.
Wypracowanie spojrzenia na codzienne praktyki **Agile** uwzględniającego problem **bezpieczeństwa aplikacji**.

Czas i forma szkolenia

- 28 godzin (4 dni x 7 godzin), w tym wykłady i warsztaty praktyczne.

Plan szkolenia

Bezpieczeństwo
a. Modele bezpieczeństwa aplikacji
b. Rodzaje i charakterystyka ataków

Agile
a. Proces wytwórczy w kontekście bezpieczeństwa
b. Model dojrzałości procesu
c. Czynnik ludzki
d. Bezpieczny harmonogram
e. Dobre praktyki programistyczne / projektowe podnoszące bezpieczeństwo aplikacji
f. Continuous Integration, Continuous Delivery i Test Driven Development a bezpieczeństwo

Aplikacja
a. Architektura aplikacji rozproszonej i bezpieczeństwo
b. Architektura i bezpieczeństwo kanału komunikacyjnego
c. Autentykacja i autoryzacja
d. Bezpieczeństwo danych
e. Bezpieczeństwo sesji

Frameworki, konfiguracja i implementacja polityki bezpieczeństwa
a. Spring Security
b. Node.js
c. Single Page Application dla Ember i Angular

Narzędzia i techniki testowania
a. Projektowanie testów w kontekście bezpieczeństwa
b. Testy penetracyjne
c. Kali Linux