Um Framework de Testes Automatizados de Segurança em Api Rest

Fernando cruz

frlc@cesar.school

Abstract. *Nowadays, the world is totally interconnected, several applications on social networks are intertwined, making use of their basic communication functions. By the way, today your equipment tweets, your Nike plus displays your racing data on social networks.*
Startups use geographic maps from vendors like Google Maps and regardless of their programming language or platform. Everyone can take advantage of the Google team effort and create a FourSquare from this powerful map API.
I like to think that SOA is socializing your API, making it social, democratic and with endless possibilities of use, giving freedom to the creativade of the developers. However, what would happen if Google Maps were to go off the air? How many startups and even internal Google applications would be impacted? Developing this type of API requires a lot of responsibility from the developer, both in the quality of the API and in the maintenance of its state.
Based on the following list of known attacks: Cross Domain Attack, Broken Authentcation and Session Management, Missing Rate Limit, Cross-Site Request Forgery, JWT none algorithm vulnerability, JWT Sign Key, SQL Injection, XSS-Attack, Open redirection, XML External Entity Attack, CRLF Injection, Missing Security Headers This article provides developers and test analysts with guidelines on how to create more secure Restful APIs by running battery automated testing where the focus of the tests is primarily on the behavior at runtime of the services , developing a test framework in Python and BDD language aimed to this end.
Palavras-chave: *Automation, Security, Python, BDD*

Resumo. *Atualmente, o mundo está totalmente interligado, diversos aplicativos nas redes sociais estão entrelaçados, fazendo uso das suas funções básicas de comunicação. Aliás, hoje seus equipamentos twittam, seu Nike plus exibe seus dados de corrida nas redes sociais.*
Startups utilizam mapas geográficos de fornecedores como Google Maps não importando sua linguagem de programação ou plataforma. Todos podem se valer do esforço da equipe do Google e criar um FourSquare à partir dessa poderosa API de mapas.
Estamos na era de socialização da sua API, torná-la social, democrática e com infinitas possibilidades de utilização, dando liberdade à criatividade dos desenvolvedores. Contudo o que aconteceria se o Google Maps saísse do ar? Quantas startups e até mesmo aplicativos internos do

Google seriam impactados? Desenvolver esse tipo de API exige bastante responsabilidade do desenvolvedor tanto na qualidade da mesma, quanto na manutenção do seu estado.
Com base na seguinte lista de ataques conhecidos: Cross Domain Attack, Broken Authentcation and Session Management, Missing Rate limit, Cross-Site Request Forgery, JWT none algorithm vulnerability, Weak JWT sign key, SQL Injection, XSS-Attack, Open redirection, XML External Entity Attack, CRLF Injection, Security Headers Missing, este artigo fornece aos desenvolvedores e analistas de teste diretrizes sobre como criar APIs Restful mais seguras executando bateria de testes automatizados onde o foco dos testes é principalmente sobre o comportamento em tempo de execução dos serviços desenvolvendo um framework de teste em linguagem Python e BDD voltado para este fim.
Palavras-chave: *Automação, Security, BDD, Python.*
[Este é apenas um texto explicativo. Altere através do menu esquerdo.]

1. Introdução

A qualidade e segurança de dados dos sistemas é um enorme desafio mediante a alta complexidade dos atuais sistemas desenvolvidos envolvendo questões humanas, técnicas, de negócio e políticas. No cenário atual é comum juntamente com o desenvolvimento a execução muitas vezes de testes manuais para verificar se tudo está funcionando conforme a especificação sendo normal encontrar defeitos e vulnerabilidades. Os testes manuais bem como sua variante o pentest são rápidos e essenciais mas a execução e repetição destes testes manuais é uma tarefa muito onerosa e cansativa. Na busca de diminuição destes problemas são utilizadas formas para a automatização destes testes. Com objetivo resolver o problema da repetitividade casos de testes voltados a segurança da informação que necessitam serem executados em aplicações que utilizam o estilo arquitetural REST (Representational State Transfer), este artigo visa propor um framework de automação de testes de segurança para APIs com esta arquitertura. Criando uma bateria de testes automatizados que possa funcionar com o objetivo de proceder com testes de segurança para aplicações REST, e seus principais métodos GET, POST, PUT, DELETE, OPTIONS disponíveis no HTTP para responder às requisições.

1) 1.Objetivos

1.1.1. Objetivo Geral

Implementação de framework testes automatizados na linguagem Python utilizando padrão de projeto Page Objects e BDD, para a realização de testes de segurança para requisições do tipo REST. A principal motivação para construção deste projeto e agilizar execução testes de segurança a cada novo deploy de versão de API podendo ser integrado em pipelines para garantir assim o minimo de segurança nos dados bem como em suas transações.

2. Revisão da Literatura

- API Rest

REST é uma sigla que significa Representational State Transfer. REST é um modelo arquitetural concebido por um dos autores do protocolo HTTP (o doutor Roy Fielding), e tem como plataforma justamente as capacidades do protocolo, onde se destacam: Diferentes métodos (ou verbos) de comunicação (GET, POST, PUT, DELETE, HEAD, OPTIONS), utilização de headers HTTP (tanto padronizados quanto customizados), definição de arquivos como recursos (ou seja, cada um com seu próprio endereço), utilização de media types.
A própria web é baseada nesses princípios, e quando se navega por uma página, automaticamente está utilizando esses conceitos. REST é fortemente ligado ao HTTP (a ponto de não poder ser executado com sucesso em outros protocolos). Ele é baseado em diversos princípios que fizeram com que a própria web fosse um sucesso. Estes princípios são:

1) URLs bem definidas para recursos;
2) Utilização dos métodos HTTP de acordo com seus propósitos;
3) Utilização de media types efetiva;
4) Utilização de headers HTTP de maneira efetiva;
5) Utilização de códigos de status HTTP;
6) Utilização de Hipermídia como motor de estado da aplicação.

Os web services REST também devem ter métodos HTTP bem definidos para realizar as operações. Se quisermos que algo seja inserido em nossa base de recursos, utilizamos o método POST. Se quisermos recuperar dados, utilizamos o método GET. Se quisermos atualizar/apagar dados, utilizamos PUT ou DELETE, respectivamente, para utilização efetiva de nossos serviços REST, podemos (e devemos) utilizar os códigos de status HTTP para realizar a comunicação com nossos serviços. Isso facilita o reconhecimento do estado da requisição:

1) 200 OK
2) 201 Created
3) 202 Accepted
4) 204 No Content
5) 304 Not Modified
6) 400 Bad Request
7) 401 Unauthorized
8) 403 Forbidden
9) 404 Not Found
10) 405 Method Not Allowed
11) 409 Conflict
12) 415 Unsupported Media Type
13) 500 Internal Server Error

- Python

Python foi criado no final dos anos oitenta(1989) por Guido van Rossum no Centro de Matemática e Tecnológia da Informação (CWI, Centrum Wiskunde e Informatica), na Holanda. Python é uma linguagem de programação interpretada cuja filosofia enfatiza uma sintaxe favorecendo um código mais legível além de ser “free”. Atualmente, é gerenciada pela Python Software Foundation, possui uma licença de código aberto chamada Python Software Foundation License que é compatível com a GNU General Public License a partir da versão 2.1.1 e incompatível em certas versões anteriores. É uma linguagem de programação multi-paradigma pois suporta orientação de objeto, programação imperativa e em menor escala programação funcional. É uma linguagem interpretada têm seus códigos fontes transformados em uma linguagem intermediária (específica de cada linguagem) que será interpretada pela máquina virtual da linguagem quando o programa for executado. Usa tipagem dinâmica e forte isso significa que o próprio interpretador do Python infere o tipo dos dados que uma variável recebe sem a necessidade que o usuário da linguagem diga de que tipo determinada variável. Linguagem que possui uma gama imensa de bibliotecas para efetuar testes de API dentre outras.

- BDD

BDD é uma técnica de desenvolvimento de software ágil que surge através de uma crítica de Dan North ao Test Driven Development(Desenvolvimento orientado a testes) onde ele visava otimizar o conceito de ‘verificação e validação’ já aplicado e tornar mais eficiente a construção de cenários a serem testados e/ou desenvolvidos. O desenvolvimento orientado ao comportamento (ou BDD) é uma técnica ágil de desenvolvimento de software que incentiva a colaboração entre desenvolvedores, controle de qualidade e participantes não técnicos ou empresariais em um projeto de software. BDD como a união de várias práticas consideradas ágeis e úteis no desenvolvimento de software, cuja ênfase está nas funcionalidades de alto valor e na redução dos custos de mudança por meio da identificação do que de fato está sendo testado/desenvolvido. Além disso, pode-se dizer também que BDD é a evolução do TDD, isto porque, os testes ainda orientam o desenvolvimento primeiro se escreve o teste e depois o código.

Desenvolvedores que se beneficiam destas técnicas escrevem os testes em sua língua nativa em combinação com a linguagem ubíqua (Ubiquitous Language - é uma linguagem estruturada em torno do modelo de domínio e usada por todos os membros da equipe para conectar todas as suas atividades com o software). Isso permite que eles foquem em por que o código deve ser criado ao invés de detalhes técnicos e ainda possibilita uma comunicação eficiente entre as equipes. A linguagem de negócio usada em BDD é extraída das estórias ou especificações fornecidas pelo cliente durante o levantamento dos requisitos. Alguns frameworks utilizam o conteúdo das estórias escritas em um arquivo de texto como cenários para os testes. Dan North apresentou este conceito em 2003 ele sugeriu um padrão para escrita destes arquivos é apenas um modelo, ou seja, não é obrigatório. Entretanto Dan North denota que é extremamente importante a equipe seguir um padrão para facilitar a comunicação entre os envolvidos no projeto, esta técnica foi amplamente usada na implementação do

framework para o design de teste de segurança propostos facilitando assim seu entendimento.

Figura 1. Substituir por imagem real BDD

```
Título (uma linha descrevendo a história)

Narrativa:
Como [o papel]
Eu quero [recurso]
Assim que [benefício]

Critérios de Aceitação: (apresentado como Cenários)

Cenário 1: Título
Dado contexto []
  E [um pouco mais de contexto] ...
Quando [eventos]
Então [resultado]
  E [outro resultado ...]

Cenário 2: ...
```

Vantagens BDD

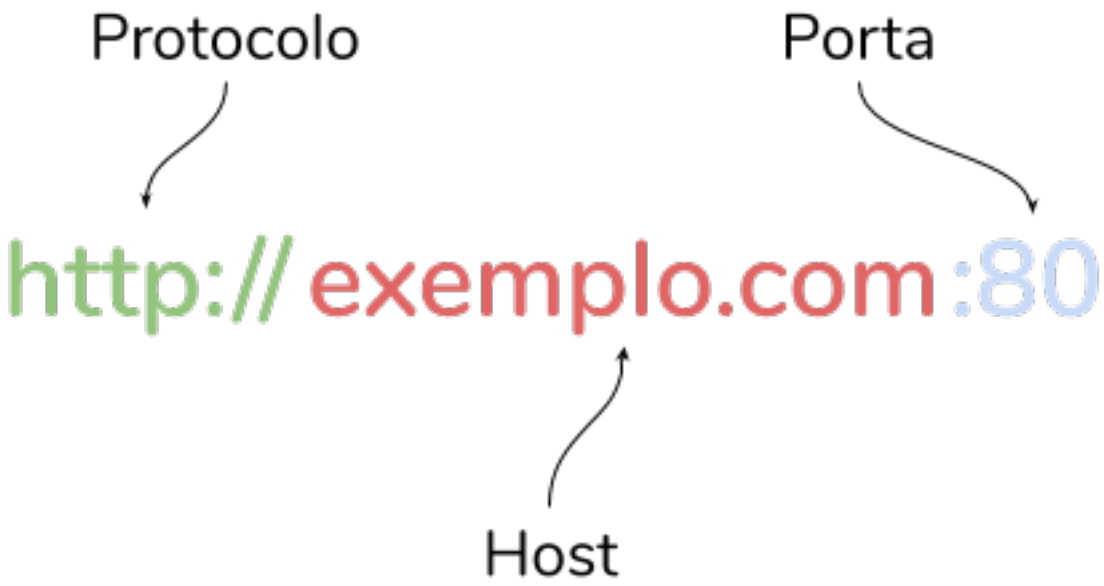
- Comunicação entre equipes
- Compartilhamento de conhecimento
- Documentação dinâmica
- Visão do todo

- Application Security Risks

1) Cross Domain Attack

Ataque que utiliza configuração incorreta permitindo que o invasor envie uma solicitação de domínio cruzado e possa ler dados arbitrários de outros usuários. Esta configuração de recurso chamado CORS(Cross-Origin Resource Sharing) mecanismo que gerencia o compartilhamento de recursos entre diferentes origens baseado em regra ou política same-origin que só autoriza a troca de recursos entre as mesmas origens (domínios). Caso precise troca entre diferentes origens é necessário configurar as chamadas para que contenham os cabeçalhos CORS corretos. A utilização do CORS é importante porque além de poder restringir quem pode acessar os recursos de um servidor também pode especificar como esses recursos devem ser acessados.

Figura 2. Definição de Origem



Sendo necessario obter recurso da origem acima a politica de same-origin não seria mais valida devido a requisição não ser da mesma origem, seria necesssario a politica de cross-origin.

O mecanismo de cross-origin funciona informando nos cabeçalhos HTTP nos responses de um determinado servidor para que o mesmo efetue um compartilhamento de recursos seguro entre origins diferentes liberando acesso assim métodos (GET, POST, PUT, DELETE, etc.) que possam ser utilizados.

Quando o site faz uma requisição que pode causar efeitos colaterais no servidor o navegador realiza uma “pré-requisição” ou preflight request para que o servidor retorne que é possível realizar a requisição que é pedida originalmente e só então ela é enviada ao servidor. Existem algumas condições que necessitam para se fazer as pré-chamadas. Portanto existem dois tipos de requisições:

- Requisições simples: Requisição que em teoria não causa efeitos colaterais no servidor

i) Metodos permitidos: GET, HEAD, POST.
ii) Cabeçalhos pré-configurados e permitidos: Accept, Accept-Language, Content-Language, Content-Type, DPR, Downlink, Save-Data, Viewport-Width, Width.
iii) Valores permitidos Content-Type: application/x-www-form-urlencoded, multipart/form-data, text/plain.

- Requisições preflight: Requisição que não atende as condições da requisição simples sendo necessário uma pré-requisição com OPTIONS que contém informações sobre a requisição original neste caso necessário

configurar os cabeçalhos para a chamada OPTIONS porque ela que realiza a autorização da requisição original, caso o OPTIONS seja rejeitado nada será processado pelo servidor caso aprovado a request original será enviada automaticamente pelo browser.

As requisiçoes apresentam uma lista de headers dentre os quais podemos citar os exemplos abaixo:

- Access-Control-Allow-Headers: Content-Type
- Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS
- Access-Control-Allow-Origin: *
- Access-Control-Max-Age: 3600

Dentre os headers mais importantes o Access-Control-Allow-Origin é preenchido com a lista de dominios cujo server está apto a receber/processar requests.

Resumindo em um primeiro momento acreditava-se que seria melhor se as requests só podiam ser feitas dentro de um mesmo domínio porém, em algum momento percebeu-se que seria interessante permitir cross-domain requests mas simplesmente quebrar a Same-Origin-Policy seria perigoso demais com isso surgiu o CORS, que específica algumas coisas para permitir que as requests cross-domains pudessem ser realizadas só que um pouco mais seguras. O CORS não traz segurança para o servidor ele tenta proteger o lado do cliente de possiveis ataques.

2) Security Headers Missing

TO DO

3. Materiais e Métodos

TO DO

4. Resultados e Discussão

4.1. Resultados

TO DO

4.1.1. Dicas

- TO DO

4.2. Discussão

TO DO

5. Conclusão

Escreva aqui sua conclusão...