

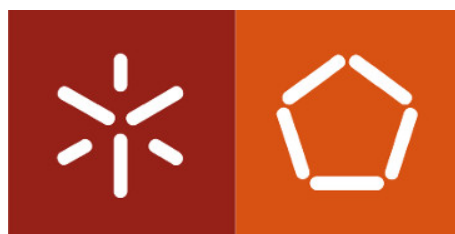
Engenharia de Segurança

24 de Junho de 2021

Grupo 7

a83899	André Morais
a84485	Tiago Magalhães

Projeto de desenvolvimento (PD)



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1	Introdução	4
2	(Pré)Fase de Formação	5
3	Fase de Requisitos	5
3.1	Requisitos Funcionais	5
3.2	Requisitos Não Funcionais	5
3.3	Requisitos de Segurança	6
4	Concepção/Desenho da resolução	7
4.1	Diagrama de <i>Use Cases</i>	7
4.2	Arquitetura e componentes	8
4.3	Descrição da arquitetura	8
4.4	Requisitos de <i>Design</i>	9
4.4.1	Tecnologias a utilizar	9
4.4.2	Definição de métricas e relatórios de conformidade	9
4.4.3	Definição de usos de <i>standards</i> criptográficos	9
4.5	Gestão dos <i>Third-party Components</i>	10
4.5.1	Dependências	10
4.5.2	<i>CVE's</i>	12
5	Fase de Codificação	15
5.1	<i>Injection Vulnerability</i>	15
5.2	<i>Quebra na autenticação</i>	16
5.3	<i>Cross-Site Scripting</i> (XSS)	16
5.4	<i>Cross Site Request Forgery</i> (CSRF)	17
5.5	<i>Controlo de acesso a funções</i>	18
5.6	<i>Buffer Overflow</i>	18
6	Fase de Verificação	19
6.1	Análise estática do código	19
7	Manual de Instalação	21
8	Conclusão	22

9	Anexos	23
9.1	Aula5 - 1.3	23
9.2	Aula5 - 2.3	23

Glossário

NIST Instituto tecnológico que promove padrões e tecnologia de forma a ampliar a segurança.

The OWASP Top 10 É um documento *standard* para desenvolvedores e segurança de aplicações *web*. Representa um consenso acerca dos riscos de segurança mais críticos para aplicações *web*.

Siglas

CRL Certificate Revocation List.

OSCP Online Certificate Status Protocol.

1 Introdução

Este relatório visa descrever o trabalho efetuado, bem como, os resultados obtidos do Projeto de Desenvolvimento da Unidade Curricular Engenharia de Software

Para a realização deste projeto foi necessário basear-nos no que foi aprendido ao longo do semestre e das fichas semanais, para deste modo, ser desenvolvida, da forma mais correta, uma plataforma para emissão de **certificados de teste**, **CRL** (*Certificate Revocation List*), **timestamps** e ainda disponibilize o serviço **OCSP** (*Online Certificate Status Protocol*). Foi também criada uma interface *web* para os utilizadores, que os permita autenticar e registar, emitir certificados nas hierarquias configuradas, aceder aos seus certificados e validar se os certificados ainda estão válidos.

A emissão de certificados, CRL e OCSP é baseado em **DogTag** e para a componente de emissão de *timestamps* utilizamos a **BouncyCastle Crypto API**.

Assim, serão apresentados neste relatório as técnicas desenvolvidas para obter *software* seguro, as ferramentas e indicadores de qualidade de *software* utilizados, bem como, os testes efetuados e, por fim, o modo de testar o código desenvolvido.

De salientar, que para a construção deste projeto guiarmo-nos pelo **Software Life Cycle Process**, o qual vai ser explicado detalhadamente neste relatório, e que, resumidamente, é um processo utilizado pela indústria de *software* para adquirir, fornecer, desenvolver, operar e manter *software* e tem como objetivo produzir um *software* de alta qualidade.

2 (Pré)Fase de Formação

Esta **primeira fase** ou **fase 0**, como lhe gostamos de chamar, é para fazer uma preparação dos membros da equipa de desenvolvimento de *software* para lidarem com os aspetos de segurança.

A nossa preparação consistiu nas aulas teóricas e na resolução das fichas semanais que o professor ia lançando, como por exemplo tentar corrigir erros de *Buffer Overflow* ou aprender a defender-nos de *SQL Injections* e de *Cross-site scripting*.

3 Fase de Requisitos

Nesta fase, o mais importante é identificar os requisitos funcionais e não funcionais da aplicação, assim como definir requisitos mínimos de segurança.

3.1 Requisitos Funcionais

1. Registo e autenticação de um utilizador;
2. Permitir a emissão de certificados nas hierarquias configuradas;
3. Utilizadores devem visualizar os seus certificados;
4. Validação de certificados(no CRL e OCSP);
5. Obter e validar *timestamps*.
6. *Compliance* com o RGPD (Regulamento Geral de Proteção de Dados).

3.2 Requisitos Não Funcionais

1. Aplicação de uso intuitivo;
2. Aplicação funcional durante todos os dias da semana;

3.3 Requisitos de Segurança

Uma vez que a aplicação desenvolvida é *web*, tivemos em conta as vulnerabilidades de segurança *web* presentes no *The OWASP Top 10* [4].

1. Certificados só podem ser acedidos pelo seu utilizador;
2. Mitigar vulnerabilidades de *Injection*;
3. Mitigar vulnerabilidades de quebra na autenticação;
4. Mitigar vulnerabilidades de *Cross-Site Scripting(XSS)*;
5. Mitigar exposição de dados sensíveis;
6. Mitigar *Cross Site Request Forgery* (CSRF).

4 Concepção/Desenho da resolução

No que diz respeito à fase de desenho, a segurança é uma preocupação central e é tomada em conta desde o início e que problemas com erros, falhas, vulnerabilidades e comportamentos maliciosos existem e acontecem.

4.1 Diagrama de *Use Cases*

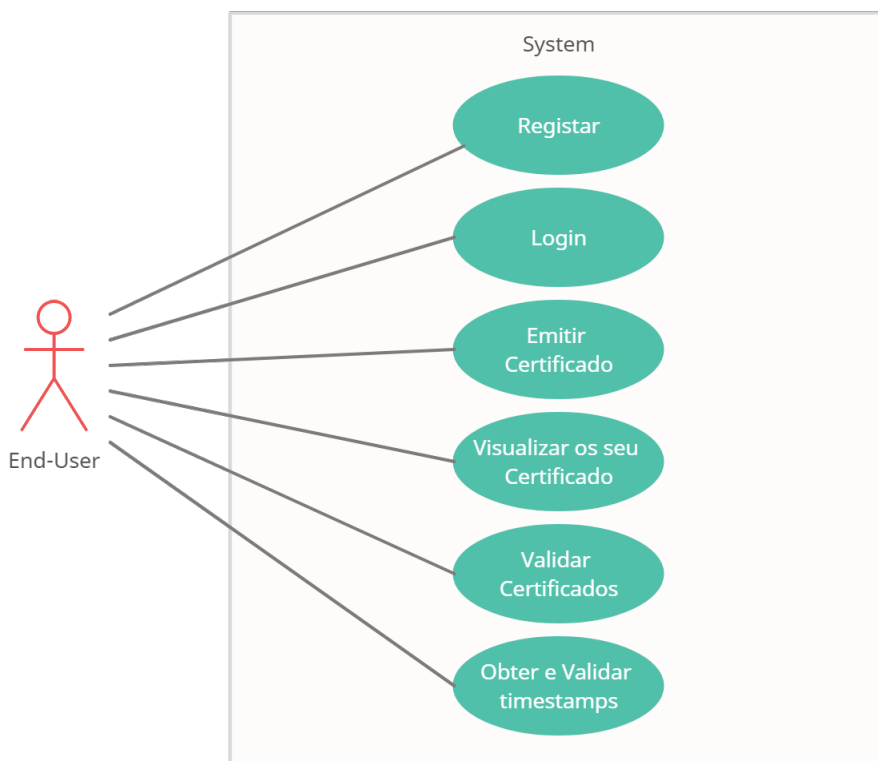


Figura 1: *Use Cases* da Aplicação

4.2 Arquitetura e componentes

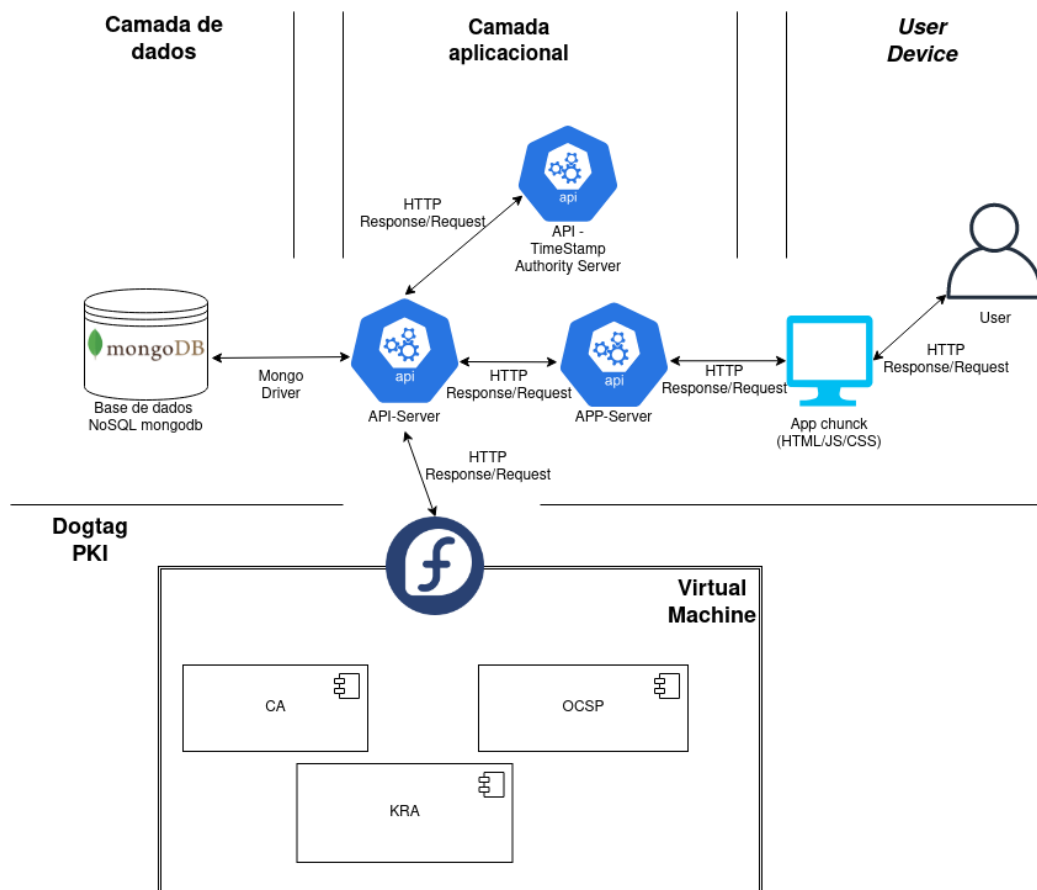


Figura 2: Arquitetura de alto nível do sistema

4.3 Descrição da arquitetura

Através do modelo do sistema podemos observar que existem 5 componentes principais sendo estas:

- **API-Server:** Servidor que assegura a comunicação entre o *App Server*, base de dados, componente Dogtag e *TimeStamp Authority Server* (TSA), através de *requests* do *App Server* o *API Server* devolve informação relativa aos casos de uso definidos anteriormente.

- **TSA:** Servidor *web* que tem como objetivo obter e validar *timestamps*.
- **Dogtag:** *Third-Party Component* que permitiu usar os seus subsistemas para emitir certificados e sua validação(OCSP e CRL).
- **App Server:** Servidor que fornece páginas dinâmicas ao cliente e que consome dados do *API Server*.
- **MongoDB:** Base dados que guarda o *email*, *hash*, *nome* e certificados associados.

4.4 Requisitos de *Design*

4.4.1 Tecnologias a utilizar

A componente **TSA** será realizada utilizando *Java* uma vez que a biblioteca auxiliar a utilizar é compatível com *Java*. O *API-Server* e *APP-Server*, serão desenvolvidos em *JavaScript* utilizando a *framework* *express*, uma vez que os elementos do grupo já estão familiarizados com esta ferramenta. A utilização da uma base de dados *NoSQL*, deve-se ao facto de não existirem estruturas de dados relacionadas, pretendendo-se apenas guardar coleções com informação dos utilizadores.

4.4.2 Definição de métricas e relatórios de conformidade

O **TSA** deve seguir o RFC 3161 [2], que é um documento técnico desenvolvido e mantido pelo IETF (*Internet Engineering Task Force*), instituição que especifica os padrões que serão implementados e utilizados em toda a internet.

Os sistemas que usem a ferramenta *express* devem seguir as práticas recomendadas pela documentação da mesma [6].

4.4.3 Definição de usos de *standards* criptográficos

Para a componente **TSA** estamos a usar a biblioteca *BouncyCastle* [3], que se encontra na lista produtos credenciados pelo *NIST* [4]. Para o armazenamento de *passwords* na base de dados é utilizado o método *bcrypt*, que se encontra entre as melhores práticas para *password hashing* [5], para isso é utilizada a biblioteca *bcrypt* do node.

4.5 Gestão dos *Third-party Components*

Uma vez que foram usadas *API's* externas necessárias para o funcionamento de aplicação, entra-se na parte da gestão de *TPC's*, onde se estabelece um conjunto de aplicações que vão auxiliar a definir a segurança de todo o projeto e dessa forma descompilar também a sua utilização por futuros utilizadores/*developers*.

4.5.1 Dependências

Identificar as *Third-party Components* usadas num produto pode ser bastante complicado. Uma simples *TPC* pode usar outras sub-componentes de outro *TPC* e assim em diante. No nosso caso, um exemplo de uma *TPC* com multi-níveis de dependências é o **NPM** usada no *framework* **Express**[9].

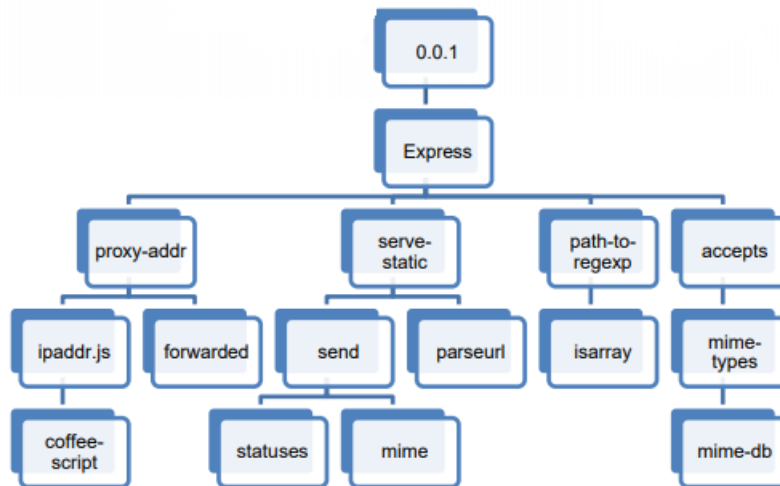


Figura 3: Uma pequena parte do gráfico de dependências para o módulo *NPM* "0.0.1"

Como pudemos ver na Figura 3, se quisermos utilizar uma dependência, temos que confiar nas "ligações" que esta tem com outras.

Para mostrar as dependências usadas, construímos uma tabela para ser mais fácil a leitura e compreensão destas

Nome da TPC	Data da última atualização	Contactos Developers	Vulnerabilidades Recentes/Última Vulnerabilidade
Express	Janeiro de 2021	Sim	Sim/2021
DogTag API	Junho de 2021	Sim	Sim/2020
BouncyCastle Crypto API	Junho de 2021	Sim	Não/2019

Figura 4: Tabela das dependências usadas

4.5.2 CVE's

Nesta seção vamos apresentar o último *CVE*[10] público de cada dependência usada

4.5.2.1 Express

- *CVE*: CVE-2021-32819
- *CVE score*: 8.8 (*High*)
- *Data de Publicação*: 05/14/2021

🚩 CVE-2021-32819 Detail



Current Description

Squirrelly is a template engine implemented in JavaScript that works out of the box with ExpressJS. Squirrelly mixes pure template data with engine configuration options through the Express render API. By overwriting internal configuration options remote code execution may be triggered in downstream applications. There is currently no fix for these issues as of the publication of this CVE. The latest version of squirrelly is currently 8.0.8. For complete details refer to the referenced GHSL-2021-023.

[+View Analysis Description](#)

Severity CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

	NIST: NVD	Base Score: 8.8 HIGH	Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H
	CNA: GitHub, Inc.	Base Score: 8.0 HIGH	Vector: CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:C/C:H/I:H/A:N

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: It is possible that the NVD CVSS may not match that of the CNA. The most common reason for this is that publicly available information does not provide sufficient detail or that information simply was not available at the time the CVSS vector string was assigned.

Figura 5: CVE-2021-32819

4.5.2.2 DogTag PKI

- **CVE:** CVE-2020-15720
- **CVE score:** 6.8 (*Medium*)
- **Data de Publicação:** 07/14/2020

CVE-2020-15720 Detail


Current Description

In Dogtag PKI through 10.8.3, the `pki.client.PKIConnection` class did not enable python-requests certificate validation. Since the `verify` parameter was hard-coded in all request functions, it was not possible to override the setting. As a result, tools making use of this class, such as the `pki-server` command, may have been vulnerable to Person-in-the-Middle attacks in certain non-localhost use cases. This is fixed in 10.9.0-b1.

[+View Analysis Description](#)

Severity CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

 **NIST:** NVD **Base Score:** 6.8 MEDIUM **Vector:** CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:H/I:H/A:N

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

Figura 6: CVE-2020-15720

4.5.2.3 BouncyCastle Crypto API

- **CVE:** CVE-2019-0379
- **CVE score:** 5.3 (*Medium*)
- **Data de Publicação:** 10/08/2019

🔗 CVE-2019-0379 Detail

Current Description

SAP Process Integration, business-to-business add-on, versions 1.0, 2.0, does not perform authentication check properly when the default security provider is changed to BouncyCastle (BC), leading to Missing Authentication Check

[+View Analysis Description](#)

Severity CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

 **NIST:** NVD **Base Score:** 5.3 MEDIUM **Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

Figura 7: CVE-2019-0379

5 Fase de Codificação

Para seguir boas práticas de codificação nesta fase, para o caso de desenvolvimento *JavaScript* foi utilizada ferramenta **ESLint**, que permite formatação do código de acordo com as *guidelines* definidas pela linguagem, bem como permite criação de regras que consideremos necessárias, além de permitir encontrar erros sintáticos, o que é importante devido ao facto do *JavaScript* ser uma linguagem dinâmica e consequentemente ser difícil encontrar erros.

Foram seguidas regras de programação defensiva para reduzir vulnerabilidades, neste caso identificadas anteriormente nos requisitos de segurança.

5.1 *Injection Vulnerability*

Apesar desta vulnerabilidade estar muitas vezes associada a *SQL Injections*, no nosso caso como utilizamos **Mongodb** podem ocorrer *NoSQL injections* [11], através do uso de expressões como **\$where**, no entanto como utilizamos o *driver* **mongoose**, este antes de inserir na base dados vai formatar o *input* de acordo com um *schema* definido, exemplo:

```
var userSchema = new mongoose.Schema({
  nome: String,
  email: String,
  password: String,
  certs: [String],
});
```

Caso o servidor recebesse um nome com valor: **\$where** ele iria ser formatado para *string*.

5.2 Quebra na autenticação

Para seguimento do estado de uma sessão *web*, são utilizados *jsonweb-tokens*, que respeitam as seguintes propriedades: unívocos, imprevisíveis e curta expiração

```
1 // Midl्लeware para intercetar tokens
2 app.use(function (req, res, next) {
3   // Ignorar necessidade de token destas rotas
4   if (req.originalUrl !== "/login" && req.originalUrl !== "/"
5       registrar"){
6     // Autorizacao
7     const token = req.query.token;
8     if (!token)
9       return res
10        .status(401)
11        .json({ auth: false, message: "No token provided."});
12
13    jwt.verify(token, process.env.SECRET, function (err,
14              decoded) {
15      if (err)
16        return res
17          .status(500)
18          .json({ auth: false, message: "Failed" });
19
20      // se tudo estiver ok, salva no request para uso
21      // posterior
22      req.userId = decoded.id;
23      next();
24    });
25  } else next();
26 });
```

5.3 Cross-Site Scripting(XSS)

Para prevenir este tipo de ataques é utilizada a biblioteca: **Express XSS Sanitizer**, que adiciona um *midl्लeware* que verifica *user input*(req.body, req.query, req.headers and req.params) de modo a prevenir este ataque. Além desta mitigação foi adicionado *Content Security Policy* para também prevenir, uma vez que apresenta uma *whitelist* de origem de recursos e *scripts* a serem usados pela aplicação.

```
1 // Midl्लeware p
2 app.use(function (req, res, next) {
```

```

3 app.use(
4   helmet.contentSecurityPolicy({
5     useDefaults: true,
6     directives: {
7       defaultSrc: ["'self'", "localhost:7779"],
8       scriptSrc: [
9         "'self'",
10        "code.jquery.com",
11        "cdn.datatables.net",
12      ],
13       styleSrc: ["'self'", "use.fontawesome.com", "cdn.
14         datatables.net"],
15       imgSrc: ["'self'", "cdn.datatables.net"],
16       fontSrc: ["'self'", "use.fontawesome.com"],
17       upgradeInsecureRequests: [],
18     },
19     reportOnly: false,
20   });

```

5.4 *Cross Site Request Forgery*(CSRF)

Para mitigar este ataque foi criado nas páginas de formulários um CSRF *token*, que leva a que seja difícil para um atacante reconstruir um *HTTP Request*, uma vez que este *token* é difícil de prever e por isso o pedido irá falhar.

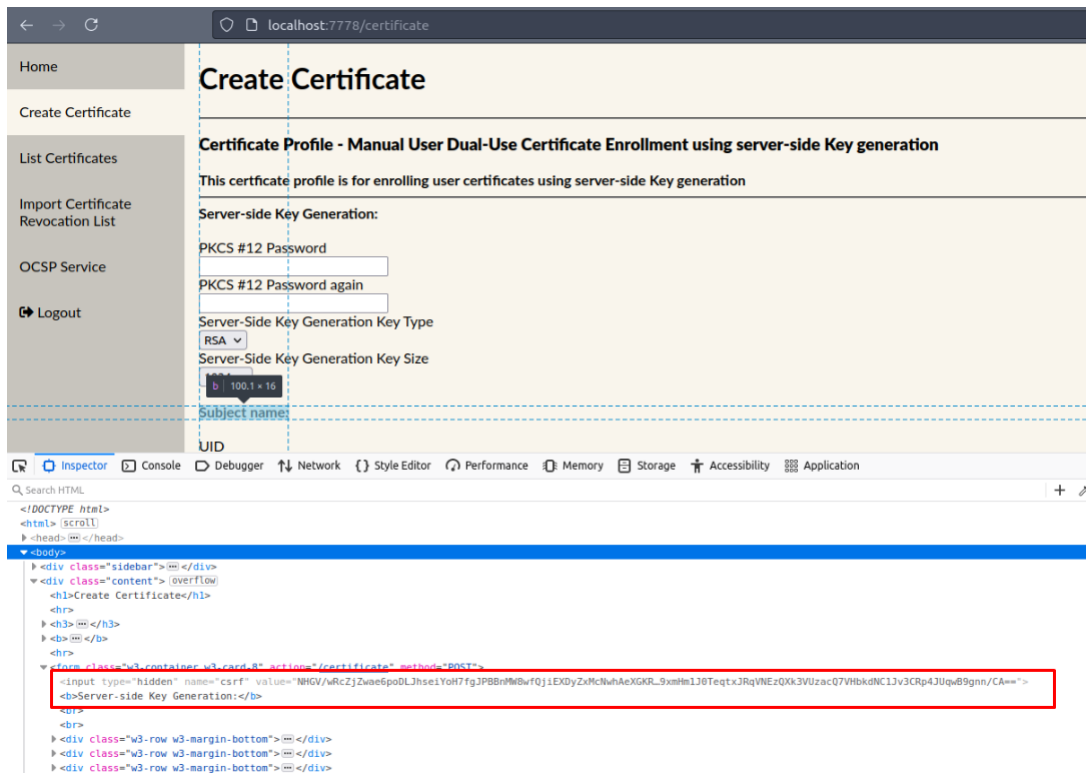


Figura 8: CSRF *Token* escondido

5.5 *Controlo de acesso a funções*

Só utilizadores registados podem aceder às rotas definidas, bem como só o utilizador com o certificado é que pode aceder a ele próprio.

5.6 *Buffer Overflow*

Muitas linguagens de programação estão propensas a ataques de *buffer overflow*. Contudo, estes ataques variam dependendo da linguagem usada para escrever o programa. No nosso caso, a aplicação *web* é feita em *JavaScript* que geralmente não é suscetível a *buffer overflows*.

Para a implementação dos *timestamps* foi usado *Java*. O *Java* foi desenhado para evitar estes ataques verificando os limites de um *buffer* (como um *array*) prevenindo qualquer acesso além desses limites estabelecidos

6 Fase de Verificação

6.1 Análise estática do código

Utilizamos o **SonarQube** que permite obter uma grande variabilidade de métricas, possibilita a detecção *bugs* e *code smells*, permitindo assim, melhorar a qualidade de código e consequentemente reduzir riscos de vulnerabilidade.

Executamos o *SonarQube* duas vezes, uma para a interface e outra para a *API*. Na verificação da interface aparecem 13 *bugs* e 803 *code smells*, mas é um "falso alarme", porque ele queixa-se de *tags* com o mesmo prefixo, o que é normal.

Na parte da *API* como podemos ver, apenas tem um *code smell* devido a uma função com demasiados argumentos, função esta que é para criar um objeto com os campos do certificado.

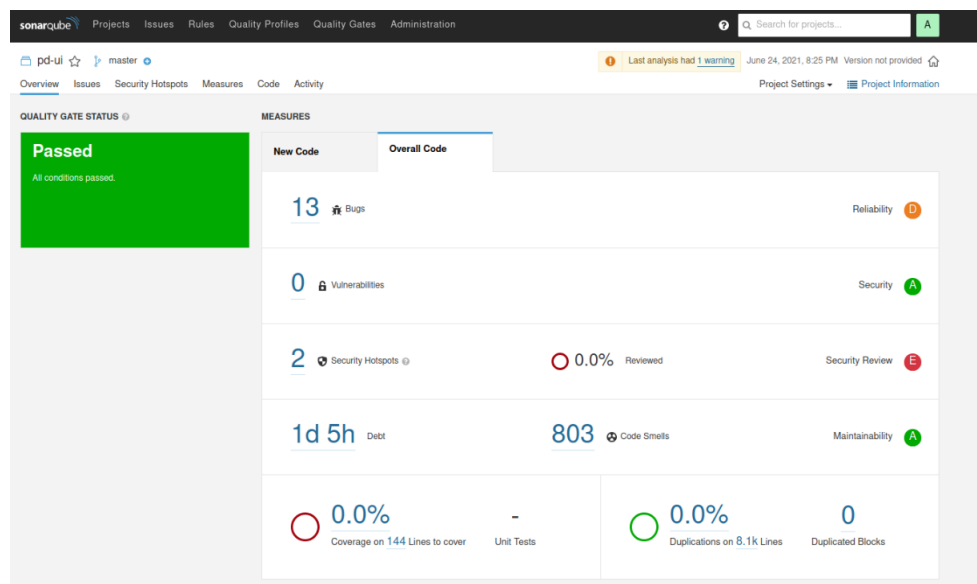


Figura 9: SonarQube Interface

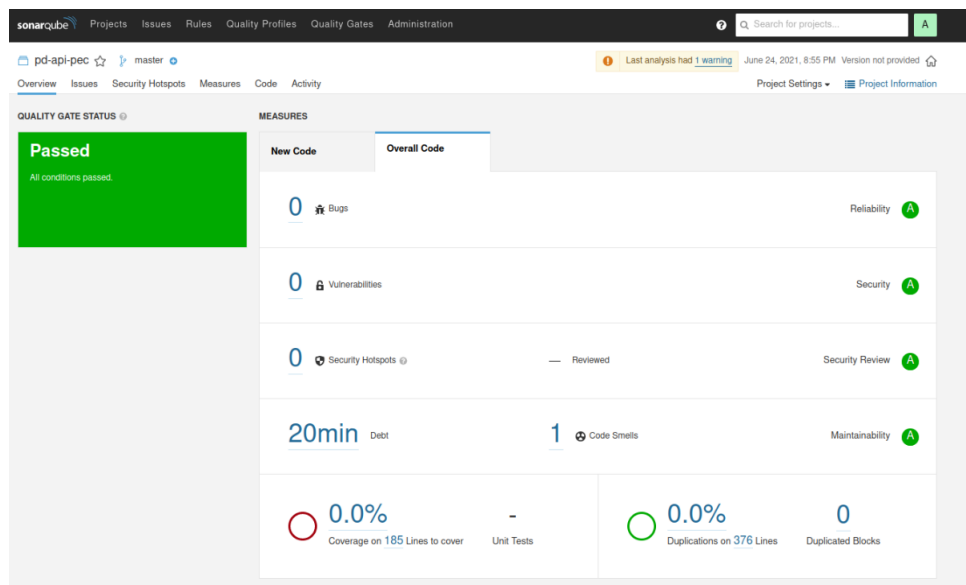


Figura 10: SonarQube API

7 Manual de Instalação

- Instalação

- Instalar a API do *DogTag* numa Máquina Virtual
- Instalar o MongoDB
- Clone/download repositório
- Para cada componente(servidor), instalar as dependências:

```
cd src/<diretoria_servidor>
```

```
npm i --save
```

- Uso

- para cada servidor fazer o seguinte:

```
cd src/<diretoria_servidor>
```

- completar variáveis de ambiente (.env api-server)

```
// Segredo do jsonwebtoken
SECRET= ***
// Nome do administrador da CA
USERNAMECA= ***
// Password do administrador da CA
PASSWORD= ***
// Endereço IP da máquina com a componente DOGTAG
PKI_HOST= ***
// Username do host da máquina com a componente DOGTAG
PKI_USERNAME= ***
```

– Ligar o servidor:

```
npm start
```

8 Conclusão

Durante a elaboração deste trabalho surgiram algumas dificuldades que, com o esforço e força de vontade, acabaram por ser ultrapassadas. Entre as quais destacam-se o desafio que foi a geração de um certificado, com a ajuda do *DogTag PKI* e a de criar *timestamps* a partir da *BouncyCastle Crypto API*.

Consideramos que objetivo principal deste trabalho prático não foi a programação, mas sim utilizar metodologias de desenvolvimento de software seguro e seguir *standards* de verificação de segurança de aplicações. Contudo, alguns dos objetivos não foram a tempo de ser realizados devido à escassez de tempo. A criação de *timestamps* e a emissão de certificados nas hierarquias configuradas, não foram programadas a tempo da entrega, o que nos deixa um pouco frustrados.

Por fim, consideramos que a consolidação de todos os conhecimentos lecionados nesta cadeira foram bem interiorizados e que sejam de enorme utilidade tendo uma perspectiva futura.

9 Anexos

9.1 Aula5 - 1.3

Visto que o nosso grupo é composto apenas por 2 elementos, não é possível englobar todos as funções e responsabilidades de segurança como no SDLC. Mesmo assim, as funções que vamos realizar são as mais vocacionadas com a programação e a arquitetura do projeto, como por exemplo:

- Desenvolvedor de *Software*
- Arquiteto de Sistema
- Administrador de Programas

9.2 Aula5 - 2.3

Excel anexado no github

Referências

- [1] Top 10 Web Application Security Risks. Acedido em Abril, em: <https://owasp.org/www-project-top-ten/>.
- [2] RFC 3161. Acedido em Maio, em: <https://datatracker.ietf.org/doc/html/rfc3161>.
- [3] Bouncy Castle. Acedido em Junho, em: <https://www.bouncycastle.org/>.
- [4] Produtos credenciados pelo NIST. Acedido em Junho, em: <https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/>.
- [5] Best practices for password hashing and storage. Acedido em Junho, em: <https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/>.
<https://tools.ietf.org/id/draft-whited-kitten-password-storage-00.html>
- [6] Production Best Practices: Security. Acedido em Junho, em: <https://expressjs.com/en/advanced/best-practice-security.html>.
- [7] DogTag PKI. Acedido em Maio, em: https://www.dogtagpki.org/wiki/PKI_Main_Page.
- [8] Github Dogtag . Acedido em Maio, em: <https://github.com/dogtagpki>.
- [9] Managing Security Risks Inherent in the Use of Thirdparty Components. Acedido em Junho, em: https://safecode.org/wp-content/uploads/2017/05/SAFECode_TPC_Whitepaper.pdf.
- [10] CVE Mitre. Acedido em Junho, em: <https://cve.mitre.org/>.
- [11] Mongodb injection. Acedido em Junho, em: <https://docs.mongodb.com/manual/faq/fundamentals/#how-does-mongodb-address-sql-or-query-injection>.