FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Escudo para Aplicações Web contra Injeção de Conteúdo através de Content Security Policy

Vítor Emanuel Freitas de Oliveira Magano

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Professor Doutor Daniel Castro Silva

Co-orientador: José Magalhães

20 de Julho de 2015

Escudo para Aplicações Web contra Injeção de Conteúdo através de Content Security Policy

Vítor Emanuel Freitas de Oliveira Magano

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Professor Doutor Rosaldo Rossetti
Arguente: Professor Doutor Pedro Henriques Abreu
Vogal: Professor Doutor Daniel Castro Silva
20 de Julho de 2015

Resumo

Este tema de dissertação centrado na segurança foi proposto pela empresa *JScrambler* que é detentora do produto com o mesmo nome que, por sua vez oferece uma solução de proteção de *JavaScript* completa. Estando a empresa a par dos progressos nesta área, o mecanismo de segurança *Content Security Policy* despertou interesse pelas suas potencialidades. De forma geral esta ferramenta limita a execução de conteúdo num *website* ao declarado através do *CSP* sendo que a tentativa de injeção de conteúdo que exceda o que foi permitido será bloqueado e reportado ao servidor.

Adjacente a esta ferramenta está a grande carga de trabalho necessária à configuração e manutenção da mesma que requer a devida atenção para evitar problemas de gravidade maior, como *downtime*, que podem ser causados por uma má configuração das políticas ou por má manutenção. Posto isto, tendo em conta este problema foi proposta a criação de uma solução que aliviasse a carga de trabalho através da geração de políticas de *CSP* por análise de código.

Começou-se por uma análise da área que cobriu vários métodos de proteção de *JavaScript* em que o tema se insere a fim de compreender o seu estado atual e avaliar a viabilidade da proposta. Nesta análise foi feita uma comparação entre várias ferramentas existentes nas áreas de proteção e análise de código bem como ferramentas de geração de *CSP* e com a solução desenvolvida, solução esta que foi estruturada e planeado o seu desenvolvimento atendendo a toda a pesquisa efetuada. A grande vantagem desta face às restantes ferramentas passa pela integração total com o projeto que provém da análise do código que vai permitir a geração de políticas perfeitamente adequadas ao mesmo. O planeamento do projeto teve em conta a adoção da metodologia ágil de desenvolvimento *SCRUM* com iterações semanais que permitiram o desenvolvimento da solução de geração aqui apresentada.

A geração de *CSP* parte da análise estática de código que é feita a um *website* seja este carregado para o servidor ou indicado o *URL* para *crawl* da aplicação. São obtidos os recursos detetados na análise e após especificação de um formato final é gerada uma política de *CSP* específica para a linguagem ou *webserver* especificados.

Por fim foi feita a validação através de testes de performance e eficácia para a obtenção de resultados relativos à eficácia da aplicação desenvolvida tendo-se constatado um avanço na automatização da geração de *CSP* e assim o cumprimento dos objetivos propostos inicialmente.

Esta solução pode ser vista como um passo em prol do aumento da segurança voltado para a proteção contra ataques *XSS*.

Abstract

This MSc thesis topic focused on security was proposed by the company *JScrambler* which holds the product with the same name and in turn provides a complete *JavaScript* protection solution. This company, being aware of the progress in this area, found the security mechanism *Content Security Policy* an interesting tool that worth look into. Broadly speaking this tool limits the content to run in a website to the declared through *CSP* being that the attempt to content injection exceeding what was allowed will be blocked and reported to the server.

Following this tool is the great workload required for its configuration and maintenance which requires proper attention to avoid more serious problems, such as *downtime*, which can be caused by a misconfiguration of policies or poor maintenance. That said, looking forward to solve this problem the creation of a solution was proposed to ease the workload by generating *CSP* policies through code analysis.

On a first stage an analysis of the area that covered several of *JavaScript* protection methods in which the subject is inserted was made in order to understand their current status and evaluate the feasibility of the proposal. In this analysis a comparison between several existing tools in the areas of protection and code analysis and between tools for generation of *CSP* and the solution to be developed was conducted.

The solution was proposed and planned their development given all the research performed. The great advantage of it over the other tools is the full integration with the project that comes from the analysis of the code that will enable the generation of suitable and appropriate *CSP* policies. The project planning has taken into account the adoption of agile development methodology *SCRUM* with weekly iterations that allowed the project development presented in this report concerning *CSP* policy generation.

This generation uses static code analysis that is performed on a website that can be uploaded or crawled through a given URL. The resources are gathered on the analysis and after a specification of a final format the pretended policy is generated for the specified language or webserver.

At last the application was validated through performance and efficiency tests. The initially proposed goals were achieved and the automatization of *CSP* generation surpassed the existing solutions.

The solution reflect a great advance in the web security concerning Cross-site scripting attacks.

Agradecimentos

Aos meus pais por todas as oportunidades e incentivos, aos meus irmãos pelos momentos de descompressão, aos dois orientadores que me acompanharam e aconselharam e a todos os que me suportaram de qualquer forma.

Vítor Magano

""Adapt what is useful, reject what is useless, and add what is specifically your own.""

Bruce Lee

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

API	*Application Programming Interface*
CSS	*Cascading Style Sheets*
CSP	*Content Security Policy*
DOM	*Document Object Model*
DNS	*Domain Name System*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
JSONP	*JSON with padding*
HTML	*HyperText Markup Language*
URL	*Uniform Resource Locator*
XSS	*Cross-site Scripting*
W3C	*World Wide Web Consortium*
WWW	*World Wide Web*

Capítulo 1

Introdução

Com o surgimento da internet e a possibilidade de acesso a uma grande quantidade de informação surgiu também a preocupação pela segurança desta mesma informação que nem sempre deve estar acessível a qualquer utilizador. Desta forma, vários métodos e técnicas têm sido desenvolvidas procurando sempre acompanhar a evolução tecnológica. O *JavaScript*, sendo uma poderosa linguagem que é interpretada pelos *browsers* do lado do cliente tem o seu código fonte exposto a toda a gente e consequentemente tem sido alvo de alguma investigação com o intuito de melhorar ou providenciar alguma segurança ao mesmo.

1.1 Contexto/Enquadramento

Com o principal objetivo de inovar na área da segurança informática em 2007 é criada a empresa *AuditMark* que se focou no desenvolvimento de um produto que procurava oferecer um nível de segurança superior ao existente. Assim, em 2010, é lançado o *JScrambler*. Este produto acabou por ser a principal imagem da empresa e em 2014 para reforçar essa ideia o nome do mesmo foi adotado como nome da empresa. O *JScrambler* é uma solução de proteção de *JavaScript* que oferece um nível de ofuscação que vai além da tradicional minificação e ainda otimização de código. Esta ferramenta tem acompanhado as principais evoluções do mundo tecnológico tendo recentemente estendido a sua compatibilidade para *Node.js*. [1, 2] Importante referir a ferramenta que está na base deste tema de dissertação, o *Content Security Policy (CSP)*, é um mecanismo de segurança criado pela *Mozilla Corporation* em 2010 e surge com a intenção de mitigar os ataques provenientes de injeção de conteúdo e aqui é importante dar destaque ao *Cross-site Scripting (XSS)* que é um tipo de vulnerabilidade que permite através da injeção de *scripts* escapar aos controlos de acesso. Esta ferramenta prima pelo bloqueio da execução de conteúdo que vá além do especificado através de fontes seguras. A restrição é aplicada através do bloqueio de execução do *script* e envio de uma mensagem de erro para o servidor. [3]

1.2 Motivação e Objetivos

Este tema de dissertação focando-se na segurança online pode trazer um grande avanço neste campo na contribuição para acabar com um certo tipo de ataques e aumentando consideravelmente a segurança na navegação online. Partindo de um conjunto de problemas identificados à partida que vão desde a grande recorrência de ataques de *XSS* à baixa popularidade e elevada complexidade do mecanismo de segurança *CSP*. Seguindo tal motivação foram traçados um conjunto de objetivos a alcançar durante todo o processo de investigação e desenvolvimento.

- **A**nalisar o que foi feito na área da segurança ao nível de *JavaScript*;
- **I**nferir a viabilidade e se possível desenvolver uma solução que permita automatizar o processo de integração e manutenção do *Content Security Policy* nas aplicações web a fim de reduzir a carga de trabalho associada à utilização deste;
- **I**ntegrar a solução com a ferramenta *JScrambler* incrementando a sua funcionalidade na proteção de dados.

1.3 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais 5 capítulos.

- No capítulo 2, é contextualizado o problema e são analisadas ferramentas relevantes para este tema focando-se em Mecanismos de segurança que antecederam o *CSP* e outros importantes para o desenvolvimento da solução pela abordagem tomada.
- No capítulo 3, é analisado o mecanismo de segurança que está na base da solução desenvolvida apresentando-se também soluções de geração já criadas para a geração de *CSP*. Por fim é ainda apresentada uma visão geral da solução e é feito um comparativo com as já existentes.
- O capítulo 4 representa uma análise em detalhe da solução e de todo o processo de desenvolvimento da mesma.
- No capítulo 5 é apresentado o processo de validação e análise dos resultados obtidos.
- No capítulo 6, é apresentada a satisfação dos objetivos definidos inicialmente e nas várias fases de desenvolvimento e também é apresentada uma secção de trabalho futuro em prol de colmatar as limitações da solução atual. São ainda apresentadas as conclusões para a solução desenvolvida.

Capítulo 2

Contextualização do Problema e Soluções Existentes

Para melhor compreender e analisar o estado atual de métodos utilizados para proteção do *JavasScript* foi feita uma análise da evolução da segurança especificamente referente a esta linguagem e foram assim desenvolvidas três principais secções, uma para introdução aos conceitos a serem abordados e duas para abordar o conjunto de ferramentas relevantes para este tema de dissertação para a automatização do processo de automatização de políticas. Estas três secções estarão ordenadas por ordem de relevância para este tema da seguinte forma, ***Web Browsing* e *JavaScript***, ***Sandboxing* de *JavaScript*** e **Análise Estática de Código e *JScrambler***.

2.1 *Web Browsing* e *JavaScript*

Ao longo deste capítulo vão ser abordados vários conceitos relacionados com a linguagem de programação *JavaScript*. Esta é bastante poderosa e atualmente utilizada em praticamente todas as aplicações *Web*. É uma linguagem imperativa que executa do lado do cliente o que significa que é executada pelo *browser* de cada um dos utilizadores, ou seja, quando uma página é carregada o código de *JavaScript* presente na aplicação é descarregado e armazenado localmente de forma temporária para ser executado pelo *browser*. Isto faz com que o tempo de execução após o descarregamento esteja apenas limitado pelo sistema do utilizador que geralmente é bastante rápido.

O facto de ser executado do lado do cliente traz no entanto um conjunto de problemas associados à segurança. Estando o código acessível, é possível que este seja consultado por qualquer utilizador e reutilizado/roubado sem qualquer tipo de autorização. Outro problema comum é que estando o código exposto é mais fácil para um atacante[1] procurar por vulnerabilidades, estando

[1] Utilizador que explora falhas num sistema e tira proveito destas.

assim mais exposto a ataques. Quando se fala de ataques, existe um tipo em particular que está relacionado com o *JavaScript*, o *Cross-Site Scripting (XSS)*. Este tipo de ataques consiste na injeção de pedaços de código *JavaScript* na aplicação e, posteriormente, extração de dados a utilizadores que sejam vítimas deste tipo de ataques como por exemplo dados de sessão desse *website*. Atualmente são reconhecidos três tipos de ataque provenientes do *XSS* que têm dimensões de magnitudes diferentes.

Stored XSS Attacks, também referidos como *Persistent*, consistem no tipo de ataques em que o *script* foi diretamente injetado no servidor da aplicação e portanto todos os utilizadores que acedam ao *website* estão sujeitos ao ataque pelo que as consequências do mesmo são à partida elevadas.

Reflected XSS Attacks, também referidos como *Non-Persistent*, este tipo de ataques é bastante mais comum. O *script* malicioso é injetado já fora do servidor e pode ter um alvo pequeno, uma única pessoa por exemplo, ou bem maior quando por exemplo se acede ao *website* através de um motor de busca. Daí a preocupação de os sistemas bancários por exemplo em pedir que os clientes tenham o cuidado de aceder às suas aplicações *Web* através do link e nunca através dos motores de busca que podem conter conteúdo malicioso.

DOM Based XSS, este é um tipo de ataque resultante da alteração do *DOM* [2] da vítima de tal forma que a aplicação execute de uma maneira inesperada. [4]

2.2 *Sandboxing* de *JavaScript*

Quando se incluem *widgets* numa aplicação *Web* pode-se estar a carregar conteúdo malicioso para o *website* e como tal estar a comprometer todo o conteúdo ou até os utilizadores da aplicação. Numa tentativa de minimizar os riscos de incluir conteúdo de fontes desconhecidas surge então o *sandboxing* de *JavaScript* que de certa forma por análise do código vai avaliar a sua qualidade e permitir ou negar a sua execução.

2.2.1 Definição

O *sandboxing* de *JavaScript* é portanto a denominação para um tipo de ferramentas/soluções que executem código desta linguagem num ambiente protegido de forma a avaliar a integridade do código sem comprometer de qualquer forma dados ou outro código da aplicação em questão.

2.2.2 Objetivo

O *sandboxing* de *JavaScript* tem então como objetivo restringir ou atenuar o acesso à restante informação presente no site através da criação de *sandboxes* (*frames* de conteúdo onde serão mostrados os *widgets*). Para tal existe um conjunto de ferramentas que fazem verificações estáticas do código e que fornecem os *wrappers* com acesso limitado aos restantes métodos da aplicação. [5]

[2] Convenção para a representação e interação com objetos *HTML*

2.2.3 Análise de Soluções

Sendo este um tema que tem preocupado grandes empresas ligadas ao mundo da informática, elas mesmas procuraram desenvolver soluções próprias para o *sandboxing* de *JavaScript*.

Análise Estática

google-caja [6], ***ADsafe*** [5], ***BrowserShield*** [7] ferramentas que partem de uma análise estática de código para determinar a qualidade do conteúdo de terceiros e fornecem ainda um ambiente protegido com acesso limitado aos recursos da aplicação. Todas estas ferramentas possuem um compilador próprio que permitem a execução do código de forma protegida e oferecem ao utilizador das mesmas uma API que estabelecerá a ligação entre os *widgets* de forma segura e mediante as necessidades. Não confundir esta secção com a análise que será abordada na próxima secção, esta análise refere-se a código de *widgets* e não de código próprio.

Iframes [8]

Elemento HTML que oferece um meio "protegido" para a execução de conteúdos de terceiros com permissões e acessos controlados sem afetar o resto do conteúdo.

JavaScript in JavaScript (js.js) [9]

Solução distinta tratando-se de um interpretador de *JavaScript* sendo superior às atuais ferramentas de análise estática de código e permite de forma igual limitar os acessos a conteúdos, oferece um suporte completo de *JavaScript*, é compatível com todos os *browsers* e permite prever problemas no código como usos excessivos de memória, redirecionamentos para outras páginas e alterações do *DOM*.

2.2.4 Aplicações Conhecidas

O ***google-caja*** foi utilizado pela empresa *Yahoo* nos serviços *My Yahoo!* e *Yahoo Mail*. Este serviço era essencial para quem quisesse desenvolver aplicações e usar *JavaScript* de forma segura. Atualmente é ainda utilizada no *Google Sites* tendo sido descontinuada dos serviços *Yahoo*. [10]

O *Facebook* também desenvolveu uma ferramenta própria de *sandboxing*, *FBJS*, sobre a qual o próprio *website* executava todos os elementos de terceiros. Também se aplicava para aplicações desenvolvidas por terceiros para a rede social. Atualmente esta ferramenta também foi substituída por API's e o uso de *iframes*.

Já para o *Javascript in JavaScript* não foi encontrado nenhum cenário de utilização no mundo real no entanto esta é uma solução passível de o ser.

2.2.5 Conclusão

Os métodos de *sandboxing* ainda estão a ser explorados e estão de certa forma a ser combinados com novas ferramentas como o *Content Security Policy* para em conjunto acabarem com

os ataques de *XSS* que ainda se encontram entre os ataques mais comuns da *Web* por falta de preocupação por parte dos desenvolvedores em validar a data enviada para o servidor.[11]

2.3 Análise Estática de Código e *JScrambler*

Nesta secção será feita uma análise do atual estado de ferramentas de análise de código, das principais aplicações e utilidades da mesma análise e um comparativo com o principal produto da empresa onde se insere o tema.

2.3.1 Definição

No mundo cada vez mais informatizado e orientado para a Web em que vivemos, a performance e proteção de código criado é motivo de grande preocupação por parte dos criadores de aplicações. Como tal existem aplicações para análise de código fonte e assim fornecer ao utilizador uma visão geral da qualidade do código seja ao nível de qualidade performativa, seja ao nível de segurança e vulnerabilidades. Quando falamos de segurança do código *JavaScript*, estamos essencialmente a referir minificação e ofuscação. Estes dois termos vão ser referidos várias vezes ao longo desta secção e para que a compreensão seja total serão definidos abaixo com recurso a figuras que mostram código antes e após as transformações.

```
function NewObject(prefix)
{
    var count=0;
    this.SayHello=function(msg)
    {
          count++;
          alert(prefix+msg);
    }
    this.GetCount=function()
    {
          return count;
    }
}
var obj=new NewObject("Message : ");
obj.SayHello("You are welcome.");
```

Figura 2.1: Código exemplo sem qualquer transformação aplicada

- **Minificação** - Método de remoção de todos os carateres desnecessários à execução da linguagem, essencialmente carateres que não afetem a funcionalidade do código a ser executado. Espaços e comentários são exemplos de remoções ocorrentes na minificação.

  Este método está associado a segurança, no entanto o nível de segurança é muito baixo e facilmente contornado. A minificação foca-se essencialmente na redução do tempo de

carregamento da página que, por sua vez está diretamente relacionada com a redução do número de *bytes* a serem carregados devido à eliminação dos carateres. Nas figura 2.2 está representado um exemplo de minificação aplicado ao exemplo apresentado na figura 2.1.

```
function NewObject(e){var t=0;this.SayHello=function(o)
{t++,alert(e+o)},this.GetCount=function(){return t}}var obj=new
NewObject("Message : ");obj.SayHello("You are welcome.");
```

Figura 2.2: Código exemplo com minificação

- **Ofuscação** - Este método consiste na colocação de obstáculos à compreensão do código e aqui existem muitas formas de o fazer e podem ir desde uma simples renomeação das variáveis e métodos à criação de estruturas de dados complexos e muito pouco legíveis. A ofuscação geralmente também aplica uma minificação no código. Associado à ofuscação costuma estar também associada uma perda de performance na execução do código transformado. O resultado da ofuscação feita pela ferramenta *JavaScript Obfuscator* [12] pode ser visto na figura 2.3

```
var _0xd77b=
["\x53\x61\x79\x48\x65\x6C\x6C\x6F","\x47\x65\x74\x43\x6F\x75\x6E\x74","\x4D\x65
\x73\x73\x61\x67\x65\x20\x3A\x20","\x59\x6F\x75\x20\x61\x72\x65\x20\x77\x65\x6C\
x63\x6F\x6D\x65\x2E"];function NewObject(_0xec44x2){var
_0xec44x3=0;this[_0xd77b[0]]=function(_0xec44x4)
{_0xec44x3++;alert(_0xec44x2+_0xec44x4);};this[_0xd77b[1]]=function(){return
_0xec44x3};}var obj= new NewObject(_0xd77b[2]);obj.SayHello(_0xd77b[3]);
```

Figura 2.3: Código exemplo com ofuscação

2.3.2 Enquadramento Histórico

A preocupação em seguir boas práticas de programação existe praticamente desde o surgimento das linguagens de programação e em 1985 este conceito passa a ser acompanhado da análise estática de código com o surgimento da ferramenta *PC-lint* para *C/C++*. [13] Desde então o surgimento deste tipo de ferramentas tem crescido e têm surgido cada vez com mais funcionalidades. As aplicações *Web* são cada vez mais recorrentes e consequentemente a utilização de *JavaScript* tem acompanhado este crescimento. As ferramentas de análise estática de *JavaScript* também começaram a surgir cada fez mais frequentemente e com mais funcionalidades.

2.3.3 Objetivo

A análise estática de código tem como principais objetivos identificar más práticas, aumentar a performance, através de eliminação de caracteres ou até otimização de código, e ainda ofuscar

o código de forma a torná-lo menos legível e consequentemente mais seguro. Olhando especialmente ao *JavaScript* este último objetivo tem sido um desafio uma vez que este é interpretado pelo que se encontra guardado em ficheiros de texto acessíveis a qualquer um. [1] O foco desta secção será então a análise estática de *JavaScript* e o seu estado atual referente a aplicações existentes. [14]

2.3.4 Análise de Soluções

Quando abordamos a análise estática de código *JavaScript* destacam-se dois tipos de aplicações, aplicações que através da análise de código detetam erros e possíveis problemas que possam estar no código, ou seja, ferramentas de controlo de qualidade e aplicações que, através da análise de código minificam e/ou ofuscam o mesmo aumentando a sua performance e segurança. Para o primeiro caso destacam-se três aplicações. ***Closure Linter***, desenvolvida e utilizada pela *Google* que garante que o código *JavaScript* siga um conjunto de diretrizes previamente definidas. [15] ***JSLint***, é uma ferramenta online que permite a análise de código *JavaScript* tendo suporte para Node.js. [16] ***JSHint***, outra ferramenta online que faz uma deteção em tempo real de erros e possíveis problemas no código, conta também com suporte para Node.js. [17]

Olhando agora a ferramentas de minificação/ofuscação destacam-se 3 ferramentas online e destas, 2 minificam o código e oferecem um nível básico de ofuscação, ***Daft Logic Online Javascript Obfuscator*** [18] e ***Closure Compiler*** [19]. Este nível é facilmente revertido com recurso a um ***beautifier*** [20] que, por análise estática do código consegue apresentá-lo de uma forma que o utilizador talvez consiga interpretar.

Outra das ferramentas, ***Javascript Obfuscator*** [12], e esta mais poderosa do que as referidas anteriormente. Dispõe ainda da possibilidade de usufruir de mais funcionalidades através de planos pagos que, por questões monetárias, não puderam ser testadas. Os resultados obtidos após o recurso ao *beautifier* já não são completamente legíveis no entanto ainda é possível de certa forma compreender o resultado.

Como estas foram encontradas mais algumas ferramentas que são úteis na otimização do *JavaScript* principalmente pela remoção de *whitespaces* reduzindo assim o tempo necessário para o carregamento dos *scripts* no entanto o nível de segurança oferecida é muito baixo. [21, 22] Alguns autores falam da combinação de duas destas ferramentas para aumentar o nível de segurança. [23]

Por fim existe uma solução que de todas as analisadas é a mais completa, o ***JScrambler***, esta ferramenta utiliza técnicas de minificação e ofuscação para transformar o código num formato muito complexo. É atualmente a solução mais completa de proteção e otimização de código. Oferece também suporte para Node.js na sua mais atual versão. [1]

Apesar de algumas pessoas se manterem céticas ao uso deste tipo de ferramentas, é evidente a evolução das mesmas e, é também natural que o crescente uso das linguagens *web* venha acompanhado de um aumento da procura e consequentemente da oferta destas ferramentas o que pode, de certa forma impulsionar a evolução da segurança neste sentido.

De seguida está representada uma tabela (tabela 2.1) que, de forma geral permite perceber as funcionalidades de cada uma das ferramentas.

A ofuscação será avaliada em três níveis, baixo, médio e elevado indicando a legibilidade do código após utilização do *beautifier* já referido acima.

O nível baixo é completamente legível o que significa que o código mantém a sua estrutura apenas com pequenas alterações a nomes de variáveis e funções.

No médio é possível interpretar mas requer algum conhecimento da linguagem. Neste nível existe alteração da estrutura de código com aumento da complexidade do mesmo.

O nível elevado representa um código com um nível de complexidade muito elevado, alteração da estrutura do código com criação de estruturas de dados complexas, novos métodos e novo código que não vai ser executado.

Para além desde parâmetro foram avaliados mais três campos.

Minificação, já definido anteriormente

Otimização que representa para além da tradicional minificação (por si só já é uma forma de otimização), uma simplificação de expressões constantes e remoção de código desnecessário ao bom funcionamento da aplicação.

Controlo de Execução, um tipo de segurança apenas presente numa das ferramentas mas que trás um enorme valor à aplicação podendo-se limitar a execução a determinados *browsers* e sistemas por exemplo.

Ferramenta	**Minimização**	**Otimização**	**Ofuscação**	**Controlo de Execução**
Daft Logic Online Javascript Obfuscator	Sim	Não	Baixo	Não
Closure Compiler	Sim	Não	Baixo	Não
Javascript Obfuscate and Encoder	Sim	Não	Baixo	Não
JavaScript Obfuscator - Free Online JavaScript Packer	Sim	Não	Baixo	Não
JavaScript Obfuscator	Sim	Não	Médio	Não
JScrambler	Sim	Sim	Elevado	Sim

Tabela 2.1: Comparativo de ferramentas de análise de código estática

Após esta análise pode-se concluir que existe uma ferramenta que se destaca claramente das restantes sendo a única a oferecer o controlo de execução, otimização e um nível de ofuscação elevado. Devido às semelhanças entre muitas das ferramentas pode-se ainda agrupar as mesmas em três principais tipos.

- Ferramentas de tipo 1 - Ferramentas que oferecem um nível de ofuscação muito básico e 100% reversível. [18, 19, 21, 22]
- Ferramentas de tipo 2 - Ferramentas que oferecem um nível de ofuscação intermédio. [12]
- Ferramentas de tipo 3 - Ferramentas que oferecem um nível de ofuscação avançado com funcionalidades extra. [2]

2.3.5 Aplicações Conhecidas

Como já referido a análise estática de código é essencialmente utilizada para aumentar a performance das aplicações *Web* reduzindo o tempo de carregamento das páginas no caso da minificação ou então para um aumento da segurança do código com alguma perda de performance inerente às transformações efetuadas pela ofuscação.

O *JScrambler* abre novos horizontes. Desde aplicações *Web* direcionadas para *smartphones* a jogos de *browser* inteiramente desenvolvidos em *JavaScript* e *HTML5*, a mais recente versão de *HTML*, surge aqui a possibilidade de investir em áreas que até então eram impensáveis pois não existiam ferramentas realmente eficazes nesta área com a preocupação de aliar uma ofuscação com um novo nível de segurança à otimização de código . [2]

2.3.6 Conclusão

A análise estática de código é vastamente usada e tem várias aplicações desde análise de qualidade do código a otimização e ofuscação. Neste caso em específico existe uma ferramenta completa que oferece uma solução muito superior a todas as existentes e que tem vindo a evoluir acompanhando o surgimento de novas tecnologias.

Capítulo 3

Content Security Policy

Este capítulo tem o foco na mais importante fase de investigação do tema de dissertação sendo o mecanismo que está na base da solução implementada. Foi feita uma análise da evolução e funcionalidade da ferramenta de segurança *CSP*. É apresentada uma análise do estado atual, limitações, suporte e desenvolvimentos futuros em prol da ferramenta.

3.1 Definição

Content Security Policy é um mecanismo usado por aplicações web para minimizar as possibilidades de ataques provenientes de injeção de conteúdo destacando-se aqui ataques de *Cross Site Scripting (XSS)*. Para tal são declarados através deste fontes seguras que vão carregar o conteúdo que o utilizador confia e isso permite que *scripts* provenientes de outras fontes que não as declaradas sejam detetados e bloqueados. [3]

3.2 Enquadramento Histórico

Esta ferramenta começou a ser discutida pela comunidade em geral em 2010 e assim começou a ser desenvolvida pela *Mozilla Foundation* sendo por fim lançada em 2011 uma implementação experimental do *CSP* na nova versão do *browser* da *Mozilla*, o *Firefox 4*. [24] Rapidamente foi reconhecido pelo acréscimo de segurança que isto poderia trazer a grande parte das aplicações web e assim em finais de 2012 é lançado pela *World Wide Web Consortium (W3C)* como uma *Candidate Recomendation* a versão 1.0 de *Content Security Policy*. [3] Desde então esta a integração desta ferramenta nos *browsers* tem vindo a evoluir com a introdução de novos *headers* e ainda com a contribuição para novas versões sendo que o *CSP2* já possui grandes melhorias a nível de segurança tendo sido lançada uma versão final em Janeiro de 2015 aprovada como uma *Candidate Recomendation* no mês seguinte [25]. Atualmente a versão 3 do *CSP* já se encontra em desenvolvimento com a introdução de uma nova diretiva em relação à versão anterior não existindo ainda qualquer suporte por parte dos *browsers*. [26]

3.3 Objetivo

Os ataques de *Cross-site Scripting (XSS)* consistem em "enganar" um *Website* levando-o a executar código malicioso juntamente com o o código necessário à execução do site. Isto pode comprometer dados que à partida não deviam ser partilhados. Com a ideia acabar com este tipo de ataques surge então uma ferramenta cujo principal objetivo é controlar o conteúdo carregado, controlo este que fica ao encargo da equipa de desenvolvimento do *Website* que por sua vez dispõe de um conjunto de diretivas que permitem ajustar a segurança às suas necessidades. Este mecanismo funciona com uma espécie de *whitelist* que declara as fontes de onde são esperadas que os recursos sejam carregados e assim impede a utilização de fontes não declaradas pelas políticas de *CSP* invalidando qualquer recurso maliciosos ou não que possa ser injetados por um atacante. *Scripts* são obviamente os recursos que mais riscos apresentam ao nível de segurança no entanto o *CSP* permite ir mais além e controlar praticamente todos os recursos que são carregados numa página *Web* desde imagens e estilos até conteúdos embebidos e origens a que uma página pode conectar *(WebSockets)*. [24]

O *Content Security Policy* é encarado como uma forma de defesa em profundidade e não substitui os métodos de segurança até aqui existentes pelo que não deve ser usado como única medida de segurança, é mais uma nova camada de proteção. [3]

3.4 Barreiras a Ultrapassar

Na versão atual do *CSP* a criação/manutenção das políticas à medida que a aplicação se vai desenvolvendo é muito trabalhosa uma vez que requer algumas práticas a fim de garantir todas as potencialidades e para além disso requer a constante verificação das políticas pois a qualquer momento uma política que até então funcionava perfeitamente pode entrar em conflito com um novo *script* ou *plugin* que foi incluído no *Website*. Entre estas práticas destaca-se a não utilização de código *inline* e isto por vezes podes ser complicado levando à alocação de muitos recursos, o que é dispendioso, ou até, num pior cenário, pode levar à desistência de utilização da mesma. A má utilização das políticas de *CSP* pode levar a *downtime* e isto numa situação crítica por levar a perdas irreversíveis.

3.5 Vulnerabilidades

Apesar de em teoria esta ferramenta ser bastante segura por apenas permitir a execução de recursos *whitelisted* estes mesmos podem ser comprometidos se, de alguma forma o ataque focar estes recursos (*scripts* ou *plugins*), a primeira versão do *CSP* não providencia qualquer forma de limitar o número de *scripts* a serem executados pelo que o utilizador não tem qualquer tipo de controlo sobre *scripts* ou *plugins* que estejam especificados nas políticas. Dependendo das políticas outros recursos podem também ser utilizados para atacar, daí a elevada importância em implementar políticas o mais restritas e específicas possível.

Uma vulnerabilidade que tem sido alvo de alguma discussão surge a par da implementação de *JSONP* que em condições normais permite a especificação de uma função *callback* no *URL* tornando assim o *CSP* obsoleto e desta forma a aplicação fica vulnerável a injeção de conteúdo, apesar de esta não estar diretamente ligada ao mecanismo de segurança não existe nenhuma forma de bloquear os *scripts* injetados desta forma.

O *CSP2* introduz alguns novos conceitos que procuram melhorar o nível de segurança. Uma das principais melhorias surge ao nível da especificação dos recursos *whitelisted* em que contrariamente à versão anterior permite a inclusão de caminhos para diretórios e ficheiros o que aumenta bastante o nível de segurança, principalmente quando estamos a importar *scripts* de terceiros. [26]

Outra novidade que surge com a nova versão é a implementação de um *meta element* para definição de *CSP* diretamente no *HTML*. Apesar de parecer uma boa ideia a sua utilização não é recomendada. Mesmo a implementação deste elemento sendo feita no *head* onde injeções de conteúdo não são muito vulgares é possível fazê-lo e assim de alguma forma evitar que estas políticas sejam aplicadas. [24]

O *CSP* está ainda sujeito a vulnerabilidades de terceiros. Exemplo disto são vulnerabilidades que possam surgir no *Domain Name System (DNS)* [1] ou até nos *browsers* que de certa forma podem afetar ou até impedir o correto funcionamento do mesmo. [26]

Por fim, ao longo do desenvolvimento a ferramenta está sempre sujeita a vulnerabilidades e *bugs* que possam surgir nos *browser* sendo da responsabilidade das empresas detentoras dos mesmo reduzir este risco ao máximo e procurar sempre fornecer o suporte o mais atualizado possível.

3.6 Implicações

O mecanismo de segurança carrega neste momento um custo derivado da adoção de algumas práticas de desenvolvimento. De certa forma as práticas recomendadas pela especificação do *CSP* não estão erradas, no entanto, por vezes, pode ser complicado a adoção das mesmas e estas, apesar de não impedirem a utilização da ferramenta, afetam drasticamente a sua eficácia.

- Remoção de todo o código *JavaScript inline*;
- Não utilização de métodos *JavaScript* cuja funcionalidade passe por avaliar os argumentos e possivelmente executá-los (*eval()* e *setTimeout()*);

A adoção destas duas regras de ouro permite a posterior restrição da utilização das mesmas e assim reduzir os riscos de ataques deste tipo. Este problema é facilmente contornável e fica um bocado ao critério do desenvolvedor que, se não entender como obrigatório, pode ignorar estas práticas e mesmo assim integrar o *CSP* no seu projeto tomando consciência ele próprio dos riscos acrescidos que tem. A versão 1.1 do *CSP* (também referida como *CSP2*) apresenta uma

[1] sistema responsável pela tradução dos domínios de um *website* nos seus respetivos endereços de *IP* que são identificadores de uma máquina na *Internet*

funcionalidade que traz um acréscimo de segurança ao código *JavaScript inline* através do uso de *nonce* e *hash*.

O *nonce* é um identificador único declarado numa política e no *script* que se pretende identificar ficando a execução de código *inline* limitada ao identificado pelo *nonce*.

A *hash* é uma forma de obter um identificador único do código a ser incluído *inline*. Para o propósito pode ser utilizada uma codificação SHA-256 ou SHA-512 e para declarar basta que a *hash* do código permitido seja incluída na política para que a execução do mesmo seja validada.

Existe, no entanto, um problema maior que tem impedido a vulgarização desta ferramenta, atualmente a criação de políticas envolve elevada carga de trabalho que, consequentemente, implica a alocação de alguns recursos que aumentam significativamente os custos de desenvolvimento. O mesmo sucede para a manutenção sendo ainda necessário que seja dada a devida atenção à mesma a fim de evitar problemas maiores com a aplicação, como *downtime*.

Assim, os recursos alocados em primeira instância para a integração do mecanismo com a aplicação a ser desenvolvida têm de ser mantidos e possivelmente reforçados ao longo do desenvolvimento a fim de garantir o constante bom funcionamento das mesmas políticas. Desta forma, neste momento a integração do *CSP* tem de ser bem balanceada com o esforço que será aplicado na restante aplicação para evitar alocação excessiva de recursos ou problemas que possam afetar a experiência de utilização final.

3.7 Geração e Análise de Performance

A geração de políticas de *Content Security Policy* tem sido alvo de alguma investigação e como tal já existem algumas soluções que vão desde ferramentas podem ser integradas com projetos e fornecem um conjunto de métodos que vão posteriormente gerar as políticas respetivas até *websites point and click* em que o utilizador mesmo sem grande conhecimento da notação do *CSP* pode gerar políticas para o seu *website*. [27, 28, 29, 30] Estas ferramentas já representam alguns avanços relativamente aos obstáculos na criação de políticas, no entanto nenhuma das soluções permite a análise de código e consequentemente a manutenção das políticas continua a ser um obstáculo pois o desenvolvimento continua a implicar a constante verificação das mesmas. Das soluções analisadas o mais próximo que poderíamos chegar de uma solução ideal seria com a utilização de duas ou mais ferramentas a fim de permitir a integração com o projeto e a sugestão de políticas baseadas nos relatórios gerados pelas já existentes. Nesta secção foram analisadas 4 ferramentas que primam na geração de políticas de *CSP* e manutenção das mesmas.

PHP Content Security Policy Generator [27]

Esta ferramenta foi desenvolvida em *PHP* e pode ser integrada no projeto a ser desenvolvido disponibilizando um conjunto de métodos que posteriormente vão gerar políticas de *CSP*. Dispensa do conhecimento da notação da ferramenta, é *open-source* e muito fácil de utilizar em *PHP*.

No entanto esta solução serve apenas para a geração de políticas especificadas através dos métodos pelo que não faz qualquer tipo de análise de código o que exige uma constante verificação das políticas apesar de simplificada.

CSP Is Awesome [28]

Ferramenta online de geração de *CSP*. O utilizador dispõe de um conjunto de secções em que deve selecionar o que pretende e por fim são geradas as políticas. Não é requerido qualquer tipo de conhecimento da notação de *Content Security Policy*. A utilização não tem qualquer custo. Não possui qualquer integração com o projeto a ser desenvolvido, consequentemente, não existe análise de código.

Caspr/Enforcer [29]

Esta solução é composta por um *endpoint* para os relatórios de erro e uma extensão para o *browser* para testar as políticas e reportar para o *endpoint*. O ***Caspr*** permite gerar políticas de forma idêntica à ferramenta referida anteriormente. O recurso à extensão permite que seja feito o teste das políticas em tempo real que por sua vez envia os relatórios para o *endpoint*. Permite ainda gerar políticas sem necessariamente ter conhecimentos para tal. Mais uma vez a análise de código é inexistente e a integração com o projeto não é total.

CSP Builder [30]

Endpoint de políticas que contém um sistema de sugestões de melhorias construído através dos relatórios de erro recebidos simplificando de certa forma o processo de verificação dos pontos de falha. Sendo um *endpoint* oferece uma visão simplificada dos relatórios e para além disso por análise dos mesmos é capaz de fornecer sugestões. A integração com o projeto é limitada e a análise de código é inexistente.

CSP-Fiddler-Extension [31]

Esta é uma solução desenvolvida sob a licença *MIT* como um extensão para a ferramenta *Fiddler* que é um *proxy* [2] que permite, entre outros, monitorizar o tráfego de uma aplicação web alvo.

Esta solução cria uma política o mais restritiva possível, impedindo o carregamento de qualquer recurso e depois capta os relatórios de violações de políticas e começa a permitir os recursos à medida que o utilizador navega pela aplicação enquanto esta está a ser monitorizada pelo *Fiddler*.

Esta é uma abordagem diferente das analisadas até aqui e de certa forma melhor. Esta solução tem grandes vantagens em questões de suporte para as várias aplicações existentes uma vez que está apenas dependente dos pedidos feitos ao servidor. Desvantagens vão para a dependência da ferramenta *Fiddler* e para a necessidade do utilizador navegar pelas páginas da aplicação o que

[2] Servidor que atua como intermediário para pedidos feitos pelo cliente ao servidor da aplicação alvo.

numa aplicação de grandes dimensões pode ser inconcebível.

A solução a ser desenvolvida tem integração total com qualquer projeto que seja desenvolvido e através de uma análise do código fornecida pela já existente ferramenta *JScrambler* é possível gerar políticas de *CSP* adequadas e direcionadas ao produto pelo que vai mais além das soluções já existentes.

Fazendo uma análise comparativa às funcionalidades das soluções analisadas e da solução a ser desenvolvida é possível ver abaixo os resultados obtidos. Na seguinte tabela foram selecionadas características que se revelaram importantes entre as já existentes e a proposta.

Geração de Políticas refere a possibilidade de através da aplicação obter políticas de *CSP* da aplicação prontas a serem utilizadas, esta geração pode ser feita de várias formas desde análise de código sendo esta a que torna o processo de geração mais autónomo, geração por intermédio de uma notação . Necessário conhecer a notação como o nome indica avalia a necessidade de conhecimento de qualquer tipo de notação de *CSP* para utilizar a ferramenta. Integração com o projeto avalia a possibilidade de utilizar a aplicação com um projeto que terá posteriormente o *CSP* a executar. Sugestões representam um conjunto de mensagens a ser entregues ao utilizador da aplicação de forma a tirar o maior partido possível da aplicação. Análise de Código por fim refere como o nome indica a análise do código de *JavaScript* para o processo de geração de políticas ser o mais independente possível. As várias ferramentas aqui apresentadas foram analisadas em termos funcionais nos parâmetros já referidos e permitiram chegar às conclusões apresentadas na tabela.

Ferramenta	**Geração de Políticas**	**Necessário conhecer notação CSP**	**Integração com Projeto**	**Sugestões**	**Análise de Código**
PHP Content Security Policy Generator	Sim	Parcialmente	Sim	Não	Não
CSP is Awesome	Sim	Não	Não	Não	Não
Caspr/Enforcer	Sim	Não	Parcial	Não	Não
CSP Builder	Não	Não	Parcial	Sim	Não
CSP-Fiddler-Extension	Sim	Não	Não	Não	Não
Solução Proposta	Sim	Não	Sim	Sim	Sim

Tabela 3.1: Comparativo de ferramentas de geração de políticas de *CSP*

Pela análise feita foi possível constatar a existência de uma falha quando se fala de integração com o projeto e análise de código, desta forma a solução proposta pretende colmatar esta falha e ainda aliar as melhores funcionalidades de cada uma das soluções.

A automatização deste mecanismo (*CSP*) representa um grande avanço não só para a ferramenta como também para a segurança na *Web* em geral. Posto isto, a aplicação aplicação a ser desenvolvida terá como entrada um projeto que será analisado de forma estática em busca de recursos e permitirá gerar um conjunto de políticas de *CSP* adaptadas ao código e o mais restritivas possíveis. Esta solução terá portanto como resultado um conjunto de políticas resultantes

da análise estática de código. Esta geração desempenha também o papel de manutenção uma vez pretende-se que a pesquisa por recursos e subsequente geração de uma política seja sempre o mais restritiva possível havendo sempre uma especificação de todos os recursos da aplicação.

3.8 Aplicações Conhecidas

Twitter

Pouco após o seu lançamento o *Twitter* começou a testar o *CSP* na versão *mobile* da sua principal aplicação, o *Twitter*, todos os utilizadores do *Firefox 4* passaram a dispor desta nova camada de proteção quando acedessem ao *link mobile.twitter.com*. A adoção desta ferramenta foi feita de forma experimental daí a implementação na versão *mobile* do site por este ter menos tráfico e assim qualquer falha afetar o mínimo de utilizadores possível. Em 2013 a empresa expandiu o uso desta ferramenta para todos os seus produtos à exceção da versão *desktop* do *Twitter* por se tratar do projeto mais antigo e conter uma grande quantidade de código *inline* o que torna a migração um processo muito mais demorado. A empresa afirma que o lançamento da nova versão do *CSP* trará importantes modificações que vão tornar a ferramenta viável para uma audiência maior. Dão especial destaque à possibilidade de na nova versão ser possível permitir a execução de *inline-script* devidamente identificados no *header* e ainda ao conceito que está a ser introduzido que consiste numa *API DOM* para interagir com o *CSP*. [32, 33]

Google

Pouco depois do lançamento a *Google* incorporou este conceito no seu sistema de extensões restringindo principalmente a execução de *scripts* externos e foi notória a queda no número de ataques às vulnerabilidades das mesmas sendo que neste momento se encontra próximo de zero. [34]

GitHub

A plataforma de armazenamento e partilha de projetos que utilizam o sistema de controlo de versões *Git* utiliza o *Content Security Policy* desde 2013 e a integração da ferramenta na sua plataforma contou com a ajuda da *Google*. [35]

3.9 Versões/Suporte

O *CSP* encontra-se na versão 1.0 no entanto a versão 1.1 *(CSP2)* já está numa fase experimental podendo ser utilizada através do *Google Chrome*. Esta nova versão traz algumas novidades que são do agrado dos desenvolvedores, sendo a nova diretiva, *script-nonce*, a que tem despertado mais interesse ajudando assim a ultrapassar os problemas do *inline code* através da inclusão de uma chave única tanto no *script* como na diretiva e desta forma, a não ser que o atacante saiba a chave *nonce*, qualquer *script* injetado por ele não será executado. Algum destaque também para a possibilidade de com esta versão se poder restringir a execução de *scripts* a ficheiros ao invés de diretórios inteiros.

À data desta análise o *CSP* é suportado pelos *browsers Google Chrome 43* e *Firefox 38* lançados em Maio de 2015, *Opera 29* disponível desde Abril do presente ano 2015 e *Safari 8* disponibilizado com a nova versão do sistema operativo *Yosemite* em Outubro de 2014. É ainda parcialmente suportado pela última versão do *Internet Explorer 11* que recebeu a última atualização em Dezembro de 2014.

O novo *browser* da *Microsoft*, *Edge* lançado em Maio de 2015 já apresenta suporte para a ferramenta de segurança.

Não foram encontrados dados relativos ao suporte para a nova versão de *CSP* já validada pela *W3C*.

3.10 Conclusão

Desta forma podemos constatar que estamos perante uma tecnologia que ainda está em desenvolvimento e a sofrer melhorias mas que pode ser a solução para acabar com vulnerabilidades de injeção de conteúdo.

Capítulo 4

Especificação e Implementação da Solução

Tendo em atenção as funcionalidades indicadas anteriormente, foi desenvolvida um solução que parte da análise estática de código e recolhe os recursos a fim de criar uma política o mais restrita possível de forma a garantir a segurança da aplicação na área de injeção de conteúdos, sendo a principal meta a atingir um cenário em que a injeção de conteúdo por *cross-site scripting* seja impossível.

Neste capítulo serão abordadas as metodologias adotadas, requisitos, funcionalidades e ainda estrutura e funcionamento da solução desenvolvida.

4.1 Metodologia

Seguindo a ideia de utilização da metodologia ágil de desenvolvimento *SCRUM*, foi adotada uma versão simplificada da mesma com *sprints* semanais em que no fim de cada um era feita uma análise do trabalho desenvolvido bem como um planeamento do próximo *sprint*. Difere do método tradicional no cálculo da velocidade e do peso de cada uma das *user stories* que não foi aplicado neste projeto.

As principais vantagens da adoção desta metodologia vão desde um melhor controlo do progresso do desenvolvimento do projeto bem como a possibilidade de semanalmente poder avaliar a correta execução do mesmo e corrigir alguma falha sendo sempre priorizada a qualidade e a funcionalidade do projeto. A soluções foi desenvolvida ao longo de 3 meses representados por 11 *sprints*.

Nesta secção será apresentada apenas uma visão geral do desenvolvimento da aplicação pelo que serão apresentadas alguns conceitos que terão o devido destaque posteriormente nas secções correspondentes.

Sprint 1

Este *sprint* focou essencialmente a familiarização com a ferramenta *Content Security Policy* e exploração das suas funcionalidades, suporte em *browsers* e integração em diferentes linguagens de programação.

Para além disso foram ainda estudadas as principais ferramentas a serem utilizadas no desenvolvimento.

Relativamente ao suporte pode-se concluir o que já foi constatado anteriormente que todos os browsers testados (*Chrome, Firefox e Safari*) suportam as principais diretivas de *CSP*, (diretivas definidas na atual *W3 Recommendation* [3]) e que a *Google* tem sido a empresa que tem investido mais esforços no desenvolvimento da ferramenta apresentando já suporte para diretivas propostas para a nova especificação do *CSP*.

Quanto à integração em diferentes linguagens de programação foram feitos testes para as duas linguagens mais utilizadas na *Web*, *PHP e ASP.NET* [36], para a linguagem a ser utilizada no desenvolvimento da solução, *Node.JS* e para dois *web servers* mais utilizados, *Apache e Nginx* [37].

Por fim foi feito o planeamento do projeto e foram estabelecidas as prioridades para este projeto.

Sprint 2

O segundo *sprint* marcou o início do desenvolvimento da solução. Sendo o objetivo automatizar o processo de criação de *CSP* começou-se pela análise de código para a deteção de recursos a serem incluídos na política.

Na sequencia da investigação relativa à integração do conteúdo gerado em diferentes linguagens de programação foi também criado o módulo que a partir de uma estrutura de dados *JSON* vai criar uma política final sob a notação de uma linguagem fornecida inicialmente.

Sprint 3

Na terceira semana de desenvolvimento, deu-se continuidade à parte da análise de recursos e definiu-se a arquitetura da solução que está descrita com mais detalhe na secção seguinte 4.3.

Sprint 4

Até aqui a solução foi desenvolvida de forma modular sendo que cada um dos módulos funcionava independentemente, esta prática facilitou a integração da mesma num esqueleto com uma estrutura bem definida desenvolvido pela empresa.

Estendeu-se ainda a análise estática a ficheiros *CSS* e estabeleceram-se standards para a estrutura de armazenamento dos recursos detetados. No anexo A.1 é possível ver o formato definido para o resultado da análise de código.

Uma visão mais detalhada desta estrutura modular pode ser vista na secção 4.3.

Neste *sprint* também se começou a pensar na parte de interação do utilizador com a aplicação.

Sprint 5

Nesta fase foi criado um módulo que recolhe todos os recursos encontrados e cria uma estrutura de dados *JSON* num *standard* já definido anteriormente. O anexo A.2 apresenta a estrutura definida para esta estrutura de dados.

Foi estabelecida uma ligação entre os três módulos de lógica até aqui desenvolvidos, análise de código envia os resultados para o módulo de geração de uma estrutura de dados standard que por sua vez envia os resultados para o módulo de geração de uma política de uma linguagem previamente definida.

Para além do desenvolvimento da solução começaram a desenvolver-se testes unitários para toda a lógica do sistema.

Sprint 6

Com o início dos testes aos módulos começaram a ser detetados *bugs* e portanto neste *sprint* o foco foi essencialmente a correção dos bugs. Posteriormente foi desenvolvida uma *API* para tratar dos pedidos feitos pelo lado do cliente.

Sprint 7

No *sprint* 7 foi feita a extensão da deteção de recursos para as restantes diretivas e a conclusão dos módulos até aqui desenvolvidos. **Análise de Código**, **Geração de um Standard CSP** e **Geração de um CSP final**.

Sprint 8

Completa a lógica definida inicialmente os esforços voltaram-se para a interface de interação do utilizador com a aplicação e integração da mesma com a lógica do servidor. Esta interface será analisada em detalhe na secção 4.5.

Sprint 9

Esta foi uma fase de reestruturação e definição de novos objetivos pois após alguns testes da solução até aqui desenvolvidos detetaram-se algumas falhas relevantes para a viabilidade da mesma ao nível de suporte a várias várias linguagens e *frameworks*. Estes problemas estão descritos em maior detalhe na subsecção relativa a este assunto presente na secção 4.7.

Foram então criadas novas tarefas para melhorar a solução e colmatar a falha e começou-se a desenvolver esta nova abordagem que conta com três principais funcionalidades, *Crawl Website*, *Regex* e Suporte para linguagens de *template*, estas funcionalidades serão descritas com maior detalhe na secção referente às funcionalidades da aplicação.

Sprint 10

Continuação do desenvolvimento das funcionalidades referidas anteriormente e integração com o resto da solução.

Sprint 11

O último *sprint* ficou reservado para teste da solução final e correção de *bugs* que surgissem nesta fase. O processo de validação e testes efetuados será descrito em detalhe no capítulo 5.

4.2 Requisitos da Aplicação

O desenvolvimento da solução passou por duas principais fases de definição de requisitos que acompanharam a evolução da solução. A primeira fase começou com o planeamento inicial do projeto, cumpridos os requisitos desta e avaliada a funcionalidade da solução em cenários de utilização do mundo real foram apontadas as limitações e definidos novos requisitos para uma segunda fase a fim de ultrapassar as mesmas.

Aos requisitos definidos foram atribuídas prioridades seguindo uma escala qualitativa de avaliação com três níveis de prioridade, **Alta**, **Média**, **Baixa** para requisitos obrigatórios e essenciais para o funcionamento da solução, opcionais que trariam mais funcionalidade à solução e opcionais que não acrescentam grande valor respetivamente.

#	Requisitos	Prioridade
1	**Fase 1**	
1.1	Análise Estática de Código	Alta
1.2	Geração de CSP Standard	Alta
1.3	Geração de CSP Final	Alta
1.4	Interface de Utilizador	Alta
1.5	Editar Recursos	Média
2	**Fase 2**	
2.1	Crawl Website	Alta
2.2	Suporte para linguagens de servidor	Alta
2.3	Suporte para linguagens de template	Alta
2.4	Hash de código inline	Média
2.5	Expressões Regulares para deteção	Baixa
2.6	Gerar CSP por página	Baixa

Tabela 4.1: Definição de Requisitos da Solução de Geração de *CSP*

A tabela 4.1 permite de uma forma geral visualizar a distribuição dos requisitos definidos bem como prioridades para cada um deles.

De notar que os requisitos que foram definidos com prioridade baixa acabaram por não ser implementados.

Para o caso das **Expressões regulares para deteção**, a ideia neste requisito é procurar através de expressões regulares *url's*/recursos de tal forma que numa situação em que o este não fosse detetado pela análise de código haja a possibilidade de ser detetado por este método.

Este requisito foi adiado para trabalho futuro e será referido na secção 6.3. Na fase em questão deu-se prioridade à afinação da solução através de testes para correção de erros e *bugs*.

Já para o requisito *Gerar CSP por página* a ideia foi abandonada dada a complexidade e o reduzido valor acrescentado à solução. Na implementação atual, os recursos são todos declarados numa só política que após integrada no projeto alvo será enviada para um cliente sempre que uma página é carregada. Sendo o principal foco da solução a segurança, este requisito não traz nenhum acréscimo nesta área, contribuindo apenas para um aumento de performance.

A figura 4.1 permite de uma forma geral como foram executados os requisitos ao longo do tempo. pode-se obter uma visão geral do progresso de desenvolvimento da solução.

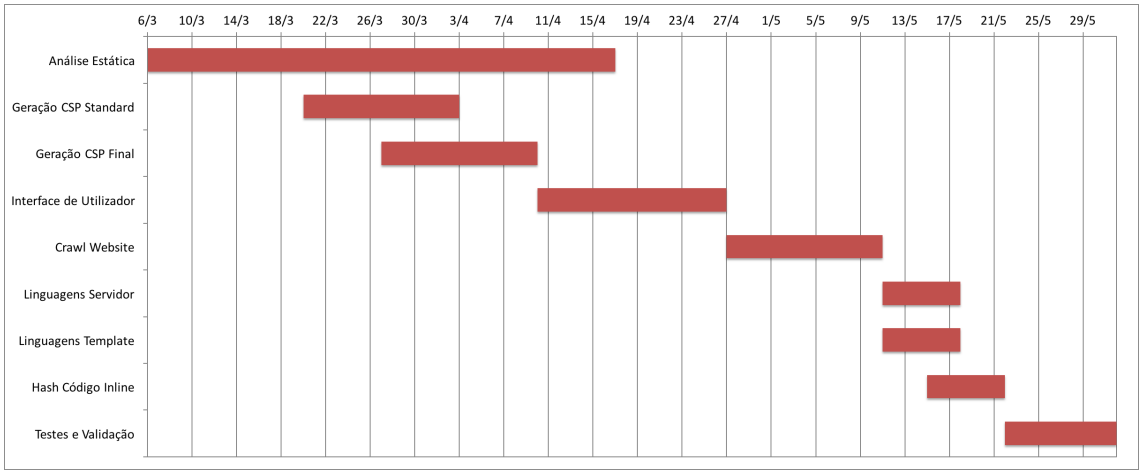

Figura 4.1: Diagrama de Gantt

4.3 Arquitetura da Aplicação

A aplicação está dividida em 5 principais módulos interligados entre si. O critério de divisão dos módulos foi atribuir a cada um deles uma funcionalidade distinta.

Cada um dos módulos foi desenvolvido de forma modular e capaz de atuar de forma independente isto possibilita a integração de qualquer um dos módulos noutro projeto ou até a alteração da funcionalidade dos mesmos sem afetar os outros. Atendendo ao facto do *CSP* ser uma ferramenta muito recente, isto é importante pois assim, é fácil estender a funcionalidade da aplicação desde deteção de novos recursos, suporte para mais linguagens ou até reestruturação dos formatos das políticas.

A solução no seu estado atual funciona independentemente do produto da empresa tendo sido tomada a opção de uma solução *standalone* com uma lógica modular que facilmente pode ser integrada em outros projetos. Assim, a ideia de integração com o *Jscrambler* é possível e representa uma extensão da funcionalidade do mesmo.

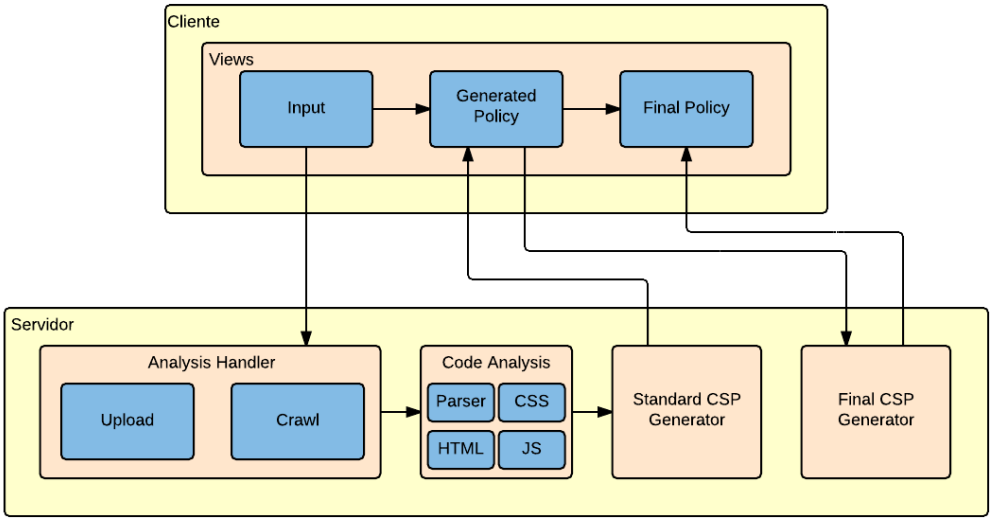

Figura 4.2: Diagrama de Arquitetura da Aplicação

4.3.1 Views

O primeiro módulo é a de interação com o utilizador, *Views*, este módulo conta com três sub módulos e define a interação do utilizador com a aplicação.

Input

Área inicial da aplicação, permite que o utilizador defina entre dois métodos de interação com o servidor e a comunicação vai ser estabelecida através de uma *API REST* com dois tipos de pedidos distintos. As interações com o servidor serão especificadas com mais detalhe na secção seguinte (secção 4.6).

Generated Policy

Após uma primeira interação com o servidor, o utilizador vai receber uma listagem dos recursos detetados que serão mostrados neste sub-módulo. Os mesmos podem ser personalizados e enviados para o servidor a fim de gerar a política final.

Final Policy

Esta é a última área de interação com a aplicação, aqui o utilizador pode ver a política gerada e copia-la para a área de transferência a fim de a integrar com o projeto.

4.3.2 Analysis Handler

O servidor está preparado para receber dois tipos de pedidos e consoante o tipo, vai partir de métodos diferentes para fazer a análise de código.

Neste módulo é feita a análise do pedido feito pelo cliente e este é reencaminhado para o respetivo sub módulo.

Upload

Quando é carregado um projeto para o servidor, este vem compactado num formato *zip* e neste sub módulo o ficheiro é descompactado e os ficheiros são avaliados sendo distribuídos de acordo com os formatos. É preciso ter em atenção que por muito completa que seja a solução o suporte para as inúmeras *frameworks* e linguagens de programação vai sempre ser um fator limitador. Assim, este método executa três tipos de busca no projeto carregado.

Ficheiros *HTML, CSS, JavaScript*, são diretamente enviados para o módulo seguinte, *Code Analysis*.

Ficheiros de *template engines*, são convertidos para a linguagem traduzida, no caso, (*HTML*) e de seguida são enviados para o módulo seguinte, *Code Analysis*.

Ficheiros de linguagens de servidor, são testados sob uma expressão regular que faz uma busca por *strings* e de seguida, o conteúdo do ficheiro em questão é enviado para o módulo seguinte juntamente com as strings encontradas.
A grande desvantagem deste método é a limitação da análise às linguagens suportadas.

Crawl

O segundo método é geralmente mais demorado (dependendo da dimensão do website) no entanto mais eficaz na deteção de recursos, isto porque não está dependente de qualquer linguagem ou *framework* sendo que os ficheiros resultantes do *crawl* vão sempre ser ficheiros *CSS, HTML ou JavaScript* já obtidos apartir do servidor e portanto ultrapassam já algumas barreiras do dinamismo, desta forma apenas os recursos carregados a partir do *JavaScript* é que não vão ser detetados.

Posto isto facilmente se percebe que o dinamismo é o único obstáculo da solução que na abordagem feita ao problema seria muito complicado resolver.

4.3.3 Code Analysis

Neste módulo é feita toda a análise ao código que foi recolhido do módulo anterior, esta análise começa com o sub módulo *parser*, o *parser* agrega a funcionalidade de três módulos de terceiros (*css, htmlparser2 e esprima*) que interpretam o código e constroem uma árvore hierárquica em formato *JSON* do código, esta vai depois ser recolhida pelo respetivo sub módulo.

Os sub módulos de análise do código *CSS, HTML, JavaScript* têm uma estrutura semelhante, para cada uma delas existe um método que vai percorrer a árvore de forma estática e para cada nó vai ser verificada a existência de recursos correspondentes ao tipo de ficheiro, sendo, caso existam,

armazenados numa estrutura de dados com um formato standard que posteriormente será enviada ao módulo seguinte.

Após análise das diretivas de *CSP* foram avaliadas as formas como os vários recursos podem ser carregados estaticamente tendo-se chegado à seguinte conclusão.

- **CSS** Podem ser carregados recursos de fontes e imagens.
- **HTML** Podem ser carregados recursos a partir das *tags base, (i)frame, form, img, video, audio, object, embed, applet, script e link* e podem também ser utilizados *inline styles e inline scripts*
- **JavaScript** Podem ser carregados recursos apartir de comunicações com o servidor (*EventSource, WebSocket e XMLHTTPRequest*) e pode também ser utilizado o *eval* que não é recomendado por questões de segurança.

4.3.4 Standard CSP Generator

Este módulo recebe um *JSON* com todos os recursos detetados pelo módulo anterior e percorrendo um a um cria uma nova estrutura de dados com um formato standard a ser interpretado pelo módulo seguinte.

O principal objetivo deste módulo é associar os recursos às respetivas diretivas de *CSP* independentemente de ter sido encontrado no *CSS, HTML ou JavaScript*. Este módulo também é responsável pela atribuição das palavras chave às diretivas que não tenham nenhum recurso detetado ou que recorram a *eval*. As possíveis diretivas e palavras chave do *CSP* podem ser vistas nas respetivas secções do anexo C.

No anexo A.2 é ainda visível o formato resultante da geração desta estrutura de dados *standard*.

4.3.5 Parser to Project Language

Este é o último módulo, o que retorna a política final na linguagem do projeto carregado pronta a ser integrada na aplicação *web*. Este módulo tem suporte limitado para um conjunto de linguagens do lado do servidor ou tipos de servidores. A solução desenvolvida tem suporte para três linguagens do lado do servidor (*ASP.NET, Node.js e PHP*) e dois tipos de servidor (*Apache e Nginx*). No anexo A.3 pode-se ver um exemplo de uma política gerada para cada uma das linguagens suportadas.

Quando o utilizador valida os recursos detetados e especifica o formato (linguagem ou *webserver* respetivo) e modo de imposição que pretende (Normal ou de *Report*), é enviado para este módulo uma política com a estrutura definida no módulo anterior. A partir do formato é então criada uma *string* que vai conter a política na notação respetiva e no modo desejado. O resultado da geração para os vários formatos disponíveis pode ser visto na secção A.3 do anexo A

4.4 Funcionalidades da Aplicação

A aplicação de geração de *CSP* tem um conjunto de funcionalidades que pretendem melhorar a experiência de utilização e automatizar o processo de geração com um mínimo de intervenção do utilizador na mesma.

Atendendo aos requisitos que foram especificados, a aplicação tem definidos dois modos de atuação.

Carregamento de Projeto em formato *zip*

- Suporte para CSS, HTML, JavaScript
- Suporte para a linguagem de *template* Jade [38]
- Suporte para as linguagens de servidor PHP, ASP.NET e Node.JS

Crawl de Website apartir de um URL

- Suporte para CSS, HTML, JavaScript enviado pelo servidor

Aliado a estes dois métodos estão um conjunto de funcionalidades que vão ser responsáveis pela deteção da maior parte dos recursos.

- *Parse* e análise de todo o código *CSS, HTML e JavaScript*
- Criação de uma política provisória que pode ser editada pelo utilizador
- Geração de uma política final no formato desejado.

Os ficheiros *CSS, HTML e JavaScript* são enviados diretamente para o respectivo *parser* que no caso do *HTML* deteta estilos e *scripts inline* e estes são também enviados para os *parsers* correspondentes. Para os recursos *inline* é ainda gerada uma *hash* que pode ser declarada na política e codifica o recurso limitando a execução de código *inline* ao declarado.

Relativamente às linguagens de *template*, foi desenvolvida uma solução para *jade* como prova de conceito indicativo que o suporte pode ser estendido a múltiplas linguagens de *template* e *frameworks*.

O suporte para as linguagens de servidor é limitado, é utilizado o *parser* de *HTML* que com recurso a expressões regulares simples vai executar sobre todo o código *HTML* que seja encontrado. foram selecionadas as duas linguagens com maior utilização em *websites (ASP.NET e PHP)* e foi ainda selecionada a linguagem *Node.JS* por ser a linguagem também utilizada na implementação da solução.

O *crawl* de um *website* uma vez que já obtém todos os ficheiros num formato interpretado pelo *browser* (*CSS, HTML, JavaScript*) apenas tem de enviar todos os ficheiros para os respetivos *parsers*.

O *parse* de código é feito por módulos de terceiros, *css-parser [39], htmlparser2 [40] e esprima [41]*, traduzem o código respetivo uma árvore *JSON*. Este resultado é depois analisado pelo módulo criado para o propósito que vai percorrer a árvore e procurar por recursos do vários tipos e armazená-los numa estrutura de dados.

Os recursos encontrados são todos agrupados num só *JSON* e enviados para o módulo seguinte que traduz a entrada num formato *standard* e esta pode ser editada pelo utilizador da ferramenta que pode adicionar, editar e remover recursos, pode definir o formato da política final entre linguagens de servidor (*ASP.NET, Node.JS, PHP*) ou mesmo mesmo o servidor em si (*Apache, Nginx*) e ainda o modo de imposição da política que pode estar definida em modo normal (*Content-Security-Policy*) em que a política e imposta e os carregamento de recursos que as violem é impedido ou então modo de *report* (*Content-Security-Policy-Report-Only*) em que a política é imposta e é geralmente utilizada para testar o efeito da mesma num *website* sendo que os recursos não são bloqueados, apenas são gerados os relatórios de erro para os recursos que sejam bloqueados.

Por fim a geração de uma política final num formato desejado entre os formatos acima referidos, o utilizador apenas tem de copiar a política e integrá-la no seu projeto.

4.5 Interface da Aplicação

O foco da solução a desenvolver não passava pela interface gráfica da aplicação no entanto tendo em conta a natureza da mesma, definiu-se como uma mais valia o desenvolvimento da mesma para se ter uma ideia das potenciais vantagens da solução e que tipo de interação o utilizador poderia ter com a aplicação. Foi ainda uma mais valia na fase de testes pois permitiu de uma forma mais precisa e direta obter os resultados.

A aplicação tem três principais *views* que surgem através do processo de geração de políticas para uma aplicação *web*.

4.5.1 Input Area

Esta secção surge inicialmente e pode ser observada na figura 4.3 onde é visível a possibilidade de interagir com a aplicação através do *upload* de um projeto ou da inserção de um *url* para *crawl*.

4.5.2 Generated Policy

Após a análise de código o utilizador tem acesso à área apresentada na figura 4.4 onde pode ver todos os recursos encontrados, adicionar/remover recursos, definir o formato e o modo de imposição da política e ainda verificar algumas definições avançadas (existência de *eval* ou definição de um *url* para os relatórios de erro gerados por violação das políticas).

Figura 4.3: Área de interação para entrada de um projeto por parte do utilizador

4.5.3 Final Policy

Esta é a área final que apresenta ao utilizador a política gerada adequada às definições impostas na área anterior pronta a integrar no projeto que foi analisado. A figura 4.5 apresenta um exemplo de uma política gerada na interface do utilizador.

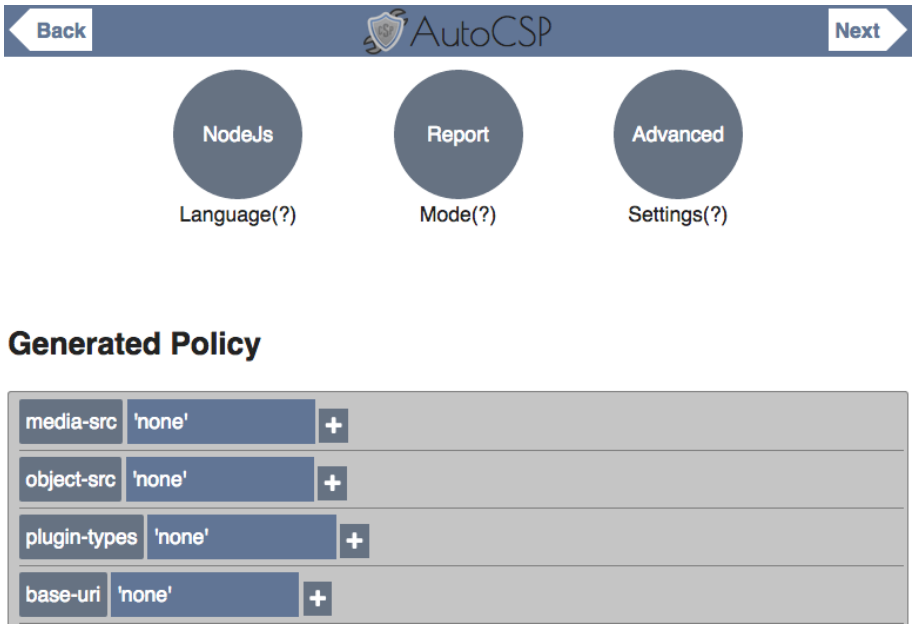

Figura 4.4: Área de interação para validação e edição dos recursos, formato e modo da política

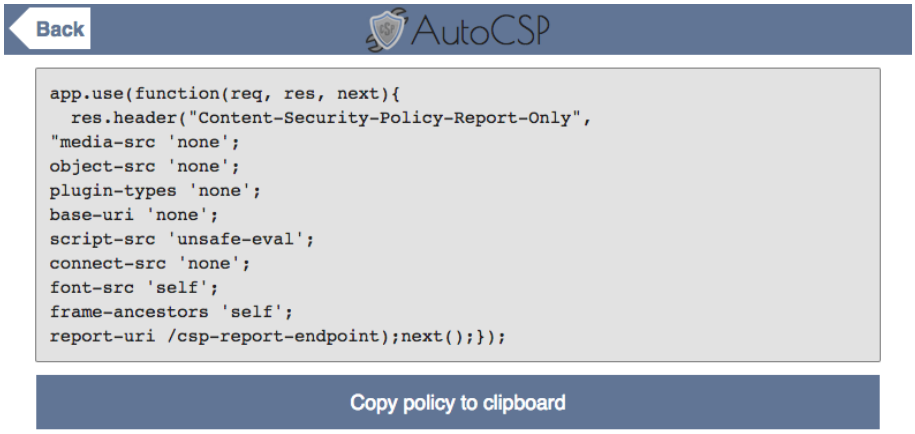

Figura 4.5: Área de interação com a política final de *CSP*

A interface foi desenvolvida para efeitos de teste da aplicação pelo que podem existir alguns problemas de usabilidade por exemplo quando o número de recursos detetados for muito elevado. O propósito desta era essencialmente uma prova do conceito para estabelecer contacto com a lógica da solução desenvolvida. Posteriormente a empresa pretende integrar a solução no seu principal produto pelo que nessa fase será pensada numa interface adaptada aos requisitos da empresa, estando assim fora do contexto do tema de dissertação

4.6 Interação com o servidor

Foi desenvolvida uma *API* que estabelece a ponte entre o cliente e a lógica do sistema e possibilita assim a geração das políticas adequadas à sua solução. Podem ser feitos três pedidos distintos à aplicação em duas fases da geração dos políticas que serão descritas nesta secção.

4.6.1 Input

Quando o utilizador acede à aplicação é esperado um de dois tipos de interação com o servidor, carregamento de um projeto que deve ser enviado num formato *zip* ou envio do *url*. Para ambos os tipos, é feito um *POST request* ao servidor que vai interpretar o pedido e encaminha-lo para o módulo já referido *Analysis Handler*.

Após todas as operações do lado do servidor é enviada resposta ao pedido efetuado pelo cliente que por sua vez mostra os recursos que foram captados da análise estática do código na página seguinte.

Um Diagrama de Sequência para o *upload* de um projeto representativo da interação acima descrita pode ser visto na figura 4.6

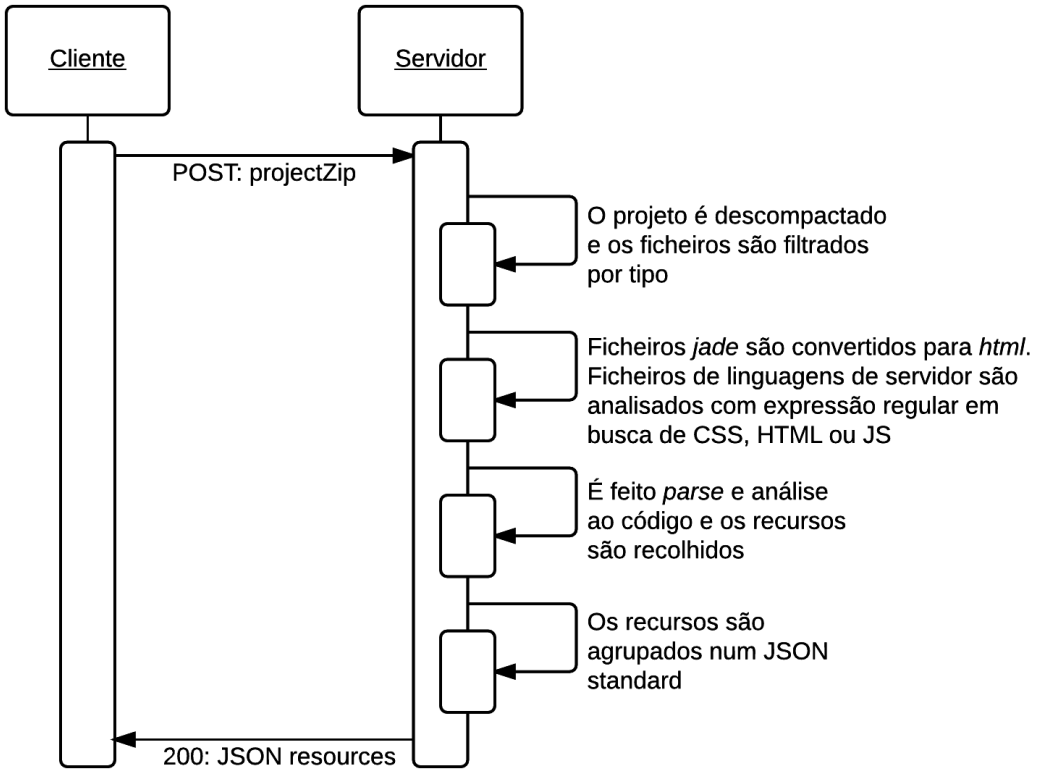

Figura 4.6: Diagrama de Sequência para o *upload* de um projeto

Nesta figura é possível de uma forma geral, ver como atuam os vários módulo da lógica do servidor para a obtenção dos recursos e geração de uma política standard. Semelhante a esta figura é a figura 4.7 que permite ver o processo de *crawl* a um *website*, é também notável que o processo de análise é mais simples uma vez que os tipos de ficheiros obtidos já estão limitados aos formatos *CSS, HTML e JavaScript*.

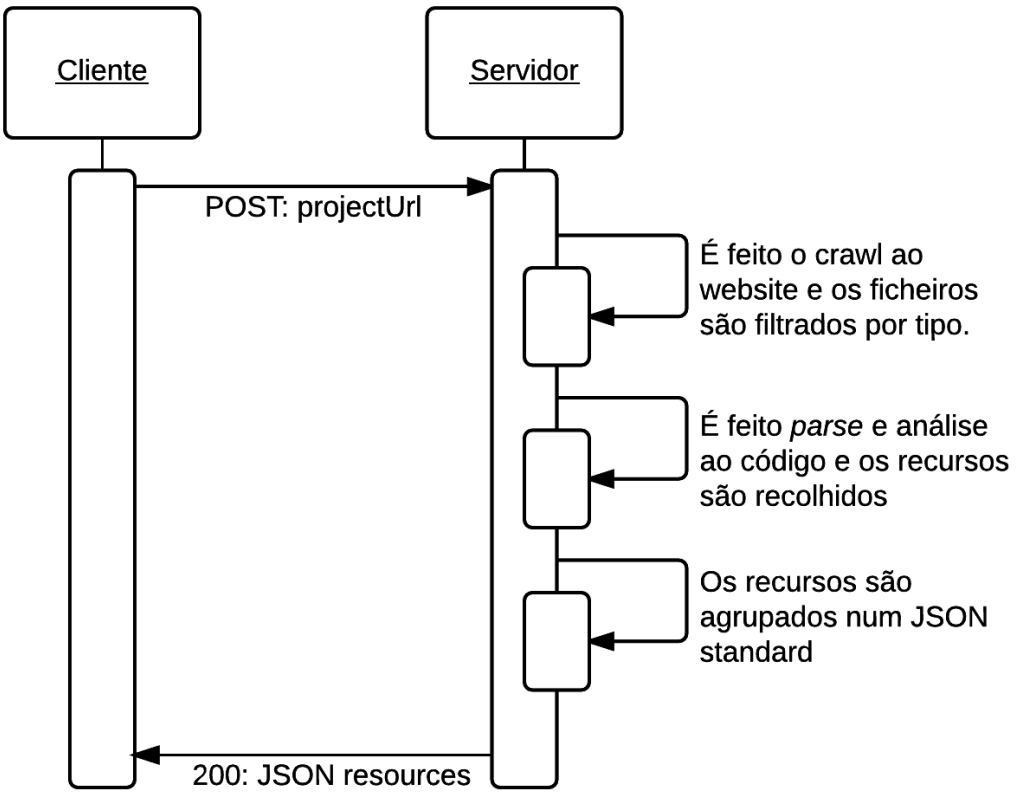

Figura 4.7: Diagrama de Sequência para o *crawl* de um *website*

4.6.2 Generated Policy

Nesta página é dada ao utilizador a possibilidade de adicionar, remover ou editar recursos. Para além disso pode ainda alterar a linguagem da política final a ser gerada e definir se pretende que a política esteja em modo normal ou de *report* e por fim submeter os dados que satisfaçam as suas necessidades. Nesta fase vai ser feito um novo pedido ao servidor, *POST request*, que vai receber um *JSON* e do lado do servidor vai traduzir os recursos numa política específica para a linguagem especificada e pronta a ser utilizada que vai ser mostrada na página seguinte que por sua vez não tem qualquer interação com o servidor. Esta interação pode ser vista no diagrama de sequência da figura 4.8

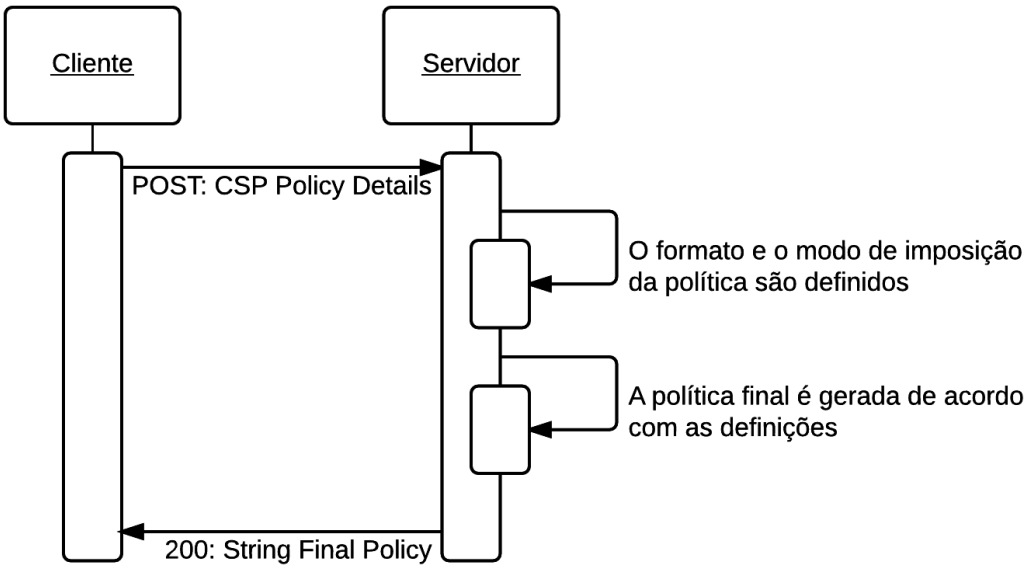

Figura 4.8: Diagrama de Sequência para a geração da política final

4.7 Principais Obstáculos

O desenvolvimento da solução passou por alguns obstáculos que não foram detetados a tempo de os evitar, desta forma é importante dar destaque aos mesmos.

4.7.1 Deteção de Recursos

A maior parte dos recursos carregados estaticamente são relativamente fáceis de detetar, o principal obstáculo surgiu na análise de código *JavaScript*, seja a deteção de *eval* seja a deteção de *connect* podem se tornar bastante complexas.

O *eval* em *JavaScript* pode ser feito de incontáveis formas diferentes, ao longo do desenvolvimento e da fase de testes, sempre que foi encontrada uma forma de *eval* esta foi adicionada à deteção de recursos. Para o *connect* a dificuldade estava em manter controlo do estado das variáveis que vão interferir nos pedidos feitos ao servidor. Dificuldade esta que pôde ser ultrapassada graças ao projeto *jscrambler-parser* que no *parser* de *JavaScript* passou incluir o estado de todas as variáveis e tornou assim possível a deteção dos pedidos ao servidor.

4.7.2 Suporte

A solução passou por duas fases distintas, e isto deve-se ao facto de numa primeira fase não se ter pensado nas limitações impostas pelas múltiplas linguagens e *frameworks*. Assim, após se tomar conhecimento desta falha foi estendida a funcionalidade da aplicação tendo sido acrescentada

a análise por *crawl* de um *website* e ainda a melhoria da análise por carregamento de um projeto que passou a suportar linguagens do lado do servidor e *frameworks*.

4.8 Tecnologias Utilizadas

Para o desenvolvimento da solução, foram seguidos os padrões de implementação da empresa em que se insere o tema. Desta forma, a solução é totalmente desenvolvida em *JavaScript*.

Para a lógica da solução foi utilizado *Node.js*, uma linguagem dinâmica de desenvolvimento *web* do lado do servidor. Para a parte de interação do utilizador com a aplicação foi utilizada a biblioteca *JavaScript* para criar interfaces criada pelo *Facebook*, *React* [42]. Por fim para desenvolvimento de testes unitários *Jest* [43] foi a ferramenta adotada, também criada pelo *Facebook* e foi desenvolvida apartir da *framework* de testes para *JavaScript*, *Jasmine* [44].

Para além destas tecnologias, foram utilizados, como já referido, módulos de terceiros para o *parse* do código, *css-parse* [39] para o *parse* do código *CSS*, *htmlparser2* [40] para o *parse* do código *HTML* e *esprima* [41] para o código *JavaScript*.

Por fim foi utilizado o módulo *simplecrawler* [45] que vai ser responsável pelo *crawl* dos *websites* através do método correspondente.

Capítulo 5

Validação e Resultados

A validação e análise de resultados de uma solução desenvolvida neste âmbito é uma das fases mais importantes pois não só permite identificar situações de erro que não são detetadas na fase de desenvolvimento como também permite avaliar o comportamento da solução em cenários reais de utilização.

5.1 Processo de Validação

Para validação da dissertação foram utilizados dois métodos de teste para as duas funcionalidades distintas da aplicação.

Para a análise puramente estática que resulta do upload de um projeto, foi feito o upload de alguns projetos desenvolvidos ao longo do percurso académico e encontrados na plataforma de controlo de versões *GitHub*. Os critérios de seleção para esta análise focavam principalmente a utilização das tecnologias suportadas e o tipo de análise desejado. Foram analisados três tipos de *websites* para testar três funcionalidades distintas, *websites* estáticos para o teste das funcionalidades básicas da solução, suporte para *CSS, HTML, JavaScript* sem qualquer recurso dinâmico. Para o efeito foram selecionadas 25 aplicações *web* listadas na sub secção B.1.1 do anexo B.

Foram também testados *websites* sob o mesmo critério dos anteriores mas que utilizassem a linguagem de *template Jade* a fim de avaliar o suporte para a mesma e, mais uma vez, a deteção de recursos sob estas condições. Foram também selecionadas 25 aplicações *web* nas plataformas já referidas que estão enumeradas na sub secção B.1.2 do anexo B.

Por fim, para este tipo de análise foram testados 25 *websites* completos que utilizassem *ASP.NET, PHP ou Node.js* como linguagens de servidor. O objetivo nestes testes era o teste da expressão regular definida para a deteção de recursos *HTML* em ficheiros do lado do servidor bem como os resultados em cenários do mundo real com poucas restrições a nível de tecnologias e práticas utilizadas.

Na sub secção B.1.3 do anexo B estão listados os *websites* utilizados para este teste.

Já para a análise resultante do *crawler* foram utilizados cenários reais em que foi feita a análise a páginas pequenas/médias por uma questão de simplicidade (dada a natureza do teste) e de seguida com recurso à extensão *Caspr: Enforcer* [29] aplicou-se a política gerada a fim de confirmar e validar a eficácia do *CSP* gerado para os recursos encontrados. Para esta análise foram selecionados 100 *websites*, tendo-se sempre atendido à dimensão dos mesmos sendo que a seleção foi feita a partir de uma página que agrupa aplicações *single-page*[46] que são aplicações cada vez mais recorrentes em que os recursos são geralmente carregados num só pedido ao servidor ou de forma dinâmica e para o utilizador é criada uma experiência de utilização mais fluída sendo que todo o conteúdo da aplicação, como o nome indica, está agrupado numa só página *web*.

Como já referido, a solução desenvolvida não permite a deteção de recursos carregados dinamicamente pelo que os resultados apresentados excluem os mesmos. Não foi possível contabilizar todos os recursos carregados pelo que a viabilidade da solução em casos reais é apenas verificável para a amostra testada.

Para se compreender melhor o comportamento da solução num cenário de utilização real foram ainda selecionadas duas aplicações *Web* entre os *websites* analisados a partir de um *URL*.

5.2 Resultados Obtidos

Foi feita a análise e recolhidos os resultados que serão apresentados ao longo desta secção com breves descrições de cada uma delas e apresentação de dados estatisticos das mesmas.

5.2.1 Análise por Carregamento de um Projeto

Nesta análise, a deteção de recursos esteve sempre dependente do suporte por parte da aplicação às tecnologias utilizadas. Assim, para efeitos de teste foram sempre utilizadas aplicações que apenas recorrem a tecnologias suportadas pela aplicação.

Foram feitos três tipos de testes para esta análise.

Websites estáticos

Nestes *websites* não existe qualquer dinamismo e a implementação é feita sem qualquer recurso a *frameworks* o que implica que todos os recursos estão declarados estaticamente e que as únicas linguagens utilizadas são *CSS, HTML e JavaScript* e portanto a deteção vai facilmente detetar todos os recursos carregados para a aplicação. Neste tipo de *websites* foram sempre detetados 100% dos recursos e assim, a aplicação da política não afetou de forma alguma a execução do mesmo.

Websites com linguagens de template

Para linguagens de *template* existem métodos de conversão das mesmas nas linguagens que serão convertidas para a respetiva linguagem base *CSS, HTML ou JavaScript*. Nestes testes foram utilizadas aplicações com níveis variáveis de dinamismo. Para 25 *websites* testados, foram sempre detetados 100% dos recursos estáticos.

Websites com páginas carregadas dinamicamente

Este é o teste menos eficaz e portanto é aquele de onde se esperavam piores resultados. É impossível estabelecer uma relação entre os projetos analisados pois a deteção depende apenas da forma como o programador declara os recursos. Caso as páginas geradas dinamicamente não dependam de estados ou os recursos não sejam declarados através de variáveis são detetados, em caso contrário não são.

Assim é possível afirmar que a taxa de deteção de recursos estáticos neste tipo de testes também foi 100%.

5.2.2 Análise por Crawl a um Website

Na deteção de recursos por *crawl* a um *website*, tendo-se a vantagem do tipo de páginas encontrado ser limitado, a deteção de recursos é, em relação ao método anterior, mais precisa, essencialmente por não ter qualquer dependência de linguagens ou *frameworks*, dado que as páginas geradas dinamicamente do lado do servidor já estão construídas não havendo qualquer dependência das mesmas. Neste tipo de análise a deteção para recursos estáticos foi sempre 100%.

5.2.3 Tempos de Execução

Para a análise da geração foram ainda medidos os tempos de execução da lógica desenvolvida para os dois principais métodos de análise.

Na análise por upload de um projeto a única dependência são os métodos de análise do código pelo que as variações dos tempos de resposta nunca foram muito grandes para as aplicações testadas.

Já para a análise do *crawl* a um *website* os tempos de execução dependiam de mais dois fatores associados ao módulo de *crawl simplecrawler*; o intervalo entre o *crawl* de cada uma das páginas, para evitar sobrecarregar a página e a concorrência que define o número máximo de pedidos efetuados em simultâneo. Para estes, os valores definidos foram 250 milissegundos e 5 pedidos em simultâneo respetivamente, valores recomendados pelo próprio autor do módulo.

Os tempos de execução para análise por *crawl* são referentes a cada uma das páginas analisadas. Os resultados para os *websites* analisados podem ser vistos na tabela 5.1.

Tipo	**Mínimo**	**Máximo**	**Média**	**Desvio Padrão**
Upload - Websites Estáticos	148	251	200.4	30.48
Upload - Linguagens de Template	152	253	208.5	31.55
Upload - Websites carregados dinamicamente	143	258	211.6	37.58
Crawl	1581.45	3808.35	2670	602

Tabela 5.1: Tempos de execução para a geração de uma política de *CSP* em milissegundos

Pelos resultados pode-se concluir naturalmente que a análise por carregamento de um projeto é mais rápida principalmente devido ao intervalo no *crawl* já referido anteriormente. A análise por *crawl* está portanto dependente do número de páginas que com uma concorrência de 5 terá

um aumento de 250 milissegundos por cada número igual de páginas analisadas. Outro fator determinante é o tempo de resposta do servidor para os pedidos efetuados que afeta também a performance deste tipo de análise.

Claro que a dimensão das páginas também é um fator determinante mas, dada a dimensão da amostra testada, é expectável a existência de ficheiros de dimensões muito variadas pelo que de certa forma os tempos já incluem este fator. Para a amostra recolhida foram analisadas as dimensões das páginas pelo número de recursos de cada tipo recolhidos. Obteve-se um mínimo de 1 recurso *HTML*, 0 recursos *CSS* e um recurso *JavaScript* numa página e um máximo de 201 recursos *HTML*, 417 recursos *CSS* e 1546 recursos *JS*. Sendo dados muito dispares, não é possível estabelecer qualquer tipo de relação entre eles.

5.2.4 Deteção atual no mundo real

A solução apresentada foi desenvolvida com o intuito de automatizar o processo de geração de políticas com uma abordagem diferente de qualquer uma das já existentes.

A análise de código é o principal ponto forte procurando desprender o utilizador da necessidade de navegar pela página a fim de detetar os recursos que vão sendo carregados. Este objetivo foi atingido, no entanto é evidente que os resultados obtidos estão dependentes dos cenários de utilização uma vez que não existe qualquer deteção de recursos carregados dinamicamente.

Tendo em conta estas limitações, foram então feitos os testes em duas aplicações escolhidas aleatoriamente da amostra previamente selecionada para *crawl* [1] [2]. O total de recursos foi contabilizado foi avaliado o nível de deteção. Todos os recursos estáticos foram encontrados. No primeiro caso[1], num total de 355 recursos existentes foi gerada uma política que cobre 118 recursos. Nesta aplicação verificou-se então um nível de deteção a rondar os 33%. Já para o segundo caso[2], para um total de 74 recursos existentes, a política final gerada pela solução implementada declara 49 recursos. Assim, nesta aplicação o nível de deteção sobe para os 66%.

Dada a disparidade dos resultados é impossível estabelecer uma correlação entre as aplicações testadas que estão sempre dependentes de práticas de programação na declaração de recursos (estáticos e dinâmicos).

No capítulo 6 serão discutidas possíveis soluções sob a forma de trabalho futuro para colmatar esta limitação.

5.3 Conclusões

Esta fase permitiu validar a funcionalidade da solução e concluir que no seu estado atual, faz uma deteção perfeita de recursos estáticos. Ao longo dos testes efetuados foram detetados e corrigidos alguns erros de implementação. Pode-se então afirmar que salvo algum erro de implementação que possa não ter sido identificado em antemão os resultados estão de acordo com o expectável.

[1] *https://www.jsconfar.com/*
[2] *http://www.neubible.co/*

Capítulo 6

Conclusões e Trabalho Futuro

6.1 Satisfação dos Objetivos

Para analisar a satisfação dos objetivos definidos ao longo do projeto é feita uma distinção entre investigação que representa a toda a pesquisa efetuada acerca da ferramenta e tecnologias relacionadas com a mesma e desenvolvimento que está associado ao desenvolvimento da solução de geração de *Content Security Policy*.

6.1.1 Investigação

A investigação efetuada seguiu todos os objetivos definidos e permitiu tomar conhecimento do estado atual das várias ferramentas e assim perceber o que seria possível fazer com as mesmas, tendo tudo isso em conta foi possível propor uma solução de acordo com a proposta inicial e de execução viável.

Esta fase estendeu-se para além do período especificado para o propósito por toda a fase de desenvolvimento, isto porque sendo esta uma área ainda em desenvolvimento e alvo de muita investigação foram surgindo vulnerabilidades e respetivas soluções e ainda avanços na geração de *CSP*.

Considera-se que os objetivos ao nível de investigação foram cumpridos e há uma perfeita compreensão do estado atual das ferramentas utilizadas, desde funcionalidades, limitações, estado de desenvolvimento e aceitação.

6.1.2 Desenvolvimento

O desenvolvimento da solução foi marcado pelas duas fases já referidas e consequentemente redefinição dos objetivos. Inicialmente a ideia de uma solução seria a análise estática feita a *CSS, HTML, JavaScript*, a estruturação e integração do projeto foram planeadas e executadas. Completa esta fase, foi feita uma retrospetiva que permitiu compreender as principais falhas e assim definir

novos objetivos para colmatar as mesmas. Objetivos estes foram cumpridos e assim chegou-se à solução final que corresponde às expectativas na análise de recursos carregados estaticamente.

6.2 Conclusões

A aplicação desenvolvida permitiu perceber o quão longe é possível chegar com a análise estática de código e geração das políticas da ferramenta com consequente aumento da segurança. Olhando ao trabalho produzido pode-se facilmente verificar os pontos fortes e pontos fracos face às atuais soluções.

Pontos Fortes

- Deteção de recursos sem a necessidade de intervenção do utilizador
- Suporte para várias linguagens do lado do servidor
- Análise de um *website* a partir de um *url* com um *crawler*

Pontos Fracos

- O dinamismo do *JavaScript* é o principal obstáculo na deteção de recursos
- A deteção por carregamento está limitada às linguagens suportadas no momento
- O *crawl* não oferece garantias quanto à cobertura e traz um grande aumento do tempo de execução

O *Content Security Policy* tem tido progressos significativos ao nível de funcionalidades e suporte o que mostra que esta é uma ferramenta que apesar de recente está a ter uma aceitação muito grande e parece ser uma grande força no combate ao *Cross-site Scripting*. Pôde-se também constatar que se começam a dar os primeiros passos na geração de políticas tendo surgido já uma ferramenta ainda que com algumas limitações.

6.3 Trabalho Futuro

Qualquer trabalho na solução apresentada terá foco evidente no aumento da capacidade de deteção por extensão da funcionalidade para captação de recursos dinâmicos. Para que tal seja possível podem foram detetadas três formas distintas de o fazer.

Expressões Regulares

Definir um conjunto de expressões regulares que procurem por *urls* em código que não seja analisado pelos já existentes métodos e tentem associar esses ao tipo adequado por exemplo pela análise do *header* do recurso ou pela extensão do mesmo.

Reverse Proxy

É um servidor que atua como um intermediário para comunicações feitas entre o servidor e o cliente, e permite assim captar todos os pedidos feitos ao servidor e consequentemente recolher os recursos da aplicação. A diferença para um *proxy* normal é que atua do lado do servidor ou seja, o cliente estabelece comunicação com o *reverse proxy* que por sua vez vai reencaminhar o pedido para o servidor, a ideia é que para o cliente pareça que está a comunicar com um mero servidor ao qual faz os pedidos e do qual obtém as respostas.

Com esta solução será possível gerar, enquanto o utilizador navega pela aplicação um política o mais restritiva possível com a condicionante que para que os recursos sejam captados é necessário que o utilizador navegue pela aplicação até que sejam recolhidos todos os recursos o que pode ser muito dispendioso.

Headless Browser

Ferramenta que permite automatizar a navegação num *website* monitorizar pedidos feitos ao servidor e desta forma também captar os recursos da aplicação (ex. PhantomJS)

Com este trabalho extra, a deteção de recursos permite criar um *CSP* altamente restritivo sem qualquer tipo de falhas e independente de qualquer dinamismo, linguagens ou *frameworks*.

Referências

[1] Alex Ivanovs. Protect Your Node.js Sources With JScrambler Before Unleashing it Into the Wild Wild Web | Alex Ivanovs, 2014. Last accessed on 2015-01-19. URL: http://www.huffingtonpost.com/alex-ivanovs/protect-your-nodejs-sourc_b_5780168.html.

[2] JScrambler. JScrambler | UPTEC, 2015. Last accessed on 2015-01-19. URL: http://uptec.up.pt/empresa/jscrambler.

[3] Brandon Sterne e Adam Barth. Content Security Policy 1.0, 2012. Last accessed on 2015-01-19. URL: http://www.w3.org/TR/2012/CR-CSP-20121115/.

[4] Types of Cross-Site Scripting - OWASP. Last accessed on 2015-02-15. URL: https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting.

[5] Joe Gibbs Politz, Spiridon Aristides Eliopoulos, Arjun Guha, e Shriram Krishnamurthi. Adsafety: Type-based verification of javascript sandboxing. Em *Proceedings of the 20th USENIX conference on Security*, páginas 12–12. Usenix Association, 2011.

[6] Google. google-caja - Compiler for making third-party HTML, CSS and JavaScript safe for embedding - Google Project Hosting, 2008. Last accessed on 2015-01-19. URL: https://code.google.com/p/google-caja/.

[7] Rob Knies. BrowserShield: Helping Make the Web Safe for Surfers - Microsoft Research, 2006. Last accessed on 2015-01-19. URL: http://research.microsoft.com/en-us/news/features/browsershield.aspx.

[8] W3Schools. HTML iframe sandbox Attribute, 2015. Last accessed on 2015-01-19. URL: http://www.w3schools.com/tags/att_iframe_sandbox.asp.

[9] Jeff Terrace, Stephen R Beard, e Naga Praveen Kumar Katta. Javascript in javascript (js. js): Sandboxing third-party scripts. *USENIX WebApps*, 2012.

[10] Metablocks. Metablocks | Yahoo!/Google Caja Javascript Sandbox - Social Media Apps for Brands + Music + Entertainment, 2009. Last accessed on 2015-01-19. URL: http://www.metablocks.com/2009/06/yahoogoogle-caja-javascript-sandbox/.

[11] Top 10 2013-A3-Cross-Site Scripting (XSS) - OWASP. URL: https://www.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_(XSS).

[12] CuteSoft Components Inc. Free Javascript Obfuscator - Protects JavaScript code from stealing and shrinks size, 2015. Last accessed on 2015-01-19. URL: http://javascriptobfuscator.com/.

[13] Gimpel Software LLC. The Leader in Static Analysis for C/C++ – PC-lint and FlexeLint, 2014. Last accessed on 2015-01-19. URL: http://www.gimpel.com/html/index.htm.

[14] Filipe Manuel Gomes Silva. Ferramenta de ofuscação de código javascript. Tese de mestrado, FEUP, July 2009.

[15] Google. Closure Tools — Google Developers, 2012. Last accessed on 2015-01-19. URL: https://developers.google.com/closure/utilities/.

[16] Douglas Crockford. JSLint,The JavaScript Code Quality Tool, 2014. Last accessed on 2015-01-19. URL: http://jslint.com/.

[17] Anton Kovalyov. JSHint, a JavaScript Code Quality Tool, 2013. Last accessed on 2015-01-19. URL: http://jshint.com/.

[18] Daft Logic. Online Javascript Obfuscator, 2009. Last accessed on 2015-01-19. URL: http://www.daftlogic.com/projects-online-javascript-obfuscator.htm.

[19] Google. Closure Compiler Service, 2009. Last accessed on 2015-01-19. URL: http://closure-compiler.appspot.com/home.

[20] Einar Lielmanis e Liam Newman. Online JavaScript beautifier, 2013. Last accessed on 2015-01-19. URL: http://jsbeautifier.org/.

[21] Dan. Javascript Obfuscate and Encoder. Last accessed on 2015-02-10. URL: http://www.jsobfuscate.com/index.php.

[22] Vimtech. JavaScript Obfuscator - Free Online JavaScript Packer, 2008. Last accessed on 2015-02-10. URL: http://packer.50x.eu/.

[23] Rukshan Ranatunge. Better Javascript Obfuscating Method To Protect Your Code | Techy Zilla, 2012. Last accessed on 2015-01-19. URL: http://techyzilla.blogspot.com/2012/09/better-javascript-obfuscating-method-to-protect-your-code.html.

[24] Mike West. An Introduction to Content Security Policy - HTML5 Rocks, 2012. Last accessed on 2015-01-19. URL: http://www.html5rocks.com/en/tutorials/security/content-security-policy/.

[25] Mike West, Adam Barth, Dan Veditz, e Brandon Sterne. Content Security Policy Level 2. URL: http://www.w3.org/TR/CSP2/.

[26] Nicolas Golubovic. *autoCSP-CSP-injecting reverse HTTP proxy*. Tese de doutoramento, BSc thesis, Rhur Bochum University-Google, 2013.

[27] Tom Postma. PHP Content Security Policy generator: Generate CSP headers to prevent security attacks - PHP Classes, 2014. Last accessed on 2015-01-19. URL: http://www.phpclasses.org/package/8919-PHP-Generate-CSP-headers-to-prevent-security-attacks.html.

[28] Mal Curtis. Content Security Policy Header Generator, 2012. Last accessed on 2015-01-19. URL: http://cspisawesome.com/.

[29] C0nrad. Generating Content-Security-Policies, the easy way, 2014. Last accessed on 2015-01-19. URL: http://c0nrad.io/blog/csp.html.

[30] Paweł Krawczyk. Content Security Policy Builder. Last accessed on 2015-01-19. URL: https://cspbuilder.info/static/#/main/.

[31] David Risney. CSP-Fiddler-Extension. Last accessed on 2015-03-30. URL: https://github.com/david-risney/CSP-Fiddler-Extension.

[32] Mark Percival. Improving Browser Security with CSP, 2011. Last accessed on 2015-01-19. URL: https://blog.twitter.com/2011/improving-browser-security-csp.

[33] Neil Matatall. CSP to the Rescue: Leveraging the Browser for Security. Last accessed on 2015-01-19. URL: https://blog.twitter.com/2013/csp-to-the-rescue-leveraging-the-browser-for-security.

[34] Google. Content Security Policy (CSP) - Google Chrome. Last accessed on 2015-01-19. URL: https://developer.chrome.com/extensions/contentSecurityPolicy.

[35] Joshua Peek. Content Security Policy. Last accessed on 2015-04-05. URL: https://github.com/blog/1477-content-security-policy.

[36] Usage Statistics and Market Share of Server-side Programming Languages for Websites, June 2015. Last accessed on 2015-06-19. URL: http://w3techs.com/technologies/overview/programming_language/all.

[37] Usage Statistics and Market Share of Web Servers for Websites, June 2015. Last accessed on 2015-06-19. URL: http://w3techs.com/technologies/overview/web_server/all.

[38] Jade - Template Engine. Last accessed on 2015-06-19. URL: http://jade-lang.com/.

[39] Necolas. css-parser, 2014. Last accessed on 2015-06-15. URL: https://github.com/reworkcss/css-parse.

[40] Felix Böhm. Htmlparser2, 2015. Last accessed on 2015-06-15. URL: https://github.com/fb55/htmlparser2.

[41] Ariya Hidayat. Esprima. Last accessed on 2015-06-23. URL: http://esprima.org/.

[42] A JavaScript library for building user interfaces | React. Last accessed on 2015-06-15. URL: http://facebook.github.io/react/.

[43] Jest | Painless JavaScript Unit Testing. Last accessed on 2015-06-15. URL: https://facebook.github.io/jest/.

[44] Gregg Van Hove. Jasmine: Behavior-Driven JavaScript. URL: http://jasmine.github.io/.

[45] Christopher Giffard. node-simplecrawler, 2015. URL: https://github.com/cgiffard/node-simplecrawler.

[46] One Page Love. URL: https://onepagelove.com/.

REFERÊNCIAS

Anexo A

Estruturas Standard

Entre as camadas desenvolvidas foram definidas várias estruturas standard para os *outputs* das diferentes fases da deteção de recursos e geração da política. Este anexo apresenta os standards e formatos para os principais outputs das camadas.

A.1 Análise de Código

Da análise de código resulta um *JSON* que agrupa os recursos encontrados em todo o *CSS, HTML e JavaScript*.

```
{
    "HTML":[
        {
            "type": "included-script",
            "source-list": [
                "cenas/file.js",
                "http://code.jquery.com/jquery-latest.min.js"
            ]
        },
        {
            "type": "image",
            "source-list": [
                "cenas/file.jpg",
                "http://images.google.com/image.png"
            ]
        },
        {
            "type": "media",
            "source-list": [
                {
                    "audio": [
                        "link1",
                        "link2",
```

```
                        "link3",
                        "link4"
                    ]
                },
                {
                    "video": [
                        "links"
                    ]
                },
                {
                    "audio": [
                        "links"
                    ]
                }
            ]
        },
        {
            "type": "style",
            "source-list": [
                "cenas/file.css",
                "http://fonts.googleapis.com/css?family=Open+Sans"
            ]
        },
        {
            "type": "frame",
            "source-list": [
                "links"
            ]
        },
        {
            "type": "form-action",
            "source-list": [
                "links"
            ]
        },
        {
            "type": "inline-script",
            "hash": [
                "bWFT86eMjA+y4Rsn3d3JxAdziMGCHwp/jzuc2OPilEo="
            ]
        }

    ],
    "JavaScript":[
        {
            "type": "eval"
        },
        {
            "type": "connect",
```

```
            "source-list": [
                "cenas/file.js",
                "http://code.jquery.com/jquery-latest.min.js"
            ]
        }
    ],
    "CSS":[
        {
            "type": "image",
            "source-list": [
                "links"
            ]
        },
        {
            "type": "font",
            "source-list":[
                "links"
            ]
        }
    ]
}
```

Listing A.1: JSON Standard resultante da Análise de Código

A.2 Standard CSP

Esta camada cria uma estrutura *JSON* que contém os recursos já associados às respetivas diretivas sem distinção da sua origem (*CSS, HTML, JavaScript*).

```
{
    "Server-Language": "language or server",
    "Type of Policy":[ //Content-Security-Policy or Content-Security-Policy-Report-
        Only
        {
            "directive": "name",
            "source-list": [
                "list of sources",
                "you can have more than one"
            ]
        },
        {
            "directive": "script-src",
            "source-list": [
                "'self'",
                "'nonce-Nc3n83cnSAd3wc3Sasdfn939hc3'",
                "http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/"
            ]
```

```
18          }
19      ]
20  }
```

Listing A.2: JSON para um formato standard de CSP

A.3 CSP Final

A política final é gerada conforme os requisitos definidos pelo utilizador em que pode ser definida uma linguagem e um modo. Apresentam-se de seguida figuras para a notação de cada uma das linguagens. A figura A.4 apresenta o modo de imposição da política em *Report-Only*.

```
1  Response.AddHeader("Content-Security-Policy",
2      "media-src 'none';
3      object-src 'none';
4      plugin-types 'none';
5      base-uri 'none';
6      form-action 'none';
7      child-src 'none';
8      img-src 'self';
9      script-src 'self'; 'unsafe-eval';
10     style-src 'self';
11     connect-src 'none';
12     font-src 'none';
13     frame-ancestors 'self';
14     report-uri /csp-report-endpoint");
```

Listing A.3: CSP Final para a linguagem ASP.NET

```
1  app.use(function(req, res, next) {
2    res.header("Content-Security-Policy-Report-Only",
3      "media-src 'none';
4      object-src 'none';
5      plugin-types 'none';
6      base-uri 'none';
7      form-action 'none';
8      child-src 'none';
9      img-src 'self';
10     script-src 'self'; 'unsafe-eval';
11     style-src 'self';
12     connect-src 'none';
13     font-src 'none';
14     frame-ancestors 'self';
15     report-uri /csp-report-endpoint");
16   next();
```

```
17 });
```

Listing A.4: CSP Final para a linguagem Node.js

```
1  header("Content-Security-Policy,
2    media-src 'none';
3    object-src 'none';
4    plugin-types 'none';
5    base-uri 'none';
6    form-action 'none';
7    child-src 'none';
8    img-src 'self';
9    script-src 'self'; 'unsafe-eval';
10   style-src 'self';
11   connect-src 'none';
12   font-src 'none';
13   frame-ancestors 'self';
14   report-uri /csp-report-endpoint");
```

Listing A.5: CSP Final para a linguagem PHP

```
1  Header set Content-Security-Policy
2    "media-src 'none';
3    object-src 'none';
4    plugin-types 'none';
5    base-uri 'none';
6    form-action 'none';
7    child-src 'none';
8    img-src 'self';
9    script-src 'self'; 'unsafe-eval';
10   style-src 'self';
11   connect-src 'none';
12   font-src 'none';
13   frame-ancestors 'self';
14   report-uri /csp-report-endpoint";
```

Listing A.6: CSP Final para o webserver Apache

```
1  add_header Content-Security-Policy
2    "media-src 'none';
3    object-src 'none';
4    plugin-types 'none';
5    base-uri 'none';
6    form-action 'none';
7    child-src 'none';
```

```
8   img-src 'self';
9   script-src 'self'; 'unsafe-eval';
10  style-src 'self';
11  connect-src 'none';
12  font-src 'none';
13  frame-ancestors 'self';
14  report-uri /csp-report-endpoint";
```

Listing A.7: CSP Final para o webserver Nginx

Anexo B

Amostra para Testes e Validação

Foram selecionados 175 aplicações *web* para validar o a solução desenvolvida.

Para a análise por carregamento de um projeto foram utilizados 75 dos 175 estando definido 25 para cada tipo específico de análise. *Websites* estáticos, *websites* com linguagens de template e *websites* com páginas carregadas dinamicamente. Dada a consistência dos resultados obtidos com a dimensão da amostra (100% de deteção de recursos estáticos) entendeu-se não haver ganhos com a sua extensão.

B.1 Análise por carregamento de um projeto

B.1.1 Websites Estáticos

1. **HTML - 4 Aplicações Web**
   https://github.com/TelerikAcademy/HTML
2. **HTML**
   https://github.com/mrmrs/html
3. **Really Simple Responsive HTML Email Template**
   https://github.com/leemunroe/responsive-html-email-template
4. **html**
   https://github.com/ubuntufag/html
5. **HTML5 Demos and Examples - 18 Aplicações Web**
   https://github.com/remy/html5demos

B.1.2 Websites com linguagens de template *jade*

1. **Node Kickstart**
   https://github.com/sbstjn/node-kickstart
2. **Jade Wireframes**
   https://github.com/kramapa/jade_wf
3. **Ejemplos Jade**
   https://github.com/wilsson/jade-examples
4. **Jade Examples**
   https://github.com/cancerhermit/jade.examples
5. **JadeStrap**
   https://github.com/chilts/jadestrap
6. **Client Server Jade**
   https://github.com/kulor/client-server-jade
7. **Jade Example**
   https://github.com/crguezl/jadeexample
8. **Jade-Example**
   https://github.com/balzohrd/jade-example
9. **Microscope Jade**
   https://github.com/alavers/microscope-jade
10. **jade-example-web**
    https://github.com/phamkhactuy/jade-example-web
11. **Static Jade Site Example**
    https://github.com/snlacks/static_jade_node_express_example
12. **Sea Js**
    https://github.com/Mooxe000/SeajsDemo
13. **belt-jade-handlebars-example**
    https://github.com/meteorbelt/belt-jade-handlebars-example
14. **gulp-jade-test**
    https://github.com/otmjka/gulp-jade-test
15. **hatch-jade-example**
    https://github.com/coderena/hatch-jade-example
16. **Basic Express example with Jade**
    https://github.com/MBing/express-jade-example

17. **Jade-Examples**
    https://github.com/maniacneron/Jade-Examples

18. **NodeChatWithJade**
    https://github.com/jcadruvi/NodeChatWithJade

19. **Express and Jade example**
    https://github.com/gastongarcia/express-jade-example

20. **jade**
    https://github.com/grunzwei-fresh/jade

21. **jade-examples**
    https://github.com/jansanchez/jade-examples

22. **PHP Jade**
    https://github.com/sbstjn/php-jade

23. **Sharing Jade Templates between an Express and Backbone Application**
    https://github.com/kwhinnery/template-example

24. **Studdle - LDSO 2014/2015**
    Projeto desenvolvido no âmbito da unidade curricular de Laboratório de Desenvolvimento de Software em Node.js e Jade

25. **CSP examples - PDIS 2014/2015**
    Projeto desenvolvido no âmbito da unidade curricular de Preparação para a Dissertação em Node.js e Jade

B.1.3 Websites com páginas carregadas dinamicamente

ASP.NET

1. **kendo-examples-asp-net-mvc**
   https://github.com/telerik/kendo-examples-asp-net-mvc

2. **kendo-examples-asp-net**
   https://github.com/telerik/kendo-examples-asp-net

3. **ASP.NET MVC3 Invoicing Application**
   https://github.com/iloire/ASP.NET-MVC-ACME-Invoicing--App

4. **Aspose.Words for .NET**
   https://github.com/asposewords/Aspose_Words_NET

5. **ASP.NET Role-Based Security Example**
   https://github.com/TypecastException/AspNetRoleBasedSecurityExample
6. **GuestbookDemo**
   https://github.com/SteveSanderson/GuestbookDemo
7. **ASPWebAPIExample**
   https://github.com/johnvpetersen/ASPWebAPIExample
8. **Getting Started**
   https://github.com/bradoyler/WePayASPNet
9. **ASPNETMVC5Bootstrap3ModalExamples**
   https://github.com/klabranche/ASPNETMVC5Bootstrap3ModalExamples

PHP

1. **Example PHP project**
   https://github.com/travis-ci-examples/php
2. **php**
   https://github.com/larryrubin/php
3. **PHP**
   https://github.com/zmwebdev/php-examples
4. **iContact API PHP Examples**
   https://github.com/icontact/icontact-api-php
5. **Resources for PHP**
   https://github.com/phpmentoring/resources-php
6. **MailChimp API v2.0 PHP Example Application**
   https://github.com/mailchimp/mcapi2-php-examples
7. **Plivo PHP Examples**
   https://github.com/plivo/plivo-examples-php
8. **PhpMQTTClient**
   https://github.com/tokudu/PhpMQTTClient
9. **Facebook/Heroku sample app – PHP**
   https://github.com/heroku/facebook-template-php
10. **The Word - News Social Network - LBAW 2013/2014**
    Projeto desenvolvido no âmbito da unidade curricular de Laboratório de Bases de Dados e Aplicações Web em PHP e HTML5

Node.js

1. **nodejs**
   https://github.com/narayand4/nodejs
2. **Code Examples for Sams Teach Yourself Node.js in 24 Hours**
   https://github.com/shapeshed/nodejsbook.io.examples
3. **NodeJS Examples**
   https://github.com/hicapacity/nodejs-examples
4. **node_demos**
   https://github.com/rlr/node_demos
5. **MailChimp API v2.0 NodeJs Example Application**
   https://github.com/mailchimp/mcapi2-node-examples
6. **NodeJS module examples**
   https://github.com/sw360cab/nodejs-module-examples

B.2 Análise por *crawl* de um *website*

1. **yay** - 14 HTML, 1 CSS, 2 JS
   http://www.collectif-yay.com/
2. **Sakura - Ya** - n/a
   http://sakura-ya.org/
3. **Sounds of Mumbai** - 2 HTML, 10 CSS, 18 JS
   http://www.soundsofmumbai.in/
4. **Work & Co** 5 HTML, 1 CSS, 10 JS
   http://work.co/
5. **FS Millbank** 1 HTML, 1 CSS, 3 JS
   http://www.fsmillbank.com/#/fs-millbank
6. **Samuel Reed** 1 HTML, 1 CSS, 1 JS
   http://strml.net/
7. **Brancott Estate Wine Explorer** 3 HTML, 2 CSS, 5 JS
   http://wineexplorer.brancottestate.com/
8. **Beagle** 3 HTML, 5 CSS, 7 JS
   https://getbeagle.co/

9. **NeuBible** 1 HTML, 4 CSS, 3 JS
   http://www.neubible.co/

10. **One John St** 2 HTML, 2 CSS, 7 JS
    http://onejohnst.com/

11. **Melanie Daveid** 4 HTML, 2 CSS, 4 JS
    http://melaniedaveid.com/

12. **Bia Costa** 1 HTML, 1 CSS, 3 JS
    http://biacosta.com/

13. **Tapwater** 2 HTML, 4 CSS, 6 JS
    http://www.tapwater.co/

14. **Jlenia Dog Sitter Roma** n/a
    http://www.jleniadogsitterroma.it/

15. **Google Ventures: Year in Review 2014** 668 HTML, 8 CSS, 1336 JS
    http://www.gv.com/2014/

16. **Smartphones... Dumb Users** n/a
    http://www.mobiles.co.uk/smart-phone-dumb-users/

17. **Schoolrunner** 1 HTML, 0 CSS, 1 JS
    http://schoolrunner.org/

18. **JSConf** 1 HTML, 1 CSS, 4 JS
    https://www.jsconfar.com/

19. **Google Calendar** n/a
    http://www.google.com/calendar/about/

20. **25 years of the Internet** n/a
    http://onyx.net/25-years-of-the-internet

21. **Send Message To** 1 HTML, 2 CSS, 3 JS
    http://sendamessage.to/

22. **Beard Design Labs** 201 HTML, 417 CSS, 1546 JS
    http://bearddesign.co/labs/

23. **Bakken & Bæck** 2 HTML, 1 CSS, 4 JS
    http://bakkenbaeck.com/

24. **Movement of Data** n/a
    http://akita.co.uk/movement-of-data/

25. **TETHR** n/a
    http://www.invisionapp.com/tethr

26. **Hunger Crunch**
    http://www.hungercrunch.com/

27. **Degordian Academy**
    http://academy.degordian.com/

28. **New Jumo Concept**
    http://www.newjumoconcept.com/

29. **I Remember**
    http://i-remember.fr/

30. **Garden Estúdio**
    http://gardenestudio.com.br/

31. **WILD**
    http://wild.as/

32. **Diamonds in the Sky**
    http://www.77diamonds.com/diamonds-in-the-sky/

33. **The Colors of Motion**
    http://thecolorsofmotion.com/films

34. **Interior Design by Decade**
    http://www.harveywatersofteners.co.uk/history-interior-design/

35. **Mercedes Benz Premium**
    http://mbpremium.nl/

36. **Epic Exit**
    http://www.epicexit.be/

37. **Visionare**
    http://www.bevisionare.com/

38. **FLTR**
    http://fltr.io/

39. **CUPS Annual Report**
    http://www.cupsannual.ca/

40. **World of SWISS**
    http://www.world-of-swiss.com/

41. **Astronomyy.fm**
    http://www.astronomyy.fm/

42. **Dangerous Robot**
    http://dangerousrobot.com/

43. **Happy 25th Birthday Game Boy**
    https://ihatetomatoes.net/happy-25th-birthday-game-boy/

44. **World Food Clock**
    http://worldfoodclock.com/

45. **Pete Nottage**
    http://www.petenottage.co.uk/

46. **Mapping Place Pins**
    http://creative.pinterest.com/features/place-pins/

47. **Highway One Road Trip**
    http://www.exsus.com/highway-one-roadtrip

48. **Vtcreative**
    http://www.vtcreative.fr/

49. **Heikopaiko**
    http://www.heikopaiko.com/

50. **Terminal Wedding**
    http://wedding.jai.im/

51. **Smoking Lung**
    https://www.health-on-line.co.uk/smoking-lung/

52. **Make Your Money Matter**
    http://makeyourmoneymatter.org/

53. **Your Brain Map**
    http://www.opencolleges.edu.au/informed/learning-strategies/

54. **Robby Leonardi**
    http://www.rleonardi.com/interactive-resume/

55. **Wrist**
    http://www.wrist.im/

56. **Museum of Mario**
    http://mario.ign.com/

57. **Alchemy**
    http://www.alchemy-digital.co.uk/

58. **Angry Bear**
    http://www.fleeangrybear.com/

59. **How Much Does It Cost To Make An App?**
    http://howmuchtomakeanapp.com/estimator

60. **Kocha**
    http://www.kocha.com.au/

61. **KITKAT**
    https://www.kitkat.com/android/

62. **Digitized 2013**
    http://www.digitized.gr/

63. **So You Want To Go To RISD**
    http://soyouwanttogotorisd.com/

64. **Ordinary Day**
    http://ordinaryday.co.za/

65. **De Vriend (Friend) E-book Trailer**
    http://devriend.submarinechannel.com/en/

66. **Play-Dot-To.com**
    http://play-dot-to.com/

67. **FlatGuitars**
    http://www.flatguitars.com/

68. **Kick My Habits**
    http://www.leedsbuildingsociety.co.uk/resources/kick-my-habits/

69. **Sam Markiewicz**
    http://sammarkiewi.cz/

70. **Alexey Bokov Studio**
    http://bokovdesign.ru/

71. **Justin Aguilar**
    http://justinaguilar.tumblr.com/

72. **The Interactive Ear**
    http://www.amplifon.co.uk/interactive-ear/index.html

73. **Shadow**
    http://discovershadow.com/

74. **iStrategyLabs Portfolio Review**
    http://islreview.com/

75. **Pi's Epic Journey**
    http://journey.lifeofpimovie.com/

76. **Adam Hartwig**
    http://www.adamhartwig.co.uk/

77. **Meet The Pros 2013**
    http://www.meetthepros.org/2015/

78. **WNDRMNT**
    http://www.wndrmnt.com/

79. **Andrew McCarthy**
    http://andrevv.com/

80. **My Own Corks**
    http://www.myowncorks.com/

81. **Oakley Airbrake MX**
    http://moto.oakley.com/

82. **Minimal Monkey**
    http://minimalmonkey.com/

83. **Every Last Drop**
    http://everylastdrop.co.uk/

84. **Ian James Cox**
    http://www.supremo.tv/

85. **Joyn DayCare**
    http://joyndaycare.com/

86. **Pudding Rush**
    http://www.puddingrush.com/

87. **Diografic**
    http://www.diografic.com/

88. **3D CSS Hartwig chess set**
    http://codepen.io/juliangarnier/full/BsIih

89. **Everpix** 3 HTML, 2 CSS, 4 JS
    http://www.everpix.com/

90. **State of Financial & Economic Education in US**
    http://surveyofthestates.com/

91. **Last Pants Standing**
    http://www.duluthtrading.com/store/mens/mens-pants/work-pants-for-men/work-pants-for-men.aspx

92. **Alder Cass**
    http://www.aldercass.com/

93. **Always Creative**
    http://alwayscreative.net/

94. **Intracto – Land a job**
    http://www.intracto.com/nl/jobs

95. **Characters mac app**
    http://getcharacters.com/

96. **Progetty Studio Arcade Version**
    http://progettystudio.com/

97. **Gregory Sujkowski**
    http://gregory.sujkowski.fr/

98. **Theory**
    http://madebytheory.com/

99. **Inception Explained**
    http://www.inception-explained.com/

100. **We believe in...**
    http://www.soleilnoir.net/believein/

Anexo C

Content Security Policy

Uma política de CSP pode portanto ter dois modos de imposição e neste momento tem definidas 15 principais diretivas. Para além disto existe ainda um conjunto de palavras chave que podem ser utilizadas nas diretivas.

C.1 Modo de Imposição

- **Content-Security-Policy** - Modo normal em que a execução de recursos não declarados é bloqueada.
- **Content-Security-Policy-Report-Only** - Modo *report* em que a execução de recursos não declarados é permitida sendo apenas gerados os relatórios de erro.

C.2 Diretivas

- **base-uri** - Indica os *url's* que podem ser utilizados na *tag HTML base* que por sua vez define os *url's* base a partir dos quais serão carregados os recursos com caminhos relativos.
- **child-src** - Indica os caminhos a partir dos quais podem ser carregados documentos para as *tags html frame e iframe*.
- **connect-src** - Indica os caminhos que podem carregar recursos através de *XHR, WebSocket ou EventSource*.
- **default-src** - Define um conjunto de restrições para um conjunto pré definido de diretivas. *child-src, connect-src, font-src, img-src, media-src, object-src, script-src, style-src*.
- **font-src** - Restringe as fontes que podem ser carregadas para a aplicação às definidas nesta diretiva.

- **form-action** - Limita os recursos que podem ser utilizados como *action* nos elementos *form* de *HTML*.
- **frame-ancestors** - Limita os *websites* que podem carregar a aplicação alvo em *frames e iframes*.
- **frame-src** - Diretiva deprecada englobada pela diretiva *child-src*
- **image-src** - Define o conjunto de caminhos de imagens que podem ser carregadas para a aplicação.
- **media-src** - Define o conjunto de caminhos de audios ou videos que podem ser carregadas para a aplicação.
- **object-src** - Define o conjunto de recursos que podem ser carregados para a aplicação através das *tags HTML object, embed ou applet*.
- **plugin-types** - Define os tipos de recursos que podem ser carregados para a aplicação através das *tags HTML object, embed ou applet*.
- **script-src** - Define scripts que podem ser carregados para a aplicação.
- **style-src** - Define estilos que podem ser carregados para a aplicação.
- **report-uri** - Define um *endpoint* para os relatórios de erro gerados na violação das políticas de *CSP*

C.3 Palavras Chave

- ***** - Permite a execução de qualquer recurso para a diretiva em questão.
- **'none'** - Impede a execução de todos os recursos para a diretiva em questão.
- **'self'** - Apenas permite a execução de recursos que estejam na mesma origem da aplicação.
- **'unsafe-inline'** - Permite a execução de *scripts* ou *styles inline*.
- **'unsafe-eval'** - Permite a execução de *eval*.