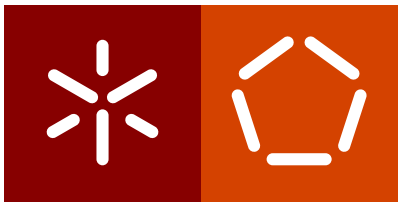
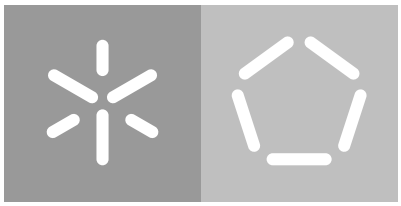


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Paulo Filipe Silva Ribeiro

Aprendizagem automática aplicada à deteção de vulnerabilidades

Outubro de 2021



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Paulo Filipe Silva Ribeiro

Aprendizagem automática aplicada à deteção de vulnerabilidades

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Orientador
Rui Manuel Ribeiro Castro Mendes
Supervisores
Igor Terroso, David de Paula Santos Silva

Outubro de 2021

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Paulo Filipe Silva Ribeiro

_____

AGRADECIMENTOS

Na realização da presente dissertação, recebi a ajuda de várias pessoas às quais pretendo expressar a minha gratidão.

Em primeiro lugar, não posso deixar de agradecer ao meu orientador, Professor Rui Manuel Ribeiro Castro Mendes por toda a ajuda prestada ao longo das diversas fases de desenvolvimento apresentadas neste documento.

Agradeço também à empresa Celfocus, nomeadamente ao Igor Terroso, David Santos Silva, Pedro Tarrinho, Luís Carlos Santos e José Morgado, inicialmente pelas sugestões durante a fase de investigação do estado de arte, pelos recursos disponibilizados quando necessários e por todo o acompanhamento e motivação que me deram ao longo do desenvolvimento deste trabalho.

Aos meus colegas do Mestrado integrado em Engenharia Informática que quando solicitava a sua ajuda, sempre se mostravam disponíveis para solucionar os eventuais problemas que surgiam.

À minha família por todo o apoio, incentivo e paciência demonstrada em todos os momentos da elaboração da dissertação.

Por fim, quero agradecer a todos os que tornaram possível a realização desta dissertação. Obrigado.

ABSTRACT

The HTTP protocol is a stateless protocol, that means, each request made by the user is an independent request, there is no notion of state. So, to add it to the applications we need an additional tool to implement this notion of state. For this, cookies are used, allowing the websites to identify the authenticated users. A cookie is a file stored in the customer's browser and sent together with HTTP requests, allowing the website to recognize the customer and send a response corresponding to the request made.

This dissertation aims to strengthen the protection of data associated with authentication sessions through the identification and analysis of authentication cookies using machine learning techniques. If web applications are vulnerable to malicious attacks, such as Broken Authentication or XSS (Cross-Site Scripting), attackers can gain access to the information stored in the cookie. Using this information they can steal the user's session, being able to authenticate themselves in the web application to obtain access to data/services.

Using machine learning techniques, we can identify within a set composed of several types of cookies, which cookies are associated with authentication. The objective is the recognition of this type of cookies, since this is the one that needs greater security, taking care in case the attacker even gaining access to this file, there is no possibility of deciphering the information that puts the users session at risk.

In addition to the classification of cookies, the detection and analysis of the encoding used will be carried out. The tool will then be integrated into the security testing software, Burp Suite, working as an extension in order to facilitate and reduce the time necessary for a QA analyst to spend checking cookies.

Keywords: Authentication, Burp Suite, Cookie Detection, Machine Learning, Security

RESUMO

O protocolo *HTTP* é um protocolo *stateless*, ou seja, cada pedido efetuado pelo utilizador é um pedido independente, não existe uma noção de estado. Logo para adicionarmos isto nas aplicações *web* precisamos de uma ferramenta adicional para implementar esta noção de estado. Para tal, são utilizados os *cookies*, permitindo aos *websites* identificar os utilizadores autenticados. O *cookie* é um ficheiro armazenado no navegador do cliente e que é enviado juntamente com os pedidos *HTTP*, permitindo ao *website* reconhecer o cliente e enviar a resposta correspondente ao pedido efetuado.

Esta dissertação tem como objetivo reforçar a proteção dos dados associados às sessões de autenticação através da identificação e análise dos *cookies* de autenticação recorrendo a técnicas de *machine learning*. Caso as aplicações *web* estejam vulneráveis a ataques maliciosos, como por exemplo *Broken Authentication* ou *XSS(Cross-Site Scripting)*, os atacantes podem conseguir acesso à informação armazenada no *cookie*. Utilizando esta informação podem roubar a sessão do utilizador, conseguindo autenticar-se na aplicação *web* para obter acesso a dados/serviços.

Recorrendo a técnicas de *machine learning* podemos identificar dentro de um conjunto composto por vários tipos de *cookies*, quais os *cookies* associados à autenticação. O objetivo é o reconhecimento deste tipo de *cookies*, dado que este é o que necessita de uma maior segurança, precavendo-se para o caso do atacante mesmo conseguindo acesso a este ficheiro, não haja a possibilidade de decifrar a informação que coloca em risco a sessão dos utilizadores.

Para além da classificação de cookies, será realizada a deteção e análise do *encoding* utilizado. A ferramenta será depois integrada no *software* de realização de testes de segurança, *Burp Suite*, funcionando como uma extensão do mesmo, de forma a facilitar e reduzir o tempo necessário que um analista de *QA(Quality Assurance)* irá dispender com a verificação dos *cookies*.

Palavras-Chave: Autenticação, Burp Suite, Deteção de Cookies, Machine Learning, Segurança

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

ACRÓNIMOS

A

AUC Area under the curve.

C

CV Cross-Validation.

CWE Common Weakness Enumeration.

H

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

J

JSON JavaScript Object Notation.

JWT Json Web Token.

M

MD5 Message-Digest 5.

ML Machine Learning.

N

NB Naive Bayes.

O

OWASP Open Web Application Security Project.

Q

QA Quality Assurance.

R

ROC Receiver operating characteristic.

S

SHA1 Secure Hash Algorithm 1.

SVM Support-vector machine.

X

XSS Cross-site scripting.

1

INTRODUÇÃO

Este primeiro capítulo consiste na introdução ao projeto, as motivações que levaram ao seu desenvolvimento, as metodologias utilizadas para a sua realização e a estrutura do presente documento.

1.1 MOTIVAÇÃO

No processo de desenvolvimento de *software*, a implementação dos vários conceitos de segurança da solução desenvolvida é um dos pontos fulcrais para o sucesso e credibilidade da mesma. Neste sentido são efetuados vários testes a possíveis vulnerabilidades às quais o sistema desenvolvido possa estar exposto. É de extrema utilidade tirar partido de técnicas de Machine Learning para efetuar o *scan* de vulnerabilidades permitindo aumentar a eficiência e precisão destes mecanismos como enunciado em More and Rohela (2018).

Com base nesta informação surge a necessidade da criação de uma ferramenta com vista a ser integrada na área de controlo de qualidade. Esta ferramenta será também dotada de técnicas de *Machine Learning* para que seja possível automatizar em parte a análise dos *cookies* utilizados em aplicações *web*, tornando assim este processo mais rápido e eficiente.

Utilizando a ferramenta desenvolvida, será possível não só detetar os *cookies* associados à autenticação, como também alertar os profissionais de *QA* para o *encoding* utilizado no conteúdo do *cookie* através da sua análise.

1.2 OBJETIVOS

Esta dissertação tem como objetivo a aplicação de técnicas de *Machine Learning* na área da segurança, ou seja, criação de uma ferramenta que permita a automação de análise/teste de *cookies* garantido uma melhoria na eficiência deste procedimento.

Como resultado final espera-se a implementação de uma aplicação fiável, utilizando o classificador desenvolvido e integração da mesma no processo de análise de *software* executado pelos profissionais de *QA*.

A seguir são apresentados os objetivos da dissertação:

- Pesquisa e recolha de informação sobre a aplicação de técnicas de *machine learning* na área da segurança, mais precisamente, na deteção e classificação de *cookies*
- Criação de um classificador que permita identificar os *cookies* de autenticação dentro de um conjunto de dados composto por vários tipos de *cookies*
- Deteção e análise da segurança do *encoding*/cifragem utilizado nos *cookies* de autenticação
- Com base na análise anterior, invocação de uma vulnerabilidade no *Burp Suite* caso seja encontrado um *cookie* inseguro na aplicação *web*

1.3 METODOLOGIA

Com vista a atingir os objetivos propostos, esta dissertação seguirá a seguinte metodologia:

- Reuniões de acompanhamento do trabalho com a equipa de *QA* e *Machine Learning* da empresa Celfocus
- Revisão bibliográfica
- Desenvolvimento de um classificador de *cookies*
- Deteção e análise dos encodings utilizados
- Desenvolvimento do *plugin* para o *Burp Suite*
- Integração do plugin desenvolvido no *Burp Suite*
- Validação do *software* utilizando o *SonarQube* e testes unitários

1.4 ESTRUTURA DO DOCUMENTO

Este documento encontra-se organizado em 7 capítulos. O primeiro capítulo descreve a motivação do tema desenvolvido, os objetivos propostos para esta dissertação e a metodologia utilizada.

O segundo capítulo consiste nos conceitos básicos associados ao tema apresentado neste documento. Neste capítulo são introduzidos os *cookies*, identificando os vários tipos, explicando a sua importância e as questões associadas à privacidade do utilizador na utilização de *cookies*. São apresentados os *encodings* utilizados nos *cookies* e conceitos relacionados com a

criação de modelos de *machine learning*. Por fim, relativamente à segurança e comportamento do código é explicada a utilização do *SonarQube* e testes unitários.

O terceiro capítulo é o Estado de Arte, este apresenta um artigo que desenvolve uma ferramenta de classificação de *cookies* e analisa os resultados obtidos por outros classificadores disponíveis no mercado.

No quarto capítulo será feita uma introdução aos dados a utilizar no desenvolvimento do classificador de *cookies*. Por fim, será apresentada a arquitetura a utilizar no sistema a desenvolver.

O quinto capítulo explica toda a fase de desenvolvimento, evidenciando as tecnologias utilizadas, a razão para as utilizar e como foram implementadas. No final neste de capítulo, é mostrada a verificação do *software* desenvolvido recorrendo aos testes unitários e ao *SonarQube*.

O sexto capítulo consiste na demonstração das funcionalidades da extensão, mostrando alguns exemplos de utilização da ferramenta desenvolvida em diversas aplicações *web* e analisando detalhadamente os resultados obtidos.

Por fim, o sétimo capítulo é a conclusão do documento, é feito um resumo do trabalho enunciando os objetivos alcançados. São também indicados os desafios encontrados ao longo de todo o desenvolvimento do projeto. Termina a conclusão com algumas sugestões de trabalho futuro e considerações finais.

2

CONCEITOS BÁSICOS

Este capítulo explica conceitos importantes associados aos temas abordados ao longo da dissertação. Os temas apresentados são os *cookies*, *encodings* utilizados nos *cookies*, criação de modelos de *machine learning* e segurança do código.

2.1 COOKIES

2.1.1 *Definição*

Como enunciado em Mozilla e OWASP, os *cookies* são pequenos ficheiros gerados pelos *websites* e enviados ao navegador. Este armazena os *cookies* por um determinado período de tempo de acordo com o parâmetro tempo de expiração contido neste ficheiro. Para além do tempo de expiração, estes contêm o nome do *cookie*, o valor, o domínio que identifica onde pode ser utilizado e por fim duas *flags*. Estas *flags* são *HTTPONLY* caso seja verdadeira o *cookie* só pode ser enviado pelo protocolo *HTTP* e a *flag Secure* indicando que só pode ser enviado através de uma ligação segura, mais conhecida como, *HTTPS*. Os *cookies* são uma das formas de autenticação de um utilizador numa aplicação *web*, caso contrário sempre que um utilizador acede/troca de página *web* tem de inserir as suas credenciais, invalidando a existência de uma noção de estado do utilizador na aplicação. Para além da autenticação de clientes, existem outros tipos de *cookies* que ajudam os *websites* a personalizar os dados que vão mostrar ao utilizador. Ainda, em serviços de venda de produtos online, os *cookies* permitem à aplicação *web* obter acesso a informações como por exemplo quais os itens que o utilizador adicionou ao "carrinho de compras". Por fim, convém relembrar que estes ficheiros podem ser consultados no navegador utilizando as ferramentas de desenvolvedor disponibilizadas.

2.1.2 *Utilidade*

As 3 principais utilidades da utilização de *cookies* são a autenticação *web*, ou seja, a implementação da noção de "estado"do utilizador na aplicação, a personalização de *websites* de

acordo com as preferências dos utilizadores e para análise do comportamento de um usuário como enunciado em Mozilla.

A autenticação *web* é garantida através da criação de *user sessions* para que uma dada atividade seja associada ao utilizador. Estes *session cookies* contêm uma *string* (identificador) que permite fazer a correspondência entre a acção efetuada e o utilizador associado.

Observando a figura 1 em que temos o servidor de um *website*, quando um utilizador efetua a autenticação, o *website* envia um *cookie* de sessão para o *browser* do utilizador. De seguida o *browser* envia o *cookie* recebido para o servidor, o *website* passa a apresentar o conteúdo da página associado ao utilizador em questão. Existem mais exemplos de utilização deste mecanismo como quando um utilizador pretende consultar um produto na aplicação *web*, o navegador envia um pedido *HTTP* para o *website* com o conteúdo que pretende visualizar e o *cookie* de sessão. Desta forma, o *website* reconhece o utilizador que enviou o pedido e envia a resposta com o conteúdo da página *web*.

Outra função é o armazenamento de um "histórico"de utilizador, os *cookies* dão informação aos *websites* das acções que estes fizeram nas suas páginas, assim podem ser apresentadas páginas customizadas melhorando a experiência do utilizador.

Por fim, recorrendo aos *cookies*, as aplicações podem ter acesso aos *websites* que os utilizadores visitaram. Esta informação é enviada para o servidor que gerou o *cookie* quando o *browser* volta a uma das páginas associada a esse servidor. A aplicação mais simples desta funcionalidade dos *cookies* são os anúncios, ou seja, pesquisamos um dado produto no navegador e depois quando acedemos a um *website* que utiliza este tipos de *cookies*, o mais provável é ser-nos apresentados anúncios desse produto que pesquisamos anteriormente.

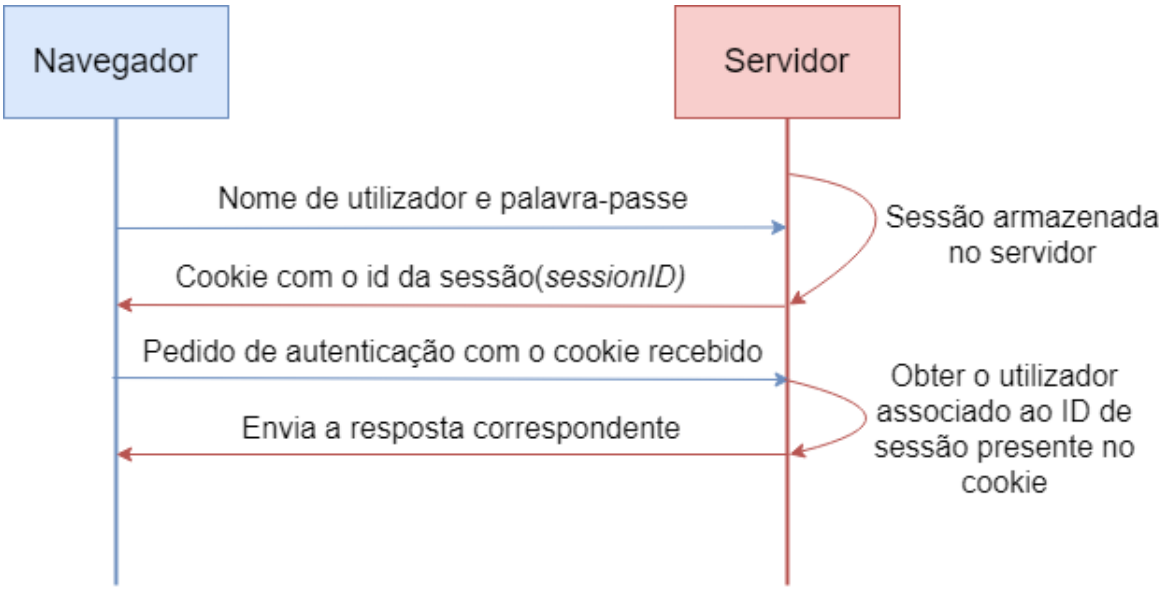


Figura 1: Autenticação de um utilizador numa aplicação *web*

2.1.3 *Tipos de cookies*

Ao longo desta secção serão abordados os seguintes tipos de *cookies* de acordo com Mozilla e OWASP:

Cookies de sessão

Os *cookies* de sessão, conhecidos como *Session cookies*, estão integradas nos *websites* para monitorizar a sessão do utilizador, permitem acompanhar todos os movimentos que este faz na aplicação mantendo a sessão ativa. Estes *cookies* normalmente não têm uma data de expiração, ou seja, a única forma de expirarem é quando o utilizador efetua o *logout* ou fecha o *website*, assim o browser tem a função de apagar o ficheiro quando a sessão termina.

Cookies persistentes

Este tipo de *cookies*, ao contrário dos *cookies* de sessão têm uma data de expiração mas como o nome indica, são persistentes, por isso mantêm-se no navegador até atingir esta data. Estes são utilizados para guardar informações, preferências do utilizador no *website* ou até credenciais para este se autenticar.

Cookies de autenticação

Os *cookies* de autenticação facilitam o processo de autenticação, estes são gerados quando o utilizador faz o *login* na aplicação e asseguram a entrega de informações ao utilizador especificado através da associação entre o conteúdo da *cookie* e o identificador do mecanismo de autenticação da aplicação *web*.

Tracking cookies

Estes são gerados pelos *websites* que possuem serviços de *tracking* e servem para guardar informação sobre a atividade do utilizador no browser. Quando o utilizador visita uma página associada a um serviço que utiliza estes *cookies*, o servidor irá receber informação para que sejam apresentadas páginas personalizadas com base no conteúdo visualizado pelo utilizador. O exemplo mais comum são os anúncios personalizados que nos são apresentados quando acedemos a um determinado *website*. Os *tracking cookies* têm uma outra utilização para além da exposição de anúncios personalizados, que é a recolha de dados da experiência do utilizador para futura geração de estatísticas.

Zombie cookies

Tal como o nome indica, estes são *cookies* que se regeneram após serem eliminados. O funcionamento é o seguinte, são criados *backups* fora da localização habitual(no navegador)

e utilizando estes *backups* são repostas quando são eliminadas do navegador. Este tipo de *cookies* está, normalmente, associada a falhas de segurança no navegador levando à criação destas por aplicações mal-intencionadas.

First-Party vs Third-Party Cookies

Um *cookie* é considerado *First-Party Cookie* quando pertence ao serviço onde está a ser utilizado, enquanto que os *Third-Party* são *cookies* pertencentes a outro domínio que são apresentados num *website* que não o seu. Alguns exemplos simples disto são quando navegamos num *website* em que estamos autenticados é usado um *cookie* de sessão para nos identificar, esta pertece ao domínio em que estamos, logo é um *First-Party Cookie*. Quando nos encontramos neste mesmo *website* e são-nos apresentados anúncios de produtos fornecidos por outros provedores ou aparecem botões associados a redes sociais que permitem fazer *login* ou gostar/partilhar o conteúdo que estamos a visualizar nas suas redes socais, tratam-se de *Third-Party Cookies*.

2.1.4 *Cookies e a privacidade dos utilizadores*

Como mencionado anteriormente, os *cookies* podem ser usados para recolher dados sobre a navegação do utilizador no *browser*. Um problema é que muitos utilizadores não querem de todo que o seu "histórico"e as suas pesquisas sejam monitorizados por outros através dos *tracking cookies*. Outro problema deve-se ao facto de que apesar de os utilizadores saberem que estão a ser recolhidos dados sobre as suas pesquisas, estes não têm de qualquer forma acesso ao que é feito com as suas informações. Mesmo que a identidade do utilizador se mantenha anonimizada, através de todos os dados recolhidos poderá ser estabelecido um elo de ligação que identifique o utilizador.

Como ponto positivo já existe legislação neste sentido, enunciada no GDPR(General Data Protection Regulation). Existem então normas definindo que o utilizador deve ser notificado sobre os *cookies* que estão a ser utilizados e se concorda com os mesmos. Esta informação deverá ser precisa e explícita sobre quais os dados que o *cookie* irá monitorizar antes que o utilizador aprove a sua utilização.

A única exceção são os *cookies* que são estritamente necessários para o normal funcionamento da aplicação *web*, ou seja, nestes casos não é necessário o consentimento do utilizador.

Uma das regras afirma que o esforço necessário para consentir o uso de um *cookie* deverá ser exatamente o mesmo da remoção destas permissões. Por fim, os utilizadores devem conseguir aceder aos serviços fornecidos pelo *website* mesmo que recusem o uso de determinado tipo de *cookies*.

2.2 ENCODINGS

Nesta secção serão apresentados os *encodings* a detetar pela nossa extensão:

- MD5 - O algoritmo MD5 é um algoritmo de *hashing one-way*, ou seja, não é reversível. Apartir de um determinado *input* é aplicado o *hashing MD5* gerando uma chave de 32 caracteres, como se trata de um *hashing one-way* não é possível fazer o processo inverso e obter o *input* inicial. O *MD5* é muito popular, sendo conhecida a sua estrutura composta por uma sequência de 32 caracteres hexadecimais(números de 0-9 e letras de A-F) como enunciado em Ratna et al. (2013).

  Exemplo: "5ea09914a43a59c5bf18ab5504672608"

- SHA1 - O algoritmo SHA1 é um *hashing one-way* muito semelhante ao exemplo anterior MD5, a única diferença na sua estrutura é a utilização de 40 caracteres hexadecimais em vez dos habituais 32 caracteres como enunciado em Ratna et al. (2013).

  Exemplo: "d10695e793910cb6d9ace3cc44c469fa7b27fa3b"

- Base32 - O base32 é um sistema numérico de *encoding* de dados reversível ao contrário dos exemplos anteriores *MD5* e *SHA1* que são *one-way*. Este tem uma estrutura composta por sequências de 8 caracteres, ou seja, o tamanho do valor de um *cookie* com *encoding* base32 será um múltiplo de 8 excluíndo o zero. Os caracteres admitidos nesta estrutura são as letras de A-Z, os dígitos de 2-7 e o = completando os 32 caracteres possíveis como enunciado em Josefsson et al. (2006). O caracter = é classificado como padding porque é utilizado para completar o texto base32 quando não temos uma sequência de caracteres múltipla de 8.

  Exemplo: "MV4GK3LQNRXQ===="

- Base64 - O base64 é uma variação do *encoding* anterior, base32, sendo que hoje em dia o base64 é muito utilizado pelos desenvolvedores de software para codificação de dados ao contrário do base32 que é menos comum. Este *encoding* é reversível e é constituído por sequências de 4 caracteres, utilizando números de 0-9, letras maiúsculas e minúsculas de A-Z e por fim, os caracteres + / e = completando os 64 caracteres possíveis como enunciado em Josefsson et al. (2006). O igual é utilizado como *padding* tal como foi explicado no exemplo anterior do base32.

  Exemplo: "CNFo/tf3fVZT1xYgy6bv/g=="

- Hexadecimal - O hexadecimal é uma forma de representação de números na base 16. Este *encoding* é reversível e é constituído pelos números de 0-9 e letras de A-F como enunciado em Latif et al. (2011).

  Exemplo: "ffffffff122ad8b445525d5f4f58455e445a4a420130"

- JWT - O *JWT (JSON Web Token)* é um método compacto e seguro para representar a transferência de dados entre duas partes como indicado em RFC7519. Pode ser assinado utilizando uma chave secreta e é depois inserido no valor do *cookie* para permitir a autenticação na aplicação *web* como enunciado em Jones et al. (2015). Estão divididos em três campos, *header*, *payload*, e *signature*. Com base nesta informação, o nosso mecanismo fará a verificação destes três campos de forma a comprovar se o texto corresponde a um *JWT*.

  Exemplo: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkV4ZW1wbG8iLCJpYXQiOjE1MTYyMzkwMjJ9.wdXcY0OID2Q-pdhfW6ByVsS6tKb-FmVaxXtu2B1deiA"

- PHPSESSID - O PHPSESSID é um *encoding* muito específico encontrado em aplicações que recorrem à linguagem PHP. São gerados *cookies* com um identificador no campo value que utiliza esta codificação, permitindo dar *track* à sessão do utilizador. Este *encoding* é caracterizado pelo uso de números de 0-9 e letras de a-z constituíndo um tamanho total do valor do *cookie* de 26 caracteres como referido em PHP.

  Exemplo: "ioh4eo3vo08tb91ndfvr11hpc2"

- JSON - Por fim, foram encontrados alguns casos em que o valor do *cookie* continha estruturas de formato JSON. Normalmente é necessário um URL decode porque caracteres que fazem parte da sintaxe *JSON*(JSON) como o caracter "{" e o caracter "}" não podem ser adicionados diretamente no *cookie*. Após o processo de *URL Decode* do *cookie* se forem encontradas sequências de caracteres começadas pelo caracter "{" e terminadas no caracter "}" provavelmente será um *cookie* com uma estrutura *JSON*.

  Exemplo: '%7B%22token%22%3A%22lNdsARBVv+mbRLk7v3NJaW1ihnLtxNq9WwlWSzDlUNifvINUbaoyJ/mJK/9YVkdG%22%7D'

  Exemplo após aplicar o *URL Decode*: '{"token":"lNdsARBVv mbRLk7v3NJaW1ihnLtxNq9WwlWSzDlUNifvINUbaoyJ/mJK/9YVkdG"}'

2.3 MACHINE LEARNING

Como enunciado em Wirth and Hipp (2000), será explicado o processo de desenvolvimento de um modelo de *machine learning* seguindo a metodologia *CRISP-DM (CRoss Industry Standard Process for Data Mining)*. Esta baseia-se nas seguintes etapas:

- *Business Understanding* - Esta fase inicial está focada na perceção dos objetivos e requisitos do projeto. Com base nestes dados, fazer a conversão para um problema de *data mining*. Determinar os recursos disponíveis, os riscos, as contingências e elaborar uma análise custo-benefício. Para além dos objetivos do projeto devem ser estudados também os objetivos do problema de *data mining*. Por fim, definir as tecnologias e ferramentas a utilizar em cada fase do projeto.

- *Data Understanding* - A fase de *data understanding* começa com uma recolha inicial de dados e com atividades para identificar a qualidade destes dados. Há uma ligação entre este passo de *Data Understanding* e o *Business Understanding* porque para definirmos os objetivos do projeto e do problema de *data mining* é necessário conhecer os dados disponíveis. Como resumo, esta etapa baseia-se na recolha do conjunto de dados a utilizar, análise das propriedades deste conjunto como por exemplo o formato, número de registos e os campos que o compõem. Depois será efetuada a tarefa de exploração dos dados através de métodos de visualização de dados e identificação de relações nos mesmos. Por fim, é analisada a qualidade dos dados recolhidos.

- *Data Preparation* - A *data preparation* representa todas as atividades associadas à construção do *dataset* final. As tarefas mais comuns são a seleção de atributos, limpeza dos dados, construção de novos atributos e transformação de dados para enviar ao modelo. Uma grande parte do desenvolvimento do projeto foca-se nesta etapa para garantir que obtemos um bom conjunto de dados para enviar ao modelo de *machine learning*. As etapas da *data preparation* são a seleção dos *datasets* a utilizar, limpeza dos dados através da remoção de erros no conjunto de dados como por exemplo *outliers* e *missing values*. Derivação de novos atributos que podem ser úteis na tomada de decisão do nosso modelo, por exemplo através da operação entre duas variavéis já existentes no conjunto de dados. A integração de dados é também importante e pode ser feita apartir da combinação de vários conjuntos de dados de fontes diferentes. Após a construção dos *datasets*, a última etapa é a normalização das variáveis por exemplo se o conjunto de dados contém uma coluna em formato textual e queremos executar operações matemáticas, é necessário converter essa coluna para valores numéricos.

- *Modeling* - Esta fase consiste na seleção dos modelos a utilizar e na calibração dos seus parâmetros para os valores ótimos. São efetudas as etapas de seleção dos algoritmos de *machine learning* com base no conjunto de dados preparado na *data preparation*. Depois, definir a estrutura da divisão dos dados em treino, teste e validação. Por fim, são construídos os modelos e consequentemente avaliados de acordo com os resultados gerados e os objetivos definidos nas fases anteriores.

- *Evaluation* - A fase de *evaluation*, analisa qual dos modelos desenvolvidos na etapa de *modeling* se enquadra melhor no modelo de negócio definido no *business understanding*. Consiste nas etapas de avaliação dos resultados obtidos, revisão do trabalho desenvolvido, ou seja, verificar se todos os passos foram executados corretamente ou se é necessário fazer alguma correção. Para terminar, definir os próximos passos a seguir para proceder ao *deployment* do projeto.

- *Deployment* - Um modelo não é útil até que o utilizador possa aceder aos seus resultados por isso é necessário implementar a última fase que é o *deployment*. Nesta fase é desenvolvido e documentado um plano para fazer o *deployment* do projeto. Uma das etapas é também o planeamento da monitorização e manutenção para evitar problemas durante a fase operacional do modelo. É necessária a produção de um relatório final para documentar todas as fases de desenvolvimento e pode incluir uma apresentação final dos resultados gerados. Depois é elaborada uma retrospetiva sobre o que correu bem no projeto, o que poderia ter sido melhor e como melhorar no futuro.

2.3.1 *Divisão do conjunto de dados*

O conjunto de dados é normalmente dividido em dados de treino e dados de teste para avaliar o desempenho do modelo de *machine learning*. Esta divisão é utilizada para algoritmos de aprendizagem supervisionada. Este procedimento consiste na divisão do *dataset* em dois conjuntos. O primeiro conjunto (train *dataset*) é utilizado para treinar o modelo. O segundo conjunto (test *dataset*) não é utilizado para treinar o modelo mas sim para avaliar o modelo desenvolvido, ou seja, são enviados os *inputs* deste conjunto e o classificador de *machine learning* irá fazer uma previsão do *output*. Depois, estas previsões serão comparadas com os valores esperados para tirar conclusões sobre o desempenho do modelo.

De forma geral, a metodologia baseia-se no treino do modelo com os dados de treino e depois testar o modelo com dados que este não conhece, de modo a prever o seu comportamento em uma situação real. Convém realçar também que não existe um valor ótimo para as percentagens da divisão dos dados em treino/teste, depende essencialmente dos objetivos do projeto em questão.

Em modelos de classificação, os dados podem não estar balanceados de forma a que seja semelhante o número de exemplos de cada classe. Para estes casos, tem de haver um cuidado adicional da divisão dos dados em treino/teste para manter a mesma proporção de dados de cada classe nos dois conjuntos. Para resolver este problema é utilizada a divisão *stratified* garantindo que o conjunto de treino e teste têm a mesma percentagem de elementos de cada classe contida no *dataset*.

Um dos problemas é o *overfitting* quando estamos a otimizar os parâmetros para o conjunto de teste até que o classificador tenha o desempenho esperado. Desta forma, o modelo pode obter informações sobre os dados de teste que influenciem as previsões, logo as métricas de avaliação deixam de ser válidas e já não se pode verificar uma generalização dos resultados. Para resolver este problema, o *dataset* é dividido em mais uma parte designada como dados de validação. O treino do modelo é feito com os dados de treino e depois é feita a avaliação com os dados de validação. Quando esta avaliação retornar valores satisfatórios, é efetuada a avaliação final com os dados de teste.

No entanto, ao dividir os dados em 3 conjuntos, a quantidade de registos para treino do modelo é reduzida drasticamente. Assim, os resultados serão influenciados de acordo com a partição aleatória do conjunto de dados.

Uma solução para isto é a utilização de *CV (Cross-Validation)*. Os dados de teste são utilizados na avaliação final do modelo mas os dados de validação deixam de ser necessários. Na abordagem básica, chamada *k-fold CV*, os dados de treino são divididos em k conjuntos. O prodecimento consiste no treino do modelo utilizando k-1 conjuntos como dados de treino. Os resultados do modelo são validados com o conjunto que restou da utilização de k-1 para treino, este irá funcionar como um conjunto de teste. A avaliação do desempenho do classificador reportada pelo *k-fold cross-validation* consiste na média dos valores calculados ao longo do ciclo de seleção dos k conjuntos.

Esta abordagem é computacionalmente dispendiosa mas não utiliza demasiados dados como é o caso da criação de um conjunto de dados de validação. Esta é uma grande vantagem especialmente em problemas que contêm um pequeno conjunto de dados associado.

2.3.2 *Otimização de parâmetros do modelo*

Após a divisão em dados de treino e dados de teste, são utilizados os dados de treino para treinar o algoritmo de *Machine Learning*. Nesta etapa é selecionado o algoritmo de *Machine Learning* a utilizar e são testados diferentes valores para os parâmetros de configuração do algoritmo.

Para obter os parâmetros ótimos é utilizado o *gridsearch*, ou seja, os valores com os quais o classificador devolve os melhores resultados. O *gridsearch* irá testar todas as combinações dos parâmetros recebidos para encontrar qual a combinação que permite ao modelo de

machine learning obter o melhor desempenho. Este também permite definir a métrica de avaliação do modelo que queremos maximizar e o método de *cross-validation* a utilizar. O *cross-validator* mais rigoroso é o *Stratified K-Fold*, este permite dividir o conjunto de dados no número de *folds* definidos e preserva a percentagem de elementos de cada classe em cada conjunto de treino/teste.

2.3.3 *N-grams*

Como enunciado em Srinidhi, um *N-gram* significa uma sequência de N palavras. Por exemplo, *"Medium blog"* é *2-gram(bigram)*, *"A Medium blog post"* é *4-gram* e *"Write on Medium"* é *3-gram(trigram)*.

O conceito de *N-gram* é utilizado na criação de modelos de *machine learning* de classificação de texto. Dependendo do problema a estudar, o modelo pode interpretar o texto palavra a palavra(*n-gram=1*) ou sequências de palavras com um *n-gram* maior que 1. Este número é ajustado de acordo com a avaliação de resultados obtidos.

2.3.4 *Métricas de avaliação do classificador de machine learning*

As métricas para avaliação do classificador binário baseiam-se na análise da matriz de confusão das previsões geradas. De seguida serão explicadas as métricas utilizadas ao longo do documento:

- *Sensitivity* - A *sensitivity* (sensibilidade) representa dentro dos *cookies* de autenticação (classe 1), qual a percentagem destes que é classificada corretamente. É calculada apartir da divisão entre os verdadeiros positivos e a soma dos verdadeiros positivos com os falsos negativos como observamos na seguinte expressão matemática:
$$sensitivity = \frac{tp}{tp + fn}$$
- *Specificity* - A *specificity* (especificidade) consite na percentagem de *cookies* que não são de autenticação (classe 0) e que foram classificados corretamente. O cálculo desta métrica é feito apartir da divisão entre os verdadeiros negativos e soma dos verdadeiros negativos com os falsos positivos como observamos na seguinte expressão matemática:
$$specificity = \frac{tn}{tn + fp}$$
- *F-measure* - A *F-measure* é a métrica que calcula o equílibrio entre as duas métricas anteriores. O resultado é obtido recorrendo ao cálculo da média harmónica entre as métricas *sensitivity* e a *specificity*. A sua expressão matemática é a seguinte:
$$F - measure = 2 \cdot \frac{specificity \cdot sensitivity}{specificity + sensitivity}$$

- *ROC curve* - Como enunciado em Pereira et al. (2021), a *ROC (Receiver operating characteristic) curve* é um método popular de análise de classificadores. Esta demonstra o desempenho de um classificador binário apartir do desenho do valor de 1-*specificity* no eixo x e o valor da *sensitivity* no eixo y. Com base nos valores destas duas métricas *sensitivity* e *specificity* é desenhada uma curva que dará origem à *AUC* (Area under the curve).

- *AUC* - A *AUC* (Area under the curve) mede a área bidimensional abaixo de toda a *ROC curve* desde o ponto (0,0) ao ponto (1,1) do gráfico. Esta métrica tem duas principais vantagens: Em primeiro lugar, os valores resultantes da aplicação da *AUC* não são afetados caso haja um desbalanceamento no conjunto de dados do modelo. Isto significa que caso seja utilizado um modelo que atribui previsões de forma aleatória, o valor da *AUC* será 50%. Em segundo lugar, os valores da *AUC* são facilmente interpretáveis. Normalmente, os valores do *AUC* podem ser interpretados da seguinte forma: 50% - desempenho de um classificador aleatório, 60% - razoável, 70% - bom, 80% - muito bom, 90% - excelente e 100% - perfeito como enunciado em Pereira et al. (2021).

2.4 BURP SUITE

Como enunciado na documentação do Burp, o *Burp Suite* é uma ferramenta para realização de testes de segurança a aplicações *web*. Para efetuar testes de segurança no *Burp*, é possível utilizar o navegador desenvolvido pelo *Burp* que está embebido na aplicação ou um navegador externo sendo necessária a sua configuração de acordo com os requisitos do *Burp* e instalação dos devidos certificados.

Após esta etapa, ao visitarmos qualquer página *web* no navegador, o tráfego será armazenado pelo *proxy* do *Burp* no *proxy history* da aplicação. É possível utilizar também a funcionalidade *intercept* em que cada pedido *HTTP* enviado pelo navegador é registado na aba *Intercept* do *Burp*. Podemos editar esta mensagem e enviar para o servidor associado para verificarmos o comportamento da aplicação *web* ao fazermos esta alteração. De relembrar que podemos ter o *Intercept* ligado ou desligado dependendo se queremos navegar normalmente na página *web* ou se pretendemos observar cada pedido enviado pelo navegador.

Para o nosso caso de estudo, a funcionalidade mais interessante é o *scanner* do *Burp*, este permite analisar automaticamente um *website* definido ou enviar ao *scanner* um determinado número de mensagens *HTTP* para este analisar. O *scanner* irá analisar a aplicação *web* em questão e reportar as vulnerabilidades de segurança encontradas.

2.5 SONARQUBE

O *SonarQube* é uma ferramenta automática para avaliação de qualidade do código, detetando *bugs*, *code smells* e vulnerabilidades de segurança. Esta avaliação é obtida através do uso de métricas desenvolvidas pelo *SonarQube*. Passando à explicação de algumas destas métricas de acordo com a documentação do SonarQube:

- Bug - É um problema que representa algo incorreto no código. Se este *bug* ainda não foi encontrado, certamente irá aparecer no futuro por isso tem de ser corrigido imediatamente.
- Code Smell - O *code smell* é um problema relacionado com a manutenção do código. Caso não seja solucionado irá, na melhor das hipóteses, dificultar a adição de futuras alterações e a manutenção do mesmo. Dificulta a perceção do *código* podendo levar à inclusão de erros quando forem feitas alterações.
- Vulnerability - Consite numa falha de segurança no código desenvolvido. Pode constituir uma *backdoor* do sistema para os atacantes explorarem. O *SonarQube* verifica as vulnerabilidades enunciadas em CWE, OWASP_Foundation e SANS.
- Security Hotspot - A definição de *security hotspot* é parecida com uma vulnerabilidade, a diferença é que um *security hotspot* é preciso ser revisto antes de saber se são necessárias

alterações no código. Quando é encontrada uma vulnerabilidade sabemos que esta tem um impacto na segurança da aplicação e precisa de ser resolvida imediatamente. Enquanto que o *security hotspot* representa um pedaço de código sensível que precisa de ser revisto para determinar se são necessárias alterações de forma a manter a segurança da aplicação. Neste caso sabemos que não afeta globalmente a segurança da aplicação ao contrário de uma vulnerabilidade.

- Complexity - É a complexidade ciclomática calculada com base no número de caminhos através do código. Sempre que existe numa função um fluxo de controlo que divide a execução, a complexidade é aumentada em 1. A complexidade mínima por função é 1.
- Code Duplication - Como o nome indica, esta métrica verifica se existem blocos de código duplicados, indicando o ficheiro e as linhas em que ocorre.
- Code coverage - O objetivo desta métrica é responder à seguinte questão: "Qual a percentagem de código fonte que está coberto por testes unitários". Verifica quais linhas de código foram executadas durante a execução dos testes unitários, resultando depois numa percentagem final de cobertura.
- Reliability - A métrica *reliability* está relacionada com o número de *bugs* detetados no código da aplicação, a classificação atribuída por esta varia entre A a E. Se a aplicação não tiver nenhum *bug* é atribuída a melhor classificação, ou seja, "A". A avaliação dada pelo *SonarQube* varia de acordo com o número de bugs e a sua gravidade.
- Security - A *security* está associada ao número de vulnerabilidades presentes no código. Tal como na métrica *reliability*, a classificação varia de acordo com o número de vulnerabilidades detetadas e a sua gravidade. Os resultados enquadram-se entre A-E, sendo "A" o melhor resultado e "E" o pior resultado possível.
- Security Review - A métrica *security review* avalia a percentagem de *security hotspots* que foram revistos, sendo estes corrigidos ou considerados seguros após a análise do utilizador. A gama de valores de classificação é entre A-E, atribuíndo "A" se esta percentagem for superior a 80% e "E" se for inferior a 30%.
- Maintainability - A *maintainability* está associada ao número de *code smells* detetados no código ao longo do desenvolvimento e ao esforço do desenvolvedor a corrigi-los, ou seja, o tempo necessário para os corrigir após a deteção. Os valores possíveis para esta métrica variam entre A-E sendo "A" a melhor classificação e "E" a pior.

É também possível criar um relatório evidenciando, em modo gráfico ou textual, os resultados das diversas métricas. Por fim, é possível observar graficamente a evolução destes parâmetros ao longo do tempo, à medida que é feita a codificação.

Na Figura 2 temos um exemplo do painel geral da análise das métricas efetuada pelo *SonarQube*.

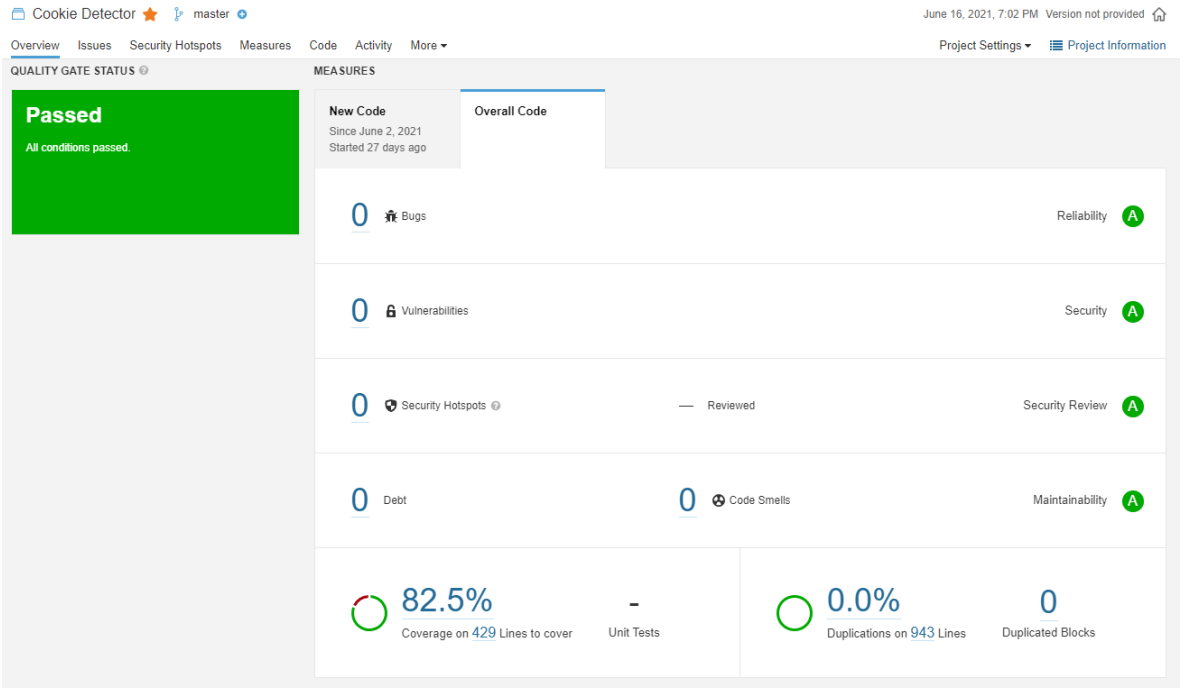


Figura 2: Exemplo da análise geral das métricas de avaliação do código segundo o *SonarQube*

2.6 TESTES UNITÁRIOS

Os testes unitários são uma automatização para verificar o comportamento de uma função ou secção do código. A metodologia consiste no envio de múltiplos *inputs* à função alvo e observar se retorna os *outputs* esperados. Para este procedimento existem vários módulos na linguagem python que podem ser utilizados, sendo que a nossa escolha para esta implementação foi o *pytest*.

3

ESTADO DA ARTE

Ao longo do estudo, foram encontrados os artigos científicos Calzavara et al. (2014) e Calzavara et al. (2015), que desenvolvem um método semi-automático de recolha de *cookies*. Este processo resulta num *dataset* verificado no qual todos os *cookies* de autenticação são isolados e identificados corretamente. O *dataset* disponibilizado é composto por 2464 *cookies* recolhidos dos 215 *websites* mais populares de acordo com a *Alexa ranking*. De notar que a cada 7 *cookies* presentes no conjunto de dados, apenas 1 deles corresponde a um *cookie* de autenticação. Desta forma o trabalho de desenvolvimento de um classificador com recurso a técnicas de *machine learning* torna-se muito mais complexo, sendo necessário utilizar métodos de balanceamento como por exemplo *Undersampling* ou *Oversampling* do *dataset* como enunciado em More (2016).

A utilização deste contributo é de extrema importância porque nos permite avaliar o desempenho do detetor de *cookies* de autenticação e perceber a proteção que este nos oferece.

Ao longo do artigo são também avaliados os detetores de *cookies* disponíveis no mercado utilizando este *dataset*. As métricas utilizadas para a avaliação baseiam-se na matriz de confusão das previsões gerada pelo classificador de *machine learning*. São utilizadas três métricas, a *sensivity* (sensibilidade) que representa dentro dos *cookies* de autenticação, qual a percentagem destes que é classificada corretamente, a *specificity* (especificidade) tem uma abordagem similar mas para os *cookies* que não são de autenticação, ou seja, calcula a percentagem destes *cookies* classificada corretamente. Por fim a *F-measure* calcula a média harmónica entre as duas métricas mencionadas anteriormente.

As expressões matemáticas destas métricas são apresentadas no capítulo 2 (Conceitos básicos).

Para explicar o comportamento dos detetores de *cookies*, é importante a compreensão das seguintes vulnerabilidades de acordo com Calzavara et al. (2015):

- *XSS (Cross-site scripting)* - Os navegadores impedem que os *cookies* registados em um domínio sejam acedidos por *scripts* em execução em nome de um domínio diferente, de acordo com a política *same-origin*. Este mecanismo simples de proteção pode ser contornado recorrendo a ataques de injeção de código como o *XSS*, onde um *script* criado pelo atacante é executado no contexto de segurança de um site confiável. O ataque é possibilitado por uma falha no processo de verificação dos *inputs* inseridos no site (*input sanitization*), o que permite a injeção de código malicioso no *input* enviado à página *web*. Uma vez que os ataques *XSS* são tão populares hoje em dia, os servidores *web* devem utilizar a *flag HTTP-Only* nos *cookies* para que não seja possível executar *scripts* maliciosos. Assim, os *cookies HTTP-Only* só poderão ser acedidos pelo navegador quando é feito um pedido *HTTP/HTTPS* para o domínio que definiu o *cookie*. Os trabalhos de pesquisa desenvolvidos nesta área sugerem a aplicação automática desta *flag HTTP-Only* aos *cookies* de autenticação presentes no navegador quando o servidor não aplica esta medida de proteção.

- *Eavesdropping* - Os *browsers* adicionam todos os *cookies* correspondentes ao domínio para o qual é enviado o pedido *HTTP/HTTPS*. Por isso, quando uma página carregada por *HTTPS* retorna conteúdos adicionais, como por exemplo uma imagem utilizando uma conexão *HTTP* no mesmo domínio, os *cookies* de autenticação podem ser acedidos por um atacante que consiga intercetar o tráfego *web* não encriptado. De notar que mesmo os *cookies* de autenticação de *websites* totalmente implementados em *HTTPS* podem ser acedidos indevidamente caso o atacante consiga injetar *links HTTP* inexistentes, dado que o *browser* vai tentar aceder a esses *links*. Para retificar este problema, deve ser utilizada a *flag Secure* pelos servidores *web* para designar os *cookies* que só devem ser enviados por conexões *HTTPS* e nunca adicionados a pedidos *HTTP*. A *flag Secure* assim como a *flag HTTP-Only* pode ser seletivamente aplicada a *cookies* de autenticação no lado do cliente, obtendo proteção adicional contra possíveis ataques de rede.

- *Session Fixation* - Em uma ataque de *Session Fixation*, o atacante obtém o valor do *cookie* de autenticação o que permite identificar a sessão do utilizador. O atacante primeiro obtém um conjunto de cookies do *website* destino e depois envia-os apartir do *browser* do utilizador para esse mesmo *website*, através da exploração de uma vulnerabilidade *XSS*. Se o *website* não atualizar o valor do *cookie* de autenticação quando o estado da sessão se altera, por exemplo quando o utilizador se autentica ao enviar a *password*, o atacante pode "roubar"a sessão do utilizador fazendo-se passar por este enviando o conjunto de *cookies* inicial capturado. Um solução do lado do servidor contra este tipo de ataques é através da geração de um novo conjunto de *cookies* de autenticação

quando este recebe a *password* enviada pelo utilizador. Assim, como os novos *cookies* são diferentes dos *cookies* capturados pelo atacante, este não conseguirá aceder à sessão do utilizador. Se o servidor não implementar esta medida de segurança, a probabilidade deste tipo de ataques também pode ser reduzida significativamente do lado do cliente exigindo que os *cookies* de autenticação sejam registados apenas através de *headers HTTP/HTTPS*. A lógica deste procedimento é que os *headers HTTP/HTTPS* são um vetor de ataque muito menos comum do que *XSS*.

De seguida serão explicados os 4 detetores de *cookies* apresentados e avaliados no artigo Calzavara et al. (2015):

- *SessionShield* [Nikiforakis et al. (2011)] - É um *proxy* entre o navegador e o servidor com o objetivo de proteger sessões baseadas em *cookies* de autenticação contra ataques *XSS*. A ideia é automaticamente detetar e armazenar os *cookies* de autenticação em uma base de dados privada, inacessível com código *JavaScript*, emulando assim o comportamento do *browser* quando é utilizada a *flag HTTP-Only*.

- *Serene* [De Ryck et al. (2012)] - É uma solução do lado do cliente contra ataques de fixação de sessão(*session fixation*). Esta aplica um algoritmo de deteção de *cookies* que provavelmente são utilizados na autenticação, mas ainda não foram registados nos *headers HTTP*. Como estes *cookies* podem ser alvos de um ataque de *session fixation* utilizando *scripts* maliciosos, então são removidos das mensagens *HTTP* e nunca são utilizados para autenticação.

- *CookiExt* [Bugliesi et al. (2014)] - É uma extensão para o *Google Chrome* desenhada para assegurar a segurança dos *cookies* de autenticação contra ataques *XSS* e *Eavesdropping*. Esta extensão adiciona aos *cookies* as *flags Secure* e *HTTP-Only*, forçando o redirecionamento de *HTTP* para *HTTPS* se o *website* suportar.

- *Zan* [Tang et al. (2011)] - O *ZAN* é uma extensão para o navegador *OP2* destinado a proteger aplicações *web* contra diversos tipos de vulnerabilidades. Automaticamente aplica a *flag HTTP-Only* para os *cookies* de autenticação detetados e protege-os contra ataques *XSS*.

Como podemos observar na Figura 3 estes detetores demonstram claramente uma grande percentagem de falhas na classificação, evidenciando uma falsa confiança na segurança promovida pelos mesmos. Logo concluimos, tal como é enunciado no artigo, que estes classificadores propostos não têm resultados suficientemente satisfatórios para serem eficazes na prática.

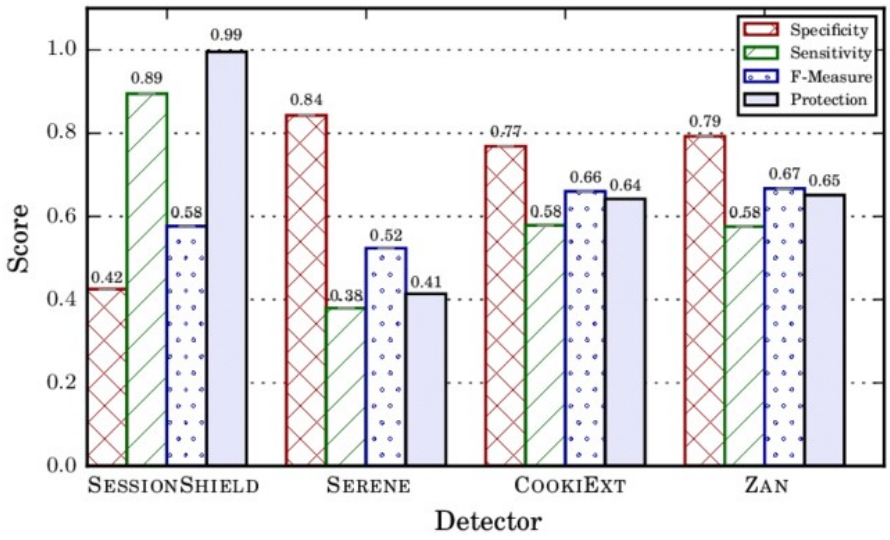


Figura 3: Desempenho de diferentes detetores de cookies apresentado em Calzavara et al. (2015)

O terceiro e final contributo desenvolvido neste artigo, consiste na criação de um classificador binário de *cookies* que permita de forma precisa identificar os *cookies* de autenticação baseado em modelos de *machine learning*. Utilizando técnicas de *machine learning* tais como *supervised learning* é possível superar os valores expostos na figura acima obtidos por outros sistemas de deteção de *cookies*.

Um dos objetivos propostos nesta dissertação é também o desenvolvimento de um classificador de *cookies* de autenticação tal como no artigo mencionado. A meta é superar os resultados das métricas apresentadas no gráfico anterior, assim como, os resultados finais do classificador desenvolvido pelos autores do artigo que serão apresentados mais à frente. Para além deste objetivo, será criada uma ferramenta para deteção e análise da segurança dos *encodings* utilizados nos *cookies*. E para terminar, será feita a integração com o *Burp Suite* para reportar vulnerabilidades quando for encontrado um *cookie* inseguro.

4

ABORDAGEM PROPOSTA

Neste capítulo são descritos os problemas encontrados nesta fase do projeto e quais as soluções encontradas para os resolver. Será apresentado o *dataset* a utilizar para o treino do classificador. No contexto do *dataset*, será demonstrado o pré-processamento de dados aplicado antes do desenvolvimento do classificador. Por fim, terá lugar a apresentação da arquitetura da ferramenta a desenvolver.

4.1 DATASET

O *dataset* base utilizado no treino do classificador é o *dataset* disponibilizado no artigo científico Calzavara et al. (2015), com a adição de uma coluna que identifica o *encoding* utilizado no *cookie*, o que irá resultar numa melhoria dos resultados gerados. Como observamos na Figura 4 abaixo, é constituído por 7 colunas pertencentes ao conjunto de dados inicial e adicionada a coluna *encoding* que permite identificar alguns dos tipos de *encoding* utilizados no campo *value* do *cookie*.

	name	value	website	secure	httponly	javascript	authentication	encoding
216	user_id	XU3y5FynDNWI%2FJS0tsrfqa%2BWyB%2BFcPmBqUFxNRtT...	armorgames	0	0	0	1	base64
217	logged_in	30f52c002f65831ebe551b1753105a93	armorgames	0	0	0	1	md5
218	session_id	i0h4eo3vo08tb91ndfvr11hpc2	armorgames	0	1	0	1	PHPSESSID

Figura 4: Excerto do *Dataset* utilizado no treino do classificador

De forma mais detalhada o *dataset* é composto pelo seguintes campos:

- Name: identifica o nome do *cookie*, para o qual será associado um determinado valor formando assim o par(*name, value*)
- Value: Este parâmetro representa o valor para o qual foi mapeada a chave *Name*. Algumas implementações aplicam o *encoding* deste campo de forma a melhorar a segurança do *cookie*. O par (*name,value*) representam o conteúdo do *cookie* que será depois utilizado pelas aplicações *web*

- Website: Como o nome indica, este parâmetro representa o *website* onde foi gerado o *cookie*
- Secure: *Flag* identificadora que o *cookie* só deverá ser enviado ao servidor caso o pedido seja feito utilizando um protocolo seguro, nomeadamente, em *HTTPS* (HTTP Secure). Aumenta a proteção a algumas ameaças que tentam obter a informação do *cookie* por exemplo ataques *man-in-the-middle*
- HTTPOnly: *Flag* define que o *cookie* só deverá ser enviado recorrendo a um protocolo *HTTP*. Esta permite mitigar o risco de ataques de *cross-site scripting* acederem ao conteúdo do *cookie*
- Javascript: Determina se o *cookie* foi gerado através de código *Javascript*
- Authentication: Por fim, temos o valor que pretendemos estimar, ou seja, determinar se o *cookie* contém dados associados à autenticação do utilizador.
- Encoding: Este parâmetro identifica o *encoding* utilizado no valor do *cookie*. Existem 9 valores possíveis que este campo pode assumir, sendo eles, *MD5*, *SHA1*, *Base32*, *Base64*, *Hexadecimal*, *JWT*, *PHPSESSID*, *JSON* e *undefined*. Se o mecanismo de deteção de *encodings* não conseguir enquadrar o *encoding* encontrado em nenhum dos tipos desenvolvidos, irá preencher o campo com o valor *undefined*.

4.2 Pré-processamento de dados

Para classificar os *cookies* presentes no *dataset*, optamos por encarar este problema como um problema de classificação de texto. Como o nome e o valor do *cookie* não são números mas sim texto, é necessário fazer algum processamento para depois enviar ao classificador de *machine learning*. O pré-processamento feito inicialmente baseia-se no seguinte:

- O *Name*, *Value* e *Encoding* não necessitam de processamento porque já se encontram em formato de texto
- No caso do *Secure*, *HTTPOnly* e *Javascript*, estes têm valores binários (0 ou 1), logo vamos transformá-los em texto. Caso o valor destes parâmetros seja 1, substituímos o número 1 pelo nome da coluna, ou seja, *Secure*, *HTTPOnly* e *Javascript*. Por outro lado se o valor do parâmetro for zero, adicionamos a string "no"antes do nome da coluna
- Após a transformação em texto das *flags* mencionadas no item anterior, o próximo passo é juntar todos os parâmetros num só parâmetro, utilizando um espaço em branco entre eles. Cada *cookie* ficará com apenas duas colunas associadas. Uma será o texto resultado da junção dos 6 parâmetros, nomedamente, *Name*, *Value*, *Encoding*,

Flag Secure, *Flag HTTP ONLY* e *Flag Javascript* como observamos abaixo na figura 5. A segunda coluna será o resultado que queremos prever, ou seja, determina se é ou não um *cookie* de autenticação.

A Figura 5 destaca um diagrama com o exemplo da concatenação de dados que será feita. Consistindo na junção das 6 informações que temos sobre cada *cookie* em uma só variável.

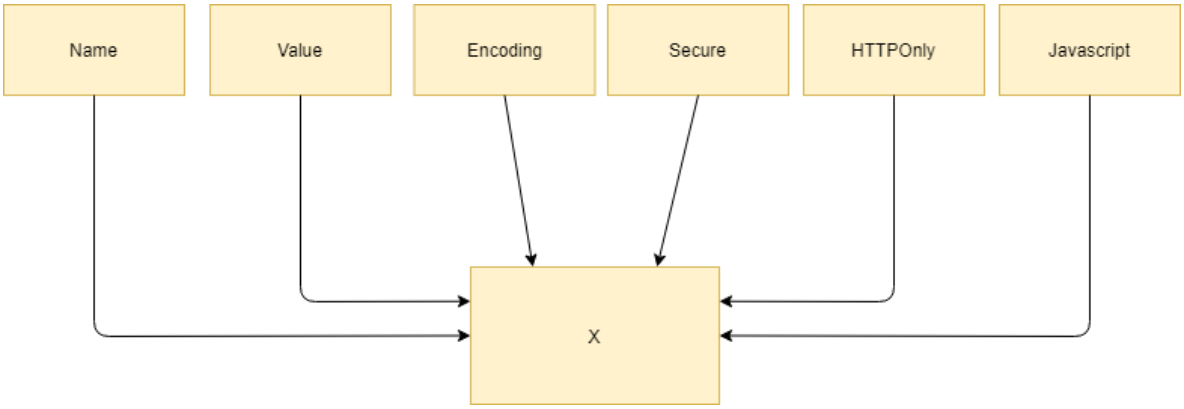


Figura 5: Concatenação dos parâmetros

Na figura 6 temos o exemplo de um *cookie* após aplicar o pré-processamento, como podemos observar, temos os 6 parâmetros separados pelo espaço em branco. As sequências de caracteres representam, da esquerda para a direita, o Nome do *cookie*, valor do *Cookie*, *encoding* detetado(neste caso não foi encontrado, logo coloca o *undefined*, *flag secure*, *flag httponly* e *flag javascript*. Neste caso as *flags* tinham o valor 0, logo colocamos o prefixo "no"antes do nome da coluna associada como já foi explicado.

```
'B b=C044684D819D9EFC undefined nosecure nohttponly nojavascript'
```

Figura 6: Exemplo de um cookie após o pré-processamento

4.3 Arquitetura do Classificador de cookies

Como já referido, para que o classificador funcione, este tem de receber números em vez do texto que temos neste momento em uma só variável. Para ultrapassar esta dificuldade, o método a utilizar consiste na vetorização do texto que temos neste momento, transformando o texto em números.

A vetorização tal como enunciado em Brownlee (2020), irá criar uma matriz em que as colunas serão todas as palavras encontradas nos *cookies* analisados. As linhas da matriz correspondem aos *cookies* em si, ou seja, cada linha é uma *cookie* presente no *dataset*.

Por exemplo, o valor da célula associada à linha do *cookie* com o nome i e à coluna da palavra com o nome j, corresponde ao número de ocorrências da palavra j no *cookie* i. Abaixo são apresentadas 3 frases, em que cada frase irá representar um *cookie*:

- Exemplo simples com gatos e rato
- Outro exemplo simples com cães e gatos
- Outro exemplo simples com rato e queijo

Considerando que para o nosso exemplo, cada frase acima representa um *cookie*, ao conjunto dos 3 *cookies* será aplicada a vetorização, transformando o texto em uma matriz que associa a cada *cookie* o número de ocorrências de cada palavra encontrada no conjunto total.

Na Tabela 1 temos o resultado da aplicação da técnica de vetorização de texto, tal como foi mencionado, nas colunas temos as palavras encontradas e nas linhas os respetivos *cookies* onde estas foram encontradas. Depois os valores correspondem ao número de ocorrências das palavras no respetivo *cookie*.

	e	**outro**	**gatos**	**queijo**	**cães**	**exemplo**	**rato**	**simples**	**com**
0	1	0	1	0	0	1	1	1	1
1	1	1	1	0	1	1	0	1	1
2	1	1	0	1	0	1	1	1	1

Tabela 1: Exemplo de vetorização de texto

Após a aplicação do vetorizador, o próximo passo será a aplicação de uma fórmula matemática conhecida como *Tf-Idf*. Com a matriz resultado obtida após a execução do vetorizador, vamos calcular a frequência relativa de cada palavra encontrada em cada *cookie* do *Dataset*. O valor da célula da matriz associada ao *cookie* e à palavra correspondente, representa a importância da ocorrência dessa palavra no *cookie*.

Como observamos na fórmula matemática abaixo, para calcular o *Tf-Idf* de um termo t em um documento (no nosso exemplo cada documento representa um *cookie*) d é aplicada a multiplicação do tf(t,d) pelo idf(t). O tf(t,d) representa o *term frequency*, ou seja, o número de ocorrências do termo t no documento d e o idf(t) é o *inverse document frequency* do termo t. Na expressão do idf(t), a letra n representa o número total de documentos, para o nosso caso é o número total de *cookies* e df(t) é o *document frequency*. O *document frequency* simboliza o número de documentos no conjunto total de documentos que contêm o termo t.

A fórmula apresentada abaixo é a fórmula utilizada pelo *scikit-learn* enunciada em scikit learn.

$$tf - idf(t,d) = tf(t,d) \cdot idf(t) \qquad idf(t) = log\frac{1+n}{1+df(t)} + 1$$

A cada célula da matriz da Tabela 1 será aplicada a fórmula *Tf-Idf* apresentada para calcular o valor correspondente. Este valor irá representar a importância de ocorrência de um determinado termo em um determinado *cookie* e varia entre 0 e 1. Quanto maior o valor do *Tf-Idf* maior a importância atribuída pelo mecanismo de *machine learning* a esse termo em questão. Esta métrica irá auxiliar o classificador de *cookies* a identificar os *cookies* de autenticação.

Para facilitar a compreensão da técnica explicada, apresentamos em abaixo, na Tabela 2, o resultado da aplicação do *Tf-Idf* aos resultados da Tabela 1.

	e	**outro**	**gatos**	**queijo**	**cães**	**exemplo**	**rato**	**simples**	**com**
0	0.0	0.000000	0.067578	0.000000	0.000000	0.0	0.067578	0.0	0.0
1	0.0	0.057924	0.057924	0.000000	0.156945	0.0	0.000000	0.0	0.0
2	0.0	0.057924	0.000000	0.156945	0.000000	0.0	0.057924	0.0	0.0

Tabela 2: Aplicação da fórmula matemática Tf-Idf ao exemplo da tabela 1

Se analisarmos a Tabela 1, as palavras que têm o valor 1 em todas as frases, ou seja, estão presentes em todas elas, são "e", "exemplo", "simples"e "com". Ao calcular o Tf-Idf estas palavras vão ter o valor 0 para todas as frases porque não são consideradas importantes dado que aparecem em todas elas. O valor 0 normalmente é associado às palavras que não têm influência na identificação do tema/sentido da frase, são geralmente conhecidas como *stop words*. Já os outros exemplos como "gatos", "cães", "queijo"e "rato", em algumas células têm um valor superior a 0 porque não aparecem em todas as frases, atribuindo-lhes alguma importância.

Para o nosso caso de estudo, as palavras presentes nos *cookies* que assumirem um valor de Tf-Idf superior serão as mais importantes na decisão do classificador de *machine learning*.

4.3.1 *Divisão do dataset*

Neste momento já temos uma matriz composta por números com a representação das diversas *features* encontradas nos *cookies* e os valores de importância correspondentes. Logo, a parte de processamento de dados para o treino do modelo de *machine learning* está completa.

O próximo passo será efetuar uma operação antes de enviar os dados ao classificador de *machine learning*. Esta é a divisão do *dataset* em dados de treino e teste. No treino e otimização dos hiperparâmetros do modelo será utilizada a *k-fold CV (Cross-Validation)* como explicado no capítulo 2.

De relembrar que os dados que estamos a tratar são desbalanceados, logo temos de garantir que os dados de treino e teste têm a mesma percentagem de elementos de cada uma das classes. Para isto, nos métodos de divisão do conjunto de dados disponibilizados pelo *scikit-learn* é possível garantir esta divisão em que os dados de treino/teste têm a mesma percentagem de elementos de ambas as classes utilizando o parâmetro *stratified*.

Nos dados de treino temos 256 *cookies* de autenticação (classe 1) e 1653 *cookies* pertencentes à classe 0. Quanto aos dados de teste registamos 86 *cookies* da classe 1 e 551 *cookies* da classe 0.

4.3.2 *Desbalanceamento*

Um problema encontrado ao longo do desenvolvimento do classificador foi o desbalanceamento do *dataset*. Para isso serão experimentadas algumas técnicas para contornar o problema, sendo elas o *Oversampling* e *Undersampling*.

Passando a explicar, o *oversampling* como enunciado em More (2016) consiste na maximização da classe em minoria, de maneira a que haja o mesmo número de elementos em ambas as classes do conjunto de dados.

No *undersampling* como enunciado em More (2016), a abordagem baseia-se na redução do número de elementos da classe maioritária, de forma a que, tal como no exemplo anterior existam o mesmo número de elementos em ambas as classes do *dataset*.

Para além destas técnicas, existem outras formas de lidar com o desbalanceamento do dataset como o uso de parâmetros de balanceamento de dados disponíveis nos classificadores de *machine learning*. Estes baseiam-se no balanceamento do peso da função de *loss* do classificador, ou seja, a penalização quando há uma falha na previsão de um elemento da classe minoritária é aumentada. Isto é feito através da multiplicação da função de *loss* pelos pesos inversamente proporcionais à dimensão das classes do *dataset*.

Para além do balanceamento da função de *loss* de acordo com as proporções do conjunto de dados, alguns classificadores como o *Naive Bayes* utilizam outra técnica. Esta técnica altera a prioridade das classes, por predefenição ambas têm uma prioridade de 0.5. Com este parâmetro podemos alterar a prioridade atribuída a cada classe entre 0 e 1, de forma a combater o desbalanceamento dos dados.

4.4 ARQUITETURA DO SISTEMA

O sistema será composto pelo detetor de *cookies* de autenticação, para além da classificação será feita uma análise do *encoding* utilizado no conteúdo do *cookie* de forma a alertar o analista de *QA* sobre a sua segurança. Este sistema será integrado no plataforma de testes de segurança *Burp Suite* funcionando como uma *extensão* para esta ferramenta.

Utilizando as ferramentas disponibilizadas pelo *Burp Suite* é possível interceptar requisições *HTTP*. Com base nisto, podemos recolher as *cookies* recorrendo à sua *API*. Após efetuar esta recolha, os *cookies* serão enviadas ao nosso modelo para efetuar a classificação e análisar a segurança do *encoding*.

O diagrama da figura 7 representa o diagrama de fluxo da arquitetura do sistema. Para que a extensão seja utilizada é necessário o carregamento da mesma no *Burp Extender*. Após este passo, a extensão estará pronta a ser utilizada, logo podemos avançar para a ativação do *scanner* do *Burp* em um *website*. Durante o *scan*, quando for encontrado um *cookie* este será enviado para o nosso classificador e analisador de *cookies* que irá informar ao *Burp* se o *cookie* é seguro. De seguida, há duas hipóteses possíveis, se o *cookie* não for vulnerável o *scanner* continuará a execução normalmente. Caso o *cookie* seja vulnerável, será reportada uma vulnerabilidade no *Burp*. Esta vulnerabilidade irá informar o utilizador da existência de um *cookie* inseguro na aplicação, indicando também a razão deste ser classificado como *inseguro* e algumas sugestões para resolver a vulnerabilidade.

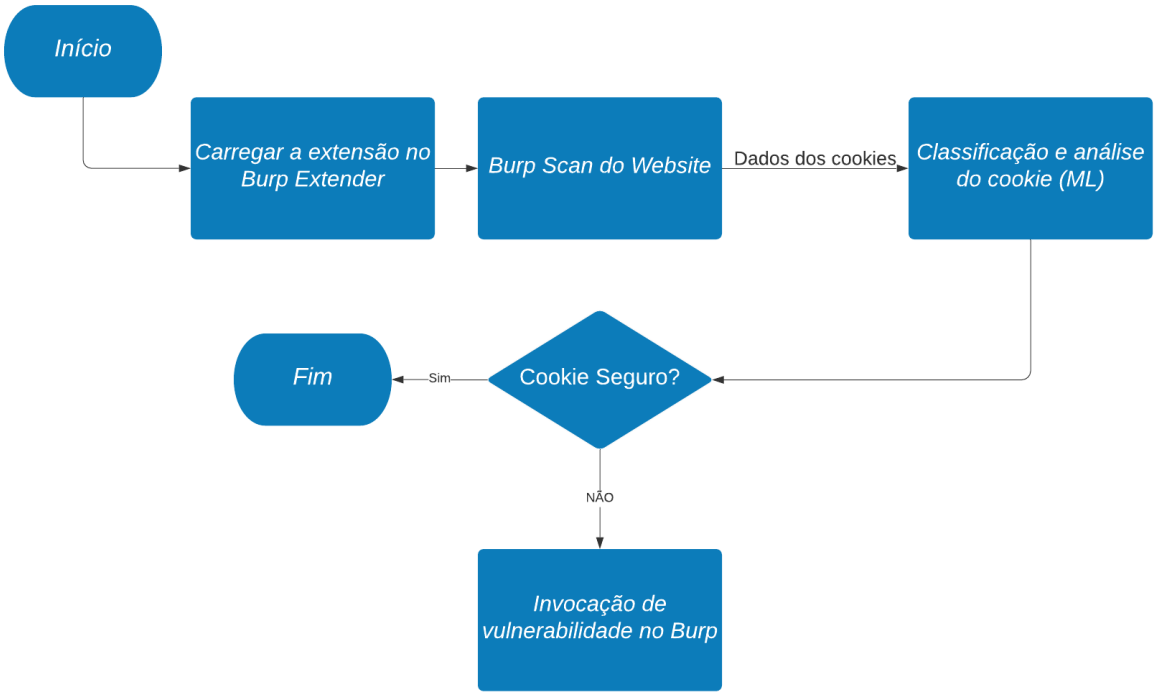


Figura 7: Diagrama de fluxo da arquitetura do sistema

5

DESENVOLVIMENTO

Após a pesquisa do estado de arte e elaboração da arquitetura do sistema, passamos agora ao desenvolvimento da ferramenta. Neste capítulo serão detalhados os procedimentos efetuados ao longo do desenvolvimento da extensão para o *Burp Suite*.

5.1 CLASSIFICADOR DE *cookies*

Nesta secção será apresentado o classificador utilizado e os resultados obtidos nas métricas de avaliação do mesmo. Será mostrada também uma comparação entre os resultados gerados com o conjunto de dados apresentado no capítulo 4 e com a adição de uma variável que identifica o tamanho do valor do *cookie*.

Para representar o tamanho do *cookie*, adicionamos uma *label* ao classificador que indica o tamanho do mesmo. Utilizando *label encoding*, associamos a esta *label* a sequência de caracteres **short** se o tamanho for inferior a 50 caracteres e caso seja superior aplicamos a expressão **long**. Após esta modificação voltamos a treinar o classificador e fazer a previsão com os dados de teste para obter os resultados das métricas de avaliação.

Ao longo fase de desenvolvimento foram testados vários classificados, nomeadamente, *SVM(support vector machine)*, *Random Forest* e dentro dos classificadores *Naive Bayes*, o *Gaussian*, *Multinomial* e *Bernoulli*.

O classificador que obteve melhores resultados com este conjunto de dados foi o *BernoulliNB*, este pertence à família de classificadores *Naive Bayes*. Estes são modelos probabilísticos que funcionam muito bem em conjuntos de dados compostos por *features* independentes e são classificadores indicados sobretudo para problemas de aprendizagem supervisionada. Outra particularidade, que se enquadra no nosso problema, é conseguirem retornar bons resultados quando treinados com poucos dados de treino.

O classificador *BernoulliNB* é muito popular na classificação de pequenos textos, sendo esta uma das razões que levou ao teste deste modelo.

Para encontrar os parâmetros ótimos a utilizar no classificador, invocamos uma função *grid search* que testa um conjunto de parâmetros e devolve aqueles que obtiveram melhores resultados.

Na Figura 8 podemos observar os parâmetros utilizados no algoritmo de *gridsearch*. O primeiro passo é a vetorização dos *cookies*, ou seja, a criação de uma matriz que represente o número absoluto de ocorrências de cada palavra encontrada em cada um dos *cookies*. De seguida é aplicada a fórmula *TF-IDF(term frequency–inverse document frequency)* que utilizando a matriz gerada, calcula a importância de cada palavra da matriz para o *cookie* em questão. Por fim, os dados são enviados ao classificador de *Machine Learning* e é treinado várias vezes com diferentes parâmetros estipulados no *gridsearch* com o objetivo de encontrar os valores ótimos.

Passando agora à explicação dos parâmetros definidos no *gridsearch*, o *ngram_range* é o número de *ngrams* que serão utilizados na 1º fase de vetorização dos *cookies*. Se o *ngram_range* for igual a (1,1) cada registo da matriz de vetorização irá conter apenas uma palavra, se o *ngram_range* for igual a (1,2) serão armazenados registos de sequências de uma e duas palavras. Basicamente, o primeiro número da variável *ngram_range* define o número mínimo de palavras em sequência que serão capturadas e o segundo número, o número máximo de palavras em sequência.

O *use_idf* define se o *Tf-Idf* irá calcular apenas a *Tf(term frequency* de cada termo ou se também procede à divisão do valor obtido pela *Idf(inverse document frequency)*. A *Tf(term frequency* consiste na transformação do valor absoluto para valor relativo de occorências de uma palavra num *cookie*. A *Idf(inverse document frequency)* irá calcular o peso de cada palavra presente nos *cookies* em relação ao conjunto total de *cookies*, por exemplo, uma palavra que aparece muitas vezes terá um peso menor do que outra que apareça raramente.

As duas últimas variáveis estão associadas ao classificador e otimizam o *alpha* utilizado e o valor de prioridade atribuir a cada classe (*class_prior*) para minimizar o desbalanceamento do *dataset*.

```
parameters = {
        'vect__ngram_range': [(1, 1), (1, 2),(1,3),(2,2),(3,3)],
        'tfidf__use_idf': [True,False],
        'clf__alpha': [0.0001,0.001,0.01,0.1,1],
        'clf__class_prior': [(0.1,0.9),(0.2,0.8),(0.3,0.7),(0.4,0.6)]
        }
```

Figura 8: Parâmetros utilizados no *gridsearch*

De seguida podemos observar na Figura 9 os parâmetros ótimos encontrados pelo *gridsearch* para aplicar ao classificador *BernoulliNB*.

```
{'clf__alpha': 0.01,
 'clf__class_prior': (0.1, 0.9),
 'tfidf__use_idf': True,
 'vect__ngram_range': (1, 1)}
```

Figura 9: Parâmetros ótimos *gridsearch*

Nas Figuras 10 e 11 apresentamos as matrizes de confusão obtidas utilizando, respetivamente, os dados de treino com o atributo do tamanho do *cookie* e os dados de treino sem esse atributo. Antes de iniciar a análise dos resultados apresentados nas matrizes de confusão, convém mencionar que a matriz de confusão do lado esquerdo é gerada utilizando uma divisão de dados de 70% para treino e 30% para teste enquanto que na matriz do lado direito a divisão é de 75% treino e 25% teste. Após a divisão em dados de treino/tese foi aplicado o algoritmo de otimização de hiperparâmetros *gridsearch* e o gerador de *cross-validation* utilizado foi o *StratifiedKFold* com 5 *folds*. A matriz do lado esquerdo é composta por 764 *cookies* e do lado direito são 637. Foram experimentadas várias percentagens na divisão dos dados, foram escolhidas aquelas que permitiram obter um melhor desempenho.

A métrica mais significativa é a diminuição dos falsos negativos, ou seja, evitar que *cookies* de autenticação não sejam classificados como tal. Como observamos na imagem, apesar da maior quantidade de dados do lado esquerdo, o número de falsos negativos é menor(16) do que do lado direito indicando uma notável melhoria neste indicador.

A nível de verdadeiros positivos temos do lado esquerdo 87 e 68 do outro lado, é uma boa indicação que o número de *cookies* de autenticação bem classificados tenham aumentando.

Quanto aos verdadeiros negativos os valores são 553 do lado esquerdo e 473 do lado direito, o número também aumentou após as alterações como seria de esperar dado que o conjunto de dados também é superior.

Por fim, os falsos positivos são 108 e 78, respetivamente do dado esquerdo e direito, houve um elevado aumento neste indicador. Esta alteração deve-se à prioridade superior que foi atribuída aos *cookies* de autenticação, ou seja, a minimização das falhas na classificação dos *cookies* de autenticação(falsos negativos).

De qualquer forma, como o objetivo é minimizar o números de *cookies* de autenticação não classificados como tal, esta alteração foi claramente positiva permitindo uma melhoria nos resultados das métricas de avaliação que serão apresentadas.

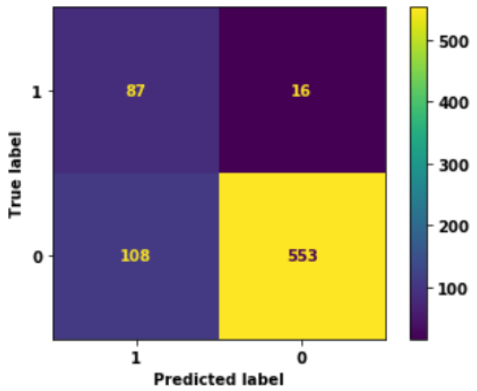


Figura 10: Matriz de confusão utilizando os dados de treino com o tamanho do *cookie*

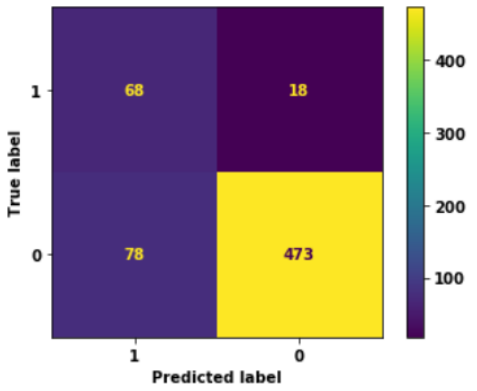


Figura 11: Matriz de confusão utilizando os dados de treino sem o tamanho do *cookie*

Nas Figuras 12 e 13 observamos as *ROC curves*, na figura do lado esquerdo temos a curva do classificador desenvolvido utilizando a variável tamanho do *cookie* no conjunto de dados e do lado direito o classificador sem essa variável. As *ROC (Receiver Operating Characteristic) curves* são uma ferramenta útil na previsão das probabilidades de um classificador binário retornar um dado valor. Esta ferramenta permite estabelecer uma comparação entre vários modelos através da métrica *AUC (area under the curve)* que indica de forma resumida o desempenho do classificador. Os eixos de um gráfico *ROC* representam a taxa de falsos positivos (eixo do x) e a taxa de verdadeiros positivos (eixo do y). A taxa de verdadeiros positivos é o mesmo que a métrica *Sensitivity* apresentada no capítulo 3 e a taxa de falsos positivos é o resultado da subtração entre 1 e a métrica *Specificity* também apresentada em 3.

Em termos práticos, se forem verificados valores baixos no eixo do x concluímos há um baixo número de falsos positivos e elevado número de verdadeiros negativos. Caso os valores do eixo do y sejam altos então o número de verdadadeiros positivos é alto e os falsos negativos são baixos.

Para além desta relação entra a sensibilidade e a especificidade, temos no gráfico como foi mencionado a *AUC* para avaliar o modelo, esta varia entre 0 e 1 e o objetivo é maximizar este valor. Quanto maior o valor da *AUC* melhor será o desempenho do classificador binário desenvolvido. Depois, é apresentada também a métrica *F-measure* que tal como explicado no capítulo 3 consiste na média harmónica entre a sensibilidade e a especificidade. Como tal, o valor desta métrica irá escalar de acordo com as alterações na sensibilidade e especificidade sendo que o propósito é maximizar esta métrica.

Passando agora à comparação entre os gráficos da Figura 12 e 13, os valores de *AUC* e *F-measure* são muito semelhantes. Após a adição da variável que representa o tamanho do valor do *cookie*, a *AUC* diminuiu 0.1 e a *F-measure* aumentou 0.2. Como o nosso objetivo é maximizar principalmente os valores de *F-measure* consideramos positivas as alterações efetuadas no classificador.

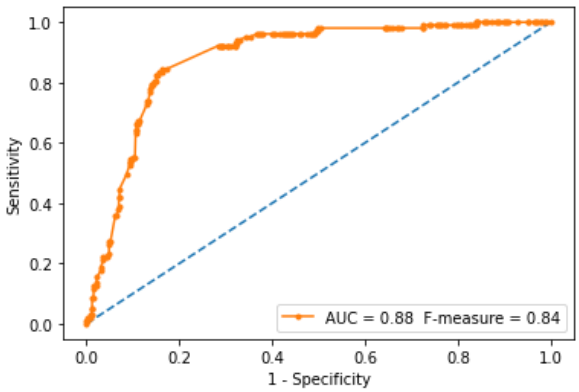


Figura 12: *ROC curve* do classificador utilizando os dados de treino com o tamanho do *cookie*

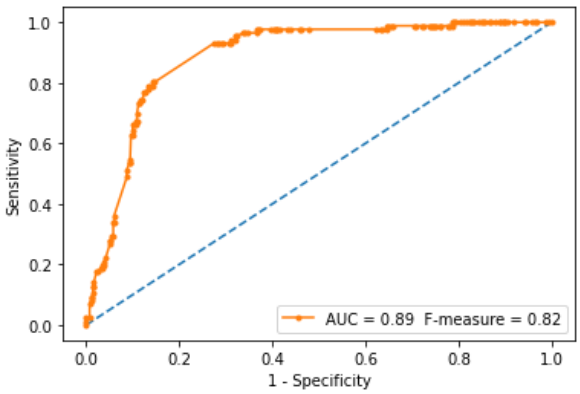


Figura 13: *ROC curve* do classificador utilizando os dados de treino sem o tamanho do *cookie*

Na Tabela 3 são apresentados de forma detalhada os valores das métricas de avaliação *Sensitivity*, *Specificity* e *F-measure* obtidos com o classificador desenvolvido nesta dissertação sem a variável representativa do tamanho do *cookie*, com essa mesma variável e por fim os resultados do melhor classificador desenvolvido no artigo Calzavara et al. (2015).

A primeira coluna da tabela representa os resultados do classicador *BernoulliNB* utilizando o conjunto de dados sem a variável do tamanho do *cookie*, a segunda coluna é o classificador *BernoulliNB* utilizado o conjunto de dados com a variável tamanho do *cookie* e a terceira coluna é a *Random Forest* desenvolvida pelo artigo apresentado no capítulo 3 dado que este foi o classificador que obteve melhores resultados no artigo.

A sensibilidade é a métrica que representa dentro dos *cookies* de autenticação qual a percentagem que foi identificada corretamente, ou seja, estabelece uma relação entre o número de verdadeiros positivos e falsos negativos. A especificidade é a métrica que representa dentro dos *cookies* que não são de autenticação qual a percentagem que foi identificada corretamente, ou seja, estabelece uma relação entre o número de verdadeiros negativos e falsos positivos.

Para o nosso exemplo é mais crítica uma falha na classificação de um *cookie* de autenticação (falso negativo) do que classificar um *cookie* como sendo de autenticação e este não estar associado à autenticação (falso positivo). Logo, o mais importante é maximizar a *F-measure* e a sensibilidade do que a especificidade.

Comparando os resultados da primeira e segunda coluna da tabela, observamos uma elevada melhoria na sensibilidade, uma leve descida na especificidade e uma subida na *F-measure* dado que o aumento da sensibilidade foi mais acentuado do que a descida na especificidade. Por fim, como a meta é a maximização da sensibilidade (diminuição dos falsos negativos) e consequente *F-measure* consideramos que as alterações efetuadas no classificador foram positivas.

Relativamente à comparação entre o nosso classificador final e ao classificador desenvolvido pelos autores do artigo, constatamos que o nosso classificador tem um valor de sensibilidade inferior em 0.04, quanto à especificidade está 0.04 acima e a métrica de avaliação global *F-measure* é superior em 0.01. Os resultados são bastante parecidos mas de forma global o nosso classificador tem um desempenho ligeiramente superior porque a métrica de *F-measure* obtida pelo nosso modelo é mais elevada.

	BernoulliNB(Antes)	**BernoulliNB(Depois)**	**RandomForest(Estado de Arte)**
Sensitivity	0.79	0.84	0.88
Specificity	0.86	0.84	0.80
F-measure	0.82	0.84	0.83

Tabela 3: Comparação do desempenho do classificador desenvolvido com o Estado de Arte

O gráfico da Figura 14 representa os resultados das métricas de classificação obtidos em diversos modelos experimentados pelos autores do artigo Calzavara et al. (2015) enunciado no Estado de Arte. Os valores de *F-measure* apresentados no artigo variam entre os 0.82-0.83 dependendo do classificador de *Machine Learning* utilizado. Quanto ao nosso classificador de *cookies*, o melhor valor de *F-measure* reproduzido é 0.84 como demonstrado na Tabela 3.

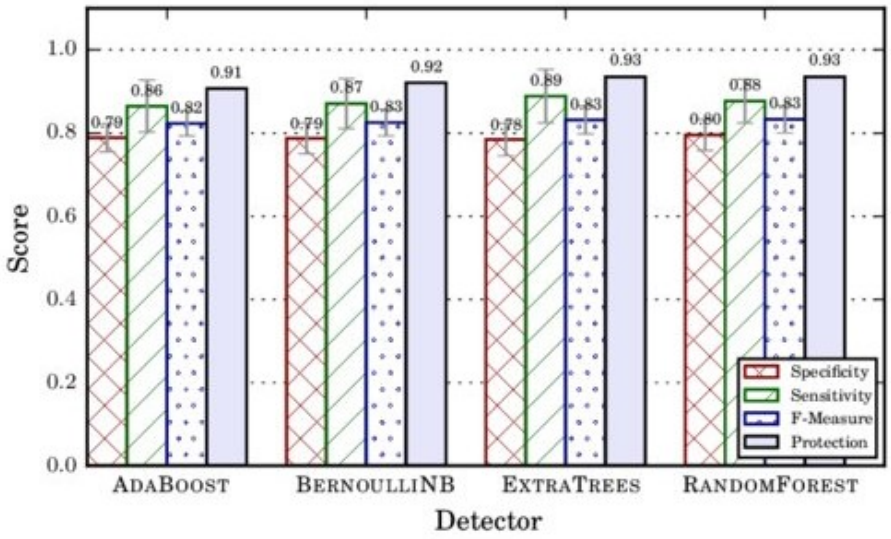


Figura 14: Resultado das métricas de avaliação no classificador desenvolvido pelos autores do artigo Calzavara et al. (2015)

Na Tabela 4 são apresentados os resultados da métrica *AUC* para os classificadores apresentados ao longo do artigo. A primeira linha da tabela representa o valor da *AUC (Area under the Curve)* para o classificador *BernoulliNB* desenvolvido sem a utilização da *feature* tamanho do *cookie*, o valor da *AUC* para o *BernoulliNB* utilizando esta *feature* que indica o tamanho do campo *value* do *cookie* e a última coluna é o resultado da *AUC* para o classificador desenvolvido no artigo Calzavara et al. (2015) apresentado no Estado de Arte.

Como foi explicado no capítulo 2, a *AUC* permite avaliar o desempenho do classificador de *machine learning* sendo que os resultados desta métrica não são afetuados pelo desbalanceamento do *dataset*. Desta forma, podemos comparar os classificadores utilizando a *AUC*. Como observarmos na Tabela 4, os classificadores *BernoulliNB* têm um valor de *AUC* superior, logo têm um melhor desempenho que o *RandomForest*.

	BernoulliNB(Antes)	**BernoulliNB(Depois)**	**RandomForest(Estado de Arte)**
AUC	0.89	0.88	0.85

Tabela 4: Comparação da *AUC* dos classificadores desenvolvidos com o Estado de Arte

5.2 DETEÇÃO DE ENCODINGS

Nesta secção serão apresentados os tipos de *encoding* detetados pelo nosso detetor de *cookies*. Na Figura 15 temos o conjunto de *encodings* detetados pela nossa extensão e na Figura 16, as expressões regulares para identificar cada um dos *encodings* apresentados.

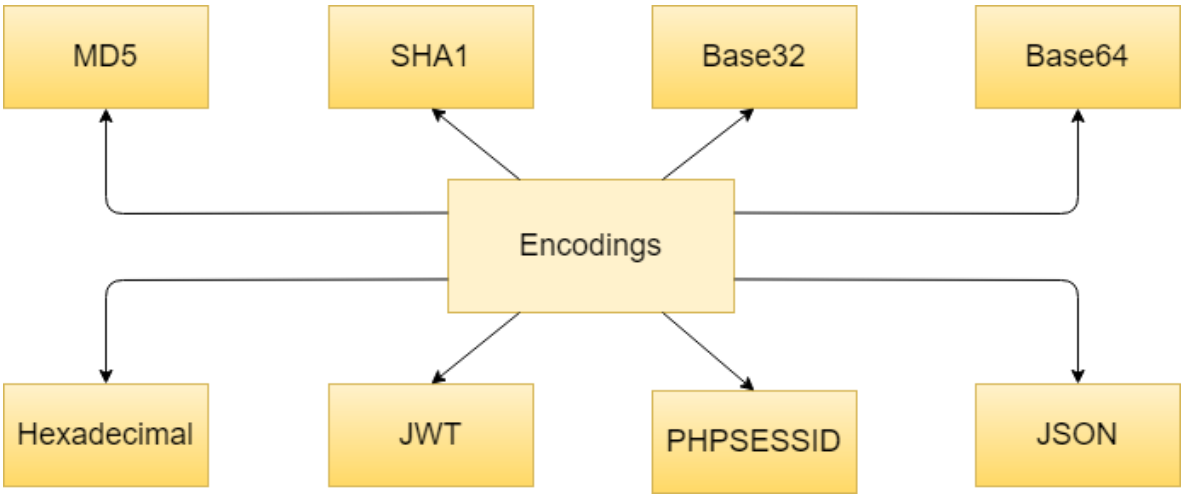


Figura 15: Conjunto de encodings detetados pela extensão

```
#Expressões regulares associadas a cada encoding
MD5 = r"[0-9A-Fa-f]{32}"
SHA1 = r"[0-9A-Fa-f]{40}"
BASE32 = r"^(?:[A-Z2-7]{8})*(?:[A-Z2-7]{2}={6}|[A-Z2-7]{4}={4}|[A-Z2-7]{5}={3}|[A-Z2-7]{7}=)?$"
BASE64 = r"^([A-Za-z0-9+/]{4})*([A-Za-z0-9+/]{3}=|[A-Za-z0-9+/]{2}==)?$"
HEXADECIMAL = r"[0-9a-fA-F]+"
JWT = r"[^.]+\.[^.]+\.[^.]*"
PHPSESSID = r"[0-9a-z]{26}"
```

Figura 16: Expressões regulares para identificação do encoding

Agora, serão explicadas as expressões regulares da Figura 16 para deteção de cada um dos *encodings*:

- MD5 - Para detetar o *encoding MD5* é aplicada a expressão regular que verifica se os caracteres do campo *value* do *cookie* são todos hexadecimais e se o comprimento deste campo é 32.
- SHA1 - Para o *SHA1* a verificação da expressão regular é semelhante ao *MD5*, verificando se os caracteres são todos hexadecimais e se o comprimento do campo *value* é de 40 caracteres.
- Base32 - A expressão regular do base32 analisa o número de caracteres que compõem o *cookie*, dado que este tem de ser múltiplo de 8. Analisa também a gama de caracteres

utilizada porque os caracteres permitidos neste *encoding* são os números de 2-7, letras de A-Z e o caracter "="que é o *padding* do base32 como explicado no capítulo 2.

- Base64 - No base64 é feita a verificação do número de caracteres que compõem o *cookie* porque tem de ser um número múltiplo de 4. Os caracteres permitidos neste *encoding* são as letras de A-Z, número de 0-9 e os caracteres "+", "/", e "=". O caracter "="é o caracter de *padding* sendo que um *encoding* base64 pode terminar com "=", "=="ou com uma letra/número se o comprimento for múltiplo de 4, não é necessário o "=".

- Hexadecimal - A verificação do hexadecimal consiste apenas na verificação se todos os caracteres do campo *value* do *cookie* pertencem ao sistema de numeração hexadecimal, ou seja, letras de A-F e número de 0-9.

- JWT - O *JWT (JSON Web Token)* está dividido em três campos, *header*, *payload*, e *signature* separados pelo caracter ".". Com base nesta informação, a nossa expressão regular fará a verificação dos dois/três primeiros campos de forma a comprovar se o texto corresponde a um JWT. Confirmamos apenas a existência dos 2/3 campos separados pelo caracter ".", dependendo se o *JWT* utiliza algum algoritmo criptográfico para assinatura do *token*. Se não utilizar nenhum algoritmo criptográfico, o último campo(*signature*) estará vazio logo apenas verificamos os dois primeiros campos(*header* e *payload*). Caso utilize um destes algoritmos no *JWT* estarão contidos os 3 campos logo temos de os verificar.

- PHPSESSID - A expressão do *PHPSESSID* verifica se o texto do campo *value* possui um comprimento de 26 caracteres e se estes caracteres são números de 0-9 ou letras de a-z.

- JSON - Para o caso do *JSON*, a expressão regular não consta na Figura 16 mas consiste na verificação se o campo *value* do *cookie* começa com o caracter "{"e termina com o caracter "}"como é típico em estruturas *JSON*.

Caso o *encoding* do *cookie* não se enquadre em nenhum dos tipos apresentados, este será classificado como *undefined*.

5.3 SEGURANÇA DOS ENCODINGS DETETADOS

Um dos objetivos, para além da classificação e deteção do *encoding* dos *cookies*, é a análise da segurança dos *encodings* e posterior categorização do *cookie* como seguro ou inseguro. Será agora explicado detalhadamente o processo de classificação de cada *encoding*:

- MD5 - Como já é conhecido, o *hashing MD5* é considerado criptograficamente inseguro pela rapidez com que um atacante pode descobrir a *password* associada à *hash*. Existem atualmente bases de dados com milhões de correspondências entre *passwords* e *hash MD5*, sendo muitas das vezes possível com apenas um *lookup* desencriptar a informação. Considerando um exemplo em que o campo *value* do *cookie* contenha uma *password* encriptada com *hashing MD5*, um atacante utilizando algo como um ataque *man-in-the-middle*, obtinha toda a informação do *cookie* como enunciado em Sotirov (2012) e Sasaki et al. (2007). Depois, poderia conseguir decifrar a password enviada no *cookie* recorrendo a uma destas bases de dados com chaves *MD5* mapeadas. Pela razões explicadas optamos por classificar os *cookies* com *encoding* MD5 como *cookies* inseguros.

- SHA1 - O *SHA1* como o *MD5*, é um algoritmo de hashing mas contém um nível de complexidade superior. A chave *SHA1* tem um comprimento maior, são 160 *bits* em vez dos habituais 128 do *MD5* tornando o algoritmo mais lento e aumentando a dificuldade da descodificação *brute force* por parte do atacante como enunciado em Ratna et al. (2013). Como a chave é maior, o número de combinações possíveis é também maior levando a que o número de operações necessárias seja superior. Quando é encontrado um *cookie* que utilize este *encoding* é classificado como *cookie* seguro.

- Base64 - O base64 é um método muito utilizado para codificação de dados, permite transformar qualquer caracter em uma sequência de caracteres contendo uma gama de 64 caracteres possíveis. Uma codificação base64 "normal"é *two-way*, ou seja, é possível fazer o encode base64 de um texto e depois fazer o *decode* para o texto original. Caso seja utilizada alguma encriptação antes da aplicação do encode base64, não será possível fazer o decode do base64 para texto claro, ou seja, neste caso a aplicação deste *encoding* será considerada segura. Quando é encontrado um *cookie* com *encoding* base64 é aplicada uma verificação que indica se é possível descodificar o *encoding* de forma a consultar os dados em texto claro. Um exemplo que não permite a conversão para texto claro é quando o *encoding* base64 contém um salto porque funciona como chave necessária para aplicar a conversão. Por outro lado, caso seja possível a descodificação para texto claro, o *cookie* é classificado como *cookie* inseguro. Se não for possível descodificar então o *cookie* é seguro.

- Base32 - O prodecimento para o base32 é exatamente igual ao base64, a diferença é que a gama de caracteres disponíveis no base32 é menor. É feito o mesmo teste do exemplo

anterior, se for possível descodificar a informação para texto claro é classificado como *cookie* inseguro. E caso contrário, será um *cookie* seguro.

- Hexadecimal - Para os *encodings hexadecimais*, a verificação consiste na tentativa de descodificação dos dados para texto claro. Caso seja possível consultar a informação em texto claro, o *cookie* é classificado como inseguro. Para os exemplos como no base64, em que são aplicados algoritmos de encriptação como o *AES* e depois são guardados os resultados em hexadecimal. Não é possível converter para texto claro logo estes exemplos são classificados como *cookies* seguros.

- JWT - Para o *JWT(JSON Web Token)*, são aplicados prodecimentos para verificar se o campo "*alg*"no *header* do *token* utiliza um algoritmo válido e se os *encodings* utilizados no *payload* são seguros . Constatámos que há alguns *JWT* em que no campo *algorithm* do *header* do *token* não utilizam nenhum algoritmo de encriptação logo este campo é preenchido com *"None"*, a nossa extensão fará a verificação deste campo. Se o algoritmo não for válido o *cookie* é classificado como inseguro. O segundo passo é a análise dos *encodings* utilizados no *payload* do *JWT*. Cada campo é enviado ao nosso detetor de *encodings* e sujeito a uma classificação que indica se o *encoding* utilizado é seguro ou inseguro. Caso seja encontrado algum campo do *payload* com um *encoding* inseguro, o *cookie* é imediamente classificado como inseguro. Por fim, se o *token* em questão, passar nestes 2 testes apresentados, estamos na presença de um *cookie* seguro.

- PHPSESSID - O *PHPSESSID* é um tipo específico de *encoding* utilizado em *cookies* gerados pela linguagem *PHP*. Como já foi explicado, consistem em identificadores de sessões de utilizador composto por uma sequência de 26 caracteres. Quando é encontrado um *cookie* com este tipo *encoding*, é classificado como *encoding* seguro.

- JSON - Tal como foi explicado na secção anterior, surgiram exemplos de *cookies* que utilizam estruturas da linguagem *JSON* dentro do campo valor do *cookie*. Para este *encoding*, o protocolo consiste na avaliação de cada campo da estrutura *JSON*. Cada campo é enviado ao nosso detetor de *encodings* e sujeito a uma classificação que indica se o *encoding* utilizado é seguro ou inseguro. Por exemplo, se um *cookie* com *encoding JSON* incluir um campo com uma *password* encriptada com *hashing MD5* será imediatamente classificado como inseguro. Resumindo, todos os campos são analisados e se algum destes for classificado como inseguro, o *cookie* é também inseguro. Pelo contrário se todos forem seguros, o *cookie* é também seguro.

- Undefined - Esta categorização é utilizada quando o *encoding* não se enquadra em nenhum dos tipos disponíveis na nossa ferramenta. Para estes, é aconselhável o utilizador da nossa extensão verificar manualmente o *cookie*. Por isso, é classificado como inseguro para serem realizadas mais verificações.

5.4 INTEGRAÇÃO COM O BURP SUITE

Após a criação do classificador de *cookies* de autenticação e do detetor de *encodings*, resta agora fazer a integração destas ferramentas com o *Burp Suite*. A integração foi feita recorrendo à *API* do *Burp Extender*, utilizando os seguintes módulos:

- IBurpExtender - Este é o módulo padrão do *Burp Extender*, é invocado quando a extensão é carregada no *Burp*, por isso todas as extensões têm de o utilizar. Contém a função *registerExtenderCallbacks* que será responsável pela invocação das várias ações que a extensão vai executar.
- IBurpExtenderCallbacks - Os *extenderCallbacks* contêm o corpo da extensão, aqui é definido o nome da extensão e são invocadas todas as ações a executar. No nosso exemplo, nesta secção é invocado o nosso scanner para verificação dos *cookies*.
- IExtensionHelpers - Este módulo do *Burp* é utilizado para facilitar algumas tarefas como o tratamento de mensagens *http* capturadas pelo *proxy* do *Burp*. Disponibiliza sobretudo, funções para auxiliar nas tarefas que surgem ao desenvolver uma extensão no *Burp Extender*.
- IScannerCheck - O registo de um *scanner check* permite customizar o *scanner* do *Burp* e reportar as vulnerabilidades encontradas. Para o nosso caso, no *scanner*, foi adicionada uma verificação adicional dos *cookies* e criado um relatório personalizado para apresentar quando é encontrado um *cookie* vulnerável.
- IScanIssue - O *Burp* tem um conjunto de *issues*/vulnerabilidades definidas que podem ser apresentadas nos relatórios do *scanner*. Este módulo permite a criação de *issues*/vulnerabilidades personalizadas para além das já definidas pelo *Burp*. No nosso exemplo, recorrendo a esta funcionalidade foi criado um *issue* personalizado para apresentar de acordo com o *cookie* encontrado.

Surgiram alguns problemas na integração com o *Burp Suite* porque este é totalmente desenvolvido na linguagem *java*, enquanto que o código da ferramenta criada para classificação de *cookies* foi escrito na linguagem *python* causando algumas incompatibilidades. Para permitir a compilação de código *python*, o *Burp Extender* disponibiliza o *jython* que é um compilador responsável pela conversão do código para *Java*. O *jython* não permite resolver todos os problemas de compatibilidade porque ainda utiliza a versão 2.7 do *python* levando a que não suporte algumas bibliotecas de *Machine Learning* como o *numpy* e o *scikit-learn*.

A solução encontrada foi a execução do classificador de *cookies* fora do *Burp Extender*, ou seja, localmente num compilador *python3*. Foi desenvolvido código para estabelecer a comunicação entre os *cookies* capturados pelo código executado no *jython* e a ferramenta

de análise dos *cookies* executada num compilador *python3* tradicional.É utilizada a função *check_output* do módulo *subprocess*, criando um subprocesso que executa localmente no compilador *python* o ficheiro de classificação de *cookies* e análise do *encoding*. Esta irá permitir ao código da extensão do *Burp*, enviar o *cookie* capturado e aceder aos resultados gerados pela execução da ferramenta de análise de *cookies* tendo como input o *cookie* enviado. Podemos observar na Figura 17 abaixo a comunicação entre o código da extensão do *Burp* executado no compilador *jython* e o código da ferramenta de análise executado no *python*.

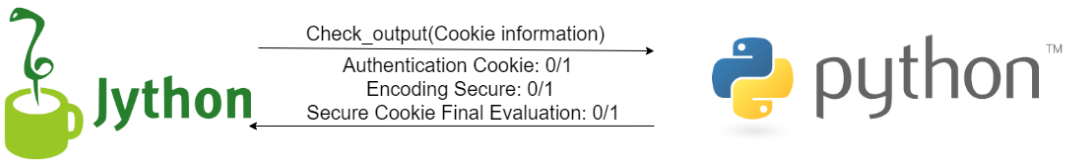


Figura 17: Comunicação entre o código compilado no *jython* e no *python*

Após o desenvolvimento do código *python* para classificação de *cookies*, análise do *encoding* e o protocolo de comunicação com o *Burp*. É agora necessário efetuar o carregamento do código *python* da extensão no *Burp Extender* para poder executar o *scan* personalizado.

Como observamos na Figura 18, dentro do *Burp Extender* podemos selecionar a opção *Add* para adicionar extensões ao *Burp*. Ao carregar nesta opção, será gerada uma janela onde iremos selecionar o nosso ficheiro *python* da extensão para ser executado no compilador. Após o carregamento, podemos utilizar as funcionalidades disponibilizadas pela extensão.

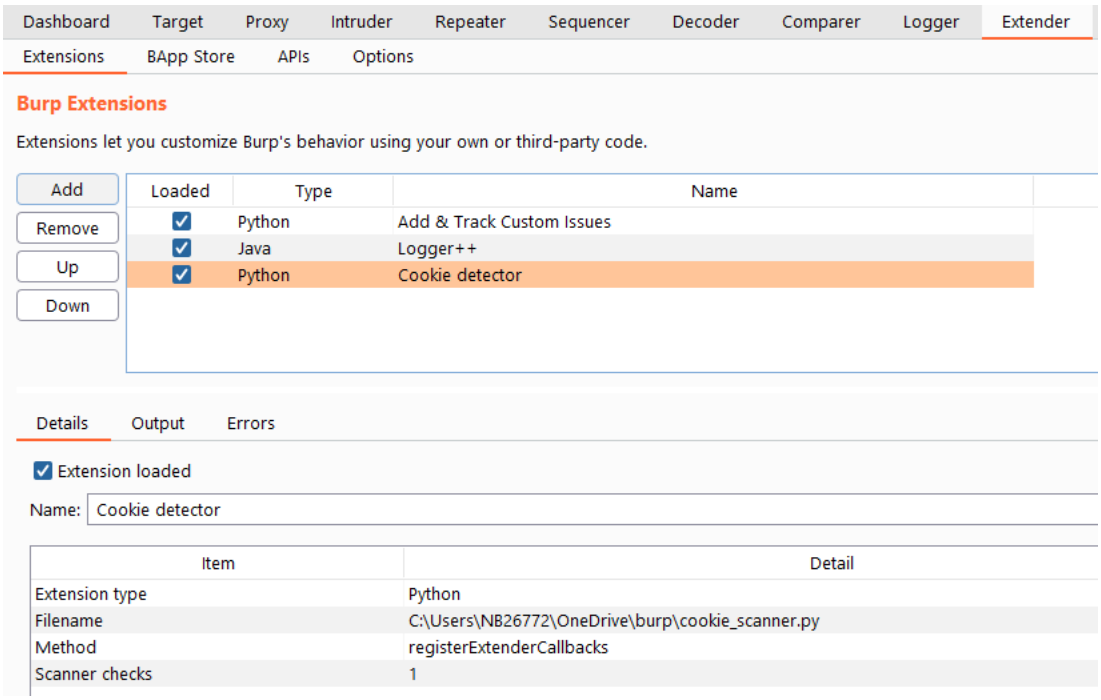


Figura 18: Carregamento da extensão no *Burp Extender*

5.5 SONARQUBE

Para análise do código desenvolvido foi utilizado o *SonarQube*, nesta secção serão apresentados os resultados para diversas métricas de avaliação.

Como podemos observar na Figura 19 da evolução temporal de *issues* no código, temos 3 cores, uma para cada *issue*, representado os *bugs*, *code smells* e *vulnerabilidades*.

Começando pelos *bugs*, ao longo do desenvolvimento do código nunca foi detetado nenhum *bug* pelo *Sonarqube*. Quanto aos *code smells*, como seria de esperar, à data do 1º upload do código no repositório foi quando o número de *code smells* foi mais elevado, alcançando cerca de 48. Depois foram todos resolvidos e o número baixou para os zero mantendo-se assim algum tempo até uma data em que surgiram cerca de 28. Estes foram novamente corrigidos e voltaram ao valor zero. Por fim, o número de vulnerabilidades foi constante, tomando o valor zero ao longo de toda a evolução.

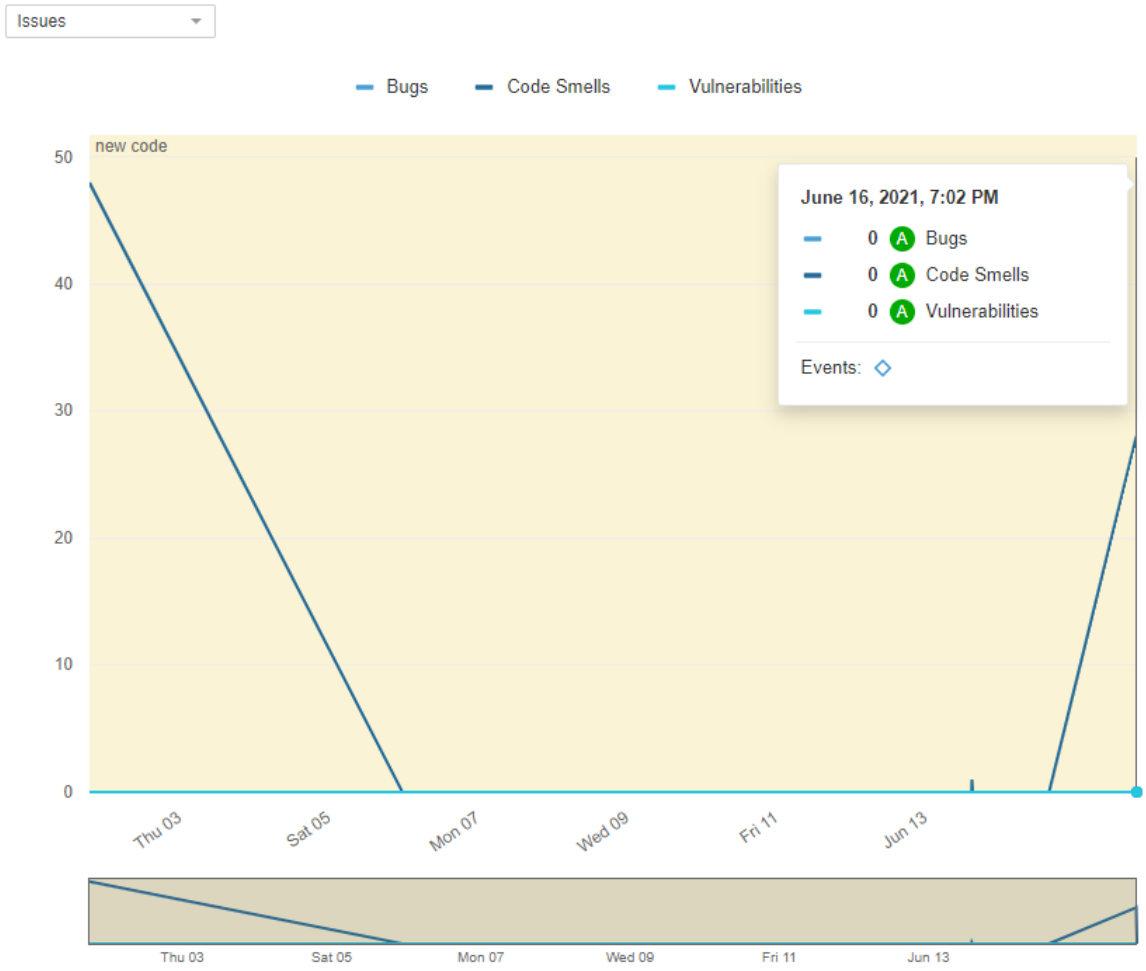


Figura 19: Evolução temporal de *issues* no código segundo o *SonarQube*

A Figura 20 representa a *coverage* do código, ou seja, dentro do número de linhas de código quais as que estão cobertas por testes unitários como explicado no capítulo 2. Temos em um tom de azul claro o número total de linhas e em um azul mais escuro o número de linhas cobertas pelos testes unitários.

Na 1º parte do gráfico, do lado esquerdo, o número de linhas são cerca de 300 e o número de linhas cobertas é zero porque ainda não tinham sido desenvolvidos os testes unitários. Depois o centro do gráfico está vazio porque foi um período em que não houve alterações no código. Por fim, na parte final do gráfico, observamos uma subida no número total de linhas de código do projeto e um elevado aumento no número de linhas cobertas pelos testes unitários.

Esta rápida subida indica o período em que foram adicionados os testes unitários ao código desenvolvido. Como análise final, constatamos que existem atualmente, 429 linhas de código e 354 destas linhas estão cobertas por testes unitários, contabilizando uma taxa de cobertura de cerca de 83%.

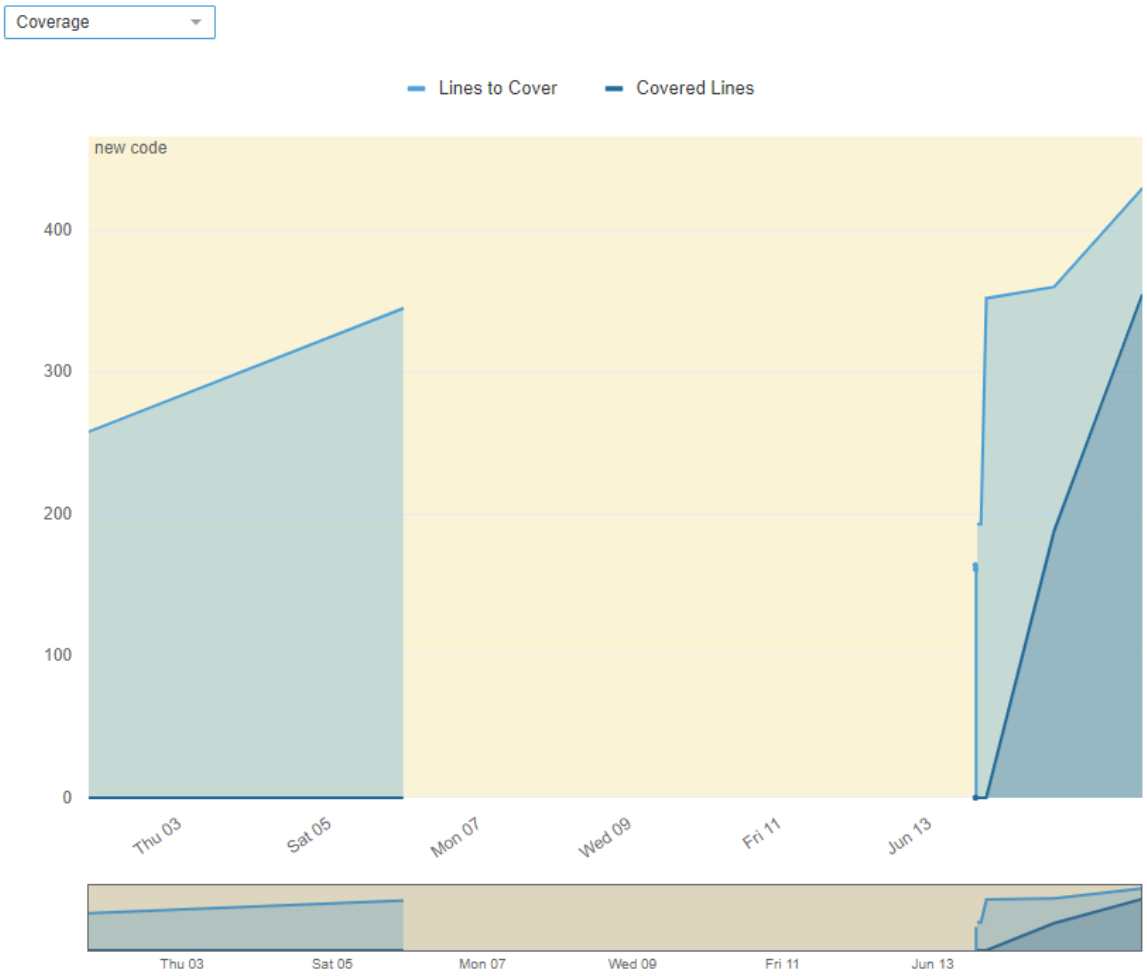


Figura 20: Evolução temporal da *coverage* do código segundo o *SonarQube*

O gráfico da Figura 21 acima representa a duplicação de código face ao número total de linhas escritas. Num tom de azul mais claro temos o número total de linhas de código e a azul mais escuro o número de linhas duplicadas.

Como podemos verificar rapidamente, o número de linhas duplicadas é sempre o mesmo, ou seja, zero enquanto que o número de linhas de código totais vai aumentando ao longo do tempo.

Por fim, aquela faixa central do gráfico por preencher entre o dia 6 e o dia 14, tal como explicado no exemplo da Figura 21, representa a interrupção temporal no desenvolvimento do código da ferramenta.

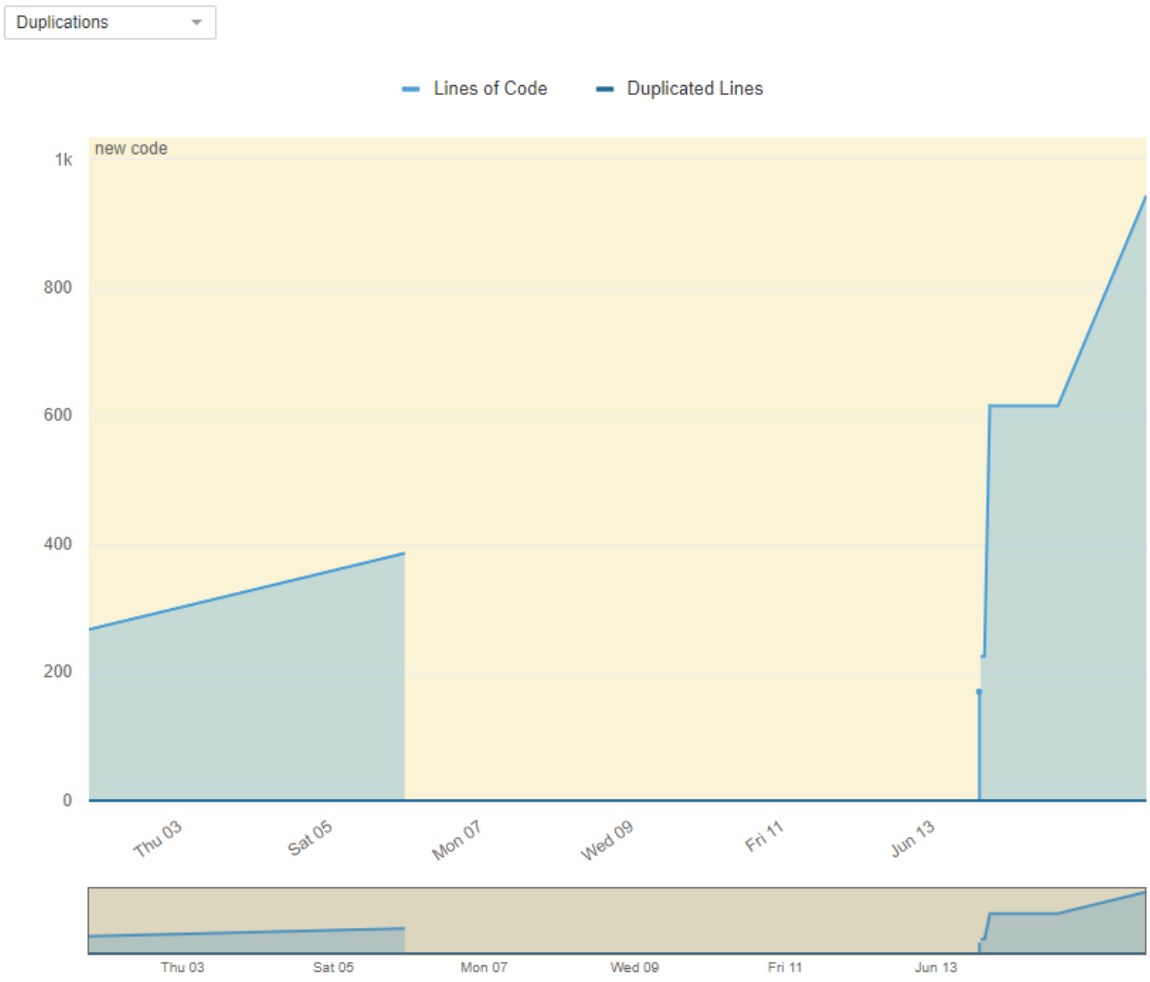


Figura 21: Evolução temporal do número de linhas e duplicação de código segundo o *SonarQube*

Por fim, na Figura 22 observamos a *interface* do *SonarQube* que representa a análise geral das métricas mais significativas.

Como já foi referido, atualmente, o número de *bugs*, *vulnerabilities* e *code smells* é zero. Quanto à percentagem de *coverage* do código, ou seja, a percentagem de código que está coberto por testes unitários é de 82.5%. A percentagem de duplicação de código é de 0%. Em conclusão, para as métricas *Reliability*, *Securiy*, *Security Review* e *Maintainability* a nota atribuída foi A, dentro de uma escala de A a E.

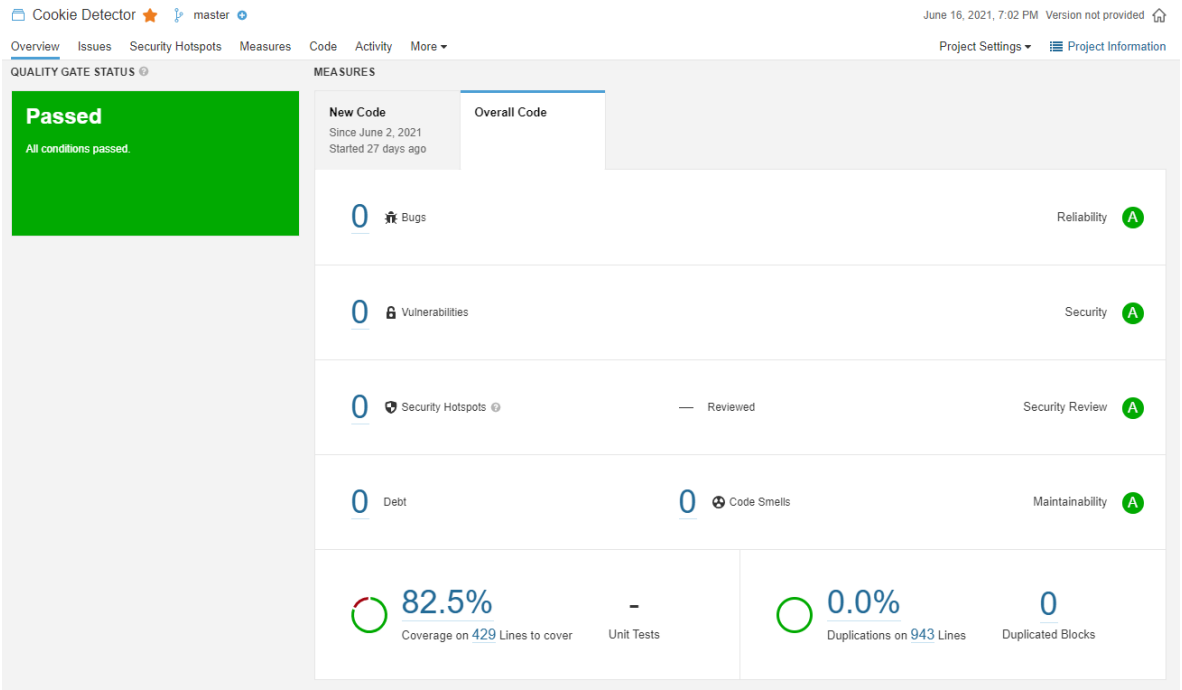


Figura 22: Análise geral das métricas de avaliação do código segundo o *SonarQube*

5.6 TESTES UNITÁRIOS

A cobertura do código está associada ao número de testes unitários desenvolvidos e ao número de linhas de código executadas por estes. Nesta secção serão apresentados os testes unitários desenvolvidos com o *pytest* para a nossa extensão.

A Figura 23 contém os testes unitários executados para o ficheiro que contém as funções relacionadas com a classificação dos *cookies* e a análise do *encoding* utilizado. Para este ficheiro foram desenvolvidos no total 30 testes unitários definidos com vários *inputs* parametrizados para testar o comportamento de todas as funcionalidades disponibilizadas. Todos os testes devolveram o *output* esperado por isso foram executados com sucesso como observamos na Figura 23.

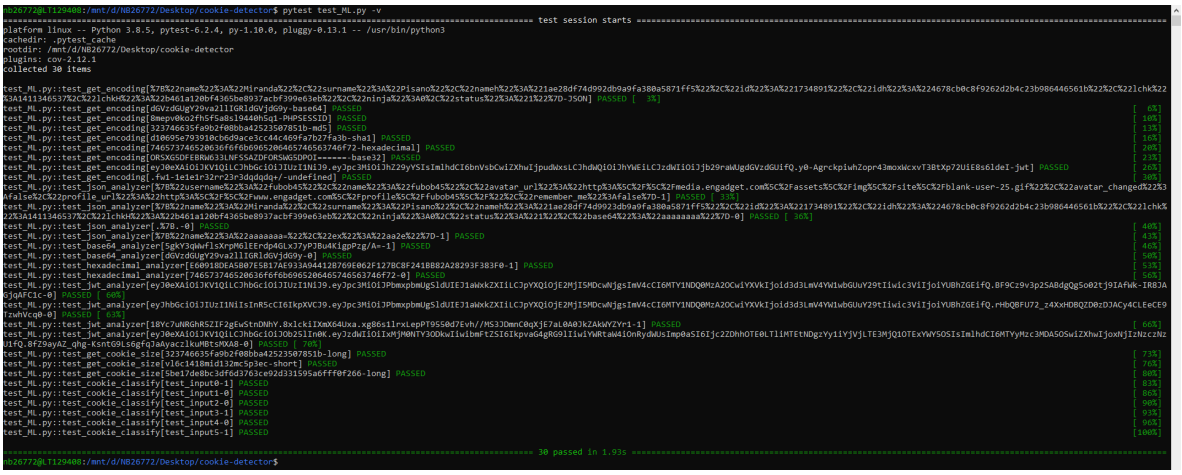


Figura 23: Testes unitários do ficheiro de classificação e análise de *cookies*

Na Figura 24 abaixo temos os testes unitários desenvolvidos para o ficheiro que é carregado no *Burp Extender*, incluindo todos os testes às funções que utilizam a *API* do *Burp*. Para este ficheiro foram criados 17 testes unitários e todos devolveram o *output* esperado, por isso foram executados com sucesso.

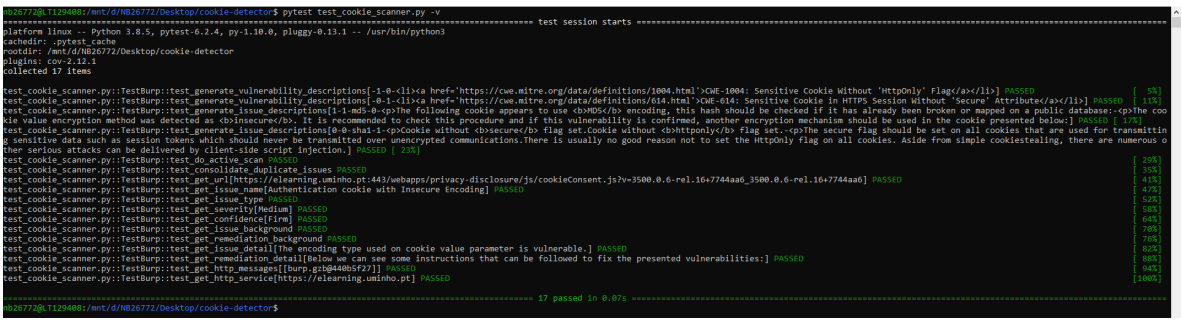


Figura 24: Testes unitários do ficheiro da extensão executada no *Burp*

6

RESULTADOS

Neste capítulo serão apresentados alguns resultados obtidos em diversas aplicações *web* após uma breve explicação do funcionamento da extensão. O *Burp* através do *proxy* armazena todas as interações *HTTP request/response* no seu *HTTP history*.

Como vemos na Figura 25, temos o *Proxy History* do *Burp*. Este contém diversas linhas que representam as interações *request/response* capturadas pelo *proxy* na porta previamente especificada.

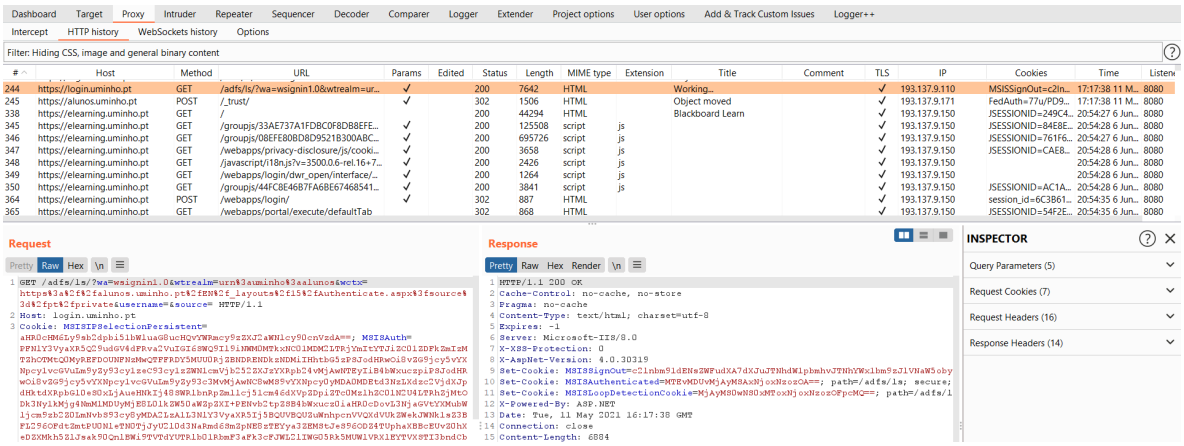


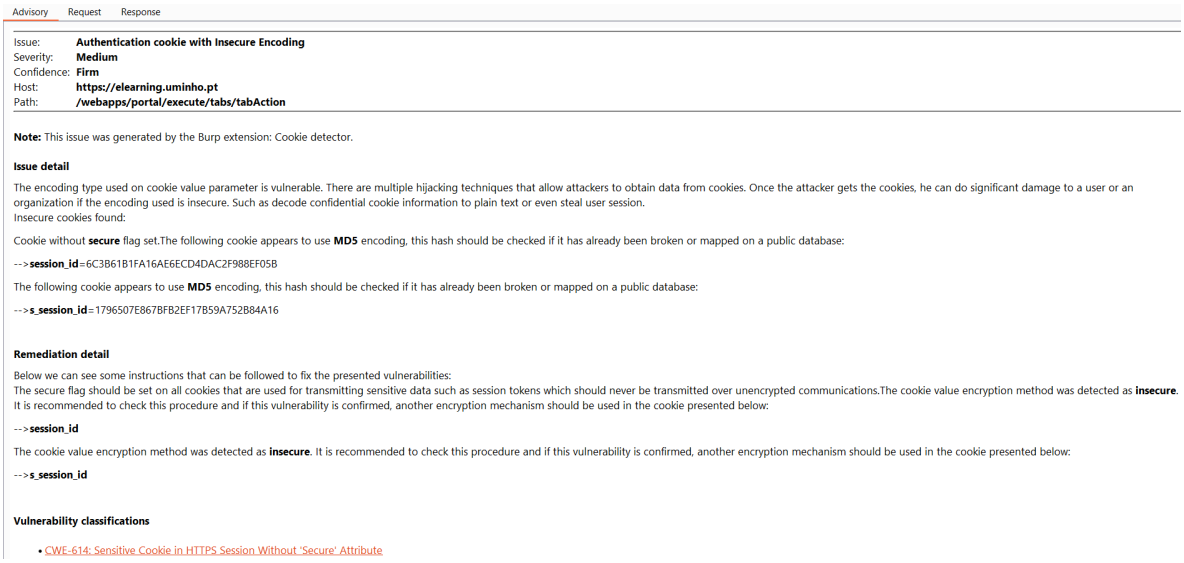
Figura 25: Exemplo de *proxy history* do *Burp*

Ao carregar a nossa extensão **Cookie Detector** no *Burp Extender*, o mecanismo de classificação e análise de *cookies* será integrado no *scanner*. Depois, podemos selecionar múltiplas linhas para fazer o *scan*, neste será utilizado o *scanner* do *Burp* em conjunto com a nossa extensão de análise de *cookies*. A *tag* que identifica a criação de um novo *cookie* é o **Set-Cookie:** dentro de uma mensagem *HTTP response*. Logo sempre que for encontrado este padrão, será enviado para a *extensão* analisar.

Quando a extensão recebe os *cookies* capturados pelo *Burp*, estes serão classificados e analisados. Se eventualmente algum *cookie* for avaliado como inseguro, será reportada uma nova vulnerabilidade no painel principal e nos resultados do scan do *Burp*.

6.1 EXEMPLO 1 - TESTE DE COOKIE MD5(UMINHO)

Na Figura 26 observamos o exemplo de uma vulnerabilidade gerada pela nossa extensão. Seguindo o padrão do *Burp*, a vulnerabilidade é constituída por três páginas, a *Advisory* que apresenta o relatório da vulnerabilidade encontrada e depois a *Request* e *Response* que deram origem a esta.



Advisory Request Response

Issue: **Authentication cookie with Insecure Encoding**
Severity: **Medium**
Confidence: **Firm**
Host: **https://elearning.uminho.pt**
Path: **/webapps/portal/execute/tabs/tabAction**

Note: This issue was generated by the Burp extension: Cookie detector.

Issue detail

The encoding type used on cookie value parameter is vulnerable. There are multiple hijacking techniques that allow attackers to obtain data from cookies. Once the attacker gets the cookies, he can do significant damage to a user or an organization if the encoding used is insecure. Such as decode confidential cookie information to plain text or even steal user session.
Insecure cookies found:

Cookie without **secure** flag set.The following cookie appears to use **MD5** encoding, this hash should be checked if it has already been broken or mapped on a public database:

-->**session_id**=6C3B61B1FA16AE6ECD4DAC2F988EF05B

The following cookie appears to use **MD5** encoding, this hash should be checked if it has already been broken or mapped on a public database:

-->**s_session_id**=1796507E867BFB2EF17B59A752B84A16

Remediation detail

Below we can see some instructions that can be followed to fix the presented vulnerabilities:
The secure flag should be set on all cookies that are used for transmitting sensitive data such as session tokens which should never be transmitted over unencrypted communications.The cookie value encryption method was detected as **insecure**. It is recommended to check this procedure and if this vulnerability is confirmed, another encryption mechanism should be used in the cookie presented below:

-->**session_id**

The cookie value encryption method was detected as **insecure**. It is recommended to check this procedure and if this vulnerability is confirmed, another encryption mechanism should be used in the cookie presented below:

-->**s_session_id**

Vulnerability classifications

- CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute

Figura 26: Relatório de vulnerabilidade detetada pela extensão

Agora, passando a explicar o relatório, temos os campos padrão do *Burp* que indicam dados como o nome da vulnerabilidade, a gravidade do problema encontrado, o grau de confiança e os dados da fonte onde foi identificada. Seguem os três campos mais importantes do relatório, o *Issue detail*, *Remediation detail* e *Vulnerability classifications*.

O *Issue detail* explica detalhadamente a vulnerabilidade encontrada, a razão pela qual foi gerada e apresenta os *cookies* analisados pela extensão que resultaram na invocação do *issue*. Nos *cookies* apresentados temos, a negrito o nome do *cookie* e à direita, o campo valor do *cookie*. Quanto ao *Remediation detail*, é feita a sugestão para cada *cookie* dos procedimentos a seguir para resolver a vulnerabilidade. Por fim, as *Vulnerability Classifications* indicam os *links* das *CWE*(Common Weakness Enumeration) associadas às vulnerabilidades encontradas nos *cookies*.

Falando agora sobre os *cookies* em questão, que foram detetados como inseguros, no 1º *cookie*, sendo um *cookie* de autenticação e não utilizando a flag *Secure* é automaticamente classificado como inseguro. Relativamente ao 2º *cookie*, é classificado como inseguro porque, como foi explicado no capítulo 5, o *hashing MD5* não é seguro atualmente. Depois da explicação das razões pelas quais foram avaliados como inseguros, surgem algumas indicações para adicionar as flags *Secure* no 1º *cookie* e é sugerido verificar a *hash MD5* para o 2º *cookie*.

Como foi mencionado, a *response* é o local onde são criados *cookies* através do *header Set-Cookie*. Abaixo na Figura 27 temos a *response* que gerou a vulnerabilidade apresentada na Figura 26.

De relembrar também que a extensão desenvolvida possui uma funcionalidade que destaca os *cookies* identificados como inseguros. Podemos observar na Figura 27 que os *headers Set-cookie* são destacados com uma cor vermelha para os *cookies* que foram detetados como vulneráveis.

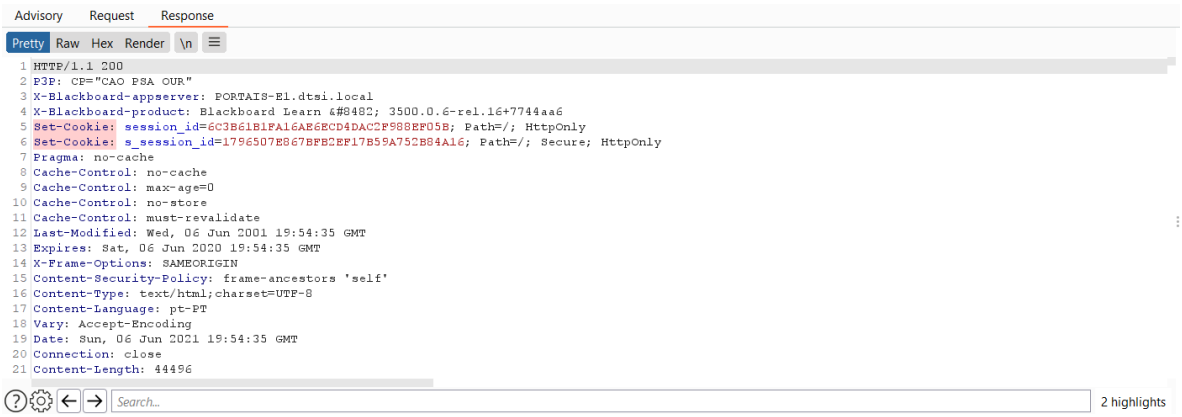


Figura 27: Resposta *HTTP* na qual foi detetada a vulnerabilidade

A Figura 28, representa os *logs* gerados pela extensão, pode ser especialmente útil para o utilizador verificar todos os da classificação atribuída ao *cookie*.

Temos na 1º linha, todas as informações do *cookie* enviadas ao modelo de *Machine Learning*, nomeadamente, nome, valor, *encoding*, *flags* e comprimento do *cookie*. De seguida, a indicação se o modelo classificou o *cookie* sendo ou não de autenticação. Depois, o resultado da análise ao *encoding* aplicado, indicando se é seguro. E por fim, a avaliação final que determina se o *cookie* é vulnerável.

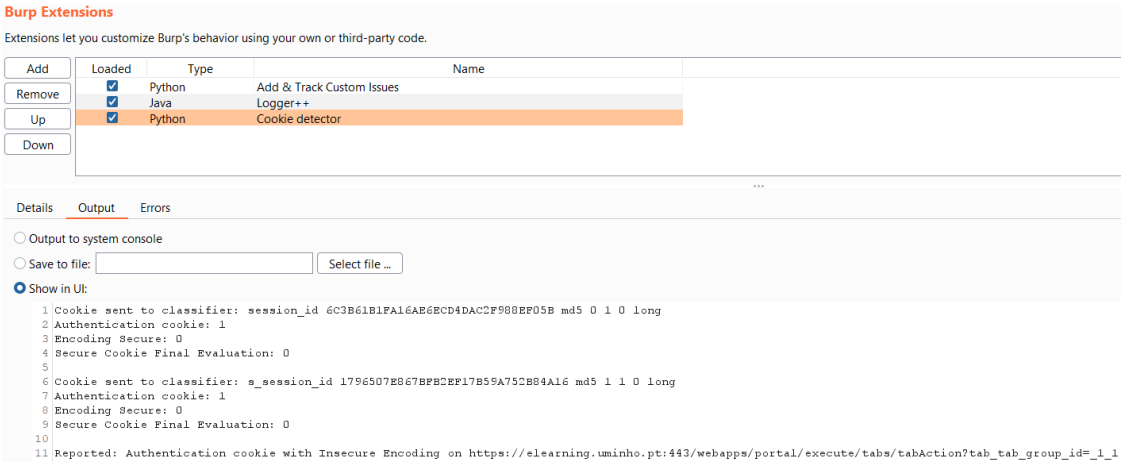


Figura 28: *Logs* da extensão para o exemplo da vulnerabilidade apresentada

6.2 Exemplo 2 - Teste de cookie MD5(Outlook)

No exemplo da Figura 29, foram capturadas as mensagens *HTTP* geradas após efetuar uma autenticação no serviço de *email outlook*. A imagem contém um dos *cookies* gerados no processo de autenticação, este será enviado para o *scanner* do *Burp* de forma a testar a nossa extensão.

Figura 29: Mensagem *HTTP* capturada após autenticação no *outlook*

A Figura 30 corresponde à vulnerabidade reportada após a análise da mensagem *HTTP* associada à autenticação no serviço de *email outlook* apresenta na Figura 29.

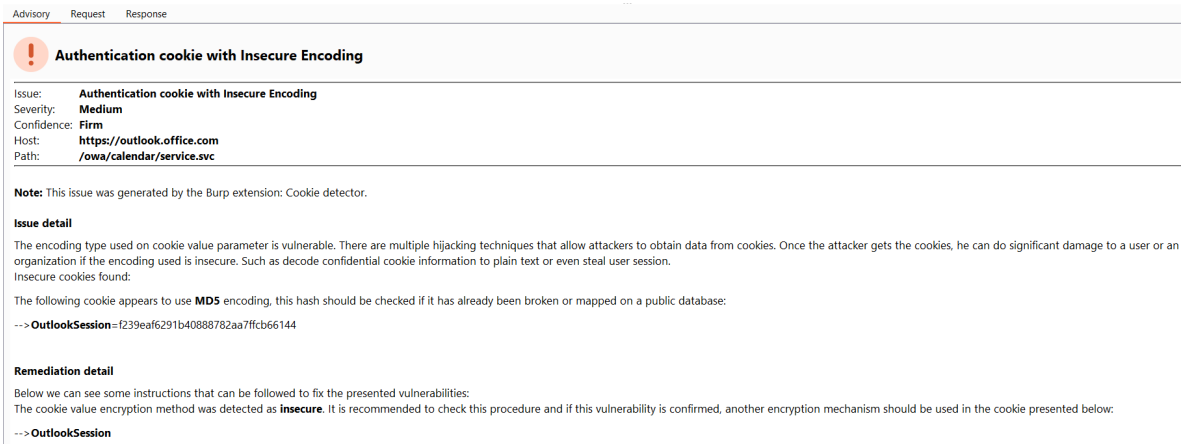


Figura 30: Vulnerabilidade reportada pela extensão para o exemplo do *outlook*

Inicialmente é enunciada de forma geral a gravidade da utilização de *cookies* vulneráveis e depois, a explicação da razão pela qual o *cookie* analisado é vulnerável.

O *cookie* foi classificado como inseguro porque, uma vez mais, utiliza o *encoding md5*. De seguida, surge a típica recomendação para corrigir a vulnerabilidade, tal como no exemplo anterior.

De relembrar, que a extensão funcionou como seria esperado, porque classificou corretamente o *cookie* de autenticação e lançou o alerta de que o *encoding* utilizado poderá estar comprometido.

Consultando os *logs* da extensão, na Figura 31, após enviar o *cookie* ao scanner, observamos que foi classificado corretamente pelo modelo de *machine learning* como sendo um *cookie* de autenticação.

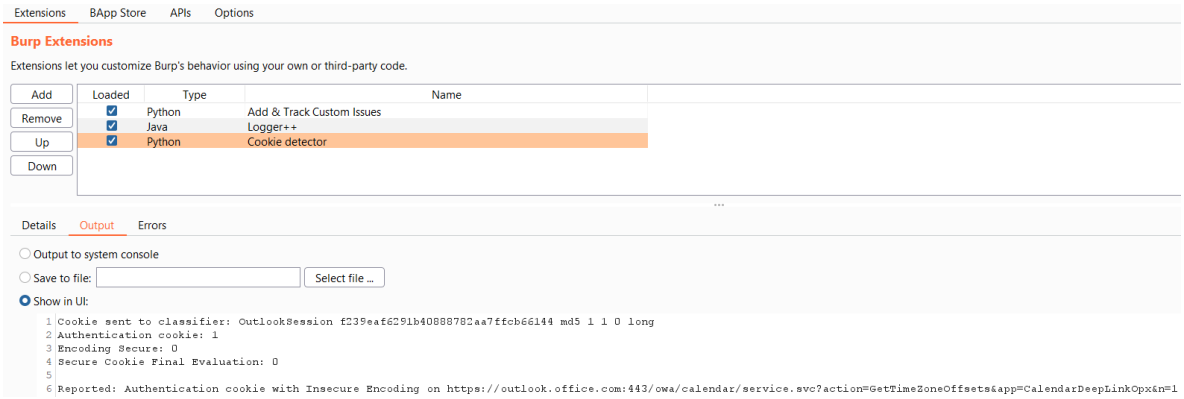


Figura 31: *Logs* da extensão após análise do *cookie* gerado pelo *outlook*

Quanto ao *encoding*, como já foi enumerado múltiplas vezes, o *MD5* não é considerado seguro logo é atribuído o valor zero à segurança do *encoding*.

Conjugando estes dois resultados, dado que é um *cookie* de autenticação e não utiliza um *encoding* seguro, então é atribuída a classificação final de *cookie* vulnerável.

Como o *cookie* é vulnerável, a extensão criará uma vulnerabilidade para ser reportada no *Burp*, tal como indica a última linha dos *logs* na Figura 31.

6.3 Exemplo 3 - Teste de cookie JWT(Microsoft Forms)

A mensagem *HTTP* da Figura 32 foi capturada pelo *proxy* do *Burp* após autenticação no *Microsoft Forms*. Como vemos, quando é feita a autenticação neste serviço são gerados vários *cookies* que serão analisados pela nossa extensão.

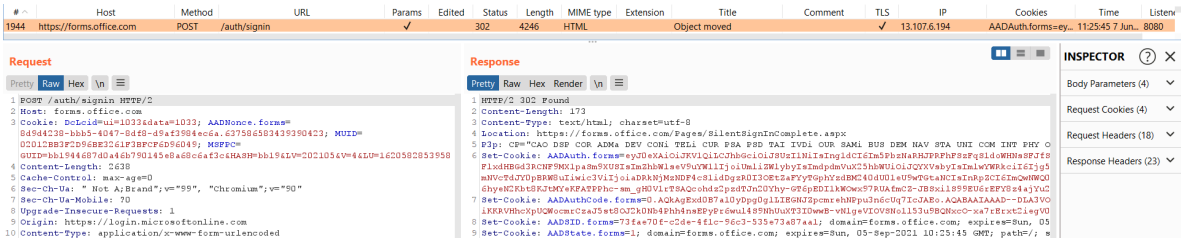


Figura 32: Mensagem *HTTP* capturada após autenticação no *Microsoft Forms*

A Figura 33 representa os *logs* gerados pela extensão quando são analisados os *cookies* do *Microsoft Forms*.

Figura 33: *Logs* da extensão após análise do *cookie* do *Microsoft Forms*

Analisando os resultados, concluímos que houve uma falha no classificador de *machine learning*, a todos eles foi atribuído o valor zero no parâmetro *Authentication cookie* e o valor real seria um porque são responsáveis pela autenticação.

Nos dois primeiros *cookies*, o *encoding* foi atribuído corretamente, nomeadamente, o *JWT*. Nestes dois, os *JWT* foram verificados e devidamente classificados como seguros levando a uma avaliação final de *cookie* seguro.

No 3º *cookie* o *encoding* utilizado não foi detetado, logo foi classificado como *undefined*. Dado que houve esta falha na deteção, então o valor que define a segurança do *encoding* é zero. Não faz sentido classificar como seguro um *encoding* que não temos conhecimento, daí a aplicação desta abordagem. A classificação final é que o *cookie* é vulnerável mas devido ao facto de não ter sido considerado um *cookie* responsável pela autenticação, a vulnerabilidade não é reportada no *Burp*. Constituíndo outra falha na extensão, se um *cookie* de autenticação não possui um *encoding* conhecido, o utilizador deve ser alertado para verificar o mesmo.

Por fim, no 4º *cookie*, o nome é *AADState.forms* e o valor é 1, o que provavelmente representa uma alteração do estado do utilizador na aplicação passando de não autenticado para autenticado. Como já foi mencionado houve uma falha atribuíndo o valor zero no campo que identifica se é um *cookie* de autenticação. Para além desta falha, como o campo valor do *cookie* é enviado em texto claro e possui apenas 1 caracter, não é possível reconhecer este tipo de dados e o detetor de *encodings* atribuiu erradamente o *encoding* hexadecimal.

De relembrar que devido ao facto de nenhum dos *cookies* ser classificado como *cookie* de autenticação, caso sejam avaliados como inseguros não serão reportados.

6.4 Exemplo 4 - Teste de cookie PHPSESSID(Guru99)

Este exemplo da Figura 34 baseia-se na experimentação feita em um *website* desenhado para testes chamado *guru99*. Na figura temos a mensagem *HTTP* gerada após autenticação no *website* mencionado que será utilizada para testar a nossa extensão. Esta mensagem contém o *cookie* de autenticação que será análisado.

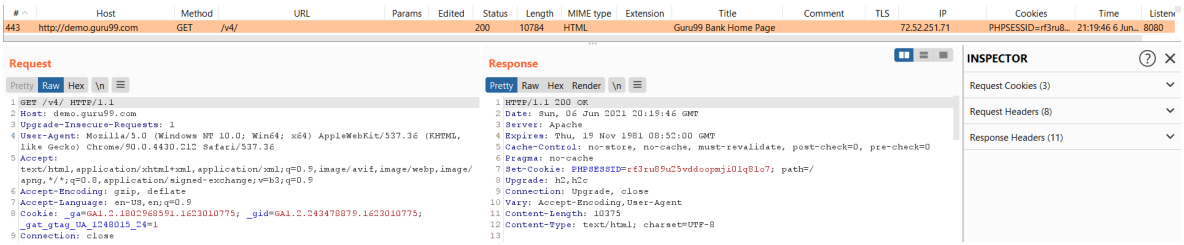


Figura 34: Mensagem *HTTP* capturada no *website* de teste *guru99*

Na Figura 35 temos a vulnerabilidade reportada para este exemplo do *guru99*. A informação do *Issue detail*, indica o alerta geral sobre a utilização de *cookies* vulneráveis e depois explica que a razão da criação da *vulnerabilidade* foi a falta de uso das *flags secure* e *HttpOnly*. O *Remediation detail* recomenda a ativação destas *flags* e avisa sobre os possíveis perigos associados a esta falha. No fim, são apresentados os *links* para as *CWE* relacionadas com os problemas encontrados na análise do *cookie*.

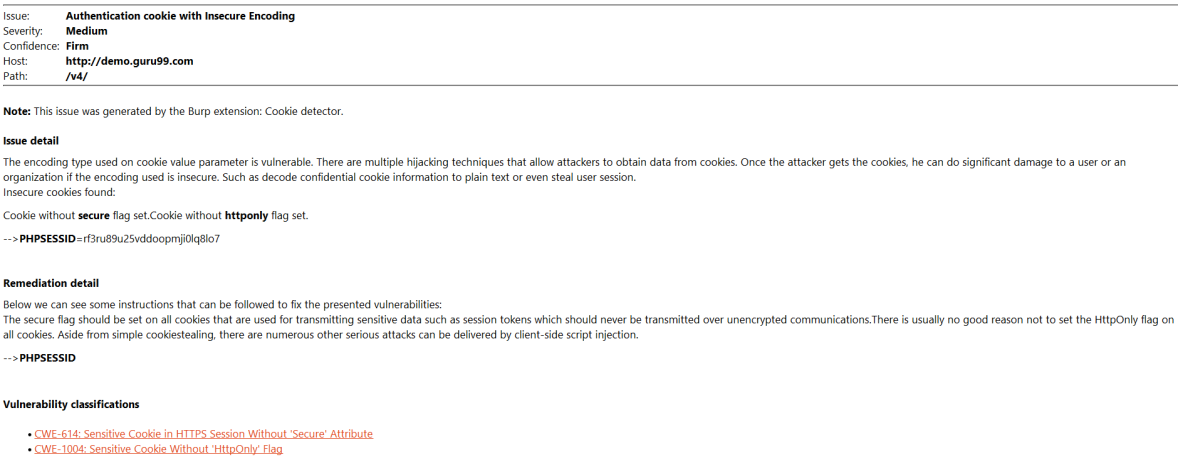


Figura 35: Vulnerabilidade reportada pela extensão para o exemplo do *guru99*

Analisando a Figura 36, percebemos que o *cookie* criado tem o nome *PHPSESSID*, o valor é a chave de 26 caracteres da imagem e o *encoding* é um *PHPSESSID*. Os restantes campos na 1º linha indicam que o *cookie* não utiliza as *flags secure* nem *HttpOnly* e por fim, o comprimento do *cookie* é classificado como *long*. O mecanismo de *machine learning* classificou corretamente como sendo um *cookie* de autenticação, a análise do *encoding* também foi correta, definindo-o como seguro. Quanto à avaliação final, foi avaliado como vulnerável, isto deve-se ao facto do *cookie* não utilizar as *flags secure* nem *HttpOnly*.

Este caso serve como exemplo porque a segurança do *encoding* do *cookie* não é suficiente. Apesar do *encoding* analisado ser *seguro*, se as *flags* não forem utilizadas devidamente, a extensão lança a vulnerabilidade no *Burp*.

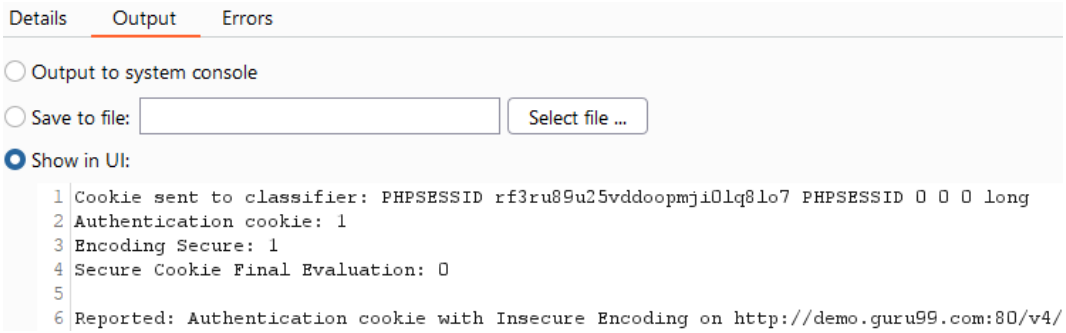


Figura 36: *Logs* da extensão após análise do *cookie*

6.5 Exemplo 5 - Teste de cookie JWT(Desafio Web Security Celfocus)

Na Figura 37 temos uma *request/response HTTP* recolhida no desafio de *Web Security* da Celfocus. Como observamos na *response*, o *cookie* não está contido na *tag* normal "*Set-Cookie:*"mas sim no final da mensagem na *tag token* dentro de "{ }". Este *cookie* utiliza o *encoding JWT* e será analisado pela nossa extensão.

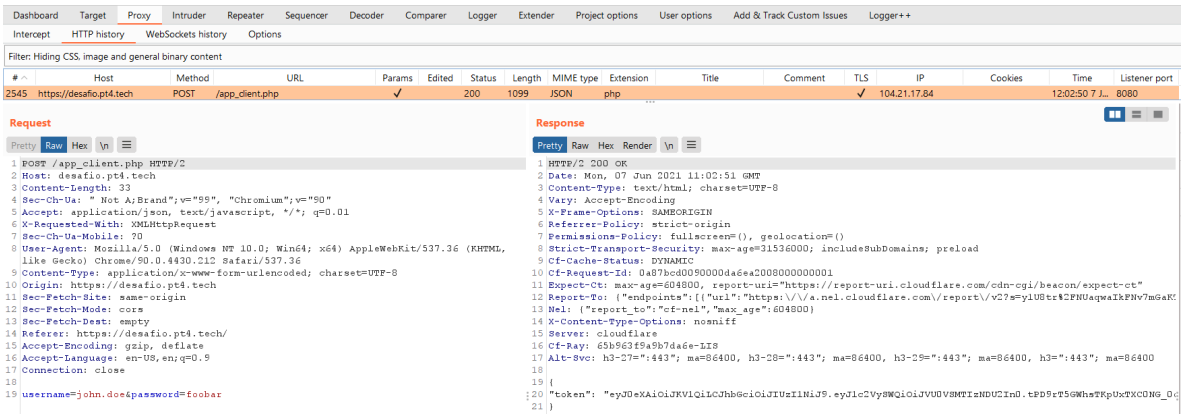


Figura 37: Mensagem *HTTP* capturada no *website* de teste da Celfocus

A Figura 38 apresenta o resultado da descodificação do *cookie JWT* efetuada no *website* "JWT.IO". Quanto ao *header* este identifica o encoding como *JWT* e o algoritmo de assinatura da chave secreta *HS256*. No campo *payload* é registado o identificador do utilizador que fez a autenticação no *website*.

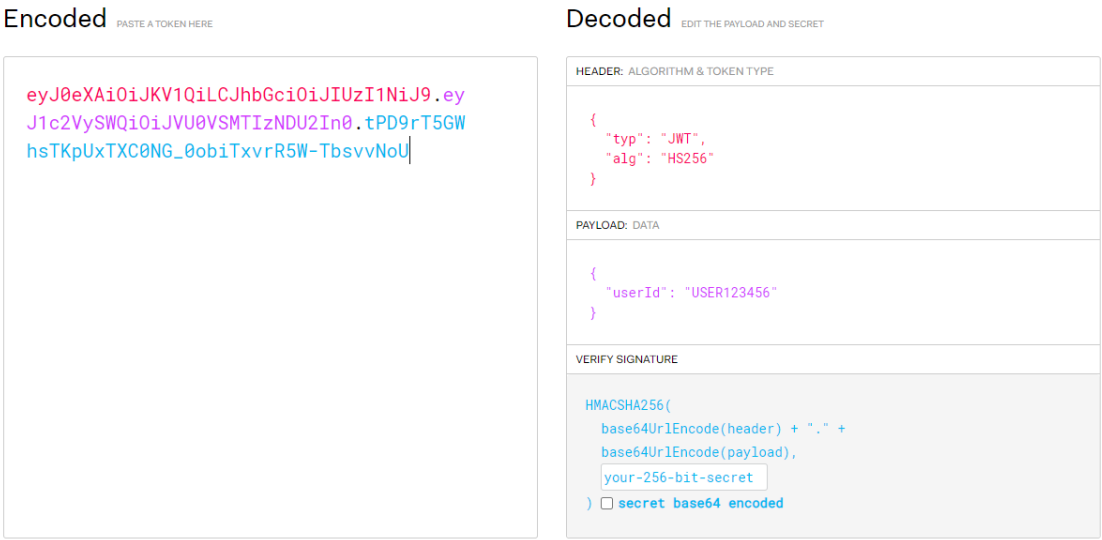


Figura 38: Descodificação do cookie utilizando o JWT.IO

É de extrema importância o uso de um algoritmo de assinatura da chave secreta para garantir que o *JWT* não é modificado indevidamente. Quando há uma alteração do *payload* do *JWT* os *websites* têm de verificar se a assinatura é válida , ou seja, se a chave secreta inserida coincide com a assinatura do *JWT*. Se não for utilizado um algoritmo de assinatura não haverá validação da assinatura logo o utilizador pode modificar o conteúdo do *JWT*. Considerando este exemplo da Figura 38, se não existisse assinatura do *JWT* ele poderia alterar o campo *userId* do *payload* e efetuar a autenticação na plataforma com outro utilizador elevando os seus privelégios se utilizasse por exemplo o identificador do administrador do sistema. Como a extensão desenvolvida apenas analisa os *cookies* dentro da *tag "Set-Cookie"* e neste exemplo o *cookie* está dentro de uma *tag* chamada *token* será necessário enviar manualmente o *cookie JWT* à extensão.

Na Figura 39 temos os resultados da análise do *cookie JWT* gerado pela autenticação no desafio de *Web Security* da Celfocus. A classificação do *cookie* é correta, dado que foi classificado como *cookie* de autenticação. Quanto à sua segurança, utiliza o algoritmo de assinatura *HS256* e o *encoding* dos dados contidos no *payload* não representa nenhuma vulnerabilidade. Logo, o *cookie* é classificado acertadamente como sendo um *cookie* seguro.

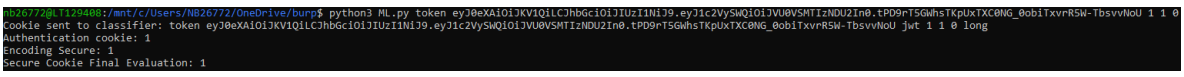


Figura 39: Resultados da análise do *cookie JWT* da Figura 37

6.6 Exemplo 6 - Teste de cookie JWT(Desafio Web Security Celfocus) sem algoritmo de assinatura

A Figura 40 apresenta uma sequência *request/response* associada a outra autenticação no *website* do desafio de *Web Security* da Celfocus. Este exemplo utiliza também um *cookie JWT* como observamos na figura.

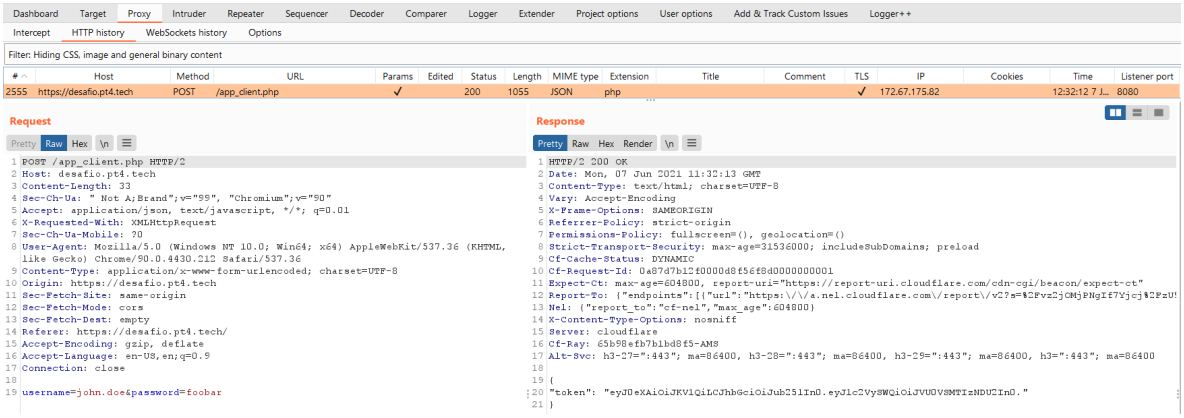


Figura 40: Mensagem *HTTP* capturada no *website* de teste da Celfocus

Na Figura 41 temos o resultado da descodificação do *cookie JWT* da Figura 40 efetuada no *website* "JWT.IO". Este *cookie* está dividido em 3 campos, como é normal no *JWT*, mas o 3º campo está vazio porque não utiliza um algoritmo de assinatura da chave secreta como observamos no campo *alg* do *header*. Neste exemplo como não existe algoritmo de assinatura da chave secreta e consequentemente o campo *signature* é vazio, o utilizador pode modificar os campos *payload* e enviar ao *website* porque não há forma de validar o *JWT*.

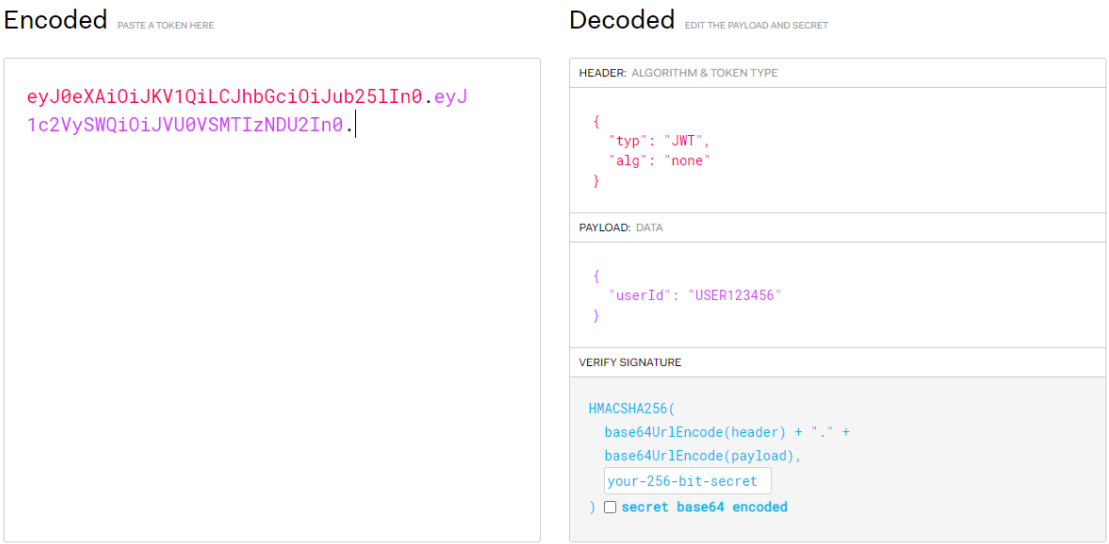


Figura 41: Descodificação do cookie utilizando o JWT.IO

Por fim, na Figura 42 observamos os resultados do envio do *cookie* à nossa extensão de classificação e análise de *cookies*. A classificação inicial é acertada porque é um *cookie* de autenticação. Como o *JWT* não contém um algoritmo de assinatura da chave secreta, ou seja, este campo é preenchido com *None* então o *cookie* é automaticamente classificado como *cookie* inseguro.

```
nb26772@LT129408:/mnt/c/Users/NB26772/OneDrive/burp$ python3 ML.py Token eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VySWQiOiJVU0VSMTIzNDU2In0. 1 1 0
Cookie sent to classifier: Token eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VySWQiOiJVU0VSMTIzNDU2In0. jwt 1 1 0 long
Authentication cookie: 1
Encoding Secure: 0
Secure Cookie Final Evaluation: 0
```

Figura 42: Resultados da análise do *cookie JWT* da Figura 40

7

CONCLUSÃO

7.1 RESUMO DA SOLUÇÃO

A integração de técnicas de aprendizagem automática na área da segurança informática, especialmente, na deteção de vulnerabilidades é um paradigma em crescimento, dado o seu incrível potencial.

Com base nesta conjectura, surgiu este projeto de criação de uma extensão para classificação automática de *cookies* recorrendo a modelos de *machine learning*. Para além da classificação, a extensão faz também a deteção/análise dos *encodings* utilizados nos *cookies* de autenticação.

Após o desenvolvimento destes mecanismos de classificação/análise de *cookies* seguiu-se a integração com a aplicação de testes de segurança, *Burp Suite*, permitindo a automatização do processo de verificação de *cookies* de autenticação. A extensão reporta uma vulnerabilidade no *Burp Suite* quando for encontrado um *cookie* vulnerável.

7.2 OBJETIVOS CONCLUÍDOS

À data da criação deste projeto foi definida uma lista de objetivos e uma metodologia a seguir para que possam ser alcançados.

Neste projeto, foi possível melhorar o trabalho já desenvolvido pelos autores do artigo Calzavara et al. (2015) apresentado no Estado de Arte. Estes autores criaram um conjunto de dados que foi utilizado no treino do nosso classificador de *cookies*. Foi algo indispensável, dado que seria muito trabalhoso construir um *dataset* destes. Relativamente aos resultados do classificador, foram muito satisfatórios porque foi possível superar os resultados apresentados no Estado de Arte. Como foi explicado, o nosso classificador obteve uma *F-measure* de 0.84 enquanto que o classificador apresentado no Estado de arte obteve 0.83.

Dois dos objetivos desta dissertação consistiam na pesquisa e desenvolvimento de um mecanismo de aprendizagem automática para classificação de *cookies*, como demonstrado este objetivo foi alcançado.

O objetivo seguinte, que também foi concluido, consistia na deteção do *encoding*/cifragem utilizada no valor dos *cookies* para auxilar o modelo de *machine learning* e avaliar a segurança do *cookie*.

O último objetivo baseava-se na geração de um relatório após a análise dos *cookies* encontrados em uma aplicação *web*. Este foi atingido com a integração do *Burp Suite*, permitindo capturar os *cookies* da aplicação, enviar para a nossa extensão e reportar uma vulnerabilidade caso seja encontrado algum *cookie* inseguro.

7.3 DESAFIOS

Ao longo do trabalho desenvolvido surgiram os seguintes desafios:

- O desbalanceamento do *dataset* foi um dos problemas porque apenas 1 em cada 7 *cookies* era um *cookie* de autenticação. Este desequilíbrio causava uma falha no modelo de *machine learning*, os *cookies* eram quase todos classificados como não sendo *cookies* de autenticação. Para resolver este problema foram testadas várias técnicas como o *undersampling/oversampling* mas não retornavam os resultados esperados. Depois, com a utilização do parâmetro de balanceamento de dados (*class_pior*) disponível no classificador *Bernoulli Naive Bayes* foi possível resolver o problema e obter os resultados desejados.

- A quantidade de tipos de *encodings* utilizados no valor dos *cookies* é uma área muito extensa. A abordagem que escolhemos para minimizar este problema foi focar-nos nos *encodings* mais comuns/utilizados. Com isto, conseguimos para o exemplo do *dataset* desenvolvido pelos autores do artigo enunciado no Estado de Arte, detetar grande parte dos *encodings* presentes. A deteção não é infalível como demonstramos no Capítulo 6, há casos em que ainda existem algumas falhas.

- O último desafio foi a integração da ferramenta desenvolvida com o *Burp Suite*, de modo a funcionarem em conjunto. Esta integração tem de ser feita utilizando a API do *Burp* e a extensão executada no compilador *Jython* do *Burp Extender*. Este compilador causou problemas porque não suporta a execução de programas que utilizem algumas bibliotecas de *machine learning* como o *numpy* e o *scikit-learn*. Estas bibliotecas foram imprescindíveis no desenvolvimento das funcionalidades da nossa extensão por isso foi necessário arranjar uma solução. A solução encontrada foi a execução do código incompatível com o *jython* fora do *Burp Extender*, ou seja, localmente num compilador *python3*. Assim, foram desenvolvidos mecanismos para establecer a comunicação entre o código executado no *jython* e o código executado no *python*. Desta forma, foi possível manter operacionais todas as funcionalidades da nossa extensão.

7.4 TRABALHO FUTURO

Como sugestão de trabalho futuro serão descritas algumas melhorias que poderiam ser feitas para expandir as funcionalidades do trabalho desenvolvido:

- O ponto fundamental seria o aumento do número de *encodings* detetados pelo nosso detetor, como já foi explicado, esta é uma àrea muito extensa, existem dezenas de abordagens diferentes utilizadas nos *cookies* de autenticação
- Seria interessante a melhoria na deteção de alguns dos atuais *encodings* utilizando técnicas mais rigorosas para melhorar a precisão na deteção. Por exemplo no *JWT* há verificações adicionais que podem ser adicionadas de forma a melhorar a deteção e garantir a segurança do *cookie*
- A precisão alcançada pelo classificador de *machine learning* é superior à precisão apresentada no Estado de Arte mas acreditamos que há margem para melhoria, por exemplo, através da expansão do *dataset* de treino do modelo

7.5 CONSIDERAÇÕES FINAIS

Este projeto consiste no desenvolvimento de um mecanismo de aprendizagem automática aplicado à deteção de vulnerabilidades. Dentro desta área, optamos por abordar a classificação e deteção de vulnerabilidades em *cookies* de autenticação.

No ínicio do trabalho foram definidos objetivos e uma metodologia a seguir ao longo do projeto para os alcançar.

Em conclusão, como demonstrado ao longo deste documento, os objetivos definidos foram atingidos, a extensão desenvolvida disponibiliza todas as funcionalidades enunciadas nos objetivos e enquadra-se nos pressupostos definidos.

BIBLIOGRAFIA

Jason Brownlee. How to encode text data for machine learning with scikit-learn, Jun 2020. URL https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/.

Michele Bugliesi, Stefano Calzavara, Riccardo Focardi, and Wilayat Khan. Automatic and robust client-side protection for cookie-based sessions. In *International Symposium on Engineering Secure Software and Systems*, pages 161–178. Springer, 2014.

Burp. How to use burp suite for penetration testing. URL https://portswigger.net/burp/documentation/desktop/penetration-testing.

Stefano Calzavara, Gabriele Tolomei, Michele Bugliesi, and Salvatore Orlando. Quite a mess in my cookie jar! leveraging machine learning to protect web authentication. In *Proceedings of the 23rd international conference on World wide web*, pages 189–200, 2014.

Stefano Calzavara, Gabriele Tolomei, Andrea Casini, Michele Bugliesi, and Salvatore Orlando. A supervised learning approach to protect client authentication on the web. *ACM Transactions on the Web (TWEB)*, 9(3):1–30, 2015.

CWE. Common weakness enumeration. URL https://cwe.mitre.org/.

Philippe De Ryck, Nick Nikiforakis, Lieven Desmet, Frank Piessens, and Wouter Joosen. Serene: Self-reliant client-side protection against session fixation. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 59–72. Springer, 2012.

Michael Jones, Brain Campbell, and Chuck Mortimore. Json web token (jwt) profile for oauth 2.0 client authentication and authorization grants. *May-2015.{Online}. Available: https://tools. ietf. org/html/rfc7523*, 2015.

Simon Josefsson et al. The base16, base32, and base64 data encodings. Technical report, RFC 4648, October, 2006.

JSON. Introducing json. URL https://www.json.org/json-en.html.

JWT.IO. Jwt.io allows you to decode, verify and generate jwt. URL https://jwt.io/.

Shahid Latif, Junaid Qayyum, Muhammad Lal, and Faheem Khan. Complete description of well-known number systems using single table. *International Journal of Engineering and Computer Science (IJECS-IJENS)*, 11(03), 2011.

Ajinkya More. Survey of resampling techniques for improving classification performance in unbalanced datasets. *arXiv preprint arXiv:1608.06048*, 2016.

Shraddha More and Arpit Rohela. Vulnerability assessment and penetration testing through artificial intelligence. *International Journal of Recent Trends in Engineering & Research*, 4(1): 217–224, 2018.

Mozilla. Cookies http. URL https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Cookies.

Nick Nikiforakis, Wannes Meert, Yves Younan, Martin Johns, and Wouter Joosen. Sessionshield: Lightweight protection against session hijacking. In *International Symposium on Engineering Secure Software and Systems*, pages 87–100. Springer, 2011.

OWASP. Session management cheat sheet. URL https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html.

OWASP_Foundation. Owasp top ten 2017. URL https://owasp.org/www-project-top-ten/2017/.

Pedro José Pereira, Paulo Cortez, and Rui Mendes. Multi-objective grammatical evolution of decision trees for mobile marketing user conversion prediction. *Expert Systems with Applications*, 168:114287, 2021.

PHP. Php manual. URL https://www.php.net/manual/en/function.session-id.php.

Anak Agung Putri Ratna, Prima Dewi Purnamasari, Ahmad Shaugi, and Muhammad Salman. Analysis and comparison of md5 and sha-1 algorithm implementation in simple-o authentication based security system. In *2013 International Conference on QiR*, pages 99–104. IEEE, 2013.

RFC7519. Jjson web token (jwt). URL https://datatracker.ietf.org/doc/html/rfc7519.

SANS. Cwe/sans top 25 most dangerous software errors. URL https://www.sans.org/top25-software-errors/.

Yu Sasaki, Go Yamamoto, and Kazumaro Aoki. Practical password recovery on an md5 challenge and response. *IACR Cryptol. ePrint Arch.*, 2007:101, 2007.

scikit learn. Tfidftransformer. URL https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html.

SonarQube. Sonarqube documentation. URL https://docs.sonarqube.org/latest/.

Alex Sotirov. Analyzing the md5 collision in flame. *Presentation at SummerCon, slides available at http://www. trailofbits. com/resources/flame-md5. pdf*, 2012.

Sunny Srinidhi. Understanding word n-grams and n-gram probability in natural language processing. URL https://towardsdatascience.com/understanding-word-n-grams-and-n-gram-probability-in-natural-language-processing-9d9eef0fa058.

Shuo Tang, Nathan Dautenhahn, and Samuel T King. Fortifying web-based applications automatically. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 615–626, 2011.

Rüdiger Wirth and Jochen Hipp. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, volume 1. Springer-Verlag London, UK, 2000.