ISCTE IUL
Instituto Universitário de Lisboa

Departamento de Ciências e Tecnologias da Informação

Verificação da robustez das palavras-passe de forma distribuída e voluntária através da WWW

Luís Carlos Lima Marques

Dissertação submetida como requisito parcial para obtenção do grau

de

Mestre em Engenharia Informática

Orientador

Doutor Carlos Serrão, Prof. Auxiliar, ISCTE-IUL

Setembro 2018

“If you think technology can solve your security problems, then you don't understand the problems and you don't understand the technology.”

Bruce Schneier

Resumo

A segurança da informação tem vindo a ganhar cada vez mais importância nas organizações, e esse aumento é devido aos processos de negócios e o suporte de funções por sistemas informação informatizados. Um dos mecanismos de controlo de acesso mais usados no mundo da segurança da informação é a autenticação por palavra-passe. No entanto, é também um dos mecanismos de autenticação mais problemáticos, porque sua segurança depende da robustez da palavra-passe escolhida. As organizações devem se preocupar com a qualidade das palavras-passe e determinar a robustez das palavras-passe como medida de segurança.
Este trabalho pretende aplicar computação distribuída através da *Web*, utilizando navegadores da *Web* e computação voluntária, para verificar a robustez das palavras-chave. Usa um servidor Web Node.js e através do *Javascript* permite gerir as tarefas que são distribuídas e executadas por utilizadores finais voluntários. O suporte técnico é baseado em *WebWorkers* no navegador da *Web*. Pretende-se avaliar a qualidade do sistema proposto em comparação com os resultados obtidos com outras ferramentas, como "*John the Ripper*" e "*Hashcat*". Desta forma, podemos ver se a computação distribuída baseada no navegador da *Web*, pode ser um valor acrescentado em contribuir para a segurança da informação através da contribuição para promover palavras-passe mais robustas.

Palavras-chave: Computação distribuída, quebra de palavras-passe, Javascript, multiplataforma

Abstract

Information security has become increasingly important in organizations, and this increase is mostly due to business processes and functions supported by computerized information systems. One of the most commonly used access control mechanisms in the world of information security is password-based authentication. However, it is also one of the most problematic authentication mechanisms, because its security depends mostly on the robustness of the selected password. Organizations should be concerned with the quality of passwords and determine the robustness of passwords as a security measure.

This work intends to apply Web distributed computing, using Web browsers and voluntary computing, to verify passwords robustness. It uses a Node.js web server and implements Javascript logic to enable and manage the tasks that are distributed and executed by voluntary end-users. The system is based on Web Workers within the Web browser. In order to validate the developed system its results were compared with other tools such as “John the Ripper” and “Hash cat”. This way, it will be possible to validate if the distributed computation based on the Web browser, can contribute to information security through the promotion of more robust passwords.

Keywords: Distributed Computing, Password Crack, Javascript, Multiplatform

Índice

Lista de tabelas

Lista de figuras

Abreviaturas

AJAX	Asynchronous Javascript And XML
ATM	Automated Teller Machine
CIA	Confidentiality, Integrity, and Availability
COBIT	Control Objectives for Information and Related Technologies
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DSR	Design Science Research Methodology
FIFO	First In, First Out
GPU	Graphics Processing Unit
HPC	High-performance computing
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IP	Internet Protocol
ISO	International Organization for Standardization
LUDS	Lower and uppercase letters, digits and symbols
MIT	Instituto de Tecnologia de Massachusetts
NIST	National Institute of Standards and Technology
NoSQL	Not Only Structured Query Language
NPM	Node Package Manager
PCFG	Probabilistic Context-free Grammar
PIN	Personal Identification Number
QoS	Quality of Service
REST	Representational State Transfer
REST	Representational State Transfer
SANS	System Administration, Networking and Security
SDK	Software Development Kit
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
UTP	User Datagram Protocol

Capítulo 1 Introdução

1.1 Motivação

A área da segurança da informação tem vindo a ganhar maior importância, o que se traduz na consciencialização em relação ao impacto que pode ter nas organizações. Percebemos isso porque cada vez mais as relações entre as organizações estão mais digitais, mais processos e funções de negócio recorrem a estratégias digitais e porque cada vez mais as relações entre pessoas são igualmente digitais (Hinings, Gegenhuber, & Greenwood, 2018). Aliado a isso, a proliferação da tecnologia, gerou novas oportunidades para indivíduos mal-intencionados, que podem comprometer os sistemas de informação das organizações (Safa et al., 2015).

Um dos mecanismos mais comuns de autenticação em sistemas de informação pessoais e corporativos são as palavras-passe (*passwords*). No entanto, este mecanismo é dos mais frágeis (Sahin, Lychev, & Wagner, 2015). Assim, e para garantir a integridade, confidencialidade e disponibilidade dos sistemas de informação, as palavras-passe têm um papel de especial importância, em particular, na robustez das mesmas. Neste sentido, é de todo o interesse garantir que as palavras-passe, que são usadas e guardadas nos sistemas, sejam robustas por forma a mitigar o risco dos sistemas das organizações, pois estes podem ser comprometidos através de ataques realizados contra as palavras-passe (Pan, White, & Sun, 2017).

O principal objetivo desta dissertação é investigar e desenvolver um sistema distribuído através da WWW, que possa ser operado através de *browsers Web* padrão. O sistema deve e permitir verificar a robustez das palavras-passe, em particular recorrendo a ataques de força bruta.

1.2 Questão de investigação

A questão de investigação apresentada que dá suporte a este trabalho, visa abordar um dos mecanismos de autenticação mais utilizados para garantir acesso a sistemas de informação, as palavras-passe. Este mecanismo apesar de ser um dos mecanismos mais utilizados para autenticação, continua a ser um dos mais frágeis, principalmente devido ao facto dos utilizadores optarem por usar palavras-passe pouco robustas, muitas vezes recorrendo a palavras comuns ou combinações óbvias das mesmas (Safa et al., 2015). Assim, a principal questão de investigação deste trabalho é a seguinte:

- Será possível determinar e melhorar a robustez das palavras-passe através da utilização de computação distribuída e voluntária na *Web*, resultando no aumento das recomendações de segurança para utilização correta deste mecanismo de autenticação?

1.3 Objetivos

O principal objetivo desta dissertação é investigar e desenvolver um sistema distribuído, baseado em tecnologias *Web*, que possa ser executado em *browsers Web*. Para explorar este sistema, vão ser estudados alguns dos principais mecanismos criptográficos envolvidos na criação e proteção de palavras-passe, as principais características associadas às mesmas e a envolvente da computação distribuída na *Web*. O propósito a investigar é a verificação da robustez das palavras-passe, em particular recorrendo ao modelo de ataque de força bruta que possa ser realizado de forma distribuída, permitindo determinar, no menor espaço de tempo possível a sua robustez a este tipo de ataques. Os objetivos de operacionalidade do sistema deverão ter as seguintes características:

- Possibilidade de um cliente adicionar uma palavra-passe a ser verificada de forma distribuída;
- Visualização dos resultados através de relatórios automáticos, dispostos por tabelas e gráficos que possibilitem aferir a robustez da palavras-passe;
- Configuração e adaptação do sistema, para permitir a variabilidade de testes de robustez e de ataques contra palavras-chave a serem realizados
- Criar um sistema distribuído que deve ser capaz de lidar com eventuais falhas dos nós cliente.

- O sistema deve funcionar em *browser* padrão sem recurso a software complementar.

1.4 Metodologia de investigação

A metodologia utilizada para a realização deste trabalho de investigação baseia-se nas atividades preconizadas pelo *Design Science Research Methodology* (DSR). Esta metodologia permite resolver problemas reais e é orientada para a criação de artefactos (Hevner & Chatterjee, 2010).

Na sequência da metodologia DSR existem um conjunto de fases essenciais que vão permitir construir o artefacto final. Na primeira fase, é realizada a definição do objetivo, o *design* e desenvolvimento que segue uma abordagem iterativa ao estilo de uma metodologia ágil de desenvolvimento de Software (Fox, Sillito, & Maurer, 2008). A fase de demonstração coloca em prática a verificação da robustez das palavras-passe. Mesmo sendo esta fase de demonstração em simulações ou experimentos de teste, nesta fase deve ser possível ver algo tangível em termos de resultados. A fase seguinte permite aferir os resultados obtidos, entre esta fase e a fase anterior. Também são comparados os resultados obtidos com ferramentas relacionadas. Por fim, na última fase, a comunicação do artefacto, demonstrar a sua utilidade e usabilidade. Assim, esta metodologia permite estruturar o presente trabalho numa definição do problema, estado da arte, desenvolver e desenhar soluções para resolver um problema real, analisar resultados, discutir e proferir conclusões do artefacto conseguido.

Ainda nesta metodologia, foi definido uma lista de verificação com questões que permitem descrever uma linha de orientação na manobra de investigação e construção do artefacto.

Qual a orientação chave desta investigação?

Estudar os diversos modelos de verificação da robustez das palavras-passe, direcionando a atenção para perceber de que forma um sistema distribuído na WWW pode beneficiar a segurança da informação.

Como se descreve o artefacto produzido?

O artefacto final deve consistir num sistema distribuído, baseados em tecnologias WWW que deve consistir num servidor que permita distribuir tarefas aos clientes para serem processadas. O artefacto final deve permitir os clientes uma usabilidade simples, que permita utilizar a capacidade de processamento dos clientes e que não os iniba de realizar outras tarefas no *browser Web*.

Que ferramentas são consideradas para aferir a qualidade dos resultados?

A qualidade dos resultados pode ser aferida tendo em consideração as ferramentas *John the Ripper* e *Hashcat*. Estas ferramentas proporcionam elementos passiveis de comparação que nos permitem aferir a qualidade do sistema desenvolvido.

De que forma outras ferramentas relacionadas podem conduzir a melhorias no presente trabalho?

As ferramentas referidas na questão anterior, fazem parte da distribuição *Kali Linux 2018.3*. Estas ferramentas podem proporcionar bons indicadores comparativos, como por exemplo tempos de execução, velocidade e alcance da verificação das palavras-passe.

1.5 Estrutura do documento

Esta dissertação encontra-se organizada em seis capítulos que visam proporcionar uma leitura agradável e de fácil identificação de conteúdos. Assim, este primeiro capítulo introduz o leitor à motivação que esteve na origem desta dissertação, a questão de investigação, metodologia de investigação de enorme valor para a criação do documento e do artefacto final. O segundo capítulo, efetua o levantamento do estado da arte e como se encontram as matérias relacionadas com a temática desta dissertação. O terceiro capítulo tem como propósito descrever a conceptualização da solução e mostrar ao leitor aspetos importantes como os requisitos, a arquitetura seguida e as tecnologias utilizadas. O quarto capítulo, que aborda a implementação da solução, descreve pontos como a base de dados que suporta o sistema, o funcionamento do próprio sistema e os seus modos de utilização. O capítulo quinto descreve os resultados obtidos e a validação da solução, aqui podemos visualizar os valores resultantes dos testes quer no artefacto desenvolvido, quer noutras soluções já existentes. Por fim, o último capítulo apresenta as principais conclusões deste trabalho e conduz o leitor a um trabalho futuro e que descreve pontos de expansão e evolução do sistema.

Capítulo 2 Estado da arte

2.1 Introdução

Ao longo deste capítulo irá ser analisado o estado da arte, onde vão ser descritos projetos similares, dar um enquadramento teórico e temático sobre os temas que suportam esta dissertação. São abordados os temas da computação distribuída, com maior incidência onde a tecnologia é baseada na Web. Também será abordada a problemática de segurança, complexidade e robustez relacionada com as palavras-passe.

2.2 Segurança da informação

Nesta secção abordaremos a importância da segurança da informação e os conceitos mais importantes desta temática. Sendo a segurança da informação um conjunto de domínios bastante alargado conforme ilustra a Figura 1. Pretende-se nesta secção aprofundar com mais detalhe questões relacionadas com a autenticação, com destaque aos mecanismos baseados em palavras-passe. Para isso, iremos abordar a problemática inerente a este meio de autenticação, a robustez das palavras-passe e os modelos e técnicas de ataque que podem ser aplicados a estes mecanismos de autenticação.

Figura 1: Os diferentes domínios da *Cybersecurity*

A Figura 1 tem como fonte (Jiang, 2017).

2.2.1 Conceitos basilares da segurança da informação

O conceito de *Confidentiality, Integrity, Availability* (CIA) que significa confidencialidade, integridade e disponibilidade estão amplamente difundidos nas principais *frameworks* de segurança como por exemplo o Cybersecurity Framework - NIST (National Institute of Standards and Technology), Critical Security Controls for Effective Cyber Defense - SANS (System Administration, Networking and Security), Guidelines for cybersecurity - ISO (International Organization for Standardization) e o COBIT (Control Objectives for Information and Related Technologies).

Na sua generalidade bem como na literatura no seu sentido mais amplo, encontramos muitas vezes este conceito referido como tríade que é ilustrado conforme a Figura 2.

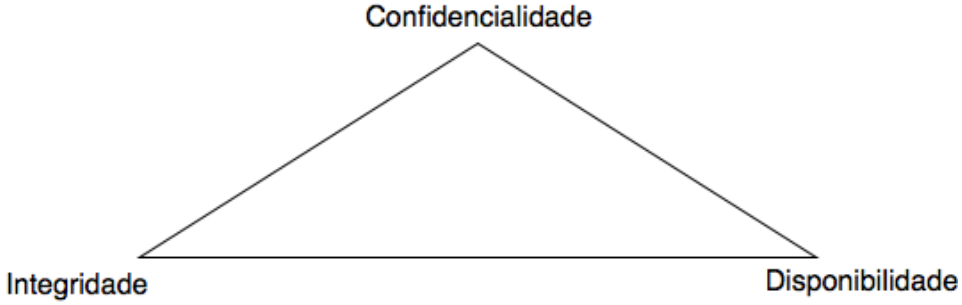


Figura 2: Representação da tríade CIA

A confidencialidade é um conceito que tenta impedir a divulgação não intencional, do conteúdo de uma determinada informação. A perda de confidencialidade pode ocorrer de várias formas, como por meio da divulgação intencional de informações das organizações, ou seja, fuga de informação que pode ocorrer com dolo ou sem dolo. Também pode existir perca de confidencialidade por negligência no manuseamento de mecanismos de segurança. Neste contexto, a confidencialidade pertente assegurar que o nível necessário de sigilo é aplicado em cada operação de processamento de dados e assim impedir a divulgação não autorizada de informação (Tchernykh, Schwiegelsohn, Talbi, & Babenko, 2016).

A confidencialidade deve prevalecer, enquanto os dados residem em sistemas e dispositivos dentro da rede, quando são transmitidos, e quando da chegada ao seu destino. Os atacantes podem tentar iludir os mecanismos de confidencialidade, pela monitorização da rede, praticando *shoulder surfing*, roubo de ficheiros de palavras-chave e através de engenharia social (Schaub, Deyhle, & Weber, 2012).

A integridade visa garantir que não são feitas modificações indevidas a dados, processos e memória de sistemas operativos. Desta forma, pretende garantir a consistência da informação residente nos sistemas de uma determinada organização. Os mecanismos de hardware, software e comunicação devem estar em sintonia e trabalhar em conjunto para garantir em absoluto a consistência da informação e evitar ameaças internas e externas (Shoufan & Damiani, 2017).

A disponibilidade é um conceito que pretende garantir o acesso confiável e oportuno a dados ou recursos de computação por utilizadores legítimos. Dito de outra forma, a disponibilidade pretende garantir que os sistemas funcionem sempre que necessário. Além disso, este conceito pretende garantir que os serviços de segurança estejam em pleno funcionamento. Neste sentido, os sistemas devem ser capazes de se recuperar de interrupções de uma forma segura e rápida para que a produção de sistemas não seja afetada de forma negativa. Assim, pontos únicos de falha devem ser evitados. Devem ser elencadas medidas de apoio a ser

tomadas, mecanismos de redundância devem ser postos em prática quando necessário, e os efeitos negativos da componente ambiental deve ser evitado. Os mecanismos de proteção necessários devem ser implementados com o objetivo de proteger contra as ameaças internas e externas que podem afetar a disponibilidade e a produtividade da rede, sistemas e informações das organizações (Aminzade, 2018).

Os seguintes conceitos são igualmente de enorme importância para a segurança da informação:

Auditabilidade: este conceito visa permitir que os sistemas de informação possuam a capacidade de poderem ser auditados. Um mecanismo comum é a utilização de registos (*logs*) para indagação de ações e eventos nos sistemas de informação (Zerbino, Aloini, Dulmin, & Mininno, 2018).

Privacidade: diz respeito ao nível de confidencialidade e proteção de privacidade dado a um utilizador, num sistema de informação. É geralmente considerado um compromisso importante do controlo de segurança da informação (Naarttijärvi, 2018).

Identificação: considera-se os meios pelos quais os utilizadores assumem sua identidade perante os sistemas de informação. Um exemplo da sua utilização respeita ao controlo de acessos onde a identificação é necessária para autenticação e autorização (Barkadehi, Nilashi, Ibrahim, Zakeri Fardi, & Samad, 2018).

Autorização: são os direitos, reservas e permissões concedidos a um indivíduo ou processo por forma a permitir o acesso a um recurso. Depois que é aferida a identidade e a autenticação de um utilizador ou processo, são estabelecidos os níveis de autorização que determinam a extensão dos direitos no sistema a que um determinado utilizador ou processo podem manter (Aminzade, 2018).

2.2.2 Tipos de Identificação e autenticação

A identificação e autenticação são os pilares da maioria dos sistemas de controlo de acesso. O conceito de identificação é a ação de um utilizador declarar uma identidade perante um sistema. Funciona geralmente na forma de um nome de utilizador, designado no contexto dos sistemas informáticos como *login*. A identificação estabelece e atribui a responsabilidade a um utilizador pelas ações num sistema. A autenticação é uma verificação da identidade reivindicada pelo utilizador é válida, geralmente é executada por meio de uma palavra-passe que é fornecida em conjunto com um nome utilizador no momento da autenticação (Barkadehi et al., 2018). A autenticação pode ser baseada em três tipos distintos:

Tipo 1: Algo que o utilizador conhece, *personal identification number* (PIN) ou uma palavra-passe.

Tipo 2: Algo que o utilizador possui, como um cartão de *automated teller machine* (ATM) ou um *Smartcard*.

Tipo 3: Algo que o utilizador possuir em termos de características físicas, como uma impressão digital ou uma leitura de retina ocular.

2.3 Palavras-passe

As palavras-passe são um meio de autenticação muito utilizado, isso proporciona aos utilizadores mal-intencionados oportunidades para explorarem as fragilidades deste meio de autenticação (Li, Wang, & Sun, 2017). Em sistemas em que as palavras-passe estão guardadas de forma clara, o risco é ainda mais elevado para a garantia da segurança da informação. Por forma a dificultar as ações dos atacantes, as palavras-passe podem ser guardadas sobre o resultado de uma função de resumo criptográfico. Contudo, estas funções se forem implementadas de forma direta e/ou incorreta também colocam em causa a segurança da informação por se tratar de uma simples transposição de texto claro para um resumo criptográfico (*hash)*. Mesmo sendo estas funções irreversíveis são conhecidos estudos de ataques que podem quebrar essa mesma irreversibilidade (Bonneau, Herley, van Oorschot, & Stajano, 2015).

2.3.1 Problemática de segurança das palavras-passe

Sendo a autenticação um requisito para a segurança da informação, torna-se necessário perceber a problemática das palavras-passe (Jose et al., 2016);(Li et al., 2017). Investigação e estudos associados à problemática das palavras-passe têm vindo a ser realizados com muita frequência (Taneski, Hericko, & Brumen, 2014). Um dos primeiros trabalhos desenvolvidos nesta área foi o de Morris e Thompson que descobriram que na maioria dos casos as palavras-passe eram relativamente simples (R. Morris & Thompson, 1979). Trabalhos mais recentes apontam ainda na mesma direção, ou seja, as palavras-passe estão longe de ser aleatórias e complexas (Li et al., 2017). Os utilizadores baseiam as suas palavras-passe em simples palavras de dicionário, e formas simples de digitar como "*qwerty*" ou "*123456*" seguindo combinações do teclado como se ilustra na *Figura 3*. Ur et al. estudaram o comportamento dos utilizadores em relação às palavras-passe e perceberam que os utilizadores consideram as palavras-passe um incómodo (Ur et al., 2016). Neste sentido, os utilizadores abordam esta temática de forma leviana e insegura, utilizando informação pessoal como datas de nascimento ou mnemónicas nas palavras-passe, quando utilizam várias palavras na mesma palavra-passe, em grande maioria dos casos, essas palavras estão fortemente correlacionadas (Li et al., 2017).

Outro comportamento inseguro associado a esta matéria é a utilização das mesmas palavras-passe em diversos sistemas, este comportamento é inseguro na medida que um sistema vulnerável pode colocar em causa a segurança dos restantes sistemas (Ur et al., 2016). Shen el al. referem que, mesmo que não exista uma utilização igual das palavras-passe, os

utilizadores fazem variações textuais ligeiras, como por exemplo, substituindo "*a*" por "@" (Shen, Yu, Xu, Yang, & Guan, 2016).

Figura 3: Combinações comuns utilizadas em palavras-passe

A Figura 3 tem como fonte (wpengine, 2015).

2.3.2 Robustez das palavras-passe

Diversos trabalhos já foram desenvolvidos sobre as métricas da robustez das palavras-passe, sendo que a maioria desses trabalhos identificam inúmeras fragilidades nas medições das palavras-passe. Sendo as métricas da robustez das palavras-passe, estimativas da resistência real, denotam um impacto na segurança da informação das organizações (Safa et al., 2015). Neste sentido, é importante perceber de que forma as métricas se comportam e a forma como tratam da verificação das palavras-passe. As métricas de robustez são funções que recebem uma palavra-passe e classificam com uma pontuação relativa a essa mesma palavra-passe. Desta forma, o principal objetivo da métrica é inibir a criação de uma palavra-passe fraca, e

conduzir o utilizador à criação uma palavra-passe de acordo com uma política, tendo em conta a maximização da segurança (Jose et al., 2016).

No entanto, diversos autores consideram que os medidores utilizados não conduzem a uma palavra-passe robusta, nem ajudam o utilizador a escolher uma boa palavra-passe, simplesmente indicam que a palavra-passe não é suficientemente segura e não explicam o motivo da insegurança (Galbally, Coisel, & Sanchez, 2017b). Galbally et al., consideram que os medidores forçam os utilizadores a contornar as medições, levando-os a criar palavras-passe frágeis, dizem mesmo que na maioria dos casos os medidores de força são ultrapassados colocando a primeira letra em maiúscula e um número no final da palavra-passe original (Galbally, Coisel, & Sanchez, 2017a).

A métrica normalmente associada à verificação da robustez das palavras-passe é a complexidade/entropia, sendo que este conceito refere o quão é difícil descobrir uma palavra-passe. Sahin et al., definem a palavra-passe como apenas uma série finita de caracteres que vêm de algum alfabeto finito, e que define a complexidade de uma determinada palavra-passe sobre algum alfabeto no contexto de um conjunto de regras (Sahin et al., 2015). A entropia da palavra-passe é uma métrica normalmente usada para indicar uma medida de proteção fornecida, por uma política que aumenta com o número e tipologia de caracteres de uma determinada palavra-passe (Li et al., 2017). No entanto, verificou-se que a entropia de Shannon dificilmente pode descrever com precisão o nível de segurança das palavras-passe (Kelley et al., 2012).

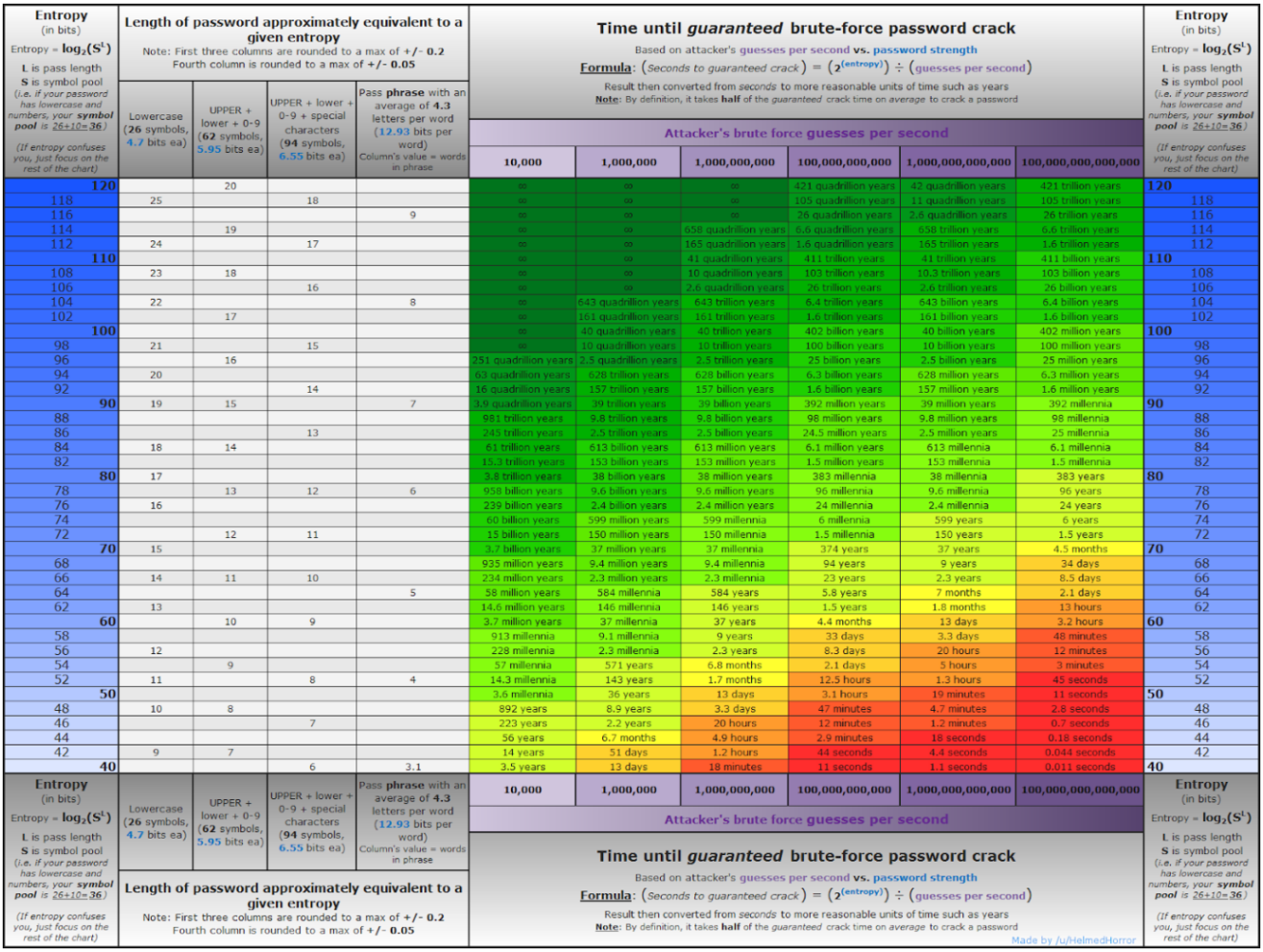
Weir et al., desenvolveu o modelo *Probabilistic Context-Free Grammar* (PCFG) e argumentou que métodos de estimativa da entropia como o recomendado pelo NIST são imprecisos (Weir, Aggarwal, De Medeiros, & Glodek, 2009).

Em junho de 2017, o NIST procedeu a atualizações no documento e disponibiliza a versão B entre outras, faz três sugestões importantes (Fenton et al., 2017).

- Descontinuidade de regras de complexidade, estas regras tornam as palavras-passe mais difíceis de memorizar por parte dos utilizadores, consequentemente leva os utilizadores a negligenciar a escolha da palavra-passe.

- Palavras-passe com datas de expiração, esta política não tem propriamente efeitos positivos, pelo contrário, os utilizadores ultrapassam a validade da palavra-passe adicionando mais um caracter, criando assim uma nova palavra-passe. Além disso, como são guardadas as palavras-passe antigas, ainda vai permitir fornecer mais informação em caso de invasão por parte de atacantes.

- Recomendação da utilização de “chaveiros” digitais: existe software específico (designados por “chaveiros” digitais) que facilitam a criação e a gestão das palavras-passe (Fenton et al., 2017). A Figura 4 ilustra a estimativa de tempos para ataques de força bruta, tendo em conta a entropia da palavra-passe.



Entropy (in bits) Entropy = $\log_2(S^L)$ L is pass length S is symbol pool (i.e. if your password has lowercase and numbers, your symbol pool is 26+10=36) (If entropy confuses you, just focus on the rest of the chart)	Length of password approximately equivalent to a given entropy. Note: First three columns are rounded to a max of +/- 0.2. Fourth column is rounded to a max of +/- 0.05. Lowercase (26 symbols, 4.7 bits ea)	UPPER + lower + 0-9 (62 symbols, 5.95 bits ea)	UPPER + lower + 0-9 + special characters (94 symbols, 6.55 bits ea)	Pass phrase with an average of 4.3 letters per word (12.93 bits per word) Column's value = words in phrase	Time until guaranteed brute-force password crack. Based on attacker's guesses per second vs. password strength. Formula: (Seconds to guaranteed crack) = ($2^{(entropy)}$) ÷ (guesses per second). Result then converted from seconds to more reasonable units of time such as years. Note: By definition, it takes half of the guaranteed crack time on average to crack a password. Attacker's brute force guesses per second: 10,000	1,000,000	1,000,000,000	100,000,000,000	1,000,000,000,000	100,000,000,000,000	Entropy (in bits) Entropy = $\log_2(S^L)$ L is pass length S is symbol pool (i.e. if your password has lowercase and numbers, your symbol pool is 26+10=36) (If entropy confuses you, just focus on the rest of the chart)
120		20			∞	∞	∞	421 quadrillion years	42 quadrillion years	421 trillion years	**120**
118	25		18		∞	∞	∞	105 quadrillion years	11 quadrillion years	105 trillion years	118
116				9	∞	∞	∞	26 quadrillion years	2.6 quadrillion years	26 trillion years	116
114		19			∞	∞	658 quadrillion years	6.6 quadrillion years	658 trillion years	6.6 trillion years	114
112	24		17		∞	∞	165 quadrillion years	1.6 quadrillion years	165 trillion years	1.6 trillion years	112
110					∞	∞	41 quadrillion years	411 trillion years	41 trillion years	411 billion years	**110**
108	23	18			∞	∞	10 quadrillion years	103 trillion years	10.3 trillion years	103 billion years	108
106			16		∞	∞	2.6 quadrillion years	26 trillion years	2.6 trillion years	26 billion years	106
104	22			8	∞	643 quadrillion years	643 trillion years	6.4 trillion years	643 billion years	6.4 billion years	104
102		17			∞	161 quadrillion years	161 trillion years	1.6 trillion years	161 billion years	1.6 billion years	102
100					∞	40 quadrillion years	40 trillion years	402 billion years	40 billion years	402 million years	**100**
98	21		15		∞	10 quadrillion years	10 trillion years	100 billion years	10 billion years	100 million years	98
96		16			251 quadrillion years	2.5 quadrillion years	2.5 trillion years	25 billion years	2.5 billion years	25 million years	96
94	20				63 quadrillion years	628 trillion years	628 billion years	6.3 billion years	628 million years	6.3 million years	94
92			14		16 quadrillion years	157 trillion years	157 billion years	1.6 billion years	157 million years	1.6 million years	92
90	19	15		7	3.9 quadrillion years	39 trillion years	39 billion years	392 million years	39 million years	392 millennia	**90**
88					981 trillion years	9.8 trillion years	9.8 billion years	98 million years	9.8 million years	98 millennia	88
86			13		245 trillion years	2.5 trillion years	2.5 billion years	24.5 million years	2.5 million years	25 millennia	86
84	18	14			61 trillion years	613 billion years	613 million years	6.1 million years	613 millennia	6.1 millennia	84
82					15.3 trillion years	153 billion years	153 million years	1.5 million years	153 millennia	1.5 millennia	82
80	17				3.8 trillion years	38 billion years	38 million years	383 millennia	38 millennia	383 years	**80**
78		13	12	6	958 billion years	9.6 billion years	9.6 million years	96 millennia	9.6 millennia	96 years	78
76	16				239 billion years	2.4 billion years	2.4 million years	24 millennia	2.4 millennia	24 years	76
74					60 billion years	599 million years	599 millennia	6 millennia	599 years	6 years	74
72		12	11		15 billion years	150 million years	150 millennia	1.5 millennia	150 years	1.5 years	72
70	15				3.7 billion years	37 million years	37 millennia	374 years	37 years	4.5 months	**70**
68					935 million years	9.4 million years	9.4 millennia	94 years	9 years	34 days	68
66	14	11	10		234 million years	2.3 million years	2.3 millennia	23 years	2.3 years	8.5 days	66
64				5	58 million years	584 millennia	584 years	5.8 years	7 months	2.1 days	64
62	13				14.6 million years	146 millennia	146 years	1.5 years	1.8 months	13 hours	62
60		10	9		3.7 million years	37 millennia	37 years	4.4 months	13 days	3.2 hours	**60**
58					913 millennia	9.1 millennia	9 years	33 days	3.3 days	48 minutes	58
56	12				228 millennia	2.3 millennia	2.3 years	8.3 days	20 hours	12 minutes	56
54		9			57 millennia	571 years	6.8 months	2.1 days	5 hours	3 minutes	54
52	11		8	4	14.3 millennia	143 years	1.7 months	12.5 hours	1.3 hours	45 seconds	52
50					3.6 millennia	36 years	13 days	3.1 hours	19 minutes	11 seconds	**50**
48	10	8			892 years	8.9 years	3.3 days	47 minutes	4.7 minutes	2.8 seconds	48
46			7		223 years	2.2 years	20 hours	12 minutes	1.2 minutes	0.7 seconds	46
44					56 years	6.7 months	4.9 hours	2.9 minutes	18 seconds	0.18 seconds	44
42	9	7			14 years	51 days	1.2 hours	44 seconds	4.4 seconds	0.044 seconds	42
40			6	3.1	3.5 years	13 days	18 minutes	11 seconds	1.1 seconds	0.011 seconds	**40**
Entry (duplicate)	Lowercase (26 symbols, 4.7 bits ea)	UPPER + lower + 0-9 (62 symbols, 5.95 bits ea)	UPPER + lower + 0-9 + special characters (94 symbols, 6.55 bits ea)	Pass phrase with an average of 4.3 letters per word (12.93 bits per word) Column's value = words in phrase	10,000	1,000,000	1,000,000,000	100,000,000,000	1,000,000,000,000	100,000,000,000,000	Entry (duplicate)
	Length of password approximately equivalent to a given entropy. Note: First three columns are rounded to a max of +/- 0.2. Fourth column is rounded to a max of +/- 0.05.				Attacker's brute force guesses per second. Time until guaranteed brute-force password crack. Based on attacker's guesses per second vs. password strength. Formula: (Seconds to guaranteed crack) = ($2^{(entropy)}$) ÷ (guesses per second). Result then converted from seconds to more reasonable units of time such as years. Note: By definition, it takes half of the guaranteed crack time on average to crack a password. Made by /u/HelmedHorror						

Figura 4: Relação da entropia com o tempo de ataque do modelo de força bruta

A tem Figura 4 como fonte (“Time required to brute-force crack a password depending on password entropy (strength),” 2015).

A Figura 5 ilustra um diagrama que representa de forma generalizada as abordagens dos ataques realizados com vista à quebra das palavras-passes. O retângulo externo representa o espaço de todas as palavras possíveis. O círculo representa o subespaço limitado de palavras-passe selecionadas por utilizadores. Os retângulos internos descrevem os subespaços de palavras-passe criadas por computador para diferentes ataques. Os métodos de ataque descritos são apenas um exemplo típico de subconjunto selecionado para fins ilustrativos. Os retângulos regulares representam ataques baseados em heurísticas, enquanto retângulos de

canto arredondado representam ataques baseados em probabilidades, como por exemplo, as cadeias de Markov em camadas e a cadeia de Markov simples. As áreas sombreadas representam as palavras-passe já conhecidas. Um tom mais escuro corresponde a palavras-passe que, em média, são mais difíceis de quebrar, ou seja, é preciso um número maior de tentativas. A seta inferior representa a força da palavra-passe, em que uma tonalidade mais escura corresponde a palavras-passe mais difíceis de quebrar, ou seja, mais robustas.

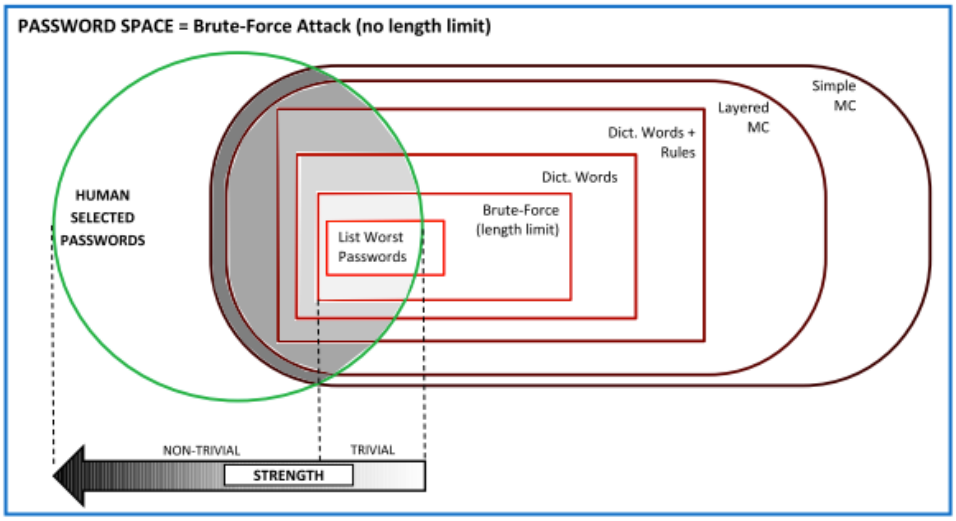


Figura 5: Espaço de palavras-passe em ataques de força bruta

A Figura 5 tem como fonte (Galbally et al., 2017b).

2.3.3 Funções criptográficas de geração de resumos

É usual a utilização de funções criptográficas de geração de resumos para guardar as palavras-passe em segurança. O conceito subjacente a estas funções é tornarem impossível a reversão do resumo (*hash*) no texto original. No entanto, para dificultar a tarefa dos possíveis atacantes, é comum mascarar a palavra-passe original com um *salt*. Assim, mesmo que um atacante consiga reverter o *hash*, não fica a conhecer a palavra-passe, se não conhecer o *salt* que foi usado.

Algumas definições de funções criptográficas de geração de resumo referem que sendo h uma função de geração de resumos, temos uma colisão se dadas duas mensagens x e y, tivermos $h(x) = h(y)$. Outra definição é a que indica que seja x uma palavra, em que uma função de geração de resumo h é francamente livre de colisões se for computacionalmente impraticável encontrar uma palavra $x' \neq x$ tal que $h(x) = h(x')$. Por último uma outra definição, diz que uma função de resumo h é fortemente livre de colisões se for

computacionalmente impraticável encontrar duas mensagens x e x' diferentes tais que $h(x) = h(x')$. A *Figura 6* mostra um esquema alusivo às colisões das funções de resumo.

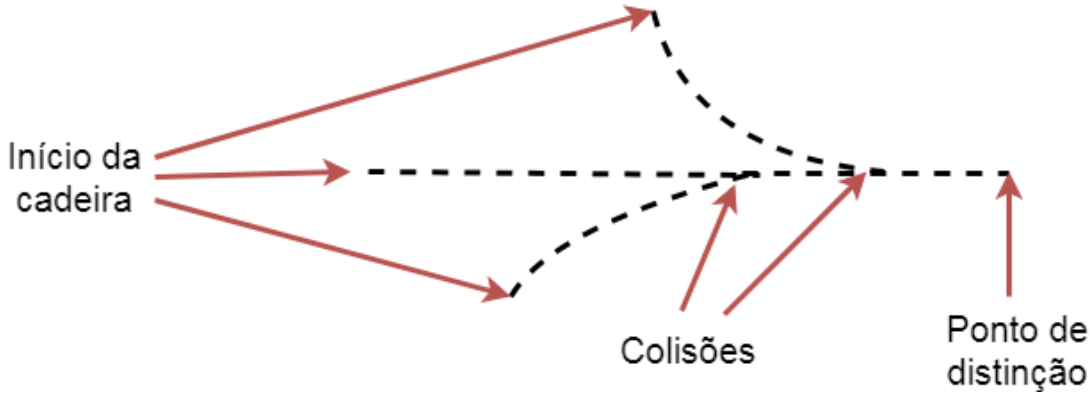


Figura 6: Colisões das cadeias de *hashing*

Existem dois tipos de funções de geração de resumos: baseadas em problemas matemáticos que daí retiram a sua segurança, pois são necessárias provas matemáticas complexas. Contudo, este tipo de funções não é muito utilizado devido à complexidade inerente, sendo que esta complexidade torna a sua operacionalidade muito lenta. No entanto, estes tipos de funções são considerados comprovadamente seguras (Almeida & Napp, 2017). Fazem parte deste tipo as funções baseadas no problema de logaritmos discretos, raízes quadradas modulares e curvas elípticas no problema do *Knapsack* (Maitin-Shepard, Tibouchi, & Aranha, 2017). Estão relacionadas com sistemas criptográficos de McEliece e de Niederreiter e baseado na transformada rápida de Fourier (Vambol, Kharchenko, Potii, & Bardis, 2018).
Sobre o outro tipo de funções de resumo, têm como base não um problema matemático de difícil resolução. Têm como objetivo serem resistentes e difíceis de quebrar, todavia não existem demonstrações formais sobre a resistência das mesmas. Fazem parte deste tipo de funções de resumo, entre outros, o MD5, SHA-1, SHA-2 e WHIRLPOOL (Almeida & Napp, 2017). Existem outras técnicas relacionadas com as funções de resumo, como o SCrypt que é utilizado em moedas digitais como o Litecoin[1], os algoritmos Scrypt e Argon2 tem uma forte dependência a funções sequenciais de memória o que torna mais difícil descobrir o *hash* original. Outra técnica é o PBKDF2 trata-se de uma função de derivação de chave que repete certas funções *hash* criptográficas, muitas vezes na entrada para dificultar a quebra por via de força bruta. (Pan et al., 2017). Por último, outra técnica de *hashing* é o BCrypt que por exemplo, pode ser configurado com um fator de custo o que permite aumentar de forma

[1] https://www.litecoin.com

exponencial o tempo de execução, exigindo uma série sequencial de cálculos (Shay et al., 2016).

2.3.4 Modelos e técnicas de verificação baseados em heurísticas

Este método está associado às recomendações NIST referidas no documento NIST 800-63B (Fenton et al., 2017). Fenton et al., propõem medir a robustez das palavras-passe através do número de bits de entropia, baseou o seu trabalho na teoria de informação de Shannon. A verificação da robustez das palavras-passe, baseadas em heurísticas são conhecidas como *Lowercase letters, Uppercase letters, Digits and Symbols* (LUDS), que visam contar o número de letras maiúsculas e minúsculas, dígitos e símbolos. Embora alguns trabalhos baseados em heurísticas estejam a afastar-se deste paradigma LUDS, a maioria das políticas de palavras-passe implementadas ainda são baseadas nesse conceito (Shay et al., 2016) ;(Wheeler, 2016).

Tipicamente fazem parte deste modelo os ataques de força bruta. Estes ataques são versões mais completas de ataques de dicionário. Este modelo de ataque de força bruta também envolve por norma grandes volumes de palavras-passe. Assim, este modelo recorre a todas as combinações de cada caractere até que uma palavra-passe correta seja encontrada. Este modelo é mais moroso que o modelo de dicionário por considerar um segmento de trabalho de maior amplitude. Um ataque de força bruta será sempre bem-sucedido se existir tempo e espaço suficiente para concluir o ataque. Cada *bit* adicional de comprimento duplica o tempo para executar um ataque de força bruta porque o número de chaves potenciais duplica.

Podemos explicar esse sucesso relacionado com o tempo, dizendo que os computadores que trabalham num sistema binário são proporcionais ao número de operações *bit*. No caso de uma operação de 32 *bits* que pode assumir valores entre 1 e 4.292.967.296, um computador binário para encontrar um número concreto dentro desse intervalo, teria de o fazer *bit* a *bit*. No entanto, se o número tivesse compreendido nesse conjunto seria encontrado. Como as palavras-passe são baseadas em dicionários linguísticos e cadeias de espaços finitos, pode existir um algoritmo de tempo polinomial que resolve a descoberta da palavra-passe.

Número de operações bit $= O(k_1^{d_1} \ \ldots \ k_r^{d_r})$

Uma tarefa é de tempo polinomial se existir um algoritmo de tempo polinomial que resolva esta tarefa. Considera-se assim, operações na classe P.

$c \leftarrow primeiro\ (P)$

enquanto $c \nexists faz\ se\ válido(P, c)$ ***se não*** $saída(P, c)\ c \leftarrow próximo\ (P, c)$

O pseudo-código anterior ilustra um algoritmo de força bruta, que consiste num ciclo iterativo em que cada iteração verifica se (P) pertente ao conjunto c. Se a condição se verificar a iteração termina, caso contrário é chamado o $próximo\ (P)$.

2.3.5 Modelos e técnicas de verificação baseados em ataques

Este modelo de ataque baseia-se na resistência das palavras-passe. Como as palavras-passe são na sua grande maioria previsíveis, e pouco aleatórias porque precisam ser memorizadas, os utilizadores escolhem deliberadamente palavras-passe fáceis de adivinhar (Sahin et al., 2015). Esta modalidade é utilizada em ferramentas automáticas que possibilitam ataques cada vez mais eficientes à medida que o número de iterações aumenta (Galbally et al., 2017b).
Os ataques com recurso a listas pré-elaboradas enquadram-se neste modelo de ataque. Estas listas são denominadas de tabelas arco-íris (*rainbow tables*), que permitem potenciar o modelo da força bruta. As tabelas do arco-íris fornecem valores pré-computados de palavras-passe, normalmente são utilizadas em sistemas que armazenam as palavras-passe sobre funções de geração de resumo. Consideramos que temos uma palavra sobre uma função de resumo denominada H e um conjunto finito de palavras denominadas P. O propósito de possuir uma lista pré calculada que dada qualquer saída de h da função de resumo, consegue localizar o p em P em que $H(p) = h$ ou determinar que não existe p em P.

2.3.6 Modelos e técnicas de verificação baseados em probabilidades

Este método tem como base a avaliação estatística das palavras-passe e a maioria das implementações é feita baseada em modelos de Markov (Galbally et al., 2017b). Algumas técnicas estudadas na modelagem natural da língua são aplicadas neste método, e podem ser aplicadas em abordagens que caracterizam a sequência de caracteres nas palavras-passe (Sahin et al., 2015). Outra empregabilidade do processamento de linguagens naturais neste modelo é a probabilidade gramatical da Figura 7 ilustra a distribuição de frequências na língua portuguesa (Weir et al., 2009).

Tipicamente fazem parte deste modelo os ataques de dicionário que geralmente são ataques *offline*, contra uma palavra-passe. Um ataque de dicionário envolve obter uma lista de palavras, frequentemente um dicionário, que contém uma lista de palavras-passe candidatas. Este modelo consiste em tentar cada palavra até que uma palavra-passe válida seja encontrada, muitas listas de palavras estão disponíveis na Internet em sítios *Web* públicos como *Pastebin*[2], *Reddit*[3] ou *GitHub*[4]. Por forma a melhorar a eficiência das listas, é recorrente utilizar variantes, como palavras seguidas por um ou dois números ou substituições de letras simples como é o caso de tornar a primeira letra maiúscula.

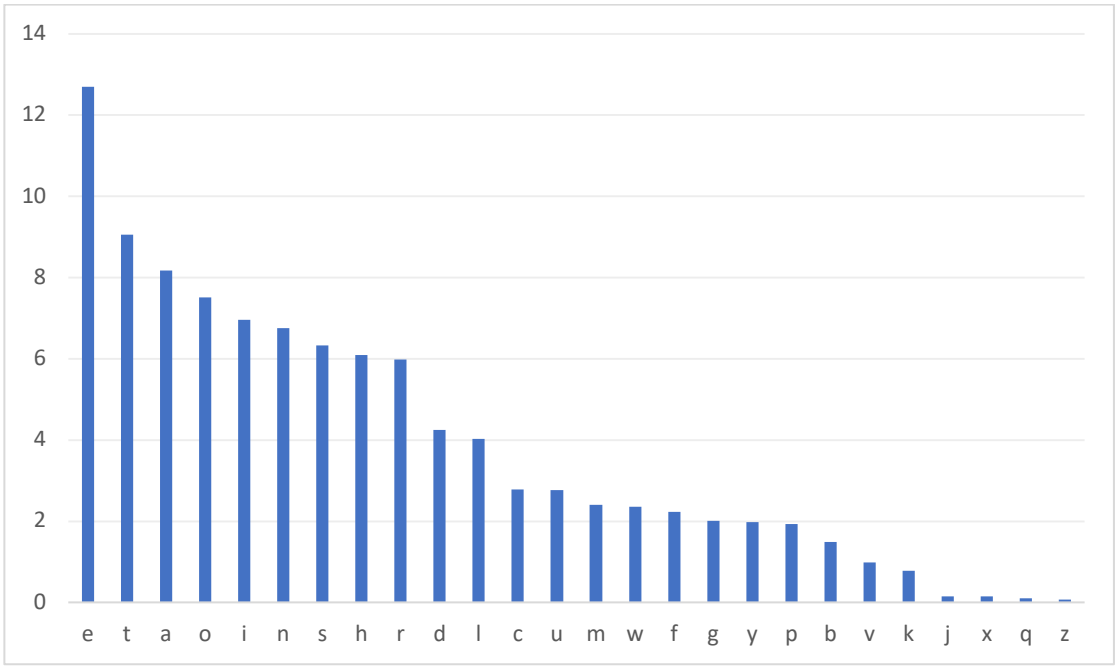


Figura 7: Distribuição de frequências da língua portuguesa

[2] https://pastebin.com/
[3] https://www.reddit.com/
[4] https://github.com

A Figura 7 tem como fonte (Almeida & Napp, 2017).

2.4 Computação distribuída

Ao contrário dos sistemas de *High Performance Computing* (HPC) que necessitam que os recursos de *hardware* como a memória e o *Central Processing Unit* (CPU) tenham uma definição mais rígida, os sistemas distribuídos são amplamente mais dinâmicos (Spenke, Balzer, Frick, Hartke, & Dieterich, 2018). A computação distribuída começou a ganhar importância nos anos 90 do século XX, onde diversos trabalhos contribuíram para este tipo de computação (Chorazyk et al., 2017).

No entanto, não existe propriamente uma definição comum para sistemas distribuídos na literatura, no entanto a maioria das definições conduz-nos a um conceito que pode ser descrito como um conjunto de recursos que trabalham para um mesmo objetivo. Esta descrição denota duas características, se por um lado um sistema de computação distribuída é um conjunto de recursos computacionais, por outro, um sistema distribuído transparece um sentido de unidade, e que na perspetiva dos utilizadores trata-se de apenas um único sistema (van Steen & Tanenbaum, 2016).

Os dispositivos pertencentes a um sistema distribuído são independentes e não partilham equipamentos físicos como a memória ou processadores. Estes dispositivos comunicam entre si utilizando mensagens e informações transmitidas de um dispositivo para outro através da rede. Os sistemas de computação distribuída podem assumir em muitos casos diversas formas, desde computadores de alto desempenho até computadores de menor capacidade conectados entre si. Neste sentido, os recursos são programados para atingir interesses em comum. Um aspeto importante na computação distribuída é a noção de tempo. Como os recursos são normalmente heterogéneos cada recurso interpreta o tempo de forma própria. Assim, num sistema distribuído não existe o conceito de tempo global. Este paradigma da existência de um tempo global levanta questões de especial importância, nomeadamente em relação à coordenação e sincronismo. Um aspeto importante de um sistema distribuído é o facto de ser composto por um conjunto de recursos, e a inevitabilidade desses mesmos recursos falharem. Desta forma, o sistema pode conduzir a um comportamento inesperado em que um único recurso pode inviabilizar todo o trabalho realizado por outros recursos (van Steen & Tanenbaum, 2016).

2.4.1 Arquiteturas de computação distribuída

Bote-Lorenzo et al., classificam os diversos tipos de arquiteturas da computação distribuída (Bote-lorenzo, Dimitriadis, & Gómez-Sánchez, 2004). De acordo com as categorias que se apresentam de seguida.

- **Supercomputadores distribuídos**

Este tipo de computação permite agregar recursos computacional como o propósito de reduzir o tempo de conclusão de um trabalho ou resolver problemas que não podem ser resolvidos num único computador. Este tipo de arquitetura coloca desafios técnicos que incluem a necessidade de recursos dispendiosos, escassos e também do ponto de vista da necessidade de protocolos e algoritmos que possam lidar com inúmeros sistemas heterogéneos. A Figura 8 e a Figura 9 mostram trechos de programação de paralelismo. No caso específico da Figura 8 visualizamos um trecho de código em *OpenML*[5] que tem como propósito o paralelismo de tarefas. No caso da Figura 9 o trecho ilustrado utiliza *Message Passing Interface*[6] (*MPI*) em que o propósito é a comunicação e utiliza a memória de forma distribuída.

```
void simple(int n, float *a, float *b) {
 int i;
#pragma omp parallel for
for (i=1; i< N; i++){
   b([i] = a[i] * 3.1416;
}
```

Figura 8: Exemplo de um trecho de código em *OpenMP*

A Figura 8 tem como fonte (Breternitz, 2017).

[5] https://www.openml.org
[6] https://www.open-mpi.org

```
if (my_rank == 0) {
    number = -1;
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
} else if (my_rank == 1) {
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n",
           number);
}
```

Figura 9: Exemplo de um trecho de código em MPI

A Figura 9 tem como fonte (Breternitz, 2017)

- **Computação de elevado rendimento**

Esta arquitetura é usada para agendar um elevado número de tarefas fracamente acopladas ou independentes, com o objetivo de alocar ciclos de processador não utilizados, muitas vezes de estações de trabalho inativas. A Figura 10 ilustra um tipo fluxo utilizado na computação de alto rendimento.

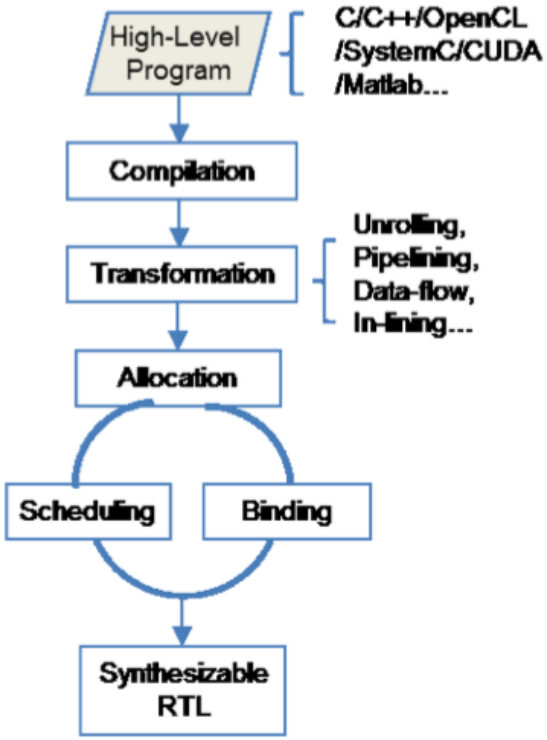


Figura 10:Tipico fluxo em computação de elevado rendimento

A Figura 10 tem como fonte (Kim, Chen, Xiong, & Hwu, 2017)

- **Computação a pedido**

Esta arquitetura permite acoplar recursos ou serviços remotos que podem ser consumidos. Associado a esta arquitetura está a natureza dinâmica das necessidades dos utilizadores.

- **Computação intensiva de dados**

O propósito desta arquitetura é o processamento intensivo de grandes volumes de dados promovendo tratamento de informação.

- **Computação voluntária**

O conceito principal associado a esta arquitetura é a utilização de ciclos ociosos dos sistemas dos utilizadores voluntários. Em contraste com os supercomputadores e a computação de alto rendimento, nesta arquitetura, o trabalho é delegado pelo servidor para os diferentes nós voluntários. Assim, os voluntários solicitam ao servidor, tarefas disponíveis, podem receber algumas delas para contribuir com o seu *hardware*. Por se tratar de trabalho voluntário, isso pode oferecer alguma imprevisibilidade na conclusão das tarefas, o que pode colocar questões do âmbito de confiabilidade e integridade.

2.4.2 Distribuição e gestão de tarefas

O agendamento de tarefas é parte integrante da computação distribuída. Agendar tarefas em recursos heterogêneos num sistema de computação distribuída é um problema de tempo polinomial não determinístico completo (Reda, Tawfik, Marzok, & Khamis, 2014).
Podem existir muitos nós candidatos à execução de uma tarefa, a primeira pergunta a ser respondida é como atribuir as tarefas. Este procedimento é conhecido como alocação de tarefas ou agendamento global, e é realizado por algoritmos de agendamento (He, Hsu, & Leiserson, 2008).
Chandak et al., define que um algoritmo de agendamento pode ser classificado em clarividente ou não clarividente, no que diz respeito ao conhecimento das características das tarefas. Um algoritmo de agendamento clarividente pode usar informações das características de tarefas para promover uma melhor demanda, enquanto um algoritmo não clarividente não

assume nada sobre as características dos trabalhos (Chandak, 2011). Leitner et al., propõem um algoritmo de agendamento de reconhecimento de recurso que aproveita dois algoritmos de agendamento de tarefas existentes, Min-min e Max-min. Usa uma estimativa do tempo de conclusão das tarefas e do tempo de execução dos recursos. O algoritmo apresentado, combinada entre os dois algoritmos, dependendo das tarefas a tratar (Leitner, Hummer, Satzger, Inzinger, & Dustdar, 2012). Olteanu, A. et al., propõem um algoritmo dinâmico que é adequado para tarefas de restrições arbitrárias, as suas dependências podem ser organizadas como um grafo, tendo como nós as tarefas e como arestas as restrições (Olteanu, Pop, Dobre, & Cristea, 2012). Foster, I. et al., apresentam um algoritmo que suporta *Quality of Service* (QoS) para execução em *clouds* híbridas, nesta modalidade de *Cloud Computing*, a infraestrutura de *cloud* é uma composição de duas ou mais distintas infraestruturas de *Cloud* (privada, comunitária ou publica). Os autores apresentam resultados da execução do algoritmo numa *cloud* híbrida real, com aplicações que exigem apenas uma pequena quantidade de dados. O algoritmo proposto reduz os tempos de execução (Foster & Llorente, 2009). Durillo, J. et al., apresem um artigo sobre o agendamento de fluxo de trabalho eficiente, tendo em consideração a energia. Os autores descrevem um sistema de agendamento baseado em heurísticas com o objetivo de encontrar soluções ótimas e equilibradas na distribuição de tarefas e ponderando o consumo energético (Durillo, Nae, & Prodan, 2014).

Durillo, J. et al. apresem um artigo sobre o agendamento de fluxo de trabalho eficiente, tendo em consideração a energia consumida. Os autores descrevem um sistema de agendamento baseado em heurísticas com o objetivo de encontrar soluções ótimas e equilibradas na distribuição de tarefas e ponderando o consumo energético (Durillo et al., 2014). Alebrahim e Ahmad propõem um algoritmo que é baseado em diferentes grafos acíclicos direcionados, estes grafos sem ciclo são gerados aleatoriamente. Avaliam os resultados comparando o algoritmo com outras abordagens e mostram a eficácia do algoritmo na redução do *makespan* de execução (Alebrahim & Ahmad, 2016).

É essencial que a gestão das tarefas em computação distribuída seja capaz de tolerar falhas e evitar perdas de trabalho. Para isso os sistemas devem oferecer qualidade de serviço (QoS). Idris et al. propõem um algoritmo baseado na otimização em colonia, e visa assegurar que as tarefas são efetivamente executadas com sucesso, mesmo quando ocorre falhas de recursos. Assim, os autores utilizam uma técnica de uso por taxa de falhas de recursos, bem como uma estratégia de recuperação baseada em pontos de verificação que visam reduzir a quantidade de trabalho que é perdido, evitando assim recomeçar do zero. Esses pontos de verificação

funcionam numa perspetival de estados do sistema (Idris, Ezugwu, Junaidu, & Adewumi, 2017).

A Tabela 1 ilustra um conjunto de métricas de desempenho do tratamento das tarefas na computação distribuída.

Referência	Parâmetro	Descrição
(Kanani & Maniyar, 2015)	Makespan	Diz respeito à distância no tempo que decorre desde o início do trabalho até ao fim
(Ding, Kang, & Wang, 2013)	Utilização de recursos	A utilização de recursos é definida como a quantidade de recursos ocupada na executando tarefas.
(Dai & Wang, 2006)	Confiabilidade de Serviço	É definido como a) Acessibilidade - O serviço está disponível quando é solicitado. b) Continuidade - O cliente tem um serviço consistente para o tempo solicitado. c) Desempenho - Atende à expectativa do consumidor
(Ding et al., 2013)	Desvio de equidade	Equidade na utilização de recursos. Aborda o conceito de oportunidade igual para os clientes disponibilizarem os seus recursos e assim podem obter um lucro justo de acordo com sua capacidade.
(Ding et al., 2013)	Execução de tarefas com sucesso	A execução de tarefas com sucesso significa que a tarefa é executada dentro do prazo.
(Dai & Wang, 2006)	Tempo de resposta	O tempo de resposta de uma tarefa é o intervalo de tempo entre uma determinada tarefa chegar ao cliente até que seja concluída.

Tabela 1: Métricas de desempenho das tarefas na computação distribuída.

2.4.3 Protocolos de comunicação

Um sistema de computação distribuída é composto a partir de protocolos e interfaces de vários tipos que abordam questões fundamentais como autenticação, autorização, descoberta e acesso a recursos. Num ambiente distribuído em larga escala, cada recurso é integrado a partir de múltiplas instituições, cada uma com as suas próprias políticas e mecanismos. Portanto, é importante que esses protocolos e interfaces sejam padronizados (Berman et al., 2001).

A Tabela 2 apresenta uma visão geral de um conjunto de protocolos utilizados na computação distribuída.

Referência	**Protocolo**	**Descrição**
(Raja & Noordeen, 2017)	Grid Information Protocol (GRIP)	É um protocolo para a descoberta de novos recursos distribuídos e consulta de recursos de conhecidos.
(Raja & Noordeen, 2017)	Grid Registration Protocol (GRRP)	É usado por vários componentes e serviços de monitorização e descoberta de recursos dentro do ambiente distribuído, como endereços IP, para informar outros componentes sobre sua existência.
(Ali, 2014)	Grid Security Infrastructure File Transfer Protocol (GSIFTP)	É um protocolo de transferência de arquivos criado sobre a infraestrutura de segurança do sistema distribuído. Permite a transferência segura de dados entre componentes distribuídos.
(McEwan, 2010)	Grid Service Reference (GSR)	Contém informações específicas da instância do serviço distribuído, como ligação de protocolo, definição de método e endereço de rede.
(Aloisio, Cafaro, & Epicoco, 2002)	GridFTP	Este protocolo fornece a capacidade de fazer transferência de forma segura, robusta, rápida e eficiente de grandes volumes de dados num ambiente distribuído.
(Komarov, Konovalov, & Kazantsev, 2016)	Remote Procedure Call (RPC)	É um protocolo que permite que um programa em execução numa determinada máquina invoque um procedimento noutra máquina remota.
(S, Kaviyaraj, & Susmitha, 2018)	Secure socket layer (SSL)	É um protocolo para comunicação segura em redes heterogenias. O SSL usa um par de chaves (pública / privada). A chave pública é conhecida por todos e a chave privada ou secreta é conhecida apenas pelo destinatário da mensagem criptografada.
(Young, 2018)	Security assertion markup language (SAML)	É um protocolo padrão baseado em XML que suporta a troca de informações de identidade em diferentes ambientes.
(Anwar Hayder	Simple object	É um protocolo de comunicação baseado em

& Hassan Husain, 2018)	access protocol (SOAP)	XML, que pode ser usado por duas partes que se comunicam numa rede heterogenia.

Tabela 2: Protocolos utilizados na computação distribuída.

2.4.4 Computação distribuída baseada na Web

Komarov et al., apresentam uma comparação das gerações de tecnologias e paradigmas associadas à computação distribuída realizada através do browser *Web* (Komarov et al., 2016).

Na primeira geração de propostas de computação voluntária baseada na WWW, os trabalhos foram delineados e implementados recorrendo ao uso de *applets* Java. Os *applets* continham os dados de entrada necessários ou recorriam a um servidor, solicitando trabalho. Depois de terminar o trabalho, o *applet* enviaria os resultados e solicitaria outra tarefa para ser executada. Algumas soluções permitiam computação *off-line,* ou seja, um utilizador tinha a possibilidade de descarregar o *applet*, desconectar-se da rede, realizar o trabalho computacional e posteriormente o *applet* enviaria os resultados. Umas das primeiras implementações utilizando esta metodologia foi o *Charlotte* (Komarov et al., 2016).

Embora o *Javascript* remonte à década de 90 do século XX, foram as técnicas AJAX que, a partir de 2005, permitiram o desenvolvimento de um novo conjunto de soluções baseadas no *browser Web*. Nas abordagens da segunda geração, um utilizador poderia contribuir para um sistema distribuído visitando simplesmente uma página *Web*. Por exemplo, na proposta de Boldrin et. al., existiam clientes com um papel de contribuintes para a resolução de um problema e servidores responsáveis por particionar o problema, distribuindo tarefas, e posteriormente recolhendo o trabalho dos clientes, ou seja, agregando os resultados (Boldrin, Taddia, & Mazzini, 2007). Assim, poderia manter um controlo dos clientes. Komarov considera que a segunda geração termina em 2009, com duas implementações de prova de conceito, de *hashing* distribuído e com recurso a *MapReduce* que permite processar grandes volumes de dados em paralelo e foi desenvolvido por Ilya Grigorik (Komarov et al., 2016); (Dean & Ghemawat, 2008). Grigorik propõe que um nó contribuinte faça primeiro um pedido HTTP GET, solicitando ao servidor e recebendo uma unidade de trabalho alocada que redireciona para um sítio *Web* devidamente preparado, que contém principalmente código *Javascript* para ser executado. Esse código conteria funções típicas de *MapReduce,* e em seguida, submeteria os resultados de volta ao servidor, criando um formulário invisível, preenchendo-o com os resultados e enviando-o ao servidor.

As soluções que surgem depois de 2010 com a terceira geração de computação voluntária baseada no *browser Web* foram fortemente impulsionadas com o *Chrome* V8[7] do *Google* em 2008 (Komarov et al., 2016). Novos mecanismos de *Javascript*, com compiladores em tempo real permitiram que o código *Javascript* pudesse rivalizar em termos de desempenho com o Java. Com o surgimento dos *WebWorkers*[8] em 2010 finalmente permitiu o *multithreading* em aplicações *Javascript* (Komarov et al., 2016). A Figura 11 ilustra um modelo de eventos, onde demostra as relações entre o *script* principal e um *worker*. Na terceira geração, os contribuintes normalmente carregam uma página com *Javascript*, e então solicitam trabalho ao servidor enviando uma solicitação HTTP GET (Komarov et al., 2016). A Tabela 3 apresenta um resumo das gerações da computação distribuída baseada no *browser Web*.

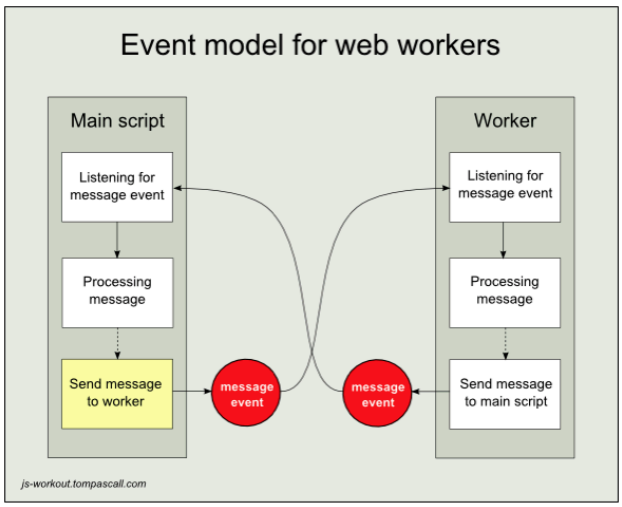


Figura 11: Modelo de eventos de *WebWorkers*

A Figura 11 tem como fonte (Mahoney, 2018).

	1ª Geração	**2ª Geração**	**3ª Geração**
Tarefas escritas	*Java Applets*	*Javascript*	*Javascript*
Servidor HTTP	Utilização fraca	Utilização predominante	Utilização predominante
Suporte a *threads*	Sim, com recurso a *Java Virtual Machine*	Nenhum	Sim, com recurso a *WebWorkers*
Suporte a REST	Nenhum	Ocasional	Comum

[7] https://github.com/v8/v8/wiki/Introduction
[8] https://developer.mozilla.org/pt-PT/docs/Web/API/Web_Workers_API

Protocolos	HTTP, UTP, TCP, Java RMI	HTTP e AJAX	HTTP, AJAX e *WebSockets*

Tabela 3: Resumo das gerações da computação distribuída via *Web Browser*.

2.5 Trabalhos relacionados

Nesta secção serão descritos alguns projetos relacionados e as suas características, os objetivos e aspetos de operabilidade de cada um deles.

2.5.1 BOINC

É uma *framework* de computação distribuída desenvolvida pela Universidade de Berkeley. O projeto *Berkeley Open Infrastructure for Network Computing*[9] (BOINC) é dos projetos de mais populares dentro da computação distribuída (G. J. Martínez & Val, 2015). Em maio de 2018 conta com 35 projetos associados como *World Community Grid*, *SETI@Home*, *ClimatePrediction.net*, *Einstein@home* entre muitos outros. A computação distribuída numa modalidade de voluntariado como é o caso do BOINC, permite criar um poder de computação de enorme capacidade, estando várias vezes no topo de lista do *Top 500*[10] que é uma classificação dos 500 supercomputadores comerciais mais poderosos do mundo (Chorazyk et al., 2017). O BOINC funciona em múltiplas plataformas como Windows, *Linux*, *MacOS*, *Android* e na consola de jogos *PlayStation* (G. J. Martínez & Val, 2015). A Tabela 4 descreve os 10 maiores contribuintes do BOINC.

#	**Participante**	**Contribuição em GFLOPS**	**País**
1	valterc	69,138	Itália
2	Alejandro V. Mena	64,53	México
3	stoneageman	58,272	Reino Unido
4	EG	51,036	Tuvalu

[9] https://boinc.berkeley.edu
[10] https://www.top500.org/

5	Syracuse University	45,609	E. U. América
6	HA-SOFT, s.r.o	39,468	República Checa
7	pool.gridcoin.co	39,18	Internacional
8	oldjerryANO2011	35,244	República Checa
9	Roald	34,398	Reino da Noruega
10	Anchesa	29,824	Internacional

Tabela 4: Os dez dos maiores contribuidores do projeto BOINC

A Tabela 4 tem como fonte (Universidade de Berkeley, 2017)

2.5.2 Globus Toolkit

O *Globus Toolkit*[11] foi desenvolvido no final dos anos 90 para apoiar a criação de aplicações de computação distribuída, orientadas a serviços e infraestruturas. Existem componentes dedicados ao tratamento de questões relacionadas com a segurança, acesso, gestão de recursos e transferência de dados. Esses componentes, por sua vez, foram usados para desenvolver uma ampla gama de suporte a aplicações distribuídas por exemplo (*caBIG* e *Advanced Resource Connector*). A principal característica da versão mais recente é ser baseado em serviços da *Web*, que fornece melhorias significativas em relação às versões anteriores em termos de robustez, desempenho, usabilidade, documentação, conformidade com padrões e funcionalidade disponíveis (Le-Khac, Kechadi, & Carthy, 2017)

[11] http://toolkit.globus.org/toolkit/

2.5.3 The Exascale Computing Project

O projeto *The Exascale Computing Project* [12] é desenvolvido pelo departamento de energia dos Estados Unidos da América. Teve início em 2016 como parte de um esforço coordenado para alcançar a próxima geração de HPC que permitisse acelerar a descoberta científica e competitividade económica.

Além de possibilitar a inclusão de aplicações de supercomputação tradicionais neste projeto, ainda visa contribuir para a convergência emergente de supercomputação, análise de grandes volumes de dados e aprendizagem automática numa ampla variedade de domínios da ciência e engenharia. Este projeto tem como objetivo promover todos os aspetos do ecossistema de computação: aplicações que exigem computação escalável que permitam lidarem com problemas anteriormente intratáveis.

2.5.4 QMachine

O *QMachine*[13] é um projeto de domínio público, distribuído em código aberto que atua como um sistema de mensagens para distribuir tarefas e recuperar resultados por HTTP. Qualquer *browser* atual pode enviar tarefas e voluntariar-se para executá-las sem instalar nenhum *plug-in* ou programa extra. Uma biblioteca cliente fornece modelos de distribuição de alto nível, incluindo o *MapReduce*. O *QMachine* contém três componentes: um servidor de *Application Programming Interface* (API), um servidor da *Web* e um *Website*. A API e os servidores da web são escritos completamente em *Javascript* e o *Website* é desenvolvido em HTML5, CSS e *Javascript*.

2.5.5 ComcuteJS

O projeto ComcuteJS[14] tem uma arquitetura composta por três camadas: A camada *Comcute Core* que é responsável pelo controlo das tarefas no particionamento dos trabalhos e agregação de resultados. A camada *Computational Task Dispatcher* pode consistir em muitos nós, agrupados em conjuntos, cada conjunto é controlado por um nó da camada central o *Gateway Core*. Por último a camada *Computational Workers* que é responsável pela execução das tarefas computacionais.

O *ComcuteJS* foi criado com o uso do *ComcutePDK*, que é destinada a suportar a construção de plataformas baseadas no browser *Web* para cálculos em grande escala. Estes três *Software*

[12] https://www.exascaleproject.org/exascale-computing-project/.
[13] https://www.qmachine.org/
[14] https://journals.agh.edu.pl/csci/article/view/113

Development Kit (SDK) são chamados: *ComcuteJDK*, *ComcutePDK* e *ComcuteCDK*. Cada SDK é baseado numa tecnologia diferente, e é dedicado a diferentes tipos de sistemas diferindo, por exemplo, em escalabilidade e custo de implementação. A Figura 12 mostra o diagrama de sequência desta solução.

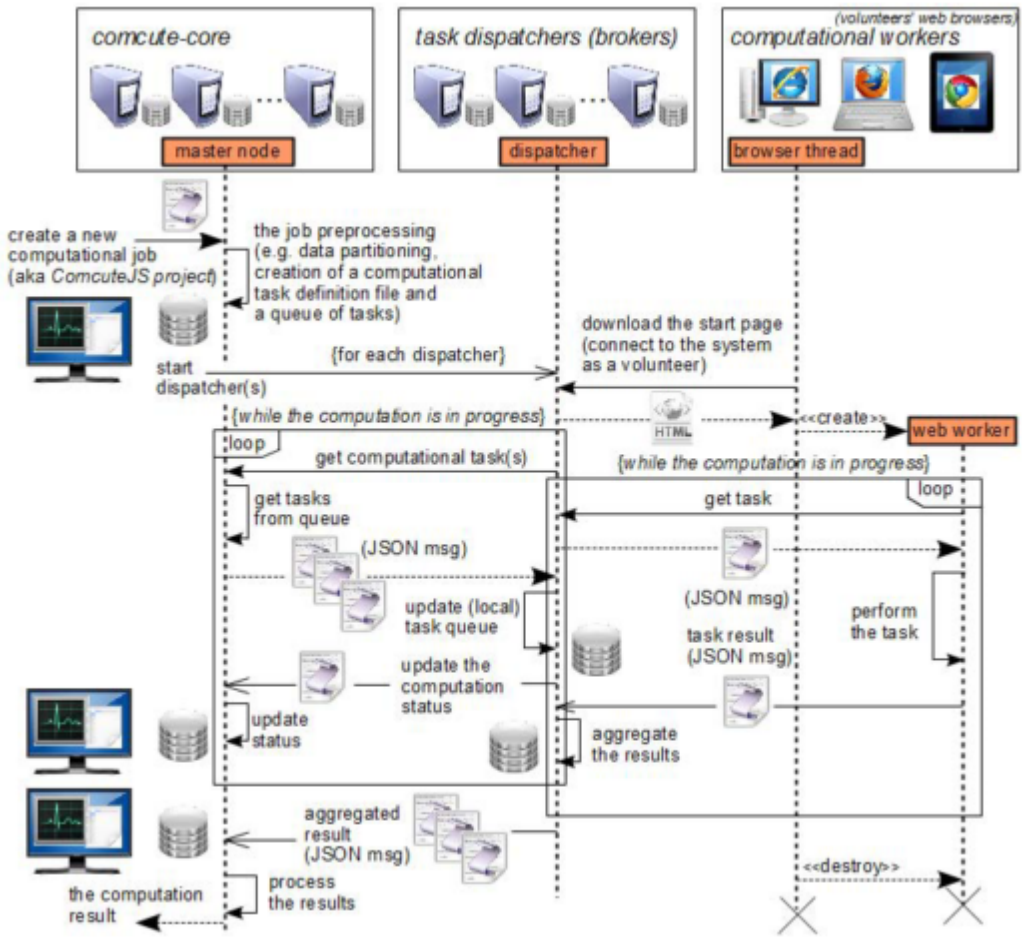


Figura 12: Diagrama de sequência do projeto *ComcuteJS*

A Figura 12 tem como fonte (Debski, Krupa, & Majewski, 2013)

2.5.6 CrowdCL

O *CrowdCL*[15] é uma framework de código aberto que promove o desenvolvimento rápido de computação voluntária e aplicações *Open Computing Language* (OpenCL) na *web*. É inspirado nas bibliotecas existentes, como o PyCUDA[16]. O CrowdCL fornece uma camada de abstração para o *Web Computing Language* (WebCL) que visa reduzir o código fonte e melhorar a sua legibilidade. O *CrowdCL* também fornece aos programadores uma estrutura para executar facilmente cálculos em ambiente de fundo numa página da *Web*, o que permite aos programadores distribuir tarefas numa rede de clientes voluntários e agregar resultados num servidor central. O *CrowdCL* está disponível no *GitHub*. As últimas mudanças no código bases foram introduzidas em 2013, o que indica que o projeto não teve mais nenhum progresso até à data. O *WebCL*. A única maneira de usar no *Mozilla Firefox* ou no *Google Chrome* é através de *plug-ins* adicionais. A Figura 13 ilustra a arquitetura do projeto *CrowdCL*.

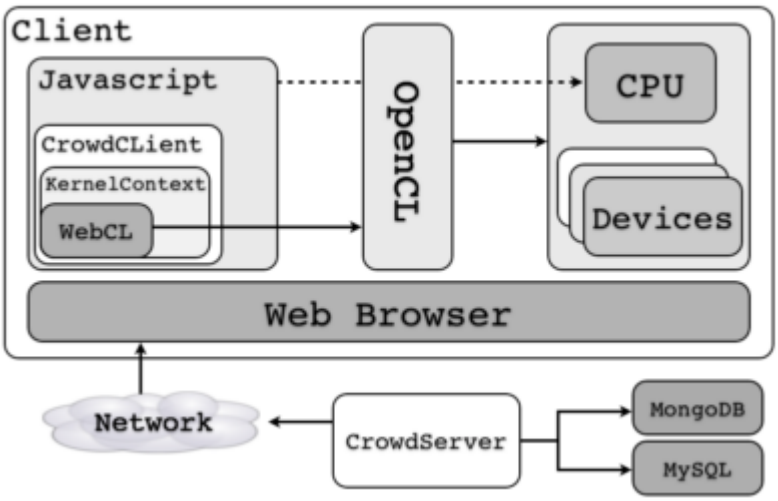


Figura 13: Arquitetura do projeto *CrowdCL*

A Figura 13 tem como fonte (MacWilliam & Cecka, 2013)

[15] https://github.com/tmacwill/crowdcl
[16] https://documen.tician.de/pycuda/

2.5.7 Cracklord

O *CrackLord*[17] é um projeto que tem como objetivo disponibilizar um sistema escalável e distribuído que permite não só quebrar palavras-passe, bem como realizar outros trabalhos que possam ser submetidos. Este projeto permite gerir recursos como o CPU e *Graphics Processing Unit* (GPU) dos clientes voluntários, de forma a criar um único serviço. O sistema está subdividido em três serviços.

(1) *Queue:* é um serviço que fornece uma interface para os clientes gerirem trabalhos.

(2) *Resources:* é um serviço que fornece acesso ao *hardware* existente. (por exemplo CPU e GPU).

(3) *Tools:* são um conjunto de *plugins*, configurados em recursos, que executam as tarefas subjacentes, como a execução das ferramentas : *John the Ripper*[18] (ver secção 5.3.1), *Hashcat*[19] (ver secção 5.3.2), e *Nmap*[20] (L. Morris & McAtee, 2005).

2.5.8 Outros trabalhos

Boldrin et Al., propuseram uma abordagem para a computação distribuída que incorporava AJAX (G. J. Martínez & Val, 2015). Krupa et al., propuseram uma arquitetura de computação voluntária baseada no *browser Web* (Pan, White, Sun, & Gray, 2015). Outro trabalho de Konishi et al., que avaliou a computação baseada no navegador com recurso a AJAX e o desempenho comparativo do *Javascript*, este trabalho, incluía algoritmos para verificar a robustez das palavras-passe (Konishi, Ohki, Ishii, Umestu, & Konagaya, 2007). Martínez et al., propuseram uma biblioteca denominada Capataz[21] que permite executar algoritmos distribuídos por via de *browsers Web* sem recurso a software adicional (G. Martínez, Val, Martinez, & Val, 2014).

[17] https://github.com/jmmcatee/cracklord
[18] http://openwall.com/john/
[19] https://Hashcat.net/Hashcat/
[20] https://nmap.org
[21] https://github.com/LeonardoVal/capataz.js

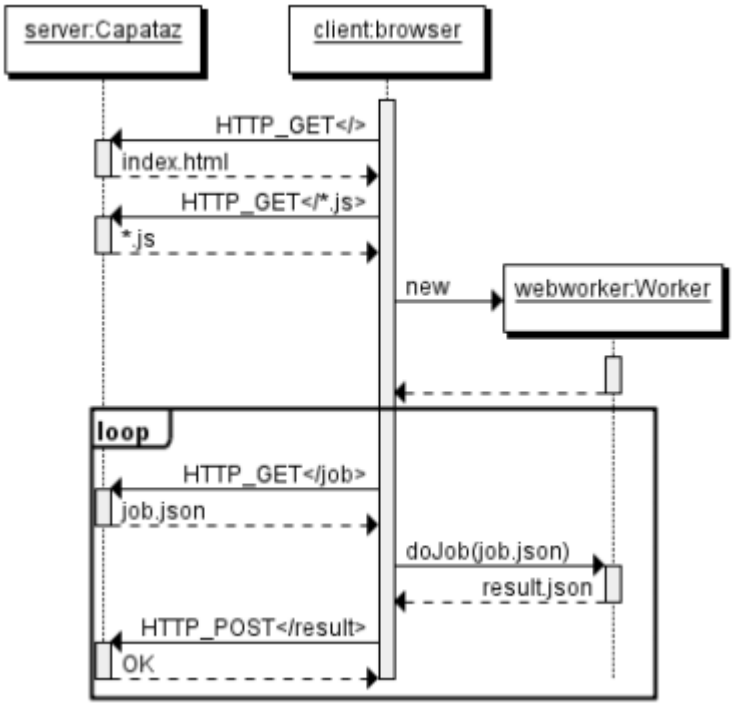


Figura 14: Diagrama de sequência do projeto Capataz

A Figura 14 tem como fonte (G. J. Martínez & Val, 2015).

2.6 Conclusões

A computação distribuída suporta projetos de inúmeros propósitos, veja-se o caso do BOINC com um conjunto alargado de projetos associados. Também existem projetos de grandes dimensões que pretendem ter um impacto na sociedade de grande envergadura, como o caso do "*The Exascale Computing Project*". Muitos destes projetos são fortemente associados a investigação científica e com patrocínios estatais. Existem projetos como o caso do *Capataz* de âmbito académico, no entanto, também existem projetos que denotam funcionalidades de especial interesse como é o caso do *CrackLord*. Do ponto de vista técnico, diversos trabalhos baseados no *browser Web* têm vindo a surgir, no entanto a maioria deles recorre a software complementar para conseguir tirar melhor partido do *hardware* dos voluntários. Não obstante, os projetos de computação distribuída com maior dimensão e disseminação não são baseados no *browser Web*.

2.6.1 Comparação entre trabalhos

Na Tabela 5 são comparadas as características dos diversos trabalhos relacionados apresentados, tendo em consideração as suas principais funcionalidades, capacidades, tipologia de solução e adequação aos objetivos definidos na secção 1.3.

Característica 1 (C1): Possibilidade de um cliente adicionar uma palavra-passe a ser verificada de forma distribuída;

Característica 2 (C2): Visualização dos resultados através de relatórios automáticos, dispostos por tabelas e gráficos que possibilitem aferir a robustez da palavras-passe;

Característica 3 (C3): Configuração a adaptação do sistema, para permitir a variabilidade de testes de robustez e de ataques contra palavras-chave a serem realizados;

Característica 4 (C4): Criar um sistema distribuído que deve ser capaz de lidar com eventuais falhas dos nós cliente;

Característica 5 (C5): O sistema deve funcionar em browser padrão sem recurso a software complementar.

Projetos	**C 1**	**C 2**	**C 3**	**C 4**	**C 5**
BOINC	Não	Não	Não	Sim	Não
Globus Toolkit	Não	Não	Não	Sim	Não
The Exascale Computing Project	Não	Não	Não	Sim	Não
QMachine	Não	Não	Não	Sim	Sim
ComcuteJS	Não	Não	Não	Sim	Sim
CrowdCL	Não	Não	Não	Sim	Não
Cracklord	Sim	Sim	Sim	Sim	Não
Capataz	Não	Não	Não	Não	Sim

Tabela 5: Quadro das caraterísticas dos vários projetos

Capítulo 3 Conceptualização da solução

Ao longo deste capítulo irão ser apresentadas as principais funcionalidades definidas para o artefacto (sistema que pretende validar a robustez das palavras-passe recorrendo a computação distribuída) e o desenho e definição da arquitetura subjacente. Irão igualmente ser apresentados alguns diagramas estruturais mais relevantes, serão abordadas as tecnologias utilizadas.

3.1 Funcionalidades

Pretende-se nesta secção identificar quais foram as funcionalidades de alto nível que foram consideradas para o desenvolvimento do artefacto. Assim, e tendo em conta a lista de questões apresentada na secção 1.4, identificamos as seguintes funcionalidades.

- Permitir que os utilizadores possam introduzir uma palavra-passe a ser verificada pelo sistema de forma distribuída.

- Dotar o sistema com capacidade de administração, através de uma conta de administração específica para o efeito.

- Permitir que os utilizadores possam escolher o tipo de algoritmo de *hash* a utilizar na verificação das palavras-passe, definir o conjunto de caracteres e refinar o conjunto de dados como por exemplo só considerar algarismos numéricos, alfanuméricos ou ambos. Permitindo assim, definir uma segmentação do conjunto de dados por forma a tornar a uma verificação mais simples ou mais abrangente.

- Dotar o sistema de funcionalidades de visualização de informação nomeadamente gráficas e textuais.

- Permitir a produção de relatórios que possam traduzir os trabalhos voluntários realizados e a sua assertividade tento em conta a verificação da robustez das palavras-passe.

3.1.1 Mapa estrutural de funcionalidades

A Figura 15 ilustra o mapa das funcionalidades da solução. Fazem parte do mapa 5 nós que se descrevem em seguida. Para compreender esses caminhos o mapa apresenta os mesmos de forma numérica e colorida.

(0) Utilizador: cada nó se inicia por um utilizador que pode seguir um caminho estrutural. Os detalhes de cada um dos nós são apresentados em seguida:

(1) Autenticação como administrador: diz respeito à autenticação como administrador, aqui o utilizador tem a possibilidade de se autenticar e poder fazer a administração da plataforma.

(1.1) Administração do sistema: visa administrar a plataforma, o utilizador tem a capacidade de seguir para o nó.

(1.1.1) Lista de palavras: que permite ao administrador enriquecer as listas de palavras os adicionando novas entradas.

(1.1.2) Gerir trabalhos: o administrador tem a capacidade de proceder à gestão de trabalhos na plataforma.

(1.2) Gerir credenciais: possibilita a gestão das credenciais do administrador.

(2) Consultar trabalhos: abrange a consulta de trabalhos, permite seguir o nó 2.1 que tem como propósito a visualização de relatórios que por sua vez ainda se estende ao nó 2.1.1 que define a modalidade dos relatórios a visualizar. Ainda no nó 2 existe outra ramificação com a numeração 2.2. Este nó, visa cobrir a capacidade de o utilizador visualizar os trabalhos inseridos na plataforma.

(3) Registar trabalhos: abrange o registo de trabalhos, este caminho conduz aos nós, 3.1 trabalho na modalidade força bruta. Estes vão ser os modelos de ataque que vai ser implementado na solução, e será descrito em maior detalhe na secção 4.3.4.

(4) Ser voluntário: tem como propósito permitir o utilizador dar-se como voluntário e assim permite a utilização do seu *hardware* para resolver trabalhos na plataforma.

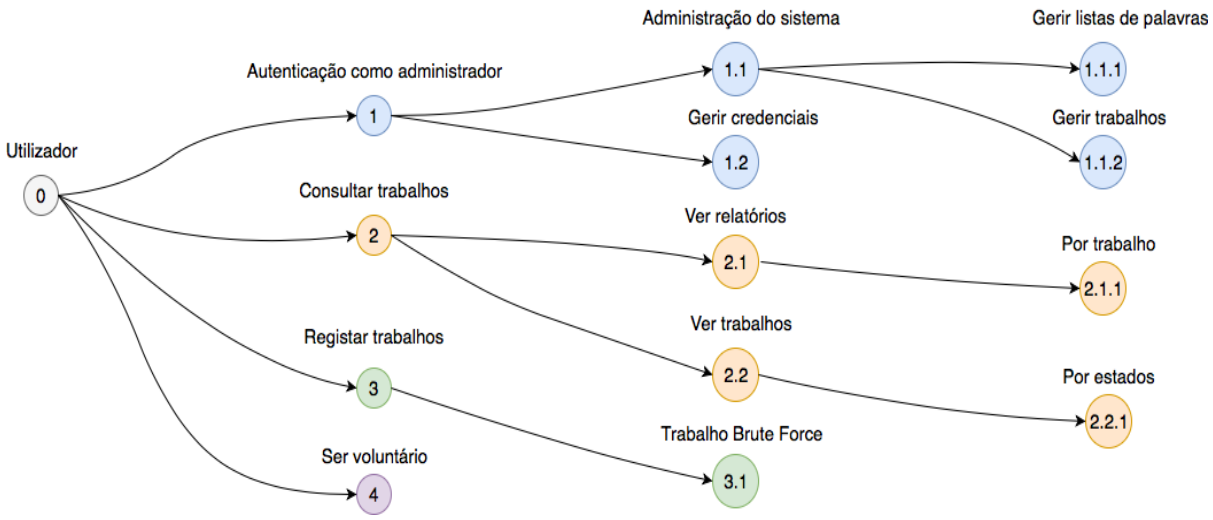


Figura 15: Mapa estrutural de funcionalidades

3.2 Arquitetura

Nesta secção vamos abordar a arquitetura da solução descrevendo as suas funcionalidades, tipos de utilizadores, diagramas e tecnologias utilizadas na concretização da solução.

3.2.1 Utilizadores da plataforma

Fazem parte desta plataforma dois tipos de utilizadores que se descrevem em seguida:

Utilizadores voluntários: Este tipo de utilizadores tem a capacidade de adicionar trabalhos e doar os seus recursos para realizar tarefas distribuídas. Estes utilizadores também podem consultar informação relacionada com os trabalhos no sistema.

Utilizador Administrador: Este utilizador tem as mesmas valências que o utilizador voluntário acrescido de um perfil de administração e tem como propósito administrar a plataforma, estes utilizadores podem adicionar informação de suporte como se descrever na secção

3.2.2 Estado dos trabalhos no sistema

A Tabela 6 descreve os estados possíveis que os trabalhos inseridos por utilizadores podem assumir.

#	**Nome do estado**	**Designação**
1	Ready	O trabalho encontra-se capacitado de ser processado por utilizadores voluntários
2	Not Found	A lista de palavras candidatas foi percorrida, contudo, nenhuma palavra satisfaz a palavra-passe definida no trabalho.
3	Success	A lista de palavras candidatas foi percorrida, e a palavra-passe definida no trabalho foi encontra com sucesso na lista.

Tabela 6: Estados possíveis dos trabalhos no sistema

3.2.3 Arquitetura de alto nível do sistema

A Figura 16 ilustra a arquitetura de alto nível da solução que demostra os diferentes componentes existentes e como eles se relacionam entre si. Os utilizadores finais interagem com o sistema através da "*WebApp*" e iniciam a sua interação introduzindo os parâmetros necessários para criarem trabalhos na plataforma - estes parâmetros devem incluir uma palavra-passe a verificar em conjunto com uma caracterização da verificação a realizar, que pode incluir a seleção do tipo de conjunto de dados bem como a dimensão do conjunto de dados. Os trabalhos resultantes da interação descrita anteriormente são enviados através de uma API REST que os persiste na base de dados. O "*Engine"* também tem como propósito organizar a distribuição das tarefas. Um trabalho é subdividido em tarefas que são resolvidas por uma lista de utilizadores voluntários que estejam ligados ao sistema. Assim, cada trabalho é resolvido em tarefas que são distribuídas pelos utilizadores finais que têm uma abordagem de voluntariado.

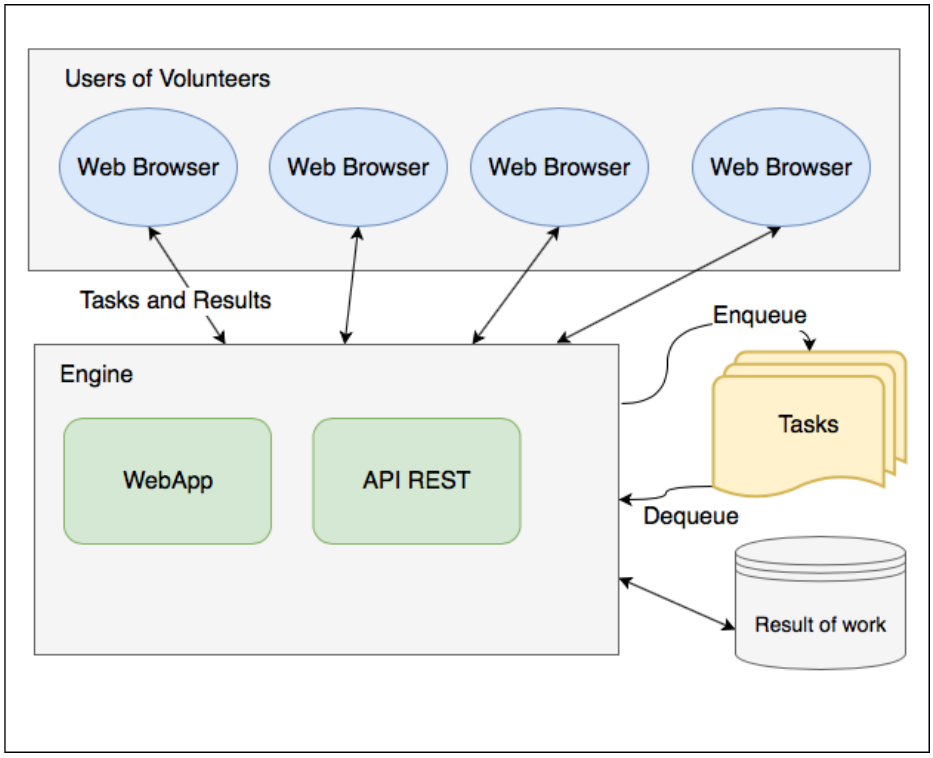


Figura 16 Arquitetura de alto nível do sistema DistPass

3.2.4 Diagrama de caso de uso (use-case)

A Figura 17 ilustra o caso de uso da plataforma. Neste diagrama apresentado, existem os dois tipos de utilizadores descritos na secção 3.2.1, quatro casos principais e cinto extensões dos casos principais. A Tabela 7, Tabela 8 e Tabela 9 descrevem os casos com maior detalhe.

Gestão de trabalhos	
Condições	Sem condições
Descrição	Este caso de uso envolve tudo o que está relacionado com os trabalhos a serem realizados pela plataforma. Aqui os utilizadores podem adicionar trabalhos que vão ser resolvidos em forma de tarefas distribuídas que são processadas por utilizadores voluntários.
Extensões	**Adicionar trabalhos**: Aqui podem ser adicionados novos trabalhos a serem submetidos na plataforma. Para que os trabalhos sejam considerados válidos, o utilizador deve indicar uma palavra-passe bem com um conjunto de dados que compõem a natureza da verificação. Consideramos aqui o tipo de conjunto de dados a utilizar bem como as características que vão compor a verificação, nomeadamente o

	segmento do conjunto de dados a utilizar. **Participar em trabalhos**: Esta extensão possibilita um utilizador participar na resolução de um trabalho contribuído com a resolução de tarefas.

Tabela 7: Descrição do caso de uso consultar trabalhos

Gerir perfil	
Condições	O utilizador é válido para realizar as ações neste caso
Descrição	Este caso permite atualização de credenciais de acesso, como se trata de uma área restrita a administração os administradores necessitam de uma conta de utilizador bem como uma palavra-passe para se autenticarem no sistema e poderem proceder à gestão do mesmo.
Extensões	**Alterar credenciais**: Permite aos utilizadores administradores alterarem as suas credenciais de acesso à plataforma.

Tabela 8: Descrição do caso de uso gerir perfil

Gerir aplicação	
Condições	O utilizador é válido para realizar as ações neste caso
Descrição	Este caso permite gerir a plataforma para que tenha uma evolução no sentido de se tornar mais eficiente com o melhoramento dos conjuntos de dados bem como a garantia que os trabalhos na plataforma são uteis para a comunidade de utilizadores.
Extensões	**Eliminar trabalhos:** Os administradores têm a capacidade de removerem os trabalhos que entenderem. **Gerir listas de palavras**: Esta extensão permite aos administradores enriquecerem o conjunto de dados.

Tabela 9: Descrição do caso de uso gerir aplicação

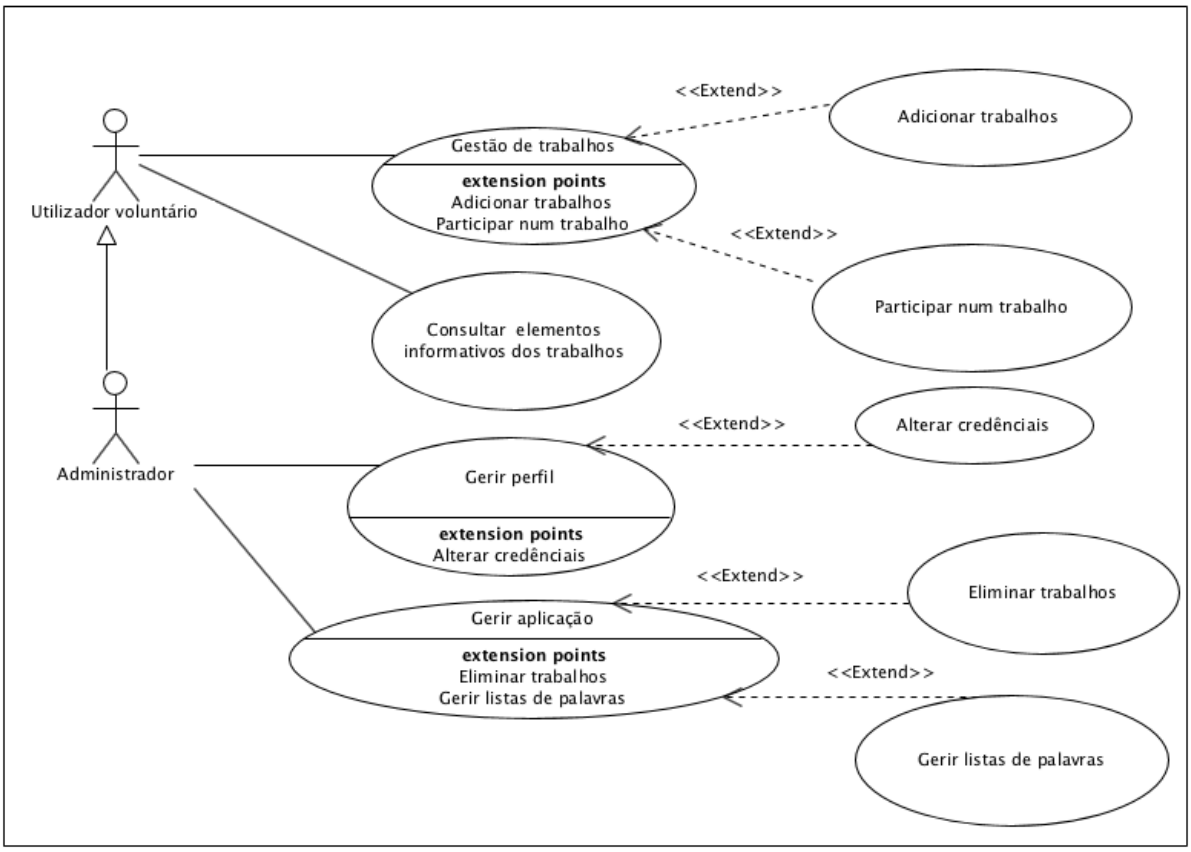


Figura 17: Diagrama de caso de uso

3.2.5 Diagramas de sequência

Nesta secção são apresentados dois diagramas de sequência, que representam duas das atividades mais importantes do sistema desenvolvido: a adição de um novo trabalho e a distribuição de tarefas pelos diversos utilizadores voluntários existentes.

3.2.5.1 Adição de uma nova tarefa

O diagrama de sequência ilustrado na Figura 18 representa a inserção de um novo trabalho. Um utilizador referido como “Utilizador voluntário” inicia a sequência enviando os dados pertencentes a um novo trabalho à API REST. Estes dados, por sua vez, são persistidos na base de dados. Consequentemente, o registo é realizado e inicia-se uma sequência inversa com o propósito de informar o utilizador do registo com sucesso do trabalho.

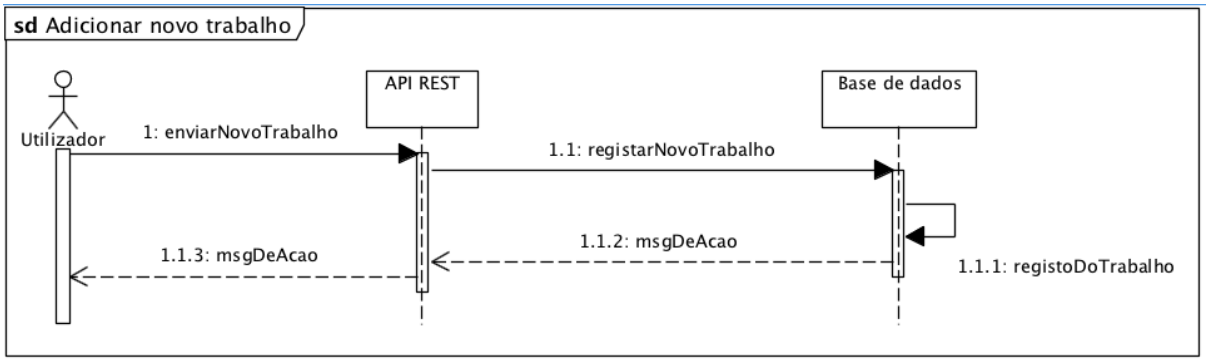


Figura 18: Diagrama de sequência de adicionar novo trabalho

Sequência de mensagens:

(1) enviarNovoTrabalho: esta mensagem é enviada com o conteúdo dos dados para a API REST com o propósito de ser persistidos à base de dados.

(1.1) registoNovoTrabalho: aqui a mensagem é efetivamente enviada para a base de dados para ser inserido o trabalho na base de dados.

(1.1.1) registoDoTrabalho: a mensagem aqui presente é enviada ao próprio objeto com o propósito de garantir o sucesso do registo do trabalho na base de dados

(1.1.2) msgDeAcao mensagem de retorno que tem como objetivo fazer chegar à API REST o resultado do registo do trabalho. Este resultado pode ser sucesso ou insucesso.

(1.1.3) msgDeAcao: trata-se da mesma mensagem do ponto **1.1.2**, contudo, tem como destinatário o utilizador.

3.2.5.2 Distribuição de tarefas dos trabalhos

A Figura 19 ilustra o diagrama de sequência de processar um trabalho, esta sequência pretende descrever paulatinamente as etapas inerentes ao processamento de um trabalho.

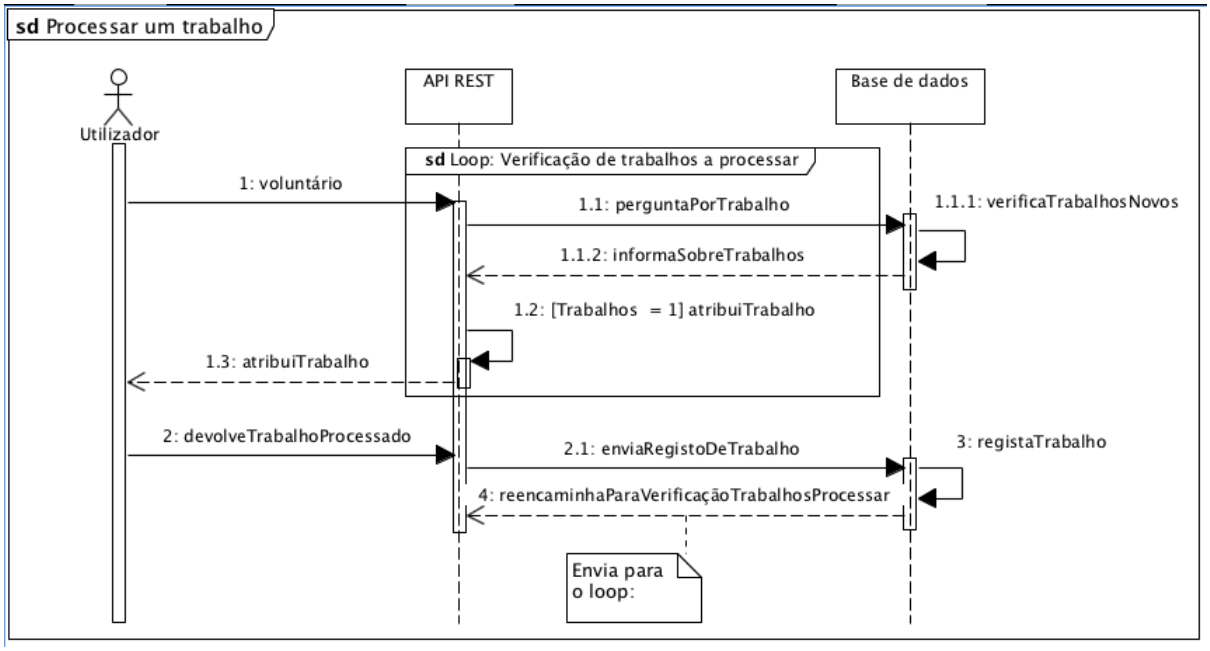


Figura 19: Diagrama de sequência de distribuição de trabalhos

Sequência de mensagens:

(1) voluntário: esta mensagem representa o utilizador final a notificar que se encontra disponível para ser voluntário e assim contribuir para a resolução de um trabalho resolvendo tarefas.

(1.1) perguntaPorTrabalho: esta mensagem encontra-se dentro de um ciclo interativo de verificação de trabalhos a processar. Ou seja, trabalhos que ainda não estão concluídos e que podem ser resolvidos através de tarefas.

(1.1.1) verificaTrabalhosNovos: esta mensagem tem como propósito consultar a existência de trabalhos novos.

(1.1.2) informaSobreTrabalhos: é uma mensagem de retorno que informa a API REST da existência de novos trabalhos.

(1.2) atribuiTrabalho: esta mensagens têm uma condição que caso seja satisfeita segue para a sequência 1.3, caso contrário o ciclo iterativo é continuado.

(1.3) atribuiTrabalho: Trata-se da mensagem de retorno da API REST para o cliente a informar que vai-lhe ser atribuída uma tarefa que pode ser computada.

(2) devolveTrabalhoProcesso: esta mensagem é enviada do utilizador para a API REST com os dados do trabalho processado nomeadamente o sucesso da verificação da palavra-passe entre outra informação relacionada com o processamento da tarefa.

(2.1) enviaRegistoTrabalho: esta mensagem é enviada entre a API REST e a base de dados. É aqui que os dados do processamento da tarefa que o utilizador voluntário realizou são registados na base de dados.

(3) registaTrabalho: é uma mensagem enviada para o próprio objeto da base de dados e tem como objetivo informar o sucesso da operação.

(4) reencaminhaParaVerificaçãoTrabalhosProcessar: esta mensagem de retorno tem como propósito reencaminhar o utilizador para o ciclo de atribuição de trabalhos a processar. O utilizador entrará nesse ciclo para encontrar um trabalho que possa ser processado e lhe seja atribuída uma tarefa de um determinado trabalho.

3.2.5.3 Verificação de tarefas a realizar

A Figura 20 ilustra o diagrama da verificação de tarefas a realizar no sistema desenvolvido. Este diagrama pretende descrever a forma como as tarefas são atribuídas aos utilizadores. De seguida descrevemos cada sequência com maior detalhe.

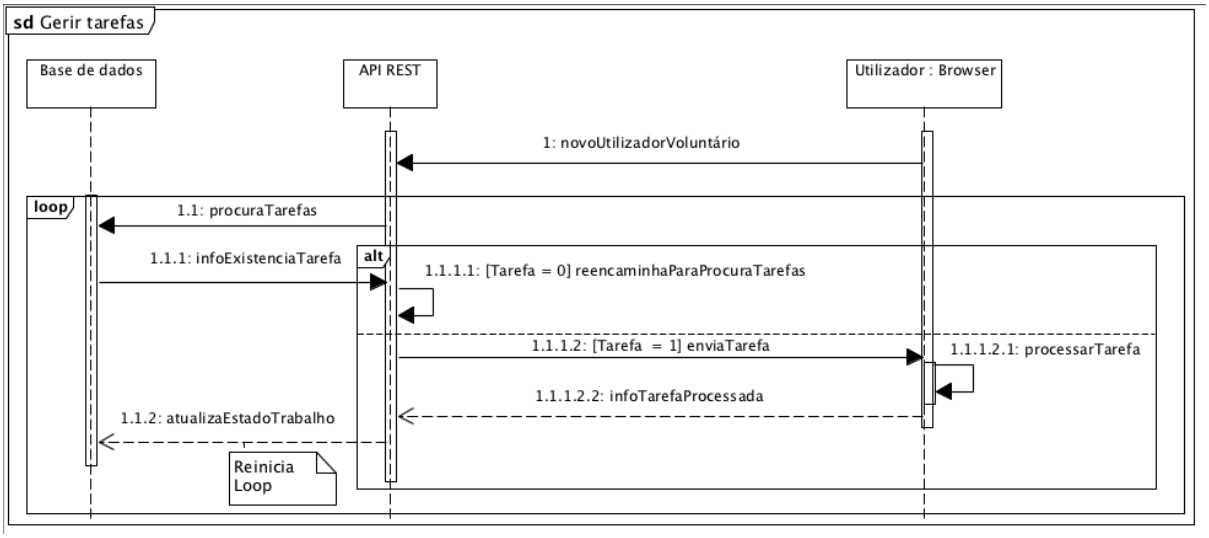


Figura 20: Gestão de tarefas do sistema DistPass

(1) novoUtilizadorVoluntario: esta mensagem ocorre na ação de um utilizador dar-se como voluntario, aqui é estabelecida uma ligação *websocket* entre o servidor e o utilizador. Esta ligação é importante na medida que vai permitir identificar univocamente um utilizador no sistema. Esta identificação é necessária para gerir as palavras do conjunto de dados que estão em análise. Na eventualidade da ligação ser quebrada, todas as palavras que estavam a ser analisadas por este utilizador, são libertadas com vista a serem novamente distribuídas.

(1.1) procurarTarefas: esta mensagem trata-se de despoletar a verificação da existência de trabalhos por concluir. A conclusão de um trabalho é dada quando atinge um estado de "Success" ou "Not found", caso um trabalho não tenha um destes estados, vão existir tarefas ainda por realizar.

(1.1.1) infoExistenciaTarefas: trata-se de uma mensagem que contem a existência de tarefas por realizar, esta mensagem vai entrar num bloco de condição, se existir um trabalho por concluir esta mensagem tem uma lista de palavras passiveis de serem verificadas em conjunto com o tipo de *hash* e a palavra-passe submetida a teste. Caso não exista nenhum trabalho passível de ser concluído, o utilizador é reencaminhado para a "procuraTarefas".

(1.1.1.1) [Tarefa = 0] reencaminhaParaProciraTarefas: como não existem trabalhos a serem processados, e consequentemente não existe tarefas, o *loop* do diagrama é novamente inicializado.

(1.1.1.2) [Tarefa = 1] enviaTarefa: esta mensagem significa que existe uma tarefa que o utilizador voluntario pode realizar. Para isso, a mensagem segue com uma lista de palavras que fazem parte da tarefa e que vão ser analisadas no *browser* do utilizador. Para alem desta lista, segue a palavra-passe que foi passada, e o tipo de algoritmo de *hash* que se pretende analisar.

(1.1.1.2.1) processarTarefa: esta mensagem é envia ao próprio objeto e tem como propósito realizar a verificação da palavra-passe.

(1.1.1.2.2) infoTarefaProcessada: esta mensagem é enviada à API REST e tem como propósito informar o resultado da verificação.

(1.2.2) atualizaEstadoTrabalho: esta mensagem pretende atualizar o estado do trabalho tendo em conta as palavras do conjunto de dados já analisadas e o próprio estado de conclusão do trabalho.

3.3 Tecnologias utilizadas

Como o objetivo desta dissertação é desenvolver um sistema para a WWW. É de todo o interesse recorrer a tecnologias que permitam um rápido desenvolvimento de ambientes gráficos, e que disponibilizem recursos padronizados que possam oferecem componentes previamente concebidos que facilitam e potenciam o desenvolvimento, quer na perspetiva de tempo de desenvolvimento quer na capacidade de abstração e reutilização de recursos.
É neste contexto que a opção da utilização de uma tecnologia *Javascript*, como o *Angular*. Esta *framework* Javascript permite de forma simples e controlada, aceder a recursos próprios e de outras *frameworks* como o *Bootstrap* (ver secção 3.3.2) que potenciam e agilizam de forma pragmática a construção do sistema (Fat, Vujovic, Papp, & Novak, 2016);(Manitz, Harbering, Schmidt, Kneib, & Schöbel, 2017).
Do lado do servidor, optou-se pela utilização do *Node.js* (ver secção 3.3.3) com os recursos próprios de serviços HTTP, como componentes extensíveis como o *Express* (ver secção) e *Mongoose* (ver secção 3.3.7) para estabelecer ligação à base de dados suportada em *MongoDB* (Bangare, Gupta, Dalal, & Inamdar, 2016).
Descrevemos de seguida de forma mais detalhada cada biblioteca utilizada.

3.3.1 Angular

O ***Angular***[22] é uma *framework* com licença MIT, de código aberto que é mantida pela *Google*. É baseada em *TypeScript*. A construção de aplicações assenta sobretudo em blocos *NgModules* que fornecem um contexto de compilação para componentes. Uma aplicação escrita em Angular tem sempre um módulo de raiz que permite a auto iniciar-se e normalmente tem outros módulos de recursos. Existem conjuntos de objetos gráficos que o Angular pode modificar de acordo com a lógica e comportamento da aplicação. Qualquer aplicação escrita em Angular tem pelo menos uma componente raiz. Os componentes usam serviços, que fornecem funcionalidades específicas e podem ou não ser diretamente relacionada à manipulação de objetos gráficos. Os provedores de serviços podem ser

[22] https://angular.io

injetados em componentes como dependências, criando assim modularidade, reutilizável e eficiente (Fat et al., 2016).

3.3.2 Bootstrap

O *Bootstrap*[23] é uma *framework* de *frontend* criada por Mark Otto e Jacob Thornton. Possui uma licença MIT e é igualmente disponibilizado em formato de código aberto. O propósito desta *framework* é permitir o desenvolvimento de componentes de *frontend* e de interação com o utilizador final. É destinada ao desenvolvimento de aplicações *Web* e utiliza HTML, CSS e *Javascript*. Esta *framework* é baseada em modelos de estruturas padrão como *Navbar* ou *grid*. Este conceito proporciona uma experiência agradável ao utilizador final por ser bastante responsivo e adaptável a diversos *browsers Web* (Jain, 2014).

3.3.3 Node.js

O *Node.js*[24] foi inicialmente desenvolvido por Ryan Dahl. Está igualmente abrangido por uma licença de código aberto MIT. Trata-se de uma plataforma baseada em eventos que correm em tempo de execução e que recorre ao motor *Javascript* V8 desenvolvido pela *Google* para executar *Javascript* fora do *browser Web*. Como o tempo de execução em *Javascript* é orientado a eventos assíncronos, o *Node.js* foi projetado para criar aplicações de rede escaláveis para poder dar resposta a muitas conexões em simultâneo. Em cada conexão, o retorno da chamada é acionado, mas se não houver trabalho a ser realizado, o *Node.js* entrará em suspensão. Este modelo contrasta com modelos mais comuns, em que o modelo *multithread* de desenvolvimento de aplicações de rede pode ser relativamente ineficiente e muito difícil de usar. O *Node.js* proporciona uma eficiência relativa ao problema de processos ficarem parados. Isso deve-se a quase nenhuma função do *Node.js* executar diretamente operações de *I/O* em modo síncrono, portanto, o processo nunca é bloqueado. Podemos considerar o ciclo de eventos do Node.js como sendo uma fila FIFO (*first in, first out*) em que cada elemento detém uma cadeia de instruções que normalmente são definidas dentro de uma função que devem ser executadas pela aplicação. Por conseguinte, quando ocorre um evento é adicionado uma nova entrada ao final dessa fila (Tilkov & Vinoski, 2010).

3.3.4 Gulp

[23] https://getbootstrap.com
[24] https://nodejs.org/en/

O *Gulp*[25] é um conjunto de ferramentas *Javascript* de código aberto, desenvolvido pela Fractal Innovations. Está coberto por uma licença MIT. O *Gulp* é um executor de tarefas, construído com base no *Node.js* e no NPM (*Node.js Package Manager*), é usado para automação de tarefas inerentes ao desenvolvimento de aplicações para a *Web* que podem ser demoradas e repetitivas. O conceito de operabilidade do *Gulp* é ser baseado em fluxos de trabalho que facilitam a operação de ficheiros por meio de *pipelines*. O *Gulp* lê o sistema de ficheiros e canaliza os dados por meio do operador *pipe*. Os arquivos originais não são afetados até que todos os *plug-ins* sejam processados. Pode ser configurado para modificar os ficheiros originais ou para criar novos. Entre outras, tem as valências da capacitação de criar minimização, concatenação de código fonte, otimização e execução de testes.

3.3.5 Lodash

O *Lodash*[26] é uma *framework Javascript* de código aberto com licença MIT e foi desenvolvido pelo John David Dalton. Esta *framework* fornece funções úteis para tarefas comuns de programação como por exemplo verificar a nulidade de uma variável ou manipulação de matrizes e é usando no paradigma de programação funcional. O *Lodash* proporciona assim uma forma mais simples, fácil de compreender e de manter código *Javascript*.

3.3.6 Express

O *Express*[27] é uma *framework* de suporte à construção de aplicações *Web*. É uma estrutura de aplicações para *Web Node.js,* tem um propósito de ser minimalista e flexível e fornece um conjunto robusto de recursos para aplicações da *Web*. Esses recursos vão desde rotear URIs, utilização e criação de *middleware*, utilização de mecanismos modelo que permitem manipulação de ficheiros estáticos, tratamento de erros de solicitação de pedidos ao servidor HTTP e depuração do comportamento da aplicação *Web*.

[25] https://gulpjs.com
[26] https://lodash.com
[27] https://expressjs.com

3.3.7 Mongoose

O *Mongoose*[28] é uma ferramenta de modelagem de objetos para o *MongoDB*, foi desenvolvida como propósito de funcionar num ambiente assíncrono. Possui uma licença de código aberto do MIT e resulta do trabalho de Valeri Karpov. O Mongoose está organizado em duas áreas que providenciam funcionalidades que podem ser consumidas pelo *Node.js* em ligação ao *Mongodb*. Esta ferramenta pode ser útil na definição de esquemas para as bases de dados, estabelecer conexões, definir modelos que são construtores compilados de esquemas e manipulação de documentos e subdocumentos persistidos na base de dados.

3.3.8 node-restful

O *node-restful*[29] disponibiliza um conjunto de recursos do *MongoDB* e rotas REST padrão que são construídas automaticamente. O *node-restful* tem uma licença de código aberto do MIT e é mantido por Ben Augarten. O *node-restful* interage com o *Mongoose* e pode obter informação dos esquemas definidos para a base de dados. A API do *node-restful* manuseia os verbos do protocolo HTTP e disponibiliza métodos pré-definidos como *GET*, *POST*, *PUT* e *DELETE*. Também é possível executar rotas personalizadas, para isso é invocado o método *route* que possibilita a criação de um *middleware* desenvolvido à medida.

3.3.9 PM2

O *PM2*[30] é um gestor de processos de tempo de execução de aplicações *Node.js*. Resultado do trabalho de Alexandre Strzelewicz, está disponível em código aberto e possui uma licença AGPLv3. Funciona como um balanceador de carga integrado e permite manter as aplicações *Node.js* ativas por um tempo indeterminado. O *PM2* oferece uma maneira simples de monitorar o uso de recursos utilizados pelas aplicações. É possível monitorizar a memória e a utilização do CPU facilmente e diretamente no terminal. O *PM2* pode ser expandido através do *Keymetrics* que oferece um conjunto mais alargado de métricas e formas de visualização como *dashboard* e outros artefactos gráficos e textuais de visualização de informação.

[28] https://mongoosejs.com
[29] https://github.com/baugarten/node-restful
[30] http://pm2.keymetrics.io

3.3.10 Socket.IO

O *Socket.IO*[31] é uma biblioteca em Javascript para aplicações *Web* em tempo real. É o resultado do trabalho de Guillermo Rauch e atualmente está abrangida por uma licença de código aberto do MIT. Esta biblioteca usa principalmente o protocolo *WebSocket* para fazer operações de I/O em modo síncrono. O Socket.IO permite comunicação bidirecional em tempo real, entre o cliente e servidor o que permite uma adaptabilidade em relação à tecnologia do cliente, na eventualidade do cliente não ter um browser que suporte *WebSocket*, o *Socket.IO* adapta a sua usabilidade recorrendo a outras tecnologias e protocolos como *Long Polling* ou AJAX**.**

[31] https://socket.io

Capítulo 4 Implementação da solução

Este capítulo tem como objetivo descrever a implementação da solução, apresentando uma visão geral do funcionamento da solução.

4.1 Visão geral do funcionamento da solução

O sistema desenvolvido é denominado DistPass, e incorpora uma solução funcionar de portal *Web*, onde os utilizadores voluntários podem contribuir na resolução dos trabalhos bem como adicionar, consultar e visualizar relatórios. Os utilizadores administradores podem realizar tarefas de manutenção da solução como gerir listas de palavras-passe.

4.2 Esquemas da base de dados

A base de dados que foi utilizada para esta solução foi o *MongoDB*, trata-se de uma vertente do *NoSQL* que é baseada em documentos. Serve para o nosso propósito, porque precisamos de um sistema rápido que possa dar resposta à leitura e escrita da verificação das palavras-passe, e que por outro lado é altamente escalável. A Figura 21 mostra a estrutura da base de dados *NoSQL* com as suas coleções e índices aplicados nesta solução. A coleção denominada "*dictionaries*" que serve para armazenar as listas de palavras possui dois índices (*indexes*). O índice "_id_" é criado por defeito pelo *MongoDB*, tem como propósito dar mais eficiência nas pesquisas dos documentos. Quanto ao índice "*len*", foi criado com o mesmo propósito de dar mais eficiência à coleção, através da ordenação dos documentos por tamanho. Quando à coleção "*dispasses*" possui o índice por defeito *"_id_"* e foi adicionado outro *index* que organiza os documentos por nome. Por fim, a coleção "*useradmin*" possui apenas o *index "_id_".*

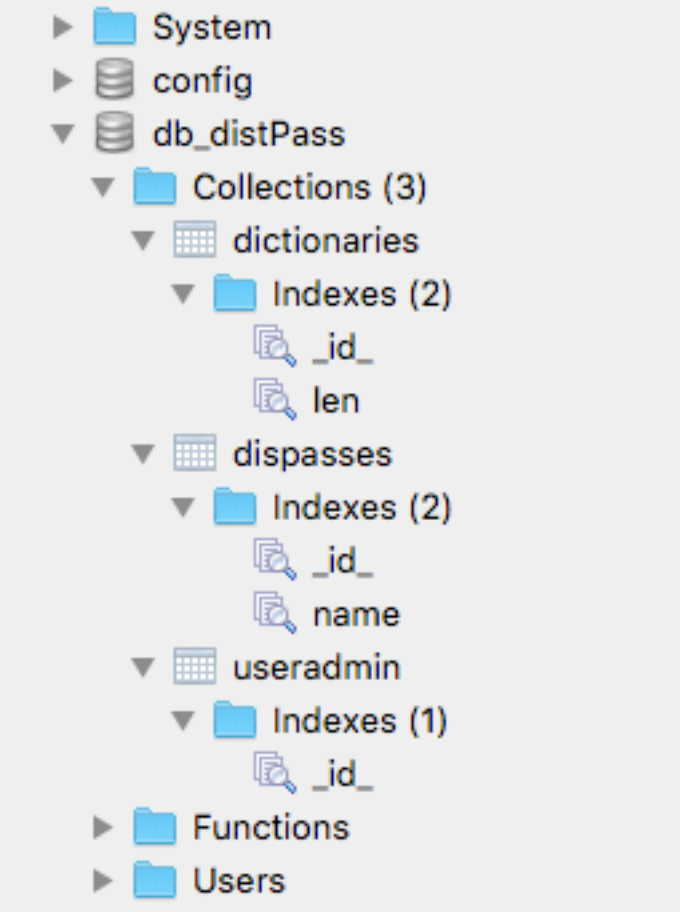


Figura 21: Estrutura da base de dados da solução

A Figura 22 ilustra um documento da coleção "*dictionaries*". Esta coleção possui os documentos que agrupam as palavras do conjunto de dados (ver Anexo I). Cada documento possui até 50 mil palavras, consoante o tamanho da combinação de caracteres. Este número foi encontrado por uma distribuição equitativa, entre as tarefas e a capacidade de processamento do servidor que serve o sistema desenvolvido.

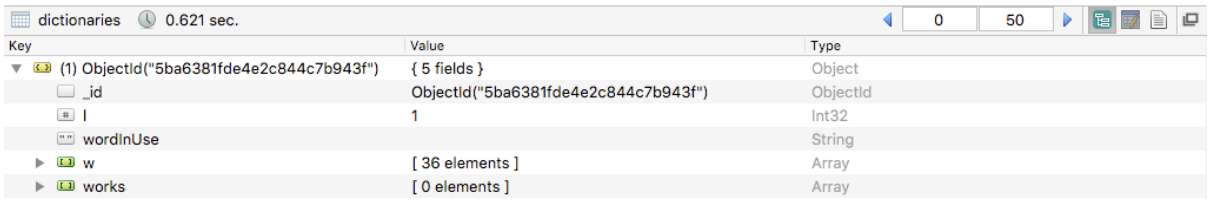


Figura 22: Documentos da coleção "*dictionaries*"

A Figura 23 ilustra um documento que persiste o trabalho "*Work 1*", este trabalho utilizou o SHA 256 e decorreu em 125 segundos.

dispasses 0.008 sec. 0 50

Key	Value	Type
(1) ObjectId("5ba78ca90498d802f783fcb1")	{ 15 fields }	Object
_id	ObjectId("5ba78ca90498d802f783fcb1")	ObjectId
name	WORK 1	String
typePassword	SHA256	String
lengthMin	1	Int32
lengthMax	5	Int32
passwordProvided	676cb75018edccf10fce6f376f2124e02c3293fa3fe8...	String
created	2018-09-23 12:52:57.881Z	Date
typeAttack	Brute Force	String
jobsCount	1181	Int32
status	Success	String
totalWordVerified	43907072	Int32
startWork	1537707197	Int32
charset	alphabet-numeric	String
__v	0	Int32
crlTimeEndWork	1537707322	Int32

Figura 23: Documento do trabalho “Work 1”.

A Figura 24 mostra um documento correspondente a um registo de um administrador do sistema.

useradmin 0.001 sec. 0 50

Key	Value	Type
(1) ObjectId("5b3009e43846ec41978fcd39")	{ 5 fields }	Object
_id	ObjectId("5b3009e43846ec41978fcd39")	ObjectId
_userA	lclms	String
_pwdA	50b17101b90c0b438e930bca7c17a8f5d19f5450	String
_mail	lclms@iscte-iul.pt	String
valid	29991231	String

Figura 24: Documento de um registo de um administrador da plataforma

4.3 Funcionamento aplicacional

Nesta secção será descrito o funcionamento do sistema desenvolvido. É feito uma explicação textual que acompanhada com elementos gráficos por forma a facilitar a interpretação da usabilidade do sistema.

4.3.1 Página inicial

A página inicial é o primeiro local que o utilizador visualiza quando acede ao sistema. Nesta página, o utilizador pode consultar os trabalhos que estão ativos na plataforma, como ceder o seu hardware para contribuir para a resolução de trabalhos pendentes. A Figura 25 ilustra o ambiente inicial do sistema, na primeira página é apresentado ao utilizador a possibilidade de doar o seu hardware, para isso estão ao dispor dois botões “**Contribute**” e “**Stop Contribute**” que permitem ao utilizador ceder ou parar a cedência dos seus recursos computacionais.

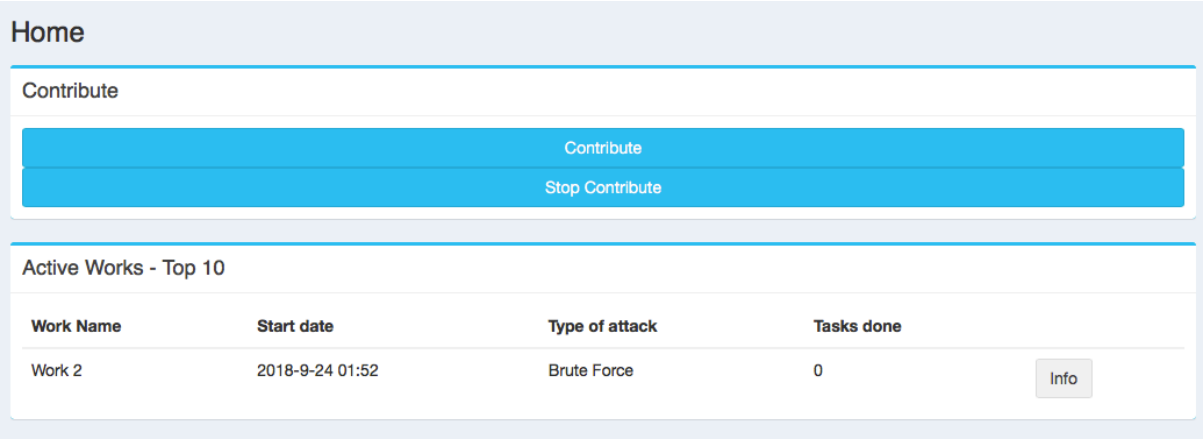


Figura 25: Página inicial do sistema desenvolvido.

4.3.2 Menu da Aplicação

A Figura 26 ilustra o menu lateral do sistema. Este menu acompanha sempre a aplicação, permitindo que o utilizador possa navegar entre as diversas páginas de uma forma simples e cómoda. A opção "***Home***" do menu é a página inicial e é aqui que o utilizador é recebido quando acede à solução. A opção "***Verifications Works***" permite consultar e adicionar trabalhos. Nesta página estão igualmente os separadores que ilustram os dados estatísticos associados a cada trabalho. A opção "***About***" descreve o propósito do trabalho e a sua motivação. Por último, a opção "***BackOffice***" permite ao administrador aceder à gestão do sistema.

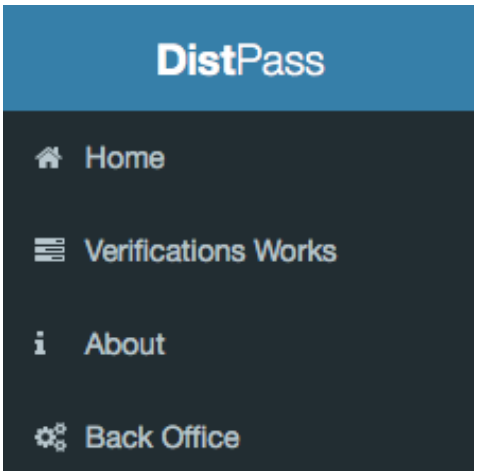


Figura 26: Hierarquia de menus

4.3.3 Administração da plataforma

A administração do sistema é feita numa página própria e pode ser acedida através da opção do menu "**BackOffice**". Para aceder a esta funcionalidade, o utilizador administrador necessita de se autenticar conforme a Figura 27. Consequentemente após este passo o utilizador é remetido para o "**BackOffice**". A Figura 28 ilustra os formulários da gestão da solução que podem ser visualizados na mesma.

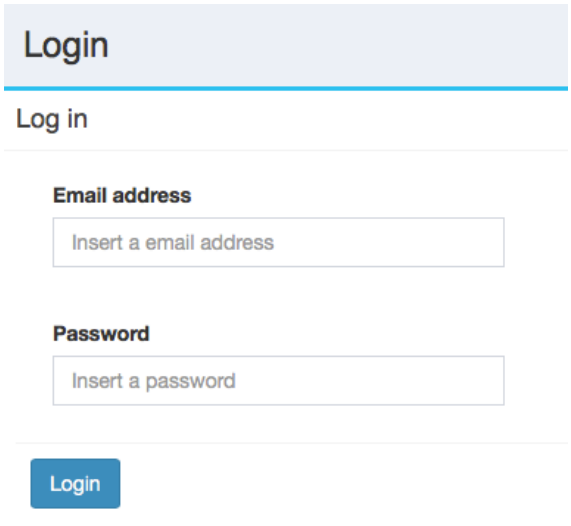


Figura 27: Autenticação como administrador no sistema DistPass

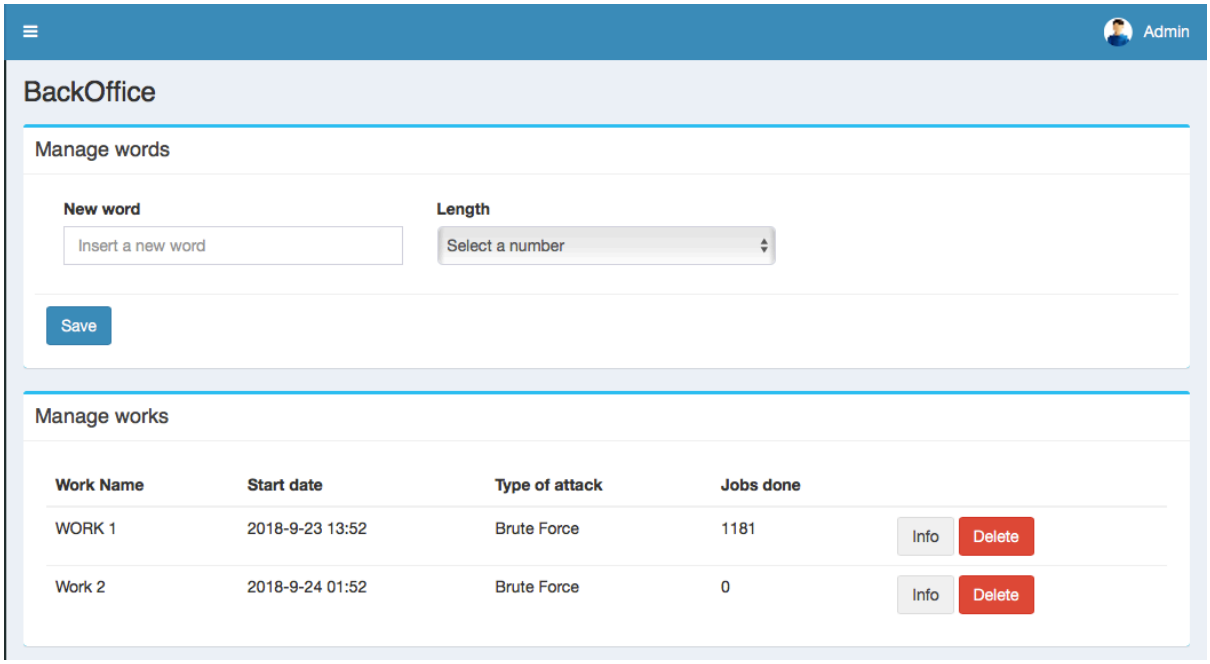


Figura 28: Gestão da plataforma, menu de "BackOffice".

4.3.4 Modo de funcionamento

A Figura 29 ilustra a página "***Verifications Works***". Esta é uma das páginas mais importante do sistema, onde os utilizadores podem consultar os trabalhos existentes e adicionar novos trabalhos a serem processados por utilizadores voluntários. Os trabalhos existentes podem ser visualizados numa tabela "***WorkList***" que apresenta cada um dos trabalhos e as informações mais importantes, como o nome do trabalho, a data da criação, o número de tarefas e o estado do trabalho. Os utilizadores têm a possibilidade de consultar mais detalhes, carregando num botão próprio denominado "***Info***". Este botão conduz o utilizador a um separador próprio que descreve com mais detalhe a informação dos trabalhos, a Figura 30 ilustra a essa mesma informação. Por fim, a Figura 31 ilustra um separador com um gráfico representativo da relação das palavras analisas por tarefa.

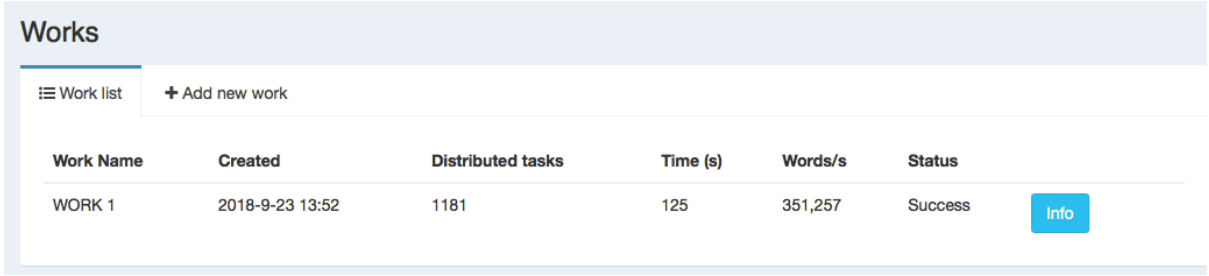


Figura 29: Tabela e trabalho "***Work list***" na página "***Verifications Works***".

Work: WORK 1

Created: 2018-9-23 13:52

Attack: Brute Force

Tasks: 1181

Password: SHA256

Password: 676cb75018edccf10fce6f376f2124e02c3293fa3fe8f953c75386198c714514

Length Min: 1

Length Max: 5

Status: Success

Word Verified: 43907072

Word time (s): 125

Charset: alphabet-numeric

Suggestion: Password too short. Increase the number of characters to make the password more complex. Use a password consisting of a phrase with multiple words.

Figura 30: Dados de um trabalho no sistema.

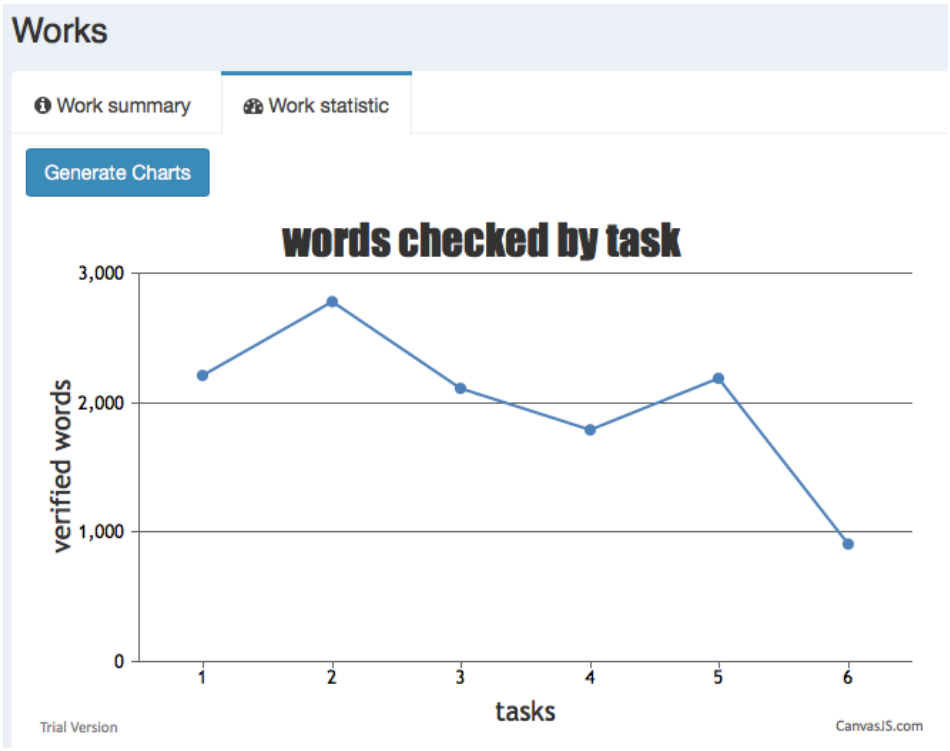


Figura 31: Gráfico representativo da relação das palavras analisas por tarefa

O separador "***Add new work***" conduz o utilizador a um novo menu. Aqui, o utilizador pode selecionar a modalidade "***Brute Force***" para submeter uma palavra-passe a ser analisada. A Figura 32 ilustra o separador "***Add new work***".

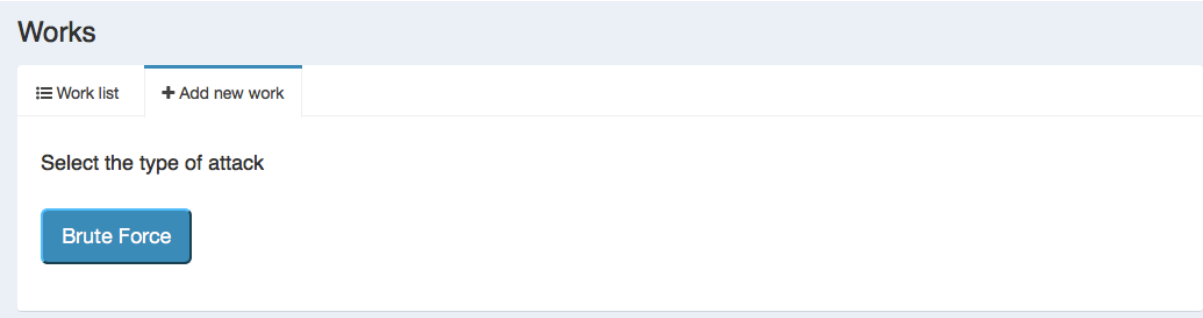


Figura 32: Separador "**Add new work**"

4.3.4.1 Ataque usando força bruta

Esta modalidade recorre ao conjunto de dados, descrito no Anexo I. O utilizador preenche o formulário ilustrado na Figura 33 onde deve selecionar um nome para o trabalho, o tipo de *hash* a ser verificado, o *charset* que define o *keyspace* de acordo o tipo de caracteres (numéricos, alfanuméricos ou ambos) e um tamanho mínimo e máximo a considerar. Por fim, o utilizador deve inserir a palavra-passe a ser verificada.

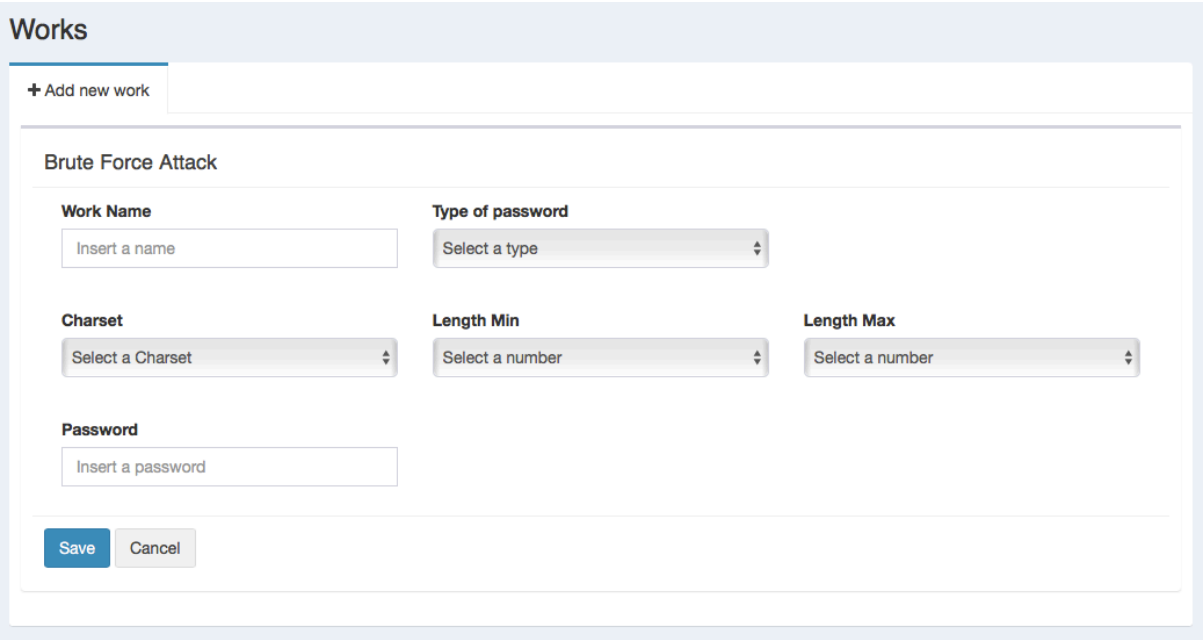


Figura 33: Formulário de um novo trabalho

Capítulo 5 Resultados e validação

Neste capítulo pretende-se introduzir o conjunto de dados utilizados na verificação do sistema, abordar os testes com os utilizadores e comparar os resultados obtidos com outras ferramentas.

5.1 Conjunto de dados utilizado

Para a criação deste conjunto de dados, foi utilizada a ferramenta **Crunch**, que é uma ferramenta que acompanha a distribuição do Kali Linux e está abrangida por uma licença GPLv2. A ferramenta tem uma conceptualização genérica, o que possibilita a criação de listas de palavras customizadas e adaptadas às necessidades dos utilizadores. Veja-se o caso demostrado na Figura 34 que ilustra a criação do conjunto de dados utilizado nesta dissertação. A configuração é simples, recorre ao comando "`crunch <min> <max> [options]`" podemos obter uma lista de palavras à medida. Para mais detalhes sobre este conjunto de dados podem ser encontrados no Anexo I.

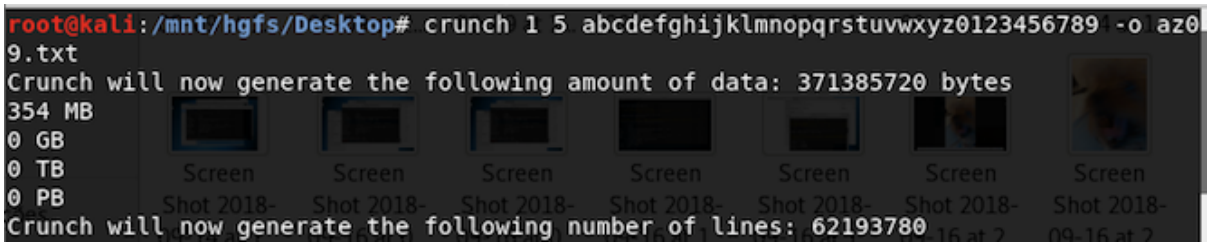


Figura 34: Criação do conjunto de dados através da ferramenta "`crunch`".

5.2 Testes com utilizadores

O sistema desenvolvido foi instalado num servidor Intel Core i7 Quad-Core 2,80 GHz com 16 GB de memória de acesso aleatório dinâmica síncrono. Para aferir o sistema desenvolvido, foram realizados três testes para cada algoritmo de hash. Cada teste teve um número de nós participantes diferente, o primeiro teste foi realizado com 4 nós, o segundo com 6 nós, o terceiro com 10 nós e o último com 15 nós ativos. A Figura 35 ilustra os testes dos nós ativos 10 e 15, o grafismo da imagem permite perceber a relação de tempo e as palavras analisadas

por segundo. O aumento de nós potência a velocidade de verificar palavras o que proporciona um efeito da redução do tempo de verificação.

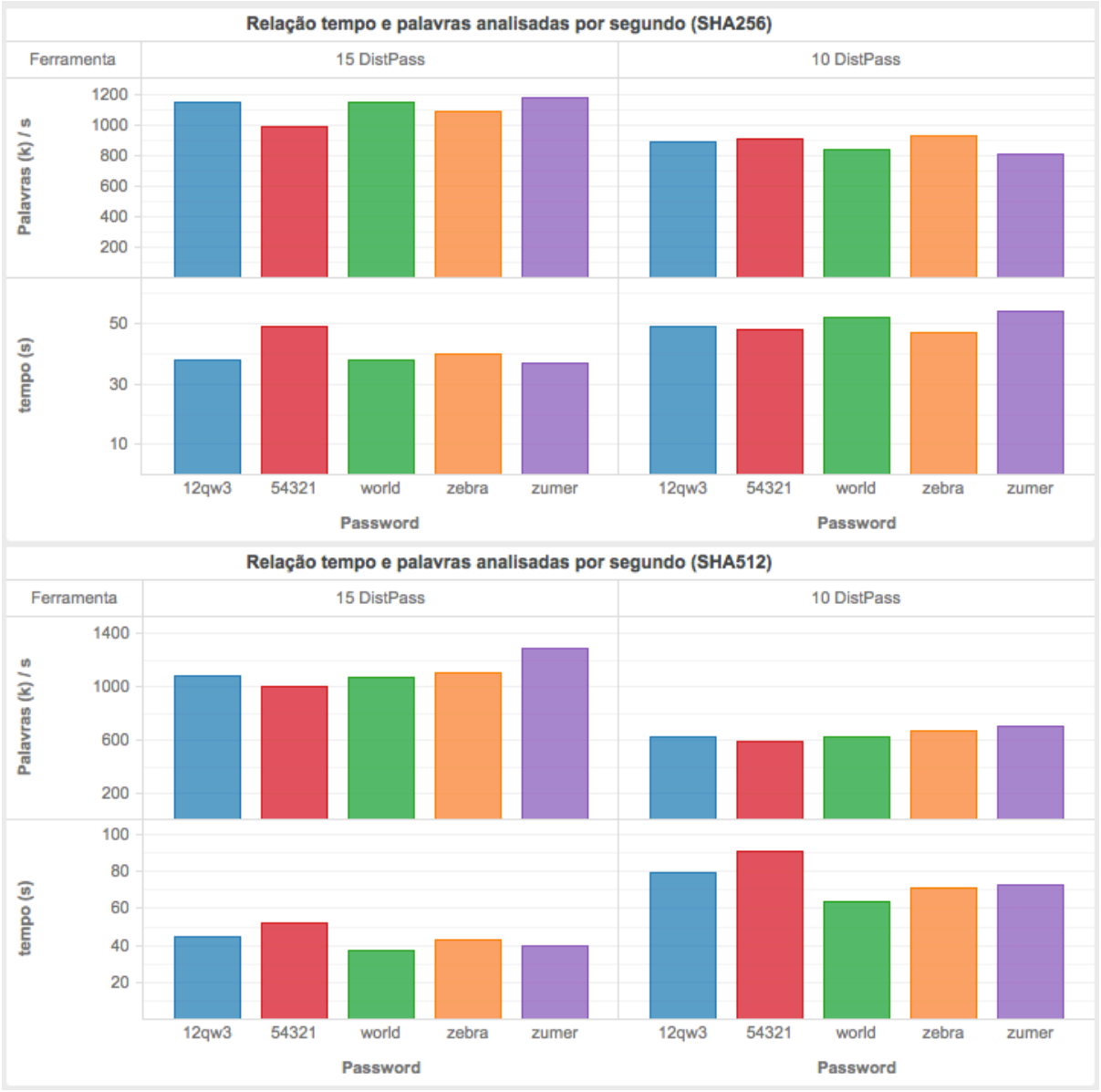


Figura 35: Resumo dos testes realizados no sistema DistPass

5.3 Comparação com outras ferramentas

Nesta secção vamos descrever o propósito de cada ferramenta, as suas valências e a forma como operam. Para efeitos de teste foram verificadas as palavras que estão ilustradas no Anexo I.

5.3.1 John the Ripper

O *John the Ripper*[32] (JtR) é um software de quebra de palavras-passe, é mantido pela Openwall e é de código aberto. Está abrangido por uma licença GPLv2. Este software abrange diversas áreas de verificação da robustez das palavras-passe, funciona de forma *offline* e permite combinar diversas funcionalidades e técnicas como extrair resumos (*hashes*) de sistemas operativos, ataques a palavras-passe através de força bruta e de listas previamente elaboradas. Também disponibiliza funcionalidades de ajuda à verificação de palavras-passe como remover palavras duplicadas em listas e utilização de algoritmos baseados em cadeias de Marcov (Weir et al., 2009).
Para isso disponibiliza um conjunto alargado de formatos e sub-formatos que são admitidos. A Figura 36 ilustra esses mesmos formatos. O *John the Ripper* guarda as palavras-passe já testadas num ficheiro chamado john.pot, o que permite relacionar trabalhos e evitar consumir recursos quando a palavra-passe a verificar já foi testada (Duermuth et al., 2015).

[32] http://www.openwall.com/john/

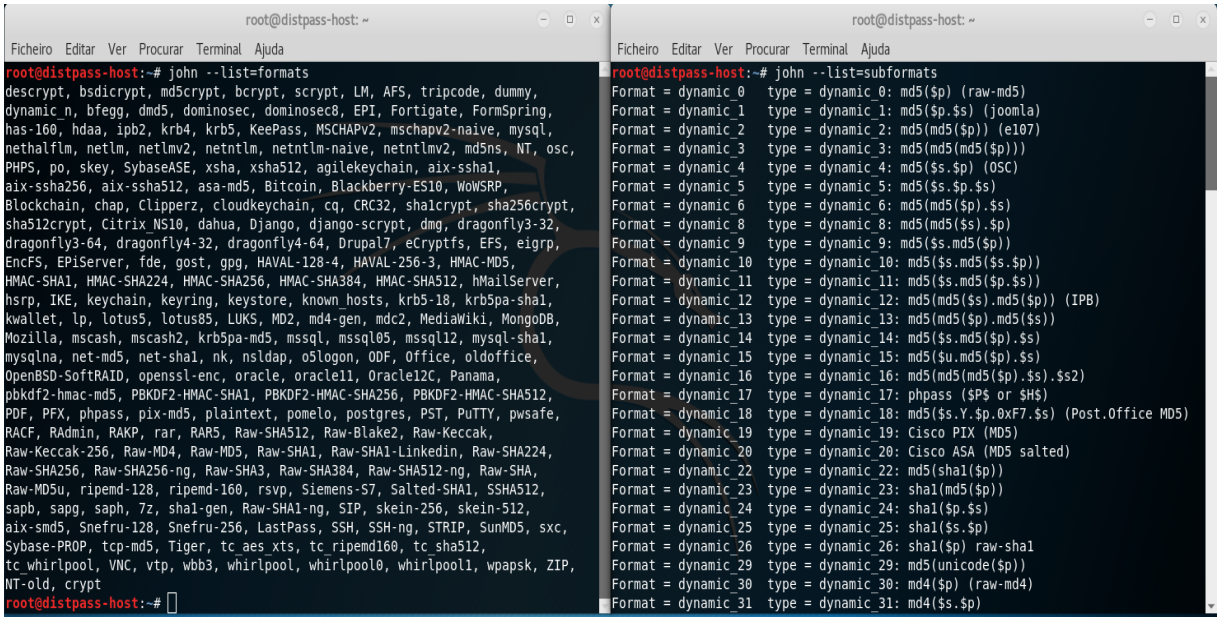


Figura 36: Formatos que são admitidos na ferramenta JtR

A Figura 37 ilustra a verificação da palavra "zebra" que pertence ao conjunto de dados de força bruta, este conjunto de dados podem ser consultado com maior detalhe no Anexo I. Na execução foi utilizado o algoritmo SHA 256 e a execução decorreu em 16 segundos, teve uma velocidade de 2703k palavras por segundo.

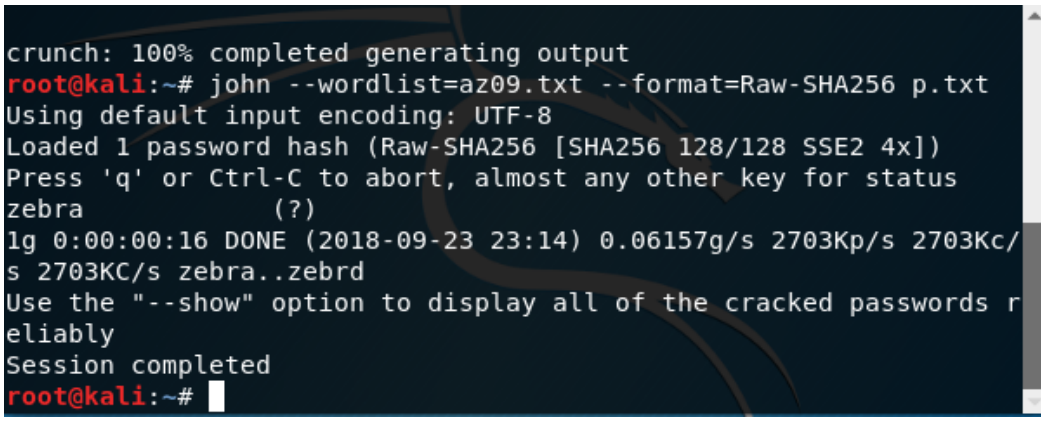


Figura 37: Uma corrida na ferramenta JtR com SHA256

A Figura 38 ilustra a corrida do *hash* da palavra "zebra" no algoritmo SHA 512. Esta corrida teve uma duração de 38 segundos e teve uma velocidade 1141k palavras por segundo.

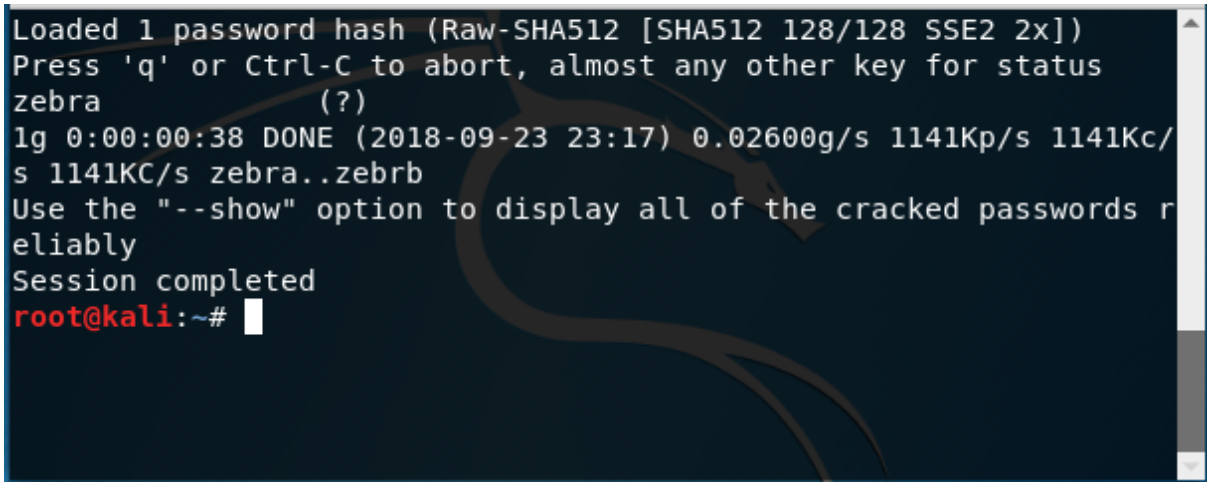


Figura 38: Uma corrida na ferramenta JtR com SHA512

5.2.2 Hashcat

O *Hashcat*[33] é um dos sistemas de recuperação de palavras-passe mais utilizado atualmente. Está abrangido por uma licença Expat. Suporta cinco modos de ataque e abrange mais de 200 algoritmos de funções de resumo (*hash*). O *Hashcat* opera com os recursos de hardware CPU e GPU e outros aceleradores de hardware. Existe distribuições para Linux, Windows e MAC OS. Tem a capacidade de funcionar com *OpenCL* o que permite a capacidade de paralelismo entre plataformas heterogenias.

A ilustração da Figura 39 demostra a execução da palavra "zebra" sobre o algoritmo SHA 256, esta execução decorreu em 44 segundos e apresenta um "*Speed.Dev.#01*" de 1173. Este valor trata-se de uma referência à capacidade de processar palavras por segundo de acordo com o hardware do utilizador. A Figura 40 segue a sequência de testes da palavra "zebra", contudo neste caso foi utilizado o algoritmo SHA 512. A execução decorreu em 46 segundos e a referência de velocidade ficou pelas 1027 palavras por segundo.

[33] https://Hashcat.net/Hashcat/

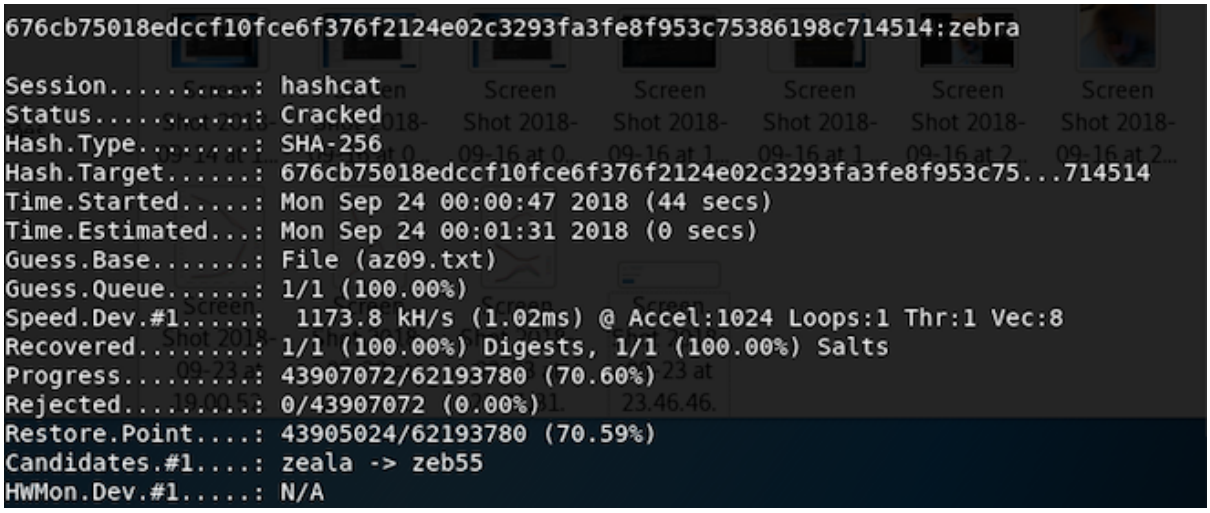


Figura 39: Uma corrida na ferramenta *Hashcat* com SHA 256

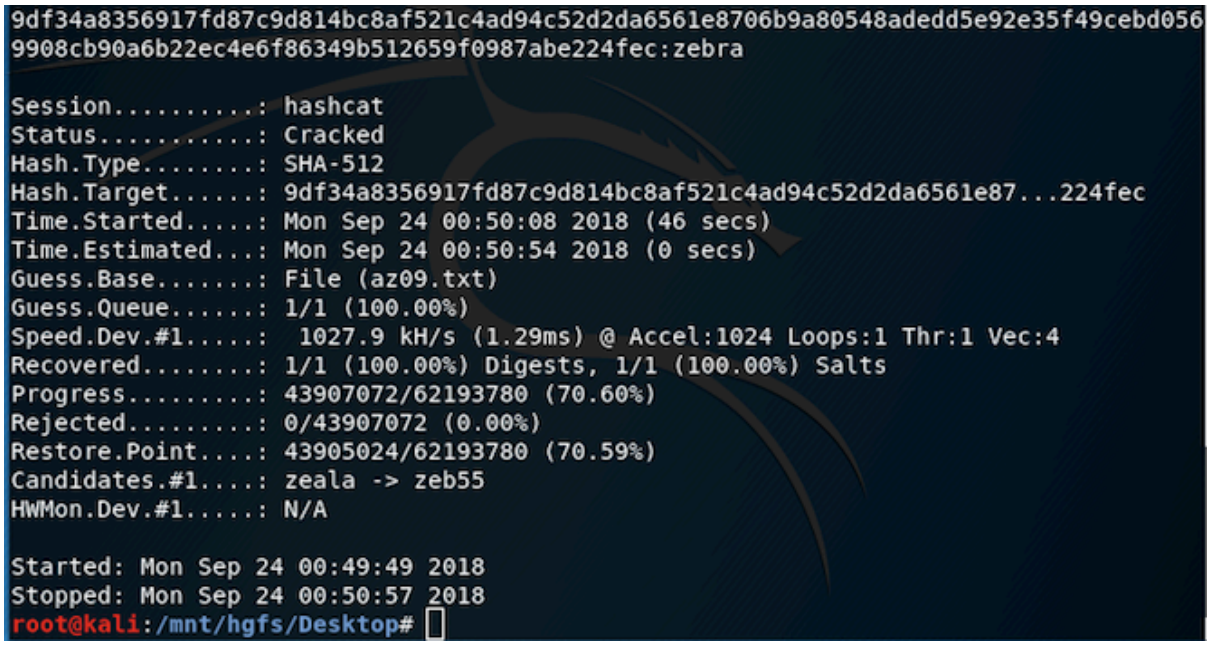


Figura 40: Uma corrida na ferramenta *Hashcat* com SHA 512

5.3.5 Conclusões

As ferramentas *John the Ripper* e *Hashcat* apresentam um conjunto bastante alargado de funcionalidades que permitem analisar palavras-passe. No entanto, para efeitos de teste nesta dissertação, apenas foi utilizado o recurso de força bruta com a inclusão de uma lista previamente elaborada (ver Anexo I). A capacidade de resolução dos *hash* é bastante satisfatória, tento em conta os tempos de execução e a velocidade que demostram. No que diz respeito à comparação entre estas ferramentas e o sistema desenvolvido, a Figura 41 ilustra os desempenhos obtidos das execuções realizadas.

Foram utilizados os algoritmos SHA 256 e o SHA 512 que se encontram ilustrados na *Figura 41*. O *John the Ripper* obteve os melhores resultados, quer ao nível da velocidade e consequentemente na rapidez da resolução do *hash*. Isso deve-se ao facto do *John the Ripper* conseguir gerir a alocação do conjunto de dados na memória RAM (*Random Access Memory*) de uma forma eficiente, dessa forma consegue ler as palavras a comparar as mesmas com elevados níveis de velocidade. Além disso, quer o *John the Ripper* quer o *Hashcat* não sofrem de problemas que possam provir da latência da rede como o DistPass, por se tratar de um sistema distribuído. Considerando também que estas ferramentas conseguem utilizar uma maior capacidade de computação porque não estão dependentes de um *browser Web.*

No caso do sistema desenvolvido, o teste realizado com 10 nós ativos não conseguiu superar o *Hashcat*. Contudo, o teste realizado com 15 nós ativos, o desempenho foi superior em nível de tempo e de velocidade conseguindo obter melhores valores em 4 das 5 palavras testadas.

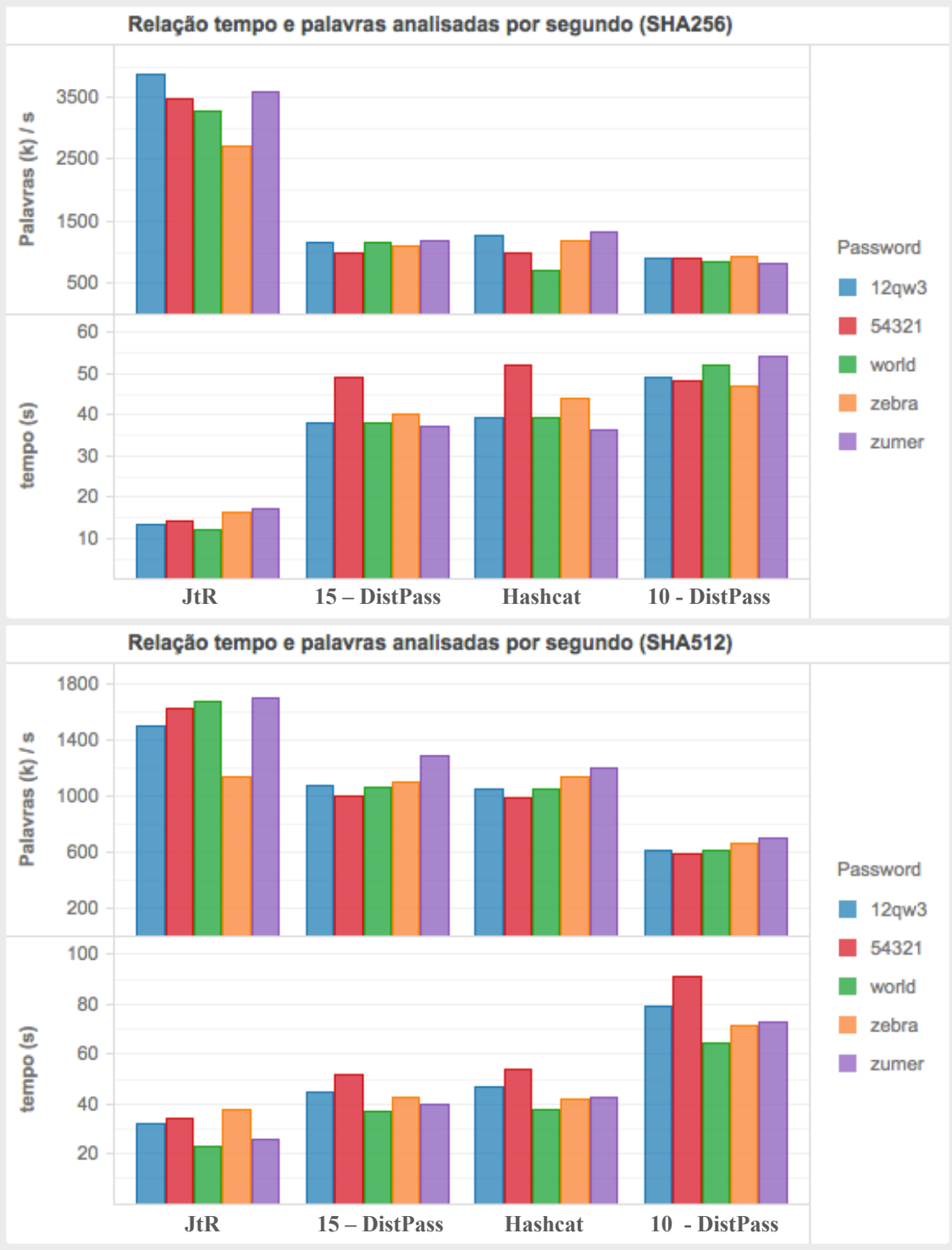


Figura 41:Resumo dos testes das ferramentas JtR, *Hashcat* e o DistPass.

Capítulo 6 Conclusões e Trabalho Futuro

Neste capítulo abordaremos as principais conclusões deste trabalho e indicamos várias direções de trabalho futuro que permitirão a evolução do sistema desenvolvido por forma a ganhar uma maior abrangência na verificação das palavras-passe.

6.1 Conclusão

O principal objetivo desta dissertação era investigar e perceber de que forma a computação distribuída e voluntária através da WWW poderia contribuir para ajudar a segurança da informação, focada em conscientizar os utilizadores a criação e utilização de palavras-passe mais seguras e robustas. Neste sentido, foi desenhado, proposto e desenvolvido um sistema que pudesse dar resposta à questão de investigação, que teve como propósito quebrar palavras-passe quer as mesmas se encontrassem em claro quer tivessem sido submetidas a funções criptográficas de resumo. Para a realização do trabalho de investigação nesta dissertação foi selecionada uma metodologia de investigação que pudesse criar um fio condutor, que resultasse na criação de um artefacto final que pudesse ser estudado e validado. Para tal, foi utilizada a metodologia *Design Science Research* que incorpora um conjunto de fases que conduziram o trabalho desta dissertação.

O levantamento do estado da arte realizado permitiu alavancar o conhecimento sobre as temáticas da análise da robustez das palavras-passe e da computação distribuída, em especial na sua vertente voluntária. Procedeu-se assim, a uma análise em duas áreas distintas. Por um lado, a temática da computação distribuída que tem vindo a crescer com a massificação dos dispositivos interligados em rede, por outro lado, a temática da análise da robustez de um dos mecanismos de autenticação mais usados - as palavras-passe. Foi possível identificar múltiplos projetos que abordam estas duas áreas, sendo que a grande maioria dos projetos identificados são projetos que se dedicam a resolução de trabalhos distribuídos, sem que estejam especificamente vocacionados para a verificação da robustez de palavras-passe, mas que podem ser adaptados para tal.

Para testar e validar o sistema desenvolvido nesta dissertação, foram utilizadas duas ferramentas de quebra e análise de palavras-passe que funcionam de forma não-distribuída. Para tal, foi elaborado um método comparativo entre o comportamento destas ferramentas

(*John the Ripper* e *Hashcat*) com a que foi desenvolvida no contexto desta dissertação. Apesar de ambas as ferramentas serem muito utilizadas a nível profissional, as mesmas são muito direcionadas para profissionais de segurança e utilizadores avançados. Um utilizador comum teria bastante dificuldade em conferir se as suas palavras-passe são verdadeiramente robustas utilizado estas ferramentas que exigem instalação das mesmas e algum conhecimento prévio e que envolvem alguma complexidade de utilização, quando comparadas com a solução desenvolvida.

Os resultados obtidos na solução proposta e desenvolvida nesta dissertação em comparação com as ferramentas supracitadas permitem concluir que as mesmas são semelhantes quer ao nível da taxa de sucesso quer ao nível do seu desempenho em termos de tempo e de velocidade. Adicionalmente, considerando que o sistema desenvolvido é extremamente escalável em termos do número de nós computacionais que podem contribuir de forma voluntária para a resolução do problema da verificação da robustez da palavra-passe, o mesmo permite aumentar a velocidade de computacional disponível para testar palavras-passe mais complexas e lidar com lotes de palavras-passe.

Acresce ainda o facto, que o sistema desenvolvido teve em consideração a capacidade de interação simples e acessibilidade oferecida aos utilizadores comuns, para que os mesmo não tivessem que ter um nível de conhecimento muito aprofundado para poderem testar de várias formas a robustez de palavras passe.

Ao longo deste trabalho houve igualmente uma preocupação de permitir que o mesmo pudesse ser escalável para suportar diversos algoritmos criptográficos de geração de resumos de palavras-passe, para que os mesmos pudessem ser suportados no futuro através de extensões desenvolvidas para o sistema. Por outro lado, e por forma a permitir precisamente isto, o código-fonte do sistema desenvolvido foi disponibilizado no *GitHub*, no seguinte endereço (https://github.com/lclms/distpass.git).

Podemos assim concluir que a questão de investigação formulada inicialmente para este trabalho pode ser respondida na medida em que a computação distribuída voluntária na WWW pode efetivamente ser utilizada para ajudar a melhorar a segurança da informação no que respeita à análise de robustez das palavras-chave, um dos principais mecanismos de autenticação utilizados. Ficou assim demostrado, que o sistema desenvolvido pode aplicar uma série de técnicas de quebra de palavras-chave, quer em texto em claro, quer usando funções criptográficas de resumo, como forma de validar computacionalmente a robustez das palavras-passe em termos do peso computacional para a quebra das mesmas, quer do tempo que demora a efetuar essa operação.

6.2 Trabalho futuro

A segurança da informação será sempre um edifício inacabado por surgirem novas tecnologias, novos produtos e paradigmas que revolucionam as tecnologias da informação. Isto obriga, por um lado, os profissionais das tecnologias de informação a uma aprendizagem contínua, como por outro, os utilizadores comuns a estarem atentos às questões relacionadas com esta temática para conseguirem antecipar-se aos ataques.

O sistema desenvolvido nesta dissertação procura contribuir para a resolução de alguns destes problemas, focando-se num aspeto muito específico que está relacionado com a robustez de um dos mecanismos mais utilizados de autenticação – as palavras-passe. Apesar do sistema desenvolvido contribuir para aferir a robustez das palavras-passe, recorrendo a computação distribuída e voluntária através da *Web*, devido aos constrangimentos temporais que estão normalmente associados à realização destes trabalhos, algumas questões foram deixadas de parte, assim como algumas limitações da solução foram identificadas, constituindo por isso trabalho a ser realizado no futuro.

Atualmente a plataforma desenvolvida está limitada a palavras existentes na base de dados, ou seja, se um utilizador colocar a teste uma palavra-passe que ultrapasse o conjunto de dados, essa palavra-passe não vai ser encontrada. Um ponto de evolução desta solução passaria por à semelhança do trabalho da resolução da palavra-passe ser feito por utilizadores voluntários, a expansão do conjunto de dados em vez de ser feito por um administrador que é um processo muito moroso, este trabalho também ser dado a realizar por utilizadores voluntários. Quando colocamos mais complexidade com mais combinações de caracteres, isso leva a um aumento exponencial do conjunto de dados, com efeito no tempo de processamento de novas combinações. Assim, o espaço de amplitude do conjunto de dados de força bruta que garante a plenitude de um dicionário limitado seria expandido para cobrir maiores combinações de caracteres. Outro aspeto de melhoria deste trabalho e que poderia constituir um futuro trabalho de investigação prende-se com a capacidade de trabalhar os valores recolhidos e poder criar elementos preditivos de análise e comportamento dos utilizadores, esta plataforma poderia evoluir para uma modalidade em que as organizações poderiam beneficiar da fonte de informação residente na solução.

Anexo I – Conjunto de dados Força Bruta

O conjunto de dados possui um total de palavras do 62193780. A tabela que acompanha este anexo, detalha de forma numérica os valores pertencentes a cada caractere bem como a sua representação percentual no conjunto de dados.

Fazem parte deste conjunto os caracteres:

a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3 4 5 6 7 8 9

Caracteres	Total de Palavras	Percentagem
1	36	0,0001
2	1296	0,0021
3	46656	0,0750
4	1679616	2,7006
5	60466176	97,222
Total	**62193780**	**100**

Anexo II – Conjunto de palavras-passe a testar

#	Palavra-passe	Tamanho	Entropia (bits)[34]	Espaço caracteres	SHA 256	SHA 512
1	zebra	5	15,1	26	676cb75018edccf10fce6f376f2124e02c3293fa3fe8f953c75386198c714514	9df34a8356917fd87c9d814bc8af521c4ad94c52d2da6561e8706b9a80548adedd5e92e35f49cebd0569908cb90a6b22ec4e6f86349b512659f0987abe224fec
2	12qw3	5	16,6	36	72142b1c635213cd4c7f7de181f3ca4ea5097c1c36b2bec4b199289770a107d4	9a4ff35204282e12193b2eb461a71c5f534148d7a00140766a9d0dccbc5dd07bc7a0cd1dad73eb0f7b8ce59b62a4857ed9f58dfd80fcc94b8843ae7baac88eae
3	54321	5	7,7	10	20f3765880a5c269b747e1e906054a4b4a3a991259f1e16b5dde4742cec2319a	e16d6b316f3bef1794c548b7a98b969a6aacb02f6ae5138efc1c443ae6643a6a77d92a0e33e382d6cbb7758f9ab25ab0f97504554d1904620a41fed463796fc2

[34] http://rumkin.com/tools/password/passchk.php

4	zumer	5	14,8	26	545c78171b1ccf75547498e166f8f9b73c53073bd77cc9c84a9e1c83461e3ae9	c16ec76e705b0f1a313a855a3329bc94cd885966b0405a58243c22738ee7702e038656452da63da2a9e1c310c35608200b606d9f89aa24a586aabf2b26df397f
5	world	5	16,8	26	486ea46224d1bb4fb680f34f7c9ad96a8f24ec88be73ea8e5a6c65260e9cb8a7	11853df40f4b2b919d3815f64792e58d08663767a494bcbb38c0b2389d9140bbb170281b4a847be7757bde12c9cd0054ce3652d0ad3a1a0c92babb69798246ee

Anexo III – Testes realizados sobre o conjunto de dados

SHA 256			
Ferramenta	Password	tempo (s)	Palavras (k) / s
JtR	zebra	38	1141
JtR	12qw3	32	1498
JtR	54321	34	1620
JtR	zumer	26	1706
JtR	world	23	1671
Hashcat	zebra	42	1135
Hashcat	12qw3	47	1044
Hashcat	54321	54	993
Hashcat	zumer	43	1197
Hashcat	world	38	1044
6 DistPass	zebra	113	420
6 DistPass	12qw3	126	386
6 DistPass	54321	145	367
6 DistPass	zumer	116	443
6 DistPass	world	102	386
4 DistPass	zebra	134	354
4 DistPass	12qw3	150	326
4 DistPass	54321	172	310
4 DistPass	zumer	137	374
4 DistPass	world	121	326
10 DistPass	zebra	71	667
10 DistPass	12qw3	79	614
10 DistPass	54321	91	584
10 DistPass	zumer	73	704
10 DistPass	world	64	614
15 DistPass	zebra	40	1091
15 DistPass	12qw3	38	1149
15 DistPass	54321	49	990
15 DistPass	zumer	37	1180
15 DistPass	world	38	1148

SHA 512			
Ferramenta	Password	tempo (s)	Palavras (k) / s
JtR	zebra	16	2703
JtR	12qw3	13	3848
JtR	54321	14	3464
JtR	zumer	17	3568
JtR	world	12	3276
Hashcat	zebra	44	1173
Hashcat	12qw3	39	1270
Hashcat	54321	52	988
Hashcat	zumer	36	1337
Hashcat	world	39	710
6 DistPass	zebra	104	420
6 DistPass	12qw3	105	476
6 DistPass	54321	108	449
6 DistPass	zumer	109	556
6 DistPass	world	111	354
4 DistPass	zebra	125	351
4 DistPass	12qw3	126	397
4 DistPass	54321	124	391
4 DistPass	zumer	127	477
4 DistPass	world	123	319
10 DistPass	zebra	47	929
10 DistPass	12qw3	49	891
10 DistPass	54321	48	910
10 DistPass	zumer	54	808
10 DistPass	world	52	840
15 DistPass	zebra	43	1101
15 DistPass	12qw3	45	1078
15 DistPass	54321	52	995
15 DistPass	zumer	40	1285
15 DistPass	world	37	1062

Tabela 1: Testes realizados sobre o conjunto de dados. A ferramenta DistPass suporta esta dissertação e aparece nesta tabela com a utilização de 4, 6, 8 e 15 nós clientes.

Bibliografia

Alebrahim, S., & Ahmad, I. (2016). Task scheduling for heterogeneous computing systems. *The Journal of Supercomputing*. https://doi.org/10.1007/s11227-016-1917-2

Ali, K. (2014). Grid Scheduling Strategies for Applications that require Data Transfer.

Almeida, P. J., & Napp, D. (2017). *Criptografia e Segurança*. Publindústria, Prudução de Comunicação, Lda.

Aloisio, G., Cafaro, M., & Epicoco, I. (2002). Early experiences with the GridFTP protocol using the GRB-GSIFTP library. *Future Generation Computer Systems*, *18*(8), 1053–1059. https://doi.org/10.1016/S0167-739X(02)00084-5

Aminzade, M. (2018). Confidentiality, integrity and availability – finding a balanced IT framework. *Network Security*, *2018*(5), 9–11. https://doi.org/10.1016/S1353-4858(18)30043-6

Anwar Hayder, W., & Hassan Husain, M. (2018). Intelligent Application Implementation Model for Automated Agent Negotiation. *Kurdistan Journal for Applied Research*, *3*(1), 68–74. https://doi.org/10.24017/science.2018.1.14

Bangare, S. L., Gupta, S., Dalal, M., & Inamdar, A. (2016). Using Node . Js to Build High Speed and Scalable Backend Database Server, (March), 61–64.

Barkadehi, M. H., Nilashi, M., Ibrahim, O., Zakeri Fardi, A., & Samad, S. (2018). Authentication systems: A literature review and classification. *Telematics and Informatics*, *35*(5), 1491–1511. https://doi.org/https://doi.org/10.1016/j.tele.2018.03.018

Berman, F., Chien, A., Cooper, K., Dongarra, J., Foster, I., Gannon, D., … Wolski, R. (2001). The GrADS project: Software support for high-level grid application development. *International Journal of High Performance Computing Applications*, *15*(4), 327–344. https://doi.org/10.1177/109434200101500401

Boldrin, F., Taddia, C., & Mazzini, G. (2007). Distributed Computing Through Web Browser. *2007 IEEE 66th Vehicular Technology Conference*, (November), 2020–2024. https://doi.org/10.1109/VETECF.2007.424

Bonneau, J., Herley, C., van Oorschot, P. C., & Stajano, F. (2015). Passwords and the evolution of imperfect authentication. *Communications of the ACM*, *58*(7), 78–87. https://doi.org/10.1145/2699390

Bote-lorenzo, M. L., Dimitriadis, Y. A., & Gómez-Sánchez, E. (2004). Grid Characteristics and Uses: A Grid Definition. *Grid Computing (First European Across Grids*

Conference, Santiago de Compostela, Spain, February 13-14, 2004. Revised Papers), 291–298. https://doi.org/10.1007/978-3-540-24689-3_36

Breternitz, M. (2017). High Performance Computing at Exascale : Application Requirements and Technology Development.

Chandak, A. (2011). An Overview of Task Scheduling and Performance Metrics in Grid Computing. *International Journal of Research and Reviews in Computer Science*, *2*(2), 30–33.

Chorazyk, P., Godzik, M., Pietak, K., Turek, W., Kisiel-Dorohinicki, M., & Byrski, A. (2017). Lightweight Volunteer Computing Platform using Web Workers. *Procedia Computer Science*, *108*, 948–957. https://doi.org/10.1016/j.procs.2017.05.091

Dai, Y. S., & Wang, X. L. (2006). Optimal resource allocation on grid systems for maximizing service reliability using a genetic algorithm. *Reliability Engineering and System Safety*, *91*(9), 1071–1082. https://doi.org/10.1016/j.ress.2005.11.008

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, *51*(1), 107. https://doi.org/10.1145/1327452.1327492

Debski, R., Krupa, T., & Majewski, P. (2013). ComcuteJS: A Web Browser Based Platform for Large-scale Computations. *Computer Science*, *14*(1), 143. Retrieved from https://journals.agh.edu.pl/csci/article/view/113

Ding, L., Kang, W., & Wang, L. (2013). An on-line auction method for resource allocation in computational grids. *Journal of Chemical and Pharmaceutical Research*, *5*(9), 241–247. https://doi.org/10.1016/j.future.2009.08.010

Duermuth, M., Angelstorf, F., Castelluccia, C., Perito, D., Duermuth, M., Angelstorf, F., … Perito, D. (2015). OMEN: Faster Password Guessing Using an Ordered Markov Enumerator.

Durillo, J. J., Nae, V., & Prodan, R. (2014). Multi-objective energy-efficient workflow scheduling using list-based heuristics. *Future Generation Computer Systems*, *36*, 221–236. https://doi.org/10.1016/j.future.2013.07.005

Fat, N., Vujovic, M., Papp, I., & Novak, S. (2016). Comparison of AngularJS framework testing tools. *2016 Zooming Innovation in Consumer Electronics International Conference, ZINC 2016*, 76–79. https://doi.org/10.1109/ZINC.2016.7513659

Fenton, J. L., Newton, E. M., Perlner, R. A., Regenscheid, A. R., Burr, W. E., Richer, J. P., … Burr, W. E. (2017). Digital Identity Guidelines.

Foster, I., & Llorente, I. M. (2009). Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, *13*(5), 14–22. https://doi.org/1089-7801

Fox, D., Sillito, J., & Maurer, F. (2008). Exploring Principles of User-Centered Agile Software Development: A Literature Review. *Proceedings of the 2008 Agile Conference (AGILE 2008)*, *61*(MAY), 63–72. https://doi.org/10.1109/Agile.2008.78

Galbally, J., Coisel, I., & Sanchez, I. (2017a). A New Multimodal Approach for Password Strength Estimation-Part II: Experimental Evaluation. *IEEE Transactions on Information Forensics and Security*, *12*(12), 2845–2860. https://doi.org/10.1109/TIFS.2017.2730359

Galbally, J., Coisel, I., & Sanchez, I. (2017b). A New Multimodal Approach for Password Strength Estimation - Part I: Theory and Algorithms. *IEEE Transactions on Information Forensics and Security*, *12*(12), 2829–2844. https://doi.org/10.1109/TIFS.2016.2636092

He, Y., Hsu, W. J., & Leiserson, E. E. (2008). Provably efficient online nonclairvoyant adaptive scheduling. *IEEE Transactions on Parallel and Distributed Systems*, *19*(9), 1263–1279. https://doi.org/10.1109/TPDS.2008.39

Hevner, A., & Chatterjee, S. (2010). Design Research in Information Systems, *22*, 9–23. https://doi.org/10.1007/978-1-4419-5653-8

Hinings, B., Gegenhuber, T., & Greenwood, R. (2018). Digital innovation and transformation: An institutional perspective. *Information and Organization*, *28*(1), 52–61. https://doi.org/10.1016/j.infoandorg.2018.02.004

Idris, H., Ezugwu, A. E., Junaidu, S. B., & Adewumi, A. O. (2017). An improved ant colony optimization algorithm with fault tolerance for job scheduling in grid computing systems. *PLoS ONE (3区, 2.806)*, *12*(5), e0177567. https://doi.org/10.1371/journal.pone.0177567

Jain, N. (2014). Review of Different Responsive Css Front-End Frameworks. *Journal of Global Research in Computer Science*, *5*(11), 6. Retrieved from www.newaeonweb.com.br/responsiveaeon

Jiang, H. (2017). The Map of Cybersecurity Domains (version 2.0). Retrieved September 8, 2018, from https://www.linkedin.com/pulse/map-cybersecurity-domains-version-20-henry-jiang-ciso-cissp/

Jose, J., Tomy, T. T., Karunakaran, V., Anjali Krishna, V., Varkey, A., & Nisha, C. A. (2016). Securing passwords from dictionary attack with character-tree. *Proceedings of the 2016 IEEE International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2016*, 2301–2307. https://doi.org/10.1109/WiSPNET.2016.7566553

Kanani, B., & Maniyar, B. (2015). Review on Max-Min Task scheduling Algorithm for Cloud Computing, *2*(3), 781–784.

Kelley, P. G., Komanduri, S., Mazurek, M. L., Shay, R., Vidas, T., Bauer, L., … López, J. (2012). Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. *Proceedings - IEEE Symposium on Security and Privacy*, 523–537. https://doi.org/10.1109/SP.2012.38

Kim, N. S., Chen, D., Xiong, J., & Hwu, W. M. W. (2017). Heterogeneous Computing Meets Near-Memory Acceleration and High-Level Synthesis in the Post-Moore Era. *IEEE Micro*, *37*(4), 10–18. https://doi.org/10.1109/MM.2017.3211105

Komarov, M., Konovalov, N., & Kazantsev, N. (2016). Browser-based harnessing of voluntary computational power, *4*, 3–42. https://doi.org/10.1515/fcds-201

Konishi, F., Ohki, S., Ishii, M., Umestu, R., & Konagaya, A. (2007). RABC: A conceptual design of pervasive infrastructure for browser computing based on ajax technologies. *Proceedings - Seventh IEEE International Symposium on Cluster Computing and the Grid, CCGrid 2007*, 661–666. https://doi.org/10.1109/CCGRID.2007.91

Le-Khac, N.-A., Kechadi, M.-T., & Carthy, J. (2017). Admire framework: Distributed data mining on data grid platforms. Retrieved from http://arxiv.org/abs/1703.09756

Leitner, P., Hummer, W., Satzger, B., Inzinger, C., & Dustdar, S. (2012). Cost-efficient and application SLA-aware client side request scheduling in an infrastructure-as-a-service cloud. *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, 213–220. https://doi.org/10.1109/CLOUD.2012.21

Li, Y., Wang, H., & Sun, K. (2017). Personal Information in Passwords and Its Security Implications. *IEEE Transactions on Information Forensics and Security*, *12*(10), 2320–2333. https://doi.org/10.1109/TIFS.2017.2705627

MacWilliam, T., & Cecka, C. (2013). CrowdCL: Web-based volunteer computing with WebCL. *2013 IEEE High Performance Extreme Computing Conference, HPEC 2013*. https://doi.org/10.1109/HPEC.2013.6670348

Mahoney, S. (2018). An Introduction to Web Workers. Retrieved August 8, 2018, from https://medium.com/@siobhanpmahoney/a-brief-introduction-to-web-workers-e5d6e39d9d28

Maitin-Shepard, J., Tibouchi, M., & Aranha, D. F. (2017). Elliptic Curve Multiset Hash. *Computer Journal*, *60*(4), 476–490. https://doi.org/10.1093/comjnl/bxw053

Manitz, J., Harbering, J., Schmidt, M., Kneib, T., & Schöbel, A. (2017). Source estimation for propagation processes on complex networks with an application to delays in public transportation systems. *Journal of the Royal Statistical Society. Series C: Applied Statistics*, *66*(3), 521–536. https://doi.org/10.1111/rssc.12176

Martínez, G. J., & Val, L. (2015). Capataz: a framework for distributing algorithms via the

World Wide Web. *Clei Electronic Journal*, *18*(1), 1–12.

Martínez, G., Val, L., Martinez, G., & Val, L. (2014). Implementing crossplatform distributed algorithms using standard web technologies. *2014 XL Latin American Computing Conference (CLEI)*, 1–8. https://doi.org/10.1109/CLEI.2014.6965143

McEwan, A. A. (2010). Concurrency and Computation Practice and Experience: Guest editorial. *Concurrency Computation Practice and Experience*, *22*(8), 909–911. https://doi.org/10.1002/cpe.1455

Morris, L., & McAtee, M. (2005). cracklord. Retrieved June 26, 2018, from http://jmmcatee.github.io/cracklord/

Morris, R., & Thompson, K. (1979). Password security: A case history. *Communications of the ACM*, *22*(11), 594–597. https://doi.org/10.1145/359168.359172

Naarttijärvi, M. (2018). Balancing data protection and privacy – The case of information security sensor systems. *Computer Law and Security Review*, *000*, 1–20. https://doi.org/10.1016/j.clsr.2018.04.006

Olteanu, A., Pop, F., Dobre, C., & Cristea, V. (2012). A dynamic rescheduling algorithm for resource management in large scale dependable distributed systems. *Computers and Mathematics with Applications*, *63*(9), 1409–1423. https://doi.org/10.1016/j.camwa.2012.02.066

Pan, Y., White, J., & Sun, Y. (2017). Assessing the threat of web worker distributed attacks. *2016 IEEE Conference on Communications and Network Security, CNS 2016*, 306–314. https://doi.org/10.1109/CNS.2016.7860498

Pan, Y., White, J., Sun, Y., & Gray, J. (2015). Gray computing: An analysis of computing with background javascript tasks. *Proceedings - International Conference on Software Engineering*, *1*(May), 167–177. https://doi.org/10.1109/ICSE.2015.38

Raja, M. M., & Noordeen, A. M. (2017). DOMAIN BASED RESOURCE INFORMATION. *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, (Iciss), 1212–1218.

Reda, N. M., Tawfik, A., Marzok, M. A., & Khamis, S. M. (2014). Sort-Mid tasks scheduling algorithm in grid computing. *Journal of Advanced Research*, *6*(6), 987–993. https://doi.org/10.1016/j.jare.2014.11.010

S, S. M. S., Kaviyaraj, R., & Susmitha, U. (2018). Smart Attendance Monitoring System Using Wifi and Mac Address, (2), 2393–2395.

Safa, N. S., Sookhak, M., Von Solms, R., Furnell, S., Ghani, N. A., & Herawan, T. (2015). Information security conscious care behaviour formation in organizations. *Computers and Security*, *53*, 65–78. https://doi.org/10.1016/j.cose.2015.05.012

Sahin, C. S., Lychev, R., & Wagner, N. (2015). General Framework for Evaluating Password Complexity and Strength, 1–11. Retrieved from http://arxiv.org/abs/1512.05814

Schaub, F., Deyhle, R., & Weber, M. (2012). Password entry usability and shoulder surfing susceptibility on different smartphone platforms. *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia - MUM '12*, 1. https://doi.org/10.1145/2406367.2406384

Shay, R., Cranor, L. F., Komanduri, S., Durity, A. L., Huh, P. (Seyoung), Mazurek, M. L., … Christin, N. (2016). Designing Password Policies for Strength and Usability. *ACM Transactions on Information and System Security*, *18*(4), 1–34. https://doi.org/10.1145/2891411

Shen, C., Yu, T., Xu, H., Yang, G., & Guan, X. (2016). User practice in password security: An empirical study of real-life passwords in the wild. *Computers and Security*, *61*, 130–141. https://doi.org/10.1016/j.cose.2016.05.007

Shoufan, A., & Damiani, E. (2017). On inter-Rater reliability of information security experts. *Journal of Information Security and Applications*, *37*, 101–111. https://doi.org/10.1016/j.jisa.2017.10.006

Spenke, F., Balzer, K., Frick, S., Hartke, B., & Dieterich, J. M. (2018). Adaptive parallelism with RMI: Idle high-performance computing resources can be completely avoided. Retrieved from http://arxiv.org/abs/1801.07184

Taneski, V., Hericko, M., & Brumen, B. (2014). Password security - No change in 35 years? *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2014 - Proceedings*, 1360–1365. https://doi.org/10.1109/MIPRO.2014.6859779

Tchernykh, A., Schwiegelsohn, U., Talbi, E. ghazali, & Babenko, M. (2016). Towards understanding uncertainty in cloud computing with risks of confidentiality, integrity, and availability. *Journal of Computational Science*. https://doi.org/10.1016/j.jocs.2016.11.011

Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, *14*(6), 80–83. https://doi.org/10.1109/MIC.2010.145

Time required to brute-force crack a password depending on password entropy (strength). (2015). Retrieved June 26, 2018, from https://www.reddit.com/r/dataisbeautiful/comments/322lbk/time_required_to_bruteforce_crack_a_password/

Ur, B., Bees, J., Segreti, S. M., Bauer, L., Christin, N., & Cranor, L. F. (2016). Do Users'

Perceptions of Password Security Match Reality? *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*, 3748–3760. https://doi.org/10.1145/2858036.2858546

Vambol, A., Kharchenko, V., Potii, O., & Bardis, N. (2018). McEliece and Niederreiter Cryptosystems Analysis in the Context of Post-Quantum Network Security. *Proceedings - 2017 4th International Conference on Mathematics and Computers in Sciences and in Industry, MCSI 2017*, *2018–Janua*, 134–137. https://doi.org/10.1109/MCSI.2017.31

van Steen, M., & Tanenbaum, A. S. (2016). A brief introduction to distributed systems. *Computing*, *98*(10), 967–1009. https://doi.org/10.1007/s00607-016-0508-7

Weir, M., Aggarwal, S., De Medeiros, B., & Glodek, B. (2009). Password cracking using probabilistic context-free grammars. *Proceedings - IEEE Symposium on Security and Privacy*, (May 2009), 391–405. https://doi.org/10.1109/SP.2009.8

Wheeler, D. L. (2016). zxcvbn : Low-Budget Password Strength Estimation This paper is included in the Proceedings of the zxcvbn : Low-Budget Password Strength Estimation.

wpengine. (2015). Unmasked: What 10 million passwords reveal about the people who choose them. Retrieved June 26, 2018, from https://wpengine.com/unmasked/

Young, I. (2018). The Entity Category Security Assertion Markup Language (SAML) Attribute Types, 1–12.

Zerbino, P., Aloini, D., Dulmin, R., & Mininno, V. (2018). Process-mining-enabled audit of information systems: Methodology and an application. *Expert Systems with Applications*, *110*, 80–92. https://doi.org/10.1016/j.eswa.2018.05.030