TreasureHunt: um gerador automático de competições de Segurança Computacional

Ricardo de la Rocha Ladeira
Instituto Federal Catarinense
Campus Blumenau
Blumenau, Brasil
ricardo.ladeira@ifc.edu.br

Rafael R. Obelheiro
Universidade do Estado de Santa Catarina
Campus Joinville
Joinville, Brasil
rafael.obelheiro@udesc.br

Richard Robert Dias Custódio
Instituto Federal Catarinense
Campus Blumenau
Blumenau, Brasil
richardsoncusto@gmail.com

Vinícius Manuel Martins
Instituto Federal Catarinense
Campus Blumenau
Blumenau, Brasil
vinicius.m.m.2002@gmail.com

RESUMO

Segurança Computacional é uma área emergente e, embora importante, carece de profissionais, pois há falta de formação nesta área. No contexto da educação formal, a aquisição de habilidades práticas está pouco presente, e uma das estratégias para resolver esta questão é o uso de jogos. Os jogos de desafio vêm ganhando espaço e ajudam a resolver essas e outras questões, tais como conscientização e interesse pela área. A dificuldade de operacionalizar estes jogos e de criar exercícios que não percam o fator surpresa indicam que gerar problemas automaticamente e que utilizem aleatorização podem ser uma solução viável. Neste sentido, este trabalho relata o desenvolvimento do TreasureHunt, uma ferramenta de geração automática de problemas e competições do tipo desafio para ensino de Segurança Computacional. Os resultados obtidos com competições geradas pelo TreasureHunt indicam que a geração automática de desafios é viável e que receptividade da atividade pelos estudantes foi positiva.

Palavras-chave

Geração Automática de Problemas; Segurança Computacional; Ensino.

CCS Concepts

•Applied computing → Education; •Social and professional topics → *Computer science education;*

1. INTRODUÇÃO

A importância da Segurança Computacional vem crescendo a cada ano. No entanto, pesquisas mostram que uma dificuldade ainda presente nesta área é a carência por profissionais com as habilidades necessárias para atuar neste campo, em contraste com uma grande demanda [7, 25]. Em vista disso, iniciativas como o uso de jogos e competições de Segurança têm sido usadas com sucesso para atrair talentos para a área [28].

Um dos tipos mais usuais de competições de Segurança Computacional é o *desafio*, em que os jogadores usam técnicas e ferramentas de Segurança para encontrar textos secretos conhecidos como *flags*. Esse tipo de competição sofre com dificuldades como o compartilhamento de respostas, que costumam ser idênticas para todos os competidores, a elaboração de problemas, tipicamente uma atividade manual, e o reaproveitamento de problemas entre competições, uma vez que é comum a divulgação de soluções após um desafio [8]. Experiências anteriores [1, 10, 18] mostram que a geração automática de instâncias de problemas com respostas únicas por competidor é uma forma de minimizar o impacto dessas dificuldades.

A geração automática de desafios de Segurança pode ser perseguida de diversas formas [1, 10, 24]. Este trabalho parte de um levantamento e classificação de 1250 problemas usados em 84 desafios de Segurança realizados nos anos de 2016 e 2017 para proporcionar as seguintes contribuições:

1. Identificação dos problemas mais populares, e como eles podem ser gerados automaticamente;
2. Um modelo representativo de competições do tipo desafio, incorporando as características mais comuns das competições analisadas;
3. Uma ferramenta para geração automática de desafios em harmonia com os resultados precedentes.

O restante do texto está organizado da seguinte maneira: a Seção 2 descreve os desafios de Segurança Computacional. A Seção 3 apresenta os trabalhos relacionados. A Seção 4 expõe uma pesquisa que mapeia informações sobre desafios recentes. A Seção 5 detalha a ferramenta desenvolvida. A Seção 6 descreve o uso da ferramenta em competições, e a Seção 7 apresenta as conclusões.

2. DESAFIOS DE SEGURANÇA

O uso de jogos e competições de Segurança Computacional colabora para o incremento de habilidades práticas, como evidenciam diversos trabalhos [29, 23, 5]. Além de aspectos técnicos, os competidores fortalecem também outras habilidades, tais como liderança e tomada de decisão [6]. Dentre os diversos tipos de jogos e competições, e a despeito de não haver uma taxonomia uniforme para estes, destacam-se os jogos de desafio, também chamados de *caça ao tesouro* ou *CTF Jeopardy!*.

Competições do tipo desafio consistem em conjuntos de problemas para os quais o objetivo é encontrar textos secretos (*flags*). Para encontrar as *flags*, os jogadores precisam de processos e ferramentas de Segurança, e tipicamente realizam estas tarefas sem interação com outros jogadores [18]. Há problemas, por exemplo, em que se busca descobrir um texto codificado em um arquivo de pacotes capturados na rede [22]; em outros, ferramentas de codificação, criptografia e esteganografia são necessárias para encontrar uma frase secreta [17].

Desafios são flexíveis quanto à complexidade de problemas e de infraestrutura. Eles podem conter tanto problemas básicos, que podem ser resolvidos com ferramentas simples e são adequados para iniciantes, quanto problemas mais complexos, que exijam infraestruturas computacionais sofisticadas, instalação e configuração de softwares e máquinas virtuais, por exemplo. Assim, desafios podem ser adaptados para um público-alvo específico, ou serem compostos por problemas de variados graus de dificuldade, que atinjam públicos mais amplos.

Desafios não preveem a interação direta entre jogadores/equipes, e, além disso, também permitem a prática individual, sem adversários. Nestes jogos, quando há adversários, a competição é baseada em sistemas de pontuação, ficando à frente aquele que obtém mais pontos. Estabelecer alvos fixos para todas as equipes promove a competição de uma maneira mais saudável do que ocorre em jogos nos quais equipes atacam umas às outras [27]. Quando não há adversários, o jogador pode testar o seu progresso verificando se consegue resolver os problemas.

Além do sistema de pontos, os desafios podem ser lineares, não lineares ou mistos. Um desafio linear é aquele em que os problemas precisam ser resolvidos em sequência (ou passando por *fases*), geralmente apresentando problemas com dificuldade crescente [3]. Em um desafio não linear, os problemas são independentes, podendo ser resolvidos em ordem arbitrária. Há ainda a possibilidade de um formato misto, combinando a natureza independente dos desafios não lineares em fases sequencialmente disponíveis ou liberadas conforme um número determinado de problemas – que eventualmente pode ser igual ao total de problemas – é resolvido dentro da fase [4].

No Brasil, entre os desafios mais conhecidos estão o Cryptorace e o Hackaflag[1], ambos da Roadsec, o Workshop de Forense Computacional do SBSeg, em 2015 e 2016, e, mais recentemente, o Hacker Security CTF[2], da Hacker Security. Nestes desafios o público-alvo não é controlado, de forma que qualquer pessoa possa participar. O nível dos exercícios também pode variar.

Em competições de desafio, um fator complicador para a replicação da atividade por parte dos criadores ou organizadores é a perda do fator surpresa dos problemas, pois, após a aplicação da competição, as respostas podem estar disponíveis, por exemplo, na *web*. Este problema é conhecido como *compartilhamento de problemas* [26]. Ademais, outro fator negativo é a possibilidade de cópia de respostas, um problema chamado de *compartilhamento de flags* [1]. Na Seção 5 deste trabalho, descreve-se o desenvolvimento da ferramenta TreasureHunt, um gerador de desafios que visa a mitigar essas ocorrências. Os problemas implementados na ferramenta foram escolhidos a partir de um levantamento de desafios já realizados, o qual será apresentado na Seção 4.

3. TRABALHOS RELACIONADOS

No âmbito da Cibersegurança, uma das ramificações de pesquisa para inserir jogos no contexto educacional envolve a geração automática de exercícios (ou problemas), pois assim evitam-se problemas como o compartilhamento de *flags* e a necessidade de descarte de exercícios. Esta Seção discute os principais trabalhos que envolvem Geração Automática de Problemas (*Automatic Problem Generation* — APG) em jogos de Segurança Computacional.

O PicoCTF [4, 1] é um jogo de desafio voltado para estudantes. Desenvolvido em 2014 e aplicado anualmente, o jogo é pioneiro em APG. A proposta do PicoCTF foi de gerar automaticamente problemas iguais, mas com *flags* diferentes, para todos os jogadores, com o objetivo de evitar o compartilhamento de respostas. Os problemas gerados automaticamente devem ser combinados manualmente pelo organizador para montar uma competição.

O MetaCTF [10] é uma competição do tipo desafio que passou a incorporar APG na versão de 2015, minimizando as possibilidades de trapaça. O jogo gera problemas distintos para cada jogador, com o objetivo de ensinar Engenharia Reversa e Análise de *Malware*, exigindo conhecimento em ferramentas como `gdb`, `readelf` e `objdump` [10]. O jogo segue utilizando APG e agora é chamado de PSU CTF[3].

O AutoCTF [12] foi uma competição do tipo desafio aplicada com estudantes, em maio de 2017. Esta competição utilizava uma ferramenta que gerava problemas aqui classificados como de Engenharia Reversa (e, pelos autores, classificada como pwn – exploração de vulnerabilidades) através da automatização da técnica de injeção de *bugs* em códigos C. Neste jogo, problemas eram liberados diariamente aos jogadores.

O SecGen [24] é uma ferramenta que gera competições automaticamente criando desafios aleatórios em máquinas virtuais distintas para cada jogador, chamadas de *cenários ricos*. Estes cenários variam em Sistema Operacional, serviços instalados, vulnerabilidades e exercícios. Os exercícios elaborados por essa ferramenta cobrem uma gama diversa de subáreas, tais como esteganografia, serviços de rede e vulnerabilidades em sistemas. Os resultados obtidos a partir do uso do SecGen indicam que a ferramenta foi bem recebida pelos jogadores, e que os problemas por ela gerados estiveram adequados em relação à complexidade.

Todos os trabalhos citados utilizam sistema de ranqueamento e adotam a jogabilidade não linear, com exceção do AutoCTF, para o qual a questão não se aplica. O uso de APG no PicoCTF se restringe à geração da *flag* e, além disso, cada exercício envolve somente uma técnica. O MetaCTF e o AutoCTF utilizam APG de forma restrita, possuindo problemas de Engenharia Reversa e, no caso do MetaCTF, eventualmente com algumas *flags* codificadas, ou seja, o conceito de composição de técnicas é utilizado de forma específica e restrita. O SecGen também trabalha a composição de técnicas de forma restrita, permitindo apenas a codificação da *flag* após a criação dos problemas. Há que se ponderar que tanto o MetaCTF quanto o SecGen, por proporcionarem a criação de problemas não uniformes, podem gerar uma competição desbalanceada, fornecendo pro-

[1] `https://roadsec.com.br/hackaflag/`
[2] `https://capturetheflag.com.br/`
[3] `https://cs201.oregonctf.org/`

Tabela 1: Comparativo entre os trabalhos relacionados e a ferramenta proposta.

Trabalho	Geração Automática	Composição de Problemas	Uniformidade de Problemas	Classes de Problemas
PicoCTF [1]	problemas	✗	✓	• Engenharia Reversa • *Web* • Miscelânea • Codificação/Criptografia
MetaCTF [10]	competição	±	✗	• Engenharia Reversa
AutoCTF [12]	problemas	✗	✓	• Engenharia Reversa
SecGen [24]	competição	±	✗	• *Web* • Forense • Miscelânea • Codificação/Criptografia
TreasureHunt	competição	✓	✓	• Engenharia Reversa • Forense • Miscelânea • Codificação/Criptografia

blemas com diferentes complexidades e que podem favorecer ou desfavorecer jogadores.

O TreasureHunt foi projetado para gerar competições inteiras de forma automática, permitindo a composição de problemas em quatro diferentes classes (a classificação de problemas é apresentada na Seção 4), com problemas uniformes e abordando técnicas de quatro diferentes classes. A Tabela 1 resume as informações da seção apresentando uma comparação entre os trabalhos relacionados e o TreasureHunt. O símbolo "✗" foi utilizado para indicar não conformidade ao atributo. O símbolo "✓" indica conformidade ao atributo, e o símbolo "±" indica que o atributo é cumprido parcialmente.

4. ANÁLISE DE DESAFIOS EXISTENTES

Conhecer o funcionamento dos desafios e os tipos mais comuns de problemas é vital para garantir a representatividade do modelo de competição e dos problemas implementados. A partir de um repositório de desafios [8], mantido por uma comunidade de entusiastas e participantes, analisou-se na íntegra o conjunto de competições ocorridas entre janeiro de 2016 e março de 2017. Embora a maioria dos problemas estejam categorizados no repositório, cada desafio adota uma classificação própria, e problemas similares podem aparecer em categorias distintas. Com base no enunciado dos exercícios e nas soluções divulgadas, procurou-se identificar as classes genéricas dos problemas, unificando categorias com problemas semelhantes. Após a análise de todos os 1250 problemas de 84 competições, emergiram cinco classes de problemas: Criptografia/Codificação, Engenharia Reversa, Forense, *Web* e Miscelânea.

A Tabela 2 resume a pesquisa realizada. Os problemas da área de Criptografia/Codificação somaram 306, enquanto 235 problemas envolviam Engenharia Reversa. 385 problemas aplicavam técnicas da área Forense, a classe que apresentou mais problemas na análise realizada. 264 problemas foram classificados como sendo de *Web* e 153, que não se enquadravam nas classes anteriores, foram classificados como miscelâneos.

Tabela 2: Levantamento de quantidade de problemas por classe nos desafios analisados.

Classe	Quantidade de Problemas
Criptografia/Codificação	306
Engenharia Reversa	235
Forense	385
Web	264
Miscelânea	153

No fim de 2017 (após a realização deste levantamento, portanto), foi publicado um trabalho com pesquisa semelhante, mapeando os desafios publicados entre 2015 e 2017, categorizando os problemas em seis áreas: criptografia, *web*, engenharia reversa, forense, *pwn* (exploração de vulnerabilidades) e miscelânea [2]. A proporção, no entanto, ficou próxima à deste trabalho se as classes *pwn* e engenharia reversa forem unidas.

Ao todo, 1250 problemas foram analisados. As técni-

cas envolvidas nos problemas foram contabilizadas e documentadas, de forma a tornar possível sua rastreabilidade e identificação de quais competições elas foram usadas. Cabe ressaltar que, dentre os 1250 problemas, 86 (6,9%) eram compostos, isto é, formados pela junção de duas ou mais técnicas. Por isso, ao somar a quantidade de problemas por classe na Tabela 2, o valor resultante é superior a 1250. Portanto, problemas compostos que envolvem mais do que uma classe de problemas são contabilizados mais de uma vez.

Para que o formato de competição modelado fosse representativo, analisou-se a característica de linearidade (Seção 2) para cada competição. Essa característica pode ser identificada em 48 competições, sendo a forma não linear vista em 46 destas (95,8% do total). A forma linear e o formato misto foram encontrados em uma competição cada um.

A análise dos desafios revelou os tipos de problemas mais frequentes em competições, bem como as características mais comuns das próprias competições. Essas informações foram particularmente úteis para comparar os trabalhos existentes (como apresentado na Seção 3) e guiar o desenvolvimento de um modelo de competição e de geradores automáticos de problemas, como será discutido na Seção 5.

5. DESENVOLVIMENTO DO TREASUREHUNT

Nesta seção, discute-se o desenvolvimento da ferramenta TreasureHunt. O texto está organizado da seguinte maneira: a Seção 5.1 descreve a modelagem das competições geradas pela ferramenta. A Seção 5.2 explica como foi realizada a seleção e como se dá a composição das técnicas na ferramenta desenvolvida. A Seção 5.3 descreve a implementação e o funcionamento dos componentes do TreasureHunt, e a Seção 5.4 cita as ferramentas necessárias para jogadores e organizadores utilizarem o TreasureHunt e participarem de desafios por ele gerados.

5.1 Modelagem da Competição

Conforme discutido na Seção 2, jogos e competições são um meio eficaz de desenvolver habilidades em Cibersegurança. Nesse contexto, jogos do tipo desafio são particularmente interessantes por serem flexíveis na complexidade de problemas e conhecimentos específicos exercitados. No entanto, organizar estas competições exige conhecimento técnico. Há também as dificuldades com elaboração e reaproveitamento de problemas, pois estes perdem o fator surpresa após a primeira aplicação e suas respostas podem ser compartilhadas. Para contornar essas dificuldades, a ferramenta TreasureHunt propõe automatizar a geração dos problemas que compõem um desafio, usando de aleatorização para gerar instâncias únicas para os problemas.

Esta Seção apresenta a proposta adotada para competições com geração automatizada de desafios de Segurança. A Tabela 3 resume os parâmetros gerais das competições modeladas pelo TreasureHunt.

A proposta do TreasureHunt é gerar competições do tipo desafio, voltadas para o âmbito educacional, como forma de incentivar alunos de cursos de Computação a se interessarem pela área de Segurança. Assim, a aplicação de uma competição gerada pelo TreasureHunt é útil tanto para os que desejam de fato competir e vencer a competição, quanto para os que desejam progredir aprimorando seus conhecimentos práticos na área.

Tabela 3: Parâmetros gerais.

Parâmetro	Valor
Tipo de competição	Desafio
Linearidade	Não linear
Classificação	Ranqueamento
Pontuação	1 ponto por problema
Desempate	Horário do último acerto
Formato da *flag*	`TreasureHunt{texto-aleatorio}`

Em desafios, cada problema costuma ser associado a uma pontuação de acordo com a sua complexidade ou com a fase em que ele se encontra, em caso de jogos lineares ou mistos. Seguindo a tendência da maioria dos jogos para os quais esta informação estava disponível, conforme exposto na Seção 4, o jogo foi projetado para ser não linear, isto é, ele contém um conjunto independente de problemas que podem ser resolvidos em qualquer ordem escolhida pelo jogador. A ferramenta proposta considera ainda que todos os problemas têm o mesmo peso, eliminando a necessidade de contagem de pontos e efetuando apenas a soma de acertos de cada jogador. A variável tempo é considerada somente em casos de desempate, ou seja, quando há empate no número de acertos, fica na frente aquele que submeteu a última resposta correta primeiro. As respostas incorretas são desprezadas para fins de ranqueamento e ficam armazenadas em tabelas do SGBD (Sistema de Gerenciamento de Banco de Dados) apenas para identificar possíveis falhas, tanto nos problemas gerados quanto na comunicação das regras do jogo, e tentativas de compartilhamento de *flag*.

Em competições do tipo desafio, a palavra secreta pode ser um texto aleatório ou pode seguir um padrão, sendo formada por uma sequência de símbolos que não deixe dúvidas de que é a resposta. Optou-se por adotar a segunda prática, fazendo com que as *flags* estejam no formato `TreasureHunt{texto-aleatorio}`. Assim, minimiza-se a chance de que um competidor acredite erroneamente que um texto qualquer encontrado na tentativa de resolução do problema seja a resposta. Esta informação está disponível nas instruções do jogo, como consta na Seção 5.3.2.

5.2 Seleção e Composição de Técnicas

A partir da análise citada na Seção 4, obteve-se um rol de cerca de 200 técnicas distintas encontradas nas competições constantes no repositório. Destas, as 40 técnicas mais frequentes foram identificadas e nove delas foram escolhidas para a construção de um dos módulos do TreasureHunt, a ferramenta geradora de desafios automáticos e compostos.

Os principais critérios utilizados na seleção de técnicas foram o perfil dos jogadores para os quais as competições seriam aplicadas e o objetivo do trabalho. Embora o perfil dos jogadores não fosse conhecido, partiu-se da premissa de que estes seriam estudantes sem conhecimentos avançados em Linux e Segurança Computacional [16].

As técnicas implementadas na ferramenta compositora de problemas são:

- **Comentário em código-fonte de página HTML (*HyperText Markup Language*)**: problema em que a *flag* é inserida arbitrariamente em alguma posição de

um arquivo HTML. A página carrega estilos e imagens diferentes em cada instância. Este exercício faz parte da classe Miscelânea e realiza análise de código-fonte. Pode ser resolvido, entre outras formas, visualmente ou por meio de comandos (inclusive programados).

- **Comentário no arquivo `robots.txt`**: problema que contém um conjunto de arquivos de um *website* válido e, entre eles, um arquivo `robots.txt`[4] que contém uma quantidade parametrizável de comentários. Todos os comentários são sequências aleatórias de caracteres, exceto um, que corresponde à *flag*. Este exercício faz parte da classe Miscelânea e realiza análise de código-fonte. Pode ser resolvido, entre outras formas, visualmente ou por meio de comandos (inclusive programados).

- **(De)codificação de arquivo em base64**: problema em que a *flag* é inserida em um arquivo de texto sorteado arbitrariamente em um diretório parametrizável. Após isso o arquivo é codificado em formato base64[5]. Este exercício faz parte da classe Criptografia/Codificação e pode ser resolvido, entre outras formas, com o uso de um decodificador base64.

- **(De)codificação de arquivo em base32**: problema análogo ao anterior, mas utilizando (de)codificação em base32[6].

- **(Des)criptografia de Cifra de César**: problema em que a *flag* é inserida um arquivo de texto sorteado arbitrariamente em um diretório parametrizável. Após isso o arquivo é criptografado com a Cifra de César[7], com chave aleatoriamente escolhida entre 1 e 25, garantindo que o texto cifrado não será igual ao original. Este exercício faz parte da classe Criptografia/Codificação e pode ser resolvido, entre outras formas, manualmente ou com uma ferramenta decodificadora.

- **(De)codificação de caractere ASCII (*American Standard Code for Information Interchange*) para inteiro**: problema em que a *flag* é inserida em um arquivo de texto sorteado arbitrariamente em um diretório parametrizável. Após isso o arquivo é codificado em valores inteiros, que representam os caracteres originais com base na tabela ASCII, separados por espaço em branco. Este exercício faz parte da classe Codificação e pode ser resolvido, entre outras formas, manualmente ou com uma ferramenta decodificadora.

- **Descompilar binário e obter fonte Java**: problema em que um código Java, que produz um texto arbitrário, é criado. Neste código é inserida uma *flag* em uma variável e o código é compilado para *bytecode*, gerando um arquivo `.class`. Este exercício faz parte da classe Engenharia Reversa, e pode ser resolvido, entre outras formas, por meio de comandos de impressão de caracteres ou ferramentas de descompilação.

- **Descompilar binário e obter fonte Python**: semelhante ao problema anterior, contém um código em Python que imprime um texto arbitrário. Neste código é inserida uma *flag* em uma variável e o código é compilado para *bytecode*, gerando um arquivo `.pyc`. Também faz parte da classe Engenharia Reversa, e pode ser resolvido, entre outras formas, utilizando comandos de impressão de caracteres ou ferramentas de descompilação.

- **Esteganografia em imagens**: problema em que a *flag* é esteganografada[8] em uma imagem escolhida arbitrariamente em um diretório parametrizável. Este exercício faz parte da classe Forense e pode ser resolvido por meio da ferramenta padrão de esteganografia utilizada no gerador de desafios (`outguess`). A versão atual permite utilizar esteganografia com ou sem senha (parâmetro `-k` no `outguess`). Por padrão, a senha é uma informação simples: o ID de cada jogador.

A Figura 1 apresenta duas instâncias distintas do problema *Comentário em código-fonte de página HTML* produzidas pelo gerador de desafios, e seus respectivos códigos-fonte. Nota-se que a *flag* aparece em linhas distintas (linha 40 na instância da esquerda e linha 36 na instância da direita), bem como não é composta pela mesma sequência de símbolos. Optou-se por *flags* de tamanho igual para problemas de mesma técnica. As imagens exibidas e o estilo das páginas são aleatórios, de forma que cada usuário receba um arquivo com configurações diferentes, ou seja, instâncias distintas do mesmo problema.

Em alguns casos, tal como ocorre com o problema *Descompilar binário e obter fonte Java*, o usuário pode chegar à solução de formas distintas. Algumas dessas formas envolvem o uso de ferramentas de descompilação e análise do código, bem como integração de uma ferramenta destas com comandos de filtragem de linhas. Outra forma é através do comando `strings`. A Figura 2 exibe a solução de quatro instâncias deste problema utilizando a ferramenta `strings`.

Esta solução deixa claro que, por mais que as *flags* sejam distintas, os mesmos comandos podem ser usados para se chegar à resposta. Portanto, compartilhar a resposta (a *flag*) não surte efeito para os jogadores, mas compartilhar os comandos sim. No contexto educacional para o qual a ferramenta foi proposta, esse compartilhamento de comandos, se ocorrer, pode ser visto de forma positiva, pois permite que os jogadores obtenham as respostas e assimilem o funcionamento das ferramentas ora em uso.

O gerador de desafios também é capaz de compor técnicas. A composição de técnicas consiste em aplicar técnicas de forma encadeada. Por exemplo, um problema que componha *(De)codificação de arquivo em base64* e *(Des)criptografia de Cifra de César* codificará um arquivo em base64, gerando

[4] `robots.txt` é um arquivo de configuração de indexação de conteúdo na *web*. É ele quem determina os diretórios e arquivos cujo acesso é permitido ou proibido aos robôs de busca [20].

[5] base64 é um esquema de codificação que representa dados binários em um formato de *string* ASCII [14], utilizando um conjunto de 64 caracteres.

[6] base32 é um esquema de codificação semelhante ao esquema base64, mas que utiliza um conjunto de 32 caracteres para ser representado (Josefsson, 2006).

[7] Cifra de César é uma cifra de substituição baseada na troca de cada caractere do texto original por outro n posições à frente no alfabeto, de forma circular, sendo n a chave [11].

[8] Técnica que consiste em esconder mensagens utilizando um meio de cobertura, como uma imagem ou um arquivo de áudio [13].

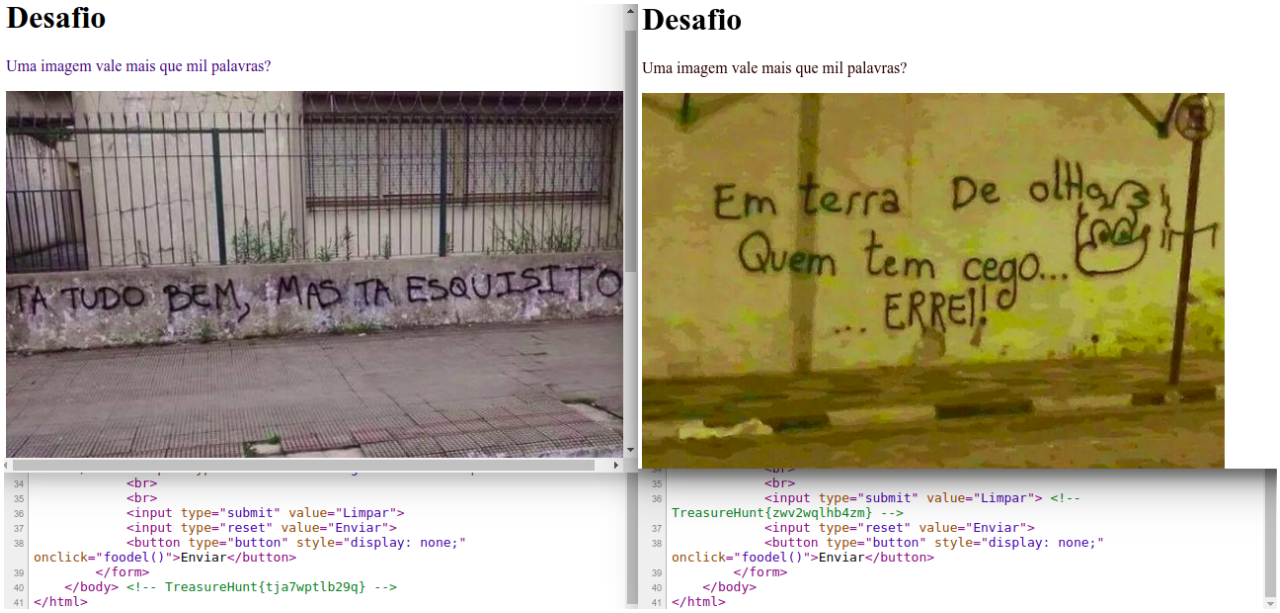

Figura 1: Duas instâncias do problema *Comentário em código-fonte de página HTML* e seus respectivos códigos-fonte.

Figura 2: Solução alternativa de quatro instâncias do problema *Descompilar binário e obter fonte Java*.

um arquivo de saída intermediário e temporário. Neste arquivo será aplicada a Cifra de César, gerando um novo arquivo de saída que compõe as duas técnicas. Este arquivo — e somente este, nesse caso — é disponibilizado ao jogador. Assim, para solucionar o exercício, o jogador precisa encontrar a chave utilizada na Cifra de César, obter o arquivo codificado em base64, decodificá-lo e procurar pela *flag*.

Há, porém, técnicas que não podem ser compostas com outras. A Tabela 4 mostra as técnicas que podem ser compostas com a marcação de um "✓" que, quando presente na linha i e coluna j, significa possibilidade de composição entre a técnica da linha i e a técnica da coluna j. Nela é possível ver, por exemplo, que as composições (*Esteganografia em Imagens* ∘ *Esteganografia em Imagens*) e (*Codificação em base64* ∘ *Cifra de César*) são possíveis, ao passo que a composição (*Cifra de César* ∘ *Cifra de César*) não é, já que a aplicação da cifra várias vezes terá sempre uma chave equivalente a sua aplicação uma única vez.

É importante citar que a composição não é uma operação comutativa, ou seja, a ordem de aplicação das técnicas influencia na instância gerada e produz exercícios diferentes. Por exemplo, aplicar a codificação base64 antes de um problema de descompilação de código Java implica gerar uma *flag*, codificá-la em base64 e depois inseri-la no código a ser compilado. Caso a ordem inversa seja aplicada, uma *flag* é inserida no código-fonte e, após a compilação, o arquivo de *bytecode* é codificado em base64. Portanto, (*(De)codificação de arquivo em base64* ∘ *Descompilar binário e obter fonte Java*) ≠ (*Descompilar binário e obter fonte Java* ∘ *(De)codificação de arquivo em base64*).

As Figuras 3 e 4 apresentam instâncias do problema que compõe *Esteganografia em imagens* e *(De)codificação de arquivo em base64*. Neste problema, o jogador recebe um arquivo com nome parametrizável, mas definido como `saida.out` por padrão. O arquivo está codificado em base64 e, portanto, deve ser decodificado. Ao proceder à decodificação, o arquivo resultante, exibido como `imagem.jpg`, representa uma imagem JPEG (*Joint Photographic Experts Group*) que contém um texto esteganografado, sem senha, com a ferramenta `outguess`. Utilizando o `outguess` é possível extrair o conteúdo da imagem e assim obter a *flag*. As imagens selecionadas para cada instância são diferentes, bem como as *flags* geradas, produzindo arquivos diferentes quando codificados em base64.

5.3 Implementação

O código-fonte do TreasureHunt está disponível publicamente em um repositório[9] *online*, sob Licença Creative Commons – Atribuição-Não Comercial 4.0 Internacional. O código contém dois componentes principais, o gerador de desa-

[9] `https://github.com/TreasureHuntGame/TreasureHunt`

Tabela 4: Matriz de composições.

Técnicas	HTML	Robots	Base64	Base32	Cesar	A2I	Java	Python	Esteg
HTML			✓	✓	✓	✓			✓
Robots			✓	✓	✓	✓			✓
Base64	✓	✓	✓	✓	✓	✓	✓	✓	✓
Base32	✓	✓	✓	✓	✓	✓	✓	✓	✓
César	✓	✓	✓	✓			✓	✓	✓
A2I	✓	✓	✓	✓		✓	✓	✓	✓
Java			✓	✓	✓	✓			✓
Python			✓	✓	✓	✓			✓
Esteg			✓	✓	✓	✓			✓

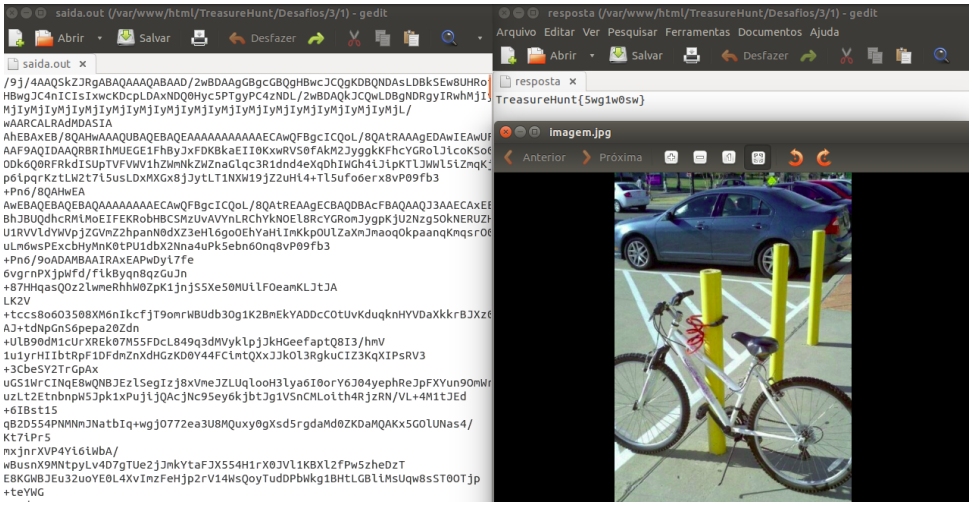

Figura 3: Primeira instância do problema composto (*Esteganografia em imagens* ∘ *(De)codificação de arquivo em* `base64`).

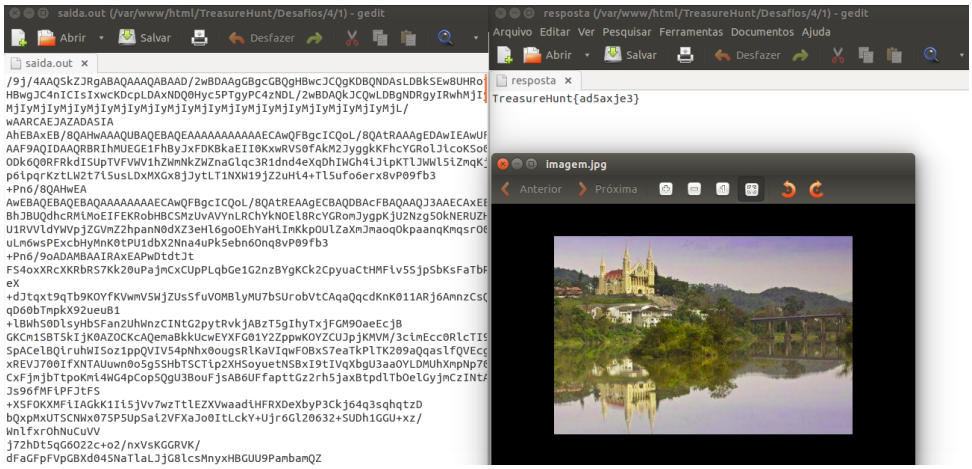

Figura 4: Segunda instância do problema composto (*Esteganografia em imagens* ∘ *(De)codificação de arquivo em* `base64`).

fios, usado pelo organizador de uma competição, e o sistema *web*, usado pelos competidores. Esses componentes são discutidos nas Seções 5.3.1 e 5.3.2, respectivamente.

5.3.1 Gerador de Desafios

A Figura 5 mostra o Diagrama de Atividades para a geração de uma competição. Pelo diagrama, percebe-se que as ações do organizador dizem respeito apenas à escolha dos parâmetros; toda complexidade técnica associada é realizada automaticamente pelo gerador, que faz as chamadas ao SGBD e ao servidor *web*. Além disso, cabe ao gerador a tarefa de criar as instâncias diferentes para cada problema, não sendo necessária qualquer ação do organizador neste sentido. O organizador não consegue fazer com que a ferramenta gere instâncias iguais para todos os jogadores.

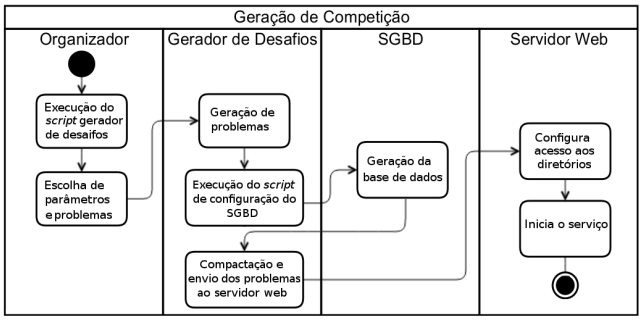

Figura 5: Diagrama de atividades para gerar uma competição.

O gerador de desafios executa em Linux, e é composto por um conjunto de *scripts* e diretórios com imagens e textos. Nestes *scripts* estão os parâmetros, tais como diretórios, nomes de arquivos, quantidade de *flags*, funções de validação de entrada, de exibição de menu, de composição de técnicas e de geração de problemas individuais. Os diretórios de imagens e textos são utilizados por problemas específicos em que o gerador sorteia os arquivos que farão parte da instância de cada jogador. Há um conjunto de arquivos padrão enviado junto com o gerador, mas o organizador pode manipular estes diretórios, inserindo ou removendo arquivos a seu critério.

O *script* principal (`jogo.sh`) solicita as informações necessárias ao organizador, que são quantidade de problemas e jogadores. A partir disso, solicita os códigos dos problemas de acordo com numeração exibida pelo *script* (Figura 6) e gera os problemas. A validação impede que problemas inexistentes sejam solicitados.

Para gerar as instâncias aleatórias, utilizou-se a ferramenta `shuf` e o arquivo `/dev/urandom`. A ferramenta `shuf` gera permutações aleatórias e foi utilizada para o sorteio de arquivos que fariam parte das instâncias de cada problema. O arquivo `/dev/urandom` cria combinações pseudoaleatórias e foi utilizado para gerar o texto contido nas palavras secretas (*flags*).

O gerador de desafios proposto neste trabalho considera o número de pontos e o horário da última submissão correta ao ranquear os jogadores. Conforme já exposto, a versão atual da ferramenta considera que cada problema vale um ponto. Futuramente, a proposta do trabalho é permitir a parametrização destes valores, de forma que o organizador — provavelmente um professor ou um treinador responsável pela elaboração e aplicação da atividade — os arbitre, assim sendo possível, portanto, manter todos os problemas com pesos progressivos ou até mesmo nulos, caso este fator não seja considerado importante. A variável tempo é considerada somente em casos de desempate, o que significa dizer que quando há empate no número de acertos, o horário da última submissão correta desempata em ordem decrescente, ficando à frente aquele que submeteu primeiro. Mesmo que o tempo não esteja sendo utilizado para fins de desempate, a última submissão correta sempre aparecerá no placar, como apresentado na Figura 9.

A Figura 6 mostra o funcionamento do *script* principal. Ao escolher uma técnica, o usuário deve apenas informar o identificador numérico dela (exemplo: `2` para *(Des)criptografia de Cifra de César*). Caso a opção seja por um problema composto, o usuário deve informar os dois códigos das técnicas na ordem em que quer realizar a composição. Por exemplo, para a composição (*(De)codificação de arquivo em base64* ∘ *(Des)criptografia de Cifra de César*), o código informado deve ser `1 2`, com espaço separando o identificador das técnicas.

```
----------
Treasure Hunt!
----------
Informe a quantidade de DESAFIOS: 8
Informe a quantidade de JOGADORES: 10
----------
Vamos criar os desafios!
----------
Lista de problemas disponíveis:
1: (De)codificação de arquivo em base64
2: (Des)criptografia de Cifra de César
3: Comentário em código-fonte de página HTML
4: Comentário no arquivo robots.txt
5: (De)codificação de caractere ASCII para inteiro
6: Descompilar binário e obter fonte Java
7: Descompilar binário e obter fonte Python
8: Esteganografia em imagens
9: (De)codificação de arquivo em base32
Obs.: escolha 1 ou 2 problemas. Exibiremos uma mensagem de erro se a composição
não existir.
----------
Informe o(s) problema(s) do desafio 1:
```

Figura 6: Execução do *script* `jogo.sh`.

O programa ainda gera um arquivo com as respostas e permite a geração de cópias das instâncias dos usuários. Por padrão, ao concluir a geração de desafios, o conjunto de problemas de cada usuário é compactado em um arquivo ZIP e enviado para o servidor *web*. Para esta etapa, pressupõe-se que o servidor *web* esteja no mesmo *host* em que o *script* é executado, e que há permissão para enviar os arquivos ZIP gerados para o diretório dos desafios no servidor (por padrão, `/var/www/html/TreasureHunt/Desafios`). Após esta etapa o jogo executa o *script* `ConfiguraBD.sh` para proceder à criação da base de dados `TreasureHunt` (nome padrão), das tabelas e inserção de registros de usuário e respostas dos problemas no SGBD.

O conteúdo sensível do Banco de Dados inclui senhas e respostas, e foi armazenado utilizando *hash* e *salt* aleatório para as senhas e *hash* para as respostas. O algoritmo de *hash* escolhido foi o SHA-256 (*Secure Hash Algorithm 256 bits*) [21].

Após a configuração do SGBD, os exercícios estão prontos e a competição já pode ser iniciada, assumindo que o servidor *web* já está em funcionamento. Se o servidor *web* não estiver em execução, o organizador deverá fazê-lo manualmente ou descomentar a instrução que realiza este serviço no *script* principal da ferramenta (`jogo.sh`). Se os jogadores estiverem na rede local, podem acessar o jogo pelo navegador *web*, informando nome ou endereço do servidor. O

acesso externo não é coberto pela ferramenta, mas também é possível se o organizador configurar o servidor e os demais componentes (*firewalls*, roteadores, etc.) ou utilizar alguma ferramenta tal como o `ngrok`[10], que disponibiliza aplicações locais na *web* por meio de túneis seguros. Por ser uma etapa opcional e não contemplada pela ferramenta, o diagrama da Figura 5 ignora a etapa de disponibilização externa do jogo.

5.3.2 Sistema Web

O sistema *web* é o componente através do qual os jogadores fazem interação com o jogo. As funcionalidades da aplicação *web* do ponto de vista do usuário podem ser vistas no Diagrama de Casos de Uso da Figura 7. Embora todos os problemas atuais possam ser resolvidos de maneira *offline*, o sistema *web* é responsável por

- apresentar instruções sobre o jogo e o contato dos desenvolvedores;
- manter os arquivos de problemas para cada jogador;
- receber e validar submissões de respostas;
- exibir o placar individual detalhado;
- exibir o placar geral.

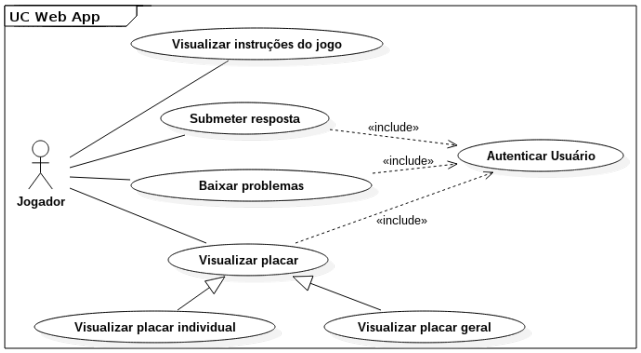

Figura 7: Diagrama de casos de uso da aplicação web.

Somente as instruções e as informações de contato podem ser visualizadas sem autenticação, assim mantendo os arquivos de cada jogador e as informações sobre seus exercícios resolvidos e não resolvidos de forma privada. Na Figura 7 é possível perceber isso, pois o cenário de uso *Visualizar instruções do jogo* não contém o estereótipo `<<include>>` para o cenário de uso *Autenticar Usuário*.

Cada jogador recebe um identificador (ID) individual numérico e uma senha, que são usados para se autenticar no sistema *web*. Ao realizar a autenticação, o usuário tem acesso a um arquivo compactado contendo todas as suas instâncias de problemas. Esse arquivo contém diretórios cujo nome é o identificador numérico de um problema, e que contêm o(s) arquivo(s) que constituem o problema. Por exemplo, o diretório `4` contém os arquivos para o problema com ID 4. Todos os jogadores recebem instâncias dos mesmos problemas, seguindo a mesma ordem, com *flags* geradas aleatoriamente, que são únicas para cada jogador. As ferramentas necessárias para a resolução dos problemas não são enviadas no arquivo compactado, sendo desejável que já estejam instaladas nas máquinas dos jogadores, embora esta decisão fique a critério do organizador da competição.

Para submeter uma *flag*, basta ao jogador informar o ID do problema e a palavra secreta descoberta (Figura 8). Ao realizar a submissão, o usuário é imediatamente informado sobre o resultado. A resposta retornada pelo sistema será uma das seguintes:

1. Problema com ID inválido!
2. Errou!
3. Errou! Considere submeter a *flag* no seguinte formato: `TreasureHunt{texto-aleatorio}`
4. Você já acertou a questão `ID_problema`!
5. Acertou! `n/m`

O caso 1 ocorre quando o usuário informa um identificador de problema inválido, tal como um número negativo ou superior ao número do último diretório contido no seu arquivo compactado. O caso 2 ocorre quando o usuário insere uma resposta no formato correto, mas contendo o texto aleatório incorreto. O caso 3 ocorre quando o usuário informa uma resposta em formato diferente do padrão. O caso 4 ocorre quando o usuário tenta ressubmeter uma resposta já acertada, o que evita submissões e inserções de registros desnecessárias. O caso 5 ocorre quando o usuário acerta a questão, e informa quantos acertos ele já obteve em relação ao total de exercícios.

Na parte inferior da página de submissão (Figura 8) é exibido o placar individual detalhado do jogador, que mostra todos os identificadores de problemas seguidos por seu *status* (*Resolvido* ou *Não Resolvido*) e o número de tentativas. O jogador também pode passar o cursor sobre a aba *Placar* para ver sua colocação durante a competição. A Figura 9 mostra que o Placar Geral exibe a colocação, o ID do jogador, o número de acertos e o horário da última submissão correta, usado em casos de desempate.

A cada autenticação o sistema registra a hora e o endereço IP do jogador. Esta funcionalidade fornece mais elementos ao organizador quanto ao compartilhamento de respostas e à troca de IDs ou acesso de mais de um jogador com o mesmo ID. Além disso, permite calcular com maior precisão o tempo despendido pelo jogador na atividade.

O sistema foi desenvolvido considerando boas práticas de segurança, com soluções de configuração do servidor *web* que definem o máximo de clientes e redirecionamentos para páginas de erro e para a página principal. Esta configuração deve ser realizada manualmente pelo organizador, razão pela qual um modelo de arquivo de configuração do servidor *web* Apache está disponível no repositório do jogo.

5.4 Ferramentas para Jogadores e Organizadores

Para solucionar os desafios incluídos no TreasureHunt, sugere-se, no mínimo, o uso das seguintes ferramentas:

- `base32`
- `base64`
- `caesar`

[10] `https://ngrok.com/`

Figura 8: Tela de submissão de *flag* do sistema *web*.

Figura 9: Placar geral de uma competição no sistema *web*.

- `outguess`
- `sed`
- `sh`
- `strings`
- Editor de texto
- Navegador *web*

Cabe ressaltar que é possível solucionar problemas propostos com outras ferramentas além das supracitadas. Um exemplo disso seria um problema hipotético de encontrar uma *flag* em um arquivo em texto claro. Neste caso, o uso de um editor de texto seria suficiente, mas soluções alternativas combinando os comandos `cat` e `grep`, e possivelmente automatizando esta rotina com algum compilador ou interpretador de comandos, poderiam atingir a mesma resposta.

As listas de ferramentas mínimas necessárias para configurar e utilizar o TreasureHunt estão disponíveis no repositório do trabalho. Com o objetivo de minimizar a complexidade de instalação e facilitar o uso da ferramenta, também está disponível no repositório um *script* (`instalador.sh`) responsável por instalar as ferramentas necessárias para que o organizador interessado possa utilizar o TreasureHunt. Este *script* instala todos os pacotes listados no arquivo `requisitos.txt`, exigindo a ação do usuário somente para concordar com a licença de uso do Oracle JDK, já que o jogo utiliza a ferramenta `javac` para problemas que envolvem a técnica *Descompilar binário e obter fonte Java*. O *script* também fornece um arquivo de registro (*log*) que permite rastrear o (in)sucesso na instalação de cada pacote.

6. APLICAÇÃO

Desde a conclusão da primeira versão, em 2017, competições geradas pelo TreasureHunt vêm sendo aplicadas anualmente no Instituto Federal Catarinense (IFC) – Campus Blumenau e na Universidade do Estado de Santa Catarina (UDESC) – Campus Joinville. No IFC, houve aplicação de competições em quatro turmas:

- Curso de Qualificação Profissional (CQP) em Configuração de Servidores Linux, no contexto de um módulo de ensino de Segurança, em 2017;
- Curso de Tecnologia em Análise e Desenvolvimento de Sistemas, na disciplina de Segurança Computacional, em 2017 e em 2018;
- Curso de Tecnologia em Análise e Desenvolvimento de Sistemas, na Semana Acadêmica do curso, em 2018.

Na UDESC, competições foram realizadas em 2017 e 2018 para estudantes de bacharelado em Ciência da Computação, nas aulas de Segurança de Redes. O público participante não era restrito aos discentes matriculados na disciplina.

Em todas as competições houve um momento prévio de explanação sobre competições de Cibersegurança. Neste momento, o foco foram os jogos de caça ao tesouro. Após, um conjunto de ferramentas de Segurança para Linux era apresentado, conjunto este suficiente para solucionar os problemas das competições. A explicação de cada ferramenta antecedia exemplos práticos e exercícios, corrigidos posteriormente.

Embora as ferramentas de Segurança apresentadas para resolver os exercícios e o próprio TreasureHunt tenham sido desenvolvidos para Linux, os jogadores podem utilizar qualquer outro sistema operacional e obter êxito nos exercícios. O foco da aula preparatória para ferramentas Linux se deve ao fato de que as instituições nas quais o experimento foi realizado utilizam o sistema operacional Ubuntu. Além disso, ao elaborar o TreasureHunt, partiu-se da premissa de que os participantes da atividade seriam estudantes com conhecimento razoável em Linux e pouco conhecimento em Segurança, embora exercícios mais complexos e voltados a ferramentas de outros sistemas operacionais também possam ser desenvolvidos e estar presentes em futuras atualizações do TreasureHunt.

6.1 Questionários

Antes e depois das competições, questionários[11] foram disponibilizados para os jogadores que se sentissem interessados em contribuir com informações sobre seu perfil e sobre os sentimentos para com a atividade. Percebe-se que, das 54 respostas computadas sobre a familiaridade com o sistema Linux, 23 (42,59%) responderam *Um pouco familiar. Uso principalmente a interface gráfica, mas sei usar alguns comandos de terminal para manipular arquivos e/ou compilar e executar programas*, sendo esta a alternativa mais assinalada. Esta resposta corrobora com o que se previu ao projetar a ferramenta. No que diz respeito à experiência em Segurança Computacional, 37 (68,52%) responderam não ter experiência na área, de um total de 56 respostas computadas. Nesta questão era possível marcar mais de uma alternativa, tais como *Estudei por conta própria* ou *Fiz um curso em outra instituição*.

Em questões aplicadas antes e depois das competições, utilizou-se escala de Likert [19] de 5 pontos, com respostas variando de *Discordo fortemente* (1) a *Concordo fortemente* (5), e calculou-se a média obtida nas respostas. De um total de 46 respondentes, *Jogos e competições me deixam mais motivado a aprender do que aulas expositivas* obteve média 4,41. Na questão *Tenho interesse em atividades práticas en-volvendo Segurança Computacional*, a média foi de 4,46. A questão *Exercícios práticos de Segurança Computacional aumentam o entendimento sobre esta área* resultou em média de 4,74 pontos, e a questão *Entendo que competições de Segurança Computacional aumentam o apelo desta área para o público geral* obteve média de 4,24 pontos. As respostas obtidas são indícios de que a atividade foi positiva para os participantes, com resultados entre *Concordo* e *Concordo fortemente*.

Cabe ressaltar que, para a etapa de avaliação do trabalho, foram analisados o Modelo de Aceitação de Tecnologia (*Technology Acceptance Model* — TAM) [9] e o Modelo de Kirkpatrick [15]. Considerando que as etapas do Modelo de Kirkpatrick tinham maior relação com o trabalho, este modelo foi escolhido como inspiração. Os questionários foram inspirados em trabalhos em que a impressão de jogadores foi avaliada antes e depois da participação em jogos de Segurança (Cheung et al., 2012; Petullo et al., 2016). Muitos resultados de questionários foram suprimidos deste trabalho devido ao foco ser no relato da construção da ferramenta. Uma análise mais completa do método de avaliação, dos resultados de desempenho dos discentes das três primeiras

[11] Disponíveis em `http://bit.ly/2AdEWMW`, `http://bit.ly/2AoVkew` e `http://bit.ly/2lYORDL`.

turmas em que as competições foram aplicadas, no ano de 2017, e das perguntas e respostas dos questionários está disponível em [16].

7. CONCLUSÃO

Jogos e competições são considerados uma forma atraente de despertar o interesse por Segurança Computacional. As competições do tipo desafio estão entre as mais usuais, tendo como principais vantagens a flexibilidade em relação à infraestrutura computacional e em adaptar-se a públicos com vários níveis de conhecimento. Por outro lado, fatores como o compartilhamento de respostas e a dificuldade de reaproveitamento de problemas representam obstáculos à popularização dessas competições.

Este trabalho apresenta um levantamento sobre desafios de Segurança que identifica os tipos de problemas mais frequentes e as características mais usuais dessas competições. O conhecimento extraído desse levantamento foi aplicado na criação do TreasureHunt, uma ferramenta geradora de desafios para o ensino de Cibersegurança. Técnicas aplicadas em desafios foram selecionadas e implementadas, trazendo como contribuição a formação de problemas exclusivos para cada jogador.

A ferramenta proposta e descrita neste trabalho cria instâncias equivalentes em dificuldade e distintas não somente na *flag* escondida. Além disso, este trabalho abstrai parte da complexidade de instalação fornecendo um *script* que automatiza esta tarefa.

O trabalho será continuado com a criação de novos problemas no gerador de desafios. Em especial, deseja-se criar problemas da classe *web*, cobrindo assim todas as classes identificadas, e permitir que mais de duas técnicas componham um problema. Há ainda a intenção de parametrizar a pontuação dos exercícios, usar mais ferramentas por técnica (por exemplo, esteganografia com `outguess` e `f5.jar`) e implementar um recurso para inserção de dicas. Pretende-se ainda incluir identificadores às competições, o que permitirá observar a evolução de jogadores com o passar do tempo e isolar jogadores de turmas diferentes mais facilmente para fins de análise de dados.

Uma outra ideia é fornecer uma interface de treinamento inteligente o suficiente para aprender o nível de experiência do usuário com base nas suas respostas. A ideia geral seria fornecer exercícios dinamicamente e, com base no tempo despendido pelo usuário na solução da tarefa, inferir a complexidade dos exercícios para fornecer o próximo problema com dificuldade compatível ao conhecimento do jogador.

8. REFERÊNCIAS

[1] J. Burket, P. Chapman, T. Becker, C. Ganas, and D. Brumley. Automatic problem generation for capture-the-flag competitions. In *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. USENIX Association, 2015.

[2] T. J. Burns, S. C. Rios, T. K. Jordan, Q. Gu, and T. Underwood. Analysis and exercises for engaging beginners in online ctf competitions for security education. In *2017 {USENIX} Workshop on Advances in Security Education ({ASE} 17)*. USENIX Association, 2017.

[3] E. Capuano. #jolt hackathon 2017, 2017. [Online]. Disponível em: `https://blog.ecapuano.com/jolthackathon-2017/`.

[4] P. Chapman, J. Burket, and D. Brumley. Picoctf: A game-based computer security competition for high school students. In *3GSE*, 2014.

[5] R. S. Cheung, J. P. Cohen, H. Z. Lo, F. Elia, and V. Carrillo-Marquez. Effectiveness of cybersecurity competitions. In *Proceedings of the International Conference on Security and Management (SAM)*, page 1, 2012.

[6] A. Conklin. Cyber defense competitions and information security education: An active learning solution for a capstone course. In *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*, volume 9, pages 220b–220b. IEEE, 2006.

[7] W. Crumpler and J. A. Lewis. The cybersecurity workforce gap. *Center for Strategic and International Studies, Washington, DC.*, 2019. [Online]. Disponível em: `https://www.csis.org/analysis/cybersecurityworkforce-gap`.

[8] CTFs. Ctf write-ups repository., 2017. [Online]. Disponível em: `https://github.com/ctfs`.

[9] F. D. Davis. *A technology acceptance model for empirically testing new end-user information systems: Theory and results*. PhD thesis, Massachusetts Institute of Technology, 1985.

[10] W. Feng. A scaffolded, metamorphic ctf for reverse engineering. In *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. USENIX Association, 2015.

[11] K. Goyal and S. Kinger. Modified caesar cipher for better security enhancement. *International Journal of Computer Applications*, 73(3), 2013.

[12] P. Hulin, A. Davis, R. Sridhar, A. Fasano, C. Gallagher, A. Sedlacek, T. Leek, and B. Dolan-Gavitt. Autoctf: Creating diverse pwnables via automated bug injection. In *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*, 2017.

[13] M. Hussain, A. W. A. Wahab, N. B. Anuar, R. Salleh, and R. M. Noor. Pixel value differencing steganography techniques: Analysis and open challenge. In *Consumer Electronics-Taiwan (ICCE-TW), 2015 IEEE International Conference on*, pages 21–22. IEEE, 2015.

[14] S. Josefsson. Rfc4648: The base16, base32, and base64 data encodings. *Internet Engineering Task Force, Tech. Rep*, 2006.

[15] D. Kirkpatrick. Revisiting kirkpatrick's four-level model. *Training and development*, 50(1):54–59, 1996.

[16] R. R. Ladeira. Treasurehunt: Geração automática de desafios aplicados no ensino de segurança computacional. Dissertação de mestrado, Universidade do Estado de Santa Catarina, Joinville, Abril 2018.

[17] R. R. Ladeira and R. R. Obelheiro. Práticas educacionais no ensino da computação forense: um relato de experiência. *Revista de Empreendedorismo, Inovação e Tecnologia*, 4(1):110–120, 2017.

[18] R. R. Ladeira and R. R. Obelheiro. Automatic challenge generation for teaching computer security. In

2018 XLIV Latin American Computer Conference (CLEI), pages 774–783. IEEE, 2018.

[19] R. Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[20] M. Mondal, B. Viswanath, A. Clement, P. Druschel, K. P. Gummadi, A. Mislove, and A. Post. Defending against large-scale crawls in online social networks. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 325–336. ACM, 2012.

[21] NIST. Secure hash standard (shs). *FIPS PUB*, 180(4), 2012.

[22] S. Northcutt. What the pcap contest actually tells us, 2016. [Online]. Disponível em: `https://linkedin.com/pulse/what-pcap-contest-actually-tells-us-stephen-northcutt`.

[23] W. M. Petullo, K. Moses, B. Klimkowski, R. Hand, and K. Olson. The use of cyber-defense exercises in undergraduate computing education. In *ASE@ USENIX Security Symposium*, 2016.

[24] Z. C. Schreuders, T. Shaw, G. Ravichandran, J. Keighley, M. Ordean, et al. Security scenario generator (secgen): A framework for generating randomly vulnerable rich-scenario vms for learning computer security and hosting ctf events. In *USENIX*. USENIX Association, 2017.

[25] M. Suby and F. Dickson. The 2015 (isc) 2 global information security workforce study. *Frost & Sullivan in partnership with Booz Allen Hamilton for ISC2*, 2015.

[26] C. Taylor, P. Arias, J. Klopchic, C. Matarazzo, and E. Dube. Ctf: State-of-the-art and building the next generation. In *2017 {USENIX} Workshop on Advances in Security Education ({ASE} 17)*. USENIX Association, 2017.

[27] G. Vigna. Teaching network security through live exercises. In *Security education and critical infrastructures*, pages 3–18. Springer, 2003.

[28] G. Vigna, K. Borgolte, J. Corbetta, A. Doupe, Y. Fratantonio, L. Invernizzi, D. Kirat, and Y. Shoshitaishvili. Ten years of ictf: The good, the bad, and the ugly. In *3GSE*, 2014.

[29] R. Weiss, J. Mache, and E. Nilsen. Top 10 hands-on cybersecurity exercises. *Journal of Computing Sciences in Colleges*, 29(1):140–147, 2013.