UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DANIEL LA RUBIA ROLIM

VISÃO GERAL SOBRE AS TRAPAÇAS E MÉTODOS ANTITRAPAÇAS EM JOGOS DIGITAIS

Entendendo a indústria, atores, métodos e preocupações acerca da privacidade na garantia de integridade competitiva entre os jogadores

RIO DE JANEIRO
2023

DANIEL LA RUBIA ROLIM

VISÃO GERAL SOBRE AS TRAPAÇAS E MÉTODOS ANTITRAPAÇAS EM JOGOS DIGITAIS

Entendendo a indústria, atores, métodos e preocupações acerca da privacidade na garantia de integridade competitiva entre os jogadores

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Paulo Henrique de Aguiar Rodrigues

RIO DE JANEIRO

2023

CIP - Catalogação na Publicação

R748v

Rolim, Daniel La Rubia
VISÃO GERAL SOBRE AS TRAPAÇAS E MÉTODOS ANTITRAPAÇAS EM JOGOS DIGITAIS: Entendendo a indústria, atores, métodos e preocupações acerca da privacidade na garantia de integridade competitiva entre os jogadores / Daniel La Rubia Rolim. -- Rio de Janeiro, 2023.
56 f.

Orientador: Paulo Henrique de Aguiar Rodrigues.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Computação, Bacharel em Ciência da Computação, 2023.

1. Trapaças e anti-trapaças em jogos digitais. 2. Segurança da Informação. 3. Privacidade. I. Rodrigues, Paulo Henrique de Aguiar, orient. II. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

DANIEL LA RUBIA ROLIM

VISÃO GERAL SOBRE AS TRAPAÇAS E MÉTODOS ANTITRAPAÇAS EM JOGOS DIGITAIS

Entendendo a indústria, atores, métodos e preocupações acerca da privacidade na garantia de integridade competitiva entre os jogadores

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 25 de abril de 2023

BANCA EXAMINADORA:

Prof. Paulo Henrique de Aguiar Rodrigues
Ph.D. (UCLA)

Prof. Severino Collier Coutinho
D.Sc. (UL-UK)

Prof. Silvana Rossetto
D.Sc. (PUC-RIO)

Dedico este trabalho a todos que têm nos jogos digitais uma fonte de diversão, amizades e alívio para as aflições do cotidiano; e que já sofreram (ou irão sofrer) com trapaceiros em suas jogatinas.

AGRADECIMENTOS

Após tantos anos, tenho a honra de poder agradecer pela graduação que se encerra. Agradeço imensamente ao meu pai, Angelo, e à minha mãe, Patrícia, que se sacrificaram muito para que eu tivesse a oportunidade de chegar até aqui. Sou grato pelas suas noites mal dormidas, por todo o trabalho que tiveram e por todos os gastos que foram colocados nos meus estudos e em meu conforto, espero não tê-los desapontado. Sou grato também ao Thiago, meu irmão e melhor amigo, que tem um coração de ouro e que eu sei que sempre vou poder contar. Que ele saiba que eu vou estar com ele até o fim.

Agradeço também à Mylena, minha esposa (quase), que me acompanhou ao longo de praticamente todo o meu percurso na UFRJ e foi, e é, um apoio essencial em todos os caminhos que eu percorro na minha vida. Se não tivesse tido o suporte dela, do meu irmão e dos meus pais, dificilmente eu teria conseguido chegar até o fim.

De todo o meu coração, agradeço aos meus familiares mais próximos. Às minhas tias e tios de consideração, minhas avós e avôs, primos e primas. Deixo uma menção honrosa e grata ao meu avô Zé, que com todo o seu entusiasmo com computadores, internet e tecnologia, fez com que já na minha infância o meu interesse nessa área fosse moldado.

A todos vocês, o meu muito obrigado pelo apoio, força e carinho. Ter uma família tão especial, que me ama e apoia, sem as cobranças tão comuns, é o maior presente e a maior alegria que eu poderia ter. O resto é só o resto.

Sou grato a todos os meus amigos e amigas, que permanecem comigo ou não. A todos os que tive a honra de conhecer ao longo da vida e que, de alguma forma, tornaram o meu momento acadêmico um pouco mais leve e divertido. Agradeço em especial ao GRIS e aos que estiveram comigo enquanto estive lá, fiz amigos que vou levar para o resto da vida e que, por me enxergar um pouco em cada um deles, me permitiram entender que eu pertencia àquele lugar. Agradeço também aos meus amigos do *Discord* e Pedro II, que continuam comigo ao longo de tantos anos. É com eles que divido um pouco da minha paixão por jogos, além das alegrias e tristezas rotineiras. E aos meus amigos da Igreja, meu muito obrigado pelos momentos compartilhados e pela calmaria que me trazem.

Nominalmente, quero agradecer ao Victor Pires, que depositou confiança em mim no início da minha trajetória acadêmica e abriu portas para que hoje eu seja um bom profissional. Além dele, agradeço ao Leonardo Ventura, que me ajudou a estudar e persistir em diversos momentos, tudo por amizade. São duas pessoas incríveis e que merecem muitas felicidades e sucesso.

E por toda a fé, perseverança e humanidade que desenvolvi ao longo desses anos, dou graças a Deus.

RESUMO

Com a democratização do acesso à internet e a evolução tecnológica, os jogos digitais deixaram de ser somente uma forma de entretenimento e se tornaram também uma profissão, com o avanço do termo Esporte Eletrônico. Devido à popularização dos jogos e profissionalização de seus cenários competitivos, a preocupação com o combate às trapaças em jogos digitais é, hoje, uma questão de extrema relevância para as Desenvolvedoras, que precisam criar meios de manter a base de jogadores e garantir um ambiente íntegro, sem trapaças, para evitar a destruição do cenário competitivo e garantir a sobrevivência do próprio jogo, que é a fonte de receita. Este trabalho busca apresentar todos os pontos principais da Indústria de Trapaças dos jogos eletrônicos, nomeando os sujeitos que formam a Indústria, as maneiras de trapacear e algumas das formas com que as Desenvolvedoras de jogos buscam impedir o abuso em seus produtos. Ao final, será abordada uma discussão acerca da privacidade do usuário, buscando justificar a necessidade das empresas que desenvolvem *software* anti-trapaça de ter níveis de acesso privilegiado no dispositivo do usuário.

Palavras-chave: segurança da informação; jogos digitais; trapaças; anti-trapaças; esportes eletrônicos; privacidade.

ABSTRACT

With the democratization of access to the Internet and technological evolution, digital games are no longer used solely for entertainment, but have also become a profession with E-Sports. Given the rise in popularity of these games and the competitive scene, game development companies have become increasingly concerned about cheating in digital games. Companies need to find means to maintain their user base and ensure a safe environment, with no cheating, to avoid the destruction of the competitive scene and guarantee the game survives since it is their source of revenue. This paper aims to introduce the main topics on the Cheating Industry of electronic games, naming who forms the Industry, ways of cheating and some of the alternatives companies use to stop abuse of their products. In the end, we address a discussion around user privacy, exploring why it may be necessary that the anti-cheat software holds some sort of privileged access to the user's device.

Keywords: information security; videogames; cheats; anti-cheats; esports; user privacy.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Surgindo na década de 40, os computadores têm evoluído de maneira crescente e muito acelerada desde então. Ano após ano, com as façanhas que têm sido alcançadas através das tecnologias computacionais, torna-se uma tarefa difícil adivinhar quais serão os rumos que a sociedade terá no que se refere ao "Mundo Digital". Os computadores, que foram criados com propósitos inicialmente militares, hoje fazem parte de todos os contextos sociais e mantêm a sociedade em movimento através da Internet. São ferramentas de trabalho, de lazer e de futuro, que só tendem a evoluir e ocupar ainda mais espaços com o passar do tempo.

Antigamente, quando se falava em esporte, era comum pensar naqueles que estavam presentes na infância e adolescência da maioria dos cidadãos, como futebol, basquete ou vôlei, por exemplo. Hoje, tem-se uma nova geração que faz com que os computadores ocupem um espaço ativo e essencial também nessa área. Os *esports*, simplificação para Esportes Eletrônicos, são uma nova modalidade de Esporte em que jogadores de dispositivos eletrônicos podem colocar à prova suas habilidades, seja com o objetivo de se divertirem ou competirem profissionalmente (WILLINGHAM, 2018).

Com o crescimento da indústria de jogos eletrônicos e a recente expansão de seus cenários competitivos, a preocupação com o fornecimento de um ambiente que possua integridade competitiva para seus jogadores é um ponto chave e uma pauta frequente em diversas discussões. Lutar contra trapaceiros é uma tarefa extremamente importante para as desenvolvedoras de jogos online, que se veem obrigadas a investir em *softwares* antitrapaça para tentar mitigar abusos em seus produtos e serviços, tentando minimizar essas ocorrências e fornecer um ambiente íntegro para os jogadores, evitando a frustração deles - visto que isso pode causar o afastamento da base de jogadores, caso os mesmos não se sintam amparados pela empresa e estejam insatisfeitos com a qualidade de vida do jogo (RODRIGUES, 2022). Entende-se por integridade competitiva, no contexto de jogos digitais, o fornecimento de um ambiente justo entre os jogadores, sem a ocorrência de abusos ou trapaças que ocasionem prejuízos para uns e benefícios para outros.

Devido ao fato de que a Internet permite acesso a qualquer tipo de conteúdo num estalar de dedos, a distribuição de programas de trapaça (popularmente conhecidos como *cheats*) é realizada com extrema facilidade e rapidez, além de que o compartilhamento de materiais educativos que ensinam a burlar sistemas, realizar engenharia reversa e buscar a exploração de brechas fazem com que haja um aumento na quantidade não só de trapaceiros que consomem as ferramentas, mas também favorece o aumento de *hackers*, que criam as ferramentas.

Diante desse desafio crescente, as empresas que realizam a criação de sistemas antitrapaça buscam diariamente novas formas de impossibilitar a trapaça em jogos, buscando

aumentar seu leque de técnicas e estratégias de negócio. Algumas das técnicas utilizadas por essas empresas podem atuar sobre a máquina dos jogadores ou diretamente sobre eles, através da utilização de seus dados, como será visto mais a frente neste trabalho. Por conta disso, são técnicas consideradas por muitos como invasivas e geram preocupações na comunidade de jogadores que, por não entender a problemática que envolve a luta contra trapaceiros a nível de *software* e *hardware*, acaba julgando os sistemas antitrapaças como igualmente maliciosos (The Riot Security Team, 2020).

Embora vários argumentos dos jogadores contra esses sistemas de proteção não façam sentido, principalmente ao serem levantados os aspectos técnicos que transpassam a questão, a preocupação com a privacidade de seus dados não só é legítima, como também é uma discussão necessária. Após o reconhecimento de que os dados dos usuários da Internet são um, senão o maior dos ativos do planeta, principalmente em decorrência do enfoque dado pela mídia após as revelações de uso indevido desses dados pelas empresas de tecnologia, começou a ser desenvolvido numa parcela da população um senso de consciência com as informações sobre si que estão sendo concedidas às empresas e como elas são utilizadas (VALENTE, 2019). As desenvolvedoras de jogos estão precisando de cada vez mais recursos para atuarem contra trapaças, e isso envolve a utilização de dados do usuário.

Este trabalho irá apresentar o contexto geral da indústria de trapaças, fornecendo uma visão ampla sobre como ela está estruturada e quais são os agentes que a compõem. Com isso, será possível entender algumas das motivações envolvidas e as principais dificuldades em combatê-la (capítulo 2). Tendo essa indústria sido apresentada, falaremos sobre as trapaças em si, categorizando-as de maneira mais técnica e apresentando os métodos mais comuns para a criação de trapaças em jogos digitais (capítulo 3). Deve estar claro que o foco do trabalho está nos métodos de trapaça que atuam ofensivamente contra o cliente ou servidor dos jogos. Posteriormente serão apresentados os métodos antitrapaça, categorizando-os e explicando as motivações por trás de cada um deles, bem como alguns dos pontos positivos ou negativos que os acompanham (capítulo 4). Depois, serão discutidas brevemente questões quanto à privacidade do usuário, num contexto em que está sendo visada a garantia de integridade competitiva nos jogos (capítulo 5). Por fim, será feita uma recapitulação do trabalho e das conclusões que foram tiradas de cada um dos temas abordados ao longo do mesmo, além de algumas sugestões para trabalhos futuros (capítulo 6).

2 INDÚSTRIA DE TRAPAÇAS

Antes de adentrar nos detalhes envolvendo os métodos de trapaça e antitrapaça em si, é importante entender o ecossistema em que essas ferramentas estão inseridas. Para que um usuário padrão faça uso delas, é preciso que esses *softwares* cheguem em suas mãos de alguma forma, portanto há um fornecedor. Para que um fornecedor distribua uma ferramenta de trapaça de maneira gratuita ou paga, é preciso que o mesmo seja criado por alguém, podendo esse alguém ser também um fornecedor e consumidor. Com esses três grandes sujeitos (criador, fornecedor e consumidor), há uma hierarquia básica definida, que mapeia o caminho que esse *software* realiza desde sua concepção até a utilização, que forma uma indústria (ALLAYES SIMON; RAUTAVA, 2016). Esses sujeitos serão apresentados brevemente ao longo deste capítulo.

Entender que existe uma indústria por detrás de grande parcela das trapaças em jogos é importante para auxiliar na compreensão da gravidade do tema. Embora muitos desenvolvedores de ferramentas de trapaça possam enxergar suas atividades apenas como um hobby, utilizando-se disso para diversão ou estudo, há pessoas que estão faturando uma grande quantidade de dinheiro com a venda dessas ferramentas de forma exclusiva ou limitada (TIDY, 2021). Dado que a utilização dessas ferramentas é contrária às políticas de uso de praticamente todos os jogos, principalmente os que permitem multijogador online, é correto dizer que há uma indústria ilegal que funciona de maneira parasitária, que depende da existência dos jogos para sobreviver, sugando deles os benefícios oriundos de sua popularidade ao mesmo tempo que os prejudica (CARBONE, 2021).

Em números, através de estimativas fornecidas pela empresa *Easy Anti-Cheat*, em uma apresentação realizada em 2016, companhias formadas por um único indivíduo podiam gerar um faturamento na faixa dos $750 mil dólares por ano, enquanto times de desenvolvedores poderiam atingir uma arrecadação de $1,5 milhões de dólares por ano. A nível global, estimava-se que o tamanho do mercado de trapaças era de mais de $100 milhões de dólares na época (ALLAYES SIMON; RAUTAVA, 2016). Com o objetivo de entender o cenário completo que envolve toda a cadeia de fornecimento e consumo de trapaças, a *Easy Anti-Cheat* criou um perfilamento para cada um dos grupos que são parte dessa cadeia, compreendendo os papéis individuais e a melhor maneira de se criar soluções com essas informações em mãos. Os resultados apresentados serão utilizados como base ao longo das próximas seções, que visam apresentar os sujeitos que compõem a indústria de forma geral. Será visto sobre os Criadores (2.1), que são todos que possam atuar na construção de uma trapaça; Fornecedores (2.2), que fazem a distribuição da trapaça para a comunidade interessada; e, por fim, os Consumidores (2.3), que são os usuários que necessariamente farão uso das ferramentas de trapaça.

2.1 CRIADORES

Conhecidos pelos mais diversos nomes, geralmente chamados *hackers*, os criadores são basicamente os responsáveis por construir as ferramentas de trapaça. Pode ser qualquer pessoa, possuindo um conhecimento mais aprofundado ou até mesmo um conhecimento bastante raso. O objetivo de um *hacker* não precisa ser fundamentalmente negativo, sendo apenas uma pessoa que possui os conhecimentos necessários para modificar as funcionalidades esperadas de um sistema, ultrapassando alguns limites previamente definidos e explorando suas fraquezas (McAfee, 2015).

Quando inserido num contexto de *game hacking*, um criador pode ser um jovem menor de idade que gostaria de adaptar o jogo aos seus desejos, "copiando e colando" tudo que encontra relacionado ao assunto; pode ser um pesquisador de segurança que deseja estudar vulnerabilidades em uma aplicação específica; pode ser até mesmo um *hacker* profissional, que desenvolve trapaças com a intenção de vendê-las e gerar ganhos financeiros (ALLAYES SIMON; RAUTAVA, 2016).

2.1.1 Script Kiddie

São conhecidos como *script kiddies* ou *scripters* os indivíduos que não possuem um conhecimento vasto e aprofundado em *hacking* e computação, que se utilizam principalmente de códigos e trapaças já prontos, copiado-os. Esta categoria compõe a maior parcela de *hackers*, que, no geral, oferecem menos risco por possuírem menos experiência e, consequentemente, não são capazes de criar ferramentas mais avançadas ou com grau inovador de complexidade. Muitas das vezes não possuem nem mesmo um conhecimento básico para entender o que está ocorrendo no código que está utilizando. Dentre as principais motivações, pode-se listar a diversão, criação de caos, atenção, vingança e também a curiosidade (LUTKEVICH, 2021b).

2.1.2 Pesquisadores de Segurança

Uma segunda parcela do grupo, que é bastante reduzida, é composta por pesquisadores de segurança. Podem ser tanto acadêmicos, quanto entusiastas ou profissionais da área de cibersegurança. Ao contrário dos *script kiddies*, possuem um conhecimento avançado e são mais experientes, sendo capazes de desenvolver provas de conceito para estudo da segurança nos jogos e quais são os mecanismos antitrapaça utilizados, além de formas de quebrá-los. As motivações não costumam ter interesse comercial (a não ser em casos específicos de *bug bounty* - recompensa ao descobrir vulnerabilidades, ou programa similar), possuindo como principal interesse a geração de conhecimento (bugcrowd, 2022).

2.1.3 Hacker

Compõe o topo da cadeia de suprimento, sendo os principais criadores das trapaças que causam problemas sérios aos jogos em que estão focados. Esse grupo é formado por especialistas, que possuem habilidades muito avançadas em desenvolvimento de *software* e engenharia reversa, entendendo todos os processos que ocorrem em baixo nível, nas camadas mais próximas do sistema operacional e do *hardware*. Ao contrário de *script kiddies* ou iniciantes, são capazes de desenvolver ferramentas avançadas que utilizam métodos complexos para burlar as defesas dos jogos, demorando mais tempo para serem identificados e corrigidos. Dentre as motivações, pode ser citado tanto o retorno financeiro gerado pelo desenvolvimento de ferramentas exclusivas ou questões ideológicas (ALLAYES SIMON; RAUTAVA, 2016)(CHAI, 2021).

2.2 FORNECEDORES

Para que as trapaças cheguem ao alcance dos usuários, é preciso que sejam distribuídos de alguma forma. O modo mais simples, e a porta de entrada para todos os utilizadores e criadores, costumam ser as comunidades abertas (fóruns), onde é fomentada a discussão sobre *game hacking*, incluindo tutoriais sobre como criar algumas ferramentas mais simples. Nesses fóruns é realizada a distribuição dos *softwares* de trapaça que são amplamente utilizados pelos jogadores. Essas comunidades podem ser tanto dedicadas a um jogo específico, quanto gerais, apresentando conteúdo de diversos jogos - geralmente os mais populares.

Quando o usuário deseja mais exclusividade e recursos, além de tentar fugir das detecções pelas ferramentas antitrapaça, é possível recorrer a algumas trapaças exclusivas (pagas), disponibilizados por *Cheat Publishers*. Essas "editoras" muitas vezes são negócios legítimos em seus países de origem, realizando pagamento de tributações, possuindo gerência profissionalizada e registro do negócio. Isso permite que, em alguns casos, algumas desenvolvedoras busquem recursos judiciais contra elas (BROUGHAN, 2021).

Por último existem as comunidades fechadas, que funcionam de maneira extremamente restrita e possuem controle de acesso de novos usuários baseados em reputação, com disponibilidade limitada. Acessar esses grupos é uma tarefa complicada, visto que são necessárias entrevistas com um membro e até mesmo cópia de documentos para estabelecimento de confiança. Os programas de trapaça compartilhados em comunidades fechadas são específicos e de valor elevado (não sendo exagero colocar a faixa de $1000 dólares). Por isso, além da reputação do criador e da comunidade que está envolvida, tenta-se criar a ferramenta de forma que a mesma seja completamente indetectável, tomando o máximo de cuidado possível (ALLAYES SIMON; RAUTAVA, 2016).

2.3 CONSUMIDORES

Por fim, a base que sustenta essa indústria são os consumidores, que são conhecidos nos jogos como *hacks* ou *cheaters* (trapaceiros). São eles quem utilizam as ferramentas para receber os mais diversos tipos de vantagens e causam irritação e frustração nas comunidades de jogos em geral (Oxford Leaner's Dictionaries, 2021). É comum que os jogadores enxerguem todos os usuários que trapaceiam da mesma forma, porém num contexto de desenvolvimento de métodos contra trapaça, é importante classificar corretamente a motivação dos ofensores para se ter uma visão holística, que permita a exploração das nuances existentes.

Pode-se dizer que a categoria mais famosa de trapaceiros é composta pelos *Griefers*. Esse termo informal é utilizado na comunidade de jogos online para definir uma pessoa que deliberadamente provoca e assedia outros jogadores, com o objetivo de destruir sua diversão (WARNER D.E. E RAITER, 2005). Um *griefer* não é necessariamente um *trapaceiro*, porque ele pode atingir seus objetivos de diversas maneiras. Entretanto, um trapaceiro que é um *griefer* será aquele usuário que não fará a menor questão de esconder o que está fazendo. Ele dará o seu máximo para destruir todo o ambiente ao seu redor, sem se importar com as consequências. Seu único propósito é causar prejuízo e suprir suas motivações, compostas por prazer, poder, controle e desafio (ACHTERBOSCH; MILLER; VAMPLEW, 2021).

Embora os *griefers* componham a categoria de trapaceiros mais famosa, não são o maior número. Isso porque a ampla maioria dos jogadores são casuais e, consequentemente, a maioria dos trapaceiros também. Ao contrário dos anteriores, o objetivo dos trapaceiros casuais não é necessariamente causar prejuízo para outras pessoas (mesmo que o façam). O objetivo primário é tornar o jogo mais fácil e prazeroso para eles próprios, buscando atingir algum nível que gostariam e acabam não conseguindo por conta própria. Enquanto os *griefers* não fazem a menor questão de esconder que estão utilizando essas ferramentas, os casuais disfarçam, querendo esconder esse uso. Eles gostam que as pessoas pensem que eles são bons, eles gostam de vencer e desejam ter essa satisfação sem que sejam descobertos. São os mais difíceis de banir e comprovar a utilização, dado que não costumam fazer um uso indiscriminado (ALLAYES SIMON; RAUTAVA, 2016).

Uma terceira parcela dos trapaceiros é conhecida como vigilantes (ou justiceiros). De forma geral, são os jogadores que são vítimas de trapaças e da falta de atuação da empresa responsável pelo jogo, que cria um sentimento de revolta. Como uma tentativa de se vingar pelas frustrações sofridas, utilizam as ferramentas como uma forma de fazer "justiça". Por vezes também se assemelham aos *griefers*, dado que não escondem a utilização de trapaça, mas a forma de agir é distinta (ACHTERBOSCH; MILLER; VAMPLEW, 2021). Já tive a experiência de estar jogando uma partida de um jogo *multiplayer* onde o jogador do outro time, que antes estava apenas levantando alguma suspeita, deixou a timidez de lado

e começou a utilizar a trapaça de forma indiscriminada. No mesmo momento, um jogador da minha equipe informou: "já retorno, vou ligar" (em referência a habilitar o *software* de trapaça). Ele voltou utilizando a ferramenta e disse que a situação estava igualada. A partida com 10 jogadores se tornou um confronto entre 2, em que venceria o que tivesse o melhor *software* de trapaça.

Existem também aqueles que são conhecidos como 'Seguidores'. Ao contrário dos outros, essa parcela é composta pela base de jogadores que é apaixonada por algum jogo e tenta fazer de tudo para permanecer nele, mantendo-o vivo. São aqueles jogadores que só irão utilizar alguma ferramenta de trapaça caso a situação saia completamente de controle, numa tentativa de nivelar o jogo ou permitir que eles continuem sendo capazes de desfrutar de alguma forma.

No fim das contas, a existência de trapaceiros pode ser tanto um problema causado diretamente pela empresa que desenvolve o jogo, quanto um problema social, causado pela sociedade como ela é. Independente dos esforços, dificilmente se espera que algum dia as comunidades de jogos estejam totalmente livres de trapaceiros, porém cabe às desenvolvedoras entenderem o seu público e utilizarem esse conhecimento para mitigar esse problema sistêmico da maneira mais efetiva possível.

No próximo capítulo, tendo portanto sido apresentados todos os sujeitos que fazem parte da indústria de trapaças, serão abordadas as trapaças em si. Para clarificar o entendimento do leitor, serão fornecidos exemplos reais e recentes, auxiliando no entendimento sobre como essas ferramentas afetam a jogabilidade e quais são alguns dos passos tomados pelos *criadores* para conseguir burlar a segurança dos jogos digitais.

3 ENTENDENDO AS FORMAS DE TRAPAÇA

Antes de adentrar nos métodos de combate às trapaças em si, é importante entender as formas como elas acontecem. Embora em todos os casos esteja presente a intencionalidade do usuário, uma trapaça pode ocorrer tanto utilizando programas de terceiros, que efetivamente realizaram uma implementação maliciosa para fornecer vantagens, quanto também podem ocorrer por erros no desenvolvimento do jogo.

A primeira forma de trapacear se dá utilizando das implementações falhas do próprio jogo. Nesse tipo de trapaça, não é necessário criar ou utilizar ferramentas de terceiros, basta que o usuário consiga identificar fraquezas presentes nas mecânicas de jogo ou em seu funcionamento para viabilizar formas de adquirir vantagens para si. Os exemplos mais comuns envolvem *bugs* e *exploits*, que podem surgir oriundos do déficit de testes no código do jogo ou pela imprevisibilidade de alguns casos, que naturalmente não passam na cabeça do desenvolvedor. Para tornar mais claro, pode-se dizer que um *bug* é criado a partir de um problema na lógica de um programa, que acaba por deixar escapar alguma condição que é importante para o funcionamento final do mesmo. Um *exploit* (no português, "explorar") nada mais é do que a exploração desse *bug*, que ocorre quando alguém entende qual é a lógica necessária para fazer com que o comportamento inesperado ocorra de maneira intencional (CoinMarketCap, 2022). Como ensinado em (MARTIN, 2009), a melhor forma de reduzir a ocorrência de *bugs* é fazer com que o *software* seja construído seguindo boas práticas e padrões de projeto, bem como torná-lo resiliente através de uma ampla cobertura de testes.

Em 2014, o jogo *League of Legends*, que é *multiplayer* online, sofreu com um *bug* que prejudicou bastante a comunidade por algumas semanas. Nesse jogo, todos os jogadores devem comprar itens ao longo da partida para que seus personagens fiquem mais fortes. Alguns desses itens possuem efeitos "especiais", que interagem diretamente com o personagem inimigo, que era o caso da "Espada do Rei Destruído". O item, ao ser ativado pelo jogador, deveria causar lentidão e um pequeno percentual de dano ao adversário. Entretanto, por uma falha no código do jogo, alguns jogadores descobriram como explorar um *bug* com o item, fazendo com que qualquer jogador adversário que se aproximasse fosse eliminado instantaneamente (KESUAUS, 2014).

Uma outra forma de abusar de um jogo sem a utilização de programas de terceiros é através da troca de itens ou serviços nos jogos através de transações com dinheiro real, fora do ambiente do jogo. Embora isso não ofereça vantagens em todos os tipos de jogos, em alguns nichos pode representar um grande problema, visto que atrapalha a economia do jogo e, obviamente, favorece os jogadores que se utilizam disso. Para as desenvolvedoras essa maneira de trapacear pode ser uma grande dor de cabeça, visto que é difícil avaliar quando um usuário está obtendo esse tipo de vantagem. Os códigos de conduta (ou

contratos de licença de usuários finais) geralmente proíbem o compartilhamento de contas entre usuários como uma maneira de inibir o uso irregular das contas em seus jogos (LEHTONEN, 2020).

O foco deste trabalho, entretanto, está nos métodos de trapaça que se utilizam de programas de terceiros, ou seja, todas as trapaças que, de alguma forma, realizam ações ofensivas diretamente contra o cliente ou servidor do jogo. Quando se pensa em métodos antitrapaça, são justamente essa categoria de trapaças que se deseja combater, que representam um grande desafio para todas as desenvolvedoras de jogos online, principalmente aquelas que não possuem muitos recursos financeiros para sustentar um serviço antitrapaça ou desenvolvê-lo internamente. A criação de jogos *indie online* (de produtoras independentes que muitas das vezes utilizam recursos dos próprios desenvolvedores) (DUTTON, 2022) pode ser um grande desafio, por exemplo. Não se espera que de início ocorra um problema envolvendo infestação de trapaceiros, mas no caso da popularização do jogo, sem sombra de dúvidas será necessário buscar um serviço para tentar mitigar as ondas que virão.

Um exemplo da dificuldade das desenvolvedoras *indie* é o *Fall Guys*, que foi lançado em 2020 pela *Mediatonic* e explodiu rapidamente. Inicialmente tentaram utilizar uma solução própria contra as trapaças, chamada de *Cheater Island* (Ilha dos Cheaters), que funcionava de maneira relativamente simples. Ao invés de banir os usuários detectados utilizando ferramentas indevidas, os mesmos seriam isolados em "ilhas", de forma que as partidas seriam formadas somente com outros trapaceiros. A estratégia, porém, não deu certo por diversos motivos. Embora estivessem confiantes de que o sistema era bom, a empresa não estava preparada para lidar com tantos jogadores, além de não esperar que os *cheaters* fossem tão longe para conseguir trapacear. Quando perceberam que estavam em uma "corrida armada", buscaram apoio com a *Easy Anti-Cheat* (especializada em *Anti-Cheat as a Service*) para tentar resolver ou, ao menos, diminuir o problema (FallGuysGame, 2020).

As seções a seguir apresentarão alguns tipos de trapaças, que se subdividem tanto pelas camadas, quanto pela forma com que são construídas. Inicialmente serão abordadas trapaças que não atacam diretamente a aplicação, como as trapaças em *Hardware* e *bots* (em alguns casos). Posteriormente, trataremos da adulteração de pacotes (que visa afetar a comunicação entre cliente e servidor) e da análise de memória e engenharia reversa (que atacam diretamente o cliente do jogo). Ao final será apresentada uma tabela contendo alguns exemplos de trapaças que são comuns em jogos online.

3.1 TRAPAÇAS EM HARDWARE

Dentre as grandes categorias, uma que não é popularmente utilizada entre os jogadores, embora seja bastante conhecida pela comunidade, são as trapaças em hardware. A

ideia é simples: utilizar um dispositivo externo que captura dados do jogo ou das ações que estão sendo realizadas (como segurar um botão do *mouse*), processa e implementa algumas melhorias ou correção nessas ações do jogador (COCKFIELD, 2022). Esse tipo de ferramenta pode ser bastante eficiente e difícil de lidar, uma vez que não está tentando burlar, de forma direta, o cliente do jogo, seus servidores e, nem mesmo, a memória do sistema operacional.

Esse tipo de trapaça costuma ser mais utilizado em jogos de tiro, uma vez que estes são mais frágeis às possibilidades existentes para essa categoria. Para exemplificar, é interessante conhecer alguns conceitos dos jogos do gênero (geralmente FPS - *First Person Shooter*):

- Mira → Como o próprio nome diz, o objetivo primário do jogador em um jogo de tiro é ter capacidade de mirar nos alvos necessários, o que requer precisão ao movimentar o *mouse* ou *joystick*;

- Recuo (*recoil/spray*) → Nas franquias de FPS mais famosas da última década, o recuo é um elemento fundamental. Consiste em um movimento de dispersão. A partir do momento que o jogador realiza disparos em rajadas, a arma perde a precisão e os tiros deixam de ir na direção da mira. Embora cada arma possua um grau de dispersão diferente, é comum que os jogos implementem esse recuo seguindo um padrão. Quanto mais tempo o usuário está atirando em uma rajada, mais os tiros dispersarão para cima e, após, para os lados.

Uma trapaça em hardware teria a capacidade de atuar em cima desses dois conceitos fundamentais. Para exemplificar, um desenvolvedor poderia implementar em um *Arduino* (uma placa de prototipagem eletrônica) um *software* que corrigiria o recuo da arma do jogador quando ele atirasse em rajada, uma vez que se conhece o tempo de tiro e o formato de dispersão padrão da arma do jogo. O desenvolvedor também poderia implementar um *triggerbot*, que identificaria quando um inimigo passasse em cima da mira do jogador e realizaria os disparos automaticamente, dando vantagem por conta de remover o tempo de reação (intervalo entre o ser humano perceber algo e agir). Mais ainda, seria possível implementar um *aimbot*, que identificaria o alvo ou inimigo na tela e corrigiria automaticamente a mira do jogador, levando-a ao local correto.

Existem diversas maneiras diferentes de realizar a implementação dessas trapaças, mas um exemplo simples pode ser dado com o *triggerbot*. O criador da trapaça poderia criar um código que lê os *pixels* exibidos na janela do jogo, escaneando uma área pequena que corresponde à mira do jogador; ao identificar que o *pixel* está na cor do inimigo (que é definida na implementação da trapaça), é enviada automaticamente uma ação para pressionar o botão esquerdo do mouse, que costuma ser o botão responsável pelo gatilho nos jogos de tiro.

Um exemplo real é apresentado pelo criador de conteúdo *Sparkles*, em seu canal no *YouTube*, onde apresenta a trapaça que comprou e denuncia sua utilização por jogadores semi-profissionais em ligas competitivas (SPARKLES, 2020a). No Brasil, em 2009, ocorreu um caso que se tornou clássico, sendo assunto até hoje entre a comunidade de jogadores de *Counter Strike*, que há mais de 20 anos perdura na lista de jogos de tiro mais populares do mundo (TE, 2014). O jogador, conhecido como *aspx*, foi banido por seis anos de todas as principais competições do Brasil por conta de uma acusação de trapaça na *World Cyber Games*, uma competição qualificatória nacional que ocorria no modelo presencial (BUENO, 2009). A acusação foi baseada no estilo de jogo "esquisito" do jogador, uma vez que não foi possível provar que o mesmo realmente utilizou algum tipo de ferramenta ilícita. A pressão dos times e da comunidade que acompanhava o torneio fez com que os organizadores decidissem pela punição. O caso representou um marco principalmente pela quebra de paradigma, uma vez que ninguém acreditava que seria possível utilização de trapaças em torneios presenciais. Como não foi comprovada a utilização de *trapaça* com evidências factuais, não se pode afirmar que esse caso representa de fato uma trapaça em *hardware*, mas é um exemplo do que sua utilização pode ter causado (MARQUES, 2017)(MELO, 2017).

3.2 BOTS

Têm-se por *bot* um diminutivo da palavra *robot* (robô), que no contexto de Ciência da Computação e Desenvolvimento é uma aplicação criada para simular o comportamento humano, realizando ações mecânicas de forma repetível e automática. Inserindo na categoria de trapaças, pode-se pensar como *bots* qualquer tipo de automação que tenha como objetivo facilitar, reduzir ou aprimorar o trabalho que um jogador teria para realizar ações, obter conquistas ou evoluir dentro do jogo (KANG et al., 2013). Por simular as ações dos jogadores, os *bots* se comunicam com o jogo da mesma forma que os jogadores o fariam, através do envio de comandos do teclado, *mouse* ou controle.

Um tipo de utilização muito comum é realizado em jogos em que o tempo jogando é relevante para atingir um nível maior no jogo, ganhar atributos ou conseguir itens. Em alguns jogos, é possível automatizar as ações do usuário de forma que o mesmo pode utilizar um programa que "jogará" por ele. O jogador pode, então, deixar que seu personagem passe sozinho de fases e ganhe experiência sem que o mesmo esteja no computador. Isso representa uma grande vantagem, dado que é possível utilizar o máximo de horas possíveis no jogo, enquanto jogadores lícitos precisam interromper sua jogatina para dormir ou realizar outras atividades. Em alguns jogos, como o já mencionado *League of Legends*, algumas pessoas utilizam *bots* para aumentar o nível da conta e, posteriormente, vendê-la, pois é atrativo para alguns usuários não precisarem gastar tanto tempo para que tenha uma conta no nível necessário para o que desejam (SMYTH, 2022).

3.3 ADULTERAÇÃO DE PACOTES

Outro meio de trapaça, que envolve um método um pouco mais elaborado de criação, visa realizar uma modificação nos pacotes que são enviados do cliente do jogo para o servidor. Para que isso seja possível é necessário que os protocolos da aplicação sejam mal projetados, de forma que o servidor confie cegamente nos dados recebidos a partir do cliente. Para exemplificar, considere um jogo onde o personagem possui uma barra de vida e, constantemente, o cliente que está em execução na máquina do jogador envia pacotes para o servidor informando a quantidade de vida que o personagem possui. Caso o servidor confie em todos os pacotes recebidos do cliente, sem realizar uma verificação de integridade, um trapaceiro pode adulterar os pacotes enviando sempre a informação de que o personagem está com a vida máxima, isso faria com que o mesmo se tornasse imortal (LEHTONEN, 2020).

Ainda no contexto de pacotes, pode-se falar sobre os *replay attacks*. O ataque de repetição é um ataque ao protocolo de segurança, que reenvia mensagens a partir de contextos diferentes para o contexto pretendido, fazendo com que os participantes honestos pensem que o protocolo está sendo respeitado e executado com sucesso[1] (MALLADI; ALVES-FOSS; HECKENDORN, 2002). A origem da mensagem define a categoria do ataque, de forma que, no caso de uma execução externa, por exemplo, um trapaceiro poderia realizar uma ação (que possui intervalo de tempo para ser realizada novamente) dentro do jogo e capturar o pacote correspondente a essa ação. Na sequência, poderia reenviar esse pacote para o servidor que, se for mal construído, faria com que a ação voltasse a ocorrer sem a restrição de tempo (LEHTONEN, 2020, apud MALLADI; ALVES-FOSS; HECKENDORN, 2002).

Outro meio de adulteração de pacotes é conhecido como *spoofing*, que em tradução literal significa "falsificação". Consiste na tentativa de fraude no jogo onde o trapaceiro tenta se passar por outro usuário. Embora o *spoofing* seja um termo geral e bastante utilizado para descrever ataques no meio de cibersegurança (CrowdStrike, 2022), no contexto de trapaças pode ser entendido como a falsificação de informações presentes no cabeçalho de um pacote, de forma que o servidor acredite que o mesmo está sendo recebido por outro remetente. Esse tipo de ataque pode ocorrer em jogos que utilizam o conjunto de protocolos TCP/IP em suas partidas ou sessões de jogo (jogos de turno, de cartas ou aqueles em que a confiabilidade é mais importante do que a latência). Como o Protocolo de Internet (IP) não possui método para validação de autenticidade, é necessário que esse controle seja realizado nas camadas de transporte e de aplicação para compensar essa lacuna. Já nos jogos que precisam de uma latência baixa e não podem cogitar a utilização de TCP (jogos de tiro, MOBA [*Multiplayer Online Battle Arena*] ou em que as mudanças

[1] No original: "*an attack on a security protocol using replay of messages from a different context into the intended z(or original and expected) context, thereby fooling the honest participant(s) into thinking they have successfully completed the protocol run.*"

de estado ocorram diversas vezes em um intervalo de tempo muito curto), o protocolo utilizado é o UDP. Esse protocolo não sofre com o *spoofing*, uma vez que não é estabelecida conexão e todos os pacotes que chegam ao servidor, independente do remetente, são recebidos. Vale mencionar também o QUIC, protocolo recente de autoria da *Google*, que está sendo desenvolvido e implementado com o objetivo de atender às necessidades oriundas da evolução tecnológica. Esse protocolo é baseado em UDP, porém acrescenta o controle de congestionamento, retransmissão de pacotes e outras características sem abrir mão da velocidade, podendo se tornar uma opção aos desenvolvedores de jogos no futuro.

3.4 ANÁLISE DE MEMÓRIA E ENGENHARIA REVERSA

Implementações de trapaças que utilizam artifícios de manipulação de memória ou que demandam prática de engenharia reversa para sua construção são bastante comuns, justamente por conta de sua efetividade e maior poder, embora não sejam *softwares* triviais de serem construídos. Entende-se por análise de memória a capacidade do *hacker* de entender os dados do jogo que estão sendo armazenados na memória do computador e ao que esses dados correspondem na prática. Engenharia reversa é uma prática de desconstrução de um programa, de forma que a pessoa que a realiza tem a capacidade de compreender como o mesmo foi construído. A integração desses dois conhecimentos, geralmente aplicada por *hackers* ou pesquisadores de segurança, torna possível as manipulações mais complexas e eficientes (MERCES, 1981).

Uma das portas de entrada mais comuns no universo de trapaças, que provavelmente todo iniciante em *game hacking* já teve contato, foi utilizando a ferramenta *Cheat Engine*. Esse software tem como propósitos ser um *scanner* de memória, *debugger*, *disassembler*, *assembler*, inspetor do sistema, entre outros (Cheat Engine, 2022). O objetivo primário de todo *hacker* iniciante era utilizar o *Cheat Engine* para escanear a memória de um jogo (geralmente *single-player* e *offline*), identificar os endereços onde estavam sendo armazenadas informações relevantes como a vida, quantidade de munição, velocidade de movimento etc. A partir do momento em que tinha acesso a esses endereços, seria possível realizar uma alteração dos valores, dando ao usuário total controle dos seus atributos dentro do jogo (Wiki Cheat Engine, 2017).

Conhecendo, portanto, algumas das formas mais comuns de trapacear e onde encontrar brechas para o desenvolvimento de trapaças, parte-se, no Capítulo 4, para os meios de evitar com que os Criadores possam abusar dos jogos. Serão apresentados diversos métodos que buscam fortalecer a segurança das aplicações tanto no dispositivo do usuário, quanto nos servidores das empresas de jogos, bem como meios de identificar quanto um usuário está trapaceando ou não.

4 MÉTODOS ANTITRAPAÇA

Ao longo dos últimos anos, o tema Segurança, dentro do contexto de Computação, está evoluindo cada vez mais e se tornou uma das bases de todos os sistemas e projetos atuais, dada a evolução das ameaças e o simples fato de que o Mundo está e continuará evoluindo tecnologicamente. A partir do momento que as comunicações, os documentos, os pagamentos e todos os já conhecidos processos presentes na sociedade estão sendo migrados para o meio digital, é necessário buscar formas de protegê-los e garantir que pilares como Confidencialidade, Integridade e Disponibilidade sejam garantidos (ABNT, 2006).

Da mesma maneira que todos os sistemas estão evoluindo, os jogos (e a forma como são jogados) também estão. Há muitos anos atrás era comum no Brasil que as pessoas aguardassem o final de semana para se conectar à internet discada (por conta das promoções de custo fornecidas pelas operadoras) e, assim, adentrassem na rede mundial de computadores (VIEIRA, 2022); enquanto hoje o maior desafio de grande parte da sociedade é permanecer desconectado. Da mesma forma que as redes sociais aumentaram a proximidade (num contexto de disponibilidade de comunicação) entre as pessoas, os jogos também se popularizaram e dominaram a Internet. Enquanto a utilização de trapaças em jogos casuais e sem comunicação com a rede possam ser mais inofensivos, aquelas trapaças que são utilizadas em jogos multijogador se tornam cada vez mais preocupantes, tanto pelo crescimento exponencial das bases de jogadores online, quanto porque, nos últimos anos, ocorreu uma gigantesca profissionalização do cenário de Esportes Eletrônicos (*esports*), ao ponto de que hoje já é discutida a possibilidade de inclusão dessa modalidade nas Olimpíadas (SABINO, 2022).

Garantir a Integridade Competitiva mais do que nunca representa uma preocupação para todas as empresas desenvolvedoras de jogos, agora não só pelo fato de que os jogadores abandonam os jogos caso se sintam injustiçados ou abandonados pelas empresas, mas também por todo ecossistema de campeonatos e premiações financeiras envolvidas (NELSON, 2019). Um passo errado nas políticas de combate às trapaças pode representar a ruína de um jogo em potencial.

Esse capítulo tem como objetivo compilar os métodos antitrapaça mais comuns que são implementados nos jogos atuais, sem se aprofundar tecnicamente em como é realizada a implementação, mas focado em apresentar o problema e como o método o resolve. Vale lembrar que muitos desses métodos são desenvolvidos diretamente pela indústria de jogos e possuem sua implementação mantida em segredo, dado que abrir as estratégias poderia fornecer alguma facilidade aos *hackers* para burlarem seus métodos. Embora o autor não acredite em Segurança por Obscuridade, a luta entre trapaceiros e os sistemas antitrapaça é uma corrida contra o tempo, então forçar que seu adversário tenha que descobrir

as lógicas de suas defesas fornece um tempo adicional que pode ser precioso. Muitas das referências que serão utilizadas não são estudos científicos e acabam se tornando informações públicas quando *hackers*, pesquisadores de segurança ou antigos funcionários de empresas do meio fazem publicações sobre o tema em fóruns ou sites especializados.

Antes de serem apresentados os métodos em si, é importante fornecer uma breve explicação a nível de arquitetura de sistemas distribuídos. Quando se fala de Internet, temos dois conceitos muito comuns que são chamados de cliente e servidor. O cliente é aquele que está acessando uma aplicação e requisitando dados desse servidor, enquanto o servidor é aquele que está respondendo às requisições realizadas por um cliente. Quando falamos de jogos online, geralmente, por boas práticas de desenvolvimento, está presente uma arquitetura de cliente e servidor, de forma que os dados dos jogadores são enviados pelos seus dispositivos (cliente) e são recebidos, processados e reenviados por um servidor centralizado, disponibilizado pela empresa desenvolvedora do jogo. Em alguns jogos pode ser realizada a implementação de uma arquitetura *peer-to-peer* (P2P), onde os dispositivos de todos os jogadores envolvidos em uma partida se comunicarão uns com os outros para que o jogo seja processado, sem que haja uma centralização (AKALTAR; GREENBEARD, 2013) (CHRISTOPH et al., 2007).

4.1 LADO DO CLIENTE

Para cada tipo de problema ou desafio que é enfrentado, uma abordagem diferente deve ser adotada. Quando se buscam medidas que podem ser aplicadas no lado do cliente, trata-se de quais serão as proteções que serão implementadas diretamente no dispositivo do usuário, uma vez que, caso haja algum tipo de antitrapaça, ele é instalado junto ao jogo. São justamente as medidas implementadas no lado do cliente que irão gerar as maiores polêmicas e discussões no que diz respeito à privacidade, como será visto posteriormente.

Em jogos *singleplayer*, não é comum que haja uma preocupação tão grande com a capacidade do usuário de utilizar trapaças, dado que ele estará afetando unicamente o seu jogo e isso não representa um risco direto, tanto para a empresa desenvolvedora, quanto para os jogadores que também possuem este jogo. Mais do que isso, uma prática bastante comum que é realizada por desenvolvedoras de jogos *singleplayer* é fornecer a possibilidade do usuário de utilizar trapaças sem que haja a necessidade de programas de terceiros. Ou seja, a própria desenvolvedora fornece comandos que darão benefícios para o jogador (Eletronic Arts, 2022). Isso é feito justamente porque o objetivo é que o usuário tenha o máximo de prazer possível jogando, e como não irá afetar ninguém, isso não representa um risco. Em jogos como *Resident Evil*, jogo de sobrevivência que exige que o jogador colete recursos e economize na utilização devido à escassez, ao finalizar a campanha principal ou obter conquistas (recompensas por concluir missões dentro do jogo) ele é beneficiado com novos "modos" de jogo, como munição infinita e outros tipos

de armas, de maneira que ele possa jogar a mesma campanha novamente e tenha uma experiência diferente (DEMARTINI, 2019).

Por outro lado, uma proteção comum em jogos *singleplayer* e que não será abordada neste trabalho são as proteções para *Digital Rights Management* (DRM), soluções focadas em direitos autorais para impedir a pirataria (Denuvo by irdeto, 2022). Não há essa necessidade em jogos *multiplayer online* uma vez que os jogadores precisam estar em constante comunicação com o servidor, garantindo a integridade do jogo. Vale salientar que soluções de DRM podem ser bastante controversas, uma vez que alguns usuários relatam perda de desempenho por conta de certas implementações e alguns impeditivos, como o *Always Online DRM* (DRM Sempre Online), que exige que o usuário possua uma conexão com a internet sempre disponível para que possa jogar um jogo, buscando garantir que o mesmo possui uma cópia original (mesmo que esse jogo não seja *multiplayer online*) (MEGAMANX503, 2018).

4.1.1 Ofuscação de Código

O primeiro método a ser abordado não é uma técnica antitrapaça propriamente dita, porém adiciona um grau a mais de dificuldade para os *hackers* que pretendem encontrar meios de burlar um jogo. A ofuscação de código é uma técnica que consiste em realizar transformações no código fonte de uma aplicação de maneira que visualmente ele se torne impossível de ser lido, embora mantenha todas as suas funcionalidades anteriores (BALAKRISHNAN; SCHULZE, 2005).

Quando um jogo é distribuído, considerando que ele possua um conjunto de arquivos e um binário que será executado no dispositivo do jogador, deve-se ter ciência de que a propriedade intelectual da desenvolvedora está sendo entregue ao usuário, perdendo-se o controle do que é feito com isso. No contexto de criação de trapaças, entender como um jogo foi implementado e quais são as suas fraquezas é essencial para que se encontrem meios de burlá-lo, e isso é feito através da engenharia reversa. A ofuscação do código torna essa tarefa mais complicada uma vez que, mesmo que se chegue no código fonte da aplicação, esse será impossível de ser lido e compreendido a olhos humanos.

Porém, há de se afirmar que considerar esse método como antitrapaça seria o mesmo que confiar na Segurança pela Obscuridade. Tornar o trabalho de compreensão mais difícil e demorado não o torna impossível de ser realizado, ao mesmo tempo que não garante que diversos problemas de implementação sejam descobertos através de uma análise mais aprofundada. A ofuscação é utilizada principalmente para proteção de código fonte proprietário, impedindo uma cópia direta de sua implementação. Também é muito utilizada para mascarar o funcionamento de *malwares*, por exemplo, para passarem despercebidos por ferramentas de segurança. Novamente, em nenhum desses casos se impede a compreensão da lógica de implementação, só se dificulta o trabalho de quem está avaliando e

ganha-se tempo, fazendo com que algumas pessoas possam desistir, por não achar que o esforço vale a pena (YOU; YIM, 2010).

Podem ser considerados quatro critérios para avaliar a qualidade de uma ofuscação, sendo eles a potência (quanta obscuridade é acrescentada no programa), a resiliência (o quão difícil é quebrar a ofuscação), a furtividade (o quanto o código ofuscado se mistura com o restante do programa) e, por fim, o custo (qual é o *overhead* computacional adicionado à aplicação ofuscada) (BALAKRISHNAN; SCHULZE, 2005). Para que se atinjam altos níveis de qualidade, podem ser aplicadas técnicas mais simples, como renomear nomes de métodos e variáveis, empacotar um programa ou inserir código inútil; ou também técnicas mais complicadas como antidebug e anti-adulteração, criptografia de strings e transposição de código (LUTKEVICH, 2021a).

4.1.2 Criptografia de Código

Embora seja um conceito parecido e que, de certa forma, se cruza com a ofuscação, a criptografia é uma técnica diferente. Enquanto a ofuscação protege contra engenharia reversa focando em gerar uma versão semanticamente equivalente de um programa, porém menos compreensível, o foco da criptografia é garantir a confidencialidade dos dados, tornando o *software* mais resistente a adulterações (CAPPAERT et al., 2008).

Existem dois conceitos base para se falar sobre criptografia de código, cujos nomes correspondem à maneira como funcionam. O primeiro é chamado de descriptografia em massa, que realiza o processo de criptografia e descriptografia no programa inteiro de uma única vez. O segundo, conhecido como descriptografia sob demanda, funciona da maneira oposta, de forma que as funções do programa são descriptografadas sob demanda quando precisam ser utilizadas, sendo encriptadas novamente quando não são mais necessárias. Enquanto a primeira possui o problema de revelar todo o código do programa de uma única vez, ela não sofre com o *overhead* que o segundo método possui, por conta da necessidade de requisitar os métodos de encriptar e decriptar de maneira recorrente (CAPPAERT et al., 2008).

Pode-se exemplificar o processo de descriptografia em massa utilizando um exemplo que é bastante comum na área de segurança, mais especificamente quando se trata de análise de *malware*. O empacotamento de código (que também pode ser utilizado com fins legítimos) é uma técnica bastante utilizada por criadores de *malwares*, com o objetivo de tornar os arquivos mais difíceis de serem analisados ou detectados. Segundo (SIKORSKI; HONIG, 2012), os programas ofuscados são aqueles em que o autor tentou ocultar a execução, enquanto os programas empacotados seriam um subconjunto de programas ofuscados, em que o programa malicioso é comprimido e não pode ser analisado de maneira estática. O desempacotamento é realizado por completo no momento da execução.

Conforme sugerido por (CAPPAERT et al., 2008), o *framework* de descriptografia sob demanda tem como objetivo solucionar o problema da descriptografia em bloco. Caso o

programa seja executado utilizando o método de bloco, todo seu código será descriptografado de uma vez, permitindo que seja realizada uma análise dinâmica. Quando todo o código está criptografado e tem suas funções sendo descriptografadas somente ao ponto em que serão utilizadas, a análise se dará observando as funções uma a uma com o avanço da execução, para só assim tentar entender o que está acontecendo com o código do jogo. Isso irá consequentemente dificultar bastante o trabalho de análise e engenharia reversa.

4.1.3 Verificação de Integridade com Hash

Funções *hash* são implementações de algoritmos matemáticos que recebem uma quantidade de dados de tamanho não definido e retornam um resultado de tamanho fixo, possuindo diversas aplicações no universo computacional. Dentre as características essenciais do *hash* estão a irreversibilidade (a partir do momento que se chega no resultado, não é possível utilizar o resultado para chegar na origem) e a colisão (como o resultado é dado em um tamanho fixo de caracteres, é possível que alguns dados de entrada resultem no mesmo *hash*). Qualquer alteração que ocorra nos dados de entrada, mesmo que seja um único bit, pode causar uma alteração no valor do *hash* (3RD; JONES, 2001)(PRESHING, 2011).

Uma das mais utilizadas aplicações do *hash* está na verificação de integridade. Alguns sites, por exemplo, disponibilizam um valor de *hash* e o algoritmo utilizado, permitindo que os usuários verifiquem os arquivos que receberam após realizar um *download*. Isso serve para validar tanto que o *download* não foi corrompido, quanto a não ocorrência de alguma adulteração desse *software* baixado. Outra aplicação comum é na Computação Forense, onde os peritos precisam levar evidências ao tribunal e essas evidências precisam possuir garantia de integridade. Quando uma evidência é coletada, é realizado o cálculo de *hash* e esse valor é anotado junto da possível prova, que é mantida protegida em todo processo de cadeia de custódia. Os investigadores trabalham em cima de cópias das evidências e, caso encontrem alguma informação relevante, utilizam o *hash* para comprovar que a análise ocorreu em cima de uma cópia idêntica e não houve adulteração ou inserção de dados (ROUSSEV, 2009).

A utilização de *hash* como método antitrapaça está na verificação de integridade dos arquivos do jogo. Como no lado do cliente ocorre todo o processamento criado através da *game engine* (motor de construção do jogo), é importante que se verifique que não houve alterações. Um exemplo simples seria a manipulação das texturas de um jogo, que são armazenadas como arquivos na pasta onde o jogo está instalado. Se um hacker adulterasse essa textura, poderia fazer com que objetos desaparecessem ou causasse um *glitch* no jogo, fazendo com que a *engine* processasse de maneira incorreta (SPARKLES, 2020b). Portanto, recomenda-se realizar uma verificação de integridade de todos os arquivos que são utilizados no jogo ao iniciá-lo, enviando os *hashes* ao servidor para validação dos

resultados, garantindo que o jogador está com uma instalação íntegra. Caso não esteja, alguma ação deverá ser tomada.

4.1.4 Identificação de Assinaturas Conhecidas

Da mesma forma que um pescador consegue observar características do clima e do mar para adivinhar que uma chuva está vindo, um *software* antitrapaça pode descobrir qual ameaça conhecida está enfrentando, observando características do sistema. O método de identificação de assinaturas nada mais é do que a observação de padrões que estão sendo repetidos, de forma que estes padrões sejam capazes de identificar uma ameaça.

Como mencionado anteriormente, existem diversas categorias de trapaças e elas possuem um modelo de distribuição variado. No caso das trapaças que sejam amplamente distribuídas em fóruns abertos, é fácil para um desenvolvedor de antitrapaças tê-la em mãos e coletar suas assinaturas. Além disso, por ser utilizado pela ampla maioria da comunidade, a identificação de suas assinaturas acaba sendo efetiva contra a maioria dos jogadores que estão utilizando o programa ilícito. As assinaturas que são utilizadas irão variar de acordo com a complexidade da implementação do sistema de identificação de assinaturas que for adotada pelos desenvolvedores, que podem fazer verificações simples através de *hashes* ou nomes de processos, ou identificações baseadas em comportamento (consumo de memória, curva de consumo do processador etc).

O funcionamento desses sistemas é relativamente simples de se compreender. Dado que o cliente do jogo está sendo executado na máquina do jogador, sabe-se que, ao lado desse cliente, estão diversos processos em paralelo que podem ser legítimos ou não. O *software* antitrapaça irá realizar um *scan* na máquina em busca de dados que possam auxiliar a identificação de programas ilícitos (nomes de processos, *hashes*, padrões de comportamento, estatísticas de execução etc) e enviar os dados ao servidor, que irá processar os dados e compará-los com o que se tem de informações disponíveis sobre trapaças mantidas em uma base do sistema antitrapaça. Caso sejam identificados padrões conhecidos, é possível tomar alguma ação contra o jogador, desde encerrar a execução do jogo ou banir o usuário (BALFE; MOHAMMED, 2007).

Deve-se ter em mente que esse método só é efetivo contra ameaças que já são conhecidas, uma vez que não é possível ter assinaturas de algo novo. Portanto representa uma boa proteção contra trapaças antigas ou populares, porém pode não causar impacto em ferramentas restritas e mais recentes. Além disso, esse método é fácil de ser burlado, uma vez, que tendo ciência de que o mesmo já é aplicado amplamente nos dias de hoje, os *hackers* podem desenvolver meios de gerar nomes de processos de forma aleatória, causar alterações intencionais no *hash* e outros tipos de estratégia que impedem uma identificação simples (HOUSE, 2018)(Steam Support, 2022).

Outro ponto importante é que aqui se tem o compartilhamento de alguns dados do jogador com o servidor, que é um começo para a discussão relacionada à privacidade do

usuário. Portanto, é importante que a empresa tenha esclarecido a forma com que esses dados serão armazenados e processados para apoiar o sistema antitrapaça, impedindo que ocorra alguma violação à privacidade (BALFE; MOHAMMED, 2007).

4.1.5 Ofuscação de Memória

Todos os programas que são executados no computador passam pela memória, portanto as informações que são necessárias para que os mesmos sejam devidamente executados também estão contidas lá, em algum lugar. O método de ofuscação de memória tem como objetivo impedir que *hackers* tenham facilidade para descobrir onde que se encontram as informações que são úteis para a realização de alguma trapaça. Quando um *software* é executado, diversas variáveis são dinamicamente alocadas na *heap*, que é o local da memória que possui a finalidade de armazenar esses valores. Identificar variáveis relevantes pode ser o ponto de entrada para se alcançar ainda mais informações posteriormente. No contexto de jogos, pode-se dizer que as variáveis vão armazenar valores como a quantidade de vida, quantidade de munição, velocidade do personagem, localização no jogo e mais (BISHT, 2022).

Durante a categorização das trapaças, na seção "Análise de memória e Engenharia Reversa", é mencionado um programa chamado *Cheat Engine*. A ofuscação de memória visa combater exatamente esse tipo de ataque ao cliente do jogo, utilizando técnicas como randomização e realocação de endereços, além de criptografar os valores das variáveis presentes na *heap*. A criptografia tem como objetivo ocultar as informações que estão armazenadas nas variáveis, de forma que, se o hacker não conhece o algoritmo que fez a encriptação, não será capaz de entender o que o valor que foi identificado representa dentro do jogo. Já a realocação de endereços faz com que as variáveis troquem de posição todas as vezes que seus valores são alterados, de forma a combater o *scan* de memória (SKILLZ, 2022).

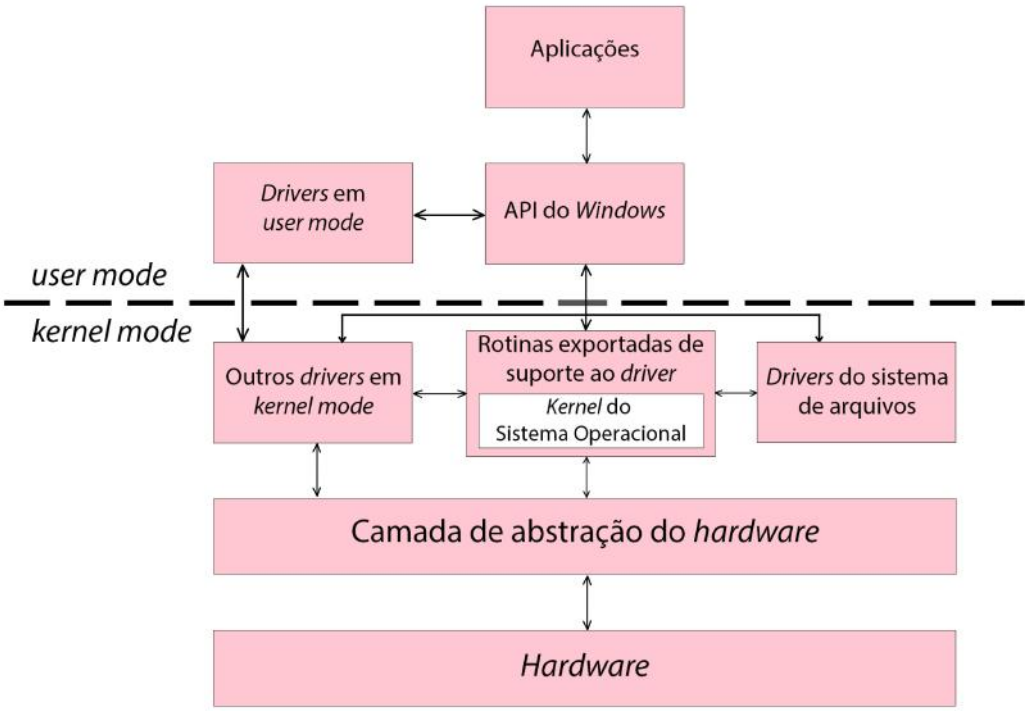
Um exemplo prático dessa implementação é mencionado no blog de tecnologia da desenvolvedora *Riot Games*. Além de explicar a filosofia adotada pela empresa no combate aos trapaceiros e quais são os principais desafios que enfrentam, explicam como realizaram a implementação dos métodos mencionados acima. Na seção *Protecting our Data*, mostram como é possível realizar a alteração na posição das variáveis na *heap* todas as vezes que os valores dessas variáveis são alterados, movendo-as para novos endereços que acabam por impedir o rastreio pelos *debuggers*. O processo de *scan* de memória é feito através de exclusão: olha-se para todos os endereços e, então, busca-se por aqueles que possuam o valor desejado; após isso, é realizada uma alteração intencional no valor do atributo desejado e é realizada uma nova busca, olhando dentre os resultados anteriores, qual endereço de memória possui o novo valor; a cada nova busca, as variáveis que não sofreram alterações de valor têm seus endereços de memória descartados. Como há uma troca de posição a cada atualização dos valores, fatalmente as variáveis chave (que se de-

seja descobrir) vão cair em regiões de memória já excluídas anteriormente e assim anulam a tentativa de rastreio por parte do *debugger*. É mencionada ainda a importância de não serem armazenados valores na memória sem criptografia. Se o jogador possui números como 100 de vida, 37 balas de munição, 158 moedas ou outros, salvar esses valores na forma de texto plano significa entregar ao desenvolvedor de trapaças a facilidade de realizar buscas diretas por eles. Deve-se sempre utilizar criptografia e, de preferência, alterar levemente os algoritmos criptográficos que serão aplicados sobre cada uma das variáveis (VANKUIPERS, 2018).

4.1.6 Antitrapaça em modo Kernel

Recentemente, nas comunidades de jogos competitivos *online*, as principais discussões envolvendo privacidade ocorrem devido ao grau de permissões requisitadas pelos sistemas antitrapaças modernos, que exigem execução em *kernel mode*. Antes de adentrar em mais detalhes, cabe esclarecer o que isso significa. Os sistemas operacionais modernos possuem dois modos de execução: *user mode* e *kernel mode*. Dependendo de qual é o processo que está sendo executado no sistema, o processador muda de contexto. Geralmente, processos oriundos de programas de escritório, jogos, *browsers* etc, são executados em *user mode*; já os componentes do sistema operacional e *drivers* tendem a serem executados em *kernel mode*.

Figura 1 – Comunicação entre componentes em *user-mode* e *kernel-mode*



Fonte: Viviano (2022)

Como pode ser observado pela figura 1, os contextos são segregados e a comunicação entre eles ocorre através da *Application Programming Interface* (API) do Sistema Operacional. Quando um processo é executado em *user mode*, é criado um espaço de endereço virtual e uma *handle table*. Cada processo é executado de maneira isolada, de forma que não tenham acesso aos espaços e dados uns dos outros. Além disso, por conta dos endereços de memória serem privados, o acesso de um processo em *user mode* aos endereços virtuais reservados pelo sistema operacional não é possível (VIVIANO, 2022).

Por outro lado, os processos que são executados em *kernel mode* compartilham um único espaço de endereçamento virtual, de forma que não são isolados entre si. Caso um *driver* falhe ou seja comprometido, ele poderá afetar os outros que estão sendo executados ao seu lado (VIVIANO, 2022). Uma explicação mais clara sobre a segregação de contextos que é realizada pelo sistema operacional pode ser ilustrada através dos Anéis de Proteção.

Anéis de Proteção, por definição, constituem um modelo de arquitetura que separa os níveis de interação que ocorrem no sistema operacional, para fornecer tolerância a falhas e proteção entre componentes, processos e aplicações, além de fornecer segurança (WILEY, 2011). Podem ser considerados, geralmente, quatro anéis numerados de 0 a 3, que possuem as características apresentadas na tabela 1:

Tabela 1 – Anéis de proteção

Anel #	**O que é executado lá**	**Observações**
Anel 3	Aplicações e programas	Nível de usuário
Anel 2	*Drivers* de entrada e saída e utilitários	Camada de abstração de *Hardware* (HAL)
Anel 1	Componentes do Sistema Operacional fora do *kernel*	Nível confiável
Anel 0	*Kernel* do Sistema operacional	Nível confiável

Fonte: Wiley (2011, pg. 988)

A comunicação entre cada uma das camadas é devidamente controlada, com os anéis se comunicando apenas com seus vizinhos. Além disso, os componentes que são executados em cada um dos anéis estão em contextos diferentes, com o núcleo do Sistema e os componentes (anéis 0 e 1) sendo executados em *kernel mode*, enquanto aplicações de usuário (anel 3) estão em *user mode*. *Drivers* e dispositivos de entrada e saída (anel 2) se utilizam de uma camada de abstração do *hardware*. Entendidas as definições sobre contextos de execução, cabe agora trazer as justificativas para a utilização de *kernel mode* por parte dos *softwares* antitrapaça, buscando-se compreender se é uma necessidade real ou não.

Como é possível observar ao longo deste trabalho, *hackers* vão até os mais baixos níveis técnicos para atingir o objetivo de identificar uma vulnerabilidade, que consequentemente permita uma forma de trapacear. Quando se trata de contexto de execução, por muitos

anos as trapaças já vêm sendo executadas a nível de *kernel*, tanto com o objetivo de não serem detectadas por *softwares* de segurança em *user mode*, quanto para que trapacear seja viável. Se uma ferramenta de trapaça não for capaz de ler e manipular a memória ou criar *hooks* para execução de determinadas funções, dificilmente será eficaz e funcional. Da mesma forma, se um antitrapaça não funciona em *kernel mode*, não há como ele garantir a integridade do *kernel* e, consequentemente, uma ferramenta de trapaça pode ser injetada no momento do *boot* do sistema, passando desapercebida pela ferramenta de segurança. Portanto, dentre as principais justificativas para que os antitrapaças possuam esse nível de privilégio, estão a necessidade de garantir a integridade do *kernel*, prevenção de manipulação indevida na memória, registrar a utilização de recursos do sistema e monitorar processos em execução (DAAX et al., 2020)(VMCALL, 2020).

Tabela 2 – Comparação de privilégios entre *user space* e *kernel space*

	User space	***Kernel space***
Acesso à memória	Acesso limitado	Acesso total
Acesso ao *hardware*	Sem acesso direto	Acesso total
Acesso às instruções da CPU	Apenas instruções não privilegiadas	Todas as instruções
Acesso a estruturas de dados críticas do SO	Sem acesso	Acesso total

Fonte: Lehtonen (2020, pg. 58)

A título de exemplo, o *Vanguard*, antitrapaça da *Riot Games* criado para o jogo *Valorant*, foi o centro das discussões na comunidade de jogadores em 2020 justamente por conta do seu nível de permissão e seus requisitos. Para que o usuário pudesse jogar, era necessário que o *Vanguard* não só estivesse instalado e fosse executado em *kernel mode*, mas deveria ser executado junto ao *boot* do sistema. Essa é uma medida tomada justamente para impedir que ocorresse alguma adulteração antes que o *antitrapaça* estivesse em execução (DOLPHINWHACKER, 2020)(RUIA, 2021).

De maneira prática, a implementação de antitrapaças em *kernel mode* cria a capacidade de interceptar tentativas de acesso indevido ao cliente do jogo, além da possibilidade de escanear a memória do sistema em busca de *softwares* indevidos. Outro ponto chave é monitorar a utilização de funções da API do Sistema que são comumente utilizados por essas trapaças, como os métodos de escrita em memória, criação de *threads* e *hooks* (LEHTONEN, 2020).

Uma das dificuldades que fatalmente são enfrentadas por esses antitrapaças é o fato de que os desenvolvedores de trapaças também podem escrever seus programas para funcionarem em *kernel mode* (e já o faziam antes dos *antitrapaças* começarem a atuar nesse nível), o que dificulta a detecção e concede ao *hacker* as mesmas ferramentas que o antitrapaça possui. Sistemas operacionais como o *Windows* e *macOS* colocam uma condição que

exige que os *kernel drivers* sejam digitalmente assinados, de forma que o desenvolvedor precisa de um certificado para essa assinatura, obtido através da aplicação do programa para a *Apple* e *Microsoft* (HUDEK et al., 2022)(Apple Inc, 2015). Embora existam meios de evadir essa defesa, essa é uma outra barreira colocada contra os trapaceiros (TUNG, 2022).

Em resumo, embora a implementação de antitrapaças em *kernel mode* seja recomendada por conta da necessidade (igualando os poderes entre "protetores e trapaceiros") e dos benefícios que eles trazem, é necessário ressaltar que exigem profundo conhecimento de quem o implementa, visto que por possuírem tamanho grau de permissões, possuem acesso à todas as informações do usuário e uma implementação mal feita pode colocar em risco a sua segurança (STOLYAROV; LARIN, 2018). Devem também as empresas que desenvolvem esses sistemas se comprometerem com o respeito aos dados do usuário, bem como atestarem o comprometimento de seus colaboradores com a ética, para que não ocorra desvio de finalidade com os dados coletados pela aplicação e seu funcionamento.

4.2 LADO DO SERVIDOR

Ao contrário da seção anterior que abordava os métodos antitrapaça que são implementados na máquina do jogador, nesta seção são abordados os mecanismos que podem ser utilizados unicamente no servidor. Para que haja um servidor, é preciso que o jogo tenha alguma comunicação com a Internet, que tende a ser o caso dos jogos *multiplayer* online. Ainda que existam alguns jogos *singleplayer*, como o modo história da franquia *Call of Duty*, que exigem a comunicação constante com a Internet para serem jogados, esses casos costumam ser exclusivamente por questão de DRM (proteção contra pirataria) (COCA, 2019).

Como esses métodos antitrapaça não são implementados no cliente, eles não terão acesso a informações importantes que podem estar presentes na máquina do jogador e nem terão muito poder para impedir alguma execução suspeita ou realizar outro tipo de ação mais direta na máquina, ficando o servidor restrito a interromper a conexão ou tomar outra medida à nível de política, como suspender ou banir o jogador. O foco, quando se trata das ações realizadas no lado do servidor, está principalmente na garantia de uma comunicação segura entre cliente-servidor e na utilização de métodos estatísticos para detecção baseadas em padrões de comportamento.

4.2.1 Política de Confiança Zero no cliente

Quase que uma lei no desenvolvimento de jogos na arquitetura cliente-servidor, a não confiança no cliente é essencial, como será visto ao longo da seção. Por definição, a confiança zero exige que todas as informações recebidas por parte do cliente sejam verificadas e validadas, isso porque deve-se considerar que todas as informações que saíram

do mesmo podem ter sido adulteradas. Hoje, na área de Segurança da Informação, já é comum a implantação da Arquitetura de Confiança Zero em ambientes corporativos, que embora tenha um foco e definição completamente diferentes dos métodos antitrapaça, se assemelha através do cerne, onde a confiança nunca é garantida de forma implícita, mas precisa ser constantemente avaliada (KOSTER, 2022)(ROSE et al., 2020).

Além do fator de trapaça que pode estar presente nas informações que são recebidas pelo servidor, outro ponto chave está relacionado com a autoridade sobre a informação. Considerando um cenário legítimo, em que não está ocorrendo nenhuma tentativa de adulteração ou obtenção de vantagem por parte dos jogadores em uma partida, é possível que os clientes desses jogadores possuam informações conflitantes. Isso pode ser ilustrado de diversas formas, como, por exemplo, em um jogo de tiro multijogador. Caso o jogador X realize um disparo e acerte o jogador Y, o mesmo deveria receber certa quantidade de dano. O cliente do jogador que realizou o disparo informa ao servidor que o projétil acertou o alvo, porém o cliente do jogador Y não reconhece esse disparo por algum problema de comunicação ou pela latência que é própria da conexão. Nesse caso, quem possui a informação correta? Pode-se dizer que o responsável pela decisão será quem possui a autoridade da informação.

O exemplo acima também se relaciona com o *netcode*, que implementa a forma com que as informações trafegam entre os computadores dos jogadores e como o atraso no envio e recebimento dessas informações impactam no jogo. Para que a política de confiança zero seja implementada na arquitetura cliente-servidor, é preciso que o servidor onde está sendo processada a partida seja autoritativo, sendo ele o responsável pelas tomadas de decisões diante de conflitos que possam ocorrer com as informações. Embora o foco seja no combate à trapaça, essa é uma questão de extrema relevância para o projeto de jogos *multiplayer* porque dita o funcionamento do jogo, bem como a experiência do jogador (PUSCH, 2019)(DEWET; STRAILY, 2019).

Existem alguns cenários que podem justificar a implementação de uma arquitetura com cliente autoritativo, embora essa não seja a solução ideal. Considerando o projeto de um jogo de simulação de batalhas, e a depender da quantidade de cálculos e informações que compõem o jogo, é possível que o tráfego de rede acabe ficando sobrecarregado com a troca de dados entre cliente-servidor, e isso pode ser um impeditivo para que algumas regiões do Mundo tenham acesso à esse jogo (em locais com Internet de baixa qualidade ou alto custo por consumo). Uma saída para diminuir o impacto na rede seria realizar os cálculos e o processamento diretamente do dispositivo do usuário e enviar somente o resultado ao servidor, de forma que não haja necessidade dessa troca de informações durante todo o processo de simulação e, consequentemente, mitigue-se significativamente o impacto na rede (LEHTONEN, 2020). Mesmo que essa escolha de projeto abra uma brecha para trapaças, é possível que ter um cliente autoritativo seja uma decisão de mercado.

De toda forma, a recomendação máxima é que sempre se tenha um servidor capaz de identificar o estado do jogo e validar as informações que estão sendo recebidas, buscando sempre ser o responsável por processar o que está ocorrendo no jogo. Esse método não só é extremamente eficaz por dificultar as trapaças relacionadas à manipulação do estado do jogo, como também não representa nenhum risco à privacidade do usuário (GAMBETTA, 2022).

4.2.2 Protocolo de Aplicação Seguro

Definir e implementar um protocolo de aplicação seguro está entre os maiores desafios encontrados no desenvolvimento de jogos *multiplayer*, tanto porque afeta tremendamente a experiência do jogador, quanto por ser um ponto chave no combate a diversos tipos de trapaças. O desenvolvimento de um protocolo robusto tem como objetivo mitigar alguns tipos de ataques clássicos que foram mencionados anteriormente na seção Adulteração de Pacotes, como o *spoofing*, *replay attacks* e outros.

Entende-se, por protocolo de aplicação, o responsável direto por estabelecer os contratos que definem como será a comunicação entre o cliente e o servidor do jogo. É nele que estão especificados os dados que irão trafegar, a forma como serão estruturados, bem como as condições para que os pacotes sejam recebidos e como serão tratados pelo cliente e servidor do jogo. Como o desenvolvimento de jogos pode demandar algumas necessidades específicas, busca-se sempre melhorias focadas em aprimorar o desempenho do jogo e favorecer uma melhor experiência do jogador.

Uma técnica comum para a obtenção de vantagens é o atraso intencional de pacotes, onde o trapaceiro deixa de enviar atualizações sobre seu estado no jogo de maneira temporária e, nesse intervalo de tempo, se movimenta ou realiza outras ações. Depois de um tempo, é realizado o envio dos pacotes com as atualizações do estado do jogador, e isso faz com que o mesmo se teleporte para outra região no jogo. Essa trapaça recebe o nome de *lag switch* caso seja realizada de forma intencional e tende a ser mais comum em jogos com arquitetura *peer-to-peer* (MAARIO et al., 2021). Entretanto, também é possível que o jogador esteja de fato sofrendo com *lag*, esse fenômeno se chamaria *Rubber Banding* (EXPERTCODER14; IMULSION, 2016). De toda maneira, o que define a forma como o jogo irá se comportar é o protocolo que foi implementado, por isso sua importância é tão grande.

Em (BAUGHMAN; LEVINE, 2001) são exploradas algumas maneiras de trapacear tanto em jogos que possuem uma arquitetura cliente-servidor, quanto em jogos *peer-to-peer*, que possuem uma arquitetura distribuída. A partir dessas trapaças conhecidas, os autores propõem um conjunto de soluções e, posteriormente, um protocolo com garantias antitrapaça. Nesse protocolo, é estabelecido que a comunicação de todas as ações dos jogadores devem ser enviadas primeiramente como um *hash*. Quando todos os jogadores já tiverem enviado suas ações, eles farão o reenvio dessas ações em texto plano, de forma

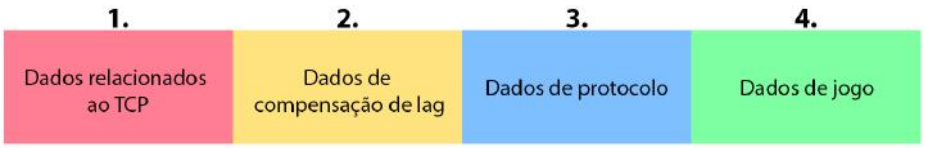
que o servidor possa validar que a ação permanece igual a primeira (enviada como *hash*), garantindo que não houve adulteração. Isso é efetivo para combater trapaças em jogos onde o jogador ser o último a tomar uma ação (após saber as ações dos outros jogadores) possa ser vantajoso (BAUGHMAN; LEVINE, 2001).

Além do protocolo de aplicação, é importante que os desenvolvedores tomem uma decisão correta em relação ao protocolo que será utilizado na camada de transporte. Como se sabe, o TCP é um protocolo orientado à conexão (POSTEL et al., 1981) que embora traga confiabilidade, pode causar *delay* em alguns gêneros de jogos, devido a troca contínua de informações entre o cliente e servidor e as possíveis perdas ou atrasos de pacotes. O impacto do TCP em jogos de turnos não seria notado, por exemplo, uma vez que as ações ocorrem em períodos de tempo mais longos e a confiabilidade no registro de ação do jogador é relevante. Já em jogos de tiro ou MOBA, o atraso em um pacote faria com que os subsequentes fossem bloqueados e, consequentemente, causaria demora na atualização de estado do jogo, prejudicando a experiência do jogador. Esse não seria um problema enfrentado no UDP, que é um protocolo orientado à transação e não possui confiabilidade em relação a entrega dos *datagramas*. Ao contrário do TCP, no UDP não há mecanismos de retransmissão de pacotes e controle de congestionamento. O UDP costuma ser bastante utilizado para *streaming* de vídeo, por exemplo, uma vez que, em caso de perda de um pacote, irá ocorrer somente uma redução na qualidade do vídeo e isso não causará atraso, com o vídeo continuando a ser processado no tempo correto, sem congelar a reprodução até que todos os pacotes que compõem um quadro cheguem ao dispositivo do usuário. O impacto nos jogos dos gêneros mencionados acima é parecido. Embora perder alguns pacotes possa ser perceptível ao jogador, sua experiência de jogo continua sendo viável - o que não seria se a implementação fosse com TCP (POSTEL, 1980)(BISHT, 2022).

Nos jogos "de ação", a escolha comum é a utilização do protocolo UDP, com a implementação de alguns recursos adicionais por parte do desenvolvedor que visa aproveitar benefícios presentes no TCP. A implementação do TCP, por exemplo, já oferece algumas funcionalidades que são diretamente efetivas contra algumas trapaças, como o *replay attack*, dado que uma vez observado o número de sequência do pacote, será notado que o mesmo já teria sido recebido e, portanto, o pacote repetido seria descartado. No caso do UDP essa verificação não existe e precisaria ser implementada pelo desenvolvedor (LEHTONEN, 2020). Na figura 1 pode ser observada uma possível estrutura do pacote UDP.

Conforme colocado por (LEHTONEN, 2020), essa estrutura visa implementar uma funcionalidade similar ao TCP no início do *datagrama* UDP, transmitindo alguns dos dados que são passados no protocolo TCP (1). A segunda parte do *datagrama* (2) contém dados relacionados ao tempo de envio, sendo utilizado para cálculos de atraso e previsão de movimentos. A terceira parte (3) contém outros dados relacionados ao protocolo que são utilizados pelos desenvolvedores. A última parte do *datagrama* (4) contém os dados

Figura 2 – Possível estrutura de pacote UDP em jogos online



Fonte: Lehtonen (2020, p. 20)

do jogo em si, com o envio das informações que são processadas no cliente e servidor.

É importante que os desenvolvedores sejam capazes de checar corretamente os cabeçalhos de pacotes que estão chegando no servidor, porque esses são locais em que se buscam vulnerabilidades. Se não estiverem ocorrendo as devidas validações, é possível fazer com que a *engine* do jogo carregue um pacote controlado pelo atacante em locais arbitrários ou abuse de alguma vulnerabilidade conhecida (p. ex. *overflow de heap*) (CLASP, 2006), através da manipulação de sequências de pacotes fragmentados. Isso pode permitir a execução remota de código, que é uma vulnerabilidade de severidade alta, por permitir a exploração mesmo quando o atacante não possui acesso prévio ao sistema, e concedendo a capacidade de roubar dados, instalar *malwares*, movimentar-se através da rede e muito mais (AURIEMMA; FERRANTE, 2013).

Fica evidente, portanto, o quão importante é a tomada de decisão por parte do time de desenvolvimento tanto na escolha de protocolo, quanto na sua implementação, dado que um planejamento e execução corretos não só dificultam a trapaça, como também melhoram a experiência do jogador.

4.2.2.1 Comunicação Criptografada

Ainda no que se refere a protocolos, é importante destacar que toda a comunicação entre cliente e servidor deve ser criptografada, tanto pela segurança do usuário quanto para dificultar tentativas de abuso e manipulação dos pacotes *on the fly*. Nos anos 2000, com a ampla utilização do protocolo HTTP para aplicações sensíveis, foi criada a RFC 2818, que descreve como deve ser utilizado HTTP sobre TLS, sendo o TLS (*Transport Layer Security*) um protocolo de segurança para a camada de transporte que visa permitir uma comunicação segura e íntegra entre duas aplicações (DIERKS; ALLEN, 1999). Essas implementações foram realizadas para evitar ataques do tipo *man-in-the-middle* (MITM), que consiste em um ator não autorizado se colocar no meio da comunicação entre aplicações ou dispositivos, sendo capaz de visualizar, capturar ou adulterar os dados que estão sendo transmitidos (PLOVER, 2006). Outro ponto importante é que realizar a encriptação dos pacotes no momento em que são construídos no cliente é de extrema relevância para que o *hacker* não seja capaz de adulterá-los.

Essa preocupação de realizar a ocultação dos dados que são recebidos e enviados,

falando-se exclusivamente no contexto de antitrapaça, se dá uma vez que os *hackers* podem tanto realizar uma adulteração nos dados que serão enviados (já mencionado), como também acabar tendo acesso a informações que deveriam estar ocultas, sendo utilizadas somente para processamento pelo cliente do jogo. Para exemplificar, alguns jogos podem realizar o envio de informações como a localização de outros jogadores para projetar algum som ou outro tipo de ação com tal informação, porém isso será utilizado somente pelo cliente. Caso o *hacker* consiga acesso a esses dados, pode utilizá-los para mapear a posição dos outros jogadores dentro do jogo, por exemplo.

4.2.3 Métodos Estatísticos

Dentre as diversas definições e subáreas presentes no campo de estudo de probabilidade e estatística, pode-se resumir que o objetivo principal é ensinar e fornecer mecanismos para que seja possível tomar decisões inteligentes, num cenário em que há incertezas e variações, através da utilização de informações obtidas a partir dos dados conhecidos (DEVORE, 2011). No contexto de antitrapaças, o uso dessas disciplinas é feito geralmente sobre os dados gerados por ações do usuário, ou por outras atividades, que ocorrem na máquina do mesmo. Como será visto ao longo da seção, os métodos estatísticos implementados para o combate de trapaças não tendem a ser utilizados de maneira isolada, sendo assim combinados com outras técnicas para facilitar a tomada de decisão, apoiando principalmente a identificação de um trapaceiro, ao invés de atuar preventivamente.

Antes de apresentar algumas técnicas em si, é interessante começar fazendo menção a alguns dos aspectos negativos presentes no método estatístico. Como se sabe, para realizar simulações, identificar padrões (e desvios) de comportamento e mais, é preciso possuir amostras. Caso um time de desenvolvimento construa um sistema antitrapaça baseado em comportamento, por exemplo, será preciso que eles coletem muitos dados por um tempo até que seja possível testar e validar o modelo. Só após ter o modelo testado será possível identificar pontos fora da curva. Além disso, a construção de modelos e implementação desses métodos tende a demandar um conhecimento mais especializado em estatística, de forma que pode ser preciso buscar pessoas com conhecimento fora da área de desenvolvimento, no caso de profissionais da área de ciência e análise de dados (MAIER, 2021).

Visto que os métodos estatísticos trabalham com probabilidade e não é nem um pouco recomendável tomar uma decisão que prejudique o usuário diante de um cenário de incerteza, geralmente é comum que esses métodos funcionem como suporte, deixando pendente algum tipo de revisão humana. Isso é um problema, principalmente por conta de recursos humanos, já que é inviável que a empresa contrate pessoas o suficiente para analisar todos os jogadores considerados suspeitos, principalmente em grandes franquias de jogos.

Uma das tentativas de minimizar essa impossibilidade de revisão de todos os jogadores suspeitos é a utilização de sistemas de denúncias, que estão presentes em quase todos os

jogos online que permitam a interação entre jogadores. As denúncias podem ser feitas tanto por questões de comportamento e toxicidade, quanto por utilização de trapaças e abuso de *bugs*, podendo ser utilizadas como uma fonte de informação para confirmar a suspeição sobre determinados jogadores.

Embora não seja tão comum, também já foram implementados pelas desenvolvedoras alguns sistemas semelhantes a um tribunal, mais especificamente um "júri popular". No *League of Legends* esse sistema levava o nome de *O Tribunal* e tinha como objetivo julgar questões comportamentais, utilizando-se da própria comunidade para avaliação de desrespeito aos Termos de Uso do jogo (Riot Games, 2013); esse sistema não está mais ativo. Um outro sistema que permanece ativo ainda hoje é o *Overwatch*, do *Counter-Strike: Global Offensive*, que tem como foco o combate direto aos trapaceiros. Nesse sistema são apresentados ao usuário (investigador) pequenos trechos da partida na qual o jogador foi denunciado e considerado suspeito. O jogador que está avaliando irá observar as evidências e, ao final, emitir uma opinião sobre o que foi possível observar a partir das evidências fornecidas. Existem diversos critérios ponderados para auxiliar na decisão da desenvolvedora de punir o usuário ou não, como por exemplo a qualidade do investigador. Caso um usuário tenha uma alta taxa de acertos em seus julgamentos (que significa acertar, na maioria das vezes, o resultado final), o peso do voto dele será maior do que o de um usuário que possui menos experiência ou taxa de acerto. Esse sistema oferecia benefícios de experiência aos jogadores para incentivar a participação popular (Counter-Strike Blog, 2013).

Falando sobre os modelos de *software* antitrapaça baseados em dados, pode-se afirmar que a sua utilização está cada vez mais comum, principalmente pelo oferecimento de alguns serviços de antitrapaça externos à desenvolvedora do jogo, que já possuem os modelos e dados prontos para serem utilizados, funcionando quase que de forma *plug and play*, como é o caso do *Easy Anti-Cheat*. Não somente o fornecimento de *Anti-cheat as a Service* traz facilidade, como também não representa riscos ao desempenho do jogo, já que estão implementados completamente no lado do servidor. O ponto negativo, como será visto posteriormente, é quando jogadores muito habilidosos acabam cruzando a linha de detecção de trapaceiros (ALLAYES SIMON; RAUTAVA, 2016).

Em (ZHANG, 2021), são propostas soluções baseadas em *deep learning* para melhorar o combate às trapaças em jogos online, através da aplicação de modelos de inteligência artificial. Dentre as sugestões, estão a análise de racionalidade do jogador baseado nos seus dados, que propõe observar valores como nível de vida, localização no mapa, movimentação do mouse e dados do tipo, tentando definir se ele está agindo conforme o esperado ou não. Uma segunda sugestão é analisar a racionalidade do jogador baseado na sua perspectiva de visão. Nesse segundo modelo se buscaria validar as ações do usuário simulando o que o mesmo está observando em sua tela.

Outra sugestão é apresentada em (KAISER; FENG; SCHLUESSLER, 2009) e tem

como objetivo ser uma contramedida ao fato de que os trapaceiros possuem a máquina do cliente, podendo agir sobre a mesma para evitar a detecção. O *Fides* é um sistema que visa utilizar emulação do cliente para detecção de trapaças remotamente, baseado na observação de anomalias. Conforme apresentado no estudo, o sistema realiza uma auditoria no cliente e valida algumas medições, comparando-as com os valores que são esperados. Os valores gerados no lado do cliente são enviados a um controlador, que é o responsável por decidir o que será medido e quando isso ocorrerá, realizando a validação com emulação do cliente e comprovação do servidor.

Também visando realizar uma identificação através de anomalias, (YEUNG et al., 2006) propõem uma solução eficiente e escalável que utiliza uma metodologia baseada em *Redes Bayesianas Dinâmicas*, que observa unicamente os estados do jogo e é executada somente sobre o servidor. Conforme explicado ao longo do artigo, são consideradas variáveis aleatórias como a probabilidade de um jogador ser trapaceiro, o jogador estar em movimento ou não, o jogador inimigo estar em movimento ou não, a direção da mira e a distância entre o jogador e o alvo em que está mirando (é utilizado um jogo do gênero FPS no estudo). Com essas variáveis, busca-se a distribuição de probabilidade da precisão da mira, com o modelo do ato de mirar sendo considerado um processo de *Markov* de primeira ordem (onde cada estado subsequente dependerá exclusivamente do estado imediatamente anterior).

Na publicação (PUNEGOV, 2021), são apresentados por *Victor Punegov*, programador à frente de times focados em antitrapaça, alguns detalhes importantes na metodologia utilizada para solucionar o problema com trapaceiros. No que se refere especificamente aos métodos estatísticos, o mesmo afirma que não é uma tarefa fácil identificar um trapaceiro por suas ações e estatísticas no jogo (exceto em casos crassos). E como nem sempre é possível ter certeza sobre o indivíduo, eles não podem ser banidos, porque se corre o risco de punir um inocente. Presume-se, então, que o banimento deve ser realizado somente em casos que possuem vestígios claros e evidentes da utilização de algo ilícito.

Para ilustrar o risco que um banimento sem certeza representa, pode-se utilizar uma situação real, que ocorreu com um jogador conhecido como Erick "aspas" Santos, que foi banido pela plataforma *Gamers Club*, em 2019, por acusação de ser trapaceiro. O mesmo atuava no cenário amador de *Counter-Strike* naquela época. Com o lançamento do *Valorant*, outro título de FPS, migrou para o jogo e continuou a levantar suspeitas por conta do nível acima do normal. Nele, entretanto, passou a atuar no cenário competitivo e sagrou-se Campeão Mundial em 2022, conquistando o título para o Brasil e defendendo a sua integridade, sendo colocado pelos analistas como um dos melhores jogadores do Mundo (VHPORTO, 2022).

Tem-se, portanto, que os métodos estatísticos são sim um grande poderio e auxiliam fortemente no processo de identificação e confirmação de trapaça, principalmente porque, ao contrário dos outros métodos, são difíceis de serem contornados pelos *hackers*. Embora

possuam algumas dificuldades para implementação, podem ser integrados facilmente aos jogos por serem executados somente no lado do servidor, o que não causa problemas de performance. Caso os métodos estatísticos não capturem dados sensíveis, também não representam risco à privacidade do usuário.

5 PRIVACIDADE E INTEGRIDADE COMPETITIVA

Nos últimos anos a palavra Privacidade tem sido fonte de não só muitas discussões públicas, documentários e brigas judiciais, mas tem despertado também nos usuários uma curiosidade acerca da utilização dos seus dados e quais são as reais necessidades de sua coleta. Pode-se dizer que uma das grandes causas dessa preocupação com a privacidade na Internet é fruto do escândalo envolvendo o *Facebook* e a empresa *Cambridge Analytica* ocorrido em 2018, após a descoberta de que dados de mais de 50 milhões de pessoas haviam sido utilizados sem o consentimento delas, com a finalidade de predizer e influenciar a escolha dos eleitores nas eleições dos Estados Unidos em 2016 (BBC, 2018).

Após o episódio, intensificaram-se os estudos que buscam esclarecer qual é o possível impacto da Internet na vida e saúde dos seres humanos, em especial o impacto que pode ser causado pelas redes sociais. No artigo (RODRÍGUEZ-PÉREZ et al., 2020) publicado pelo MIT (*Massachusetts Institute of Technology*) e adaptado pela Folha de São Paulo (ZUCKERMAN, 2017), é possível observar como os algoritmos utilizados pelas redes sociais para a distribuição de conteúdo selecionado aos usuários acaba criando bolhas ideológicas, que tendem a ser inacessíveis à quem pensa de forma diferente e aumenta o sentimento de pertencimento de grupos identitários. Além disso, favorece a disseminação de notícias falsas, tornando possível a manipulação de grandes massas que estão inseridas nessas bolhas e muitas das vezes estão fechadas à realidade, como mostra o documentário Privacidade Hackeada, da *Netflix*. Esse documentário mostra, inclusive, como o aumento das taxas de depressão e suicídio em adolescentes estão relacionadas ao uso indiscriminado das redes (AMER; NOUJAIM, 2019).

Com a massificação da Internet, os governos ao redor do Mundo consideraram necessário realizar a revisão ou criação de novas regulamentações focadas diretamente na proteção dos dados, como o Regulamento Geral sobre a Proteção de Dados, que entrou em vigor na Europa no ano de 2018 (EUROPEIA, 2016), forçando as empresas a se adaptarem a novas regras que oferecem mais garantias e controle aos usuários com seus dados, sob pena de multas e sanções em caso de descumprimento. No Brasil, a Lei Geral de Proteção de Dados (LGPD) foi publicada em 2018 e teve a maior parte de seus artigos entrando em vigor em 2020 (GOVERNO FEDERAL, 2019), sendo aplicada a qualquer organização que processe, de alguma forma, dados pessoais no Brasil. Dentre os principais pontos da LGPD estão a exigência de transparência quanto ao uso dos dados, garantindo aos cidadãos entenderem como suas informações são tratadas, armazenadas e qual é a finalidade, sendo a Autoridade Nacional de Proteção de Dados (ANPD) a responsável pela fiscalização (SENADO, 2021).

Assim como todas as outras empresas que trabalham com dados precisaram fazer, as desenvolvedoras e distribuidoras de jogos eletrônicos tiveram que atualizar as suas políticas

de privacidade para respeitar as regulamentações vigentes. A própria *Riot Games*, já mencionada anteriormente ao longo do trabalho, descreve em seu site oficial os detalhes sobre o uso de dados do usuário (Riot Games, 2021). Um dos pontos da Política trata especificamente da forma com que os dados de usuários são utilizados para antitrapaça e prevenção a fraudes, que é o ponto de interesse deste trabalho.

> "Nossos Termos de Serviço proíbem estritamente o uso de programas de terceiros não autorizados que interagem com os Serviços da *Riot*, inclusive *mods, hacks, cheats, scripts, bots*, treinadores e programas de automação. Quando você cria ou usa uma conta da *Riot* Games, compra coisas de nós, joga nossos jogos ou interage com os Serviços da *Riot*, eventualmente usamos tecnologias anti-fraude e de prevenção contra trapaças (como *softwares* antitrapaça que pode ser executado no *background* (fundo) do seu dispositivo) que podem tomar decisões automatizadas (como suspensões temporárias ou permanentes da conta, restrições de comunicação, remoção de conteúdo ou acesso limitado ao conteúdo do jogo) com base nos dados que coletamos de você ou sobre você (consulte Dados que Coletamos). Nós participamos dessas atividades a fim de gerenciar nosso relacionamento contratual com você, cumprir uma obrigação legal (por exemplo, manter os Serviços da *Riot* em segurança) e/ou porque temos um interesse legítimo (por exemplo, entender como os Serviços da *Riot* são utilizados, como mantê-los protegidos e como aprimorá-los)."

Dentre as diversas formas de utilização, é mencionado o desenvolvimento de modelos de padrões de comportamento inadequado, usados como indicativos. É mencionado o uso de ferramentas automatizadas para avaliar esses padrões e um sistema de classificação para observar as variações nas ações ao longo do tempo. Além disso, mencionam que o objetivo principal é garantir o respeito das regras e políticas, protegendo os Serviços e inibindo ações indevidas, bem como a garantia de integridade competitiva em seus jogos.

Conforme foi trazido ao longo deste trabalho, alguns dos métodos antitrapaça demandam a coleta de informações do usuário para realizarem suas tarefas. No caso dos métodos estatísticos, são coletadas principalmente informações relacionadas ao comportamento do usuário dentro dos jogos, enquanto outros métodos, como o de identificação de assinaturas, atua diretamente na máquina do jogador e envia ao servidor informações relacionadas a processos ou outras atividades identificadas lá. Como as desenvolvedoras de antitrapaças não são totalmente transparentes quanto ao que está está sendo coletado e como está sendo utilizado, justamente para não entregar dicas dos seus métodos para os trapaceiros, as preocupações podem acabar sendo elevadas para alguns jogadores mais desconfiados.

Em 2020, com o lançamento do *Valorant*, título de FPS da *Riot Games*, foi apresentado à comunidade o *Vanguard*, sistema anti-cheat proprietário que teria como principal objetivo a garantia de integridade competitiva para os jogadores, conforme abordado pela empresa, ainda na fase de desenvolvimento do jogo (Riot Games, 2019). Durante o lançamento, entretanto, houve muitas discussões por conta do nível de privilégio que o *software*

demandava e, mais do que isso, diversos tipos de problemas foram relatados pela comunidade de forma geral (SHUN-PIE, 2020). Em um relato no *Reddit*, por exemplo, um usuário afirmou que a ferramenta não deveria desabilitar programas por conta própria, sem antes perguntar ao usuário. Isso porque o antitrapaça desabilitou o sistema de resfriamento do seu computador, ao bloquear o *software* que era utilizado com essa finalidade por considerá-lo suspeito (HUGOMETEO, 2020).

O fato do antitrapaça funcionar a nível de *kernel* foi justamente o que permitiu que fosse tomada a ação de bloqueio de outro *software*, conforme relatado acima, e foi um dos principais fatores que fez com que a comunidade se dividisse sobre o quanto essa ação é válida, também levantando pontos relacionados ao impacto que poderia ser causado na privacidade do jogador. Embora as preocupações sejam legítimas e o pânico possa acabar se espalhando, de certa forma, pela falta de conhecimento, é fato que não é intenção da empresa causar danos aos usuários, justamente porque isso pode afetar o crescimento do jogo (MELO, 2020) e causar um grande prejuízo a desenvolvedora.

Como foi apresentado neste trabalho, a necessidade do antitrapaça ser executado a nível de *kernel* se deve ao fato de que os desenvolvedores precisam ter o mesmo poder que os atacantes, caso contrário seria como apagar fogo com papel. Esse poder que é dado às desenvolvedoras, porém, cobra um preço, que é o da privacidade. Quando um jogador aceita que um *software* de terceiro seja instalado em seu dispositivo, ele está depositando a sua confiança nessa ferramenta e, no caso do antitrapaça, acredita que ele será utilizado unicamente para a finalidade que possui, que é o combate às trapaças.

Embora a ampla maioria das reclamações sejam oriundas de problemas técnicos, a parcela de jogadores que possui consciência quanto à utilização dos seus dados pessoais e teme ações contra sua privacidade, naturalmente terá preocupação ao conceder acesso em seu dispositivo para um software que está sendo considerado tão invasivo. Um dos principais medos e que, de fato, representam um risco real, é o caso do *Vanguard* ter uma vulnerabilidade explorada. Para exemplificar, recentemente foi identificada uma vulnerabilidade em um *driver* da BANDAI NAMCO, desenvolvedora e publicadora de jogos, que permitia a execução de código a nível de *kernel* por conta de um *backdoor* que era aberto na máquina do usuário (STOLYAROV; LARIN, 2018).

A principal fraqueza nesse argumento, porém, é o fato de que qualquer *software* que é utilizado pelo usuário está sujeito a algum tipo de ataque e pode apresentar vulnerabilidades exploráveis, não sendo um problema exclusivo do antitrapaça (VMCALL, 2020). Todas as ações que um usuário realiza no computador e na Internet, seja navegar, utilizar serviços ou jogar, é oriundo de confiança. O usuário que instala o sistema operacional *Windows* em sua máquina, confia que a *Microsoft* fará um bom trabalho para fornecer segurança. O outro usuário, que realiza a instalação de alguma distribuição do *Linux*, confia que a comunidade ou empresa que mantém o projeto será responsável ao dar suporte e fornecer segurança ao sistema. O usuário que utiliza o *Google Chrome* para navegar

acredita que a *Google* manterá o navegador atualizado e livre de vulnerabilidades. Por fim, o usuário que instala o *Valorant* e cede espaço ao *Vanguard* em sua máquina, acredita que a *Riot* fará devidamente o trabalho de proteger o software e garantirá que seus desenvolvedores atuem de maneira ética e coletem somente os dados necessários para o trabalho contra trapaças.

Portanto, ao levantar a questão sobre colocar em risco a privacidade do jogador para a garantia de integridade competitiva, defendo que é uma necessidade. Mesmo que seja desagradável e não se deva negligenciar essa preocupação, cobrando sempre fiscalização e auditoria sobre os trabalhos da empresa em caso de ação suspeita, é inegável que a forma como as trapaças são desenvolvidas hoje demandam que os *softwares* antitrapaça tenham tais tipos de privilégios. A experiência de ter uma partida arruinada por conta de trapaceiros é extremamente desagradável e acaba por causar desânimo e frustração, sendo negativos não só para toda a base de jogadores, quanto também para os desenvolvedores que querem ver o jogo crescendo. Com o cenário de esportes eletrônicos ainda mais evoluídos, a integridade competitiva é essencial não só para que o jogo permaneça sendo jogado, como também não afete negativamente todo o ecossistema que está sendo construído ao redor do mesmo.

6 CONCLUSÃO

Neste trabalho foi realizado um grande levantamento acerca dos aspectos que envolvem as trapaças nos jogos digitais. O primeiro passo para expandir os estudos e pesquisa sobre o tema é entender que esse assunto não é simples e envolve questões que vão desde o âmbito mais individual (com utilização e criação de trapaças para algum tipo de satisfação pessoal), quanto a um cenário global, com a formação de um ecossistema econômico decorrente da produção e venda de trapaças, que gera ganhos consideráveis aos criadores e fornecedores e prejuízos para desenvolvedoras de jogos e jogadores, afetando diretamente os cenários competitivos dos jogos eletrônicos.

Inicialmente, buscou-se apresentar todos os atores que são parte direta da indústria de trapaças, com o objetivo de contextualizar o leitor acerca dos motivos que levam a uma trapaça ser criada. Essa primeira etapa foi importante para fornecer uma visão holística da área, dando compreensão quanto a existência da má-fé, boa-fé ou o simples interesse no retorno financeiro, ao colocar o fornecimento de trapaças como fonte de renda para parte dos grupos explorados neste trabalho.

Na sequência foram apresentadas as formas de trapaças presentes nos jogos digitais. De forma sucinta, foram abordadas algumas das funcionalidades adquiridas com as trapaças (recursos que são fornecidos ao usuário que as utiliza e de quais maneiras isso ocorre). Um ponto chave é entender que o conceito de trapaça em jogos digitais não se limita à utilização de ferramentas de terceiros para alterar o funcionamento do jogo, mas que o abuso intencional de falhas de implementação também é considerada uma trapaça. Neste trabalho, é relevante entender que a preocupação com os testes e implementações dos jogos é importante para garantir integridade competitiva aos jogadores, ao garantir que o funcionamento do jogo ocorra dentro do esperado. Em trabalhos futuros, pode-se abordar discussões quanto à culpabilidade de um jogador pela utilização de um *bug*, que tem se tornado um assunto relevante com o amadurecimento dos *esports*. No final de 2021, no *Valorant Champions* (campeonato mundial de *Valorant*), um time brasileiro teve sua vitória revertida ao sofrer punição por abusar de um *bug* (*exploit*). O grande problema, entretanto, é que o *bug* não estava descrito no livro de regras e o jogador afirmou não ter conhecimento de que a estratégia que ele utilizou era proibida. Nesse caso, a culpa é do jogador ou da desenvolvedora? É correta a aplicação da punição? Em casos ambíguos (onde não fica claro que um comportamento dentro do jogo é um *bug*), como se define uma solução justa durante a ocorrência desses casos em um torneio?

Tendo compreendido os atores e quais são as formas de trapacear, o trabalho passa então a abordar quais são os métodos antitrapaça mais comuns, apresentando alguns conceitos chave que devem ser adotados na tentativa de garantir que todos os jogadores sejam capazes de desfrutar dos jogos de maneira justa. Vale pontuar que o conteúdo

acadêmico sobre trapaças e métodos antitrapaça é bastante limitado, uma vez que o desenvolvimento das trapaças ocorre "por baixo dos panos" (muitas vezes em fóruns ou grupos fechados) e os trabalhos antitrapaça ocorrem de forma restrita dentro das empresas que desenvolvem esses sistemas. É, portanto, um desafio encontrar os conhecimentos que são desenvolvidos nessa área pela Indústria, uma vez que não são produzidos na mesma proporção no meio acadêmico.

Embora seja o capítulo mais técnico, buscou-se limitar o grau de profundidade que se atingiria em cada um dos métodos que foram apresentados para que o trabalho não fugisse do seu propósito de ser uma visão geral sobre o tema, englobando o máximo de tópicos correlatos possíveis. Entender o motivo da utilização de um método antitrapaça e o abuso que ele busca impedir é o suficiente para dar ao leitor um entendimento quanto às fraquezas e desafios que são enfrentados no combate às trapaças nos jogos digitais. Além disso, pode-se concluir que o "duelo" entre trapaceiros e desenvolvedoras de jogos sempre será uma perseguição entre "gato e rato".

Ainda durante a conceituação dos métodos antitrapaça, foi feita uma categorização dividindo-os entre implementações feitas no cliente e no servidor, apresentando um dos principais conceitos no desenvolvimento de jogos *online*, que é a arquitetura cliente-servidor. Essa divisão foi realizada com o objetivo de transmitir ao leitor o entendimento de que o escopo onde uma implementação é realizada possui extrema relevância, tanto pelas possibilidades de contramedidas disponíveis, quanto pelas questões que envolvem a "posse" do dispositivo do usuário.

Com esse entendimento em mãos, buscou-se levantar uma discussão acerca de como a privacidade do usuário pode ser colocada em risco e o quanto isso é necessário para as empresas que desenvolvem sistemas antitrapaça sejam capazes de fornecer aos jogadores um ambiente íntegro, sem o abuso de trapaças nos jogos eletrônicos. Embora o assunto seja complexo e envolva até mesmo questões legais (com respeito às legislações sobre dados de cada país), foi concluído que utilizar um jogo fornecido por uma desenvolvedora é uma escolha do usuário, que deve entender e decidir se aceita ou não as condições presentes nos termos de uso do mesmo. O que é possível garantir é que a utilização de métodos antitrapaça no dispositivo do usuário ainda é essencial na busca pelo fornecimento de um ambiente justo e íntegro à comunidade de jogadores.

Por fim, acredita-se que este trabalho tenha cumprido o propósito de ser uma fonte de informação relevante acerca de todo o cenário de trapaças em jogos digitais, podendo ser utilizado como ponto de partida para o desenvolvimento de estudos específicos sobre cada um dos tópicos e subtópicos que foram apresentados ao longo de seu desenvolvimento. Com a evolução dos algoritmos e técnicas de Inteligência Artificial e as suas aplicações em diversas áreas de conhecimento, é possível que métodos que foram vistos aqui fiquem obsoletos contra esses novos desafios, sendo essa uma grande recomendação de estudos futuros para a área de antitrapaças.

REFERÊNCIAS

3RD, D. E.; JONES, P. *Rfc3174: Us secure hash algorithm 1 (sha1)*. [S.l.]: RFC Editor, 2001.

ABNT. Abnt nbr iso/iec 27001 - tecnologia da informação - técnicas de segurança - sistemas de gestão de segurança da informação - requisitos. **ABNT**, 2006.

ACHTERBOSCH, L.; MILLER, C.; VAMPLEW, P. A taxonomy of griefer type by motivation in massively multiplayer online role-playing games. **Behaviour & Information Technology**, p. 846–860, 2021. Disponível em: https://www.marksgray.com/what-in-the-world-is-gatorcheats-and-why-did-they-just-agree-to-pay-2m/. Acesso em: 6 set.2022.

AKALTAR; GREENBEARD. **Limitations of p2p multiplayer games vs client-server**. [S.l.], 2013. Disponível em: https://gamedev.stackexchange.com/questions/67738/limitations-of-p2p-multiplayer-games-vs-client-server. Acesso em: 30 set.2022.

ALLAYES SIMON; RAUTAVA, A. **Anti-cheat for Multiplayer Games**. [S.l.], 2016. Disponível em: https://www.youtube.com/watch?v=hI7V60r7Jco. Acesso em: 25 ago.2022.

AMER, K.; NOUJAIM, J. **Privacidade Hackeada**. [S.l.], 2019. Disponível em: https://www.netflix.com/title/80117542. Acesso em: 1 nov.2022.

Apple Inc. **System Integrity Protection Guide**: Kernel extensions. [S.l.], 2015. Disponível em: https://developer.apple.com/library/archive/documentation/Security/Conceptual/System_Integrity_Protection_Guide/KernelExtensions/KernelExtensions.html. Acesso em: 16 dez.2022.

AURIEMMA, L.; FERRANTE, D. **Multiplayer Online Game Insecurity**. [S.l.], 2013. Disponível em: http://revuln.com/files/Ferrante_Auriemma_Multiplayer_Online_Games_Insecurity_WP.pdf.html/index.htm. Acesso em: 26 out.2022.

BALAKRISHNAN, A.; SCHULZE, C. Code obfuscation literature survey. **CS701 Construction of compilers**, v. 19, 2005.

BALFE, S.; MOHAMMED, A. Final fantasy–securing on-line gaming with trusted computing. In: SPRINGER. **International Conference on Autonomic and Trusted Computing**. [S.l.], 2007. p. 123–134.

BAUGHMAN, N.; LEVINE, B. Cheat-proof playout for centralized and distributed online games. In: **Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)**. [S.l.: s.n.], 2001. v. 1, p. 104–113 vol.1.

BBC. **Entenda o escândalo de uso político de dados que derrubou valor do Facebook e o colocou na mira de autoridades**. [S.l.], 2018. Disponível em: https://www.bbc.com/portuguese/internacional-43461751. Acesso em: 1 nov.2022.

BISHT, A. **Stack vs Heap Memory Allocation**. [S.l.], 2022. Disponível em: https://www.geeksforgeeks.org/stack-vs-heap-memory-allocation. Acesso em: 14 out.2022.

BROUGHAN, C. **What in the World Is GatorCheats and Why Did They Just Agree to Pay $2M?** [S.l.], 2021. Disponível em: https://www.marksgray.com/what-in-the-world-is-gatorcheats-and-why-did-they-just-agree-to-pay-2m/. Acesso em: 6 set.2022.

BUENO, A. **Caso aSpxgate**: Apoka fala punição. [S.l.], 2009. Disponível em: https://www.teamplay.com.br/noticias/counter-strike/2929-caso-aspxgate-apoka-fala-punicao. Acesso em: 20 nov.2022.

bugcrowd. **Public Bug Bounty Program List**. [S.l.], 2022. Disponível em: https://www.bugcrowd.com/bug-bounty-list/. Acesso em: 7 set.2022.

CAPPAERT, J. et al. Towards tamper resistant code encryption: Practice and experience. In: SPRINGER. **International Conference on Information Security Practice and Experience**. [S.l.], 2008. p. 86–100.

CARBONE, F. **Valorant: Riot Games abre processo contra desenvolvedora de cheat**. [S.l.], 2021. Disponível em: https://ge.globo.com/esports/valorant/noticia/valorant-riot-games-abre-processo-contra-desenvolvedora-de-cheat.ghtml. Acesso em: 5 set.2022.

CHAI, W. **hacker definition**. [S.l.], 2021. Disponível em: https://www.techtarget.com/searchsecurity/definition/hacker. Acesso em: 6 set.2022.

Cheat Engine. **About Cheat Engine**. [S.l.], 2022. Disponível em: https://www.cheatengine.org/aboutce.php. Acesso em: 23 set.2022.

CHRISTOPH, N. et al. Challenges in peer-to-peer gaming. **SIGCOMM Comput. Commun. Rev.**, v. 37, n. 1, p. 79–82, 2007.

CLASP. **CWE-122: Heap-based Buffer Overflow**. [S.l.], 2006. Disponível em: https://cwe.mitre.org/data/definitions/122.html. Acesso em: 26 out.2022.

COCA. **COD: Modern Warfare exige uma conexão constante à Internet no PC**. [S.l.], 2019. Disponível em: https://www.gamevicio.com/noticias/2019/10/cod-modern-warfare-exige-uma-conexao-constante-a-internet-no-pc/. Acesso em: 23 out.2022.

COCKFIELD, B. **Aimbot does it in hardware**. [S.l.], 2022. Disponível em: https://hackaday.com/2022/04/30/aimbot-does-it-in-hardware/. Acesso em: 10 set.2022.

CoinMarketCap. **CoinMarketCap Alexandria**: Bug exploit. [S.l.], 2022. Disponível em: https://coinmarketcap.com/alexandria/glossary/bug-exploit. Acesso em: 6 set.2022.

Counter-Strike Blog. **Overwatch FAQ**. [S.l.], 2013. Disponível em: https://blog.counter-strike.net/index.php/overwatch/. Acesso em: 28 out.2022.

CrowdStrike. **What is a spoofing attack?** [S.l.], 2022. Disponível em: https://www.crowdstrike.com/cybersecurity-101/spoofing-attacks/. Acesso em: 19 set.2022.

DAAX et al. **How anti-cheats detect system emulation**. [S.l.], 2020. Disponível em: https://secret.club/2020/04/13/how-anti-cheats-detect-system-emulation.html. Acesso em: 19 out.2022.

DEMARTINI, F. **Capcom lança DLC que libera todos os extras de Resident Evil 2**. [S.l.], 2019. Disponível em: https://canaltech.com.br/games/capcom-lanca-dlc-que-libera-todos-os-extras-de-resident-evil-2-136757/. Acesso em: 30 set.2022.

Denuvo by irdeto. **The global #1 Games Protection and Anti-Piracy technology helping game publishers and developers to secure PC, console and mobile games**. [S.l.], 2022. Disponível em: https://irdeto.com/denuvo/. Acesso em: 30 set.2022.

DEVORE, J. L. **Probability and Statistics for Engineering and the Sciences**. [S.l.]: Cengage learning, 2011.

DEWET, M.; STRAILY, D. **eeking into Valorant's Netcode**. [S.l.], 2019. Disponível em: https://technology.riotgames.com/news/peeking-valorants-netcode. Acesso em: 24 out.2022.

DIERKS, T.; ALLEN, C. The tls protocol version 1.0 (rfc 2246). **Internet Engineering Task Force**, 1999.

DOLPHINWHACKER. **Anticheat starts upon computer boot**. [S.l.], 2020. Disponível em: https://www.reddit.com/r/VALORANT/comments/fzxdl7/anticheat_starts_upon_computer_boot/. Acesso em: 20 out.2022.

DUTTON, F. **What is Indie?**: Diy developers discuss what it means to go solo, and whether the label really matters. [S.l.], 2022. Disponível em: https://www.eurogamer.net/what-is-indie. Acesso em: 10 set.2022.

Eletronic Arts. **Truques para The Sims 4**. [S.l.], 2022. Disponível em: https://www.ea.com/pt-br/games/the-sims/cheats. Acesso em: 30 set.2022.

EUROPEIA, P. E. e do Conselho da U. **REGULAMENTO (UE) 2016/679 DO PARLAMENTO EUROPEU E DO CONSELHO**: relativo à proteção das pessoas singulares no que diz respeito ao tratamento de dados pessoais e à livre circulação desses dados e que revoga a diretiva 95/46/ce (regulamento geral sobre a proteção de dados). [S.l.], 2016. Disponível em: https://eur-lex.europa.eu/legal-content/PT/TXT/HTML/?uri=CELEX:32016R0679. Acesso em: 2 nov.2022.

EXPERTCODER14; IMULSION. **What is "rubber banding"?** [S.l.], 2016. Disponível em: https://gaming.stackexchange.com/questions/250261/what-is-rubber-banding. Acesso em: 25 out.2022.

FallGuysGame. **The RISE and FALL of Cheater Island**. [S.l.], 2020. Disponível em: https://twitter.com/FallGuysGame/status/1305486780851007489. Acesso em: 10 set.2022.

GAMBETTA, G. **Fast-Paced Multiplayer (Part I): Client-server game Architecture**. [S.l.], 2022. Disponível em: https://www.gabrielgambetta.com/client-server-game-architecture.html. Acesso em: 24 out.2022.

GOVERNO FEDERAL. **Lei Nº 13.709, de 14 de Agosto de 2018**: Lei geral de proteção de dados pessoais (lgpd). [S.l.], 2019. Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm. Acesso em: 31 out.2022.

HOUSE, M. **Changing signature to avoid ban waves**. [S.l.], 2018. Disponível em: https://www.unknowncheats.me/forum/anti-cheat-bypass/283372-changing-signature-avoid-ban-waves.html. Acesso em: 12 out.2022.

HUDEK, T. et al. **Kernel-Mode Code Signing Requirements**. [S.l.], 2022. Disponível em: https://learn.microsoft.com/en-us/windows-hardware/drivers/install/kernel-mode-code-signing-requirements--windows-vista-and-later-. Acesso em: 16 dez.2022.

HUGOMETEO. **Reddit**: Vanguards needs to ask permission to disable a program instead of disabling it silently itself. [S.l.], 2020. Disponível em: https://www.reddit.com/r/VALORANT/comments/gek5rm/vanguards_needs_to_ask_permission_to_disable_a/. Acesso em: 3 nov.2022.

KAISER, E.; FENG, W.-c.; SCHLUESSLER, T. Fides: Remote anomaly-based cheat detection using client emulation. In: **Proceedings of the 16th ACM conference on Computer and communications security**. [S.l.: s.n.], 2009. p. 269–279.

KANG, A. R. et al. Online game bot detection based on party-play log analysis. **Elsevier**, v. 65, n. 9, p. 1984–1395, 2013.

KESUAUS. **Reddit**: Glitch/bug/hack with blade of the ruined king. [S.l.], 2014. Disponível em: https://www.reddit.com/r/leagueoflegends/comments/2mb7ut/glitchbughack_with_blade_of_the_ruined_king/. Acesso em: 7 set.2022.

KOSTER, R. **The Rules of Online World Design**. [S.l.], 2022. Disponível em: https://www.raphkoster.com/games/laws-of-online-world-design/. Acesso em: 24 out.2022.

LEHTONEN, S. Comparative study of anti-cheat methods in video games. **University of Helsinki**, 2020. Disponível em: http://urn.fi/URN:NBN:fi:hulib-202003241639. Acesso em: 9 set.2022.

LUTKEVICH, B. **obfuscation definition**. [S.l.], 2021. Disponível em: https://www.techtarget.com/searchsecurity/definition/obfuscation. Acesso em: 5 out.2022.

LUTKEVICH, B. **script kiddie definition**. [S.l.], 2021. Disponível em: https://www.techtarget.com/searchsecurity/definition/script-kiddy-or-script-kiddie. Acesso em: 6 set.2022.

MAARIO, A. et al. Redefining the risks of kernel-level anti-cheat in online gaming. In: **2021 8th International Conference on Signal Processing and Integrated Networks (SPIN)**. [S.l.: s.n.], 2021. p. 676–680.

MAIER, A. **Pattern Recognition and the Fundamental Methods of Machine Learning**: A comprehensive overview of classical ml methods. [S.l.], 2021. Disponível em: https://towardsdatascience.com/

under-the-hood-of-modern-machine-and-deep-learning-d76fc1249467. Acesso em: 27 out.2022.

MALLADI, S.; ALVES-FOSS, J.; HECKENDORN, R. B. On preventing replay attacks on security protocols. **Proc. International Conference on Security and Management**, 2002.

MARQUES, R. **De volta ao 'Counter-Strike', aspx mira carreira como streamer e crava que não usou cheat**. [S.l.], 2017. Disponível em: http://www.espn.com.br/noticia/717088_de-volta-ao-counter-strike-aspx-mira-carreira-como-streamer-e-crava-que-nao-usou-cheat. Acesso em: 20 nov.2022.

MARTIN, R. C. **Clean code: a handbook of agile software craftsmanship**. [S.l.]: Pearson Education, 2009.

McAfee. **What is a Hacker?** [S.l.], 2015. Disponível em: https://www.mcafee.com/blogs/consumer/identity-protection/what-is-a-hacker/. Acesso em: 3 set.2022.

MEGAMANX503. **dear devs who put "always online DRM"in single player games FUCK YOU. from someone who lives without internet 90% of the time**. [S.l.], 2018. Disponível em: https://www.reddit.com/r/gaming/comments/a9p5li/dear_devs_who_put_always_online_drm_in_single/. Acesso em: 1 out.2022.

MELO, G. **Vítima de aSpx na WCG 2009, rikz relembra torneio: "Não existia ideia de que o cara tava 'xitando' na LAN"**: Treinador da detona relembra caso que aconteceu na wcg 2009 e da época de ouro do cs 1.6. [S.l.], 2017. Disponível em: https://draft5.gg/noticia/vitima-de-aspx-na-wcg-2009-rikz-relembra-torneio-nao-existia-ideia-de-que-o-\cara-tava-xitando-na-lan. Acesso em: 20 nov.2022.

MELO, G. **Xeta afirma que problemas com Vanguard atrapalham crescimento de VALORANT na Coreia**: Vsegundo jogador, muitos coreanos estão interessados em jogar o fps da riot. [S.l.], 2020. Disponível em: https://valorantzone.gg/noticia/xeta-afirma-que-problemas-com-vanguard-atrapalham-crescimento-de-valorant-na-coreia/. Acesso em: 4 nov.2022.

MERCES, F. **Fundamentos da Engenharia Reversa**. [S.l.], 1981. Disponível em: https://mentebinaria.gitbook.io/engenharia-reversa/. Acesso em: 23 set.2022.

NELSON, A. **What is an aimbot in Fortnite? Why FaZe Jarvis was banned permanently for cheating**: "it didn't even cross my mind to think that i could be banned for life from fortnite from those videos". [S.l.], 2019. Disponível em: https://inews.co.uk/culture/gaming/aimbot-fortnite-what-explained-faze-jarvis-banned-life-cheating-359117. Acesso em: 29 set.2022.

Oxford Leaner's Dictionaries. **cheater definition**. [S.l.], 2021. Disponível em: https://www.oxfordlearnersdictionaries.com/definition/english/cheater. Acesso em: 14 set.2022.

PLOVER. **CWE-300: Channel Accessible by Non-Endpoint**. [S.l.], 2006. Disponível em: https://cwe.mitre.org/data/definitions/300.html. Acesso em: 27 out.2022.

POSTEL, J. **User datagram protocol**. [S.l.], 1980.

POSTEL, J. et al. **Transmission control protocol RFC 793**. [S.l.]: September, 1981.

PRESHING, J. **Hash Collision Probabilities**. [S.l.], 2011. Disponível em: https://preshing.com/20110504/hash-collision-probabilities/. Acesso em: 9 out.2022.

PUNEGOV, V. **The art of fair play: Developing the best systems to deal with players who cheat**. [S.l.], 2021. Disponível em: https://www.gamesindustry.biz/the-art-of-fair-play-developing-the-best-systems-to-deal-with-players-who-cheat. Acesso em: 29 out.2022.

PUSCH, R. **Explaining how fighting games use delay-based and rollback netcode**: How to design your game for optimal play over a network. [S.l.], 2019. Disponível em: https://arstechnica.com/gaming/2019/10/explaining-how-fighting-games-use-delay-based-and-rollback-netcode/. Acesso em: 24 out.2022.

Riot Games. **POLÍTICA E PROCEDIMENTO PARA "O TRIBUNAL"**. [S.l.], 2013. Disponível em: https://br.leagueoflegends.com/pt/legal/tribunal. Acesso em: 28 out.2022.

Riot Games. **Projeto A: Anúncio do FPS tático da Riot**. [S.l.], 2019. Disponível em: https://www.youtube.com/watch?v=4iGU6PctOBg. Acesso em: 3 nov.2022.

Riot Games. **Política de Privacidade da Riot Games**. [S.l.], 2021. Disponível em: https://www.riotgames.com/pt-br/privacy-notice-BR. Acesso em: 3 nov.2022.

RODRIGUES, M. **FACEIT implementa melhorias para aprimorar experiência em partidas ranqueadas de CS:GO na América do Sul**. [S.l.], 2022. Disponível em: https://www.gamersegames.com.br/2022/11/04/faceit-implementa-melhorias-para-aprimorar-experiencia-em-partidas-ranqueadas-de-\csgo-na-america-do-sul. Acesso em: 08 dez.2022.

RODRÍGUEZ-PÉREZ, G. et al. How bugs are born: a model to identify how bugs are introduced in software components. **Empirical Software Engineering**, Springer, v. 25, n. 2, p. 1294–1340, 2020.

ROSE, S. et al. **Zero trust architecture**. [S.l.], 2020.

ROUSSEV, V. Hashing and data fingerprinting in digital forensics. **IEEE Security & Privacy**, IEEE, v. 7, n. 2, p. 49–55, 2009.

RUIA, Y. **Valorant Anti Cheat: Cheating in Valorant at an 'all-time low' thanks to the controversial Anti-Cheat Software, Riot Vanguard.** [S.l.], 2021. Disponível em: https://thesportsrush.com/valorant-news-valorant-anti-cheat-cheating-in-valorant-at-an-all-time-low-thanks-to-the-\controversial-anti-cheat-software-riot-vanguard/. Acesso em: 20 out.2022.

SABINO, A. **COI planeja eSports nas Olimpíadas, e diretor fala em rejuvenescer os Jogos**. [S.l.], 2022. Disponível em: https://www1.folha.uol.com.br/esporte/2022/03/coi-projeta-esports-nas-olimpiadas-e-diretor-fala-em-rejuvenescer-os-jogos.shtml. Acesso em: 28 set.2022.

SENADO, A. **Punições pelo uso indevido de dados pessoais começam a valer no domingo**. [S.l.], 2021. Disponível em: https://www12.senado.leg.br/noticias/materias/2021/07/29/punicoes-pelo-uso-indevido-de-dados-pessoais-comecam-a-valer-no-domingo. Acesso em: 2 nov.2022.

SHUN-PIE. **Reddit**: Why valorants vanguard anti-cheat has to be changed asap. [S.l.], 2020. Disponível em: https://old.reddit.com/r/pcgaming/comments/g2zu1c/why_valorants_vanguard_anticheat_has_to_be/. Acesso em: 3 nov.2022.

SIKORSKI, M.; HONIG, A. **Practical malware analysis: the hands-on guide to dissecting malicious software**. [S.l.]: no starch press, 2012.

SKILLZ. **Anti-Cheating Techniques**. [S.l.], 2022. Disponível em: https://docs.skillz.com/docs/anti-cheating-techniques-overview/. Acesso em: 14 out.2022.

SMYTH, A. **What is a bottled LoL account?** [S.l.], 2022. Disponível em: https://www.unrankedsmurfs.com/blog/what-botted-lol-account. Acesso em: 17 set.2022.

SPARKLES. **I bought a HARDWARE CHEAT for CS:GO (Worlds First?!)**. [S.l.], 2020. Disponível em: https://www.youtube.com/watch?v=AIbkt6Rl8FA. Acesso em: 11 set.2022.

SPARKLES. **WALLHACK Exploit that requires no hacks... (CS:GO)**. [S.l.], 2020. Disponível em: https://www.youtube.com/watch?v=6E8yQl76QnI. Acesso em: 9 out.2022.

Steam Support. **Valve Anti-Cheat (VAC) System**. [S.l.], 2022. Disponível em: https://help.steampowered.com/en/faqs/view/571A-97DA-70E9-FF74. Acesso em: 12 out.2022.

STOLYAROV, V.; LARIN, B. **How not to use a driver to execute code with kernel privileges**. [S.l.], 2018. Disponível em: https://securelist.com/elevation-of-privileges-in-namco-driver/83707/. Acesso em: 16 dez.2022.

TE, Z. **Dust to Dust: The History of Counter-Strike**: Boom, 15 years of headshots. [S.l.], 2014. Disponível em: https://www.gamespot.com/articles/dust-to-dust-the-history-of-counter-strike/1100-6419676/. Acesso em: 13 set.2022.

The Riot Security Team. **Uma mensagem das nossas equipes de segurança e privacidade sobre o Vanguard**. [S.l.], 2020. Disponível em: https://www.riotgames.com/pt-br/notÃŋcias/uma-mensagem-das-nossas-equipes-de-seguranca-e-privacidade-sobre-o-vanguard. Acesso em: 08 dez.2022.

TIDY, J. **Police bust 'world's biggest' video-game-cheat operation**. [S.l.], 2021. Disponível em: https://www.bbc.com/news/technology-56579449. Acesso em: 3 set.2022.

TUNG, L. **These hackers used Microsoft-signed malicious drivers to further their ransomware attacks**: Attackers abused

microsoft's windows hardware developer program to get malware signed off. [S.l.], 2022. Disponível em: https://www.zdnet.com/article/these-hackers-used-microsoft-signed-malicious-drivers-to-further-their-ransomware-attacks/. Acesso em: 16 dez.2022.

VALENTE, J. **Pesquisa mostra maior preocupação das pessoas com coleta de seus dados**. [S.l.], 2019. Disponível em: https://agenciabrasil.ebc.com.br/geral/noticia/2019-11/pesquisa-mostra-maior-preocupacao-das-pessoas-com-coleta-de-seus-dados. Acesso em: 8 dez.2022.

VANKUIPERS, M. **Riot's Approach to Anti-Cheat**. [S.l.], 2018. Disponível em: https://technology.riotgames.com/news/riots-approach-anti-cheat. Acesso em: 16 out.2022.

VHPORTO. **Campeão mundial de Valorant, aspas jogou pela g3x e está banido na GC**: Relembre a trajetória do jogador, que foi acusado de cheater no cs. [S.l.], 2022. Disponível em: https://www.dust2.com.br/noticias/30973/campeao-mundial-de-valorant-aspas-jogou-pela-g3x-e-esta-banido-na-gc. Acesso em: 29 out.2022.

VIEIRA, D. **O que aconteceu com a internet discada?** [S.l.], 2022. Disponível em: https://www.tecmundo.com.br/internet/241572-aconteceu-internet-discada.htm. Acesso em: 23 set.2022.

VIVIANO, A. **User mode and kernel mode**. [S.l.], 2022. Disponível em: https://learn.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode. Acesso em: 18 out.2022.

VMCALL. **Why anti-cheat software utilize kernel drivers**. [S.l.], 2020. Disponível em: https://secret.club/2020/04/17/kernel-anticheats.html. Acesso em: 19 out.2022.

WARNER D.E. E RAITER, M. Social context in massively-multiplayer online games (mmogs): Ethical questions in shared space. **The International Review of Information Ethics**, v. 4, p. 46–52, 2005. Disponível em: https://informationethics.ca/index.php/irie/article/view/172. Acesso em: 14 set.2022.

Wiki Cheat Engine. **Cheat Engine Tutorial Guide x64**. [S.l.], 2017. Disponível em: https://wiki.cheatengine.org/index.php?title=Tutorials:Cheat_Engine_Tutorial_Guide_x64. Acesso em: 23 set.2022.

WILEY, J. J. Protection rings. In: _____. **Encyclopedia of Cryptography and Security**. Boston, MA: Springer US, 2011. p. 988–990. ISBN 978-1-4419-5906-5. Disponível em: https://doi.org/10.1007/978-1-4419-5906-5_788.

WILLINGHAM, A. **What is eSports? A look at an explosive billion-dollar industry**. [S.l.], 2018. Disponível em: https://edition.cnn.com/2018/08/27/us/esports-what-is-video-game-professional-league-madden-trnd/index.html. Acesso em: 07 dez.2022.

YEUNG, S. F. et al. Detecting cheaters for multiplayer games: theory, design and implementation. IEEE, 2006.

YOU, I.; YIM, K. Malware obfuscation techniques: A brief survey. In: IEEE. **2010 International conference on broadband, wireless computing, communication and applications**. [S.l.], 2010. p. 297–300.

ZHANG, Q. Improvement of online game anti-cheat system based on deep learning. In: IEEE. **2021 2nd International Conference on Information Science and Education (ICISE-IE)**. [S.l.], 2021. p. 652–655.

ZUCKERMAN, E. **Redes sociais criam bolhas ideológicas inacessíveis a quem pensa diferente**. [S.l.], 2017. Disponível em: https://www1.folha.uol.com.br/ilustrissima/2017/09/1920816-cada-macaco-no-seu-galho---zuckerman.shtml. Acesso em: 1 nov.2022.