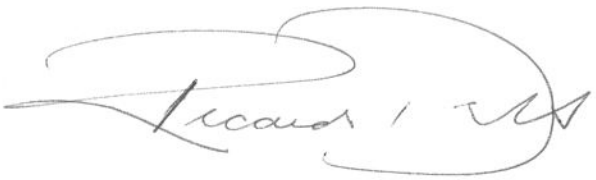
Aspectos de segurança em jogos online

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por André Gustavo Gontijo Penha e aprovada pela Banca Examinadora.

Campinas, 27 de setembro de 2007.



Prof. Dr. Ricardo Dahab (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Instituto de Computação

Universidade Estadual de Campinas

Aspectos de segurança em jogos online

André Gustavo Gontijo Penha

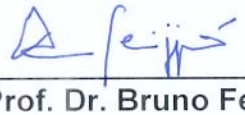
Junho de 2007

Banca Examinadora:

- Prof. Dr. Ricardo Dahab (Orientador)
  Instituto de Computação - UNICAMP
- Prof. Dr. Bruno Feijó
  Departamento de Informática, PUC-RJ
- Prof. Dr. Paulo Lício de Geus
  Instituto de Computação, UNICAMP
- Prof Dr. Ricardo de Oliveira Anido (Suplente)
  Instituto de Computação, UNICAMP

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 17 de agosto de 2007, pela Banca examinadora composta pelos Professores Doutores:



Prof. Dr. Bruno Feijó
PUC - Rio.

Prof. Dr. Paulo Lício de Geus
IC - UNICAMP.

Prof. Dr. Ricardo Dahab
IC - UNICAMP.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Júlia Milani Rodrigues CRB8a / 2116

Penha, André Gustavo Gontijo
P376a Aspectos de segurança em jogos online / André Gustavo Gontijo Penha -- Campinas, [S.P. :s.n.], 2007.

Orientador : Ricardo Dahab
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Computadores - Medidas de segurança. 2. Jogos por computador. 3. Jogos eletrônicos. I. Dahab, Ricardo. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Security issues in online games.

Palavras-chave em inglês (Keywords): 1. Computer security. 2. Computer games. 3. Electronic games.

Área de concentração: Segurança da Informação

Titulação: Mestre em Ciência da Computação

Banca examinadora:

Prof. Dr. Ricardo Dahab (IC-UNICAMP)
Prof. Dr. Bruno Feijó (Departamento de Informática-PUC-RJ)
Prof. Dr. Paulo Lício de Geus (IC-UNICAMP)
Prof. Dr. Ricardo de Oliveira Anido (IC-UNICAMP)

Data da defesa: 17/08/2007

Programa de pós-graduação: Mestrado em Ciência da Computação

Prefácio

"A arte da guerra nos ensina a confiar não na probabilidade de o inimigo não vir, mas em nossa própria prontidão para enfrentá-lo; não na eventualidade de ele não atacar, mas, antes, no fato de tornarmos nossa posição inexpugnável" (Sun Tzu).

Sob risco de incorrer em clichês, cito Sun Tzu como motivação para pensar em segurança da informação. No caso desta dissertação, preocupo-me com os problemas de segurança pertinentes ao universo dos jogos eletrônicos.

Em jogos, a motivação para ataques é ligeiramente diferente do que acontece na maioria dos sistemas distribuídos em tempo real. Um sistema cooperativo utilizado para coordenar trens de uma malha ferroviária dificilmente será atacado pelo próprio usuário ou operador de um dos *peers*. Em jogos distribuídos, há o fator competição: é, na prática, um sistema cooperativo onde os *peers* normalmente competem entre si. Cooperativo?

Esta dissertação expõe que mesmo quando jogos online não são totalmente distribuídos, i.e. em sistemas onde a maioria das decisões se concentra no servidor e os clientes são próximos de terminais de entrada e saída, há de se pensar em segurança. E assim como em várias outras situações na vida, o risco aumenta à medida que dependemos mais da honestidade de quem está distante.

Para expor estas questões, este texto classifica ataques a jogos, cita soluções para alguns destes ataques, menciona plataformas de segurança que podem ser aplicadas a jogos e, por fim, associa um protocolo de sincronização segura a arquiteturas híbridas (distribuição em peers com existência de servidores).

Agradecimentos

Eu gostaria de agradecer a meus pais, Elson e Veralice, e à Michèle, pelo estímulo necessário para estudar muito, mesmo durante o árduo trabalho de se estabelecer uma empresa e fazê-la crescer.

Agradeço também, e especialmente, a meu orientador e amigo Ricardo Dahab, que com sua incrível paciência suportou minha demora em entregar esta dissertação e incentivou a mudança do tema para uma área onde me sinto mais confortável - relacionada a jogos eletrônicos. Sem dúvida ensinou-me, além de boas noções de criptografia e segurança (desde a graduação), a me esforçar para ser um líder paciente. Gostaria de deixar explícito que sua filosofia me ajudou em minha formação como empresário e entusiasta pelo desenvolvimento da indústria brasileira de jogos eletrônicos.

Destaco ainda meu agradecimento ao CNPq, pelos seis meses de bolsa que recebi no início dos trabalhos, em 2003, e a Américo Tomé, da Intel Semicondutores do Brasil, por apoio a projetos dos quais participei durante o mestrado.

Obrigado também aos professores Ricardo Anido, do Instituto de Computação da Unicamp, e Bruno Feijó, da Pontifícia Universidade Católica do Rio de Janeiro, cujos trabalhos me serviram como semente para início das pesquisas e cujas ótimas sugestões consegui adotar apenas parcialmente. Espero um dia poder seguir as recomendações destes dois profissionais em trabalhos futuros.

Cito como importante a compreensão de meus sócios na Tectoy Digital, Rafael Nanya, braço direito que ajudou a tocar os negócios e entendeu minhas ausências durante a redação deste trabalho, e os diretores da empresa investidora Tectoy S.A., que além de

entender minhas ausências para escrever este documento, me deram condição para crescer na indústria de jogos e indiretamente contribuíram na motivação para prosseguir neste tema.

Agradeço por fim aos demais amigos que participaram indiretamente na construção de minha dissertação, em especial ao amigo e ex-sócio Sérgio Jábali, que em seus estudos de sistemas distribuídos muitas vezes parou para discutir comigo o sistema híbrido que cito algumas vezes na dissertação.

Resumo

Jogos online e jogos móveis são os nichos que mais crescem na bilionária indústria de entretenimento eletrônico. Em países fortemente afetados por pirataria, como Brasil, Rússia e China, modelos de jogos em rede são vistos por publicadores como alternativas para contornar o problema. Essa expansão online traz consigo, no entanto, as preocupações usuais de sistemas em rede: disponibilidade, escalabilidade, segurança.

Jogos, em particular, são normalmente ambientes de disputa. Joga-se em rede contra alguém, seja por batalhas sangrentas, pela construção de impérios que se enfrentam ou na administração de ferrovias concorrentes. É justamente esta concorrência que motiva grande parte dos ataques a jogos.

Para combater os ataques, um primeiro passo é entender as vulnerabilidades, que podem estar na concepção de um jogo, no software criado para implementá-lo, nos drivers (de vídeo, por exemplo), nos protocolos utilizados.

Nesta dissertação procuro identificar pontos de vulnerabilidade, apresentando uma classificação taxonômica de ataques e estudando soluções já conhecidas para alguns casos. Além disso, proponho a junção de uma solução de segurança a um modelo híbrido de distribuição de jogos (que utiliza conceitos de distribuição em peers e de arquitetura cliente-servidor).

Abstract

Online and mobile games are two of the fastest growing niches inside the billionnaire game industry. Countries like Brazil, Russia and China are strongly affected by piracy and publishers see in networks a tool to reduce the impact of such issue. This online expansion, however, brings to gaming the usual attention points on network computing: availability, scalability and security.

Games are particularly motivated by battles. People play against each other: they become soldiers in bloody wars, build and conquer new empires and administrate railroads. It is such competition that motivates most of the attacks to online games.

To avoid the attacks, a first step is to understand vulnerabilities - they may lie in different levels within a game product: the game conception itself, the software created, the drivers (video, for example), the networking protocols.

This dissertation aims to understand vulnerability points, presenting a taxonomic classification of attacks and studying known solutions. Furthermore, it proposes using a security protocol to a hybrid distributed computing model (that combines peer-to-peer and client-server architectures).

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação

1.1.1 O mercado de jogos em rede

O mercado de jogos eletrônicos movimentou em 2005, em todo o mundo, cerca de 27 bilhões de dólares; estima-se crescimento de 12% ao ano neste setor até 2010 [8]. Em tempos de mobilidade e Internet rápida nas casas, muda-se o paradigma dos jogos eletrônicos. Antes apenas caixas com CDs ou disquetes vendidas em lojas, diversão para se jogar sozinho em casa, os jogos digitais agora passam a incorporar os conceitos de comunidade virtual e universo persistente. Este conceito fica ainda mais forte com a nova geração de consoles domésticos que surgiu em 2006, cujos jogos prometem usar ainda mais recursos de rede. A Microsoft já disponibiliza a rede de serviços X-Box Live[35] há algum tempo e promete investir ainda mais em comunidades. A Sony e a Nintendo já apostam em redes Wi-Fi, disponibilizando o recurso em seus últimos portáteis (PSP e NintendoDS, respectivamente) e consoles domésticos (PlayStation-3[47] e Wii[39]). Para motivar ainda mais a mudança de paradigma, como se não bastassem a nova diversão trazida por confronto e cooperação também com inteligências não-artificiais e a maior interação social promovida pelo novo esquema, o mercado de países emergentes - como o Brasil - apresenta

características que fazem necessária a adoção de modelos de negócio menos vulneráveis à distribuição de cópias clandestinas. A pirataria, aliás, era até pouco tempo atrás a principal preocupação dos fabricantes e publicadores de jogos eletrônicos em se tratando de segurança. É fácil perceber o insucesso dos desenvolvedores nesta batalha ao notar o comércio clandestino nas grandes cidades do País ou, mais formalmente, avaliando estudos como os do IDG Consulting, que apontam que 94% dos jogos vendidos no Brasil são piratas[15]. Os jogos online não resolvem a questão da venda de software ilegal, mas contornam o problema. Investir na produção e distribuição de software para o grande público volta a ser interessante no País com o novo modelo de negócio, que pode permitir que as cópias de software sejam distribuídas livremente, mas cobra pela utilização dos servidores no jogo, seja diretamente pela aplicação de pagamentos periódicos (como no Brasil faz o Ragnarok, da Level-Up Interactive[25]), seja pela cobrança por itens do jogo[27] (como fazem no Brasil os jogos Priston Tale e Second Life, publicados no país pela Kaizen Games). A empresa DFC intelligence, especializada em análise de mercado, prevê que o mercado de jogos exclusivamente online cresça de 3,4 bilhões (2005) para 13 bilhões de dólares em 2011[24]. O significativo mercado de jogos que têm funcionalidades online, como alguns jogos em videogame (e.g. Gran Turismo 4, no Sony Playstation e Mario Kart, no Nintendo DS), e praticamente todos os grandes jogos de PC, não está incluído nesta previsão, mas também é foco dos estudos desta dissertação.

1.1.2 Segurança em jogos online

Jogos em rede, como qualquer sistema complexo online, têm diversos pontos passíveis de ataque. A exploração de pontos fracos em jogos digitais pode trazer prejuízos para os jogadores, desenvolvedores e publicadores. Algumas trapaças, feitas no intuito de beneficiar um jogador perante outros, podem deixar o jogo menos interessante para os consumidores, o que indiretamente também afeta publicadores e desenvolvedores. Outros tipos de trapaça podem comprometer o sistema de bilhetagem, permitindo que os atacantes joguem de graça ou ainda que se beneficiem na aquisição de produtos virtuais sem pagar por eles.

Este tipo de ação, de uma maneira geral, prejudica toda a indústria de jogos ao tornar os negócios mais arriscados e dificultar investimentos no setor. Certamente são inúmeras as potenciais vulnerabilidades em jogos online, mas é possível descrever uma taxonomia de ataques para facilitar o estudo de como tornar mais seguros os jogos multiusuários.

1.2 Organização

Esta dissertação se inicia mencionando alguns conceitos básicos de criptografia, no Capítulo 2. Menciona objetivos da segurança da informação, lista primitivas criptográficas importantes e descreve brevemente algumas derivações (das primitivas) citadas ao longo do documento.

No Capítulo 3, apresentam-se conceitos específicos do mundo dos jogos. Diferenciam-se jogos por turno de jogos em tempo real, citam-se exemplos de arquiteturas típicas de sistemas de jogos, fala-se sobre arquiteturas de jogos multiusuários e introduzem-se conceitos como *jitter*, *bucket synchronization* e *dead reckoning*, todos associados a jogos em rede.

Em seguida, o Capítulo 4 faz uma classificação taxonômica de ataques a jogos online e, depois de citar cada classe, resume o conteúdo em uma tabela que associa os ataques a exemplos de ocorrências e potenciais soluções.

Exemplos de ataques são o assunto do Capítulo 5, que cita casos de vulnerabilidade em jogos de tiro (*first person shooters*), jogos estratégia em tempo real (RTS) e um ambiente de mundo persistente.

Os Capítulos 6 e 7 falam, respectivamente, de soluções de segurança em nível de protocolo e de plataformas para segurança de jogos online. Dentre as plataformas aplicáveis de segurança a jogos online, cita-se Terra, uma máquina virtual que implementa uma plataforma segura para execução de aplicações, cujos autores utilizaram para construir uma versão segura do jogo Quake. Fala-se também de *Trusted Platform Module*, uma especificação de hardware para implementação de plataformas seguras.

O Capítulo final fala de soluções de segurança em arquiteturas específicas, aprofundando em segurança de jogos distribuídos; para isso, usa como exemplo uma proposta de mecanismo de distribuição de processamento em *peers* e mostra que é possível associar a este mecanismo um protocolo útil no combate a determinados ataques. Como resultado, propõe-se uma abordagem que ganha segurança em alguns aspectos, mesmo com a descentralização do processamento, recurso que normalmente torna sistemas mais vulneráveis.

1.3 Objetivos e contribuições deste trabalho

Uma visão geral sobre o assunto foi apresentada por Börje Karlsson e Bruno Feijó, da PUC-Rio, em 2004 [28]. O trabalho de Karlsson e Feijó contribuiu para o início desta dissertação ao resumir análises de trabalhos acadêmicos na área, os quais estão entre os pesquisados durante o desenvolvimento do presente trabalho.

Da mesma forma, este texto pretende motivar mais estudos na área de segurança em jogos, sobretudo análises e implementações de soluções propostas, estendendo a busca de um ponto ótimo na curva *segurança vs. eficiência* de um jogo online, em que ambos os aspectos são importantes para o fator diversão associado ao jogo.

Além de motivar estudos acadêmicos, o levantamento de casos, a associação de falhas a potenciais soluções e as discussões sobre a pertinência de problemas de segurança ao projeto de um jogo podem ser utilizados por *game designers* e engenheiros da nascente indústria de jogos eletrônicos nacional, cujas barreiras incluem também o acesso limitado a informações técnicas e de mercado.

Capítulo 2

Conceitos básicos de criptografia e segurança

Este capítulo dá uma visão macroscópica de primitivas criptográficas citadas ao longo da dissertação. Antes, no entanto, disserta-se um pouco sobre a segurança da informação em si.

2.1 Segurança vs. Robustez

Em inglês há duas palavras, *safety* e *security* que, ao serem traduzidas para o português, comumente se transformam ambas no termo "segurança". Entretanto, são dois conceitos diferentes, embora próximos. Para evitar ambigüidades, prefere-se neste trabalho - principalmente agora ao explicar as diferenças - traduzir security como "segurança" e safety como "robustez". Quando se diz segurança, principal dos temas sobre os quais disserto neste trabalho, refiro-me à resistência a ataques (normalmente voluntários) que comprometam principalmente a confidencialidade, a disponibilidade dos dados, a autenticação de entidades emissoras e receptoras de dados, a integridade dos dados e o não-repúdio de ações. Robustez refire-se à correta operação dos sistemas, mesmo que ocorra alguma falha em algum algoritmo ou hardware, ou seja, a resistência a falhas que normalmente

são involuntárias. Esta dissertação não trata diretamente de robustez.

2.2 Objetivos em segurança da informação

Embora o termo "segurança da informação" já diga muito, é interessante descrever um pouco mais sobre objetivos da segurança de informação. Em [34], listam-se como objetivos de segurança da informação:

- **confidencialidade:** a informação somente é acessível a entidades autorizadas;
- **integridade:** garantia de que a informação não tenha sido alterada, de forma imperceptível, por pessoas ou meios desautorizados ou desconhecidos;
- **autenticação de entidade:** reconhecimento de uma identidade (uma pessoa, um cartão de crédito, um computador, um processo em execução);
- **autenticação de mensagem:** identificação da origem de uma mensagem (data origin authentication);
- **assinatura:** associação de uma informação a uma entidade;
- **autorização:** concessão, a outra entidade, do direito de agir de determinada maneira ou ter acesso a local restrito;
- **controle de acesso:** restrição de acesso a um determinado conjunto de informações;
- **certificação:** endosso de uma informação por uma entidade confiável;
- **aposição de carimbo de tempo:** identificação da hora de criação, existência ou validade de uma informação;
- **testemunho:** verificação da existência ou criação de uma informação por alguém que não seja o próprio criador;

- **recibo:** confirmação de que uma informação foi recebida;
- **anonimato:** omissão da identidade de um indivíduo envolvido em um processo;
- **não-repúdio:** prevenção da negação de uma ação por parte de seu executor;
- **revogação:** cancelamento de certificado ou autorização.

Alguns destes objetivos são normalmente alcançados em conjunto, ou seja, ao se atingir um objetivo, automaticamente cumprem-se outros.

2.3 Questões de segurança em comércio eletrônico

As necessidades de segurança listadas nesta seção não são apenas pertinentes ao comércio eletrônico. No entanto, por ser bastante difundido, o *e-commerce* se torna um exemplo didático das necessidades de segurança da informação.

Além de existir uma intersecção considerável entre as necessidades de segurança em *e-commerce* e em jogos online, é cada vez mais comum haver comércio eletrônico (de itens virtuais) dentro de um jogo.

Em um site de compras, tem-se normalmente os seguintes requisitos:

1. **Disponibilidade**: embora seja uma questão mais ligada diretamente à robustez que à segurança, disponibilidade é o principal requisito para um site de compras. Simples: uma loja fechada não vende.

2. **Confidencialidade**: os dados de meu cartão de crédito, meus dados pessoais e até mesmo os itens que estou comprando são informações que não posso disponibilizar a potenciais criminosos. O que espero é que primeiro o sistema proteja estas informações enquanto trafegam de meu computador até o servidor da "loja" e depois a guarde bem (ou jogue-a fora), de maneira a tornar impossível que alguém invada a loja e roube minhas informações.

3. **Autenticação**: É necessária nos dois sentidos - primeiro preciso saber se a loja onde compro é realmente a entidade que afirma ser antes de enviar as informações de pagamento. Caso contrário, posso entregar informações confidenciais a uma entidade não-autorizada ou efetuar um pagamento a alguém que não me entregará a mercadoria. No outro sentido, quero que a loja me identifique - e se alguém mais tentar comprar, dizendo ser eu?

4. **Integridade**: Se eu fiz determinado pedido, não quero que ele seja alterado, seja por descuido ou de forma proposital. Quero receber exatamente o que comprei e pagar o preço acordado.

5. **Consistência**: O estado que eu enxergo da minha compra deve ser o mesmo visto pela loja. Ou seja, as transações devem ser ACID:

   - Atômicas: ou uma ação aconteceu 100%, ou 0%. Não quero, por exemplo, efetuar um pagamento e permitir que um ataque ou falha interrompa a transação sem que minha compra seja concluída. Se eu paguei pelo produto, quero recebê-lo. A loja tampouco quer me permitir concluir a compra e, durante o pagamento, desligar o computador da rede.
   - Consistentes: eu e a loja devemos ambos estar de acordo com a transação.
   - Independentes (ou isoladas): uma transação não interfere em outra.
   - Duráveis: se houve algum problema durante a transação, o sistema deve ser capaz de voltar ao último estado consistente. Ou seja, se eu puxar a tomada do computador na hora do pagamento, a compra não ocorre. Outro exemplo, se comprei um software cujo *download* falhou, o sistema deve voltar ao estado "software ainda não baixado" para que eu possa tentar novamente.

6. **Não repúdio**: Se Alice compra com seu cartão uma mercadoria que deve chegar para o endereço de Bob, é necessário que o sistema possa provar que foi Alice quem comprou. Afinal, vão cobrá-la por isso.

Como o comércio eletrônico é muitíssimo comum, já existe uma infra-estrutura padronizada que garante os requisitos básicos listados acima, usada em sites web que necessitam segurança.

Uma assinatura digital provê Autenticação, Integridade e Não-repúdio. A infra-estrutura de chaves públicas (ICP)[14], que no Brasil foi instituída em 2001 e é gerida pelo Instituto Nacional de Tecnologia de Informação (ITI), provê um mecanismo de certificados que, de maneira análoga ao reconhecimento de firma em cartório, permite verificar a autenticidade de uma assinatura digital.

2.4 Primitivas criptográficas

As ferramentas (providas por criptografia) para alcançar os objetivos em segurança da informação são chamadas de **primitivas criptográficas**. Em [34], Menezes, Oorschot e Vanstone as dividem em três grupos, de acordo com a necessidade de chaves. Dentro de cada grupo, destacam-se as primitivas:

2.4.1 Primitivas sem chave

- **funções de resumo:** Uma função de resumo (*hash function*) é um mapeamento (computacionalmente eficiente) de cadeias binárias de tamanho arbitrário em cadeias binárias de tamanho fixo (*message digest*). Para funções de resumo ideais cujas saídas têm n bits de tamanho, a probabilidade de uma cadeia aleatoriamente definida ser mapeada em um determinado resumo é 2^{-n}. Ajustando-se n à aplicação desejada, fica computacionalmente inviável descobrir duas cadeias que tenham o mesmo resumo (detectar colisão de resumo) ou, dado um resumo y_1 e uma função $h(x) = y$, encontrar valor(es) de x tal que $h(x_1) = y_1$.

- **seqüências aleatórias:** são usadas em criptografia sobretudo na geração de chaves.

2.4.2 Primitivas de chave simétrica

- **ciframento com chave simétrica:** Ao se cifrar uma informação, diz-se que o ciframento é de chave simétrica se, para um par de chaves de ciframento e deciframento (K, K'), é computacionalmente fácil encontrar K sabendo-se K' e encontrar K' sabendo-se K.

- **código de autenticação de mensagem (MAC):** Um mecanismo de autenticação que usa chave simétrica (i.e. se as chaves (K_g, K_v) de geração e verificação podem ser obtidas uma a partir da outra) é chamado de MAC (acrônimo para Message Authentication Code). Um MAC é utilizável na verificação da autenticidade de uma mensagem (somente alguém que possuía a chave a pode ter gerado a mensagem), bem como a sua integridade (a mensagem não sofreu alteração depois da geração do MAC). Uma derivação de MAC é o *hash message authentication code*, cuja implementação envolve a utilização de função de resumo (*hash*) com chave. O HMAC está descrito na Seção 2.5.1.

2.4.3 Primitivas de chave pública

- **ciframento com chave pública:** ao contrário do ciframento com chave simétrica, o ciframento com chave pública usa um par (K_u, K_p) de chaves **distintas**, mas relacionadas, para ciframento e deciframento. Neste caso, K_u é conhecida publicamente e K_p é de conhecimento exclusivo do destinatário de uma mensagem. A obtenção de K_p a partir de K_u é inviável computacionalmente. A mensagem cifrada com K_u somente pode ser decifrada com K_p.

- **assinatura com chave assimétrica:** um esquema de assinatura digital é análogo a uma assinatura normal, ou seja, permite a uma entidade mostrar sua identidade, autenticar um documento ou dar sua palavra a respeito de alguma ação, como por exemplo um cheque, que na prática "promete o pagamento" ao destinatário. Uma assinatura normalmente depende, para ser gerada, de uma chave privada K_p e

somente pode ser verificada pelo emprego de uma chave pública K_u. Desta forma, sabe-se que a assinatura reconhecida com K_u só pode ter sido gerada por quem possuía a chave K_p.

2.5 Derivações das primitivas

Das primitivas mencionadas obtêm-se mecanismos criptográficos e algoritmos citados ao longo dessa dissertação. Dentre eles, destacam-se:

2.5.1 HMAC - Hash Message Authentication Code

Um HMAC é um tipo de código de autenticação de mensagem (MAC) cujo cálculo utiliza a combinação de uma função de resumo (*hash*) e uma chave secreta. Assim como o MAC, o HMAC tem como objetivo garantir a autenticidade e a integridade de uma mensagem.

Em um algoritmo de MAC tradicional, a entrada é uma mensagem de tamanho definido e uma chave secreta, enquanto a saída é o código de verificação (MAC). Com o uso de uma função de resumo, é possível que a entrada seja uma mensagem de qualquer tamanho, com saída de tamanho fixo. A RFC 2104 [30] especifica a forma de combinar mensagem e chave.

2.5.2 Atestação remota

Atestação remota permite que entidades pré-autorizadas detectem certas modificações no computador do usuário. A atestação funciona com a geração de um certificado, por parte do hardware, confirmando o software que roda em um dado momento. Isso pode evitar a modificação não-autorizada de software, como citado na Seção 4.9.1 deste documento. Uma atestação, no entanto, não garante que a aplicação não tenha sido modificada após emissão do certificado.

Atestação remota é normalmente combinada com ciframento assimétrico, para evitar que entidades não-autorizadas (como o usuário do hardware, por exemplo) tenham acesso

à informação.

Embora não seja o foco deste trabalho, vale citar que atestação remota também tem aplicações no gerenciamento de direitos digitais (DRM).

Capítulo 3

Conceitos básicos e terminologia dos jogos online

Este capítulo tem o objetivo de introduzir conceitos específicos do mundo dos jogos e conceitos de computação aplicados a jogos, tratando sobretudo de termos utilizados nesta dissertação.

Alguns termos comuns em textos sobre jogos aparecem ao longo deste documento, como a palavra "jogabilidade", tradução do termo inglês *gameplay*. Jogabilidade se refere à qualidade do jogo ser divertido e atrativo, ou ainda simplesmente "bem jogável".

3.1 Jogos por turno e jogos em tempo real

Esta dissertação se refere muitas vezes a jogos por turno e a jogos em tempo real. Jogos por turno são aqueles que esperam que todos terminem sua jogada até começar um novo turno; jogos em tempo real são aqueles em que ações dos jogadores estão contínua e paralelamente ocorrendo, não importando a velocidade de outro jogador ao responder, os comandos de jogo devem ser processados imediatamente. Xadrez, por exemplo, é um jogo de turnos: eu me movimento apenas depois de entender o movimento do adversário. Também é assim um jogo de pôquer. Futebol, vôlei e corrida (como a Fórmula 1) são

esportes que acontecem em tempo real, ou seja, eu não espero que outros participantes joguem para tomar minha próxima decisão. Se eu consigo agir mais rapidamente, melhor.

3.2 Arquiteturas típicas de sistemas de jogos

A Figura 3.1 é um exemplo de arquitetura de sistema de jogos comum atualmente. Com o crescimento dos jogos 3D, na década de 90, surgiram comercialmente os motores de jogos (*game engines*), conjuntos de componentes de software que implementam recursos utilizados por vários jogos. O principal componente de um motor de jogos é o motor gráfico (*rendering engine*), responsável por transformar os vértices com coordenadas 3D e suas texturas nas imagens exibidas pelo jogo, utilizando para isso outros componentes de software e hardware, como resumem a Figura 3.1. *Game engines* normalmente ainda têm um motor de física e colisão e componentes para auxílio na implementação de inteligência artificial, bem como tratamento de som e rede. O tratamento de rede por motores de jogos pode ser desde o acesso a sockets até complexos mecanismos de auxílio a gerenciamento de peers, como o implementado no Instituto de Computação entre 2005 e 2006 [2].

Abaixo do *game engine*, mecanismos que auxiliam em acesso ao hardware, gerenciamento de memória e escalonamento de processos são concentrados no sistema operacional, que em PCs não é específico para jogos, mas pode incluir recursos interessantes, como o componente Direct3D (parte do Microsoft Windows).

O hardware de um sistema de jogos atual tem não apenas múltiplas unidades de processamento com bastante memória, mas também (e principalmente) uma GPU (Graphics Processing Unit) com alto poder de processamento e memória dedicada. A GPU é um processador otimizado para o tratamento de gráficos, como rasterização, aplicação de texturas e sombreamento (*shading*). Os jogos 3D atuais fomentam o consumo de hardwares modernos de processamento de vídeo, que contam hoje com múltiplas GPUs. A Figura 3.2 ilustra um pipeline clássico de renderização.

Em telefones celulares, jogos 3D ainda são uma novidade. Com o aumento do poder

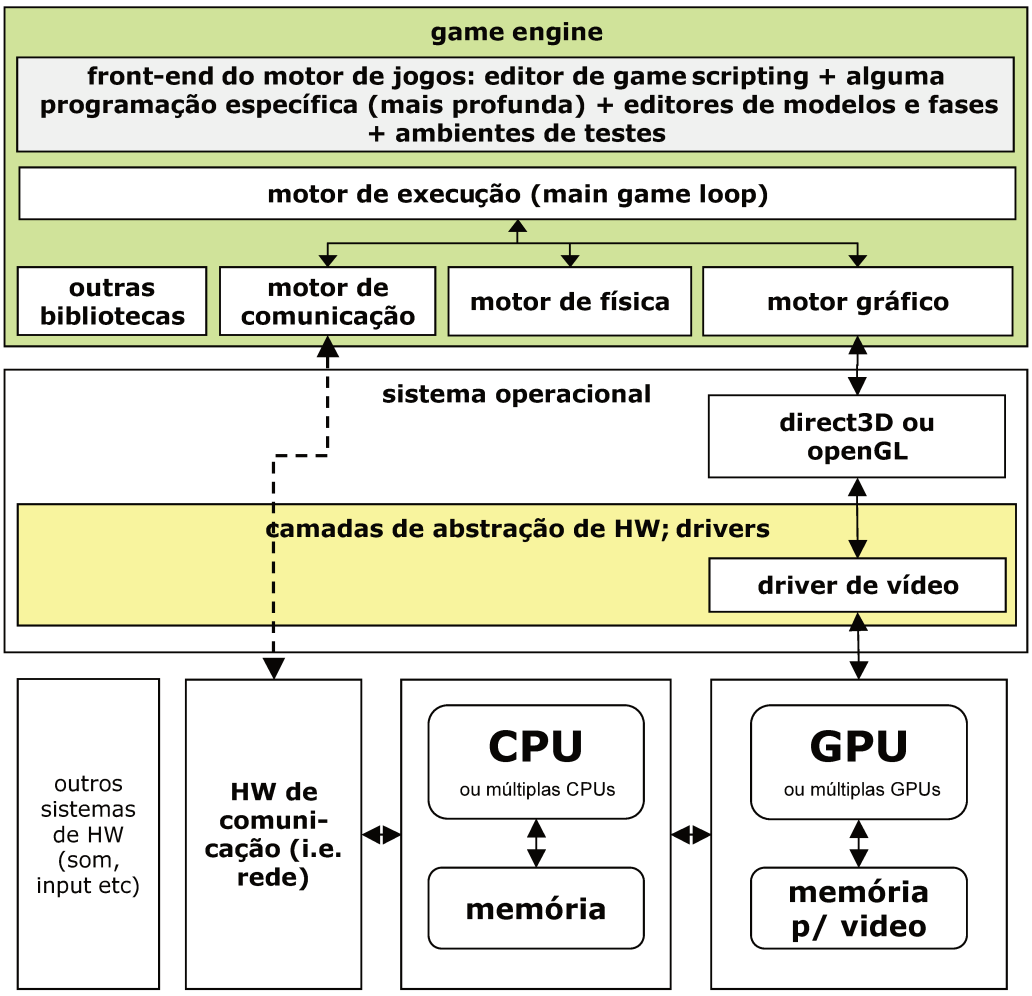


Figura 3.1: exemplo de arquitetura de um sistema de jogos.

de processamento dos telefones, é fácil prever que jogos que demandam mais recursos serão cada vez mais comuns. Também é previsível que, em breve, jogos em rede para celulares estarão mais comuns que hoje, e que os problemas para segurança em jogos online logo afetarão estes dispositivos. A diversidade de hardware em celulares é maior que em PCs e, portanto, camadas de abstração de hardware são ainda mais importantes. Duas linhas dominam as plataformas de celulares hoje em dia: Java Micro-edition [36], da Sun, e Brew [23], da Qualcomm. Ambas têm a padronização dos recursos (abstração

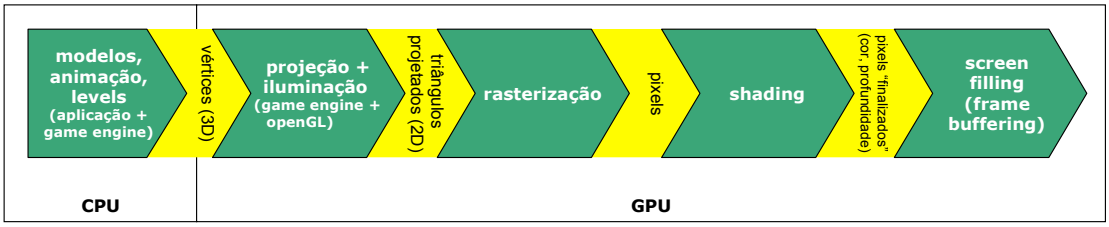


Figura 3.2: visão superficial de um *pipeline* de renderização de gráficos 3D, desde o modelo até a coloração de pixels.

do hardware) como um dos principais objetivos. A Figura 3.3 mostra macroscopicamente a arquitetura de um sistema de jogos em celulares Brew equipados com hardware para aceleração 3D.

Vale também lembrar que uma parte significativa dos celulares utiliza o sistema operacional Symbian[33]. A empresa Symbian Ltd. é um consórcio[32] formado por Nokia, SonyEricsson, Ericsson, Siemens, Panasonic e Samsung. Os principais telefones com sistema Symbian presentes no mercado brasileiro são das marcas Nokia e SonyEricsson. Em sua especificação, Symbian traz Java ME como uma camada para desenvolvimento de aplicações, apesar de a plataforma para o desenvolvimento diretamente sobre o sistema operacional ser também acessível para qualquer desenvolvedor.

Em consoles domésticos (*game consoles*), a arquitetura é comparável à dos PCs. Em gerações anteriores argumentava-se que, como consoles domésticos eram plataformas fechadas (apenas reveladas a desenvolvedores) e praticamente não tinham acesso a rede, eram provavelmente menos suscetíveis aos ataques mencionados no capítulo 4. No entanto, o desenvolvimento (de pequenos jogos e aplicações) para consoles tem se pulverizado bastante, seja oficialmente, como por exemplo através da platafoma Microsoft XNA[9], ou por iniciativas independentes, como a do grupo “PS2dev.org” [45].

Voltando à questão de segurança, quatro pontos são observados nesta dissertação: game design, mencionado em alguns aspectos de segurança mas não a questão principal do estudo; protocolos de comunicação (transmissão da informação); armazenamento de

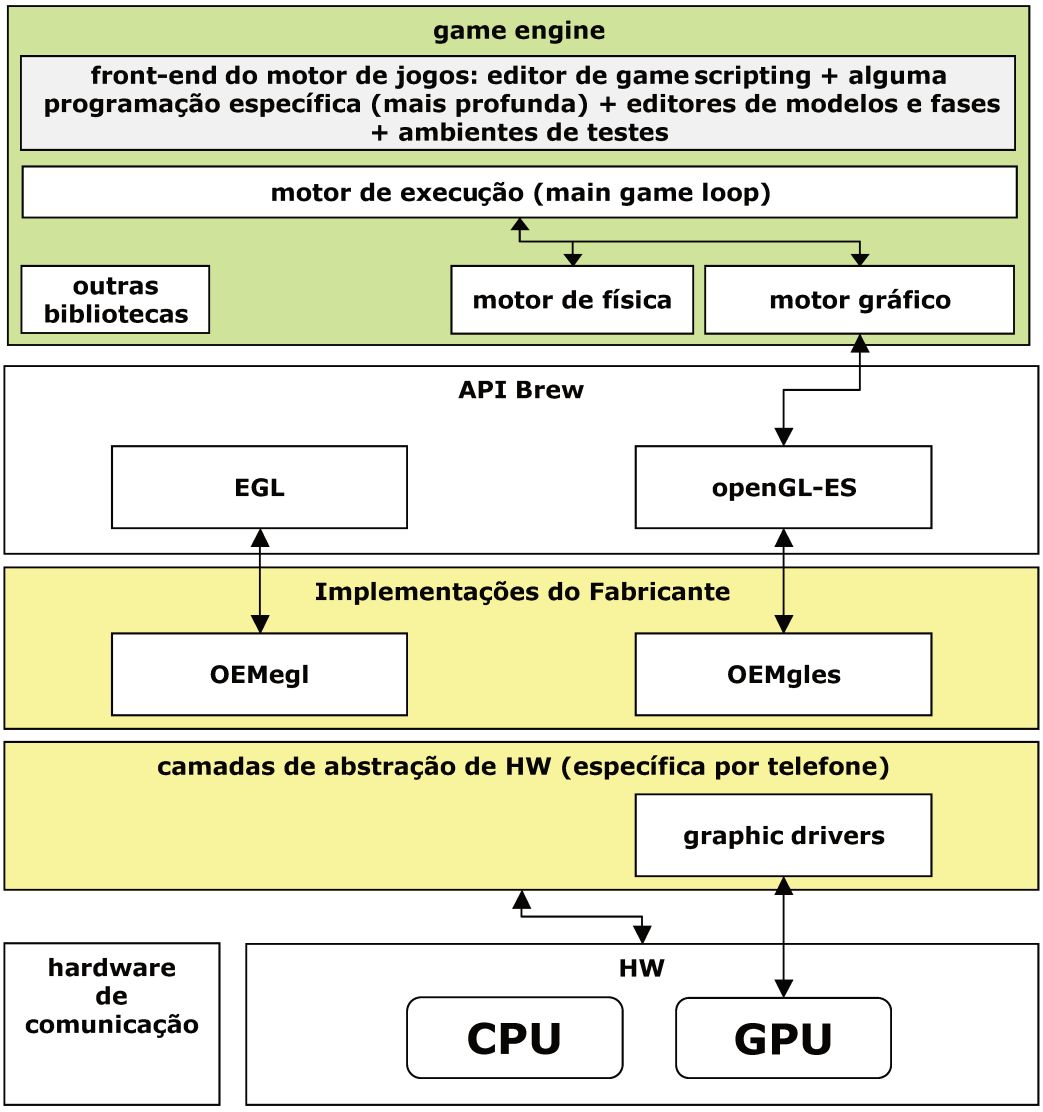


Figura 3.3: exemplo de arquitetura de um sistema de jogos em um celular Brew equipado com GPU.

informação em memória (e disco); transformação da informação ao mostrá-la em vídeo. A Figura 3.4 associa estes pontos às subdivisões definidas anteriormente na Figura 3.1.

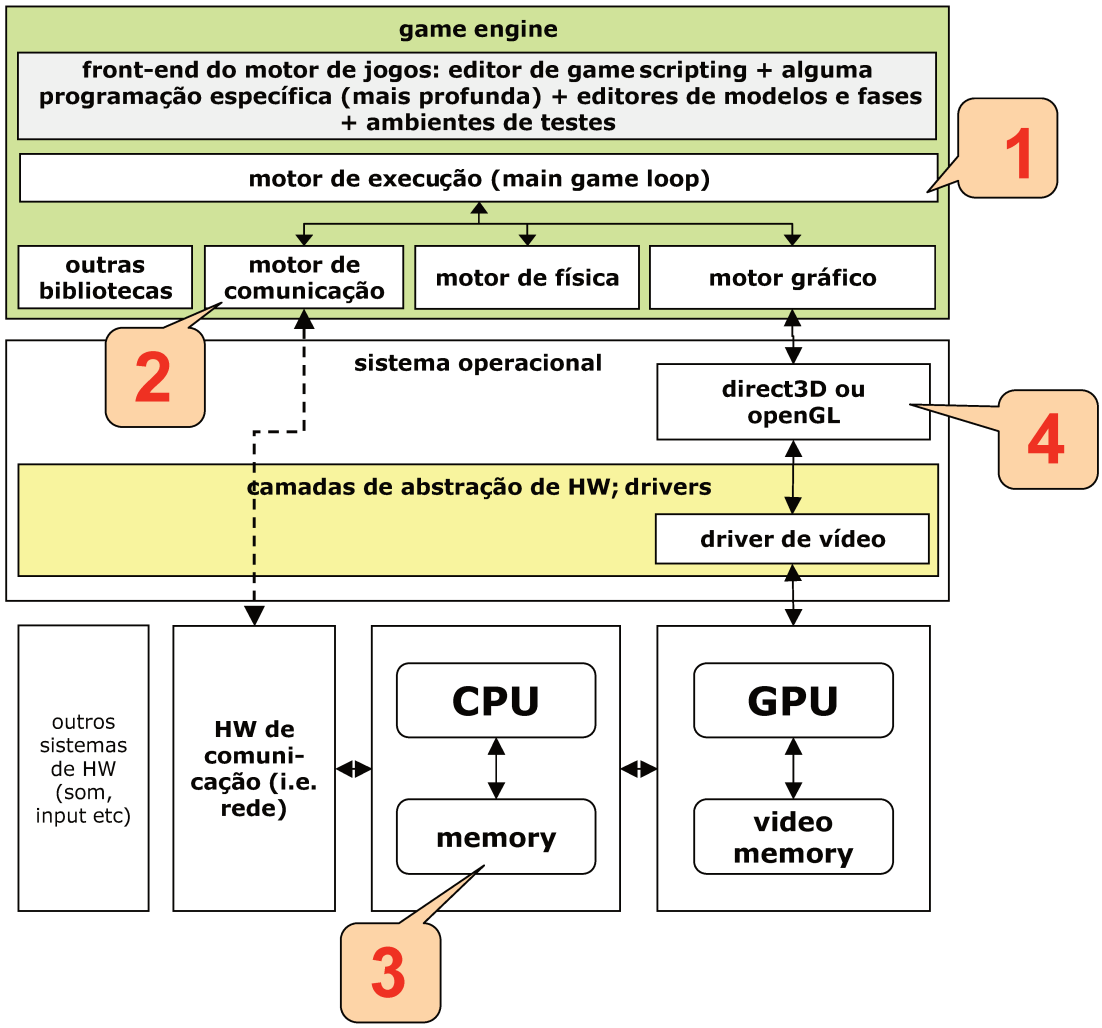


Figura 3.4: os ataques mostrados no capítulo 4 ocorrem no nível de: 1) *game design*; 2) protocolos de comunicação; 3) armazenamento em memória e disco; 4) exibição das informações em vídeo.

3.3 Arquiteturas de jogos online

Até o início da década de 90, quase todos os jogos eram jogados em apenas um computador. O desafio era um quebra-cabeças, vencer um agente com inteligência artificial ou disputar contra alguém (próximo), também controlando o mesmo PC (compartilhando o

teclado ou com dois joysticks). A proliferação dos jogos multiusuário veio com a popularização da Internet. Desde então, para permitir que pessoas interagissem à distância através de um jogo, algumas arquiteturas surgiram, associadas a modelos diferentes de jogos. Entre apenas dois modems nos primeiros jogos, pouca diferença fazia classificar a arquitetura como Peer-to-peer ou cliente-servidor. A medida que cresce o número de máquinas participantes, no entanto, cada arquitetura passa a ter vantagens e custos.

3.3.1 Modelo cliente-servidor

Nesta arquitetura, a mais comum em jogos multiusuário, o servidor é onisciente. Tudo o que acontece no jogo, todas as decisões, a inteligência artificial de cada agente não controlado por humanos, são tarefas do servidor. Ele recebe pela rede os comandos dos jogadores, processa e devolve a informação de forma que o cliente possa mostrá-la com imagens, sons e outros métodos de feedback (como vibração do joystick, por exemplo). Quando poucos players participam de uma partida, é possível que um deles tenha seu computador atuando tanto como cliente quanto como servidor, o que é comum em jogos como Battlefield [3], por exemplo. Já quando o volume de informações que o servidor controla é muito grande, é interessante ter um servidor dedicado, ou seja, uma máquina que apenas gerencia os estados do jogo e troca informações pela rede, não utilizando poder de processamento para operações de renderização de gráficos ou emissão de sons. Exemplos extremos desta situação são os *Massively Multiplaying Online Games* (MMOGs), jogos disputados por centenas, milhares ou até centenas de milhares de jogadores simultaneamente. O jogo Ragnarok[25], por exemplo, tem mais de quatro milhões de usuários em todo o planeta. Só no Brasil, o número de jogadores simultâneos neste jogo chega a 20 mil[1]. É fácil perceber que o poder de processamento e a largura de banda necessários no servidor são desafios para a escalabilidade deste modelo. A vantagem clara do modelo com servidores centrais é a maior simplicidade para implementação. Como todo o processamento de estados e todas as decisões ocorrem no servidor, a exposição a problemas de segurança é bem menor. E como os canais de comunicação são sempre entre clientes e

servidores, é mais complicado para um atacante fazer com que algum dos clientes receba informação errada.

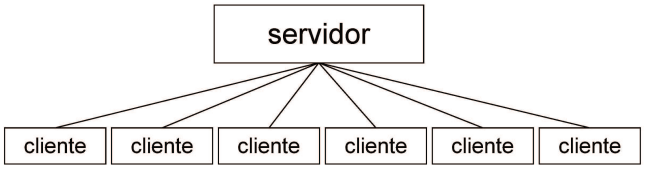


Figura 3.5: em um modelo cliente servidor, todas as comunicações acontecem entre um dos clientes e o servidor e as decisões são tomadas pelo servidor - é mais simples garantir segurança.

3.3.2 Modelo peer to peer (P2P)

Oposta ao modelo cliente-servidor, a arquitetura P2P gera tráfego de informações entre cada par de peers presente na partida. Neste caso, o número de canais de comunicação estabelecidos deixa de ser n (sendo n o número de computadores na partida) e passa a ser $\binom{n}{2}$. Em [12], argumenta-se que o volume total de dados que trafegam entre os peers é igual ao volume total de dados no modelo cliente-servidor. Neste caso, no entanto, a necessidade de banda é distribuída entre os peers, assim como acontece com a carga de processamento. O custo de manutenção de servidores em um sistema P2P é portanto menor, mas são maiores a complexidade de administração do estado do jogo (que agora está distribuído) e a manutenção da segurança do sistema. A Seção 8.2 deste documento trata mais detalhadamente deste assunto.

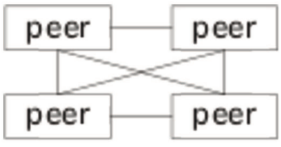


Figura 3.6: no modelo P2P todos os peers se comunicam dois a dois e o processamento é distribuído, aumentando a complexidade da manutenção de segurança.

Poucos jogos multiusuários utilizam arquitetura P2P. Age of Empires [10], da Microsoft, é um raro exemplo. Isto acontece pela complexidade da manutenção do estado e motivou a proposta de uma parceria entre a Unicamp e a Universidade Federal do Mato Grosso do Sul, um sistema híbrido que adota características interessantes de distribuição de carga entre os peers, mas mantém servidores apenas para algumas questões estratégicas. O modelo é citado no tópico 8.2.1 desta dissertação.

3.4 Jitter em jogos multiusuários

Um conceito "importado" de telecomunicações, usado freqüentemente em sistemas de tempo real, como *streaming* de áudio e vídeo, e que também é freqüente nos jogos multiusuários, é representado pela palavra *jitter*. Trata-se de uma anomalia no intervalo entre pulsos ou, no caso de transmissão digital, entre um pacote (ou datagrama) e outro, que provoca a sensação de pausa na transmissão ou saltos de etapas. Nos primórdios dos CDs, jitters por falha de leitura provocavam saltos em trechos da gravação. Nos jogos em tempo real, esta sensação de pausa pode tornar o jogo impraticável, fazendo, por exemplo, inimigos surgirem de repente ou, em um jogo de futebol, um atacante surgir na pequena área.

3.5 Bucket synchronization e dead reckoning

Em [16], Diot e Gautier aplicam em jogos um mecanismo chamado de *bucket synchronization* (BS) para tornar mais suave a execução de jogos em tempo real. Na prática, BS é a utilização de um *buffer* para manter a consistência de um jogo com p peers sem que cada um tenha que esperar, antes de executar o turno t, todas as $p - 1$ mensagens de estado sobre o turno $t - 1$. Na figura 3.7, os eventos gerados por Bob no tempo lógico t levam k turnos para chegar a Alice. Conseqüentemente, Alice só pode executar tais eventos em $t + k$. Se não fosse por BS, os jogadores teriam *jitter* de duração k. Por BS, o atraso da

execução é o mesmo, mas os movimentos são suaves (sem *jitter*).

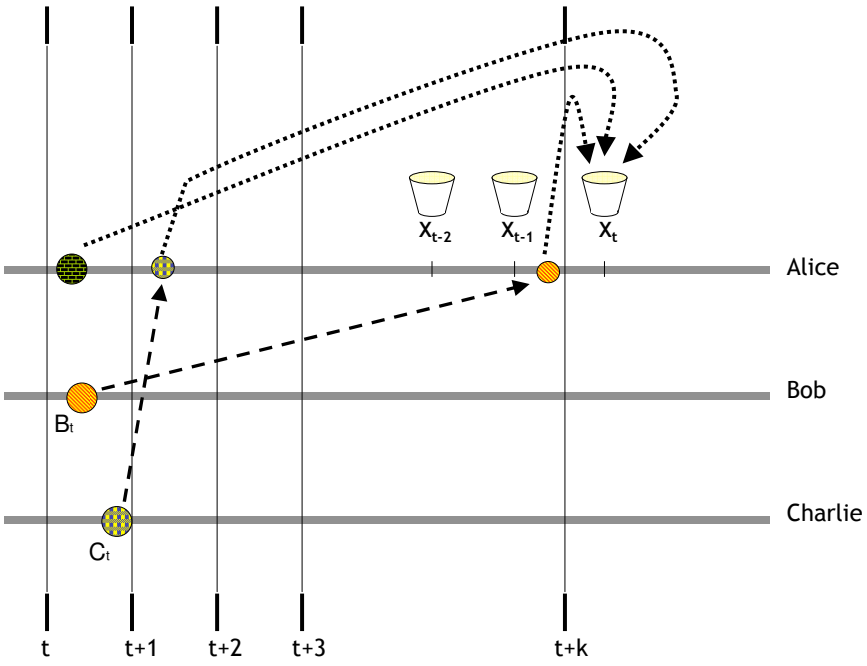


Figura 3.7: Os eventos gerados no turno t só serão interpretados por Alice no turno $t + k$, já que k é a latência entre Bob e Alice.

Para manter a suavidade nos jogos, a freqüência de *buckets*, f_X, deve ser a taxa de frames percebida pelo jogador. Em um jogo multiusuário confortável costuma-se utilizar entre 20 e 40 frames por segundo (portanto *buckets* por segundo). Portanto, para que o jogo seja viável em tal nível de conforto, a latência k deve ser menor que $1/f_X$, ou seja, menor que 50ms.

Percebe-se que também é importante para a manutenção da suavidade que em cada *bucket* haja uma notificação de evento de cada jogador. Se ao longo da execução do jogo algum dos alertas de eventos enviados por Bob a Alice se perderem ou atrasarem além de k, por exemplo, é preciso que Alice estime a movimentação de Bob. Tal estimativa é

chamada de *dead reckoning* (DR). Em [16], Diot e Gautier simplesmente repetem o último estado de Bob como DR; entretanto, outros algoritmos mais eficientes de DR podem ser aplicados. Em [5], por exemplo, utilizam-se dados de posições anteriores e posteriores para interpolar uma equação e estima-se uma posição mais adiante nesta curva. Uma maneira de fazê-lo suavemente, sempre considerando o vetor velocidade em cada instante, é o traçado de *splines cúbicas*, como exemplifica o artigo [6], publicado no site gameDev.net.

3.6 Diferenciar trapaças de smart playing

Ao me deparar com a tarefa de estudar como evitar trapaças em jogos, notei que o primeiro desafio era definir exatamente o que são trapaças. Em um jogo de cartas online, por exemplo, alguém pode usar dois usuários e dois computadores para ter acesso às cartas na mão de dois jogadores. Uma trapaça, porém difícil de evitar.

Em [50], Yan e Choi definem como trapaça qualquer comportamento que um jogador possa adotar para ter vantagens injustas (i.e. imprevistas no design do jogo) sobre os outros. Por esta definição, fica claro que o design de um jogo multiusuário deve ser cauteloso, desde a definição de regras até o estabelecimento do protocolo de comunicação entre as máquinas, seja em arquitetura cliente-servidor ou entre peers. Neste trabalho mantenho-me mais próximo das questões relacionadas aos protocolos e à manutenção de estados, o que me permite estudos mais generalizados, uma vez que as questões de segurança pertinentes à definição de regras variam muito entre um (estilo de) jogo e outro.

Capítulo 4

Taxonomia dos ataques a jogos online

A classificação dos ataques segundo métodos é importante por nos ajudar a cercar os sistemas com mecanismos de defesa. Embora possa haver ataques que não se encaixam em nenhuma das categorias abaixo, este capítulo tenta cobrir ameaças que nos são comuns hoje, algumas das quais podem ser impedidas ou dificultadas através de mecanismos como ciframento, verificação de integridade, autenticação e não-repúdio; outras são combatidas pela utilização de software que monitora a utilização dos jogos pelos usuários. Esta classificação se baseia em observações de casos de ataque, em artigos de análise de ataques específicos, como [4], [19], [20] e em artigos sobre classificação de ataques, como [50], [29], [51]. As Seções de 4.1 até 4.9 trazem classes de ataques. A Seção 4.11 traz uma tabela (Tabela 4.11) que sintetiza o conteúdo do capítulo, associando exemplos (apontados dentre os casos detalhados no Capítulo 5), nível de ocorrência (protocolo, infra-estrutura, aplicação) e possíveis soluções (dentre as detalhadas nos Capítulos 6 e 7).

4.1 Lookahead cheat

Uma trapaça trivial em um sistema distribuído é esperar chegar a mensagem de outro jogador no turno t e só então decidir qual será sua jogada no turno t, agindo como se a latência fosse maior do que ela realmente é. Exemplificando com um jogo de par-ou-ímpar, Alice escolhe par, Bob escolhe ímpar. Em um jogo em que Alice pretende mostrar o número três ($J_A = 3$) e Bob, o número zero ($J_B = 0$), Bob ganharia. Suponha que os dois enviem suas mensagens ao mesmo tempo ($t = 0$). Em $t = 2$, Bob recebe o jogo de Alice, carimbada com o tempo inicial ($t = 0$), dizendo que joga o número três ($J_A = 3$). Então Bob ganhou. Alice, em $t = 3$, recebe a mensagem de Bob (também datada de $t = 0$), dizendo que sua escolha é o número zero ($J_B = 0$). Alice então conclui que perdeu. Mas e se Alice, com más intenções, esperar chegar a mensagem de Bob e então responder, com data (falsa) $t = 0$, por exemplo $J_A = 2$, como poderá Bob saber, ao receber a mensagem, que Alice trapaceou?

Timestamp cheat e Fixed-delay cheat O exemplo acima é de uma trapaça de carimbo de tempo (*timestamp*), ou seja, Alice mente ao dizer que enviou a mensagem com $t = 0$, sendo que a enviou com $t > 0$. Uma maneira trivial, mas não muito eficaz, de perceber uma possível fraude em um sistema como este é notar que Alice sempre respondia em aproximadamente n milissegundos e, nesta ocasião especificamente, respondeu em $(n + k)$ milissegundos, sendo k um tempo ligeiramente superior à latência entre Bob e Alice. É uma tática de prevenção não muito eficiente porque, neste caso, Alice poderia aplicar um atraso fixo (*fixed-delay cheat*). Por exemplo, se a conexão entre Alice e Bob tem latência de 30ms e Alice aplica outros 50ms de atraso a todas as mensagens que responde, repito a pergunta anterior: como saberá Bob que Alice trapaceou? Protocolos como os de Baughman e Levine (seção 6.1) respondem a esta questão.

4.2 Supressão de atualizações (supressed update cheat)

De nome sugestivo, este ataque consiste em Alice receber normalmente as informações do jogo mas não respondê-las, esperando se aproximar do tempo-limite para então enviar uma mensagem, que chega logo antes que outros jogadores desconectem-na por timeout. Ou seja, Alice *provoca* jitter como forma de ataque.

A supressão de atualizações é um ataque representativo apenas para jogos em tempo real. Comparemos dois jogos do mundo real, um em realtime e outro por turnos, como exemplo. Quando se joga futebol, um jogo totalmente em tempo real, o terror de todo goleiro é que um atacante apareça de repente em um local imprevisto e marque um gol indefensável. Isto pode acontecer se, em um sistema de jogo multiusuário, o protocolo contempla o envio da localização e não dos movimentos do jogador. Em um jogo por turnos, como uma partida de "bolinha de gude", este problema não permite um ataque interessante, visto que todos esperam o final de um turno para que então o próximo jogador, depois de notar onde foi parar a bolinha do adversário, arremesse a sua.

4.3 Trapaças por conluio

Em vários jogos, saber a situação de um dos jogadores pode ser grande vantagem contra todos os outros. Um exemplo simples é um jogo de pôquer: se Alice tem dois ases e sabe que Bob tem um ás, então fica trivial dizer que ninguém mais tem dois ases, já que em todo o jogo só existem quatro deles. Se Alice estiver mancomunada com Bob, ou (melhor ainda) se Alice e Bob forem dois *logins* de uma mesma pessoa, os outros jogadores estão definitivamente em desvantagem.

4.4 Inserção de Bots

Jogos em tempo real normalmente exigem do jogador, além do raciocínio para definição da estratégia, reflexos. Os reflexos são ainda mais importantes se várias decisões têm que ser

tomadas simultaneamente - se são várias decisões simples, é preciso ter bastante reflexo e quase nenhum raciocínio. Esta é a oportunidade ideal para se trapacear com agentes artificiais "inteligentes", os chamados Bots. Dentre as maneiras de fazer um agente de software responder no lugar do jogador estão: adulteração do software-cliente; espionagem no input que o software-cliente recebe pela rede e geração de sinais nas interfaces de mouse e teclado (sinais que o software-cliente recebe do jogador). Embora o segundo modelo seja mais discreto para o sistema (portanto mais difícil de ser detectado), o primeiro é, a princípio, mais eficiente e, em jogos desprotegidos, mais simples de se implementar.

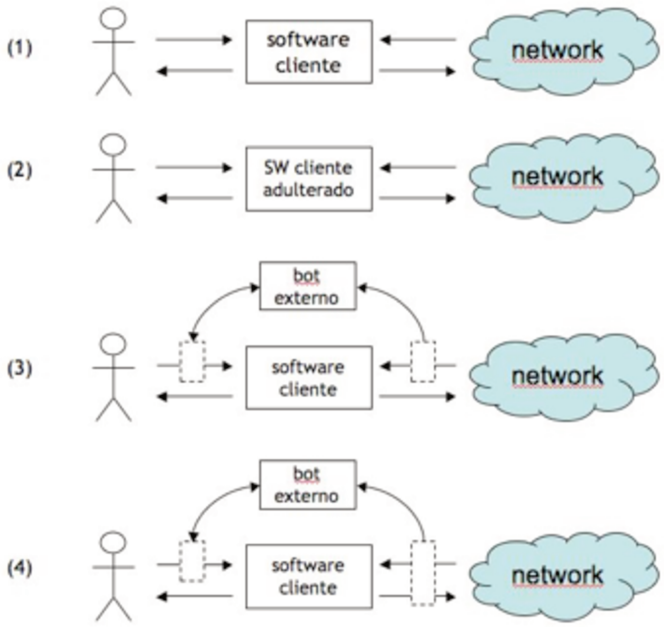


Figura 4.1: inserção de bots pode ser feita basicamente de duas maneiras - alterando-se o cliente ou usando um processo-espião externo. O modelo (1) é o jogo normal. Os modelos 2, 3 e 4 inserem agentes artificiais.

A provável maior eficiência do modelo 2 da Figura 4.1 vem do fato de, atuando "dentro" do software cliente adulterado, o robô conhece praticamente todos os estados do jogo, enquanto o robô externo ao código (Figura 4.1, modelo 3) tem acesso aos updates de es-

tado (que vêm da rede e da interface de input), mas não sabe exatemente o que acontece no software cliente. Em casos de clientes mais simples, pode-se reproduzir a máquina de estados por engenharia reversa ou escutar o output do cliente para a rede (modelo 4).

Em jogos de tiro em primeira pessoa, como Half Life e Quake, trapaças com Bots para ajudar a mirar (AimBots) são bastante comuns. Um exemplo é o pacote Unforgiven Hook, que também implementa outras trapaças, como *wall hacks*, e portanto é citado novamente em 4.9.1.

Em 7.2, discute-se o combate a Bots em jogos através da aplicação de CAPTCHAS, que parecem ser pouco adequados aos jogos digitais.

4.5 Fuga (Game escaping)

"Game escaping", seja não finalizando um protocolo em momento inadequado ou simplesmente desconectar para não completar alguma ação é um ataque bastante simples de se implementar e conseqüentemente comum. Em um jogo de xadrez, por exemplo, onde uma vitória significa ganhar pontos no ranking e uma derrota, perdê-los, a fuga é interessante para o atacante: ao entender que seu jogo está praticamente perdido, o "potencial derrotado" pode, em vez de declinar e admitir a derrota, "puxar o cabo de rede" de seu computador (ou simplesmente derrubar a conexão de maneira menos violenta) e fazer o oponente e o site de ranking pensarem que a conexão caiu acidentalmente. Se a regra do ranking for "em caso de desconexão, ninguém perde nem ganha", o atacante evita sua queda no ranking.

4.6 Falsificação de ativos virtuais

O comércio de ativos virtuais é cada vez mais comum nos jogos online. Seja com moeda virtual ou com dinheiro de verdade, os ativos virtuais têm valor real. Mesmo que só possam ser adquiridos com tempo de jogo (e não com dinheiro), a venda de ativos virtuais

no mundo real, atividade cuja "indústria" é conhecida por Gold Farming, chega a ser fonte de receita de empresas, como a UCDao[49], que chega a vender personagens pré-cultivados por 600 dólares.

Alguns jogos, como a versão brasileira do Priston Tale[26], vivem não da venda de assinaturas, mas da venda de ativos. Jogos que têm este modelo de negócio movimentam na Coréia cerca de 10 bilhões de wons por mês, ou 10 milhões de dólares americanos[38].

Ou seja, além de prejudicar a jogabilidade e conseqüentemente desvalorizar o jogo, causando prejuízo aos desenvolvedores, a falsificação de ativos virtuais interfere em uma economia paralela, formada por negócios virtuais executados com moedas reais.

Para falsificar ou roubar ativos virtuais, falhas de protocolo de troca que não observam procedimentos de "troca justa" são comumente usadas. Se Alice quer vender um item a Bob, por exemplo, o justo seria Alice entregar o item mediante pagamento de Bob. No entanto, se o protocolo não trata a interrupção da troca no meio, Alice pode receber o dinheiro de Bob e não entregar o produto.

4.6.1 Gold farming é trapaça?

Como Gold Farming é uma atividade que pode prejudicar a *jogabilidade* ao permitir, por exemplo, que um jogador "rico" compre um personagem poderoso sem ter que passar pelos caminhos naturais de evolução em um jogo (e portanto burlando regras), alguns jogadores consideram a ação uma trapaça. Porém, este trabalho não trata de combater esta atividade, já que ela acontece no mundo real e dificilmente poderá ser evitada através de mecanismos criptográficos.

A discussão, no entanto, existe. Em alguns países, está em pauta a proibição do dinheiro virtual[48], o que poderia ser prejudicial para certas economias. Pesquisa da KIPA e do Promotion Institute, mencionada em matéria do jornal Korea Times[48] estima que o mercado de jogos que cobram por itens movimente cerca de 1 bilhão de dólares por ano.

4.7 Falsificação de identidade

Falsificar identidade é um truque antigo. Assinar documentos como se fosse outra pessoa, dizer ser alguém que não é para ter acesso a locais ou informações restritos, passar por alguém ao telefone ou até mesmo usar documentos do irmão mais velho. Podem-se listar aqui inúmeras maneiras de se falsificar identidade, algumas mais modernas, outras milenares.

No mundo online, como não se vê a pessoa propriamente, torna-se aparentemente fácil mentir a identidade. Quase todos os sistemas online que se preocupam com identidade hoje tem um padrão de identificação que envolve senhas. Alguns sistemas mais sofisticados pedem uma composição de algo que a pessoa conhece (senha) com algo que ela possui (um cartão, por exemplo), como os sistemas de banco. Divido (certamente como vários outros estudantes do assunto) os métodos de se falsificar identidade no mundo virtual em duas classes: aproveitamento de falhas do sistema e engenharia social. A primeira engloba alguns tipos de ataque, desde a tentativa de descobrir senha por força bruta até o seqüestro de sessões, passando por roubo de bases de dados (que possam conter logins e senhas, por exemplo). A segunda, mais complicada de se combater, envolve a descoberta de informações, clonagem de cartões ou outro roubo de ativos-chave (virtuais ou reais) através de relações pessoais.

4.7.1 Seqüestro de sessão

Uma das maneiras de se passar por alguém no mundo online é seqüestrando uma sessão. É como se Alice telefonasse a Bob e, de repente, Evo tomasse o telefone da mão de Alice. No telefone isso seria engraçado, porque Evo teria que imitar a voz de Alice e talvez Bob percebesse. A voz ao telefone funciona como uma assinatura. No mundo online, como nem sempre se assina cada mensagem (porque isso pode custar caro), o seqüestro de sessão é menos perceptível.

Boa parte dos jogos multiusuários utilizam protocolo UDP com o intuito de eliminar a

necessidade de se manter uma conexão vitalícia, conseqüentemente ganhando desempenho. Entretanto, descrevem-se neste tópico exemplos de seqüestro de sessões TCP e UDP.

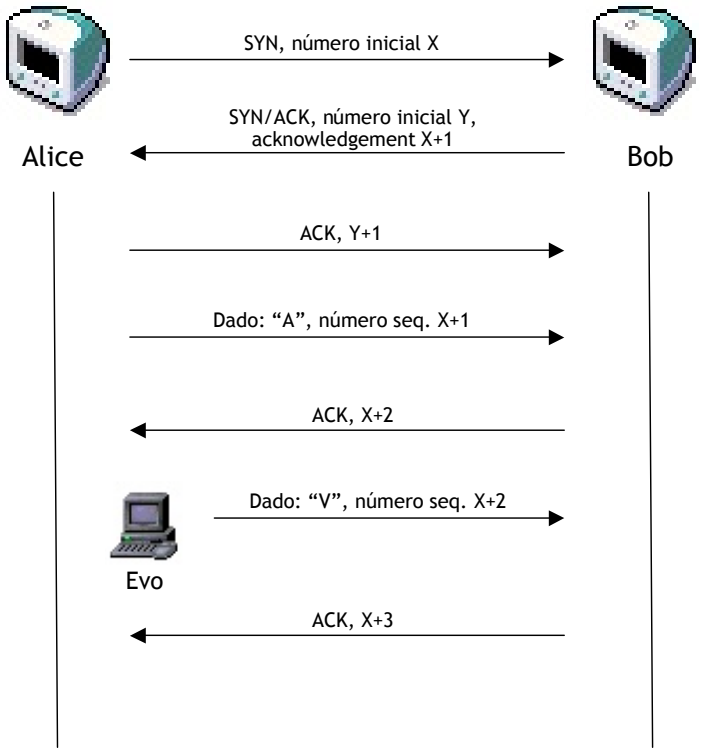


Figura 4.2: exemplo de inicialização de um protocolo similar ao TCP entre Alice e Bob, com uma tentativa de seqüestro de sessão por Evo.

Seqüestro de uma sessão TCP O protocolo TCP é utilizado (principalmente) por sua confiabilidade na entrega de pacotes em ordem. Pra isso ele é de fato muito bom. Se Alice envia a Bob a seqüência ABAADJJKIT, um caractere em cada pacote, não há risco de Bob receber fora de ordem. Mesmo que trafeguem e cheguem fora de ordem, o sistema reordena os pacotes antes de entregá-os a Bob. Na prática, se a letra “D” se atrasar, atrasam também JJKIT. Os arquitetos de sistemas distribuídos - principalmente

os que se preocupam com desempenho - gostam sempre de mencionar que, em contrapartida, TCP demanda o estabelecimento de uma conexão. Mas, como o objetivo aqui é discutir seqüestro de sessão, prefire-se neste trabalho não falar do overhead de estabelecer a conexão, mas sim em como TCP garante a ordem de seus pacotes.

A ilustração da Figura 4.2 mostra numeração de mensagens. O protocolo numera seqüencialmente os bytes.

Para seqüestrar a sessão, o atacante (na figura representado por Evo) precisaria, por alto, dos seguintes passos:

1. Falsificar o número IP para parecer Alice perante Bob;
2. Saber a seqüência certa que Alice e Bob estão usando para se comunicar (qual o número $X + k$);
3. Enviar dados falsos para Bob, acompanhados do número seqüencial $X + k$, antes que Alice envie o pacote que se inicia com o k-ésimo byte.

Para evitar que Alice envie o pacote que se inicia com o k-ésimo byte, Evo pode provocar uma inundação de requisições, atrapalhando Alice a processar sua comunicação com Bob. Esta inundação pode ser um ataque a partir de outro local, como por exemplo um ataque de negação de serviço distribuído. Falo mais sobre negação de serviço na seção 4.8 desta dissertação. Na figura fica indefinido o que Alice faz com a confirmação (Ack, $X + 3$) que recebe de uma mensagem que nunca enviou. Na verdade, existem algumas variações a partir daí, algumas das quais evitam que Alice receba a confirmação, mas prefiro não me aprofundar neste assunto.

Seqüestro de uma sessão UDP . Em UDP é, a princípio, ainda mais fácil seqüestrar uma sessão. Como o protocolo não estabelece conexão e não utiliza números seqüenciais nas mensagens em nível de transporte, falsificar identidade em UDP passa a ser quase totalmente uma questão de aplicação. Ou seja, se o atacante Evo quiser se passar por

Alice, ele pode executar o passo de ataque mostrado em TCP sem sequer se preocupar em qual é o número X.

4.7.2 Escuta clandestina (grampo)

Instalar uma escuta clandestina no PC de um jogador adversário pode revelar login, senha e permitir a falsificação de identidade. A escuta pode ser instalada de várias formas. Há vírus e cavalos de tróia criados especialmente para isto, mas a escuta também pode ser instalada em locais por onde a informação trafega aberta.

4.7.3 Usando força bruta

Embora não estejam listadas aqui todas as maneiras possíveis de se falsificar identidade, certamente usar força bruta é a mais lenta.

Usar força bruta para descobrir senha (ou até um login, às vezes) significa tentar todas as combinações possiveis. Leva um longo tempo, mas existem alternativas um pouco mais inteligentes, às quais atribuo a mesma categoria (por ter métodos de defesa similares), como varrer palavras de um dicionário. Ou seja, já que é comum o uso palavras como senhas, tentar vinte mil verbetes é mais fácil do que varrer todas as combinações de 10 letras. Se não contarmos números nem caracteres especiais, sobram as maiúsculas e minúsculas. Quarenta e seis. São 46^{10} combinações, o que é bem próximo de $(10^{1,66})^{10} = 10^{16,6}$. Considerando que o número de palavras no dicionário é da ordem de 10^4, é um trilhão de vezes mais difícil varrer todas as combinações de letras.

Defesas contra força bruta envolvem, por exemplo, dificultar que se tente várias vezes uma senha (obrigando uma espera de alguns segundos entre uma tentativa e outra).

4.8 Negação de serviço a peers (DoS to peers)

Há algumas razões pelas quais um atacante possa querer impedir um peer do jogo (ou um cliente) de se comunicar. Uma delas, como mencionado no item 4.7.1 desta dissertação,

seria facilitar um seqüestro de sessão. Outra poderia ser simplesmente impedir o peer de realizar uma jogada, fazendo com que ele perca tempo ou tornando suas reações mais lentas. Em jogos de tiro, como Quake, isso pode significar tornar-se um alvo fácil.

Os ataques de negação de serviço (DoS) podem vir de um só IP, quando fica fácil barrá-los. No entanto, o que realmente preocupa são DoS distribuídos. Normalmente o atacante faz (ou simplesmente configura) um vírus ou um *trojan* que se espalha por computadores alheios e os utiliza para atacar. Ou seja, muitas vezes quem lança rajadas para atrapalhar o serviço de um peer nem sequer nota que o faz.

4.9 Acesso a informação de forma desleal

Aqui é impossível não haver intersecção com o tópico "Falsificação de identidade"; mas, como há pontos distintos importantes, os dois métodos de trapaça merecem suas próprias categorias. Falsificação de identidade já é feita, muitas vezes, para se acessar informação confidencial, como dito na seção 4.7. Nesta seção, entretanto, refiro-me às outras formas de acesso a informação confidencial, sobretudo àquela informação que o projetista do jogo confia à aplicação cliente e pensa equivocadamente que um jogador mal-intencionado não terá acesso "já que a aplicação não permite".

4.9.1 Alteração da estrutura do cliente

A aplicação-cliente recebe, muitas vezes, informações que não revela ao jogador ou que armazena para revelar mais tarde, entre as quais estão o posicionamento dos inimigos, estado dos inimigos (energia, armas, munição etc), posicionamento de armas e munição no mapa do jogo.

Para obter estas informações, alguns atacantes modificam a aplicação, como alterar o mecanismo de Z-Buffer de motor de jogos para ver inimigos atrás de paredes, ou sabotam os recursos que ela utiliza, como substituir o OpenGL32.dll para tornar tudo no jogo levemente transparente. Estes mecanismos estão exemplificados, respectivamente, nas

Figuras 4.3 e 4.4.

Figura 4.3: exemplo de *wallhack* no jogo Counter Strike implementado por uma modificação na aplicação.

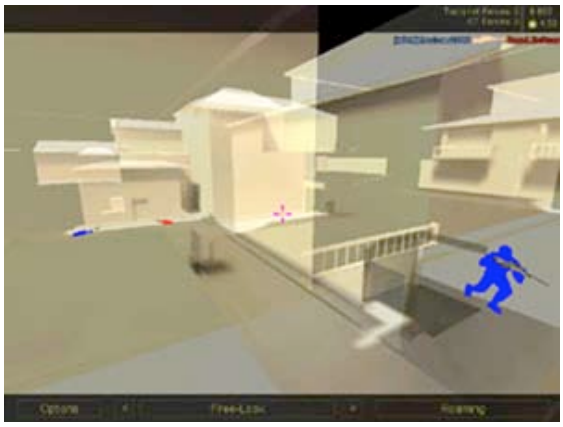


Figura 4.4: exemplo de *wallhack* em Half Life implementado através da modificação do OpenGL32.dll.

4.9.2 Leitura e modificação de dados na memória (memory hooking)

Outra forma de atacar o jogo em execução para obtenção de informações privilegiadas é acessando a região de memória utilizada pelo jogo.

É o que faz o software Unforgiven Hook no jogo Counter Strike, cujo exemplo é mostrado na Figura 4.5. Além de revelar informações na forma de texto, Unforgiven Hook faz trapaças mais sofisticadas, como mostrar inimigos que estão além de paredes (wall hack) ou destacar os inimigos (cham hack) tornando-os mais facilmente identificáveis. Implementam ainda AimBots, como os mencionados na seção 4.4.

Vários outros pacotes como Unforgiven Hook são facilmente encontrados em sites especializados.

Figura 4.5: pacote de ferramentas para *hooking* em ação.

4.9.3 Escuta dos dados enviados e recebidos pela aplicação cliente

Em jogos que não adotam mecanismos seguros de conexão, pode-se obter informação observando os dados que trafegam pela interface de rede. Neste caso, a aplicação de cheating, em vez de modificar o jogo, terá que funcionar conjuntamente com o jogo, prestando atenção nos estados do *game* e mostrando ao jogador. É um ataque menos eficiente que modificar o jogo, já que fica mais complicado entregar a informação ao usuário (esta aplicação teria que competir com o jogo na utilização da placa de vídeo em vez de usar os recursos do próprio jogo) e é mais fácil ser evitado - pode-se fazê-lo utilizando canais seguros ou simplesmente cifrando algumas informações-chave. É um modelo ineficaz para ataques como wall hacking, já que fica difícil informar ao jogador onde estão seus inimigos sem ter acesso à visão exata (renderizada).

A Figura 4.6 mostra exemplos de esquemas obtenção de informações privilegiadas em jogos.

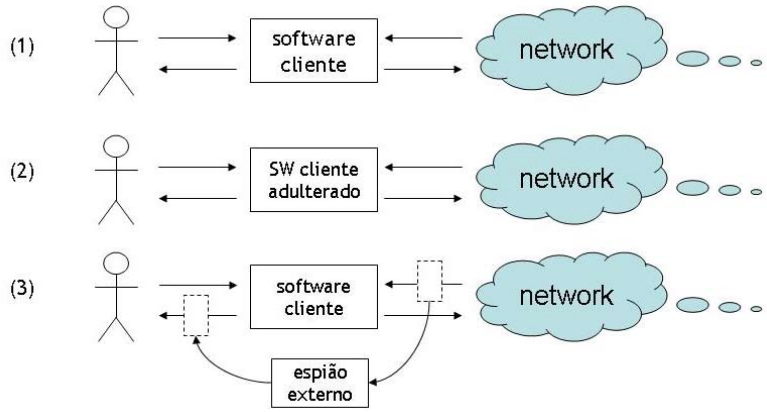


Figura 4.6: duas das várias maneiras de se obter informações privilegiadas em jogos digitais. O modelo (1) é o jogador honesto. O modelo (2) usa um cliente modificado, como citado em 4.9.1. O modelo (3), que pode funcionar principalmente em jogos mais simples, utiliza uma aplicação para escutar o que o jogo ouve via rede, interpreta a informação e entrega ao usuário.

4.10 Alteração de informação estratégica

Considerando que não se tem acesso ao servidor de jogo, a alteração da informação pode se dar de três formas: modificando os dados locais para acrescentar recursos que não estavam antes disponíveis; adaptando o cliente do jogo para criar um software-cliente "envenenado", capaz de reportar ao servidor ações que não aconteceram; utilizando um software que escuta as mensagens enviadas pelo jogo ao servidor e as altera sob comandos do jogador. A alteração de informação estratégica está exemplificada nas seções 5.1 e 5.2.

4.11 Taxonomia de ataques a jogos: síntese

Tabela 4.1: Taxonomia de ataques a jogos: classificação segundo nível de ação e exemplos de tratamento.

Ataque	Tratado em nível	Exemplo de vulnerabilidade	Exemplo(s) de tratamento
4.1 Lookahead cheat	protocolo	jogos de ação simultânea em geral	6.1.1 Lockstep
4.2 Supressão de atualizações	protocolo	jogos em tempo real	6.4.1 Basic NEO
4.3 Ataques por conluio	protocolo	lookahead cheat por conluio [6.1.2]	6.4.1 Basic NEO
4.3 Ataques por conluio	game design	jogos de carta	n/a
4.4 Inserção de Bots	aplicação	Quake [5.4]	7.1 PunkBuster ; 7.2 CAPTCHAS
4.5 Game escaping	game design	jogos online com classificação (ranking)	n/a
4.6 Roubo de ativos virtuais	game design	StarCraft e Age of Empires [5.3]	correção em versões posteriores
4.6 Roubo de ativos virtuais	protocolo	jogos com mecanismo de compra e venda	utilização de protocolos de troca justa
4.7 Falsificação de identidade			
4.7.1 seqüestro de sessão	protocolo	n/a	autenticação forte
4.7.2 grampo	protocolo	n/a	ciframento
4.8 Negação de serviço a peers	infra-estrutura	jogos online por tempo real	redundância de infra
4.9 Acesso desleal a informação			
4.9.1 alteração do SW cliente	aplicação	wall hack em Counter Strike [Figura 4.3]	7.1 PunkBuster; 7.4 Atestação
4.9.2 acesso à memória	aplicação	kits de hooking em geral [Figura 4.5]	7.1 PB; 7.4 Armazenamento protegido
4.9.3 escuta na aplic.cliente	protocolo	bots [Figura 4.1]	ciframento
4.10 Alteração de informação	aplicação	Diablo [5.2]	decisões server-side

Capítulo 5

Estudos de casos de vulnerabilidades

Este capítulo cita exemplos de ataques a jogos consagrados para facilitar o entendimento de algumas das vulnerabilidades mencionadas no Capítulo 4.

5.1 Counter-strike

Counter-Strike, publicado em junho de 1999 pela Sierra On-Line[44] (hoje pertencente à Vivendi Universal) é uma modificação do jogo Half Life, por sua vez lançado pela empresa Valve no final de 1998. O jogo, que pode ser resumido como um *First-Person Shooter* (FPS) de terroristas contra anti-terroristas, tornou-se rapidamente o jogo do gênero mais jogado na história. Em julho de 2006, Counter-Strike ainda era disputado durante quase 5 bilhões de minutos por mês[11] segundo a própria Valve, enquanto "Half-Life 2", outro jogo do gênero, atingia 33 milhões - menos de 1% do uso.

No artigo "Understanding Cheating in Counterstrike", Christopher Choo [7] descreve alguns ataques a Counter-Strike dentre os citados no Capítulo 4 desta dissertação. Ele divide os ataques em "trapaças no servidor", "trapaças no cliente" e "trapaças no driver de vídeo". Em trapaças no servidor, Choo se refere a alterar mensagens que o cliente envia para o servidor. O exemplo de Choo é o "skin cheat", ou seja, alterar as mensagens, dizendo ora que o avatar é um terrorista, ora um anti-terrorista, confundindo os inimigos

e se infiltrando em locais estratégicos sem que percebam.

Choo atribui esta falha ao servidor de Counter-Strike, tratando exclusivamente a questão de "mudança de pele" como uma funcionalidade que não precisaria estar disponível durante o jogo. Segundo ele, não demorou para que a Valve corrigisse o servidor ignorando exatamente este tipo de comando. No entanto, eu prefere-se neste trabalho generalizar a falha e considerá-la como falta de verificação da integridade e autenticidade de mensagens, o que permite a alteração. Na taxonomia do Capítulo 4, esta falha é classificada como "Alteração de informação estratégica" e resolver esta questão como propõe a Seção 6.5 dificulta, de maneira mais eficaz, falhas deste gênero. Em trapaças no cliente, Choo [7] cita como exemplo a modificação do cliente para a inserção de bots (classificada na seção 4.4). Como Counter-Strike é um jogo "expansível", alguns *gamers* aproveitam esta expansibilidade para adicionar rotinas automatizadas de mira ou outras melhorias de reflexo. Choo fala ainda de acesso a informação confidencial (classificada em 4.9), como a de enxergar através de paredes.

Quanto a trapaças modificando drivers, Choo [7] conta com a possibilidade de se reescrever um driver e pedir a uma placa de renderização 3D para renderizar as texturas com determinada transparência e permitir a visão através das paredes. No entanto, esta trapaça não é trivial de se evitar. Existem inúmeros drivers de vídeo e seria inviável para um designer de jogos ou de middleware se preocupar com cada um dos possíveis drivers, mesmo porque há novos dispositivos de vídeo a cada dia. O ideal seria utilizar um mecanismo de atestação, como o citado na seção 7.4. Adicionalmente, este é um ataque pouco interessante em termos de jogabilidade: ter transparência em todas as texturas do jogo (já que a placa não diferencia paredes no cenário de modelos 3D) faz com que ele fique menos interessante. Ou seja, mesmo que o jogador mau consiga implementar este ataque, ele teria a penalidade de ver seu jogo piorado.

5.2 Diablo e Diablo II

Outro caso conhecido de falha de segurança na indústria de jogos é com a série Diablo, da Blizzard Entertainment. A primeira edição do jogo, lançada em 1997, foi um incrível sucesso e ganhou vários prêmios de jogo do ano[1]. Sua versão multiusuário, considerada revolucionária, foi o grande diferencial. E a grande dor de cabeça também. Em tempos de Internet discada (dial-up), os desenvolvedores de Diablo decidiram poupar o servidor central e reduzir latência entre os peers distribuindo o processamento. A falta, entretanto, foi no planejamento da solução distribuída. Rápido e confortável mesmo com os modems da época, o sistema de distribuição de Diablo pecava porque deixar todas as informações sobre o personagem na máquina do cliente era um convite às trapaças de alteração de dados estratégicos.

Assim, começaram a surgir personagens super-poderosos, os quais demorariam meses para se formar honestamente, na mão de jogadores que não sabiam sequer como atacar um monstro - a primeira tarefa que se aprende no jogo.

Outro problema de segurança no jogo foi a morte súbita de jogadores. Um jogador A conseguia, através de um comando via mensagem (de rede), dizer ao cliente do jogador B que seu personagem, P_B, havia sido atacado pelo personagem P_A com força suficiente para perder 100% de energia. E o cliente utilizado por B confiava na mensagem!

Na versão seguinte do jogo, Diablo II, estes problemas foram corrigidos. A segunda edição havia vendido, até 2003, incríveis três milhões de cópias[2]. Para evitar a edição de personagens, a Blizzard ofereceu recursos opcionais que chamou de Realms; quem jogasse nos Realms poderia gravar os dados dos jogadores em um servidor. Apenas quem estivesse nos Realms jogava contra outros nos Realms. Ou seja, o jogador poderia optar por continuar no esquema antigo (mais rápido), mas sob sua conta e risco. Porém, uma outra falha menos grave foi descoberta nesta versão - uma trapaça que permitia ao jogador ver os monstros em todo o mapa. E os Realms foram invadidos em dezembro de 2000, mas

[1] `http://www.blizzard.com/inblizz/awards.shtml#diablo`
[2] `http://www.blizzard.co.uk/press/030814war3.shtml`

a Blizzard rapidamente restaurou os backups e o jogo ficou pouco tempo fora do ar. Pelas vendas registradas até 2003, parece que estes problemas menos graves não incomodaram muito os consumidores.

5.3 Age of Empires e Starcraft - a falha de cancelamento de construções

Starcraft - da mesma Blizzard mencionada acima em Diablo - e Age of Empires, da Microsoft, são ultra bem-sucedidos jogos de estratégia em tempo real (RTS). Ambos apresentaram uma falha de design que possibilitava a falsificação de ativos virtuais. O requisito para que a falha acontecesse era um atraso enorme na comunicação por rede. Durante o *lag*, o jogo ficava "congelado", mas para não parecer travado (aos olhos do jogador), a interface continuava funcionando. Se houvesse construções em execução (congeladas, neste caso), era possível emitir o comando de cancelá-las. Normalmente só era possível cancelar construções uma vez por turno de comunicação. Entretanto, como o jogo estava suspenso, os turnos não progrediam e o jogador cancelava várias vezes. Resultado: no final de um turno havia vários cancelamentos de uma mesma construção que só era destruída no turno seguinte. E o jogador já havia recebido várias vezes o dinheiro de volta pela construção cancelada.

5.4 Quake

Desde que a iD Software abriu ao público o código de Quake, um problema de segurança surgiu: o de alteração do cliente para trapacear no jogo. A solução inicial de se pedir a um cliente open-source para atestar-se não era suficiente, já que uma modificação também no mecanismo de atestação poderia fazê-lo simular a autenticidade.

Um cliente modificado pode expor informação confidencial ao jogador (vide seção 4.9) ou permitir a inserção de bots (Seção 4.4).

Carmack sugeriu em seu ".plan" (seção A) que fossem utilizados trechos de código fechados para verificar a autenticidade da parte open-source. A solução desagradou duas correntes: a de defensores de open-source e a de profissionais de criptografia aplicada à segurança, que defendem (com razão) que a segurança alcançada através de obscuridade é frágil.

Trocando em miúdos, o problema de Quake é o mesmo que acontece em inúmeros jogos: o servidor confia no cliente para a tomada de decisões e para não mostrar informação que seja confidencial. Seria possível abrandar este problema com a concentração de mais decisões sobre o servidor, com a utilização de um cliente blindado, com um canal seguro até o servidor (dificultando a aplicação de "aim bots") e reduzindo as artimanhas de desempenho de Quake, cujo servidor envia informações previamente (para ganhar eficácia ao jogar em rede) e deixa a cargo do cliente decidir quando mostrá-la.

Uma possível solução partiu dos autores do Terra VM (vide Seção 7.3), que implementaram uma versão confiável do jogo blindada por uma máquina virtual. A versão foi batizada de "Trusted Quake" (vide 7.3.3).

5.5 Second Life - pirataria de ativos virtuais

Em novembro de 2006, o emergente Second Life, universo virtual de milhões de usuários, anunciara um problema de segurança relacionado com a cópia de ativos.

O Second Life é um universo de acesso gratuito que permite aos usuários criar itens e vendê-los por moeda virtual. Com moeda virtual, um usuário pode adquirir itens virtuais, terrenos virtuais ou ainda transformar seu dinheiro virtual por dólares reais, no câmbio virtual ou em um câmbio paralelo criado pelos residentes de Second Life. Ou seja, a obtenção ilegal de ativos virtuais é facilmente conversível em obtenção ilegal de dinheiro real.

Em novembro de 2006, os sites "GameSpot" e "news.com" divulgaram uma notícia [17] preocupante: através da funcionalidade de permitir a usuários que executem scripts dentro

do Second Life para replicar seus próprios itens (como implementar uma máquina de refrigerantes, por exemplo), era possível fabricar itens idênticos a outros, ou seja, piratear ativos virtuais. O script simplesmente examinava a composição de um item e criava outro igual.

Até o momento, o ambiente não fornece uma maneira de se certificar itens para verificar a autenticidade. Um jogador não pode diferenciar, portanto, um ativo pirata de um original.

Uma solução possível, mas não tão eficiente, seria registrar cada ativo perante uma entidade confiável para seus fabricantes. Neste caso, uma cópia do ativo seria uma cópia "não autenticada". A solução soa ineficiente por não parecer compatível com o grau de liberdade e criação que o universo propõe. Cada transação (venda ou compra de um item original) teria que ser informada a uma autoridade credenciada. Talvez seja o caso, para ativos realmente valiosos, de disponibilizar (como serviço opcional) cartórios virtuais para registros de ativos. Neste caso, somente entidades autorizadas pelo cartório poderiam copiá-los e certificá-los.

Capítulo 6

Soluções de segurança em nível de protocolo

Este Capítulo reúne e estuda soluções de segurança propostas para vulnerabilidades em nível de protocolo, como *lookahead cheat*, supressão de atualizações e algumas variações de ataques por conluio (vide capítulo 4).

6.1 Baughman e Levine

No artigo "Cheat-Proof Playout for Centralized and Distributed Online Games" [4], Baughman e Levine propõem uma solução simples para o *lookahead cheat* através de confirmação antecipada utilizando funções de resumo. A primeira solução, mostrada na Seção 6.1.1, peca ao limitar a velocidade do jogo em função do peer mais lento, portanto apenas utilizável, na prática, em jogos por turno (como xadrez, batalha naval ou ainda par-ou-ímpar), ou seja, jogos que não têm interação em tempo real. A segunda versão, comentada em 6.1.2, tenta deixar o jogo mais livre e talvez possa ser utilizada em jogos realtime (desde que a latência média entre os hosts permita).

6.1.1 Lockstep protocol

A proposta é dividir cada turno em dois passos. Em um primeiro, todos enviam um resumo criptográfico de seu movimento. Depois, todos revelam seu passo em texto claro. Na prática, é como confirmar (commit) uma operação antes de revelá-la. Ou seja, somente após ter enviado o resumo de seu passo um jogador ficará sabendo o passo dos oponentes, evitando o "lookahead cheat". Segundo os próprios autores, a solução funciona em jogos por turnos mas exige uma troca de mensagens a mais em cada rodada, o que torna a comunicação lenta para jogos mais dinâmicos. Em jogos executados em tempo real, esta alternativa é praticamente inviável, uma vez que ela nivela a velocidade dos turnos conforme a latência do *peer* mais lento.

Este protocolo padece ainda de vulnerabilidade ao "supressed update attack" (vide seção 4.2), ou seja, a supressão de atualizações por parte de um dos peers pode danificar o jogo de todos os participantes. Em jogos por turno, no entanto, a supressão de atualizações impede a progressão do turno em vez de causar surpresas, o que torna a tática menos interessante para quem ataca.

6.1.2 Asynchronous synchronization (AS)

Para amenizar o problema de baixa performance no protocolo de lockstep, os próprios autores (Baughman e Levine) otimizaram o protocolo para somente implementar a sincronização quando suas ações puderem influenciar no jogo um do outro. Para isso, o autor isolou as áreas de interesse de cada peer como esferas de influência (SOI) em torno do respectivo avatar e somente sincroniza os dois avatares quando as esferas de interesse se interceptam.

A Figura 6.1 exemplifica as esferas de influência. No host local l, sabe-se exatamente qual a esfera de influência no tempo t. A informação sobre o host remoto h, no entanto, pode ainda não estar disponível. Para solucionar a questão, faz-se uma dilatação da esfera, considerando que a cada turno o host remoto pode se mover um determinado

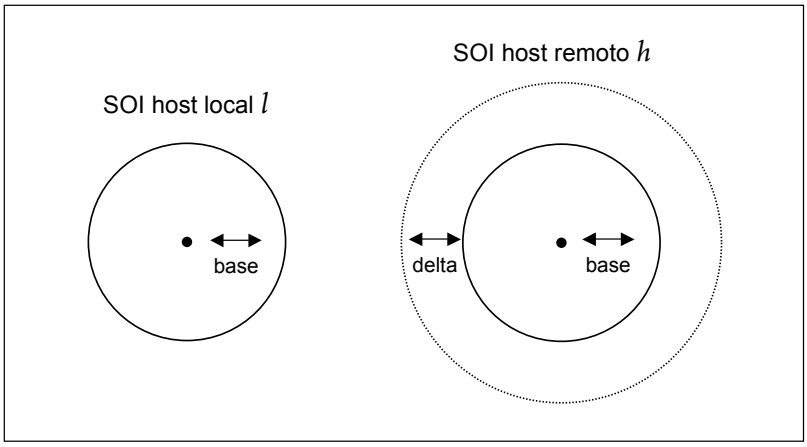


Figura 6.1: esferas de influência (SOI) de hosts locais e remotos.

delta. Supondo que o último turno disponível sobre a esfera remota seja $t - d$, soma-se $d * delta$ ao raio da SOI e obtém-se a projeção de uma zona de potencial influência do host h para o tempo t. A dilatação está exemplificada na Figura 6.2.

O protocolo Asynchronous Syncronization (AS) está descrito formalmente no quadro da Figura 6.3. Leva-se em consideração que um host l que está em um turno t recebe as informações dos hosts remotos (contidos em R) e verifica se existe dependência de algum. Se sim, ele executa o passo de sincronização do *lockstep protocol* para aqueles hosts h que, no turno t, dependem de l (e dos quais l depende, já que $a \cap b = b \cap a$). Como pode ser observado na Figura 6.3, esta sincronização se dá eventualmente entre dois ou mais dos players e portanto os peers podem progredir no jogo de maneira assíncrona, i.e. não necessariamente todos no mesmo turno, o que explica a melhor performance do AS frente ao *lockstep* original.

No entanto, por ser assíncrono, o AS é passível de ataques de *lookahead* provocados por conluio, ou seja, caso um grupo de usuários se comunique para efetuar o *lookahead cheat* ou um usuário tenha mais de um login. Exemplificando, suponha que Alice e Bob estejam em situação tal que suas esferas de influência (SOI) se interceptem no turno t. Charlie, amigo de Bob, também tem SOI interceptando a de Alice. Então Charlie efetua

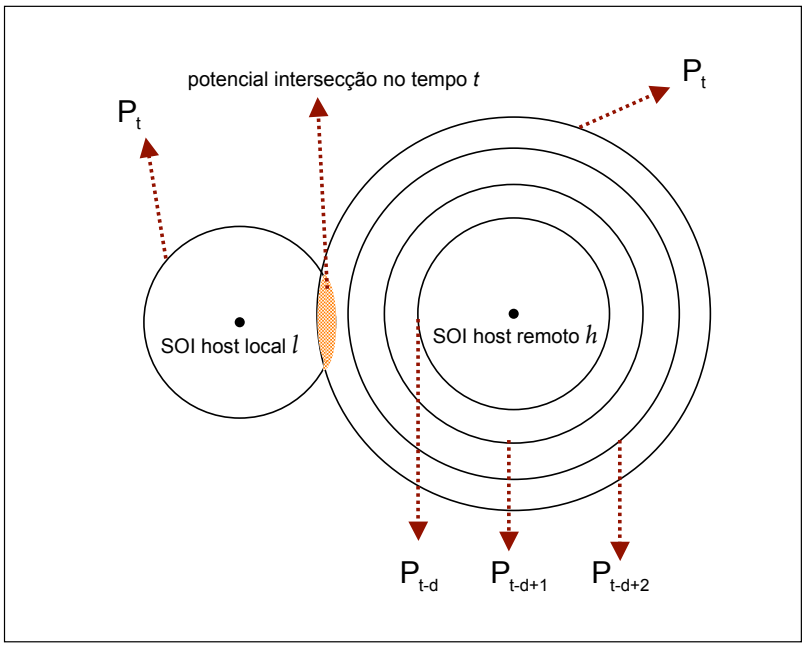


Figura 6.2: dilatação da SOI remota desde o tempo $t - d$ e verificação de potencial intersecção no tempo t.

normalmente o passo lockstep do AS e obtém informações sobre o turno t de Alice. Antes que Bob conclua tal passo, Charlie fornece a Bob informações que ele obtivera, resultando portanto no *lookahead* de Bob sobre o turno t de Alice.

6.2 Lee, Kozlowski, Lenker e Jamin: Pipelined Lockstep

Aplicando *bucket synchronization* (vide seção 3.5) ao lockstep mostrado na Seção 6.1.1, Ho Lee, Eric Kozlowski, Scott Lenker e Sugih Jamin[31] sugeriram em 2002 um protocolo de *lockstep* onde os movimentos entravam em um pipeline de execução, assim como instruções em um processador.

A princípio, jogos com *bucket sync* não eram passíveis de *lockstep*, já que o envio do

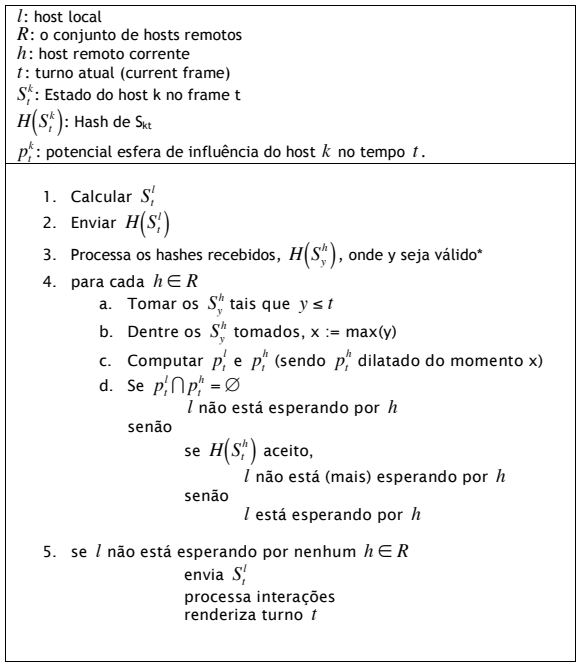


Figura 6.3: descrição do protocolo Asynchronous Synchronization em um host local l, para um turno t.

t-ésimo turno dependia do comprometimento (*commitment*) de todos os jogadores. Sem *lockstep* e com *bucket sync*, a trapaça de lookahead é fácil. Um jogador poderia, por exemplo, perceber que receberia um tiro fatal no *bucket* $t + 2$ e dizer que desviou do tiro no tempo t.

A solução proposta no artigo simplesmente generaliza a confirmação (*commitment*) de um passo para k passos, fazendo com que, ao dizer o passo t, já seja dado também o hash do passo $t + k$, sendo k o tamanho do pipeline. Ou seja, o protocolo permite que haja k *buckets* no modelo de *bucket synchronization*. Enquanto a performance no *lockstep* original (seção 6.1.1) dependia da latência do peer mais lento, a performance do *pipelined lockstep* depende também do tamanho de k.

Em um jogo single-player, a taxa máxima de atualização de quadros (*maximum frame rate*) é determinada pelo design do jogo e pelo computador do cliente (capacidade de

processamento, placa gráfica etc). Chamemos esta taxa de f_{max} , não-influenciável pelos parâmetros discutidos nesta dissertação[1].

Em um jogo multiusuário cliente-servidor, a taxa de quadros em um cliente l é determinada pela latência entre l e o servidor, aqui chamada d_l. Sendo $1/f^l$ a duração de um frame, tem-se:

$$\frac{1}{f^l} = \max\left(2 \cdot d_l, \frac{1}{f_{max}}\right) \tag{6.1}$$

ou, para trabalhar com a taxa de frames em vez de duração de um frame, tem-se:

$$f^l = \min\left(\frac{1}{2 \cdot d_l}, f_{max}\right) \tag{6.2}$$

Se o jogo é distribuído (em peers) e utiliza lockstep, a duração de cada frame no peer l é determinada pela maior latência entre os peers - chamemos d_{max} - como mostra a equação 6.3. Ou seja, o peer mais lento dita o ritmo de jogo para todo o grupo.

$$\frac{1}{d^l} = \max\left(2 \cdot d_{max}, \frac{1}{f_{max}}\right) \tag{6.3}$$

Com o *pipelined lockstep*, o peer mais lento ainda interfere. No entanto, sua interferência é reduzida pela razão k, como mostra a equação 6.4.

$$\frac{1}{f^l} = \max\left(\frac{d_{max}}{k}, \frac{1}{f_{max}}\right) \tag{6.4}$$

A partir da equação 6.4, pode-se também concluir um k ótimo em função de f_{max} e d_{max}. Se k é muito alto, o atraso total para manutenção do pipeline aumenta. Para obter $f_l = f_{max}$, basta fazer $k \geq d_{max} \cdot f_{max}$. O valor ótimo de k é dado pela equação 6.5.

$$k_{optimal} = d_{max} \cdot f_{max} \tag{6.5}$$

Mesmo com o pipeline, o lockstep não é tão eficiente para jogos que exigem alta taxa de frames. Calcular $k_{optimal}$ não é simples, já que d_{max} pode variar bastante ao longo

[1]Para simplificar a discussão, considera-se a influência dos protocolos de segurança no consumo de processamento insuficientes para influenciar f_{max}.

de uma mesma sessão do protocolo. Além disso, o lockstep não permite *dead reckoning* (seção 3.5), metodologia essencial em jogos de muita ação como os FPS.

6.2.1 Trapaça no atraso de confirmação

Se o lockstep original evitava o *lookahead cheat*, a versão com *pipeline* nem tanto. Um peer mal-intencionado que controla seu envio de mensagens pode atrasar a confirmação para implementar um ataque de *lookahead cheat*, como mostra a Figura 6.4: Alice consegue ler seis mensagens de Bob mandando apenas três. Bob pré-confirmou os seis próximos movimentos com resumos; Alice, apenas três. Ou seja, antes de decidir seu quarto movimento, Alice pôde ler os três movimentos de Bob, que seguiu até o sexto sem conhecer os primeiros de Alice.

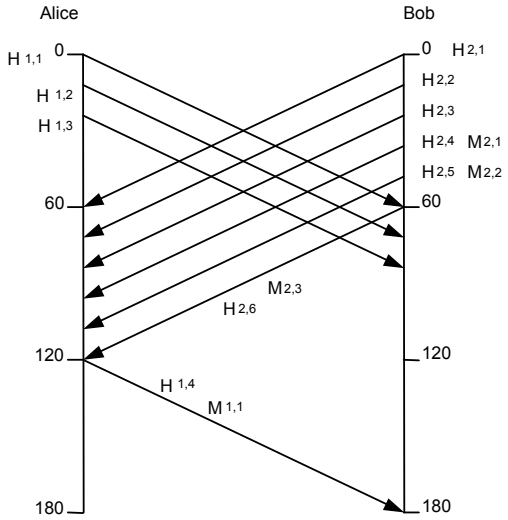


Figura 6.4: atraso no envio do resumo permite lookahead cheat.

Neste caso, k era igual a 3. Alice decidia seus passos assistindo os movimentos de Bob há $2k$ turnos enquanto Bob só conhecia os k passos de Alice.

6.3 Cronin, Filstrup e Jamin

Eric Cronin, Burton Filstrup e Sugih Jamin se depararam com as limitações do *pipelined lockstep* e propuseram novas soluções em um artigo de janeiro de 2003 [13]. A primeira, mais trivial, consiste em tornar dinâmico o tamanho k do *pipeline*. A segunda acrescenta um *send buffer* para facilitar o ajuste de tamanho do pipeline. Estas modificações deram origem ao "pipeline adaptativo" e ao "pipeline deslizante", mostrados a seguir.

6.3.1 O lockstep de pipeline adaptativo (AP)

Ao final da seção 6.2, é quase automático imaginar um protocolo que recalcula k dinamicamente. Cronin, Filstrup e Jamin o descreveram da maneira mais simples, como passo para chegar a algo mais sofisticado. Já que consideramos f_{max} como uma constante do jogo, o desafio é calcular d_{max} a cada turno. Como já dito, d_{max} é a máxima latência entre dois peers. Ou seja, dados p peers em um sistema, consiste em verificar a latência de $\binom{n}{2}$ pares de peers e comparar os resultados. Para facilitar o cálculo, os autores o dividem em duas etapas, como mostram as equações 6.6 e 6.7. Cada peer l calcula seu próprio d^l_{max} e os d^l_{max} são propagados; cada peer, ao receber outros d^l_{max}, calula o d_{max} global. O tamanho de k é calculado e o peer envia mais resumos para aumentar o comprimento do pipeline ou deixa de enviar resumos por um tempo para diminuí-lo.

$$d_{max} = \max\left\{d^l_{max} \mid l \in jogadores\right\} \tag{6.6}$$

$$d^l_{max} = \max\left\{d_{l,m} \mid m \in jogadores\right\} \tag{6.7}$$

O overhead de se calcular d_{max} e de se manter um pipeline de tamanho dinâmico pode provocar atraso na sincronização do jogo, gerando *jitter*, causando desconforto para quem joga.

Os autores apontam, no entanto, uma vantagem do AP sobre o Pipelined Lockstep tradicional: a possibilidade de detecção de trapaças por atraso de confirmação (vide 6.2.1).

O método utilizado para detecção se baseia no tempo de resposta relacionado à latência entre os dois peers. Conseqüentemente, ele acusa como suspeito todos os que trapaceam, mas pode acusar alguém em vão se houver congestionamento em rede e um pico de latência.

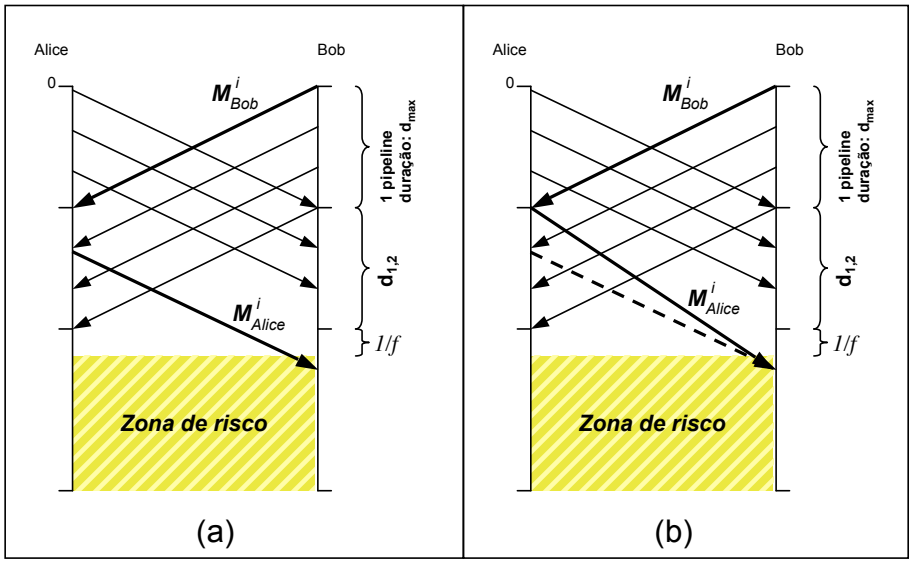


Figura 6.5: (a) Alice realmente trapaceou; (b) Alice não trapaceou, mas sua mensagem demorou mais que o esperado

Como ilustra a Figura 6.5, o método consiste em calcular o tempo que a mensagem enviada exatamente um turno depois demora para chegar. Ou seja, Bob deve cronometrar o intervalo de tempo entre a chegada de M^i_{Bob} e M^i_{Alice} . Se a mensagem chegar antes de $d_{1,2} + \frac{1}{(f^{Alice})}$, não há como ter havido atraso de confirmação. Se chegou depois disso, acontece uma das situações representadas na Figura 6.5, caso em que é preciso verificar (de outras formas) se Alice está ou não trapaceando.

6.3.2 Pipeline deslizante (SP)

Ainda no artigo de 2003 [13], Cronin, Filstrup e Jamin propõem o pipeline deslizante. Trata-se de uma generalização do pipeline adaptativo que, por comportar *dead reckoning* (vide Seção 3.5), torna o jogo mais suave (reduz *jitters*).

Nas outras versões de protocolo de lockstep, se o jogo estivesse prestes a renderizar determinado frame cujo movimento (alheio) não houvesse ainda chegado, o jogo ficava congelado até receber a mensagem. O problema também acontecia no sentido inverso: se o protocolo estivesse com pipeline cheio e não pudesse aceitar envio de uma mensagem, o jogo também parava para não gerar mais movimentos (mensagens). Como acontecia nos dois sentidos, a utilização de previsão de movimentos por dead reckoning não faria sentido - resolveria metade do problema apenas. Trocando em miúdos, era inviável para jogos de ação em tempo real, como FPSs.

Se o *peer* p ainda não pode enviar um resumo de seu movimento, o pipeline deslizante permite que p deposite a jogada em um *buffer* de envio (no fim de uma fila) e o jogo continue. Desta maneira, passa a ser útil a previsão de movimentos (*dead reckoning*).

6.4 Gauthier-Dickey, Zappala e Lo: New-Event Ordering protocol

Em 2004, Chris Gauthier-Dickey, Daniel Zappala e Virginia Lo publicaram um protocolo chamado New-Event Ordering protocol (NEO) [19], que segundo eles seria capaz de também resolver a questão de Lookahead Cheat, como no Lockstep, mas interfere no atraso do jogo independentemente da latência dos jogadores.

Abaixo estuda-se o funcionamento do protocolo NEO e vê-se que, em termos, os resultados fazem sentido, mas que as características anunciadas são questionáveis.

6.4.1 Basic NEO

O artigo de Gauthier-Dickey, Zappala e Lo [19] divide o protocolo em dois estágios: um primeiro, Basic NEO, provê defesa contra timestamp cheats (vide seção 4.1) e supressão de atualizações.

Para conter *lookahead cheat*, NEO faz um esquema equivalente ao Lockstep. No entanto, o autor propõe que, ao invés de enviar um resumo e depois enviar o movimento do turno em texto aberto, seja enviada a atualização do turno t cifrada e depois (no turno $t+1$) se envia a chave de t, junto com a atualização de $t+1$.

Diferentemente do proposto em Lockstep, que espera o comprometimento de todos para então prosseguir o turno, NEO pré-determina um atraso máximo da mensagem de cada usuário. Uma mensagem que chega tarde demais é considerada inválida, a não ser que a maioria dos outros jogadores a tenha recebido em tempo hábil. Cabe ao desenvolvedor do jogo determinar a duração de cada *round*, segundo o atraso máximo que seu *game design* comporta (sem afetar a jogabilidade) e a latência esperada dos jogadores.

No protocolo NEO, cada mensagem do peer A no round t é composta da seguinte forma:

$$M_A^t = E(S_A(U_A^t), U_A^t), K_A^{t-1}, S_A(V_A^{t-1}), V_A^{t-1}, \tag{6.8}$$

sendo $E(x)$ a versão cifrada da string x; U_A^t a atualização de A para o round t; K_A^{t-1} a chave para o update de A no round $(t-1)$, V_A^t o vetor de votos para as mensagens recebidas no round $(r-1)$; e $S_A(x)$ a assinatura de A sobre a string x.

A principal diferença entre este protocolo e o Lockstep é o vetor de votos, V_A^t. Para facilitar a administração da consistência em um sistema peer-to-peer, NEO implementa um mecanismo de votos e de tempo de mensagem que funciona sob as seguintes regras:

- Uma atualização U_A^t é aceita se e somente se ela foi gerada durante o round t, ou seja, antes que A tenha sido capaz de abrir outras mensagens pertinentes ao round t. Tal restrição evita o *lookahead cheat*.

- Um peer A vota "sim" para as mensagens que recebeu a tempo (antes de liberar a chave de seu round) e vota "não" para as mensagens que não recebeu ou que recebeu depois do tempo permitido. Assim, uma mensagem M_A^t é considerada gerada em tempo se a maioria dos *peers* vota "sim" por ela. *Peers* cujos votos não foram recebidos são considerados abstenções.

Uma abordagem possível (opcional) é tentar contactar os peers que se abstiveram caso uma mensagem M_A^t não disponha da maioria de "sim" nem da maioria de "não". Ou seja, se a abstenção foi provocada por perda de dados nas camadas inferiores, esta atitude pode evitar que uma mensagem seja descartada injustamente.

Com o vetor de votos, tem-se de certa forma uma garantia de consistência pela maioria, ou seja, uma trapaça com conluio aqui requer a conivência da maioria dos jogadores envolvidos, o que melhora a segurança do processo frente ao Asynchronous Synchronization (AS) mostrado na seção 6.1.2.

A votação permite que o jogo progrida sem que todos os players sejam ouvidos em todos os rounds, o que é uma vantagem frente ao Lockstep também apresentada pelo AS. Por outro lado, exige a princípio a confirmação da mensagem frente a maioria dos peers, enquanto o AS somente exige comunicação com as redondezas.

6.4.2 NEO com turnos em pipelines

Com a intenção de torná-lo mais adaptável a jogos em tempo real, os autores do NEO propõem que seus turnos sejam seqüenciados em pipeline, a exemplo do protocolo apresentado na seção 6.2.

Na versão básica, o tamanho do round muito pequeno requer baixíssima latência na rede para evitar constantes pausas do protocolo. O tamanho do round muito grande, por outro lado, causa grande demanda por *dead reckoning*, o que permite que o jogo tenha menos pausas, mas pode reduzir a precisão do que é mostrado em relação aos movimentos reais dos jogadores.

A integração do protocolo NEO com um modelo de pipeline consegue reduzir a necessidade de previsão de movimentos, tornando o jogo em tempo real mais suave e preciso. O atraso geral continua dependendo do tamanho do round - afinal, uma atualização enviada só terá sua chave divulgada no começo do round seguinte. O ganho é em suavidade e vem do fato de mais atualizações serem recebidas por segundo.

O tamanho do round pode ser ajustado em função da perda média de atualizações de algum *peer* ou de uma série de *peers*, desde que a maioria indique que não se trata de fraude.

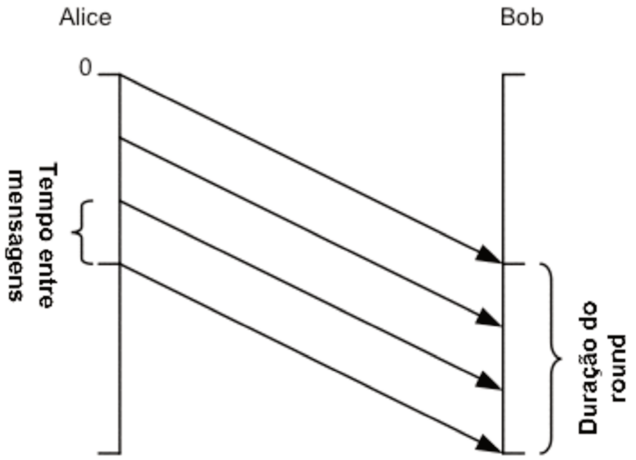


Figura 6.6: relação entre duração do round, tempo de mensagens e tamanho k de um pipeline (no exemplo, k=4)

Enfim, além de impedir o *lookahead cheat* e a supressão de atualização, o NEO também se propõe a dificultar a inconsistência e o conluio por utilizar o mecanismo de vetor de votos e abre a possibilidade de tornar movimentos mais suaves através da criação de um pipeline para recepção de atualizações.

Generalizando a Fórmula 6.8, temos sua versão para pipelines de tamanho k:

$$M_A^t = E(S_A(U_A^t), U_A^t), K_A^{t-k}, S_A(V_A^{t-k}), V_A^{t-k} \tag{6.9}$$

6.5 Integridade e confidencialidade de mensagens

Uma maneira bastante usual de se evitar ataques de alteração de mensagens, como os de Choo (mostrados na seção 5.1), ou ataques de acesso a informação confidencial através da interceptação de mensagens, é simplesmente cifrá-las. O ciframento provê integridade e confidencialidade das mensagens enviadas e pode utilizar uma mesma chave (secreta, compartilhada entre transmissor e receptor) para cifrar e decifrar (ciframento simétrico) ou chaves distintas (ciframento assimétrico), das quais normalmente publica-se a chave para ciframento e mantém-se a chave para deciframento em segredo.

No caso de ciframento assimétrico, a garantia de integridade é algo mais complicado: se o atacante tiver acesso ao texto claro, ele pode reproduzir a mensagem e cifrá-la novamente. Neste caso, a integridade da mensagem e identificação da origem podem ser implementadas por um mecanismo de assinatura.

As primitivas criptográficas mencionadas estão descritas na seção 2.4. Existem inúmeros algoritmos para cada uma delas; alguns, no entanto, são mais comuns ou ainda recomendados pelo National Institute of Standards and Technology do governo norte-americano.

Como criptografia assimétrica é normalmente mais cara (computacionalmente) do que criptografia simétrica, é comum ver, sobretudo em sistemas de execução em tempo real, exemplos onde a criptografia assimétrica é utilizada para a definição de chave simétrica, a qual é depois usada para a comunicação de fato.

Dentre algoritmos bastante utilizados para ciframento simétrico, podem-se citar o Data Encryption Standard (DES) e o Advanced Encryption Standard (AES); para ciframento assimétrico, o RSA e o ciframento por chave pública de El Gamal; para assinatura, o RSA (o mesmo algoritmo pode ser utilizado para prover assinatura ou ciframento) e o Digital Signature Standard (DSS). Sobre criptografia e estes algoritmos, vide referências [34], [40], [41], [46], [43].

Capítulo 7

Plataformas para segurança de jogos online

Este capítulo descreve plataformas de segurança que podem ser aplicadas a jogos online, cada uma delas trazendo soluções a algumas das vulnerabilidades mencionadas no Capítulo 4.

7.1 PunkBuster

Uma ação razoavelmente eficaz para reduzir ataques client-side, seja através da modificação do software-cliente ou de acesso a dados na memória, é o kit de ferramentas PunkBuster, da empresa texana Even Balance[22].

Segundo o próprio fabricante, as principais (atuais) funcionalidades são:

- Varredura (em tempo real) da memória do computador do jogador a procura de *hacks* conhecidos;
- *update* automático de um servidor confiável, que dificulta a instalação de falsas atualizações que poderiam corromper o funcionamento do sistema;

- relatórios freqüentes sobre o estado do sistema são enviados ao servidor por todos os usuários, permitindo ao PunkBuster (PB) remover jogadores desleais de um jogo e informar os outros jogadores sobre o assunto.

- remoção manual, por parte de administradores, de um determinado jogador; a remoção pode ser provisória ou permanente;

- servidor pode ser configurado para auditar jogadores (e clientes PB) por amostragem;

- administradores do PB podem obter imagens de tela (*screenshots*) de jogadores específicos ou configurar os servidores para obter screenshots de tempos em tempos;

- ferramenta opcional de identificação de nomes inapropriados facilita o combate à utilização de nomes ofensivos.

7.2 CAPTCHAS em jogos e hardware CAPTCHAS

CAPTCHA é uma sigla para "Completely Automated Public Turing Test to Tell Computers and Humans Apart", ou seja, um teste que apenas humanos são capazes de responder, como por exemplo o reconhecimento de caracteres distorcidos.

Um dos debates correntes para evitar a inserção de bots em jogos (vide Seção 4.4) é sobre a aplicação de CAPTCHAS. Em [20], Golle e Ducheneaut apontam que a utilização de CAPTCHAS não pertinentes ao jogo têm dois pontos negativos:

- a dispersão da atenção do jogador; e

- o fato de que CAPTCHAS podem ser executados por um humano mesmo que um *bot* interaja com o jogo, ou seja, ele não impede que o jogador tenha ajuda de um *bot*, mas apenas requer a presença de um jogador humano.

Os autores propõem a utilização de CAPTCHAS externos, em hardware, como um joystick especial que tenha que ser utilizado por humanos (e seja autenticado, insubstituível por outro hardware) ou um *token* especial. Este procedimento, no entanto, não ajuda na garantia de "human play", mas apenas na exigência de presença humana.

O problema de inserção de bots em jogos continua sendo um desafio, não solucionado pela aplicação de CAPTCHAS.

7.3 Terra - Uma plataforma confiável sobre máquina virtual

Aplicações que dependem de computação confiável precisam de recursos de segurança providos pelo hardware e pelo sistema operacional, como por exemplo um isolamento garantido entre aplicações distintas e mecanismos eficientes para que uma aplicação se autentique perante seus peers. Esta autenticação evitaria, por exemplo, que um jogador altere o software cliente (como descrito nas Seções 4.4 e 4.9.1 e exemplificado em 5.4) sem que um servidor (ou que um outro peer) perceba.

Em plataformas de uso específico, como alguns consoles de vídeo-game e caixas eletrônicos (ATMs), a aplicação pode confiar nos recursos de segurança oferecidos pelo hardware e pela pilha de software. A plataforma Terra propõe recursos similares através de uma máquina virtual.

Em 2003, um grupo de pesquisadores de Stanford [18] propôs uma plataforma segura com recursos comuns integrados em uma máquina virtual.

O resumo do artigo anuncia que aplicações que necessitam de segurança podem rodar simultaneamente em hardware comum (*commodity hardware*). Embora seja verdade, o próprio artigo cita (e é possível concluir) que, sem proteção de memória ou de status de processamento, continuam a existir pontos de vulnerabilidade na arquitetura.

7.3.1 Arquitetura da MV Terra

O elemento básico do sistema Terra é o Virtual Machine Monitor (VMM). O VMM é responsável pelo isolamento entre cada MV. Ou seja, é ele quem faz toda a comunicação entre a MV e o hardware (real), impedindo a princípio que uma MV consiga acessar informação de outra.

Na arquitetura Terra, a separação é tal que a única maneira de duas aplicações em máquinas virtuais isoladas conversarem entre si, mesmo que funcionando sobre um mesmo hardware, é através da utilização de interfaces de I/O (portas seriais, interfaces de rede, disco compartilhado etc).

O VMM Terra (TVMM) pode oferecer duas abstrações de máquina virtual - uma aberta, normalmente acessível, e outra fechada, cujo conteúdo não pode ser inspecionado ou manipulado. É a versão "closed box" que nos interessa, já que precisamos de um ambiente seguro inclusive contra manipulação da máquina hospedeira. A Figura 7.1 mostra superficialmente a arquitetura do TVMM.

Para administrar os recursos que o TVMM provê às máquinas virtuais, a arquitetura Terra prevê a existência de uma MV de administração. O TVMM provê a esta máquina interface para que ela configure (i.e. permita ao administrador da plataforma configurar) os recursos disponíveis para cada MV sobre o TVMM, como memória, uso de CPU, quantas MVs podem rodar etc. Esta permissão abre portas para que o administrador da plataforma ataque MVs através da não-concessão de recursos. No entanto, este poder não representa um ponto de falha, uma vez que o administrador (que vê o hardware por fora) já poderia fazê-lo, por exemplo, desconectando um cabo.

Para que se tenha um "closed box", o TVMM provê três recursos essenciais de segurança:

- **Blindagem:** Nem mesmo o projetista ou o administrador da plataforma pode violar a privacidade de uma caixa fechada.
- **Atestação:** É possível para uma aplicação que roda em uma caixa fechada identificar-

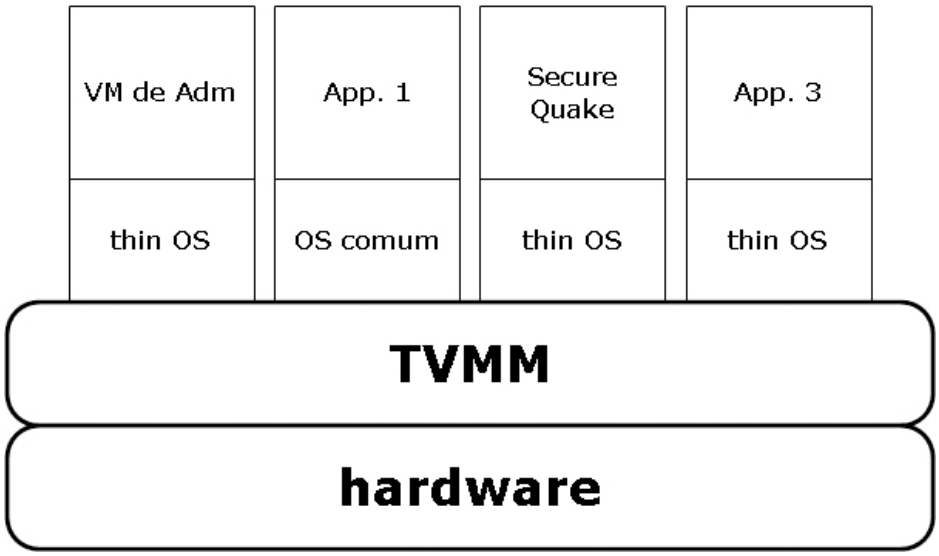


Figura 7.1: Arquitetura de um TVMM. Como múltiplas MVs podem ser instanciadas sobre um mesmo TVMM, os recursos utilizados por cada MV (memória, conexão, armazenamento em disco, processamento) são configurados através de uma "MV de administração". O TVMM fornece à MV de administração uma interface que permite estas configurações.

se perante outra parte (externa à caixa). Mais informação sobre a capacidade de identificação pode ser encontrada no tópico 7.3.2.

- **Caminho confiável:** Em um TVMM, um caminho confiável entre um usuário e a aplicação a permite definir (ou saber) com qual MV está se comunicando e permite à MV saber que realmente se comunica com um ser humano. Provê-se (ao longo do caminho) integridade e privacidade das mensagens.

7.3.2 Atestação na TVMM

O TVMM implementa uma versão de atestação remota (vide 2.5.2). No entanto, como não há ação de hardware, o mecanismo ainda é passível de ataques, já que a própria TVMM precisa se provar autêntica. Ou seja, a TVMM depende de um certificado (que não tenha sido revogado) assinado pelo fabricante confiável ou por uma autoridade certificadora. No

nível acima da pilha, o TVMM faz um resumo dos estados persistentes que identificam a MV, o *bootloader* e o SO (caso este exista como camada independente). Com este hash enviado pelo TVMM confiável, o peer remoto pode validar a integridade da pilha de software, desde a MV até a aplicação.

Embora a atestação esteja completa, ela não serve como defesa para um ataque "man-in-the-middle" posterior, ou seja, é necessário que haja canais seguros de comunicação entre as MVs que se comunicam.

7.3.3 Quake na plataforma Terra (Trusted Quake)

O jogo Quake é um bom exemplo de jogo digital sobre arquitetura cliente-servidor com histórico de vulnerabilidades exploradas. Boa parte delas está associada à modificação do código do cliente ou do servidor, já que o jogo se transformou em *open source* após o lançamento da versão subseqüente.

Os autores de Terra (Boneh, Garfinkel, Chow, Pfaff, Rosenblum) implementaram uma versão de Quake-II em caixa fechada. O protocolo de atestação é usado para autenticar o próprio Quake-II perante os outros players antes mesmo da troca de chaves (de 160 bits para SHA-1 HMAC[1] e chaves de 56 bits para DES[2]. Todo o tráfego de Quake em rede utiliza então HMAC e DES para integridade e confidencialidade, respectivamente.

O protótipo utiliza uma MV rodando um kernel Linux 2.4.20 em conjunto com uma implementação (mínima) de Debian GNU/Linux 3.0. A MV inicializa carregando diretamente Quake, não disponibilizando *shell* ou interface de configuração para usuários.

A segurança implementada pelo Trusted Quake melhora nos seguintes aspectos em relação ao jogo original:

- **confidencialidade e integridade das mensagens:** Através do mecanismo de confidencialidade fornecido pela MV, o Trusted Quake pode conversar com seus pe-

[1]sobre HMAC, vide subseção 2.5.1 neste documento). Sobre o algoritmo SHA-1, veja definição em [42]

[2]Informações sobre DES e sobre ciframento simétrico em geral podem ser visto em [34]. A definição de DES está em [40]

ers de maneira segura, evitando por exemplo ataques por observação do tráfego de rede para obtenção de informação inapropriada, como apontado nas seções 4.7 e 4.9 deste documento, úteis para falsificação de identidade ou para descobrir, por exemplo, a localização de jogadores inimigos. Além disso, a MV também promove, através do mesmo conjunto de primitivas criptográficas (ciframento e autenticação), a integridade das mensagens e a garantia de origem, evitando alteração da informação trafegada, ataque por exemplo utilizado em *bots* de mira automática, como descrito na seção 4.4.

- **integridade do cliente:** Editar o cliente do jogo pode permitir ataques que envolvem a adulteração do software-cliente, como os mostrados na subseção 4.9.1. A atestação do SW por parte do TVMM evita estes ataques.

- **integridade do servidor:** Quake é um jogo essencialmente cliente-servidor. Toda a coordenação das ações do jogo é feita em "server-side", o que permite que um servidor modificado forneça enormes vantagens a um jogador sobre outros.

- **isolamento:** Duas conseqüências imediatas do isolamento de Quake: o (restante do) sistema fica protegido em caso de ações indesejadas por parte do jogo e o jogo fica protegido de falhas do sistema ou de ataques que, por exemplo, alterem as informações do jogo na região de memória em que ele escreve.

Aspectos de segurança não tratados pelo Trusted Quake:

- **correção de bugs e funcionalidades indesejadas:** Alguns comandos de Quake podem ser usados para trapaças. Um exemplo é o número de polígonos renderizados na tela. Ficando parado, o aumento do número de polígonos pode indicar o surgimento de um inimigo ao longe.

- **prevenção contra ataques de negação de serviço:** Um ataque que impeça um peer (servidor ou cliente) de se comunicar não é tratado pela MV. Esse problema não se resolve com criptografia, mas com redundância.

- **bloqueio de trapaças por conluio em canais alternativos:** Mesmo em um sistema seguro, dois jogadores que jogam Quake em máquinas vizinhas terão a vantagem de acumular a visão de duas regiões do jogo, facilitando o planejamento de ataques.

7.3.4 Dependência de hardware

Embora a arquitetura Terra proveja recursos para tornar sistemas mais seguros tanto na operação (local) quanto na comunicação, ainda restam pontos vulneráveis que não se podem eliminar por software, isto é, abaixo da máquina virtual. Mesmo que haja proteção de áreas de memória por mecanismos criptográficos, em algum ponto terá de haver informação em texto claro. Um exemplo é a dificuldade de se armazenar a chave privada do TVMM de forma segura. Neste caso, depende-se de hardware confiável que implemente soluções, como o descrito na seção 7.4.

Recursos necessários para proteção em hardware

Para corrigir os pontos de vulnerabilidade, a arquitetura Terra dependeria dos seguintes recursos providos por hardware:

- **atestação do hardware:** O hardware precisa ser capaz de (pelo menos) atestar-se perante a primeira camada da pilha de software.
- **armazenamento protegido:** Cifra os dados com a chave privada do co-processador responsável pela atestação (TPM, vide seção 7.4). Um *hash* do SO confiável (ou do TVMM) também é incluído na informação cifrada. Através do armazenamento do hash, o co-processador consegue permitir que apenas o SO associado acesse seus dados. Esta funcionalidade seria usada, por exemplo, para que o TVMM armazene de forma segura sua chave privada.
- **Suporte de hardware para I/O seguro:** Aplicações precisam se comunicar com o mundo externo - no caso dos jogos, saída de vídeo, interface de rede, mouse

e teclado são essenciais. É preciso, portanto, que os drivers destes dispositivos sejam confiáveis. Uma solução poderia ser implementar cada driver internamente ao TVMM. Outra seria fazer com que o hardware (em cada dispositivo) fosse capaz de estabelecer comunicação segura com o TVMM ou com cada MV. No caso particular da interface de rede, esta proteção não é necessária porque Terra implementa um canal seguro de comunicação entre aplicações remotas.

- **isolamento de dispositivos em nível de hardware:** Proteção de memória e controladores de DMA e limitação de acesso ao barramento PCI são características interessantes que evitariam ataques ao hardware para obtenção de informação em pontos onde ela já (ou ainda) está em texto claro.

7.4 Trusted Computing e o Trusted Platform Module

O Trusted Platform Module (TPM) é uma especificação de microcontrolador[21] ditada por um consórcio conhecido como Trusted Computing Group (TCG). A implementação deste microcontrolador pode ser executada por diferentes fabricantes homologados. Dentre os atuais produtores, estão empresas como Atmel, Broadcom, Infineon, ST Microelectronics e Winbond.

Embora não seja propriamente uma plataforma de segurança para jogos, TPM é importante por prover recursos de segurança (em nível de hardware) desejáveis para tornar, por exemplo, a Terra VM (7.3) mais segura, como dito em 7.3.4.

O TPM é normalmente embutido na placa-mãe e utiliza características do hardware e do software básico para autenticar a plataforma.

7.4.1 O que provê um Trusted Platform Module

A função de um TPM é prover primitivas necessárias à segurança da informação (em comunicação e armazenamento), geração de chaves, geração de números aleatórios (GNA)

e locais blindados para armazenamento de poucas informações confidenciais, como chaves privadas, necessárias à segurança de outros dados.

Entre suas funcionalidades estão atestação e armazenamento protegido.

7.4.2 Arquitetura de um TPM

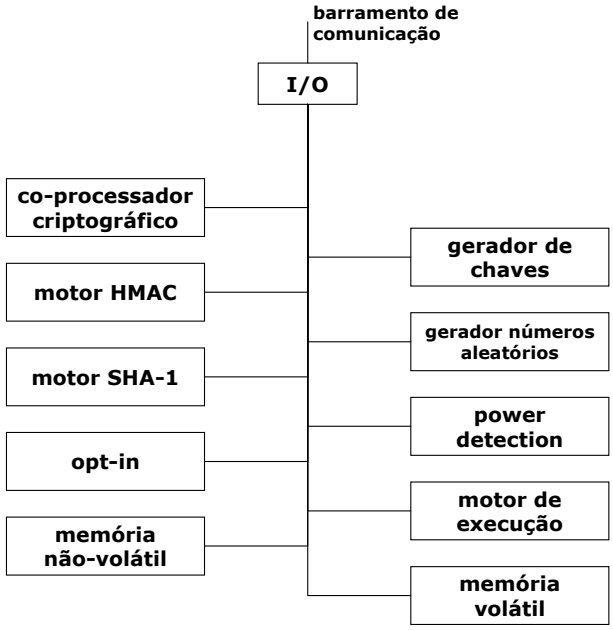


Figura 7.2: Arquitetura de um trusted platform module. Fonte: Trusted Computing Group[21]

Os componentes de um TPM têm sua comunicação coordenada por um componente de entrada e saída (I/O, na Figura 7.2). É ele o responsável por rotear a informação para o componente apropriado.

A Figura 7.2 lista os componentes básicos de um TPM, embora uma implementação específica possa ter uma arquitetura mais complexa que esta. Além do gerenciador de

entrada e saída, os principais componentes de um TPM são:

Co-processador criptográfico

Este componente implementa operações criptográficas convencionais, dentre as quais:

- geração de chaves assimétricas (RSA)
- ciframento e deciframento assimétricos (RSA)
- hash (SHA-1)
- geração de números aleatórios
- assinatura digital (RSA)

Segundo as normas do TCG, há algumas exigências para os parâmetros das funções executadas pelo co-processador criptográfico. Por exemplo, embora o TPM possa suportar outros tamanhos de chave para RSA, ele deve fornecer no mínimo a implementação de RSA com chaves de 512, 768, 1024 e 2048 bits.

Além das operações convencionais citadas, cujos parâmetros são normatizados, o co-processador criptográfico tem ainda funcionalidades que podem ser especificadas na implementação do TPM.

Sobre criptografia assimétrica, ciframento, assinatura e geração de chaves em RSA vide referências [34] e [46]. Sobre SHA-1, vide referência [42].

Gerador de chaves

O gerador cria pares de chaves RSA e chaves simétricas. As chaves privadas são mantidas em local blindado.

Motor de HMAC

O motor de HMAC fornece duas informações aos TPM: comprovação de integridade da mensagem e comprovação de origem da mensagem.

Gerador de números aleatórios

O GNA é uma máquina de estados que injeta informação imprevisível em uma função de resumo, resultando em um número aleatório. A máquina de estado pode ser incializada (com um estado inicial imprevisível) no momento de sua fabricação. A qualquer momento, a máquina pode aceitar uma entrada imprevisível (de software ou hardware) para "salgar" o número aleatório. É importante que o GNA não revele seu estado nem mesmo ao próprio fabricante que a inicializou - daí a necessidade de uma boa função de resumo.

As entradas do gerador são dadas por um sistema (software ou hardware) coletor de entropia. Normalmente se coleta entropia de fontes externas não-determinísticas, como uma seqüência de digitação, movimentos de mouse, variação de temperatura etc. Ou por um conjunto de fontes deste tipo.

Motor SHA-1

A TCG não especifica desempenho para o motor SHA-1, já que o TPM não precisa servir como acelerador criptográfico. As exigências são que o hash implementado siga as definições de algoritmo SHA-1 definida na FIPS-180-1 [42].

Motor de execução

Comanda a execução de comandos provenientes da porta de entrada e saída. Além disso, verifica se as operações estão adequadamente segregadas e se locais protegidos são de fato seguros.

7.4.3 Armazenamento protegido

Armazenamento protegido protege informação confidencial ao permitir que seja cifrada com uma chave derivada do conjunto "software+hardware" utilizado. Como resultado, apenas aquela combinação de software e hardware pode ler o que foi armazenado. Além de servir para DRM, esta funcionalidade também pode impedir ataques de *memory hooking*

como os citados na subseção 4.9.2.

7.4.4 Atestação por hardware

Uma das funcionalidades providas pelo TPM é a atestação remota 2.5.2. Uma aplicação direta da atestação poderia ser auxiliar no combate ao ataque de alteração do cliente remoto, mostrado no tópico 4.9.1.

Capítulo 8

Segurança em arquiteturas específicas

Este capítulo trata de arquiteturas de jogos online. Inicialmente compara a questão de segurança em arquiteturas cliente-servidor e *peer-to-peer*. Em seguida aplica uma proposta de arquitetura híbrida de ganho de escalabilidade à solução NEO mostrada na Seção 6.4.2.

8.1 Segurança em arquitetura cliente-servidor

Uma arquitetura cliente-servidor pode reduzir os esforços para garantia de segurança, já que decisões cruciais no jogo são tomadas pelo servidor. Se for possível garantir que o software servidor está rodando em ambiente seguro, ataques a transações comerciais dentro de jogos, por exemplo, tornam-se mais difíceis.

Também é mais fácil um servidor estar atualizado e conhecer praticamente todas as versões do software-cliente do que, em um sistema distribuído, um peer de versão mais antiga conhecer outro peer mais recente.

Uma desvantagem de um sistema cliente-servidor é que, se não há redundância, todos os jogadores sofrem quando um ataque de negação de serviço sobre o servidor acontece. Outra desvantagem nítida, não relacionada com a questão de segurança, é a escalabilidade:

manter servidores oniscientes é muito caro. Em jogos com muitos jogadores, torna-se mais caro ainda. Se as decisões tomadas no servidor demandam muitos recursos de máquina ou banda, a questão fica ainda pior.

Sobre a segurança de jogos sobre arquitetura cliente-servidor, um bom exemplo é Trusted Quake citado no tópico 7.3.3 deste documento. Em Trusted Quake, tanto o servidor quanto o cliente tem sua autenticidade verificada pelo mecanismo de atestação da TVMM.

8.2 Segurança em jogos distribuídos

À primeira vista, um sistema de jogos distribuído pode não parecer uma solução para o problema de escalabilidade na arquitetura cliente-servidor. Vejamos em um raciocínio simples: se há n clientes em um servidor, cada cliente recebe e envia $O(1)$ mensagens por intervalo de tempo para atualização de seu estado e, no universo todo, a taxa de mensagens é $O(n)$. Em um modelo de *peers*, cada *peer p* envia $O(1)$ mensagens para cada um dos outros $(n-1)$ *peers*, ou seja, são n peers enviando $O(n)$ mensagens cada, portanto $O(n^2)$ mensagens. Isso faz parecer que o volume de dados em um sistema de peers é $O(n^2)$ enquanto o volume de dados em sistemas cliente-servidor é $O(n)$. Boa parte das vezes, um engano.

Como mostrado em [12], em jogos de tempo real onde todos executam ações e alimentam o servidor com atualizações, o volume de informação nas mensagens enviadas pelo servidor a cada um dos clientes cresce proporcionalmente ao número de jogadores. Como resultado, o volume total de informação que trafega graças ao jogo é $O(n^2)$, mas com uma diferença: neste caso, a demanda por largura de banda se concentra sobre o servidor. Distribuindo-se o jogo, a necessidade de banda se distribui sobre os peers: cada um dos n clientes envia e recebe $O(n)$ mensagens sobre a atualização de um único jogador, ou seja, mensagens cujo tamanho é $O(1)$.

8.2.1 Arquitetura híbrida com peers que se falam e servidores

Como proposta para um mecanismo escalável para implementação de jogos distribuídos, um grupo composto por pesquisadores da Unicamp e da UFMS [2] propôs um sistema híbrido, em que existe a distribuição por peers, mas também existem servidores com funções específicas, com menor necessidade de robustez que os servidores centrais e oniscientes dos modelos puramente cliente-servidor.

As principais características da proposta Unicamp-UFMS são:

- a divisão do universo do jogo em mini-mundos (small worlds): um peer cujo avatar se encontra em um pequeno mundo se inscreve para receber atualizações daquele pequeno mundo e de pequenos-mundos vizinhos; um modelo análogo ao de células e telefones móveis;
- o conceito de resolução de taxa de mensagens: entidades ao longe demandam menor taxa de atualização, consequentemente reduzindo o volume de dados que trafega entre dois peers cujos avatares estejam longe um do outro.

Para coordenar a inscrição nos mundos, atribui-se a servidores ou a peers confiáveis o papel de "líder do mini-mundo". A proposta prevê a distribuição do papel de líderes aos peers jogadores, mas essa questão geraria desafios de segurança e robustez, além do problema básico de identificação de um líder caso não haja (no mínimo) um servidor de diretórios que o indique. Prefiro portanto tratar a liderança como papel atribuído a um simples servidor confiável e, para maior conforto dos que se preocupam com robustez, replicado.

Sobre a resolução de taxa de mensagens, há uma dúvida pertinente ao *game design*: qual a resolução (taxa de atualização) necessária entre dois determinados peers, p_{m1} e p_{m2} inscritos em um mesmo mini-mundo m? Uma resposta ideal para esta questão deve ser um parâmetro obtido de ajustes durante a fase de balanceamento do jogo - um posicionamento diante do *trade-off* "mínimo jitter vs. máxima escalabilidade".

No balanceamento, possivelmente este parâmetro será ajustado conforme o tempo que o avatar de p_{m1} leva para interceptar ou interagir diretamente com o avatar de p_{m2}.

8.2.2 Aplicando-se NEO à arquitetura híbrida

Ambas as características citadas como importantes da arquitetura Unicamp-UFMS podem ser aproveitadas como parâmetros de ajuste em uma nova versão do protocolo NEO com pipelines.

Pipeline para sincronização da arquitetura híbrida

Somando-se o mecanismo proposto por [2] ao modelo de *bucket synchronization* apresentado na seção 3.5, pode-se determinar, por exemplo, a *freqüência de buckets* e a qualidade do *dead reckoning* (DR) executado em função da resolução necessária e da pertinência a mini-mundos.

Para definir graus de pertinência a mini-mundos, é preciso ter múltiplas categorias de inscrição para estes espaços. Tomando mais uma vez por analogia o mecanismo de alerta em células de telefonia, podemos ter duas classes de assinantes em um mini-mundo m: um peer $p_m \in M$, cujo avatar está no mundo M, e um peer $q_n \in N$, cujo avatar está em um mundo N vizinho a M.

Voltando ao artigo [2], notamos que, embora um peer q_n não se comunique diretamente com um peer p_m, q_n é inscrito no mundo M porque pode a qualquer momento cruzar a fronteira e passar a ser um peer pertencente a M. Essa passagem deve ser suave para o usuário, o que requer que os vértices conheçam o estado dos mundos vizinhos.

Para sincronização da arquitetura híbrida, no entanto, é essencial que a *freqüência de buckets* nos peers de um mesmo mini-mundo que interagem entre si seja suficientemente alta para evitar *jitter*. A freqüência de *buckets* entre mundos vizinhos, no entanto, pode ser bem mais baixa, como exemplifica a Figura 8.1. Além disso, o fato de atrasos na compreensão do estado de N (o mundo vizinho) ser menos problemático que atrasos na compreensão de M permite a p_m que lide com um pipeline de maior comprimento para

os peers pertencentes a N, reduzindo ainda mais o risco de atraso do andamento do jogo.

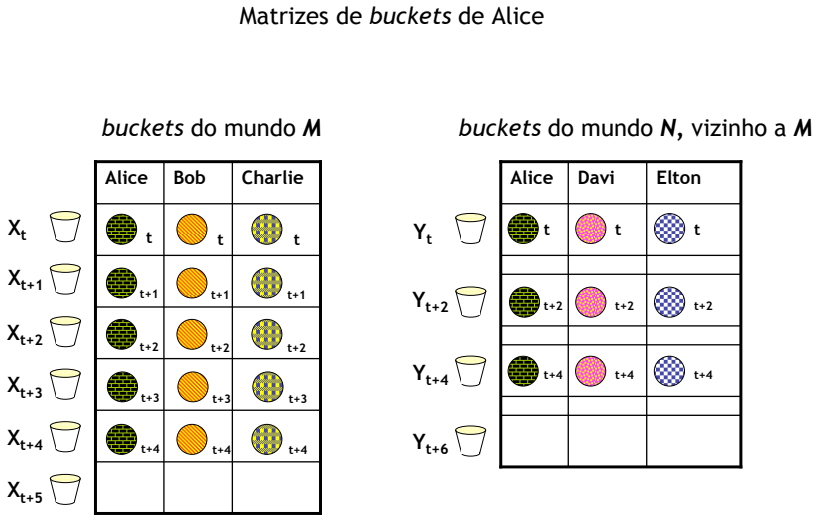


Figura 8.1: Neste exemplo, a freqüência de *buckets* entre *peers* do mesmo mundo é f_X e entre peers de mundos vizinhos é $f_Y = f_X/2$. Davi e Elton, no entanto, que estão ambos no mundo N (próximos um do outro), trocarão mensagens entre si na freqüência f_X.

Pode-se também aplicar *dead reckoning* (DR) ao modelo de *message rate resolution*, suprindo-se com estimativas as informações não-enviadas por peers distantes. Caso o número de peers seja muito grande e o trabalho de DR custe muito caro, é possível estabelecer classes de DR: peers muito distantes têm sua última posição mantida, por exemplo, e peers um pouco mais próximos têm DR feito por um algoritmo mais elaborado, como os mencionados na seção 3.5. Idealmente, a definição de distância deve ser um parâmetro obtido durante a fase de balanceamento do jogo, relacionada com o *gameplay* e com a velocidade de locomoção dos avatares e objetos dentro do universo virtual.

NEO somado a message rate resolution: segurança ao ganhar eficiência

Como mostrado na Seção 6.4.2, o protocolo NEO pode absorver o conceito de *bucket synchronization* (BS) e ser utilizado para dificultar ataques de Lookahead (seção 4.1) com reduzida possibilidade de jitters.

A vantagem de se aplicar NEO sobre o mecanismo de mini-mundos com resolução de taxa de mensagens é a redução do universo a ser verificado. Aproveita-se portanto o ganho de performance trazido pelo gerenciamento de interesse não só na taxa de mensagens, mas no volume de assinaturas a verificar e de votos a apurar em cada turno.

De maneira análoga ao proposto para BS aplicado ao gerenciamento de interesses, faz-se NEO com pipelines.

Para que a verificação de assinaturas e votos funcione com maior espaçamento de tempo para *peers* de mini-mundos vizinhos, seria preciso enviar os votos e a chave do turno $t - k$ em um momento diferente de t, o que exigiria a criação de outro mecanismo de armazenamento de jogadas e votos.

Se a freqüência de envio de mensagens a *peers* de mini-mundos vizinhos, no entanto, for um divisor inteiro de k, pode-se aproveitar estrutura de armazenamento de jogadas e geração de mensagens. Se o volume de jogadores em cada mini-mundo for suficiente, confiar nos votos do mini-mundo vizinho é uma alternativa para aproveitamento integral da estrutura NEO.

Para variação da resolução de taxa de mensagens entre peers de um mesmo mundo, também se deve atentar à necessidade de reestruturação. Para que se aproveite o mesmo mecanismo de votos, k deve ser múltiplo comum de todas as freqüências de envio de mensagens. Assim, sempre que existir a mensagem $t - k$, existirá também a mensagem t que trará a chave e o vetor de votos referentes a $t - k$, validando a Fórmula 6.9 também para esta situação, como exemplifica a Figura 8.2.

Se *peers* p_M e q_M, de um mesmo mundo M, mudam a resolução de taxa de mensagens muito freqüentemente, pode-se implementar o mecanismo NEO de maneira que ele não observe tal variação de freqüência, e que $NEO(p_M)$ (a instância em p_M da função que

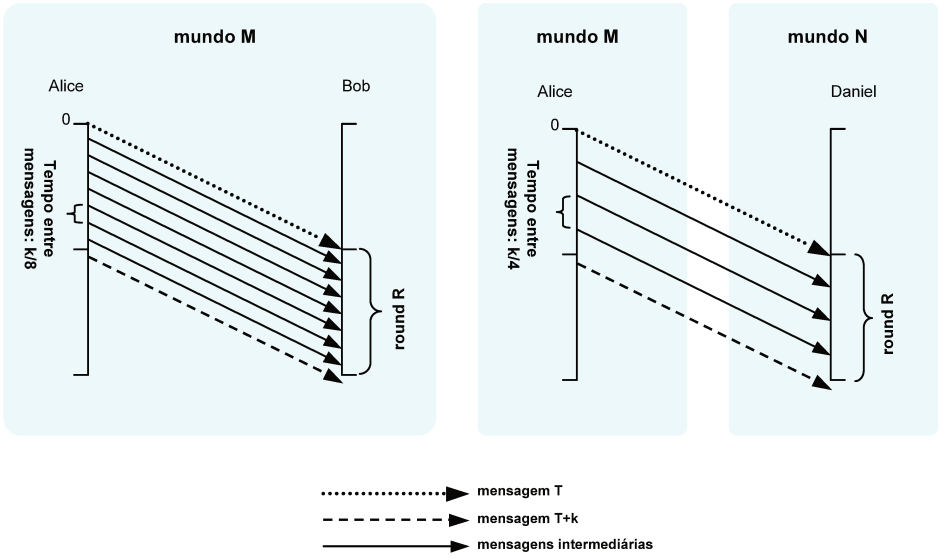


Figura 8.2: Na implementação de NEO sobre o mecanismo de resolução de mensagens, o número de mensagens entre Alice (do mundo M) e Daniel (do mundo N) é divisor de k, o número de mensagens interno ao mundo M.

implementa NEO) considere como abstenções de q_M a ausência de mensagens provocada pela redução de freqüência.

Por fim, este mecanismo também incorpora a não-necessidade de sincronizar-se com peers de mundos não-vizinhos, implementando uma versão com mais regras do que Baughman e Levine chamaram de *Asynchronous Synchronization* 6.1.2.

Capítulo 9

Conclusão

Ao analisar aspectos de segurança em sistemas de informação, sempre se depara com o *trade-off* segurança *versus* eficiência. Em jogos, esta relação é ainda mais complexa, já que em jogos de tempo real é essencial que a execução seja "suave", isto é, com atualizações freqüentes (de maneira a reduzir *jitters*) e com intervalo curto entre a ação tomada por um jogador A e percepção desta ação por B, de maneira a não comprometer a jogabilidade. Além disso, jogos já demandam muita largura de banda e grande poder de processamento, o que dificulta o equilíbrio da balança.

Jogos online de mundo persistente estão vulneráveis à falsificação de ativos virtuais; jogos de guerra em rede são passíveis de ataques de acesso a informações privilegiadas ou podem sofrer modificação para que um dos jogadores tenha sempre maior precisão nos tiros, por exemplo. Falhas não faltam. Mecanismos que se baseiam na distribuição do processamento para redução de carga sobre o servidor tornam os sistemas ainda mais vulneráveis.

Os estudos durante este mestrado, no entanto, me deram a chance de analisar vários caminhos para agregar mecanismos de segurança a soluções de ganho de desempenho, como a aplicação de um protocolo de ordenação de eventos (NEO, new event ordering protocol) a uma arquitetura híbrida de *peers* e servidores, abordagem mencionada na Seção 8.2. São possibilidades de manutenção da segurança ao ganhar em eficiência e

jogabilidade.

Encerro esta etapa de estudos com a sugestão para que sejam implementadas as propostas da Seção 8.2 (NEO aplicado a arquitetura híbrida), com tentativas de ganho de desempenho pela utilização de Trusted Platform Module (TPM, vide Seção 7.4) como processador específico, muito embora ele não seja especificado como um acelerador criptográfico. Também sugiro utilizar TPM para auxiliar um mecanismo fechado, como o Trusted Quake (versão segura do jogo Quake mencionado na Seção 7.3.3), a implementar os mecanismos criptográficos necessários para sua blindagem.

Outro ganho seria a utilização do TPM para dificultar a falsificação de ativos virtuais: o jogador se autentica (com ajuda do módulo) antes de efetuar transações. Embora designers dos jogos online ataquem esta alternativa como redução da privacidade, dizendo que ela pode se tornar um meio para investigar hábitos dos jogadores, é preciso reconhecer que, assim como no mundo real, perda de privacidade é muitas vezes um preço a se pagar pela redução de crimes, e que de certa forma os *game masters* (coordenadores do ambiente dos jogos multiusuários) já sabem muito sobre os hábitos de quem joga, sobretudo em mundos virtuais persistentes.

Por fim, o que se conclui é que a identificação do ponto ótimo na função de parâmetros segurança, eficiência e privacidade é uma decisão do game designer. Com este estudo, porém, espero ter aberto discussões para a melhoria desta relação, permitindo escolhas mais compatíveis com o tamanho atual do mercado de jogos em rede. Se a diversão está relacionada com muito dinheiro, ela se torna bastante séria.

Apêndice A

John Carmack escreve sobre a falha de segurança em Quake

O texto a seguir é atribuído ao ".plan" de John Carmack, fundador da *id Software* e criador de Quake.

```
http://www.bluesnews.com/cgi-bin/finger.pl?id=1&time=19991226003141

Name: John Carmack
Email: johnc@idsoftware.com
Description: Programmer
Project: Quake 3 Arena
------------------------------------------------------------------------
12/25/99
--------

There are a number of people upset about the Quake 1 source
code release, because it is allowing cheating in existing games.

There will be a sorting out period as people figure out what directions
the Quake1 world is going to go in with the new capabilities, but it
```

will still be possible to have cheat free games after a few things get worked out.

Here's what needs to be done:

You have to assume the server is trusted. Because of the wau quake mods work, It has always been possible to have server side cheats along the lines of "if name == mine, scale damage by 75%". You have to trust the server operator.

So, the problem then becomes a matter of making sure the clients are all playing with an acceptable version before allowing them to connect to the server. You obviously can't just ask the client, because if it is hacked it can just tell you what you want to hear. Because of the nature of the GPL, you can't just have a hidden part of the code to do verification.

What needs to be done is to create two closed source programs that act as executable loaders / verifiers and communication proxies for the client and server. These would need to be produced for each platform the game runs on. Some modifications will need to be done to the open source code to allow it to (optionally) communicate with these proxies.

These programs would perform a robust binary digest of the programs they are loading and communicate with their peer in a complex encrypted protocol before allowing the game connection to start. It may be possible to bypass the proxy for normal packets to avoid adding any scheduling or latency issues, but it will need to be involved to some degree to prevent a cheater from hijacking the connection once it is created.

The server operator would determine which versions of the game are to be allowed to connect to their server if they wish to enforce proxy

protection. The part of the community that wants to be competetive will have to agree to some reasonable schedule of adoption of new versions.

Nothing in online games is cheat-proof (there is allways the device driver level of things to hack on), but that would actually be more secure than the game as it originally shipped, because hex edited patches wouldn't work any more. Someone could still in theory hack the closed source programs, but that is the same situation everyone was in with the original game.

People can start working on this immediately. There is some prior art in various unix games that would probably be helpfull. It would also be a good idea to find some crypto hackers to review proposed proxy communication strategies.

Referências

[1] Abril.com. Um ano de ragnarok no brasil. `http://www.abril.com.br/noticia/portal/no_90964.shtml`, 2005.

[2] Ricardo Anido, Edson Caceres, Sérgio Jábali, Rafael Castro, and Leonardo Fernandes. Interest management systems for reducing network requirements in peer-to-peer based multiplayer games. In *IV Workshop Brasileiro de Jogos e Entretenimento Digital (workshop proceedings)*, pages 30–36. Sociedade Brasileira de Computação, 2005.

[3] Electronic Arts. Battlefield 2. `http://www.ea.com/official/battlefield/battlefield2/us/home.jsp`, link acessado em novembro de 2006.

[4] Nathaniel E. Baughman and Brian Neil Levine. Cheat-proof playout for centralized and distributed online games. In *INFOCOM*, pages 104–113, 2001.

[5] Wentong Cai, Francis B. S. Lee, and L. Chen. An auto-adaptive dead reckoning algorithm for distributed interactive simulation. In *PADS '99: Proceedings of the thirteenth workshop on Parallel and distributed simulation*, pages 82–89, Washington, DC, USA, 1999. IEEE Computer Society.

[6] Nick Caldwell. Defeating lag with cubic splines. `http://www.gamedev.net/reference/articles/article914.asp`, 2000.

[7] Christopher Choo. Understanding cheating in counterstrike. `http://www.fragnetics.com/articles/cscheat/`, November 2001.

[8] Price Waterhouse Coopers. Entertainment and media industry in solid growth phase. `http://www.pwc.com/extweb/ncpressrelease.nsf/docid/283F75E5D932C00385257194004DDD0A`, June 2006.

[9] Microsoft Corporation. Plataforma xna. `http://msdn2.microsoft.com/en-us/xna/default.aspx`.

[10] Microsoft Corporation. Age of empires. `http://www.microsoft.com/games/empires/`, link acessado em novembro de 2006.

[11] Valve Corporation. Steam player number statistics. `http://steampowered.com/status/game_stats.html`.

[12] E. Cronin, B. Filstrup, and A. Kurc. A distributed multiplayer game server system, 2001.

[13] Eric Cronin, Burton Filstrup, and Sugih Jamin. Cheat-proofing dead reckoned multiplayer games. In *Proceedings of 2nd International Conference on Application and Development of Computer Games*, January 2003.

[14] Comitê Gestor da ICP Brasil. Infra-estrutura de chaves públicas brasileira. `http://www.icpbrasil.gov.br`, link acessado em fevereiro de 2006.

[15] Associação Brasileira das Desenvolvedoras de Jogos Eletrônicos. Plano diretor da promoção da indústria de desenvolvimento de jogos eletrônicos no brasil. `http://www.abragames.org/docs/pd_diretrizesbasicas.pdf`, Dezembro 2004.

[16] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the internet. *IEEE Networks magazine*, 13(4), July 1999.

[17] GameSpot.com. Second life faces threat to its virtual economy. `http://www.gamespot.com/news/6161766.html?tag=gs.email`, November 2006.

[18] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 193–206, 2003.

[19] C. GauthierDickey, D. Zappala, V. Lo, and J. Marr. Low latency and cheat-proof event ordering for peer-to-peer games, 2004.

[20] Philippe Golle and Nicolas Ducheneaut. Keeping bots out of online games. In *Proc. of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, 2005.

[21] Trusted Computing Group. TPM design principles. `https://www.trustedcomputinggroup.org/groups/tpm/`, 2006.

[22] Even Balance Inc. Punkbuster. `http://www.evenbalance.com/index.php?page=info.php`, link acessado em março de 2006.

[23] Qualcomm Incorporated. Binary runtime environment for wireless, brew. `http://brew.qualcomm.com/brew/en/`, link acessado em junho de 2007.

[24] DFC intelligence. Online game market forecasted to reach $13 billion by 2011. `http://www.dfcint.com/news/prjune62006.html`, link acessado em junho de 2007.

[25] Level-Up Interactive. Ragnarok brasil. `http://games.levelupgames.com.br/ragnarok/`, link acessado em dezembro de 2006.

[26] Yedang Entertainment Kaizen Games. Priston tale brasil. `http://www.priston.com.br/`, link acessado em dezembro de 2006.

[27] Yedang Entertainment Kaizen Games. Priston tale brasil, loja de itens. `http://www.priston.com.br/pagamento/principal_vitrine.aspx`, link acessado em dezembro de 2006.

[28] Borje Karlsson and Bruno Feijó. An overview on security in networked computer games. *Scientia - Estudos Interdisciplinares em Computação - UNISINOS*, 15(2), December 2004.

[29] Junbaek Ki, Jung Hee Cheon, Jeong-Uk Kang, and Dogyun Kim. Taxonomy of online game security. *The Electronic Library: International Journal for the application of technology in information environments*, 22(1):65–73, 2004.

[30] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Rfc 2104, hmac: Keyed-hashing for message authentication. `http://www.faqs.org/rfcs/rfc2104.html`, February 1997.

[31] Ho Lee, Eric Kozlowski, Scott Lenker, and Sugih Jamin. Multiplayer game cheating prevention with pipelined lockstep protocol. In Nakatsu and Hoshino [37], pages 31–9.

[32] Symbian Ltd. Symbian ltd. ownership. `http://www.symbian.com/about/overview/ownership/ownership.html`, link acessado em junho de 2007.

[33] Symbian Ltd. The symbian os. `http://www.symbian.com/symbianos/index.html`, link acessado em junho de 2007.

[34] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press, 1996. URL: `http://cacr.math.uwaterloo.ca/hac`.

[35] Microsoft. X-box live. `http://www.xbox.com/en-us/live/`, link acessado em abril de 2007.

[36] Sun Microsystems. The Java-ME platform. `http://java.sun.com/javame/index.jsp`, link acessado em junho de 2007.

[37] Ryohei Nakatsu and Jun'ichi Hoshino, editors. *Entertainment Computing: Technologies and Applications, IFIP First International Workshop on Entertainment Com-*

puting (IWEC 2002), May 14-17, 2002, Makuhari, Japan, volume 240 of *IFIP Conference Proceedings*. Kluwer, 2003.

[38] ETNews Korea IT news. The monthly sales of partial fee-charging games reaching almost 10 billion won. `http://english.etnews.co.kr/news/detail.html?id=200607180005`, July 2006.

[39] Nintendo. Wii. `http://wii.nintendo.com/`, link acessado em junho de 2007.

[40] National Institute of Standards and Technology. Publication 46-2: Digital encryption standard. `http://www.itl.nist.gov/fipspubs/fip46-2.htm`, December 1993.

[41] National Institute of Standards and Technology. Publication 186: Digital signature standard. `http://www.itl.nist.gov/fipspubs/fip186.htm`, May 1994.

[42] National Institute of Standards and Technology. Publication 180-1: Secure hash standard. `http://www.itl.nist.gov/fipspubs/fip180-1.htm`, April 1995.

[43] National Institute of Standards and Technology. Publication 197: Advanced encryption standard. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`, November 2001.

[44] Sierra Online. `http://www.sierra.com/en/home.html`, link acessado em dezembro de 2006.

[45] PS2dev.org. Playstation independent developers. `http://www.ps2dev.org`.

[46] R. L. Rivest, A. Shamir, and L. M. Adelman. A method for obtaining digital signatures and public-key cryptosystems. Technical Report MIT/LCS/TM-82, 1977.

[47] Sony. Playstation 3. `http://www.playstation.com/products.html`, link acessado em maio de 2007.

[48] The Korea Times. Gaming bill has holes. `http://news.empas.com/show.tsp/cp_kt/20061224n02489/?kw=controversial%20%3Cb%3E%26%3C%2Fb%3E`, December 2006.

[49] UCDao. Gold farming. `http://ucdao.com/`, link acessado em dezembro de 2006.

[50] Jeff Yan and Hyun-Jin Choi. Security issues in online games. *The Electronic Library: International Journal for the application of technology in information environments*, 20(2):125–133, 2002.

[51] Jeff Yan and Brian Randell. A systematic classification of cheating in online games. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–9, New York, NY, USA, 2005. ACM Press.