UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

KNOCKID - UMA NOVA ABORDAGEM PARA AUTENTICAÇÃO BASEADA NA CAMADA DE REDE

DIÓGENES FERREIRA REIS

ORIENTADOR: ANDERSON CLAYTON ALVES NASCIMENTO

DISSERTAÇÃO DE MESTRADO EM
ENGENHARIA ELÉTRICA

BRASÍLIA/DF: AGOSTO/2012.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

KNOCK ID – UMA NOVA ABORDAGEM PARA AUTENTICAÇÃO BASEADA NA CAMADA DE REDE

DIÓGENES FERREIRA REIS

DISSERTAÇÃO DE MESTRADO ACADÊMICO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

Anderson C A Nascimento

ANDERSON CLAYTON ALVES NASCIMENTO, Dr., PPGEE/UNB
(ORIENTADOR)

Flávio Elias

FLÁVIO ELIAS GOMES DE DEUS, Dr., PPGEE/UNB
(EXAMINADOR INTERNO)

GEORGES DANIEL AMVAME-NZE, Dr., FGA/UNB
(EXAMINADOR EXTERNO)

Brasília, 09 de agosto de 2012.

FICHA CATALOGRÁFICA

REIS, DIOGENES FERREIRA

KnockID - Uma Nova Abordagem para Autenticação Baseada na Camada de Rede. [Distrito Federal] 2012.

xv, 78 p., 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2012).

Dissertação de Mestrado - Universidade de Brasília.

Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. Autenticação em Camada de Rede
2. PortKnocking
3. Segurança de serviços
4. Proteção do servidor

I. ENE/FT/UnB

II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

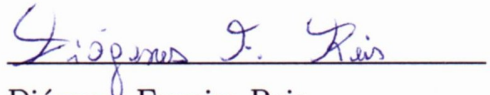
Reis, D. F. (2012). KnockID - Uma Nova Abordagem para Autenticação Baseada em Camada de Rede. Dissertação de Mestrado em Engenharia Elétrica, Publicação PPGEE.DM - 487/2012, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 78p.

CESSÃO DE DIREITOS

NOME DO AUTOR: Diógenes Ferreira Reis.

TÍTULO DA DISSERTAÇÃO DE MESTRADO: KnockID - Uma Nova Abordagem para Autenticação Baseada em Camada de Rede.

GRAU / ANO: Mestre / 2012



Diógenes Ferreira Reis

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica
70.910-900 Brasília - DF - Brasil.

DEDICATÓRIA

À minha amada esposa, à minha família
e aos meus amigos.

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus por mais uma etapa concluída na minha vida e à minha esposa que me acompanhou e apoiou que sempre me incentivou em meus trabalhos. Agradeço também a minha família, que mesmo à distância mandava palavras de apoio. Agradeço sinceramente ao orientador, Prof. Anderson Clayton, pela confiança depositada e pelo apoio durante todo o curso de mestrado. Devo agradecer também a todas as pessoas que colaboraram para a conclusão deste trabalho, principalmente o pessoal do laboratório de Crypto, em especial à Bernardo David e Adriana Bastos, os quais me auxiliram nos momentos cruciais do trabalho.

Diógenes Ferreira Reis

RESUMO

KNOCKID - UMA NOVA ABORDAGEM PARA AUTENTICAÇÃO BASEADA EM CAMADA DE REDE

Autor: Diógenes Ferreira Reis

Orientador: Anderson Clayton Alves Nascimento

Programa de Pós-graduação em Engenharia Elétrica

Brasília, Agosto de 2012

Na maioria dos serviços de rede, a autenticação do usuário é feita na camada de aplicação, o que acarreta uma certa exposição já que a interface de autenticação deve ser acessível para todos os computadores que acessam o servidor hospedeiro da aplicação. Na prática, isso significa que as portas *TCP/UDP* correspondentes às aplicações estão abertas a qualquer computador que possa alcançar o servidor, tornando o ambiente de rede do servidor vulnerável a ataques de *fingerprinting*, *port scanning* e ataques diretos as aplicações (*i.e* validação de parâmetros e estouro de pilha).

Com o propósito de solucionar esses problemas é proposto o *KnockID*, um novo protocolo de autenticação de usuário em camada de rede baseado em PortKnocking e em protocolo criptográfico de identificação. No *KnockID*, cada usuário é identificado por um par de chaves e é concedido o acesso a portas específicas depois que ele prova a sua identidade. *KnockID* é baseado num protocolo criptográfico e proporciona segurança contra *man-in-the-middle*, espionagem do tráfego de rede e ataques de repetição (*replay*). Ele pode ser implementado para atender à necessidades diferentes, tais como a construção de uma rede central de serviços de autenticação de usuário, ocultando sensíveis (e/ou inseguros) serviços de rede ou agindo como uma camada extra de segurança. Além disso, o *KnockID* pode ser transparentemente integrado em qualquer ambiente de rede executando aplicativos legados e implementado simplesmente usando bibliotecas criptográficas comuns existentes.

ABSTRACT

KNOCKID - A NEW APPROACH TO NETWORK LAYER BASED USER AUTHENTICATION

Author: Diógenes Ferreira Reis

Supervisor: Anderson Clayton Alves Nascimento

Programa de Pós-graduação em Engenharia Elétrica

Brasília, August of 2012

In most network services, user authentication is carried out at the application layer, which makes it necessary for applications to expose their own authentication interface. Practically, it means that the TCP/UDP ports corresponding to a given application are open to any host that can reach the application server, thus making the server network environment vulnerable to fingerprinting, port scanning and direct attacks to applications (*i.e.* parameter validation and buffer overflow).

In order to solve these issues, we propose KnockID, a new network layer based user authentication protocol that combines classical Portknocking techniques with cryptographic identification protocols and message authentication. In KnockID, each user is uniquely identified by a key pair and is granted access to specific ports after he proves his identity. KnockID is based on cryptographically sound protocols and provides security against man-in-the-middle, eavesdropping and replay attacks. It can be deployed to fulfill different needs such as building a central network user authentication service, hiding sensitive (and/or insecure) network services or acting as an extra security layer. Moreover, KnockID can be transparently integrated into any network environment running legacy applications and simply implemented using common cryptographic libraries and existing Portknocking services.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE PROTOCOLOS

LISTA DE SÍMBOLOS, NOMENCLATURA E ABREVIAÇÕES

IP: Internet Protocol.

TCP: Transmission Control Procotol.

UDP: User Datagram Protocol.

SPA: Single Packet Authentication.

AP: Authentication Packet.

NAT: Network Address Translation.

ISN: Initial Sequence Number.

IANA: Internet Assigned Numbers Authority.

VPN: Virtual Private Network.

AES: Advanced Encryption Standard.

CBC: Cipher Block Chaining.

CFB: Cipher Feedback.

OFB: Output Feedback.

MAC: Message Authentication Code.

HMAC: Hashed-based Message Authentication Code.

SHA: Secure Hash Algorithm.

MD5: Message-Digest Algorithm 5.

DoS: Denial of Service.

PKI: Public Key Infrastructure.

TSL: Transport Layer Security.

SSL: Security Sockets Layer.

IPSec: IP Security Protocol.

AH: Authentication Header.

ESP: Encapsulating Security Payload.

DES: Data Encryption Standard.

IDEA: International Data Encryption Algorithm.

OTK: One Time Knock.

IDS: Intrusion Detection System.

FWKNOP: FireWall KNock OPerator.

MIME: Multipurpose Internet Mail Extensions.

DNAT: Destination Network Address Translation.

GPG: GNU Privacy Guard.

PGP: Pretty Good Privacy.

MITM: Man-in-the-Middle.

SMS: Short Message Service.

CPAN: Comprehensive Perl Archive Network.

OAEP: Optimal Asymmetric Encryption Padding.

PKCS: Public Key Criptography Standards.

UC: Universally Composable.

1 INTRODUÇÃO

A comunicação acompanha a evolução da humanidade, caracterizando-se como uma peça essencial e necessária para que o homem chegasse no estágio atual. Ao longo dos séculos houveram muitos avanços no sentido de tornar os meios de comunicação mais eficazes. Notícias eram enviadas por mensageiros que percorriam longas distâncias para levar uma mensagem. Animais foram utilizados como transportadores de informação, como é o caso dos "pombos correios". Nas últimas décadas os avanços na área de comunicação foram intensificados, sobretudo pelas novas possibilidades e ferramentas proporcionadas com o surgimento da *Internet*.

A *Internet*, por si só, é um meio que fornece inúmeras possibilidades de comunicação, como textos, vídeos, músicas, *chats*. Baseia-se na pilha de protocolos *TCP/IP* (*Transport Control Protocol*/*Internet Protocol*) (SOCOLOFSKY; KALE, 1991), sendo formada por redes de computadores interconectadas em escala mundial. Foi concebida para ser uma rede robusta e confiável, no sentido de que a rede se mantém em funcionamento mesmo que parte dela falhe ou seja afetada por alguma catástrofe. Entretanto, o quesito segurança não foi uma preocupação prevista na sua concepção.

Inúmeras vulnerabilidades já foram relatadas sobre a segurança dos protocolos que compõem a pilha *TCP/IP*. A conexão de um computador na *Internet* pode expô-lo a uma série de ameaças virtuais. As aplicações ou serviços ativos no computador podem ser uma porta de entrada para um atacante, acarretando o comprometimento das informações nele contidas. Nesse caso, a forma de proteção óbvia é a restrição de acessos somente a usuários conhecidos. Uma solução comumente adotada é a delegação da autenticação dos acessos remotos às aplicações/serviços, de modo que somente os usuários autorizados tenham acesso. No entanto, o ato de encarregar às aplicações a tarefa de autenticar usuários pode ter os seus inconvenientes: algumas aplicações são grandes e complexas, apresentando falhas de implementação que poderiam ser explorado por usuários maliciosos; algumas aplicações não foram projetadas para incorporar autenticação de usuários.

A solução tradicional para limitar o acesso de usuários é o uso de *Firewall*. Um *Firewall* habitual atua restringindo o acesso de conexões de rede, baseando-se em informações

como endereço *IP*, portas *TCP* ou aplicações. Embora o emprego de *Firewall* seja eficiente em alguns casos, oferece brechas que podem ser exploradas. Regras muito restritivas baseadas em endereço *IP* podem ser contornadas por um atacante utilizando a técnica *spoofing*. Há também o fato de que nem todos os usuários autorizados tem um endereço *IP* previsível, levando ao uso de regra mais genérica, acarretando assim a abertura do *Firewall* a outros computadores além do desejado.

Uma porta aberta num *Firewall* pode oferecer informações relevantes para o início de um ataque. Por exemplo, a utilização de ferramentas como *ping sweep*, *OS fingerprinting* e *port scanner* pode revelar o sistema operacional e a existência de serviços em computadores supostamente protegidos por um *Firewall*. Esses dados podem servir como base para um ataque baseado em *exploits*, permitindo ao atacante acessar o computador. Uma forma de contornar essa questão de exploração de portas abertas é a criação de um mecanismo de autenticação na camada de rede para posterior liberação da porta no *Firewall*. A idéia básica é deixar o *Firewall*, por padrão, configurado para bloquear qualquer acesso externo, de forma que o acesso é liberado dinamicamente após a autenticação do usuário remoto na camada de rede. Esse novo mecanismo de autenticação baseada na camada de rede com integração a um *Firewall* é o objeto de estudo desta dissertação.

1.1 JUSTIFICATIVA E MOTIVAÇÃO

Mecanismos de autenticação são importantes para restringir o acesso a um sistema ou a um servidor, dando permissão somente aos usuários que realmente são autorizados. O acesso de pessoas não autorizadas caracteriza uma grave falha de segurança o que deve ser evitada. Logo, o desenvolvimento de técnicas de autenticação realmente seguras deve ser objeto de preocupação constante a fim de prover uma proteção efetiva de um sistema. Porém, verifica-se que as soluções tradicionais de autenticação pelas aplicações podem acarretar falhas de segurança. Além disso, o uso de outras técnicas de autenticação (*PortKnocking* e *Single Packet Authorization*), apesar de avançarem na questão de proteção dos sistemas, são um tanto grosseiras por se terem uma abordagem heurística. Nesse contexto, propõem-se o *KnockID*, um protocolo de autenticação na camada de rede que emprega alguns conceitos do *PortKnocking* e de protocolo criptográfico de identificação. Além de ser facilmente gerenciável, a abordagem proposta é segura contra vários ataques atuais (programas de varredura, ataques de repetição, espionagem de tráfego de rede), podendo ser utilizado como substituto para as técnicas de autenticação convencionais. O protocolo proposto pode ser usado tanto como um

serviço de autenticação centralizado ou como uma camada extra para proteção de aplicações inseguras. A abordagem utilizada no *KnockID* se difere de outras soluções de autenticação (*PortKnocking*, *Single Packet Authorization*) pelo fato de que se baseia em protocolos criptográficos com segurança demonstrável, contrastando com a abordagem heurística convencional.

1.2 TRABALHOS RELACIONADOS

No intuito de aperfeiçoar e propor novos métodos de autenticação, várias técnicas foram propostas. Destaca-se as principais abordagens que mostra a autenticação sendo realizada na camada de rede, como o *PortKnocking* e o *Single Packet Authorization* (*SPA*). O conceito de *PortKnocking* surgiu no trabalho (BARHMAN et al., 2002) com objetivo de criar um mecanismo de autenticação na camada de rede. O *PortKnocking* é uma forma de autenticação baseado no envio de uma sequência de pacotes *TCP/UDP* em diferentes portas. A sequência de portas, também chamada de sequência de *knock*, é o segredo que permite a autorizar o acesso do usuário remoto e proceder com a abertura de regra no *Firewall* para a conexão posterior ao serviço desejado.

Com o mesmo objetivo do *Portknocking*, foi também proposto no mesmo trabalho (BARHMAN et al., 2002) o desenvolvimento do *SPA*. O *SPA* também é uma forma de autenticação do usuário remoto na camada de rede, diferenciando da técnica de *PortKnocking* pelo fato de que só envia um único pacote com todo o conteúdo necessário para autenticação. O pacote do *SPA*, conhecido como AP (*Authentication Packet*), é cifrado e enviado do cliente ao servidor, o qual decifra e verifica se os dados do pacote correspondem à uma solicitação legítima de autorização. Em caso positivo é liberado uma regra no *Firewall* para permitir a conexão no serviço solicitado.

Ambas as técnicas de *PortKnocking* e *SPA* contribuiram para o avanço do mecanismo de autenticação, tanto por propor um modelo baseado em camada de rede como por visar maior proteção às aplicações em execução num servidor pelo ocultamento deste. No entanto, tais técnicas foram construídas heuristicamente, apresentando vulnerabilidades que podem ser explorados para ataque. As técnicas *PortKnocking* e *SPA*, suas principais implementações e as suas características serão estudadas no presente trabalho.

1.3 METODOLOGIA

O intuito desta dissertação é estudar protocolos criptográficos de identificação com segurança demonstrável e conciliar com os conceitos de autenticação sugeridos nas técnicas de *PortKnocking* e *SPA*. O fruto desse estudo é o a proposição de uma nova técnica/método de autenticação baseado em camada de rede, o *KnockID*, que tem segurança demonstrável e também possua algumas características interessantes, como simplicidade, ocultamento das aplicações em execução no servidor, proteção efetiva do servidor de forma integrada a um *Firewall* a fim de eliminar a maioria dos ataques de *script kiddies* e programas de varredura. Por fim, será desenvolvido o protótipo *KnockID*, o qual tem a finalidade de atestar a viabilidade de implementação do técnica proposta, além de verificar o funcionamento básico do método de autenticação.

1.4 ORGANIZAÇÃO DA DISSERTAÇÃO

Este trabalho está organizado da seguinte forma: o capítulo 2 apresenta conceitos teóricos relevantes para o entendimento de uma rede de computadores, ferramentas para a proteção de uma rede (criptografia) e ameaças que podem comprometer o funcionamento de uma rede de computadores. O capítulo 3 se trata de uma referência bibliográfica com o intuito de introduzir os principais métodos de autenticação baseados em camada de rede existentes atualmente e estudados neste trabalho, mostrando as principais implementações e suas respectivas características. O capítulo 4 descreve a proposta deste trabalho, o *KnockID*, abordando as definições do protocolo, apresentação do funcionamento do *KnockID* e a respectiva análise de segurança. O capítulo 5 apresenta os detalhes da implementação do protótipo do *KnockID* e os resultados alcançados. E, finalmente, no capítulo 6 é apresentado a conclusão do trabalho desenvolvido e recomendações para trabalhos futuros.

2 BASE TEÓRICA

Este capítulo aborda sobre conhecimentos teóricos relevantes para o desenvolvimento das idéias da presente dissertação. Serão apresentados noções gerais de redes, de segurança e criptografia.

2.1 NOÇÕES DE REDE

O conceito redes de computadores surgiu em meio à necessidade de aperfeiçoar as formas de comunicação. De forma simples, uma rede de computadores pode ser definida como um ou mais dispositivos interligados e comunicando-se entre si. Estes dispositivos podem ser computadores, *laptops*, *netbooks*, celulares, *smartphones*, *tablets*, enfim, qualquer equipamento que possa interagir de acordo com protocolos pré-definidos. Todos estes dispositivos que podem se conectar a uma rede de computadores serão denominados pelo termo computador.

A fim de diminuir a complexidade de uma rede de computadores e facilitar o seu desenvolvimento, foi sugerido a organização numa arquitetura de camadas. Nessa arquitetura, cada camada, ou nível, tem uma função específica dentro do processo de comunicação. As regras e modo de funcionamento de cada camada são regidas pelos protocolos. A arquitetura em camadas proporciona um modo estruturado de projetar as funcionalidades necessárias para a comunicação, além facilitar a atualização do sistema, devido ao seu aspecto modular. Dentre os modelos existentes de arquitetura em camadas, será utilizado o modelo *TCP/IP*, o qual é estruturado com as seguintes camadas, segundo (KUROSE; ROSS, 2006):

1. Física - responsável pela codificação e decodificação de sinais para transmitir dados num meio físico, como par de fios de cobre trançado, cabo coaxial, fibra óptica ou ar. Essa camada se preocupa com os aspectos de hardware, com modulação e melhores técnicas para transmitir os bits num meio de transmissão;

2. Enlace - no processo de comunicação entre dois computadores, pode ser necessário que dispositivos de redes intermediários (nós) auxiliem transportando a informação, permitindo fluir dados da origem para o destino. A camada de enlace é responsável pela comunicação entre cada nó da rede, ou seja, responsável

pela movimentação dos dados de um elemento de rede até um elemento subjacente. Os protocolos e funcionalidades dessa camada dependem do tipo de enlace entre dois dispositivos. O *hub* e o *switch* são dispositivos típicos dessa camada;

3. Rede - responsável pela definição do caminho a ser percorrido entre os dispositivos de redes intermediários na comunicação entre os computadores. Para tal objetivo, há nessa camada endereços globais que identificam os dispositivos de rede, chamado endereço IP. Os roteadores são os dispositivos que atuam na camada de rede e auxiliam no processo de definição do caminho de uma informação, baseando-se em rotas ou em protocolos de roteamento;

4. Transporte - responsável pela comunicação ponto-a-ponto entre os computadores. Entre as funcionalidades dessa camada, pode-se citar a segmentação das informações da camada de aplicação, verificação de erros na comunicação, multiplexação/demultiplexação de dados de diferentes aplicações, além de permitir criação de conexões confiáveis;

5. Aplicação - é onde residem as aplicações de rede e seus protocolos. É a camada que interage com o usuário e abstrai o processo de comunicação. Geralmente as funções de autenticação na comunicação entre computadores é desempenhada por esta camada.

Na arquitetura de camadas, a camada inferior presta serviço para a camada superior. Por exemplo, a camada de transporte presta serviço à camada de aplicação, segmentando as informações e adicionando dados para criar uma conexão lógica entre os computadores envolvidos na comunicação. Cada camada tem um protocolo associado, e, quando tomados em conjunto, os protocolos das várias camadas são denominados pilha de protocolo.

A *Internet* pode ser vista como uma rede de computadores de escala mundial, formada pela composição de várias redes interligadas ao redor do mundo. A relevância da *Internet* nos dias atuais é inquestionável, seja pelo fato de ser o maior repositório de informações acessíveis a qualquer usuário que esteja nela conectada, ou por permitir conectividade entre *hosts* longíquos. A *Internet* foi criada de acordo com a pilha de protocolos *TCP/IP* (SOCOLOFSKY; KALE, 1991), onde cada *host* é identificado de forma única pelo seu endereço *IP*. Logo, é necessária a compreensão dos principais protocolos que compõem a pilha *TCP/IP* para as questões relativas à segurança das comunicações advindas da *Internet*.

2.1.1 Internet Protocol

Internet Protocol (*IP*) é o protocolo da camada de rede utilizado para a Internet. De fato, o *IPv4* especificado em (POSTEL, 1981a) é a versão do protocolo *IP* que é ainda comumente utilizado. Há uma nova especificação para uma nova versão *IPv6*, que expande o número de endereços *IP* disponíveis e acrescenta novas funcionalidades. Entretanto, o *IPv6* não será discutido no presente trabalho, de forma que a referência *IP* é relativa ao *IPv4*. Ressalta-se que tal fato não impede que os estudos e resultados deste trabalho não possam ser aplicados ao *IPv6* e sim que apenas não foi testado num ambiente com o protocolo *IPv6* em funcionamento.

O protocolo *IP* estabelece que cada computador deve ter um endereço *IP* para identificá-lo na rede. Logo, o endereço *IP* deve ser único na rede para que não haja ambiguidade na comunicação. O endereço *IP* é formado por um campo de 32 bits, geralmente expresso como 4 octetos separados por pontos na notação decimal. É baseado na informação fornecida pelo endereçamento *IP* que são construídos os caminhos, ou rotas, pelos quais os dados da origem trafegam até chegar ao computador de destino. As rotas podem ser definidas por um administrador, chamadas de rotas estáticas, ou são criadas de forma dinâmica por meio de protocolos de roteamento, as rotas dinâmicas.

É interessante notar que o protocolo *IP*, apesar de definir as rotas para o tráfego dos dados, não é um protocolo confiável. É um protocolo chamado de melhor esforço, onde não há garantias de entrega do pacote ao destino. Isso significa que o pacote pode chegar fora de ordem, quando comparado aos pacotes enviados anteriormente, duplicado, ou mesmo perdido no caminho. A garantia de entrega é função da camada de transporte.

Devido ao tamanho limitado do endereço *IP* no *IPv4* e ao crescimento da Internet, foi diagnosticado que a faixa de endereços *IP* disponíveis para alocação deve se esgotar. Como uma forma de economia de endereços IP foi definido em (REKHTER et al., 1996) o conceito de endereço *IP* privado. Os endereços *IP* privados são destinados para o uso em redes privadas que, num primeiro momento, não tem necessidade de se conectar à *Internet*, servindo assim para prover conectividade numa rede particular, como por exemplo uma rede interna de uma empresa. Conforme a especificação, os endereços privados não são roteáveis na *Internet*, sendo descartados os pacotes *IP* que contem tais endereços nos roteadores.

Posteriormente, foi criado o conceito de *Network Address Translator* (*NAT*), o que

prevê a necessidade de algum computador da rede privada se conectar à *Internet*, ainda que utilize um endereço *IP* privado (SRISURESH; HOLDREGE, 1999). A idéia por trás do *NAT* é realizar a tradução dos endereços privados da rede interna para um endereço público no dispositivo de borda com a *Internet*, geralmente um roteador. O *NAT* auxilia no processo de economia dos endereços *IP*, porém tem alguns aspectos indesejáveis quanto à questão de segurança. Regras de permissão definidas em *Firewall* baseadas em endereço *IP* pode liberar acesso à computadores indesejados se houver *NAT*, uma vez que vários computadores podem compartilhar o mesmo endereço público nessa configuração.

2.1.2 Protocolos de Transporte

Na *Internet*, o *Transmission Control Protocol* (*TCP*) é um dos principais protocolos utilizados da camada de transporte. O *TCP*, entre outras atribuições, cria uma conexão entre dois computadores, controla o fluxo de dados entre os mesmos, além de garantir a confiabilidade na comunicação (POSTEL, 1981b). O *TCP* exerce o papel de preparar as informações das aplicações para ser transmitido numa rede.

No processo de comunicação, os dados da camada de aplicação são fragmentados para a transmissão na rede. Cada fragmento é chamado de segmento. Um segmento é identificado por números de sequência que são utilizados para o ordenamento dos dados e também para a verificação de recebimento de segmentos duplicados. Uma vez que um segmento é transmitido e chega no destino, o receptor deve enviar um reconhecimento ao emissor (*acknowledge*) para informar que a informação foi recebida corretamente. O reconhecimento, também conhecido como *ACK*, é associado ao próximo número de sequência esperado, indicando que todos os segmentos com número de sequência inferiores foram recebidos no destino.

A estrutura do protocolo *TCP* é apresentada na Figura 2.1. No cabeçalho do protocolo existem alguns *bits* especiais chamados de *flags*. De forma simplificada, pode-se dizer que as *flags* indicam como o protocolo *TCP* deve agir quando um determinado segmento tem alguma *flag* ativada. Por exemplo, a *flag ACK* indica que o segmento carrega um reconhecimento, o que leva à verificação do campo *acknowledge number* o segmento reconhecido.

Outra *flag* importante é a *flag SYN*, a qual indica o início ou estabelecimento de uma conexão *TCP*. Nessa fase de estabelecimento, os computadores envolvidos na comunicação trocam mensagens para a criação de uma conexão. O processo de estabeleci-

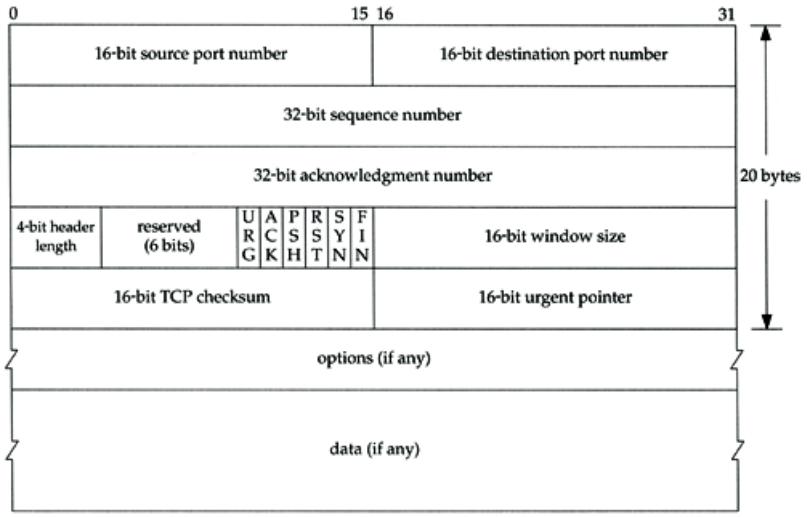


Figura 2.1: Estrutura do cabeçalho do protocolo *TCP* (STEVENS, 1993).

mento de conexão é conhecido como *three-way handshake*, consistindo em 3 mensagens trocadas entre o cliente e o servidor, conforme a Figura 2.2. A primeira mensagem é a requisição de conexão: o cliente envia um segmento com um número de sequência inicial *ISN*, que pode ser aleatório, e com a *flag SYN* ativada. O servidor ao receber o segmento com *SYN*, aloca recursos para a conexão, e envia um segmento de volta ao cliente também com *ISN* aleatório e com as *flags SYN* e *ACK* ativadas. Após o recebimento do segmento do servidor, o cliente retorna um segmento com a *flag ACK* ativada, sinalizando a criação da conexão *TCP* e o fim da fase de estabelecimento.

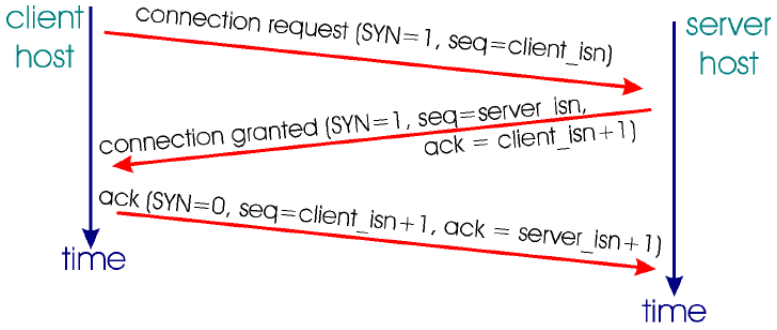


Figura 2.2: Estabelecimento de conexão *TCP* (KUROSE; ROSS, 2006).

No *TCP* os processos das aplicações são identificados pelos campos de portas. Com a porta o *TCP* consegue diferenciar os dados e entregá-los para a aplicação correspondente. Sempre há duas portas numa comunicação *TCP*. Geralmente, o cliente define uma porta aleatória (porta de origem) e se conecta ao servidor especificando a porta referente à aplicação ou serviço (porta de destino). Para que uma conexão seja criada, o servidor deve ter um processo servidor (*daemon*) para ficar "escutando" a respectiva porta da aplicação e atender às requisições de conexão de clientes. Assim, um serviço

ativo no servidor sempre fica à espera de um pedido de conexão na sua porta, e, dizemos que o servidor está com a "porta aberta" para a referida aplicação. Uma porta consiste num endereço de 16 *bits*, resultando na faixa de 0 à 65535 portas. A *Internet Assigned Numbers Authority* (*IANA*[1]) é a entidade responsável por administrar e manter atualizada a lista de portas do *TCP* (COTTON et al., 2011). As faixas 0-1023 (*well known ports*) e 1024-49151 (*registered ports*) são alocadas pela *IANA* e as demais portas são conhecidas como portas privadas para livre associação.

2.1.3 Firewall

Uma medida de segurança comumente adotada para aumentar a segurança de uma rede é a implementação de um *Firewall*. *Firewall* pode ser definido como um mecanismo de segurança que realiza o controle de acesso à uma rede ou um computador. O controle de acesso fornecido pelo *Firewall* é no sentido de limitar as conexões de rede permitidas, aceitando ou rejeitando pacotes de rede baseado nos endereços de origem e outras características. Existem *Firewalls* pessoais e *Firewalls* de rede. *Firewalls* pessoais são programas instalados em computadores comuns para dar maior segurança a usuários finais. *Firewalls* de rede correspondem a dispositivos de rede com *hardware* e *software* dedicados, colocados estrategicamente numa rede para realizar o controle de acesso dos pacotes entre a zona não confiável, a *Internet*, e a zona confiável, a rede interna protegida pelo *Firewall*.

Há alguns tipos comuns de *Firewalls*, conforme Stallings (2008): filtros de pacotes, *Firewall* de inspeção com estado, *Gateways* em nível de aplicação e *Gateways* em nível de circuito. De forma resumida, é apresentado as características de cada tipo de *Firewall*:

- Filtro de pacotes - utiliza campos de protocolos *IP* e *TCP* contidos nos pacotes de rede para realizar o controle de acesso. Os campos utilizados são os endereços *IP* de origem e destino, portas de origem e destino da camada de transporte e o tipo de protocolo de transporte (*TCP* ou *UDP*). Além dos campos de protocolos, pode-se utilizar a interface de entrada ou saída nos *Firewalls* que tem mais de uma interface de rede. Corresponde a um tipo de *Firewall* simples baseiando-se em informações limitadas para composição de suas regras e tomadas de decisão quanto ao tráfego de pacotes. A segurança percebida por um *Firewall* baseado em filtro de pacotes é pequena, é susceptível a brechas de seguranças causadas

[1] www.iana.org

por configurações impróprias, além de ser frágil a ataques que exploram vulnerabilidades de aplicações específicas, já que não examinam dados da camada superior;

- *Firewall* de inspeção com estado - este tipo de *Firewall* mantém as características de um *Firewall* baseado em filtro de pacotes, mas acrescenta a funcionalidade de analisar o estado das conexões *TCP* para tomar decisões sobre a filtragem de pacotes. Logo, deve observar as *flags TCP* e armazenar o estado de cada conexão. A análise da conexão para tomada de decisão torna o *Firewall* mais eficaz, uma vez que pode bloquear pacotes mal-formados que poderiam ser destinados a algum tipo ataque ou mapeamento da rede interna;

- *Gateways* em nível de aplicação - também é conhecido como servidor *proxy*, atua como um retransmissor de tráfego em nível de aplicação. Funciona como um servidor intermediário para autenticação de aplicações, como *FTP* ou *Telnet*, onde o usuário se conecta e deve fornecer informações de autenticação válidas. Se os dados de autenticação forem válidos, o *gateway* contacta a aplicação no computador remoto e repassa os segmentos *TCP* contendo os dados de aplicação entre as duas extremidades. Assim, o *gateway* deve implementar o código *proxy* para a aplicação específica para que o serviço seja aceito e encaminhado pelo *Firewall*. Este tipo de *Firewall* tende a ser mais seguro que filtros de pacote, já que só precisa examinar algumas aplicações permitidas ao invés de lidar com os parâmetros de protocolos *TCP* e *IP*. Porém, tem a desvantagem de apresentar *overhead* adicional em cada conexão, uma vez que cada conexão de usuário são na verdade duas conexões com o *gateway* no papel intermediário;

- *Gateways* em nível de circuito - apresenta o estabelecimento de duas conexões, num aspecto similar ao *gateway* em nível de aplicação, não permitindo uma conexão *TCP* de ponta a ponta. O *gateway* prepara duas conexões, uma entre ele mesmo e o usuário *TCP* interno, e uma entre ele mesmo e um usuário *TCP* externo. Com o estabelecimento das duas conexões, o *gateway* repassa sem examinar o conteúdo. A segurança está na determinação de quais conexões serão permitidas, como no caso do administrador confiar nos usuário internos. O *gateway* também pode ser configurado como um *gateway* em nível de aplicação para as conexões entrantes. O problema desse tipo de *Firewall* é o *overhead* pela duplicação das conexões *TCP*.

Se uma conexão não se enquadra nas políticas de segurança de um *Firewall*, esta

conexão deve ser bloqueada ou negada no serviço solicitado. Esse processo de bloqueio ou negação do serviço pode ser feito numa das seguintes formas:

- *Reject/Deny* - os pacotes são descartados e o *Firewall* envia uma mensagem *ICMP_PORT_UNREACHABLE* à origem. Essa forma fornece à origem uma informação de que existe um computador para o *IP* de destino.
- *Drop* - os pacotes são descartados de forma silenciosa, sem que nenhuma informação seja enviada à origem. Assim, a origem assume que nenhum serviço é executado pelo computador de destino, ou ainda que o esse computador não existe.

Os *Firewalls* baseado em filtro de pacotes ou inspeção de estado tem a característica de que as regras de políticas de segurança configuradas abrem brechas que podem ser exploradas por atacantes. Ao permitir uma conexão, uma porta é aberta no *Firewall* para que um usuário mal intencionado possa executar algum *port scanning* ou *OS fingerprinting*, obtendo informações que podem ser exploradas. Esse comportamento não acontece nos *Firewalls* baseados em *Gateway* em nível de aplicação e *Gateway* em nível de circuito, mas há o problema de *overhead* acarretado pela duplicação de conexões *TCP*, além do custo desses tipos de *Firewall* ser elevado.

2.2 CRIPTOGRAFIA

O termo "criptografia" é constantemente empregado quando o assunto é segurança. O seu emprego se deve ao fato de fornecer ferramentas para alcançar confidencialidade, autenticidade, integridade dos dados e irretratabilidade (STALLINGS, 2008). A sua aplicação pode ser percebida num cenário onde duas partes se comunicam num canal inseguro, tendo como premissa manter a privacidade da comunicação mesmo na presença de um adversário (Figura 2.3). Nesse exemplo, a confidencialidade trata de transmitir a mensagem de tal modo que somente o destinatário possa ser capaz de compreender o seu significado. Por sua vez, a integridade se refere à capacidade do destinatário de verificar se a informação foi alterada durante o envio no meio inseguro. Já a autenticidade é a habilidade do destinatário de identificar o emissor da informação e constatar que o mesmo a enviou. Por fim, o não-repúdio ou irretratabilidade trata da questão de que um emissor ao enviar alguma informação, este não deve ser capaz de negar a autoria da mensagem.

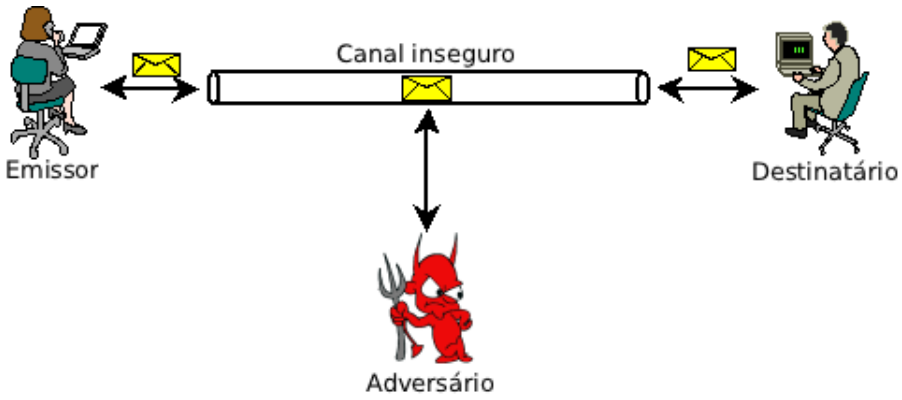


Figura 2.3: Comunicação entre duas partes num canal inseguro.

As ferramentas fornecidas pela criptografia são essencialmente as primitivas criptográficas e os protocolos criptográficos. Uma primitiva criptográfica é uma ferramenta que consiste num algoritmo ou um *software* derivado de uma construção de engenharia ou de um problema matemático com alto grau de dificuldade de ser resolvido, com objetivo de proporcionar segurança contra alguma parte que não conheça a chave criptográfica. A utilização de uma primitiva criptográfica "embaralha" os dados (cifragem), tornando inviável a extração de informação por uma parte não autorizada. A chave criptográfica corresponde ao segredo que permite solucionar a construção/problema no qual as primitivas criptográficas são baseadas de modo fácil e eficiente. Ou seja, se uma parte conhecer a chave criptográfica, ele pode decifrar o texto cifrado facilmente, caso contrário, torna-se inviável o deciframento.

Os protocolos criptográficos são construídos a partir das primitivas criptográficas, consistindo assim numa coleção de programas (equivalentemente, algoritmos, *softwares*), para cada parte envolvida na comunicação (BELLARE; ROGAWAY, 2005). No cenário apresentado na Figura 2.3, deve-se ter algum programa executando no emissor e outro no destinatário. O programa do emissor diz como a mensagem deve ser encapsulada, ou empacotada (texto cifrado), para que seja transmitida de forma segura pelo canal inseguro. No destinatário o programa em execução informa como desencapsular, ou desempacotar, a mensagem a fim de recuperar o seu conteúdo original (texto claro), e, possivelmente agrega a informação associada quanto à questão de identificação e integridade da mensagem (se a mensagem foi mesma enviada pelo emissor que o destinatário espera e se não foi alterada). Ambos programas são funções de chaves criptográficas, as quais correspondem a um segredo compartilhado entre as partes envolvidas na comunicação. O intuito do uso de uma chave criptográfica é garantir que a comunicação seja segura entre as partes que conhecem o segredo, evitando que adversários possam compreender ou influenciar nas mensagens trocadas.

No processo de construção/elaboração de protocolos ou esquemas de criptografia surge uma questão relevante: como garantir que esses protocolos ou esquemas criptográficos sejam efetivamente seguros? Um dos pioneiros a responder essa questão foi Claude Shannon com a definição de "*Perfect Secrecy*" (SHANNON, 1949). Shannon diz que para duas mensagens M_1 e M_2 quaisquer, e, um texto cifrado C, um esquema de criptografia é perfeitamente seguro se tanto na cifragem de M_1 com a chave K, como na cifragem de M_2 com a chave K, tem-se a mesma probabilidade de obter C. A idéia da definição "*Perfect Secrecy*" é que um esquema de criptografia é perfeitamente seguro se a mensagem cifrada não fornece nenhuma informação sobre a mensagem original sem o conhecimento da chave. Ou seja, um esquema de criptografia é considerado perfeitamente seguro se o texto cifrado não revela absolutamente nada sobre o texto claro. Essa definição é bastante poderosa e aplicável para determinar a segurança de esquemas de criptografia, além de garantir que um esquema é seguro independente do poder computacional do adversário. Entretanto, a noção de segurança perfeita só é aplicável nas situações onde a mensagem, a chave e o texto cifrado tem o mesmo tamanho, o que na prática não é viável, já que é usualmente as chaves são bem menores que o tamanho das mensagens.

Do ponto de vista prático é utilizado a idéia de segurança computacional para atestar a segurança de um esquema de criptografia. A abordagem computacional relaxa a noção de segurança perfeita, a qual considera que os esquemas de criptografia modernos podem ser "quebrados" dado tempo e poder computacional suficientes. Entretanto, a segurança da abordagem computacional se baseia no seguinte aspecto: a quantidade de cálculos necessários para quebrar a segurança desses esquemas levaria um tempo suficientemente grande (por exemplo, mais de 100 anos), mesmo se fosse realizado em supercomputadores, de tal forma que o sucesso de um adversário em quebrar o esquema criptográfico tem uma probabilidade muito pequena (KATZ; LINDELL, 2007). Em outras palavras, os ataques são inviáveis, mas possíveis. A segurança é proporcionada por problemas que são difíceis de serem solucionados computacionalmente, como por exemplo a fatoração de um número ou a solução de um logaritmo discreto. O estudo de problemas computacionalmente difíceis são objeto de estudo da teoria de complexidade na ciência da computação.

Conforme já fora mencionado, os protocolos criptográficos são compostos pelas primitivas criptográficas. As primitivas criptográficas são algoritmos ou protocolos de criptografia bastante simples, que atuam como "peças" para a elaboração de um protocolo de criptografia mais complexo. Os algoritmos AES (*Advanced Encryption Standard*) e

RSA são exemplos de primitivas criptográficas.

2.2.1 Criptografia Simétrica

Criptografia simétrica se refere aos algoritmos que utilizam a mesma chave criptográfica para cifrar e decifrar uma mensagem. Corresponde a uma das formas mais simples e usuais para fornecer segurança às partes envolvidas na comunicação. A figura 2.4 apresenta um esquema simplificado de um esquema de criptografia simétrica, onde *Alice* envia uma mensagem à *Bob*, a qual é cifrada por *Alice* com uma chave secreta ao enviá-la, e, decifrada por *Bob* com a mesma chave secreta ao receber a mensagem cifrada. Como requisitos para segurança da criptografia simétrica é necessário que o algoritmo de criptografia seja forte, o que quer dizer que o adversário é incapaz de decifrar o texto cifrado ou descobrir a chave ainda que tenha à sua disposição diversos textos cifrados e o texto claro correspondente, e, que a chave secreta compartilhada entre o emissor e o destinatário seja mantida protegida (STALLINGS, 2008).

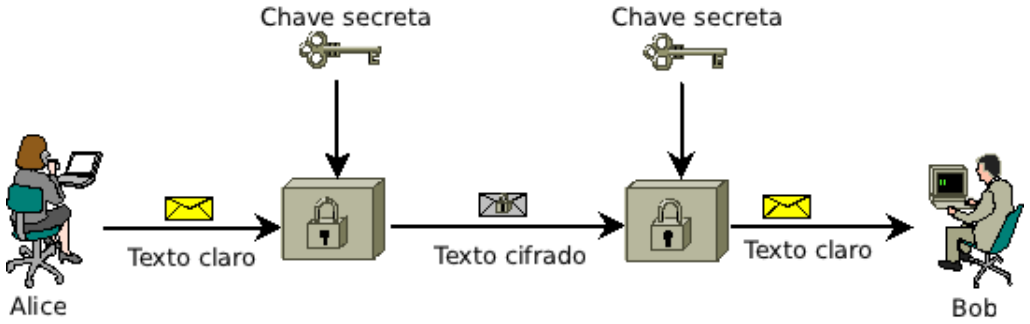


Figura 2.4: Esquema simplicado da criptografia simétrica.

Algoritmos de criptografia simétrica podem ser divididos em duas categorias: algoritmos de fluxo ou cifras de fluxo e, algoritmos de bloco ou cifras de bloco (STALLINGS, 2008). A característica de uma cifra de fluxo é que opera num único *bit* (ou algumas vezes *byte*) de texto claro por vez. Já a cifra de bloco opera em grupos de *bits* chamados blocos, onde um bloco tipicamente pode ter o tamanho de 64 ou 128 *bits*. Utilizando alguns dos modos de operação, *Cipher Feedback* (*CFB*) ou *Output Feedback* (*OFB*), uma cifra de bloco pode ser usada para conseguir o mesmo efeito de uma cifra de fluxo. Comparando as duas categorias, é necessário muito mais esforço para analisar uma cifra de bloco do que uma cifra de fluxo. Porém, a cifra de bloco pode ser empregada a um número maior de aplicações que a cifra de fluxo. Como exemplo de algoritmos que utilizam cifra de bloco, pode-se citar o *Data Encryption Standard* (*DES*) e o *Advanced Encryption Standard* (*AES*).

A criptografia simétrica permite alcançar confidencialidade dos dados e também a

autenticação da mensagem. A autenticação é alcançada com o uso da técnica de autenticação de mensagens (*MAC - Message Authentication Code*). Uma função *MAC* resulta num valor aparentemente aleatório, de tamanho fixo, que serve como autenticador, fruto do conteúdo da mensagem que se deseja autenticar e de uma chave secreta. De fato, o *MAC* é uma técnica que combina criptografia simétrica (através da chave secreta) com funções *hash*. O *Hashed-based Message Authentication Code* (*HMAC*) é uma técnica bastante difundida que utiliza o conceito do *MAC*.

Em geral, a criptografia simétrica é bastante rápida e apresenta um bom desempenho, o que é devido à sua base nas ferramentas elementares de substituição e permutação. Tal característica permite fácil implementação de criptografia simétrica em *hardware* e em *software*.

2.2.2 Criptografia Assimétrica

A criptografia assimétrica, também conhecida como criptografia de chave pública, é considerada a maior e talvez a única verdadeira revolução na história da criptografia, diferenciando de tudo que havia sido feito nessa área (STALLINGS, 2008). Uma das mudanças é que os algoritmos de chave pública são baseados em funções matemáticas, em vez das ferramentas de substituição e permutação. Mas a mudança mais radical é que a criptografia assimétrica envolve o uso de duas chaves separadas, diferentemente da criptografia simétrica, que usa apenas uma chave. Além disso, a criptografia de chave pública requer dois algoritmos: um para cifrar (com uma chave) e outro para decifrar (com a outra chave), contrastando a criptografia simétrica que utiliza um mesmo algoritmo para cifrar e decifrar com a mesma chave.

Para o funcionamento da criptografia de chave pública, cada parte da comunicação deve gerar o seu par de chaves. Uma das chaves é mantida secreta pela parte que a gerou, sendo chamada de chave privada. A outra chave é disponibilizada ou enviada às outras partes de comunicação, o que caracteriza sua denominação de chave pública. A chave pública é utilizada para cifrar as mensagens, de forma que o texto cifrado só poderá ser decifrado pela chave privada correspondente, garantindo assim a confidencialidade dos dados. Por outro lado, o uso da chave privada para cifrar uma mensagem serve como uma assinatura digital, uma vez que somente o detentor da chave privada poderia ter preparado a mensagem. Assim, numa situação onde o emissor cifra uma mensagem com a sua chave privada, o destinatário decifra utilizando a chave pública e sabe que a mensagem foi realmente enviada pelo emissor. Além disso, o destinatário tem certeza de que a mensagem não foi alterada, pois não há a possibilidade de alterá-la sem o

acesso à chave privada do emissor. Logo, a mensagem é autenticada tanto em termos da origem quanto da integridade dos dados.

A chave privada deve ser mantida secreta para que não haja comprometimento do esquema de criptografia de chave pública. Ainda pensando na segurança da criptografia assimétrica, deve-se ter a premissa de que o conhecimento da chave pública não deve comprometer a chave privada, ou seja, é inviável determinar a chave privada dado a chave pública. Tal premissa é alcançada em virtude da geração das chaves ser baseada em funções matemáticas que garantem a segurança do esquema de criptografia assimétrica. Essas funções matemáticas se baseiam em problemas considerados intratáveis, como a fatoração do produto de números primos grandes ou o cálculo de logaritmo discreto, de forma que a dificuldade de encontrar alguma solução eficiente para tais funções garante que o esquema de criptografia é difícil de ser "quebrado". Quanto ao desempenho, a criptografia de chave pública tende a ser mais lenta, sobretudo devido ao *overhead* computacional que acarreta.

2.2.3 Funções Hash

Função *hash* é uma função determinística que relaciona uma mensagem de qualquer tamanho a uma cadeia de caracteres com aspecto aleatório, chamado de valor de *hash* (STALLINGS, 2008). Esse valor de *hash* pode ser utilizado como uma espécie de autenticação de uma mensagem, já que para cada mensagem x o valor de *hash* resultante da função *hash* $f(.)$ deve ser "único". De forma mais precisa, a "unicidade" da função *hash* é entendida como resistência a colisão. Colisão é o termo utilizado quando aplica-se a função *hash* a duas entradas distintas, x e y, resultando no mesmo valor de *hash*. Logo, para ter-se uma função *hash* é necessário que esta seja resistente a colisão, ou seja, é difícil encontrar outra mensagem y que resulte no mesmo valor de *hash* para a mensagem x.

Uma função *hash* deve ter as seguintes prerrogativas:

- Deve ser fácil de calcular para qualquer mensagem x, tornando viável as implementações de *hardware* e *software*;
- Deve ser difícil de inverter, ou seja, dada um resultado de *hash* $f(x)$ é inviável calcular a sua função inversa $f^{-1}(x)$ e encontrar a mensagem x;
- Deve ser inviável encontrar alguma mensagem x que seja igual à um valor de hash $f(x)$ já calculado (propriedade unidirecional);

- Deve ser inviável, dado uma mensagem x, encontrar uma mensagem $y \neq x$ tal que os seus resultados após usar uma função *hash* $f(.)$ sejam iguais, ou seja, $f(x) = f(y)$ (resistência fraca a colisões);
- Deve ser inviável encontrar duas mensagems x e y quaisquer tal que os seus resultados após usar uma função *hash* sejam iguais (resistência forte a colisões).

Do ponto de vista prático, uma função de *hash* não fornece confidencialidade de mensagens, mas fornece uma espécie de "impressão digital" (SCHNEIER, 1996). Isso se deve à sua característica de tomar mensagens de qualquer tamanho e entregar como resultado um valor fixo com aspecto aleatório. Por exemplo, pode-se utilizar uma função *hash* para ser a "impressão digital" de arquivos: para verificar a existência de determinados arquivos entre usuários na *Internet*, bastaria que os usuários enviassem um ao outro os valores *hash* dos arquivos que possuem e consultassem esse valores para determinar a existência do arquivo desejado, evitando assim que os arquivos fossem enviados integralmente uns aos outros.

Uma função *hash* é pública, ou seja, não há segredo no seu processo. A segurança das funções *hash* é baseada na sua natureza de não inversibilidade (unidirecional) e na resistência a colisões. O *Secure Hash Algorithm* (SHA) e *Message-Digest Algorithm 5* (MD5) são algoritmos de *hash* bastante conhecidos.

2.2.4 Assinatura Digital

Uma assinatura digital é um mecanismo de autenticação que utiliza a criptografia de chave pública, e, permite ao criador de uma mensagem anexar um código que atue como uma assinatura. A assinatura é formada tomando o *hash* da mensagem e cifrando-a com a chave privada do criador. Dessa forma, a assinatura garante a autenticação da origem e a integridade da mensagem (STALLINGS, 2008).

Tanto a assinatura digital como as técnicas de autenticação de mensagem (MAC) podem ser utilizadas para assegurar a integridade (ou autenticidade) das mensagens transmitidas. Além da diferença de que a assinatura digital utiliza criptografia de chave pública e um algoritmo MAC utiliza criptografia simétrica, há algumas vantagens qualitativas no uso da assinatura digital (KATZ; LINDELL, 2007):

- verificação pública - significa que se o destinatário verifica a assinatura de uma

determinada mensagem como legítima, então é garantido que todas as outras partes que receberam a mensagem assinada a verificarão também como legítima;

- transferência - uma assinatura de uma mensagem pode ser mostrada para uma terceira parte, a qual pode atestar a legitimidade da assinatura;
- não-repúdio - uma vez o assinante disponibiliza sua chave pública, ele não pode negar a assinatura de uma mensagem, uma vez que só o assinante possui a chave privada correspondente.

2.3 AMEAÇAS

Um computador conectado na *Internet* está sujeito a uma grande variedade de ameaças. Será apresentado nesse tópico algumas das principais ameaças, descrevendo-as brevemente e apresentando as contramedidas existentes na literatura.

2.3.1 Scanners

Uma das primeiras técnicas utilizadas por usuários mal-intencionados é coletar informações para identificar prováveis vítimas em função de vulnerabilidades conhecidas. Para a coleta dessas informações, geralmente são utilizados *scanners*. *Scanners* são programas utilizados para varrer computadores em uma rede em busca de vulnerabilidades (ULBRICH; VALLE, 2004). Os *scanners* buscam sistemas que estejam desprotegidos e prontos para receber uma análise minuciosa sobre sua segurança.

Os *scanners* podem ser divididos em duas categorias:

- *Port scanner* - são *scanners* que realizam a varredura das portas *TCP/UDP*, onde o usuário mal-intencionado pode constatar quais serviços estão funcionando em um determinado computador e até determinar a versão do serviço em funcionamento. Há várias formas de varredura com um *port scanner* (FYODOR, 1997), como por exemplo, varredura com o envio de pacotes *TCP SYN*, varredura com o envio de pacotes *TCP FIN*, varredura com fragmentação de pacotes, varredura com pacotes *UDP* observando as respostas *ICMP* de portas inalcançáveis, etc. A ferramenta *nmap* é bastante utilizada para realizar varreduras;
- *Scanner* de vulnerabilidade conhecida - são *scanners* que realizam tarefas complementares ao *port scanner*, buscando as vulnerabilidades dos serviços detectados

como ativos. Ou seja, uma vez determinado os serviços que funcionam num determinado computador, e, através de uma lista de falhas conhecidas, esse tipo de *scanner* verifica se o computador executa ou não algum serviço com problemas (ULBRICH; VALLE, 2004). Entre as vulnerabilidades conhecidas, podem ser verificado erros de configuração, utilização de senhas padrão, combinações óbvias de usuários e senhas e falhas de segurança divulgadas. Como exemplo de um *scanner* de vulnerabilidade conhecida, pode-se citar o *nessus* (ULBRICH; VALLE, 2004).

2.3.2 0-Day Exploits

Alguns ataques exploram vulnerabilidades em programas ou sistemas que ainda são desconhecidas ou não foram solucionadas pelos seus desenvolvedores. Esses ataques são chamados de ataques de zero dia, ou do termo original em inglês: *0-day attacks*. *0-Day exploits*[2] é o termo designado para referenciar *exploits* provenientes de *0-day attacks*.

Vulnerabilidades geralmente decorrem de falhas de implementação no *software*, sendo posteriormente corrigidas através de um pacote de correção (*patch*) disponibilizada pelo fabricante. Quando uma pessoa detecta uma vulnerabilidade num *software*, ela geralmente comunica à companhia dona do *software*, a qual toma as providências necessárias para elaborar uma correção. No entanto, usuários maliciosos podem utilizar tal descoberta como um trunfo, mantendo a falha/vulnerabilidade secreta ou mesmo compartilhando com a comunidade *hacker*, e, explora as vulnerabilidades desenvolvendo um novo *exploit* (*0-day exploits*). Ainda que o comunicado seja feito à companhia de *software*, haverá um corrida entre desenvolvimento da correção e a busca de um *exploit* para explorar a vulnerabilidade.

Proteger um sistema de *0-day exploits* é uma tarefa complicada, já que nem sempre as falhas dos *softwares* são conhecidas para que se tome alguma providência. Algumas medidas ajudam a minimizar o risco de *0-day exploits*, como a implantação de sistemas de detecção de intrusão, ou o controle de acesso à rede (ENGELKE, 2004). Porém, trata-se de medidas paliativas, uma vez que não evita que algum usuário mal intencionado busque alternativas para explorar uma nova vulnerabilidade.

[2] *Exploits* são programas que atuam em outros programas ou sistemas de forma habilidosa, com o objetivo de levá-los a fazer o que você quer, ainda que os programas/sistemas foram projetados para prevenir tal ação (ERICKSON, 2008).

2.3.3 Malware

Malware é um programa ou *software* com código malicioso que tem um propósito prejudicial (STALLINGS, 2008). Esse propósito prejudicial pode ser desde alterar o comportamento de funcionamento do teclado, roubo de informações e até o controle total do sistema infectado. Um *Malware* pode ser enviado por *SPAM*, ou se replicar como um *SPAM*; pode se aproveitar de falhas de *software*; e pode ser utilizado para montar um ataque de *Denial of Service* (DoS) (AYCOCK, 2006).

Um *Malware* pode ser diferenciado quanto às suas características, como os que precisam de um programa hospedeiro e os que são independentes; os que se replicam e os que não se replicam. A seguir descreve-se alguns dos principais tipos de *Malware* (STALLINGS, 2008; AYCOCK, 2006):

- Vírus - *Malware* que, quando executado, tenta se replicar para outro programa. Uma replicação bem sucedida é chamada de infecção. Tem a característica de depender de um programa hospedeiro para se propagar;
- *Worm* ou verme - é um *Malware* que tem a capacidade de se auto replicar automaticamente, enviando cópias de si mesmo de computador para computador. Embora tenha várias semelhanças com um vírus, diferencia-se por não precisar de um programa hospedeiro. Para se propagar explora vulnerabilidades existentes ou falhas na configuração de *softwares* instalados em computadores;
- *Trojan horse* ou cavalo de tróia - é um *Malware* que aparentemente tem um propósito benigno, mas secretamente executa algo malicioso;
- Bomba Lógica - é um *Malware* que dispara uma ação quando ocorre uma determinada condição. Pode ser inserido num código existente ou pode ser um programa independente;
- *Backdoor* - *Malware* que permite a um invasor acessar o computador sem que este passe pela verificação normal de segurança. Pode ser alojado dentro de código legítimo ou em um programa independente;
- *Keylogger* - *Malware* que tem a capacidade de capturar as teclas digitadas num computador, geralmente com o objetivo de roubar uma informação importante, como a senha de acesso de um banco;

- *Spyware* - é um *Malware* que coleta informações do computador infectado e as transmiste a terceiros. Geralmente é utilizado para roubar informações importantes, como o *keylogger*;
- Zumbi - *Malware* alojado em computadores que são ativados para desferir ataques à outras máquinas.
- *Exploit* - é um *Malware* específico para uma única vulnerabilidade ou conjunto de vulnerabilidades.

A principal mecanismo para proteger um computador de *Malware* é a prevenção. O ideal é impedir entrada de vírus no sistema, procurando instalar programas somente de fontes confiáveis, ter cuidado com executáveis enviados por *email*, ter cautela ao navegar na *Internet*, manter o sistema operacional e os programas atualizados. Embora a prevenção auxilie na redução dos riscos de infecção de uma máquina, nem sempre é garantido que um computador esteja livre dessa ameaça. Outro mecanismo utilizado para proteger um computador de *Malware* é o uso de um programa de antivírus. O uso de antivírus combate o *Malware*, eliminando os programas maliciosos existentes no computador, e também auxilia no trabalho de prevenção, identificando a ação de *Malwares* quando estes iniciam a infecção ou sua instalação no sistema. Basicamente um antivírus detecta, identifica e remove um *Malware*, desde que este seja conhecido e esteja no seu banco de dados. Mas como todos os dias há novos *Malwares*, um antivírus não resolve por completo o problema do *Malware*.

2.3.4 Ataque de negação de serviço

Um ataque de negação de serviço, do termo em inglês *Denial of Service* (*DoS*), é uma tentativa de impedir que usuários legítimos de um serviço usem esse serviço (STALLINGS, 2008). É um ataque onde o alvo é bombardeado com uma enorme quantidade de dados, comprometendo os seus recursos, como memória, processamento e largura de banda. Ataques provenientes de um único computador são chamados de *DoS*, mas quando são vários computadores atacando ao mesmo tempo, o ataque é denominado de *Distributed Denial of Service* (*DDoS*).

Normalmente um ataque *DDoS* é um ataque coordenado, onde um atacante obtém o controle de vários computadores e envia um comando para que o ataque seja feito ao alvo. Para conseguir recrutar vários computadores sob o seu controle, o atacante pode

disseminar um *Malware* que os infecta deixando-os como zumbis, ou pode alugar um *botnet*[3] no mercado negro.

Um ataque *DoS* é mais fácil de ser remediado, já que uma vez detectado o computador de origem do ataque, pode-se bloquea-lo. Defender-se de um *DDoS* é mais complicado, pois são várias fontes, tornando-se o bloqueio uma tarefa árdua e até inviável. Porém algumas medidas podem ser tomadas para mitigar ataques *DDoS*:

- Prevenção de ataque e ação antecipada (antes do ataque) - correspondem a imposição de políticas de consumo de recursos e fornecimento de recursos reserva disponíveis de acordo com a necessidade. Além disso, os mecanismos de prevenção modificam sistemas e protocolos na *Internet* para reduzir a possibilidade de ataques DDoS;
- Detecção e filtragem do ataque (durante o ataque)- visa detectar o ataque logo no início, buscando padrões de comportamento suspeitos, e, responder imediatamente filtrando os pacotes que provavelmente fazem parte do ataque;
- Rastreamento e identificação da origem do ataque (durante e após o ataque) - se baseia na tentativa de identificar a origem do ataque para prevenir ataques futuros. Esse método normalmente não gera resultados eficientes para mitigar um ataque em andamento.

2.4 SUMÁRIO

Neste capítulo foram apresentados alguns conceitos teóricos importantes para a compreensão deste trabalho, mostrando definições básicas de uma rede de computadores, conceitos de criptografia e as ameaças que podem comprometer a segurança de um computador, um servidor ou de um sistema. De forma geral, a proteção de um sistema contra as ameaças vistas (*Malware*, *0-day attacks*, *DDoS*) é uma tarefa complicada e dispendiosa. Uma medida de proteção é a autenticação dos usuários a um sistema, de modo a restringir o acesso somente para os usuários autorizados. Com tal intuito são destacadas as técnicas de autenticação baseadas em camada de rede. O próximo capítulo apresenta as principais técnicas de autenticação baseadas em camada de rede, mostrando suas respectivas características e conceitos que serão utilizados para o desenvolvimento deste trabalho.

[3] *Botnet* corresponde a um conjunto de computadores infectados (zumbis) conectados na *Internet*, usualmente utilizado para fins maliciosos, como envio de *SPAM* ou ataques *DoS*.

3 MÉTODOS DE AUTENTICAÇÃO NA CAMADA DE REDE

A realização de autenticação na camada de rede não é uma técnica muito disseminada. Em geral, a autenticação é deixada como encargo da camada de aplicação, surgindo serviços específicos para desempenhar tal tarefa, como o *kerberos*, serviço de autenticação *X.509* e infra-estrutura de chave pública (*PKI - Public Key Infrastructure*). Entretanto, conforme já mencionado, deixar somente a camada de aplicação responsável pela autenticação pode comprometer a segurança de um sistema, na medida que possibilita a realização de varreduras e exploração de vulnerabilidades. Mesmo os protocolos de segurança da *Internet*, como o *Transport Layer Security* (*TSL*) e *Security Sockets Layer* (*SSL*), que atuam na camada de transporte não resolve a questão de exposição de um sistema. A abordagem de autenticação na camada de rede é uma alternativa para evitar esse tipo de comprometimento, na medida que não deixa os serviços em execução num computador expostos, evitando assim que o atacante possa obter informações e utilizar em benefício próprio. Motivados por essas idéias, será apresentado neste capítulo os principais métodos de autenticação na camada de rede.

3.1 VIRTUAL PRIVATE NETWORKS

Um dos métodos mais difundidos que realiza uma espécie de autenticação baseada em camada de rede é a implantação de uma rede privada virtual, ou *Virtual Private Network* (*VPN*) (SNADER, 2005), como é originalmente conhecida, entre o computador do cliente e o servidor executando a aplicação desejada. Uma *VPN* pode ser configurada para que todo o tráfego trocado entre um cliente e um servidor seja cifrado e autenticado, assegurando privacidade e autenticação de toda comunicação por meio da criação um canal privado que fica ativo durante toda a sessão estabelecida pelo usuário.

Essa abordagem é implementada em alguns padrões de segurança de redes. Basicamente pode-se estabelecer uma VPN segura utilizando *SSL* (que atua na camada de transporte) ou com o *IP Security Protocol* (*IPSec*) (KENT; SEO, 2005; SNADER, 2005; STALLINGS, 2008). O *IPSec* proporciona uma comunicação segura através de tunelamento, criptografia e autenticação. Tem dois modos de de uso: modo trans-

porte e modo túnel. O modo transporte protege apenas o conteúdo do pacote *IP*, pois o intuito é dar segurança ao conteúdo que é transportado. O modo túnel resguarda todo o pacote, incluindo o cabeçalho *IP*. O *IPSec* tem também dois protocolos para oferecer segurança: um protocolo de autenticação que fornece autenticação e integridade dos dados (não provê confidencialidade) designado por cabeçalho de autenticação (*AH - Authentication Header*); e um protocolo que fornece autenticação, integridade e confidencialidade dos dados, chamado de encapsulamento de segurança do conteúdo do pacote (*ESP - Encapsulating Security Payload*). O *IPSec* permite o uso de vários algoritmos de criptografia para cifrar mensagens, como o *Data Encryption Standard* (*DES*), *3DES* e *International Data Encryption Algorithm* (IDEA). Do mesmo modo, no *IPSec* pode-se utilizar várias primitivas criptográficas para alcançar a integridade e autenticidade dos dados, como o *HMAC*, *MD5* e *SHA-1*.

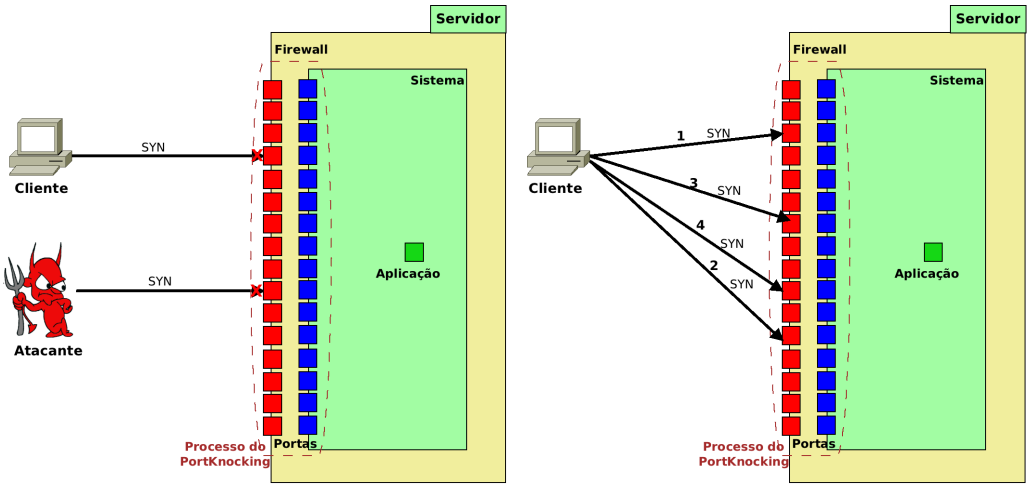
Embora o *IPSec* forneça segurança na comunicação, o mesmo apresenta mecanismo de gerenciamento, implementação e configuração relativamente complexo. Outro aspecto importante é que essa técnica acarreta um *overhead* considerável sobre o tráfego de rede regular, o que é desnecessário para uma simples autenticação de usuário. Além disso, devido ao grande número de protocolos criptográficos envolvidos, muitas fraquezas e vulnerabilidades afetam soluções utilizando IPSec, como é o caso de VPN seguras (FERGUSON; SCHNEIER, 2000).

3.2 PORTKNOCKING

PortKnocking é uma abordagem alternativa para autenticação na camada de rede proposta em Barhman et al. (2002). O nome "*PortKnocking*" foi batizado em sua primeira implementação por Martin Krzywinski (KRZYWINSKI, 2003). Na solução *PortKnocking*, todas as portas TCP/UDP de um determinado computador protegido são mantidas fechadas por um *Firewall* até que o usuário que deseja se conectar ao computador se identifique. Por conveniência, designa-se como cliente os usuários que desejam se conectar a um computador protegido, e, como servidor o próprio computador protegido. Clientes provam sua identidade pelo envio de uma sequência de pacotes de acordo com uma sequência de portas fechadas pré-estabelecida, chamada de sequência de *knock*. O servidor de autenticação monitora o tráfego recebido pelas portas fechadas sem que manifeste alguma resposta, mas abre a porta desejada para o computador de um cliente específico após receber a sequência de portas correta. A idéia é similar a um código de batida em portas, onde a pessoa que quer entrar deve bater na porta um determinado número de vezes conforme um padrão pré-estabelecido.

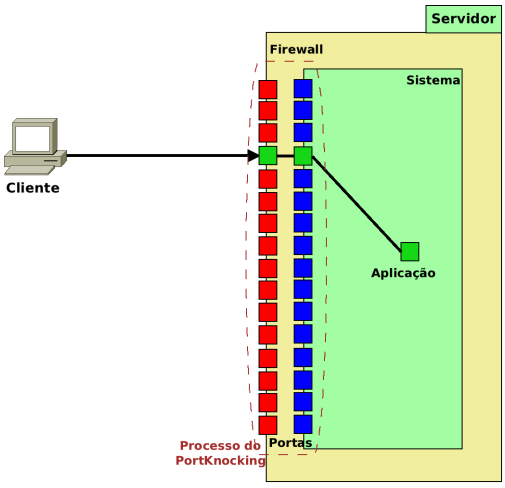
Ao se reconhecer o padrão da batida, a porta se abre.

A Figura 3.1 apresenta o funcionamento do *PortKnocking*. Inicialmente as portas estão todas fechadas pelo *Firewall*. O cliente inicia o processo de autenticação enviando vários pacotes em diferentes portas, conforme a sequência de portas pré-estabelecida (sequência de *knock*). Um processo no servidor (*daemon* do *portknocking*) monitora o tráfego recebido nas portas fechadas. Se o processo do servidor verificar que um cliente enviou a sequência de portas correta, então é disparado um comando para o *Firewall*, abrindo a porta do serviço desejado pelo cliente. Assim, o cliente pode se conectar ao servidor no serviço desejado. A porta pode ser fechada no *Firewall* com a realização do mesmo processo ou com a mecanismos que detectam ociosidade na conexão.



(a) Portas fechadas no servidor.

(b) Processo de Autenticação.



(c) Conexão do cliente ao serviço desejado.

Figura 3.1: *PortKnocking* Básico.

No *PortKnocking* a informação do serviço desejado pelo cliente é conhecida, pois é incluída na sequência de portas de acordo com a porta da aplicação solicitada. De fato,

a sequência de portas é construída de acordo com informações compartilhadas entre o cliente e o servidor. Na implantação do *PortKnocking*, cliente e servidor usualmente compartilham as seguintes informações:

- Cliente - endereço *IP* do cliente (Exemplo: 200.11.10.182);
- *Port* - porta a ser solicitada para abertura (Exemplo: 22 - *SSH*);
- *Timestamp* - data/hora atual do cliente;
- *Flags* - campo genérico, usualmente representado por um número, utilizado para comunicação entre o cliente e servidor. Pode ser utilizado para controle de número de acessos ou como uma forma de *One Time Password* (*OTP*);
- *Checksum* - campo para verificação da formação da sequência de portas. Na versão de Martin Krzywinski é utilizado um *checksum* muito simples, dado pela soma de todos os dados compartilhados módulo 256.

As informações compartilhadas são codificadas e mapeadas em portas *TCP* ou *UDP* para formar a sequência de portas. Na fase de codificação as informações compartilhadas são concatenadas numa cadeia de caracteres, convertida em binário, dividida em *N bits* e transformada novamente em decimal. O valor de *N bits* na codificação é calculado de acordo com a quantidade de portas alocadas para o compor a sequência de portas[4]. Na fase de mapeamento, os valores decimais codificados são mapeados para portas pertencentes à faixa de portas configuradas para compor a sequência de portas. Por exemplo, suponha que foi definido inicialmente que a faixa de portas disponíveis para o *PortKnocking* criar uma sequência de portas seja 8096-65335 e que um dos valores decimais codificados seja 2: a porta mapeada para compor a sequência de portas será a porta 8097 (a segunda porta da faixa de portas definida).

Uma característica do *PortKnocking* a ser observada é que cada pacote de uma sequência de portas representa o envio de uma pequena parte de autenticação, ou seja, o envio de dados de autenticação tem um meio com tamanho limitado. Considerando toda a faixa de portas *TCP/UDP* para fins de autenticação do *PortKnocking* (65535 portas), tem-se apenas 16 *bits* em cada pacote *IP* para transmitir dados de autenticação. Geralmente a quantidade de *bits* para transmitir os dados de autenticação é ainda menor

[4]Na implementação básica do *PortKnocking* é permitido a definição das faixas de portas que podem ser utilizadas para a sequência de portas.

que 16 *bits*, já que é permitido a configuração da faixa que pode ser utilizada como sequência de portas.

O *PortKnocking* básico apresentado proporciona características de segurança interessantes, tais como o ocultamento das portas, e consequentemente dos serviços do servidor; segurança contra *0-day attacks*; e simplicidade no funcionamento e implementação. Mesmo que um atacante faça uma varredura contra o servidor, não obterá sucesso em conseguir informação útil sobre os serviços em execução ou detalhes do sistema operacional, já que não receberá nenhuma resposta do computador "fechado". Entretanto, o *PortKnocking* é claramente falho, podendo ser facilmente contornado por um atacante que tenha a habilidade de monitorar o tráfego de rede e obter a sequência correta de pacotes necessários para autenticação. Nesse caso, basta que o atacante faça um ataque de repetição[5] no servidor de autenticação.

3.2.1 Protótipo Original do PortKnocking

Para estudar melhor o *PortKnocking*, será feito uma análise da versão original feita por Martin Krzywinski (KRZYWINSKI, 2009). Apesar de ser o primeiro protótipo do *PortKnocking*, o estudo dessa versão tem a vantagem de preservar a idéia original de funcionamento do *PortKnocking*, além de permitir algumas opções de execução interessantes. Esse protótipo foi desenvolvido na linguagem *Perl* e possui três modos de funcionamento:

- Modo básico - esse modo de funcionamento é essencialmente o modo básico de *PortKnocking* explicado anteriormente e apresentado na Figura 3.1. A sequência de portas é enviada em texto claro, pois os dados de autenticação são somente codificados e mapeados nas portas, para posterior envio ao servidor. Esse modo tem a desvantagem de ser vulnerável a ataque de repetição, além de possibilitar que o atacante compreenda a regra de formação da sequência de portas e possa construir uma sequência para autenticação de qualquer endereço *IP* de sua escolha;

[5] Ataques de repetição - também conhecido como ataques *replay* são ataques onde o atacante geralmente "personifica" a identidade de um cliente válido e repete as mensagens do protocolo já executado anteriormente pelo mesmo cliente, para o mesmo ou um novo servidor (MENEZES; OORSCHOT; VANSTONE, 2006). "Personificação" ou *spoofing* é uma técnica empregada por um atacante para assumir a identidade de um usuário ou computador (MENEZES; OORSCHOT; VANSTONE, 2006). *MAC spoofing* é uma técnica de personificação onde o atacante altera o seu endereço físico (endereço MAC) para assumir a identidade da vítima (ULBRICH; VALLE, 2004).

- Modo com criptografia - nesse modo os dados de autenticação são primeiramente cifrados com uma chave simétrica compartilhada entre o cliente e o servidor. Os dados cifrados são codificados e mapeadas nas portas, e então enviados para o servidor. Nota-se que este modo se diferencia do modo básico pelo fato de que os dados de autenticação são cifrados num primeiro momento. Em termos práticos, além da segurança dos dados de autenticação proporcionados pela criptografia, há um aumento da quantidade de informação a ser codificada e mapeada para a construção da sequência de portas. A vantagem deste modo é que os dados de autenticação e a regra de formação da sequência de portas é protegida pela criptografia, evitando o conhecimento do processo de criação da sequência de portas. No entanto, ainda é vulnerável a ataques de repetição. Outro aspecto negativo é que a criptografia adiciona um *overhead* considerável, devido ao aumento de dados a serem codificados e mapeados, aumentando o número de pacotes necessários para autenticação;

- Modo *One Time Knock* (*OTK*) - funciona de forma similar ao modo de criptografia, mas adiciona um campo variável nos dados de autenticação (campo *flag*), o qual é incrementado a cada autenticação. Assim, cada sequência de portas é diferente, criando uma espécie de autenticação *OTP*, chamada aqui de *OTK*. Embora tal modo melhore a questão da vulnerabilidade ao ataque de repetição, o *OTK* se baseia no incremento de uma variável, o que é considerado muito simples. Além disso, o incremento do campo *flag* exige que o servidor e o cliente estejam perfeitamente sincronizados, pois ambos incrementam a *flag* de forma independente, de modo que qualquer falha pode acarretar em valores distintos, impactando em futuras autenticações.

Embora tenha três modos de funcionamento, o protótipo *PortKnocking* é bastante simples. A implantação depende apenas da configuração dos dados compartilhados para autenticação no cliente e servidor. Essa tarefa é feita utilizando arquivos de configuração, como um serviço qualquer em ambiente *Unix*. O protótipo *PortKnocking* pode ser instalado em qualquer computador, desde que tenha o compilador *Perl* presente com os módulos necessários para o seu funcionamento. Por padrão, o protótipo foi projetado para interagir com o *Firewall IPTables*, nativo de sistemas Unix, embora tenha a flexibilidade de trabalhar com qualquer outro *Firewall* por meio de alteração do arquivo de configuração do *PortKnocking* no servidor (desde que haja a possibilidade de executar o comando do *Firewall* pelo sistema operacional).

3.2.2 Avaliação do PortKnocking

Tomando o presente protótipo do *PortKnocking* como parâmetro de avaliação, pode-se elencar alguns aspectos positivos para o *PortKnocking*:

1. Fornecimento de uma camada de proteção adicional - o funcionamento do protótipo, bem como o conceito de *PortKnocking*, colabora para que seja adicionado ao sistema mais um nível de proteção. Em sistemas convencionais a segurança é mantida apenas com o uso de *Firewall* e com a autenticação realizada pelas aplicações. Devido às características do *PortKnocking*, a sua utilização incrementa a segurança de sistemas convencionais;

2. Permite autenticação do cliente pelo processo do *PortKnocking* no servidor, antes de permitir acesso a um serviço;

3. Esconde os serviços em execução num servidor, já que todas as portas são mantidas fechadas pelo *Firewall*, e aberta somente após a autenticação do cliente;

4. Pode oferecer uma boa segurança se usada com criptografia e mecanismos anti-repetição, como o modo *OTK*;

5. Inviável comprometer o processo do *PortKnocking* do servidor com a criação de pacotes maliciosos. Como os dados de autenticação são transportados pela porta *TCP/UDP*, seria muito difícil para um atacante escrever algum pacote que possa tentar explorar alguma vulnerabilidade. Uma vez que os dados de autenticação são enviados pela numeração de portas, resta ao atacante o ataque aos protocolos *TCP/UDP*, de modo que a elaboração de um *exploit* que ataque o *PortKnocking* é inviável.

No entanto, há vários pontos não desejáveis no *PortKnocking*. Esses aspectos negativos são apresentados a seguir:

1. Apresenta um mecanismo extremamente ruidoso, devido a necessidade do envio de vários pacotes para fornecer todos os dados de autenticação. Essa característica pode ser confundida como uma varredura de um *port scanner*. Sistemas de detectção de intrusão (*IDS - Intrusion Detection System*) podem comprometer o funcionamento do *PortKnocking*;

2. Só pode ser utilizado com o modo de criptografia habilitado. Do contrário, pode ser facilmente contornado com o uso de um *sniffer*[6]. Porém o uso de criptografia impacta no volume da informação para autenticação, requerendo um número maior de pacotes para o envio dos dados de autenticação (acima de 16 pacotes (KRZYWINSKI, 2003)). Esse aspecto torna o uso de criptografia assimétrica impraticável, em virtude deste tipo de criptografia requerer chaves de maior tamanho, e, consequentemente, em maior quantidade de informação cifrada;

3. O *checksum* empregado é muito simples. A soma dos valores dos dados de autenticação com a operação de módulo 256 só permite verificar que os dados obedecem a uma regra de formação, mas não oferece proteção de integridade. Também não é útil para rastrear sequência de portas já utilizadas, para proteger contra ataques de repetição;

4. O *PortKnocking* não funciona se os pacotes chegarem fora de ordem no servidor. O servidor faz exatamente o processo inverso do cliente (mapeamento, decodificação e decifragem). A chegada de algum pacote fora de ordem acarreta um embaralhamento da informação de autenticação no servidor, o que implica na falha do processo de autenticação. Podem haver várias razões para que um pacote chegue fora de ordem, mas as principais são latência e atraso na rede. Uma possível solução para contornar essa situação é a configuração de um pequeno atraso (*delay*) no envio dos pacotes, o que pode diminuir os efeitos de latência e atraso na rede;

5. O *PortKnocking* não funciona em ambientes com *NAT*. Em ambientes com *NAT*, o endereço *IP* de origem é alterado na borda da rede de origem por um endereço *IP* Público (provavelmente o endereço do roteador de borda). Uma vez que nos dados de autenticação há o endereço *IP* do cliente, o processo *PortKnocking* do servidor verifica que o endereço de origem do pacote é diferente do endereço informado nos dados de autenticação e desqualifica o pedido de autenticação. Para que o *PortKnocking* funcione corretamente, o cliente deve ter um mecanismo para que saiba o seu endereço *IP* Público. Porém, ainda que se implemente tal mecanismo, surge outro problema: a abertura de regra no *Firewall* permite que todos os computadores que compartilham o mesmo endereço *IP* possam acessar o serviço solicitado, o que expõe o servidor;

[6] *Sniffers* são programas que "escutam" o trafego de uma rede (ULBRICH; VALLE, 2004). Um *sniffer* atua na camada de acesso ao meio do modelo *TCP/IP*, e, tem a capacidade de abrir vários protocolos das camadas superiores.

6. O *PortKnocking* não é capaz de detectar perda de pacotes. Se houver algum pacote da sequência de portas perdido na rede, o *PortKnocking* não detecta e falha na autenticação;

7. A situação multiusuário impõe uma certa dificuldade na administração de chaves no *PortKnocking*. Na versão de protótipo é previsto apenas uma chave secreta, de modo que, se houver mais de um usuário, todos compartilham a mesma chave. Tal característica não é segura, pois se um único cliente for corrompido, pode comprometer todo o sistema de proteção do *PortKnocking* no servidor. Supondo que o *PortKnocking* não tivesse essa limitação e estivesse preparado para o ambiente multiusuário, tem-se ainda o problema de gerenciamento, distribuição e configuração das chaves;

8. Ausência de associação entre a autenticação e posterior conexão. Uma vez que um cliente se autentica no processo do *PortKnocking* no servidor, a porta do serviço solicitado é aberta no *Firewall*, de modo que o cliente pode realizar a sua conexão (conexão subsequente). Entretanto, não há nada que "amarre" o processo de autenticação com a conexão subsequente, o que pode ser explorado por um atacante. Uma vez que a autenticação ocorra, um atacante poderia anular o cliente, personificá-lo com alguma técnica de *spoofing*, e, se conectar ao servidor. Outra possibilidade é que um atacante sequestre a conexão criada entre o cliente e o servidor (*hijacking*);

9. É susceptível à ataques *Man-in-the-Middle*[7] - os ataques *Man-in-the-Middle* são factíveis devido ao fato das mensagens não serem autenticadas. Conforme visto, há somente um *checksum* para certificar que as mensagens foram enviadas corretamente, mas não atesta quanto à integridade e autenticidade.

De forma geral, os aspectos negativos sobressaem em relação aos benefícios do *PortKnocking*, o que pode explicar a relutância para o seu uso. Todavia, a idéia empregada no *PortKnocking* foi revelevante, e serviu como base e inspiração para a criação de outras formas de autenticação na camada de rede, como o SPA que será abordado na próxima seção.

[7] Ataque Man-in-the-Middle (MITM) é um tipo de ataque onde o atacante se posiciona entre as partes envolvidas na comunicação, intercepta as mensagens trocadas e as reenvia a cada uma das partes, podendo apenas verificar ou mesmo o conteúdo o conteúdo (STALLINGS, 2008).

3.3 SPA - SINGLE PACKET AUTHENTICATION

Na seção anterior, viu-se o *PortKnocking* como uma alternativa para autenticação na camada de rede, com características interessantes, mas também com muitos aspectos indesejados que comprometem a sua adoção como padrão de segurança. Com o propósito de resolver alguns pontos negativos do *PortKnocking*, surgiu o *Single Packet Authorization* (*SPA*), também proposto no artigo (BARHMAN et al., 2002) e implementado posteriormente por Rash Michael (MICHAEL, 2006b). Tal como o *PortKnocking*, o *SPA* mantém todas as portas de serviço *TCP/UDP* protegidas com o bloqueio das mesmas por intermédio de um *Firewall*, até que um pedido de autenticação válido seja recebido. Devido a tal comportamento, alguns especialistas consideram o *SPA* como uma variação do *PortKnocking* (KRZYWINSKI, 2009). No entanto, o *SPA* se diferencia em relação ao *PortKnocking* no sentido em que todos os dados necessários para autenticação são encapsulados como conteúdo de um pacote *TCP/UDP* (campo *payload*) e enviados num único pacote, ao invés de uma sequência de portas.

O técnica *SPA* é basicamente a implementação do mecanismo 1 ISO/IEC 9798-2 para autenticação. O funcionamento do *SPA* pode ser compreendido pelo esboço apresentado na Figura 3.2. Inicialmente as portas *TCP/UDP* são mantidas fechadas no servidor e um processo do *SPA* (*daemon*) fica monitorando a chegada de pacotes numa porta específica. Quando um cliente desejar se conectar em alguma aplicação do servidor, deve primeiro enviar um pacote com a informação de autenticação, para que o processo do *SPA* verifique se é um cliente válido. Toda informação de autenticação é encapsulada num único pacote *TCP/UDP*, o qual é cifrado e enviado ao servidor. Esse pacote também é conhecido como pacote de autorização, ou do termo original *Authorization Packet* (*AP*). Com a chegada de um *AP*, o processo do *SPA* o decifra, atesta a validade dos dados de autenticação e, em caso positivo, envia um comando ao *Firewall* para abrir a porta do serviço solicitado pelo cliente. A partir de então, o cliente pode realizar a conexão subsequente no serviço desejado.

Após a implementação do *SPA* feita por Rash Michael (MICHAEL, 2006b), batizada de *FWKNOP* (acrônimo para *FireWall KNock OPerator*), surgiram outras implementações que utilizam o mesmo conceito. Porém o *FWKNOP* continua a ser uma das implementações mais representativas que utilizam *SPA* para autenticação na camada de rede, pela gama de funcionalidades adicionadas e pela constante atualização da ferramenta no portal `http://www.cipherdyne.org`. Será estudado um pouco o *FWKNOP* para entender melhor o funcionamento e os detalhes do *SPA*, tal como foi

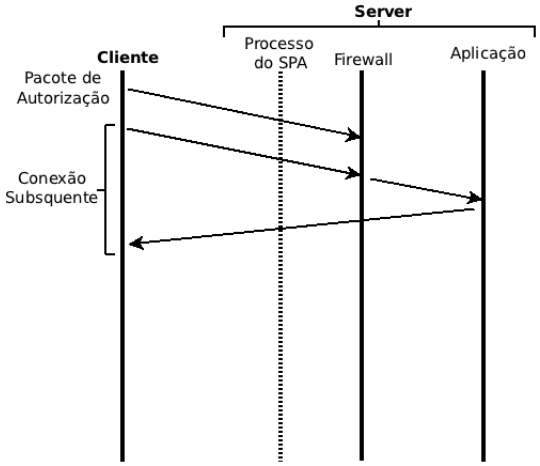


Figura 3.2: Funcionamento do SPA.

feito para o *PortKnocking*.

3.3.1 FWKNOP

O *FWKNOP* é uma ferramenta de autenticação de camada de rede que utiliza a técnica *SPA*. Também pode ser definido como um método de autenticação unilateral usando marcação de data/hora (*timestamps*) e criptografia aplicada ao conceito "autenticação de firewall". Foi desenvolvido inicialmente na linguagem *Perl*, mas a última versão foi implementada na linguagem C. O seu funcionamento básico é idêntico ao esboço apresentado na Figura 3.2 e disponibiliza mecanismos interessantes, como o uso em ambientes NAT, uso de criptografia de chave pública, flexibilidade na escolha do protocolo para envio do *AP* (*TCP*, *UDP*, *ICMP*, *HTTP*, *TCPRAW*).

O *AP* do *FWKNOP* é formado pela composição de uma mensagem a partir da concatenação dos dados de autenticação. Os dados de autenticação que compõem o *AP* são informados a seguir (MICHAEL, 2006b):

- Campo com *bits* aleatórios - são 16 *bytes* aleatórios que tem o propósito de assegurar que cada mensagem *SPA* tenha alta probabilidade de ser única, e assim evitar ataques de repetição;
- *Username* - campo com o nome de usuário, utilizado para distinguir entre vários usuários existentes no sistema, além de permitir criar níveis distintos de permissões;
- *Timestamp* - data e hora local. Este campo também auxilia no combate de ataques de repetição;

- Versão - contém o número de versão do *FWKNOP* para questões de compatibilidade com versões anteriores (muda a formatação da mensagem de acordo com a versão);

- Modo de ação - informa ao servidor se o cliente pretende obter acesso ou apenas enviar um comando para execução;

- Acesso desejado ou *string* de comando - é a porta do serviço desejado pelo cliente, se o modo de ação for acesso ou o comando propriamente dito a ser executado no servidor, se o modo de acão for comando;

- Verificador - é um *hash*, originalmente *MD5*, calculado sobre todos os dados de autenticação. O intuito desse campo é que o servidor possa verificar a integridade do *AP* após decifrá-lo. A versão atual oferece algumas opções para esse campo de verificação, disponibilizando as funções *MD5*, *SHA-1* e *SHA-256*, sendo esta última utilizada como a função *hash* padrão.

Os dados de autenticação são concatenados no *AP*, utilizando como separador o caractere ":", com codificação *base64*[8]. Após a concatenação, toda a mensagem é cifrada utilizando por padrão o algoritmo de criptografia simétrica *AES*, com chaves de até 128 *bits* compartilhada entre o cliente e servidor. Há a possibilidade de utilizar criptografia assimétrica, pois o *FWKNOP* tem como opção o uso do *GNU Privacy Guard* (*GPG*)[9].

Tal como o *PortKnocking*, no *FWKNOP* o processo do *SPA* alojado no servidor monitora o tráfego de rede para verificar pedidos de autenticação através da chegada de algum *AP*. Porém, a monitoração é feita em apenas uma única porta, a qual, por padrão é a 62201. O *FWKNOP* permite que essa porta de comunicação para autenticação seja alterada, o que pode ser desejável para dificultar o trabalho de algum atacante que desconfie do uso dessa ferramenta. Na chegada de um *AP*, o *FWKNOP* decifra a mensagem, verifica sua integridade calculando novamente o campo verificador através da função *hash* correspondente com o campo verificador recebido na mensagem, e se os dados forem válidos é realizada a liberação da porta de acesso no *Firewall* ou a

[8] A codificação base64 é também conhecida como *MIME* (acrônimo de *Multipurpose Internet Mail Extensions*). Trata-se de um esquema de codificação que transforma binários em texto, concebido para a transferência de anexo de emails através da *Internet*. A especificação base64 pode ser encontrada em Linn (1993) e Vaudreuil, Freed & Borenstein (1996).

[9] GPG é um aplicativo de segurança que possui várias primitivas criptográficas para cifrar e assinar dados. É uma alternativas do *software* livre ao *Pretty Good Privacy* (*PGP*) (KOCH, 2007).

execução do comando desejado, de acordo com a solicitação do cliente. Porém, se os dados do *AP* forem invalidados, o servidor não faz nada.

Conforme já mencionado, o *FWKNOP* agrega muitos mecanismos. É apresentado a seguir alguns desses mecanismos:

- Funcionamento de autenticação para clientes em ambientes *NAT* - se o cliente está numa rede que utiliza *NAT*, o *FWKNOP* disponibiliza uma opção para descobrir o *IP* Público e construir o pacote *IP* com este endereço. Ao utilizar esta opção, o *FWKNOP* realiza uma consulta a um *site* externo que retorna ao cliente o seu endereço IP público. Nas versões anteriores do *FWKNOP* o *site* utilizado era o `www.whatismyip.com`, mas a versão atual altera o site de consulta para a *url* `http://www.cipherdyne.org/cgi-bin/myip`[10]. É importante salientar que o acesso ao *site* e o retorno do endereço *IP* público não é cifrado. Outra opção oferecida pelo *FWKNOP* é permitir ao cliente fornecer o nome do computador (*hostname*) para qual o acesso deve ser concedido, disparando uma consulta *DNS* para determinar o *IP* do respectivo nome do computador, e assim construir o pacote *AP*. Ambas opções do *FWKNOP* permitem o funcionamento de um cliente que está numa rede com *NAT*, mas dá margem para um atacante interceptar a mensagem de retorno que contém a informação do endereço *IP* Público e trocar o endereço para um de seu interesse, de forma que o pacote *AP* será montado para abrir acesso ao endereço *IP* escolhido pelo atacante. Há ainda uma outra opção que pode ser utilizada para o funcionamento em redes com *NAT*, onde o cliente informa ao servidor que utilize o endereço *IP* de origem do cabeçalho do pacote *AP* ao invés do endereço *IP* contido no conteúdo do pacote. Essa última opção é bastante insegura, pois basta que um atacante intercepte o envio do pacote *AP* e a reenvie para ganhar acesso ao servidor;

- Funcionamento de autenticação para servidores em ambientes *NAT* - o *FWKNOP* tem um mecanismo para situações onde o servidor possui um endereço *IP* privado. Esse mecanismo permite que o *FWKNOP* possa configurar regras *Destination*

[10]Na verdade, o serviço utilizado é o mesmo das versões anteriores com a consulta sendo realizada ao *site* `www.whatismyip.com`. No acesso ao novo *site* `http://www.cipherdyne.org/cgi-bin/myip`, há provavelmente um redirecionamento do acesso para o `www.whatismyip.com`, onde o domínio da *cipherdyne* atua apenas como um mediador das consultas. Apesar de não haver uma explicação do implementador para tal alteração, pode-se supor que o motivo se deva a passar uma idéia de maior segurança (não acesso a um *site* público) ou centralização de requisições para estatística, já que o *site* intermediador pertence ao próprio desenvolvedor.

Network Address Translation (*DNAT*)[11] no *Firewall* (somente disponível para o *Firewall IpTables*) para que o servidor seja alcançado;

- Habilidade de realizar *OS fingerprinting*[12] - o *FWKNOP* é integrado com a ferramenta *p0f*, a qual realiza *OS fingerprinting* passivo no cliente. Essa característica possibilita que o processo do *FWKNOP* no servidor possa ser configurado para permitir acesso de somente alguns tipos de máquinas, aumentando a segurança do sistema (um atacante pode contornar essa restrição se souber qual tipo de máquina é permitido);

- Integração com a rede *Tor* (MICHAEL, 2006a) - a rede *Tor*, também conhecida como *Onion Router* é uma rede de computadores distribuída que tem o propósito de fornecer meios para uma comunicação anônima na *Internet*. O *FWKNOP* oferece uma opção para utilizar a rede *Tor*, o que garante anonimidade, dificulta o rastreio do pacote originado e a possibilidade de ataques *MITM*. Porém o uso desta opção requer que o processo do *FWKNOP* no servidor execute um *socket TCP* de escuta para o cliente conectar, acarretando ao *FWKNOP* a perda da característica de ocultamento de porta, já que é necessário que a porta de conexão esteja aberta.

Com o desenvolvimento da última versão com a linguagem C, o *FWKNOP* pode ser utilizado em diversos sistemas operacionais, como ambientes *Windows*, *Unix* e até *Android*. Recentemente foi disponibilizado uma versão do *FWKNOP* que pode ser implementada em roteadores caseiros que utilizam o sistema *DD-WRT*[13]. O *FWKNOP* foi projetado para interagir com o *Firewall IpTables*, mas há a possibilidade de utilizar outros tipos de *Firewall* através da execução de comandos do pacote AP.

3.3.2 Avaliação do SPA

De forma geral, o *SPA* alcança os mesmos benefícios para autenticação de um cliente e soluciona algumas questões de segurança quando comparado com o *PortKnocking*. A seguir, será apresentado um resumo com os aspectos positivos de uma implementação *SPA*:

[11] *DNAT*, também conhecido como *port forwarding*, é uma tradução que permite alterar o endereço de destino de um computador antes que sejam roteados para o seu destino final. Essa tradução é útil na situação onde um servidor está num ambiente *NAT* (NETO, 2004).

[12] *OS fingerprinting* é uma técnica empregada em varreduras que possibilita a identificação do sistema operacional do alvo.

[13] O sistema *DD-WRT* é um *firmware* livre baseado no ambiente *Linux* utilizado em roteadores sem fio. Informações adicionais sobre o *DD-WRT* podem ser encontradas no enderço `www.dd-wrt.com`.

- Fornece uma camada de proteção adicional - mesma perspectiva de incremento de segurança obtido com o uso do *PortKnocking*;
- Permite autenticação efetiva no processo do *FWKNOP* no servidor, antes de permitir acesso a um serviço;
- Esconde os serviços em execução num servidor;
- Apresenta maior resistência à ataques de repetição;
- Difícil de ser detectado - como a autenticação é feita com o envio de apenas um único pacote, é menos provável que seja detectado o uso do *SPA*. Mesmo que um atacante monitore os pacotes *AP*, dificilmente assumirá que trata-se de uma atenticação, pois o pacote contém dados aparentemente aleatórios em virtude da criptografia. Uma possibilidade para detecção é a monitoração da porta 62201 nos sistemas que utilizam *FWKNOP*, porém a porta padrão pode ser alterada;
- Menos ruidoso que o *PortKnocking*;
- Permite o uso de criptografia assimétrica;
- Soluciona o problema de chegada de pacotes fora de ordem, uma vez que os dados de autenticação são enviados num único pacote;
- Permite o funcionamento de autenticação em ambientes *NAT*.

Embora o *SPA* se apresente como um avanço na técnica de autenticação na camada de rede, deixa alguns pontos a desejar quanto a segurança, abrindo brechas que podem ser exploradas por um atacante. Esses pontos negativos são apresentados de forma resumida a seguir:

- O funcionamento para clientes em ambientes *NAT* é vulnerável - em particular no *FWKNOP*, os mecanismos que possibilitam a consulta do endereço *IP* Público do cliente dão margem para que um atacante possa interceptar a mensagem com o retorno do endereço ao cliente e alterá-lo para um de sua escolha. Dessa forma, o atacante poderia colocar o próprio endereço *IP*, de forma que ao montar o pacote *AP*, o cliente estaria solicitando a liberação de acesso para o endereço *IP* do atacante. Além disso, a abertura de regra no *Firewall* para um cliente é uma porta aberta para todos os computadores que compartilham o mesmo endereço *IP* Público;

- Não é capaz de detectar perda de pacotes - se o pacote *AP* for perdido, não há como o *FWKNOP* detectar, acarretando a falha na autenticação;

- Problema na situação multiusuário com a utilização de chaves simétricas - apesar da implementação *FWKNOP*, em particular, trabalhar com vários usuários, ainda persiste a dificuldade de administração das chaves;

- Ausência de associação entre a autenticação e posterior conexão - tal como no *PortKnocking*. Um atacante pode aguardar a autenticação de um cliente e posteriormente sequestrar a conexão (*hijacking*). Observe que o sequestro da conexão diferencia de ataque *MITM* em alguns aspectos: no sequestro o atacante deixa o cliente se comunicar normalmente com o servidor e autenticar, e, após a autenticação assume a identidade do cliente, bloqueia o tráfego do cliente, efetuando assim o sequestro da conexão. Em ataques *MITM* o atacante atua como um agente entre o cliente e o servidor repassando as mensagens originais ou modificando-as com algum objetivo específico;

- Implementação pode favorecer criação de pacotes maliciosos - como os dados de autenticação são enviados no conteúdo do pacote *AP*, o processo do *FWKNOP* no servidor deve abrir o conteúdo do pacote *AP* para validar o cliente. Observe que nessa situação não basta a verificação do cabeçalho, como era feito no *PortKnocking*. Um atacante pode explorar essa característica e implementar um *exploit* que ocasione uma falha no processo do *FWKNOP* do servidor e assim obter o acesso desejado;

- Permite ataque *Man-in-the-Middle* - embora a implementação *FWKNOP* tenha algumas mecanismos que ajudam a evitar este tipo de ataque, ainda há a possibilidade de ataques *MITM*. No trabalho (JEANQUIER, 2006), foi citado que o campo *timestamp* do pacote *AP* não é verificado no processo do *FWKNOP* no servidor, o que permite a interceptação e bloqueio de mensagens a um atacante que utilize de persofinicação;

- Não autentica o servidor - na situação de se ter dois servidores com configuração idêntica, há a possibilidade de ataques *interleaving*[14].

De uma forma geral, o *SPA* é mais consistente quando comparado com o *PortKnocking* e garante uma boa segurança para a autenticação. Tal percepção é evidente, uma

[14] Ataque *interleaving* é a repetição de mensagens para fora do protocolo em execução, o que requer dois protocolos executando em paralelo (SYVERSON, 1994).

vez que algumas distribuições *Linux* já incorporam soluções *SPA* (distribuição *Linux BackTrack* (MICHAEL, 2011)). Contudo a disseminação do uso de autenticação na camada de rede ainda é pequena, o que pode estar motivada pelos aspectos negativos do *SPA*. Outro aspecto que corrobora para a não utilização do *SPA* é a ausência de uma prova formal de segurança do protocolo.

3.4 VARIAÇÕES E MELHORIAS PARA O PORTKNOCKING E SPA

O *PortKnocking* e o SPA serviram como base para várias outras propostas de protocolos de autenticação na camada de rede. De fato, muitos autores estudaram as duas técnicas e sugeriram modificações para aperfeiçoar a autenticação e sanar alguns aspectos não desejados.

Nos trabalhos deGraff, Aycock & Jacobson (2005) e deGraff (2007), os autores realizaram um estudo das duas técnicas e propuseram um protocolo para autenticação na camada de rede baseado em protocolos desafio/resposta com o conceito de *PortKnocking*. Foram propostos três variações do protocolo, onde cada variação atende à uma situação diferente, como por exemplo a situação de um cliente num ambiente *NAT* ou autenticação mútua entre o servidor e o cliente. Porém, o funcionamento de cada variação segue o mesmo comportamento de uma comunicação entre cliente e servidor com três mensagens: uma mensagem para envio da requisição de autenticação do cliente para o servidor, uma mensagem para o envio do desafio do servidor ao cliente e uma mensagem com a resposta do desafio do cliente para o servidor. Os protocolos são brevemente apresentados nos algoritmos 3.1, 3.2 e 3.3. O conceito de *PortKnocking* é empregado no envio das mensagens da seguinte forma: as mensagens enviadas pelo cliente contendo os dados de autenticação são codificadas e mapeadas em portas *UDP* de destino (poderia ser utilizada as portas *TCP*, mas foi adotado nesse trabalho o envio em portas *UDP* devido ao menor *overhead*). Assim, há uma restrição de 16 *bits* para cada pacote, o que acarreta o envio de vários pacotes para transmitir os dados de autenticação do cliente ao servidor. Já a mensagens enviadas do servidor para o cliente não precisam seguir essas restrições: o desafio é enviado como um pacote *UDP* simples, direto para a porta de origem do cliente.

Ainda no trabalho deGraff, Aycock & Jacobson (2005) foram sugeridas duas alternativas para contornar o problema de recebimento de pacotes fora de ordem, para as

Protocolo 3.1 Protocolo de Autenticação Unilateral Básico (DEGRAFF; AYCOCK; JACOBSON, 2005)

1: $A \rightarrow B : req$

2: $B \rightarrow A : N_A$

3: $A \rightarrow B : MAC_{K_{req}}(N_A, ID_A, ID_B)$

onde A é o cliente

B é o servidor

req é a requisição para autenticação

N_A é uma cadeia de caracteres aleatória de uso único enviada para A

K_{req} é a chave compartilhada entre A e B

ID_X é a identidade de um computador X (endereço IP)

MAC é a função de autenticação criptográfica

, (uma vírgula) representa concatenação

Protocolo 3.2 Protocolo de Autenticação Unilateral em ambiente NAT (DEGRAFF; AYCOCK; JACOBSON, 2005)

1: $A \rightarrow B : req, ID_A$

2: $B \rightarrow A : PID_A, MAC_{K_{req}}(PID_A, ID_A), N_A$

3: $A \rightarrow B : MAC_{K_{req}}(N_A, PID_A, PID_B)$

onde A é o cliente

B é o servidor

req é a requisição para autenticação

ID_X é o endereço IP privado do computador X

PID_X é o endereço IP público do computador X

N_A é uma cadeia de caracteres aleatória de uso único enviada para A

K_{req} é a chave compartilhada entre A e B

MAC é a função de autenticação criptográfica

, (uma vírgula) representa concatenação

Protocolo 3.3 Protocolo de Autenticação mútua em ambiente *NAT* (DEGRAFF; AYCOCK; JACOBSON, 2005)

1: $A \rightarrow B : req, N_B$

2: $B \rightarrow A : PID_A, MAC_{K_{req}}(N_B, PID_B, PID_A), N_A$

3: $A \rightarrow B : MAC_{K_{req}}(N_A, PID_A, PID_B)$

onde A é o cliente

B é o servidor

req é a requisição para autenticação

PID_X é o endereço IP público do computador X

N_X é uma cadeia de caracteres aleatória de uso único enviada para o computador X

K_{req} é a chave compartilhada entre A e B

MAC é a função de autenticação criptográfica

, (uma vírgula) representa concatenação

mensagens do cliente ao servidor. A primeira sugestão consiste em dividir os 16 *bits* que compõem o número de porta *UDP* em dois campos: um para o número de sequência para controle dos pacotes e outro para os dados de autenticação. Essa proposta tem o inconveniente de limitar ainda mais a quantidade de informação transmitida num pacote, o que aumenta o ruído e diminui a eficiência da solução. A segunda sugestão se baseia no mapeamento dos dados de autenticação de forma que a sequência de portas seja monotonicamente crescente. Por exemplo, considere que a faixa de portas para envio de informações se inicie na porta 1024 e que a seguinte sequência de portas "121, 63, 148, 220, 7" deva ser transmitida. Alterando a codificação para fazer que a sequência de portas seja monotonicamente crescente, obtêm-se "1145, 1208, 1356, 1576, 1583". Embora a proposta de deGraff, Aycock & Jacobson (2005) avance na questão de autenticação na camada de rede, com a utilização de um protocolo desafio/resposta conciliado ao *PortKnocking*, deixa alguns pontos em aberto. Os protocolos sugeridos são vulneráveis a ataques *interleaving*, a solução *NAT* não resolve a questão de computadores que compartilham o mesmo endereço IP Público, não há uma solução prática para a conexão lógica entre a fase de autenticação e a conexão subsequente, a utilização de uma sequência de portas é bastante ruidosa, além de não funcionar com perda de pacotes.

No trabalho Tariq, Baig & Saeed (2008) o autor propõe uma solução para o problema de falta de associação entre a fase de autenticação e a conexão subsequente, presentes no *PortKnocking* e no *SPA*. A sua proposta consiste em adicionar uma cadeia de caracteres

aleatória de uso único (*nonce*) na fase de autenticação, a qual seria a chave para associar a conexão a ser realizada pelo cliente após a liberação da porta no *Firewall*. Após a autenticação, o cliente solicita a conexão com o envio de um pacote *TCP SYN*, adicionando no campo *Options* a cadeia de caracteres aleatória de uso único criada na fase de autenticação. Somente os pedidos de conexão com a cadeia de caracteres aleatória de uso único correta são admitidos no servidor.

Uma estrutura para a autenticação de usuário na camada de rede foi proposto em Liew et al. (2010). Essa estrutura utiliza um mecanismo *OTP* para a autenticação do usuário, onde as chaves *OTP* são fornecidas pelo servidor por meio da rede celular, a autenticação é realizada com o método *SPA* e a conexão subsequente é protegida com a utilização de *IPSec*. Inicialmente o usuário envia uma mensagem de texto pela rede celular ao servidor requisitando uma chave para autenticação. O servidor responde informando uma porta aleatória, que deve ser utilizada para para a autenticação *SPA*, e uma chave *OTP*, o que na realidade correspondem a 4 chaves utilizadas para a fase de autenticação e para o túnel *IPSec*. O usuário insere manualmente no computador cliente a porta aleatória e a chave *OTP* e procede com o pedido de autenticação enviando o pacote *AP*. O pacote *AP* conterá as informações para criação do túnel *IPSec*. O processo do *SPA* no servidor recebe o pacote *AP*, valida o cliente e, em caso positivo, modifica as regras no *Firewall* e configura o túnel *IPSec* conforme as informações contidas no *AP*. Finalmente, o cliente inicia uma comunicação *VPN* para realizar a conexão ao serviço desejado. Na solução proposta o servidor não envia nenhuma informação ao cliente, pois os dados para estabelecimento da *VPN IPSec* já foram informados pelo próprio cliente no pacote *AP*. Apesar da estrutura apresentar uma solução interessante em termos de segurança, tem uma complexidade e custo consideráveis, em virtude da utilização do *IPSec* e da integração com a rede celular.

Nas técnicas anteriores para autenticação baseada na camada de rede, um único cliente pode provar sua identidade e obter permissão de acesso ao serviço desejado. Entretanto, em alguns cenários, é interessante delegar a decisão de permissão para acesso a um certo grupo de clientes. Recentemente, um esquema de autenticação baseada em camada de rede com compartilhamento secreto foi introduzida em Miklosovic (2011). Essa abordagem utiliza o compartilhamento secreto de Shamir (SHAMIR, 1979) para dividir a informação de autenticação contida na sequência de portas e compartilhar entre clientes diferentes. Assim, quando um cliente precisa acessar um serviço protegido no servidor, inicialmente ele deve entrar em contato com outros clientes solicitando sua aprovação para o acesso. A permissão para acesso ao cliente é concedido pelo

servidor somente se um certo número de diferentes clientes enviam suas partes de compartilhamento secreto aprovando a nova conexão.

3.5 SUMÁRIO

Foram vistos neste capítulo alguns métodos para a autenticação em camada de rede. A utilização de *VPN IPSec* pode autenticar um cliente pela camada de rede e proporcionar confidencialidade dos dados, mas tem o inconveniente de impor um *overhead* considerável, além de ser bastante complexo refletindo em ambiguidades, ineficiência e vulnerabilidades (FERGUSON; SCHNEIER, 2000).

As abordagens *PortKnocking* e *SPA* para a autenticação em camada de rede se destacam pela sua simplicidade. A simplicidade auxilia a auditoria da implementação das soluções, evitando vulnerabilidades que possam ficar ocultas devido à complexidade do código. Além disso, os dois métodos possibilitam a autenticação em camada de rede, protege os serviços contra "*0-day attacks*" e esconde os serviços em execução de um servidor. Essas características eliminam a maioria dos ataques originados de *script kiddies* e *worms* e requer maior conhecimento para que um atacante possa comprometer o sistema.

Contudo, a autenticação em camada de rede fornecida por *PortKnocking* e *SPA* tem problemas. O *PortKnocking* é uma técnica que é bastante ruidosa, exige que a sequência de portas seja entregue em ordem e provê uma segurança efetiva se for utilizado com criptografia e mecanismo *OTP*. O *SPA*, apesar representar uma melhora em alguns aspectos em relação ao *PortKnocking*, não soluciona de forma eficiente as questões de autenticação de clientes em ambientes *NAT* e associação entre a fase de autenticação e conexão posterior ao serviço desejado. Mesmo as variações propostas para melhoria das duas técnicas, como a utilização de protocolos desafio/resposta, estrutura com utilização de senhas *OTP* e *VPN IPSec*, utilização de cadeia de caracteres aleatória de uso único (*nonce*) para associação da autenticação e conexão subsequente, apresentam problemas, como pontos de vulnerabilidade, custo e praticidade.

Outro aspecto relevante a ser ressaltado é que todas as soluções de autenticação baseadas em *PortKnocking* e *SPA* são projetados de modo heurístico, sem a apresentação de uma prova rigorosa de segurança. Acredita-se que a implementação de uma solução baseado em protocolos com prova rigorosa de segurança corrobora para dar maior confiança ao esquema de autenticação e elimina a possibilidade de vulnerabilidades

presentes nas soluções heurísticas, conforme será apresentado no capítulo 4.

4 KNOCKID

Neste capítulo será apresentada a proposta deste trabalho: o *KnockID*, um protocolo de autenticação na camada de rede que emprega alguns conceitos do *PortKnocking* e de protocolos criptográficos de identificação. Além de ser facilmente gerenciável, o *KnockID* apresenta uma abordagem segura contra grande parte dos ataques até a data deste trabalho (programas de varredura, ataques de repetição, *script kiddies*, etc), podendo ser utilizado como substituto para as técnicas de autenticação convencionais. O *KnockID* pode ser usado tanto como um serviço de autenticação centralizado ou como uma camada de segurança para proteção de aplicações inseguras. A abordagem utilizada no *KnockID* se difere das demais soluções pelo fato de que se baseia em protocolos criptográficos com segurança demonstrável, contrastando com a abordagem heurística convencional.

Na abordagem proposta, a sequência de portas (que corresponde a um mecanismo de autenticação por senhas simétricas) do *Portknocking* é substituída por um protocolo de identificação de chave pública, onde um cliente prova para um servidor que possui uma chave secreta correspondente a uma chave pública específica. Desta forma, é possível que um cliente utilize um único par de chaves para realizar autenticação em diferentes serviços, já que apenas a chave pública deve ser compartilhada com os servidores e seu vazamento não compromete a segurança do sistemas. Uma vez que o *KnockID* é baseado num protocolo de identificação com segurança demonstrável, tem-se um protocolo de autenticação de clientes resistente à diversos ataques atuais no ambiente da *Internet*.

4.1 DEFINIÇÕES

Nesta seção serão apresentadas algumas definições e a notação para a compreensão do KnockID. Tais definições serão utilizadas no desenvolvimento do protocolo KnockID.

Será utilizado o termo "*Provador*" para designar o cliente que deseja se autenticar e acessar serviços no servidor. O termo "*Verificador*" será utilizado para referenciar o servidor que autentica os clientes e disponibiliza os serviços solicitados. O termo *provador* indica que o cliente deve provar a sua identidade, enquanto o termo *verificador*

ressalta a característica do servidor que verifica e valida as solicitações de autenticação. Tais termos podem ser referenciados pelas suas iniciais, ou seja, o *provador* pode ser referenciado pela letra "P" e o *verificador* pela letra "V".

O algoritmo $Gen(1^n)$ é definido para a geração do par de chaves pública e privada com n *bits*. A execução do algoritmo $Gen(1^n)$ retorna um par de chaves (pk, sk). Ambos o *provador* e *verificador* devem executar tal algoritmo para gerar o seu respectivo par de chaves. Logo, tem-se o par de chaves (pk_p, sk_p) que corresponde às chaves pública e privada do *provador* utilizadas, respectivamente, para cifrar e assinar mensagens. De forma análoga, o par de chaves (pk_v, sk_v) corresponde às chaves pública e privada do *verificador* utilizadas, respectivamente, para cifrar e assinar mensagens.

No protocolo *KnockID* algumas informações são enviadas do *provador* para o *verificador*. Dentre tais informações, pode-se destacar as informações denominadas como ID_p, *Acao* e *Cmd*. O ID_p é a identidade do *provador*, forma pela qual o *verificador* identifica qual *provador* realiza a solicitação de autenticação. O ID_p pode ser o endereço *MAC* do *provador* ou qualquer cadeia de caracteres que o identifique de forma única. No *KnockID* foi adotado um nome para o *provador*, como um *hostname* ou um apelido, para ser utilizada no campo ID_p. *Acao* é uma informação que o *provador* fornece ao *verificador* que diz qual é a ação solicitada no momento da autenticação. No *KnockID* há dois tipos de ações previstos que um *provador* pode solicitar: a abertura de uma porta no *Firewall* para o *provador* ou a execução de um comando no ambiente do *verificador*. *Cmd* é um argumento passado pelo *provador* ao *verificador*, onde é informado o comando solicitado para execução no *verificador* ou a porta de *Firewall* a ser aberta. O argumento *Cmd* é diretamente relacionado ao campo *Acao*.

O algoritmo $Sign_{sk}(\cdot)$ é um algoritmo de assinatura de mensagem. Na assinatura de uma mensagem é utilizada a chave privada de quem está assinando, de forma que todos os que tenham a chave pública correspondente possam verificar quem foi realmente o emissor da mensagem.

O algoritmo $E_{pk}(\cdot)$ é um algoritmo para cifrar uma mensagem. Esse algoritmo utiliza a chave pública do destinatário. Assim, somente o destinatário é capaz de decifrar e compreender a mensagem.

O protocolo *KnockID* é um protocolo de desafio/resposta. Durante a troca de mensagens entre o *provador* e o *verificador* para o desenvolvimento do protocolo, algumas

cadeias de caracteres aleatórias são geradas. CH_v é uma cadeia de caracteres aleatória gerada pelo *verificador*. Essa cadeia de caracteres é o desafio que o *verificador* envia ao *provador*. CH_p é uma cadeia de caracteres aleatória gerada pelo *provador*. Essa cadeia de caracters é utilizada na resposta do *provador* ao *verificador*. Ao fim da execução do protocolo, ambos *provador* e *verificador* obtém como resultado o SID, o identificador de sessão resultante da fase de autenticação, composta pela concatenação das cadeias de caracteres CH_v e CH_p.

4.1.1 Protocolos de identificação

Protocolos de identificação permitem a um *provador* comprovar a posse de uma chave secreta sk associada a um par de chaves (pk, sk) para um *verificador* que possui pk, sem revelar sk (*i.e.* uma prova de conhecimento da chave secreta com *Zero Knowledge*). Considerando esta descrição no contexto de autenticação de usuário, os protocolos de autenticação permitem de forma simples a um usuário provar a posse de sua chave secreta para um servidor de autenticação, o qual possui a chave pública correspondente. Essa abordagem pode ser naturalmente aplicada à soluções de autenticação de usuário.

4.1.2 Noções de segurança

O *KnockID* deve ser seguro em ambientes hostis como a *Internet*. Logo, o modelo de segurança deve prever um adversário poderoso o suficiente para que um protocolo seja seguro. O *KnockID* se baseia no protocolo de identificação seguro a "*reset*" proposto por Bellare em Bellare et al. (2001).

No trabalho do Bellare são propostos protocolos seguros a ambientes "*resetáveis*", fornecendo ao adversário características que lhe dão grande poder de atuação. O objetivo do adversário é personificar o *provador*, de forma a ser aceito pelo *verificador*, como se ele fosse proprietário da chave pública pk_p. Ao adversário é garantido as seguintes propriedades:

- É permitido ao adversário reiniciar o estado interno do *provador*, enquanto interage com o *provador*. Ou seja, o adversário pode realizar um "*backup*" do *provador*, a fim de capturar suas aleatoriedades, e, também reiniciá-lo continuando a iteração com o *provador*. A vantagem do "*reset*" do *provador* poderia auxiliar o adversário na tarefa de personificação do *provador*, no sentido que,

como o adversário conhece as aleatoriedades já utilizadas, poderia reutilizá-las a partir da reinicialização do *provador*;

- O adversário pode, enquanto tenta personificar o *provador*, iteragir paralelamente, no papel de *verificador*, com várias instâncias do *provador*, reiniciando o *provador* para suas condições iniciais, e, também simultâneamente com o *verificador*, no papel de *provador*. Ou seja, o adversário pode manter o acesso aos *provadores* enquanto tenta convencer o *verificador* a aceitá-lo. Observe que essa propriedade garante ao adversário a vantagem de poder "quebrar" a segurança do protocolo simplesmente realizando um ataque *MITM*, repassando as mensagens entre o *provador* e o *verificador*.

As propriedades apresentadas asseguram um modelo adversarial forte capaz de representar, de forma equivalente, um adversário proveniente da *Internet*. Logo, o protocolo do Bellare que foi construído segundo este modelo adversarial é um protocolo robusto que atende os requisitos de segurança para ser utilizado na *Internet*. O *KnockID*, conforme será visto neste capítulo, é em essência um dos protocolos apresentados em Bellare et al. (2001). Logo, o *KnockID* herda a robustez e padrões de segurança do protocolo original necessários para sua utilização num ambiente hostil.

Para que se tenha uma noção do nível de segurança dos protocolos apresentados em Bellare et al. (2001), e, consequentemente, do protocolo *KnockID*, será apresentado, de forma suscinta, algumas definições do trabalho realizado pelo Bellare. Em particular, a apresentação será focada em definições, teoremas e corolários que mostram um esquema de identificação baseado em assinatura digital, todos extraídos Bellare et al. (2001).

Bellare define duas versões de modelo adversarial: o modelo *CR1*, que se aplica a ambientes de cartão de crédito; e o modelo *CR2*, o qual se aplica a uma rede ou *Internet*. O foco será concentrado no modelo *CR2*, já que um dos objetivos *KnockID* é permitir uma identificação segura no ambiente da *Internet*. No modelo *CR2*, também chamado de configuração *CR2*, as instâncias de *provador* e a instância do *verificador* estão disponíveis simultaneamente ao adversário. Em particular, o adversário pode repassar as mensagens do *provador* ao *verificador* e vice-versa.

Em Bellare et al. (2001) também é definido um experimento para cada modelo. No modelo *CR2* o experimento é denominado como $\mathbf{Experiment}^{id-cr2}_{\mathcal{ID},I}(k)$, onde $\mathcal{ID}$ é o protocolo de identificação executado no experimento, I é o adversário e k é o parâmetro

de segurança. O experimento começa com algumas inicializações (geração de chaves, inicialização do *verificador*). Após a inicialização, o adversário é invocado pela entrada da chave pública pk, dando andamento à próxima fase do experimento que é a fase de execução do adversário I. Nessa fase de execução, o adversário I pode ativar novas intâncias de *provador*, bem como interagir com as instâncias ativas e com o *verificador*. Na interação do adversário com o *provador*, I assume o papel do *verificador*, ou seja, o adversário passa mensagens ao *provador* como se fosse o *verificador*. Nessa interação a noção de "*reset*" do *provador* é capturada ao se permitir que o adversário possa enviar mensagens arbitrárias (válidas) para ser consultado no *provador*. Por exemplo, suponha que $MSG_1 \| MSG_2$ indique a troca de duas mensagens do protocolo $\mathcal{ID}$, onde MSG_1 é a mensagem inicial do *provador* e MSG_2 é a mensagem do *verificador*. Nesse exemplo, o adversário pode enviar diferentes valores de MSG_2, ao se passar pelo *verificador*, correspondendo a um "*reset*" sucessivo do *provador* até que este aceite MSG_2 como válida. Observe que o adversário pode interagir com várias instâncias de *provador* simultaneamente. A resposta da interação do adversário com uma instância do *provador* é a próxima mensagem do protocolo destinado ao *verificador*, e, no caso dessa resposta ser a última mensagem do protocolo, o adversário gera o identificador de sessão para o i-ésimo *provador* (SID_i), resultante de todas as mensagens trocadas no protocolo $\mathcal{ID}$.

No experimento o adversário pode interagir também com o *verificador*, assumindo o papel de *provador*. Não é permitido ao adversário I o "*reset*" do *verificador*. Se a consulta realizada pelo adversário ao *verificador* for menor que o número total de mensagens previsto no protocolo, então o *verificador* retorna uma nova mensagem dando prosseguimento ao protocolo $\mathcal{ID}$. Porém, se a consulta realizada pelo adversário ao *verificador* for a última mensagem prevista no protocolo, o *verificador* gera o identificador de sessão SID_V e a decisão se aceita ou não a identificação, fruto das mensagens compartilhadas do protocolo.

Observe que pelo experimento $\mathbf{Experiment}^{id-cr2}_{\mathcal{ID},I}(k)$, se um adversário realizar "*Man-in-The-Middle*" entre o *verificador* e alguma instância de *provador*, repassando as mensagens de um ao outro, levará o aceite da identificação pelo *verificador*. Tal comportamento não é de fato um ataque, pois não há prejuízo no caso do *verificador* aceitar a identificação sob tais circunstâncias, uma vez que na realidade o *verificador* está identificando de fato um *provador*. De qualquer forma, para finalizar o experimento deve ser definido como exatamente o adversário pode ganhar. A chave para essa definição é a idéia de "correspondência de identificadores de sessão" (*id* de sessão), explicitada

em Bellare, Pointcheval & Rogaway (2000). Um identificador de sessão compartilhado entre o *verificador* e a instância do *provador* (SID) seria como se fosse o "nome da conexão", que possibilita ao *verificador* diferenciar entre as diferentes instâncias de *provador*. O SID não é secreto, pelo contrário, é público, de forma que o adversário tem conhecimento de sua criação por meio das instâncias de *provador* que interage. A vitória do adversário I no experimento $\mathbf{Experiment}^{id-cr2}_{\mathcal{ID},I}(k)$ corresponde ao ato do *verificador* aceitá-lo com um SID que não seja utilizada por nenhuma instância de *provador*. Em outras palavras, I é vitorioso no experimento se puder se identificar junto ao *verificador* com um SID inédito. Admite-se também o sucesso do adversário no experimento se ele conseguir "confundir" o *verificador* fazendo com que duas instâncias de *provador* tenha o mesmo SID. Também é admitido que um adversário nunca repete uma consulta.

O experimento indica quais são as circunstâncias em que o adversário é declarado vitorioso. Por sua vez, a definição do protocolo é responsável por assegurar que ambas as partes devem rejeitar uma mensagem recebida se esta for inconsistente. O esquemático do protocolo $\mathcal{ID}$ é apresentado na figura 4.1, de acordo com o trabalho Bellare et al. (2001).

Provador		Verificador
pk, sk ; Aleatoriedades : $R_P = CH_P$		pk ; Aleatoriedades : $R_V = CH_V$
	Início $\longrightarrow$	
	$\longleftarrow$ CH_V	
SIG $\leftarrow \mathcal{DS}(\mathsf{sign}, sk, CH_V \Vert CH_P)$		
	$CH_P \Vert \text{SIG} \longrightarrow$	
		Saída: $\mathsf{SID}_V = CH_V \Vert CH_P$
Saída: $\mathsf{SID}_P = CH_V \Vert CH_P$		e : Decisão $= \mathcal{DS}(\mathsf{vf}, pk, CH_V \Vert CH_P, \text{SIG})$

$\mathcal{ID}(\text{keygen}, k) = \mathcal{DS}(\text{keygen}, k) - \mathcal{ID}$ tem o mesmo processo de geração de chaves que $\mathcal{DS}$	
$\mathcal{ID}(\text{prvmsg}, sk, x; R_P)$ onde $\lvert R_P \rvert = pcl(k)$ – Parsear x como $\text{MSG}_1 \Vert \ldots \Vert \text{MSG}_l$ – Se $l \notin \{0, 2\}$ então retornar $\perp$ – Se $l = 0$ então retornar Início – Se $\lvert \text{MSG}_2 \rvert \neq vcl(k)$ então retornar $\perp$ – $CH_V \leftarrow \text{MSG}_2$; $\mathsf{CH}_P \leftarrow R_P$ – SIG $\leftarrow \mathcal{DS}(\mathsf{sign}, sk, CH_V \Vert CH_P)$ – Retornar $CH_P \Vert \text{SIG}$	$\mathcal{ID}(\text{prvsid}, sk, x; R_P)$ onde $\lvert R_P \rvert = pcl(k)$ – Parsear x como $\text{MSG}_1 \Vert \ldots \Vert \text{MSG}_l$ – Se $l \neq 3$ ou $\lvert \text{MSG}_2 \rvert \neq vcl(k)$ então retornar $\perp$ – $CH_V \leftarrow \text{MSG}_2$; $\mathsf{SID}_P \leftarrow CH_V \Vert R_P$ – Retornar SID_P
$\mathcal{ID}(\text{vrfmsg}, pk, x; R_V)$ onde $\lvert R_V \rvert = vcl(k)$ – Parsear x como $\text{MSG}_1 \Vert \ldots \Vert \text{MSG}_l$ – Se $l \neq 1$ então retornar $\perp$ – $CH_V \leftarrow R_V$ – Retornar CH_V	$\mathcal{ID}(\text{vfend}, pk, x; R_V)$ onde $\lvert R_V \rvert = vcl(k)$ – Parsear x como $\text{MSG}_1 \Vert \ldots \Vert \text{MSG}_l$ – Se $l \neq 3$ ou $\text{MSG}_2 \neq R_V$ então retornar $\perp$ – Parsear MSG_3 como $CH_P \Vert \text{SIG}$ com $\lvert CH_P \rvert = pcl(k)$ – $CH_V \leftarrow \text{MSG}_2$; $\mathsf{SID}_V \leftarrow CH_V \Vert R_P$ – Decisão $= \mathcal{DS}(\mathsf{vf}, pk, CH_V \Vert CH_P, \text{SIG})$ – Retornar (SID, Decisão)

Figura 4.1: Esquemático do protocolo de identificação $\mathcal{ID}$ seguro contra *reset* na configuração *CR2* baseado no esquema de assinatura $\mathcal{DS}$ (BELLARE et al., 2001).

A definição de segurança para o protocolo $\mathcal{ID}$, a qual é reproduzida a seguir:

Definição 1 (BELLARE et al., 2001) ***[Segurança de um protocolo ID na configuração CR2]***
Seja $\mathcal{ID}$ uma descrição de protocolo de identificação. Seja I um adversário e seja k o parâmetro de segurança. A vantagem do adversário I é:

$$\mathbf{Adv}^{\mathrm{id-cr2}}_{\mathcal{ID},I}(k) = \Pr[\mathrm{Win}_I = \mathrm{true}] \tag{4.1}$$

onde a probabilidade é com respeito ao $\mathbf{Experiment}^{id-cr2}_{\mathcal{ID},I}(k)$. *O protocolo $\mathcal{ID}$ é dito polinomialmente seguro na configuração CR2 se* $\mathbf{Adv}^{id-cr2}_{\mathcal{ID},I}(\cdot)$ *é desprezível para qualquer adversário I de tempo-complexidade polinomial em k.*

Para a definição 1 são adotadas algumas convenções sobre o tempo e complexidade de consulta. O tempo-complexidade $t(k)$ de um adversário I é o tempo de execução de todo o experimento $\mathbf{Experiment}^{id-cr2}_{\mathcal{ID},I}(k)$, incluindo o tempo de inicialização, cálculo das respostas das consultas do adversário e cálculo de Win_I. Também é definido a complexidade-consulta $q(k)$ de I que corresponde ao número de consultas realizadas às instâncias de *provador* pelo adversário I no experimento. Observe que sempre $q(k) \leq t(k)$, de forma que um adversário de tempo-complexidade polinomial tem uma complexidade-consulta polinomial.

Como o protocolo utilizado é baseado na segurança da assinatura digital (o *provador* se identifica ao *verificador* mostrando que conhece a chave *sk* provando que sabe assinar), também deve-se verificar a definição de segurança de um esquema de assinatura digital. A figura 4.2 descreve o esquema de assinatura digital $\mathcal{DS}$ com todas as suas funcionalidades associadas ao esquema de assinatura. A segurança de um esquema de assinatura digital $\mathcal{DS}$ é dada pela definição 2:

$(p_k, s_k) \leftarrow \mathcal{DS}(\mathsf{keygen}, k)$ - Geração do par de chaves
SIG $\leftarrow \mathcal{DS}(\mathsf{sign}, s_k, \mathrm{MSG})$ - Cálculo da assinatura da mensagem MSG
$\mathsf{decision} \leftarrow \mathcal{DS}(\mathsf{vf}, p_k, \mathrm{MSG}, \mathrm{SIG})$ - Verifica se SIG é uma assinatura válida de MSG (aceita ou rejeita)

Figura 4.2: Descrição de um esquema de assinatura digital $\mathcal{DS}$ (BELLARE et al., 2001).

Definição 2 (BELLARE et al., 2001) ***[Segurança de um esquema de assinatura digital]***
Seja $\mathcal{DS}$ um esquema de assinatura digital, F um adversário (chamado de forjador

nesse contexto) que tem acesso a um oráculo e k o parâmetro de segurança. Define-se:

$$\textbf{Experiment}^{\text{ds}}_{\mathcal{DS},F}(k)$$
$$(p_k, s_k) \leftarrow \mathcal{DS}(\mathsf{keygen}, k); \text{Win}_F \leftarrow \mathsf{false}$$
$$(\text{MSG}, \text{SIG}) \leftarrow F^{\mathcal{DS}(\mathsf{sign}, s_k, \cdot)}(p_k)$$
Se $\mathcal{DS}(\mathsf{vf}, p_k, \text{MSG}, \text{SIG}) = \mathsf{accept}$ *e* F *nunca consultar ao oráculo a mensagem* MSG*, então* $\text{Win}_F \leftarrow \mathsf{true}$

A vantagem do forjador F é dada pela equação 4.2:

$$\textbf{Adv}^{ds}_{\mathcal{DS},F}(k) = \Pr[\text{Win}_F = \mathsf{true}] \tag{4.2}$$

onde a probabilidade é com respeito ao $\textbf{Experiment}^{\text{ds}}_{\mathcal{DS},F}(k)$*. O esquema de assinatura digital é dito polinomialmente seguro se* $\textbf{Adv}^{ds}_{\mathcal{DS},F}(\cdot)$ *é desprezível para qualquer forjador F de tempo-complexidade polinomial em k.*

Tal como na definição 1, o tempo-complexidade $t(k)$ do adversário F é definido como o tempo de execução do $\textbf{Experiment}^{\text{ds}}_{\mathcal{DS},F}(k)$. Uma observação importante a ser feita é que o esquema de assinatura deve ser sem estado e determinístico para prover segurança na configuração *CR2* (ambiente "*resetável*"), pois do contrário, é permitido a um adversário obter as assinaturas de diferentes mensagens sob o mesmo conjunto de aleatoriedade.

A segurança concreta do protocolo $\mathcal{ID}$ pode ser vista em termos da segurança do esquema de assinatura digital $\mathcal{DS}$. O teorema 1 ilustra a segurança concreta do protocolo $\mathcal{ID}$, indicando que a probabilidade de um adversário ter sucesso no ataque de $\mathcal{ID}$ é limitado pela probabilidade de um forjador atacar $\mathcal{DS}$.

Teorema 1 (BELLARE et al., 2001) ***[Segurança concreta do esquema de assinatura $\mathcal{ID}$ na configuração CR2]***
Seja $\mathcal{DS}$ um esquema de assinatura determinístico e sem estado, seja $vcl(\cdot)$ e $pcl(\cdot)$ funções limitadas polinomialmente, e, seja $\mathcal{ID}$ um esquema de identificação como identificado na figura 4.1. Se I é um adversário com tempo-complexidade $t(\cdot)$ e complexidade-consulta $q(\cdot)$ que ataca $\mathcal{ID}$ na configuração CR2, então existe um forjador F atacando $\mathcal{DS}$ tal que:

$$\textbf{Adv}^{id-cr2}_{\mathcal{ID},I}(k) \leq \textbf{Adv}^{ds}_{\mathcal{DS},F}(k) + \frac{q(k)}{2^{vcl(k)}} + \frac{q(k)^2 - q(k)}{2^{pcl(k)+1}} \tag{4.3}$$

Além disso, F tem tempo-complexidade $t(k)$ e realiza no máximo $q(k)$ consultas de assinaturas no ataque de mensagens escolhidas ao $\mathcal{DS}$.

O teorema 1 implica no corolário 1:

Corolário 1 (BELLARE et al., 2001) ***[Segurança polinomial do esquema de assinatura $\mathcal{ID}$ na configuração CR2]***
Seja $\mathcal{DS}$ um esquema de assinatura determinístico e sem estado, seja $vcl(\cdot) = pcl(\cdot) = k$, e, seja $\mathcal{ID}$ um esquema de identificação como identificado na figura 4.1. Se $\mathcal{DS}$ é seguro polinomialmente, então $\mathcal{ID}$ é seguro polinomialmente na configuração CR2.

Por sua vez, o corolário 1 juntamente com a proposição 1 implica no corolário 2.

Proposição 1 (BELLARE et al., 2001) *Se existe uma função one-way então existe um esquema de assinatura digital polinomialmente seguro sem estado e determinístico.*

Corolário 2 (BELLARE et al., 2001) ***[Existência de um esquema de assinatura $\mathcal{ID}$ seguro polinomialmente na configuração CR2]***
Assuma que exista função one-way. Então existe um esquema de identificação que é seguro polinomialmente na configuração CR2.

Todas essas definições, teorema, proposição e corolários significam que se pode provar a existência de um protocolo seguro na configuração *CR2* sob a suposição de complexidade mínima da existência de uma função *one-way*. As provas são esboçadas em Bellare et al. (2001).

4.2 PROTOCOLO KNOCKID

Algumas premissas são assumidas para o funcionamento do *KnockID*. Numa fase anterior à execução do protocolo *KnockID* pressupõem-se que:

- P executa algoritmo $Gen(1^n)$ para obter o par de chaves (pk_p, sk_p);
- V executa algoritmo $Gen(1^n)$ para obter o par de chaves (pk_v, sk_v);
- As chaves públicas são compartilhadas entre P e V.

As premissas relatadas são essenciais para que o protocolo *KnockID* possa funcionar corretamente. No entanto, não será objeto desse trabalho descrever como ocorre o compartilhamento das chaves públicas. Admite-se que exista alguma forma de compartilhamento de chaves de forma segura e eficiente, como uma infra-estrutura de chave-pública (*PKI*).

O protocolo *KnockID* é essencialmente o protocolo de identificação *reset*-seguro na configuração *CR2* baseado em assinatura digital do trabalho *Identification Protocols Secure Against Reset Attacks* (BELLARE et al., 2001). Há uma pequena modificação na composição da mensagem inicial, onde esta é cifrada para ter a confidencialidade nos dados do *provador*. O funcionamento básico do protocolo *KnockID* é apresentado no protocolo 4.1, onde há somente a autenticação do *provador*. O presente protocolo possui a particularidade de que o *provador* atesta sua identidade mostrando que sabe assinar algo escolhido pelo *verificador*. O protocolo proposto herda características do protocolo de identificação original e é seguro em ambientes com *reset*, como é o caso da *Internet*.

Observe que o protocolo 4.1 consiste em apenas 3 mensagens. Uma mensagem de requisição do *provador*, onde o *provador* solicita uma ação designada dentro da própria mensagem. Uma mensagem com o envio de desafio pelo *verificador*. E, uma mensagem de resposta do *provador*, o qual prova sua identidade assinando o desafio enviado pelo *verificador*. Ao fim das 3 mensagens, ambos *provador* e *verificador* geram o SID que é a composição das aleatoriedades do *verificador* e *provador* trocadas nas mensagens de desafio e resposta. Após a autenticação a ação é executada pelo *verificador*. Por exemplo, se a ação solicitada for a abertura de uma porta no *Firewall* para uma conexão *ssh*, o *verificador* procederá com a abertura de regra e o *provador* dará prosseguimento efetuando a conexão subsequente na porta 22.

Para facilitar o entendimento do protocolo *KnockID* é ilustrado o seu funcionamento por meio do diagrama da figura 4.3:

Protocolo 4.1 Protocolo KnockID

1: P envia uma mensagem cifrada com a chave pública de V (pk_v) para o início da autenticação. A mensagem contém a identificação de P (ID_p), a ação a ser executada (abertura de firewall ou execução de comando), uma *string* Cmd (indicando a porta do firewall a ser aberta ou o comando a ser executado) e a assinatura $SIG_1 = Sign_{sk_p}(Acao; Cmd)$ para comprovar que a origem da mensagem e garantir que a mesma não foi alterada no trânsito até o destino. Os campos são separados pelo delimitador ";".

$$P \rightarrow V : E_{pk_v}(ID_p; Acao; Cmd; SIG_1)$$

2: V decifra a mensagem recebida com sua chave secreta sk_v e verifica se a mensagem é válida por meio da assinatura. V também verifica se o ID_p existe. Se a mensagem for inválida e/ou o ID_p não existir V não faz nada.

3: Se a mensagem enviada por P à V for válida e a identificação de P existir, V envia uma mensagem à P com o desafio CH_v:

$$V \rightarrow P : CH_v$$

4: P recebe o desafio e assina a mensagem resultante da concatenação do desafio recebido CH_v com a aleatoriedade CH_p:

$$SIG_2 = Sign_{sk_p}(CH_v||CH_p)$$

5: P envia à V a aleatoriadade CH_p concatenada com a assinatura SIG_2:

$$P \rightarrow V : CH_p||SIG_2$$

6: V abre o conteúdo da assinatura SIG_2 e verifica se esta é igual à concatenação das aleatoriedades $CH_v||CH_p$ (CH_p foi recebido em claro na última mensagem de P). Se a verificação for positiva, V define como identificador de sessão SID_v a cadeia de caracteres $CH_v||CH_p$ e realiza as ações solicitadas por P. Se a verificação for negativa, V não faz nada.

7: P define como identificador de sessão SID_p a cadeia de caracteres $CH_v||CH_p$.

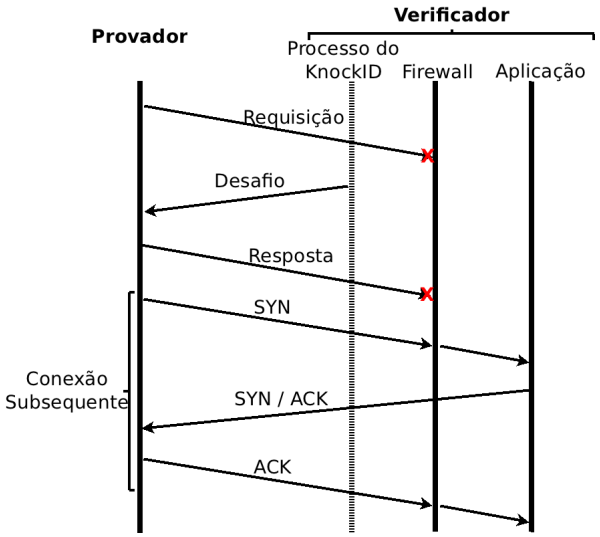


Figura 4.3: Diagrama de funcionamento do KnockID.

A seguir é apresentado um resumo do conteúdo dos pacotes na fase de autenticação:

- Mensagem *Requisição*: $E_{pk_v}(ID_p; Acao; Cmd; SIG_1)$, onde $SIG_1 = Sign_{sk_p}(Acao; Cmd)$;
- Mensagem *Desafio*: CH_v;
- Mensagem *Resposta*: $CH_p||SIG_2$, onde $SIG_2 = Sign_{sk_p}(CH_v||CH_p)$.

4.3 ANÁLISE DE SEGURANÇA

Conforme já fora mencionado, a segurança do protocolo proposto provém do protocolo de identificação utilizado como base (BELLARE et al., 2001). Este protocolo é demonstravelmente seguro em um modelo onde o adversário pode manipular o *provador* de forma e causar "*resets*" (e.g. reiniciar um dispositivo) e também realizar quaisquer outros tipos de ataques que não envolvam modificar o estado interno dos dispositivos. Ou seja, mesmo que um adversário possa reiniciar dispositivos arbitrariamente e modificar/controlar **toda** a comunicação entre eles, o protocolo de identificação de (BELLARE et al., 2001) permanece seguro.

A modificação realizada no protocolo *KnockID* em relação ao protocolo de identificação original do Bellare et al. (2001) se restringe à mensagem inicial do *provador*, destinada à requisição de autenticação ao *verificador*. A mensagem inicial do protocolo *KnockID* é enviada cifrada e contém dados de identificação do *provador*, especificação da solicitação

de comandos no *verificador*, e, também a assinatura das informações de solicitação de comandos. O envio da mensagem inicial em forma cifrada se destina a preservar a identidade do *provador* e proteger a visualização dos comandos solicitados por um *sniffer*. A assinatura dos campos de comandos se destina a dar autenticidade para a solicitação de comandos, uma vez que o *verificador* pode comprovar que a assinatura foi realizada de fato pelo *provador*. Observe que essa composição de campos empregada na mensagem inicial do *KnockID* não compromete a segurança do protocolo original: a mensagem inicial do protocolo original se destina ao início da execução do protocolo, com o objetivo claro de solicitar autenticação. Na mensagem inicial do protocolo do *KnockID* o objetivo é o mesmo. O protocolo original não entra em detalhes sobre as informações dessa mensagem inicial, de forma que as informações da mensagem inicial do protocolo *KnockID* não prejudicam o teor de segurança do protocolo. E, como o restante do protocolo *KnockID* é idêntico ao protocolo original, pode-se concluir que o protocolo *KnockID* proposto herda as características de segurança do protocolo de identicação original.

A segurança do protocolo *KnockID* segue das definições, teoremas e corolários apresentados, com prova demonstrada em Bellare et al. (2001). A segurança do protocolo é evidenciada pelo seguinte aspecto: a única ação que o adversário pode fazer com um oráculo *provador* é alimentá-lo com aleatoriedades de desafio e obter suas assinaturas. Se o esquema de identificação é seguro contra ataque de mensagem escolhida, tal ação não ajudará o adversário a forjar uma assinatura, a menos que ele adivinhe o último desafio lançado pelo *verificador*. Porém, a probabilidade de se adivinhar o desafio pode ser bem pequena se o tamanho da aleatorieadade desafio for suficientemente grande.

O protocolo *KnockID* é seguro contra ataques de repetição, já que a cada tentativa de autenticação novos dados aleatórios são utilizados. Ataques *Man-In-The-Middle* não correspondem a um ataque de fato no presente protocolo, uma vez que o *verificador* identificará o *provador*, ficando o adversário só repassando as mensagens do protocolo. Assim, conclui-se também que o protocolo *KnockID* não permite ataques de personificação, pois o adversário não é capaz de se autenticar e gerar um SID que não seja de legítimo *provador*.

4.4 SUMÁRIO

Foi apresentado neste capítulo a proposta deste trabalho, o protocolo *KnockID*. O protocolo *KnockID* é bastante simples, tratando-se de um protocolo de autenticação

desafio/resposta e é seguro em ambientes como a *Internet*. Este protocolo reune alguns aspectos do método *PortKnocking*, de modo que além de autenticar os usuários, mantém as aplicações de um servidor protegidas, pois as oculta de atacantes, eliminando a ação de programas de varredura e *script kiddies*.

O principal diferencial do *KnockID* está na segurança demonstrável. O *KnockID* é em suma o protocolo de identicação *reset*-seguro na configuração *CR2* baseado em assinatura digital do Bellare, cuja segurança foi demostrada em Bellare et al. (2001). Logo, o *KnockID* tem a sua segurança atestada pelo protocolo original.

No próximo capítulo será apresentado o protótipo desenvolvido para verificação da viabilidade do *KnockID*. Será visto os aspectos gerais para a implementação do protótipo *KnockID*, bem como apresentado os resultados obtidos com o desenvolvimento.

5 PROVA DE CONCEITO

Este capítulo se destina a implementação do protocolo *KnockID*, como uma prova de conceito, um protótipo, a fim de atestar a viabilidade da construção do protocolo, levantar os pontos de dificuldade da implementação e avaliar o comportamento do protocolo em funcionamento. A linguagem *Perl* foi utilizada para o desenvolvimento do protótipo *KnockID*. O emprego do *Perl* se deve à facilidade de sua sintaxe, o que permite um desenvolvimento rápido de um programa, e também pela existência de vários módulos com primitivas criptográficas e ferramentas para a comunicação em rede.

5.1 IMPLEMENTAÇÃO DO KNOCKID

A escolha da linguagem *Perl* para o desenvolvimento do protótipo do *KnockID* tem fundamento em vários fatores, como é destacado a seguir. *Perl* é uma linguagem de fácil aprendizado e eficiente, o que permite uma rápida familiarização a programadores e iniciantes, bem como facilita a manutenção dos códigos de programas, caso seja necessário. Sua sintaxe é bastante parecida com a sintaxe da linguagem C, porém não requer a definição de tipos de variáveis (linguagem não "tipada"), uma vez que o interpretador da linguagem *Perl* se encarrega de atribuir/converter o tipo das variáveis em tempo de execução. O gerencimento de memória também é automático, com a alocação ou liberação de memória de acordo com o necessário. Outra facilidade que se encontra no *Perl* é o grande número de códigos livres disponibilizados por programadores que utilizam essa linguagem. Muitos desses códigos são disponibilizados na forma de módulos *Perl*, uma espécie de bibliotecas que podem ser empregadas para finalidades diversas. Esses módulos são agrupados no *Comprehensive Perl Archive Network* (*CPAN*)[15], o qual atua como repositório permitindo o livre acesso e compartilhamento de módulos Perl para os inúmeros utilizadores da linguagem. Tais características do *Perl* são interessantes para o desenvolvimento de um protótipo, como é o caso do *KnockID*.

[15] *CPAN* é o *site* de referência para a linguagem *Perl*. Nesse *site* os desenvolvedores depositam suas implementações sob a forma de módulos (bibliotecas) com vários fins. O *CPAN* pode ser acessado no endereço: www.cpan.org. Todos os módulos citados na implementação do *KnockID* foram obtidos por meio do *CPAN*, onde pode-se obter também a documentação dos mesmos.

No protótipo do *KnockID* foram desenvolvidos dois programas. O primeiro sendo o processo *KnockID* no servidor (*daemon*), chamado de *KnockIDserver*. O *KnockIDserver* realiza o papel do *verificador* no protocolo proposto. O *KnockIDserver* tem por objetivo monitorar os pacotes da rede processando as requisições de autenticação correspondentes a pacotes de solicitação válidos, autenticar os clientes de acordo com o protocolo *KnockID*, e, liberar as portas do *Firewall* para os clientes autenticados ou executar os comandos solicitados. O segundo programa é o do cliente, chamado de *KnockIDclient*, assumindo o papel de *provador* no protocolo. O *KnockIDclient* é responsável por iniciar o protocolo enviando um pacote de solicitação válido e também para responder ao desafio enviado pelo *KnockIDserver*, assinando-o e assim provando sua identidade.

Em ambos programas, foi utilizada a primitiva de chave pública *RSA* para a criptografia e assinatura das mensagens. A primitiva *RSA* foi empregada através do módulo *Perl Crypt:RSA*. Para a criptografia, que é prevista na primeira mensagem do *KnockIDclient*, foi utilizado o esquema de criptografia *Optimal Asymmetric Encryption Padding* (*OAEP*), o que dá maior segurança para a cifragem da mensagem por adicionar um preenchimento pseudo-aleatório e utilizar funções de *hash*. Já na assinatura das mensagens foi utilizado o esquema de assinatura *PKCS* #1 v1.5, onde *PKCS* significa *Public Key Cryptography Standards*. A razão do emprego do esquema *PKCS* #1 v1.5 se deve ao pré-requisito explícito no teorema 1, o qual diz que deve ser utilizado um esquema de assinatura digital determinístico e sem estado.

Outro ponto de atenção na implementação do *KnockID* é a geração das cadeias de caracteres aleatórias CH_v e CH_p. Foi assumido um tamanho de 16 *bytes* (128 *bits*) para as cadeias de caracteres aleatórias, o que acarreta uma baixa probabilidade do adversário adivinhar o desafio e tentar quebrar o protocolo. As cadeias de caracteres foram geradas com o auxílio do módulo *Perl Crypt::Random*. Este módulo foi escrito seguindo premissas de segurança, com o objetivo de gerar cadeias de caracteres pseudo-aleatórias realmente seguras. Utilizou-se a função makerandom_octet desse módulo *Perl* para a construção das referidas cadeias de caracteres.

Visando dar maior flexibilidade, o protótipo permite que a autenticação seja efetuada tanto com pacotes *TCP* ou *UDP* na troca de mensagens entre o servidor e o cliente. Utilizou-se o módulo *Perl Net::RawIP* para a troca de pacotes no *KnockID*. Este módulo pode ser usado para criar, manipular e enviar pacotes *IP*, e tem suporte para o envio de pacotes nos protocolos de transporte *TCP* e *UDP* (o pacote também

pode ser enviado como *ICMP*, ainda que não fora implementado no presente protótipo). Para as mensagens *TCP* foi utilizado pacotes *"TCP Raw"*, uma vez que não tem-se *handshake* entre o cliente e o servidor. Pacotes *"TCP Raw"* correspondem a pacotes no modo *Raw*, modo cujo processamento de encapsulamento/desencapsulamento do pacote *TCP* é ignorado, ou seja, todas as coisas que o *Kernel* do sistema operacional faz na pilha *TCP/IP* é deixado a cargo da aplicação. Esse característica de envio de pacotes *"TCP Raw"* traz o inconveniente da necessidade de uso da senha de administrador (*root*) na execução do KnockID quando opera neste modo, já que o *Kernel* do sistema operacional *Linux* não permite a manipulação do modo *Raw* para usuários comuns. Por padrão, o protótipo é configurado com pacotes *UDP*, tendo em vista o menor *overhead*, maior simplicidade e a não necessidade do uso de senha de administrador.

O módulo *Perl Net::RawIP* também tem a funcionalidade de monitorar a chegada de pacotes, o que foi empregado na construção do *KnockIDclient*. O uso dessa funcionalidade garante maior simplicidade ao programa cliente, já que não é necessário carregar um outro módulo *Perl* para monitorar o tráfego de rede do cliente, quando este aguarda o recebimento do desafio enviado pelo *verificador*.

Embora o módulo *Net::RawIP* seja também utilizado no *KnockIDserver* para a criação e envio de mensagens do protocolo, a funcionalidade de monitoração do tráfego de rede, empregado no *KnockIDclient*, não foi utilizada. Essa funcionalidade foi proporcionada ao *KnockIDserver* pelo módulo *Perl Net::Pcap*. A aplicação do *Net::Pcap* se deve ao fato de que a monitoração do tráfego de rede proporcionada por este módulo ser mais consistente e robusta, o que é mais apropriado para o processo do *KnockID* no servidor que deve monitorar o tráfego de vários clientes.

O *Firewall* utilizado para manter todas as portas fechadas foi o *Iptables*, nativo do sistema operacional *Linux*. Embora o prótotipo *KnockID* foi desenvolvido no sistema operacional *Linux*, este pode ser executado em qualquer sistema operacional que execute a linguagem *Perl* e tenha o *Firewall Iptables* instalado. O *Firewall* é configurado, por padrão, para rejeitar qualquer tentativa de comunicação ao servidor, sem enviar nenhuma informação à origem.

O processo do *KnockID* no servidor monitora os pacotes que chegam na porta 52123 (*TCP* ou *UDP*, conforme a configuração utilizada para a instância em execução) e os submetem ao protocolo. Somente há um retorno do *KnockIDserver* a um pa-

cote recebido se as verificações em cada etapa do protocolo forem satisfeitas. Caso contrário, o *KnockIDserver* apenas descarta o pacote, dando a característica que o serviço *TCP/UDP* não está habilitado.

Um possível questionamento é se protocolo *KnockID* é mesmo baseado em camada de rede, uma vez que o protótipo utiliza pacotes da camada de transporte (*TCP* ou *UDP*) para transportar as mensagens. O uso da camada de transporte é por conveniência, pois os dados de autenticação podem ser transportados pela camada de rede, como, por exemplo, utilizando o protocolo *ICMP*. De fato, o uso de *TCP/UDP* ajuda a resolver problemas práticos, como *Firewalls* ou roteadores configurados para rejeitar *ICMP* ou pacotes *IP* anômalos. De qualquer modo, o *KnockID* é um protocolo baseado em camada de rede devido ao fato de que todas as portas da camada de transporte estão fechadas pelo *Firewall* no servidor.

Como disposto no protocolo, o cliente pode solicitar a execução de um comando no servidor (um comando *shell*, por exemplo) ou a abertura de uma porta no *Firewall* do servidor para posterior conexão do cliente. Se um cliente solicitar a execução de um comando no servidor, o comando é executado após a autenticação, sem que haja um retorno ao cliente, ainda que a autenticação seja bem sucedida. Se um cliente solicitar a abertura de uma porta no *Firewall*, a porta é aberta para o endereço *IP* do cliente e também é agendada a exclusão da regra de abertura após o tempo de 120 segundos, aumentando assim a segurança do sistema. Para o agendamento da exclusão foi utilizado o módulo *Perl Schedule::At*. Assim, o cliente tem dois minutos para estabelecer a conexão. Uma vez estabelecida a conexão, não haverá o problema de cair com o agendamento da exclusão da regra, já que há uma regra que permite o tráfego de conexões pré-estabelecidas.

No servidor, os dados dos clientes, como identificação e respectivas chaves, foram armazenados em arquivos de texto. Um arquivo contém a relação de todos os $ID's$ dos clientes. Logo, quando um cliente solicitar a autenticação ao servidor, basta o *KnockIDserver* verificar se o ID está presente nesse arquivo. O servidor mantém uma pasta com as chaves públicas de cada cliente, identificadas pelo respectivo ID. Assim, o *KnockIDserver* pode recuperar os arquivos de chave inerentes a cada cliente e realizar os procedimentos necessários do protocolo *KnockID*. Tais arquivos são armazenados de forma segura no servidor, com as configurações de restrição de acesso do sistema operacional.

Ambos *KnockIDclient* e *KnockIDserver* funcionam com arquivos de configuração, o que possibilita a alteração de alguns parâmetros somente com a edição desses arquivos. Por exemplo, a porta de conexão 52123 pode ser facilmente alterada editando-se os arquivos de configuração do cliente e do servidor. O arquivo de configuração do cliente permite a alteração dos dados de autenticação utilizados na primeira mensagem do protocolo, como a identificação, a especificação se a solicitação é para a execução de comando ou abertura de *Firewall* e a definição do comando a ser executado ou da porta de *Firewall* a ser aberta.

5.2 TESTES E RESULTADOS

O protótipo do *KnockID*, implementado de acordo com as explicações da seção anterior, funcionou a contento, executando o protocolo proposto, de acordo com o esperado. Para ilustrar o sucesso da implementação, o protótipo foi testado em alguns cenários. Inicialmente o *KnockID* foi verificado com a execução dos programas *KnockIDclient* e *KnockIDserver* no mesmo computador, utilizando-se a interface *loopback* para a comunicação. Posteriormente, os programas *KnockIDclient* e *KnockIDserver* foram testados em computadores diferentes dentro de uma rede local. Será apresentado nesta seção um teste mais realístico, onde os programas *KnockIDclient* e *KnockIDserver* estão em computadores localizados em redes diferentes, com o tráfego passando pela *Internet*. A figura 5.1 esboça o cenário desse último teste.

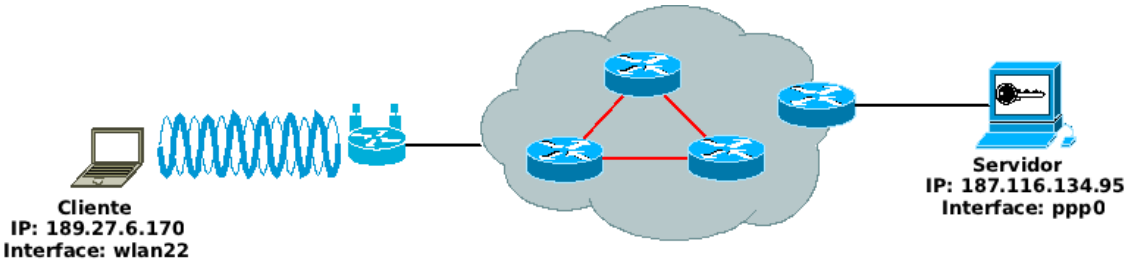


Figura 5.1: Cenário de teste do *KnockID*.

No cenário de teste da figura 5.1, o cliente está localizado numa rede local sem fio, similar à uma rede residencial. O cliente tem um *IP* Privado (192.168.25.7) na interface *wlan22*, ocorrendo uma tradução *NAT* no roteador residencial conectado ao provedor *Global Village Telecom* (*GVT*). No entanto, como o presente trabalho não trata da questão da autenticação em ambientes com *NAT*, para fins de simplificação, será considerado como endereço do cliente o *IP* Público do roteador residencial (*IP* cliente = 189.27.6.170). Para simular a situação de um servidor na Internet com *IP* Público, configurou-se um computador com uma conexão de dados celular (*3G*) da operadora

Vivo. Assim o servidor utiliza a interface *ppp0* para se comunicar na *Internet*, a qual tem *IP* Público (*IP* servidor = 187.116.134.95). Observe que neste cenário onde o cliente e o servidor tem provedores distintos, a comunicação pode ocorrer por vários caminhos, podendo passar por várias redes, roteadores e *Firewall's*, dependendo de vários fatores como congestionamento na rede, rotas em roteadores, quedas de *links*, tal como ocorre na *Internet*.

Antes de executar o *KnockID* no servidor, será verificada a configuração inicial do *Firewall*. A configuração padrão do *Firewall* é deixar o tráfego todo liberado, permitindo qualquer conexão, como apresentado na figura 5.2.

```
# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Figura 5.2: Configuração inicial do *Firewall*.

O primeiro passo para a verificação do *KnockID* é a execução do *KnockIDserver* no servidor. A figura 5.3 mostra a inicialização do *KnockIDserver* em linha de comando. Na inicialização, as configurações do arquivo de configuração do KnockIDserver são lidas e carregadas, o *Firewall* é configurado de forma dinâmica para bloquear todo o tráfego entrante no servidor e o processo do *KnockID* no servidor ativa a monitoração do tráfego de rede.

```
# perl knockIDserver
debug [+] Starting knockidserver
debug [+] Firewall configured.
debug [+] linklayer: ppp cooked interface detected
debug [+] knockidserver started
debug [+] Starting to listen on ppp0
```

Figura 5.3: Inicialização do KnockIDserver.

É bom ressaltar que o *Firewall* é configurado de forma dinâmica pelo *KnockIDserver* para bloquear toda requisição de acesso ao servidor. Os comandos utilizados pelo *KnockIDserver* para a configuração dinâmica do *Firewall* são mostrados na figura 5.4. A duas primeiras regras alteram a política de acesso no *Firewall*, bloqueiando todo o tráfego entrante ou roteado pelo servidor. A terceira regra permite o tráfego das conexões já estabelecidas. A figura 5.5 mostra o resultado dessa configuração.

Com *KnockIDserver* ativado no servidor, foi testado no cliente uma solicitação de conexão *ssh* ao servidor. A figura 5.6 apresenta a tentativa de conexão *ssh* no endereço

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -I INPUT -i ppp0 -m state --state RELATED,ESTABLISHED -j ACCEPT
```

Figura 5.4: Comandos para configuração dinâmica do *Firewall* pelo *KnockIDserver*.

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Figura 5.5: Configuração do *Firewall* após inicialização do *KnockIDserver*.

IP do servidor. Como esperado, a solicitação *ssh* do cliente não foi atendida, esgotando o tempo de resposta do servidor.

```
dio@magi:/home/client$ ssh 187.116.134.95
ssh: connect to host 187.116.134.95 port 22: Connection timed out
```

Figura 5.6: Tentativa de conexão *ssh* do cliente ao servidor sem autenticação.

Foi utilizado o programa de monitoração de tráfego de rede *Wireshark* para verificar a comunicação entre cliente e servidor. O *Wireshark* foi habilitado para monitorar o tráfego da porta *ppp0* do servidor. A tentativa anterior de conexão *ssh* do cliente foi monitorada pelo *Wireshark* e apresentada na figura 5.7 na forma de gráfico de fluxo. Observe que chegaram várias solicitações *ssh* no servidor, porém este não enviou nenhuma resposta, como esperado.

Time	189.27.6.170 → 187.116.134.95	Comment
6,448	48342 > ssh [SYN] S	TCP: 48342 > ssh [SYN] Seq=0 Win=14600 Len=0 MSS=1452 SACK_PERM=1 TSV=9721818 TSER=0 WS=3
7,392	48342 > ssh [SYN] S	TCP: 48342 > ssh [SYN] Seq=0 Win=14600 Len=0 MSS=1452 SACK_PERM=1 TSV=9722068 TSER=0 WS=3
9,388	48342 > ssh [SYN] S	TCP: 48342 > ssh [SYN] Seq=0 Win=14600 Len=0 MSS=1452 SACK_PERM=1 TSV=9722569 TSER=0 WS=3
13,440	48342 > ssh [SYN] S	TCP: 48342 > ssh [SYN] Seq=0 Win=14600 Len=0 MSS=1452 SACK_PERM=1 TSV=9723572 TSER=0 WS=3
21,452	48342 > ssh [SYN] S	TCP: 48342 > ssh [SYN] Seq=0 Win=14600 Len=0 MSS=1452 SACK_PERM=1 TSV=9725576 TSER=0 WS=3
37,572	48342 > ssh [SYN] S	TCP: 48342 > ssh [SYN] Seq=0 Win=14600 Len=0 MSS=1452 SACK_PERM=1 TSV=9729584 TSER=0 WS=3

Figura 5.7: Gráfico de fluxo das solicitações *ssh* na tentativa de conexão do cliente ao servidor sem autenticação.

Para atestar o funcionamento do protocolo proposto, foi executado o *KnockIDclient* a fim de autenticar o cliente e solicitar a abertura do serviço *ssh* no *Firewall*. A figura 5.8 ilustra a execução do *KnockIDclient*. Tal como no servidor, o cliente inicia sua execução com a leitura e carregamento do arquivo de configuração, o qual tem os dados de autenticação necessários para a execução do protocolo. A figura 5.8 mostra de maneira explícita os passos de construção e envio da mensagem de requisição, recebimento

do desafio, envio da resposta ao servidor e o SID da conexão. Observe que o SID é composto por caracteres não conhecidos. Essa característica é devida ao fato da utilização de caracteres não-imprimíveis na geração das aleatoriedades CH_v e CH_p que compõem o SID, aumentando a faixa de caracteres possíveis, o que acarreta em maior segurança (menor probabilidade do adversário adivinhar as aleatoriedades).

```
root@magi:/home/client/prototipo# perl knockIDclient
debug [+] Local port: 51361
debug [+] Created signature: 128 bytes
debug [+] Created authentication message: 256 bytes
debug [+] Sending 256 bytes message to 187.116.134.95:52123
debug [+] Received challenge: 16 bytes
debug [+] Generate random string: 16 bytes
debug [+] Signature for the answer to server: 128 bytes
debug [+] Sending 144 bytes message to 187.116.134.95:52123
[+] SID: ◆◆W~◆
              XQ◆◆W◆◆/◆
                        ię◆◆D◆◆◆◆◆◆◆◆g
```

Figura 5.8: Execução do KnockIDclient.

A figura 5.9 mostra o recebimento da requisição de autenticação e solicitação de abertura do serviço *ssh* do cliente pelo *KnockIDserver*. Pode ser visto os passos do protocolo *KnockID* no servidor. Na mensagem de requisição o *KnockIDserver* decifra a mensagem com a chave privada do servidor e realiza a verificação da identidade do cliente e também da assinatura enviada nessa mensagem. Como as verificações foram positivas, o *KnockIDserver* gera a aleatoriedade de desafio (CH_v) e a envia ao cliente. O *KnockIDserver* recebe a resposta do desafio e verifica se a assinatura do desafio é válida. Uma vez verificada de forma positiva a assinatura do desafio, o *KnockIDserver* gera o SID, correspondente ao que foi gerado no *KnockIDclient*, procedendo com a liberação do acesso solicitado no Firewall (acesso ao serviço *ssh*) e agendamento da exclusão da regra do *Firewall*.

```
debug [+] Initiating protocol for 189.27.6.170
debug [+] Message decrypted
debug [+] ID checked
debug [+] Signature checked
debug [+] Sending challenge...
debug [+] Generate random string 16 bytes
debug [+] Sending 16 bytes message to 189.27.6.170: 51361
debug [+] Checking response of challenge for 189.27.6.170
debug [+] Signature of challenge response checked!
debug [+] SID for connection of 189.27.6.170: ◆◆W~◆
                                                XQ◆◆W◆◆/◆
                                                          ię◆◆D◆◆◆◆◆◆◆◆g
debug [+] Access granted
debug [+] Job schedule sucess
```

Figura 5.9: Autenticação e atendimento de solicitação de um cliente no KnockIDserver.

A fase de autenticação entre o cliente e servidor também foi capturada pelo *Wireshark*. A figura 5.10 mostra o gráfico de fluxo das mensagens entre o cliente e servidor. Pode-

se verificar que foi utilizado o protocolo *UDP* para o transporte das mensagens de autenticação, onde o cliente utiliza a porta 51361 como origem e solicita a autenticação na porta 52123 (porta monitorada pelo *KnockIDserver*). A primeira mensagem é a mensagem de requisição do cliente, a segunda mensagem é o envio do desafio pelo servidor e a última mensagem é a resposta do cliente para efetuar a autenticação.

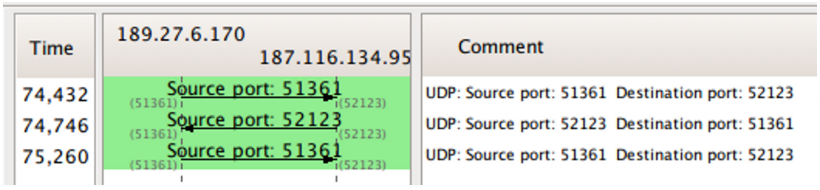


Figura 5.10: Gráfico de fluxo da fase de autenticação do cliente no servidor.

O resultado do sucesso de autenticação é percebido com a alteração da regra do *Firewall* no servidor, ilustrado na figura 5.11. Observe que a primeira regra permite a conexão do cliente, na porta do serviço *ssh* (porta 22). Nesse exemplo o cliente requisitou o acesso na porta 22, porém, por meio da edição do arquivo de configuração, o cliente poderia ter requerido qualquer outra porta TCP/UDP. O cliente tem até 2 minutos para realizar a conexão subsequente e realizar a conexão *ssh*. Após o intervalo de 2 minutos a regra é retirada automaticamente por um agendamento no servidor, mas, se o cliente tiver realizado a conexão ao serviço *ssh* durante esse intervalo de tempo e mantê-la ativa, a conexão não irá cair, uma vez que na configuração inicial do *Firewall* é previsto aceitar o tráfego de conexões já estabelecidas.

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  189.27.6.170.dynamic.adsl.gvt.net.br  anywhere            tcp dpt:ssh flags:FIN,SYN,RST,ACK/SYN
ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Figura 5.11: Configuração do *Firewall* após autenticação do cliente.

Para finalizar, foi efetuada uma nova tentativa de conexão *ssh* logo após a fase de autenticação. A figura 5.12 ilustra a tentativa de conexão *ssh* no endereço *IP* do servidor, imediatamente após a fase de autenticação. Nessa tentativa a solicitação *ssh* do cliente foi atendida como esperado, já a foi adicionada uma regra no *Firewall* do servidor liberando o acesso *ssh* para o endereço *IP* do cliente. A figura 5.13 mostra o gráfico de fluxo das mensagens entre o cliente e servidor dessa última tentativa de conexão *ssh*, capturada pelo *Wireshark*. Esse gráfico de fluxo da figura 5.13 retrata exatamente a conexão *ssh* da figura 5.12, porém na forma de pacotes *TCP*. Pode-se

notar a fase de estabelecimento de conexão, a negociação de chaves do *ssh* para a autenticação na aplicação e a finalização da conexão pelo cliente.

```
dio@magi:/home/client$ ssh 187.116.134.95
The authenticity of host '187.116.134.95 (187.116.134.95)' can't be established.
ECDSA key fingerprint is fa:71:d3:67:27:a2:2a:c3:e1:51:9c:f3:7e:b3:12:ae.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '187.116.134.95' (ECDSA) to the list of known hosts.
dio@187.116.134.95's password:
Welcome to Ubuntu 11.04 (GNU/Linux 2.6.38-13-generic i686)

 * Documentation:  https://help.ubuntu.com/

New release 'oneiric' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Aug 22 10:13:41 2012 from magi.home
dio@Scaflowne:~$ exit
sair
```

Figura 5.12: Tentativa de conexão *ssh* do cliente ao servidor após autenticação.

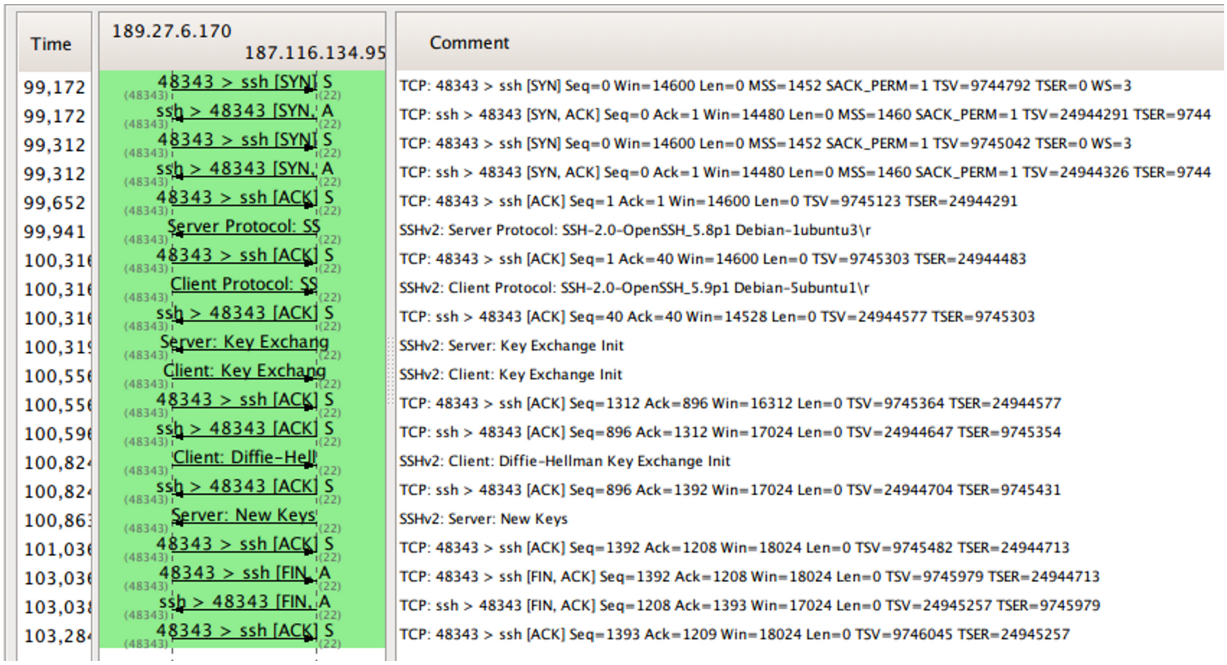


Figura 5.13: Gráfico de fluxo das solicitações *ssh* na tentativa de conexão do cliente ao servidor após autenticação.

5.3 SUMÁRIO

O desenvolvimento do protótipo *KnockID* na linguagem *Perl* permitiu uma implementação rápida, sobretudo pelo fato da linguagem de programação escolhida ser de fácil aprendizado e apresentar características que favorecem o desenvolvimento, como diversos módulos (bibliotecas) com códigos para os mais variados usos. Foram implementados dois programas - o *KnockIDclient* e o *KnockIDserver* - para desempenhar as ações do protocolo proposto. De uma forma geral, o protótipo *KnockID* implementado

nos dois programas: *KnockIDclient* e *KnockIDserver* funcionaram perfeitamente. O *KnockIDserver* atuou de forma integrada ao *Firewall*, bloqueando o acesso às aplicações e liberando o mesmo após a autenticação de um cliente. Os testes realizados não tiveram como finalidade a avaliação de performance do protótipo, mas sim a viabilidade de implementação do protocolo *KnockID*. Nesse quesito, o protótipo mostra através do exemplo ilustrado nas figuras deste capítulo que o protocolo KnockID é perfeitamente implementável. No próximo capítulo serão apresentadas as conclusões deste trabalho e recomendações de trabalhos futuros que podem auxiliar na questão de autenticação.

6 CONCLUSÕES E RECOMENDAÇÕES

O *KnockID* representa uma mudança de paradigma para a autenticação de usuários tanto pela sua proposta de autenticar usuários baseando-se na camada de rede, bem como por se basear em um protocolo com segurança demonstrável. É uma alternativa viável aos outros métodos de autenticação (autenticação na camada de aplicação, *VPN*, *PortKnocking* e *SPA*) e oferece uma camada extra de segurança para a proteção do servidor de serviços contra ataques ‘‘*0-day*”, *port scanning*, *fingerprinting*.

A segurança proporcionada pelo *KnockID* é um dos grandes diferenciais deste trabalho, garantida pela análise de segurança do protocolo do Bellare, realizada em Bellare et al. (2001). Tal fator dá maior robustez e confiança ao *KnockID*, uma vez que este se baseia num protocolo com segurança demonstrável, contrastando com as soluções baseadas em heurística. Além disso, deve-se ressaltar que o *KnockID* herda a segurança contra ataques de repetição, *man-in-the-middle* e ataques de *reset* por se basear no protocolo do Bellare.

Outro mérito do *KnockID* é a eliminação de ações de *script kiddies* e ataques “*0-day*”, graças ao seu modo de funcionamento. Com o bloqueio de qualquer requisição externa ao servidor nas regras de *Firewall*, os diversos programas *script kiddies* que realizam varreduras para verificação de serviços vulneráveis e enviam pacotes com *exploits* são anulados. O mesmo vale para os ataques “*0-day*”, pois os atacantes não tem acesso para verificar qual serviço é executado no servidor, e, mesmo que saibam que algum serviço em execução no servidor é vulnerável, não conseguem acessá-lo devido ao bloqueio do *Firewall*.

A utilização de criptografia assimétrica empregada no *KnockID* facilita na questão de gerenciamento dos usuários, sendo possível desenvolver perfis customizados com diferentes níveis de permissão de acesso aumentando a segurança do servidor, em contraste ao compartilhamento de chaves simétricas, as quais podem acarretar a comprometimento de segurança de todo o sistema em caso de extravio. Embora os dados dos clientes foram armazenados em forma de arquivos no protótipo desenvolvido, dificultando o desenvolvimento de perfis customizados, já que estes deveriam ser implementados dentro do programa, o desenvolvimento desse sistema utilizando banco de dados facilitaria

a organização dos dados de clientes e abriria várias possibilidades quanto ao arranjo de perfis customizados de clientes.

Comparando o *KnockID* com as soluções *PortKnocking* e *SPA*, percebe-se que o *KnockID* apresenta alguns aspectos mais vantajosos. Por exemplo, o *KnockID* não é tão ruidoso, não apresenta problema se houver chegada de pacotes fora de ordem na autenticação e não apresenta *overhead* impraticável ao utilizar criptografia como visto no *PortKnocking*. Em relação ao *SPA*, o *KnockID* é uma alternativa mais segura, já que o protocolo baseado em desafio/resposta é mais consistente e seguro em comparação ao envio de um único pacote de autenticação, uma vez que este último pode ser interceptado e permite a personificação do atacante.

Em geral, o *KnockID* é uma solução simples e eficiente para a tarefa de autenticação. Os seus pontos fortes são a simplicidade e a segurança dos serviços do servidor advinda de protocolos criptográficos com segurança demostrável. A simplicidade é percebida principalmente na implementação, já que o *KnockID* é implementado facilmente com o uso de bibliotecas criptográficas comuns. Tal fator auxilia numa possível auditoria ao código do programa, pois o código se torna fácil de entender, e, também favorece a integração transparente a qualquer ambiente de rede executando aplicativos legados.

O *KnockID* pode ser utilizado para diferentes necessidades. Por exemplo, poderia se pensar na utilização do *KnockID* numa rede central de serviços de autenticação de usuário. Nesse cenário, todo o cliente deve se autenticar num primeiro momento para posterior acesso à aplicação desejada, o que oculta as aplicações de rede sensíveis a ataque ou completamente inseguros e ainda age como uma camada extra de segurança, no aspecto de que o atacante deve primeiro quebrar a segurança de autenticação do *KnockID* para depois quebrar a segurança das aplicações.

6.1 RECOMENDAÇÕES PARA PESQUISAS FUTURAS

Algumas propostas podem ser elaboradas para trabalhos futuros seguindo a idéia apresentada neste trabalho. Uma proposta é a implementação otimizada do *KnockID*, já que a eficiência da solução aqui apresentada não foi objeto de preocupação. Embora a linguagem Perl seja adequada para a implementação de um protótipo, para ambientes de produção é recomendável a utilização de linguagens mais eficientes, como a linguagem C. Além da linguagem de programação, outro aspecto que pode ser otimizado é a primitiva de assinatura. No presente trabalho utilizou-se o esquema de assinatura

PKCS #1 v1.5 do *RSA*, haja vista a facilidade de encontrar o módulo *Perl* correspondente. No entanto, o esquema de assinatura seria otimizado se fosse utilizado um esquema baseado em curvas elípticas, o que diminuiria consideravelmente o tamanho final da assinatura.

Ainda pensando em aperfeiçoar a implementação do *KnockID*, uma sugestão para sua melhora é a utilização de banco de dados para a gerência das identificações dos clientes e das respectivas chaves. Um banco de dados dedicado para esse fim melhoraria a gerência de *ID's* e chaves, e também permitiria a criação de perfis customizados para cada cliente. Esses perfis poderiam restringir acessos de acordo com a identificação do usuário, de modo que, o *KnockID* verificaria se o perfil do cliente autoriza o acesso requisitado.

Com uma nova implementação otimizada do *KnockID*, pode-se pensar na realização de uma bateria de testes a fim de verificar a eficiência do protocolo e também da implementação. Os testes consistiriam na criação de vários clientes, de forma a submeter o servidor à inúmeras requisições de autenticação simultâneas. Tal teste permitiria verificar o limite de conexões suportado pela implementação e o tempo de resposta para cada requisição. Outro teste recomendado seria simular o comportamento de um adversário tentando quebrar o protocolo. Também recomenda-se realizar um teste num servidor com múltiplas interfaces a fim de verificar o comportamento do *KnockID* com múltiplas instâncias no servidor (uma para cada interface protegida no servidor). Após testes de funcionalidade básicas sugere-se a ativação do *KnockID* num servidor em produção para o verificar o comportamento da implementação num ambiente "vivo" e, assim, obter uma avaliação mais realística dos impactos de utilização da solução proposta.

Os problemas conhecidos no *PortKnocking* e *SPA* de segurança da conexão subsequente e autenticação de usuários em ambiente *NAT* ainda continuam abertos. Diante disso, um trabalho futuro seria propor novas métodos que utilizem o *KnockID* para solucionar tais problemas abertos. Ou ainda, propor novas abordagens, tal como o *KnockID*, que utilizem métodos com segurança demostrável com o propósito de mitigar esses problemas.

Este trabalho mostrou que a utilização de protocolos com segurança demonstrável para a solução de problemas de autenticação favorece a segurança do sistema. Nesse contexto, sugere-se também a análise de outros protocolos e métodos com segurança demonstrável que possam ser utilizados com finalidade de autenticação de usuários em

camada de rede com maior eficiência e facilidade de gerência, dentre eles protocolos eficientes de provas de *zero-knowledge* não interativas. Outra direção interessante no âmbito de segurança demonstrável é a formalização de protocolos de autenticação de usuários.

No protocolo proposto somente há a autenticação do cliente no servidor. Entretanto, o cliente nunca terá certeza se o servidor é verdadeiro, pois este não é autenticado. Uma proposta para trabalho futuro é a adequação desse protocolo para também autenticar o servidor. Uma possível solução seria a execução de duas instâncias do protocolo, onde uma instância tem a função de autenticar o cliente e a outra o servidor. Porém, nesse caso deve-se verificar se a segurança advinda do protocolo se mantém no quadro de composição (execução de mais de uma instância). Uma prova de segurança utilizando o *framework Universally Composable* (*UC*) (CANETTI, 2000) seria um grande resultado que mostraria a "composabilidade" do protocolo.

REFERÊNCIAS BIBLIOGRÁFICAS

AYCOCK, J. *Computer Viruses and Malware*. [S.l.]: Springer, 2006.

BARHMAN, P. et al. Techniques for lightweight concealment and authentication in ip networks. *Intel Research Berkeley, Technical Report IRB-TR-02-009*, July 2002.

BELLARE, M. et al. Identification protocols secure against reset attacks. In: B. PFITZMANN ED. *Advances in Cryptology - Eurocrypt 2001*. [S.l.]: Springer-Verlag, 2001. LNCS 2045, p. 493–509.

BELLARE, M.; POINTCHEVAL, D.; ROGAWAY, P. Authenticated key exchange secure against dictionary attacks. In: B. PRENEEL ED. *Advances in Cryptology - Eurocrypt 2000*. [S.l.]: Springer-Verlag, 2000. LNCS 1807, p. 493–509.

BELLARE, M.; ROGAWAY, P. *Introduction to Modern Cryptography*. Computer Science and Engineering, May 2005. Disponível em: <http://goo.gl/fy3w8>. Acesso em: 25/11/2011.

CANETTI, R. Universally composable security: A new paradigm for criptography protocols. *Cryptography ePrint Archive, Report 2000/067, revised Jan2005 and Dec 2005.*, 2000.

COTTON, M. et al. *RFC 6335: Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry*. [S.l.]: IETF, 2011.

DEGRAFF, R. *Enhancing Firewalls: Conveying User and Application Identification to Network Firewalls*. Dissertação (Mestrado) — Department of Computer Science, 2007. Disponível em: <http://goo.gl/SzRWX>. Acesso em: 14/09/2011.

DEGRAFF, R.; AYCOCK, J.; JACOBSON, M. Improved port knocking with strong authentication. *Proc. ACSAC*, p. 409–418, 2005. Disponível em: <http://goo.gl-/BTBGA>. Acesso em: 14/09/2011.

ENGELKE, C. *Definition: Zero-Day Exploit*. SearchSecurity - TechTarget, March 2004. Disponível em: <http://goo.gl/BV8Iv>. Acesso em: 26/10/2011.

ERICKSON, J. *The Art of Exploitation*. 2. ed. [S.l.]: No Starch Press, 2008.

FERGUSON, N.; SCHNEIER, B. *A Cryptographic Evaluation of IPsec*. [S.l.], 2000.

FYODOR. *The Art of Port Scanning*. Phrack Magazine, September 1997. Disponível em: <http://goo.gl/akYVs>. Acesso em: 26/10/2011.

JEANQUIER, S. *An Analysis of Port Knocking and Single Packet Authorization*. Dissertação (Mestrado) — Royal Holloway College - University of London, 2006. Disponível em: <http://goo.gl/hKB4x>. Acesso em: 14/09/2011.

KATZ, J.; LINDELL, Y. *Introduction to modern cryptography*. [S.l.]: Chapman & Hall/CRC, 2007.

KENT, S.; SEO, K. *Security Architecture for the Internet Protocol*. [S.l.]: IETF, dec 2005. RFC 4301 (Proposed Standard). (Request for Comments, 4301).

KOCH, W. *The GNU Privacy Guard*. 2007. Disponível em: <http://goo.gl/BPZB>. Acesso em: 28/11/2011.

KRZYWINSKI, M. Port knocking - network authentication across closed ports. *SysAdm Magazine*, v. 12, n. 6, p. 12–17, june 2003. Disponível em: <http://goo.gl-/cz8n5>. Acesso em: 06/10/2011.

KRZYWINSKI, M. *Port Knocking Implementations*. 2009. Disponível em: <http:/-/goo.gl/Q0MTD>. Acesso em: 14/09/2011.

KUROSE, J. F.; ROSS, K. W. *Redes de Computadores e a Internet - uma abordagem top-down*. 3rd. ed. [S.l.]: Pearson Addison Wesley, 2006.

LIEW, J.-H. et al. One-time knocking framework using spa and ipsec. In: *Education Technology and Computer (ICETC), 2010 2nd International Conference on*. [S.l.: s.n.], 2010. v. 5, p. V5–209 –V5–213.

LINN, J. *RFC 1421: Privacy Enhancement for Internet Eletronic Mail*. [S.l.]: IETF, 1993.

MENEZES, A. J.; OORSCHOT, P. C. van; VANSTONE, S. A. *Handbook of Applied Cryptography*. [S.l.]: CRC Press, 2006.

MICHAEL, R. *Service Cloaking and Anonymous Access: Combining Tor with Single Packet Authorization (SPA)*. DEFCON, May 2006. Disponível em: <http://goo.gl-/VjACN>. Acesso em: 29/11/2011.

MICHAEL, R. Single packet authorization with fwknop. *;login: - The USENIX Magazine*, p. 63–69, 2006. Disponível em: <http://goo.gl/I9jCS>. Acesso em: Acessado 04/10/2011.

MICHAEL, R. *fwknop - Single Packet Authorization and Port Knocking*. 2011. Disponível em: <http://goo.gl/dFgw7>. Acesso em: 14/09/2011.

MIKLOSOVIC, S. *PA018 - Term Project - Port knocking enhancements*. Dissertação (Mestrado) — Faculty of Informatics, June 2011. Disponível em: <http://goo.gl-/6nAo0>. Acesso em: 06/10/2011.

NETO, U. *Dominando Linux Firewall IPTables*. Rio de Janeiro: Editora Ciência Moderna, 2004.

POSTEL, J. *RFC 791: Internet Protocol: DARPA Internet Program Protocol Especification*. California: IETF, 1981.

POSTEL, J. *RFC 793: Transmission Control Protocol: DARPA Internet Program Protocol Especification*. California: IETF, 1981.

REKHTER, Y. et al. *RFC 1918: Address Allocation for Private Networks*. [S.l.]: IETF, 1996.

SCHNEIER, B. *Applied Cryptography: protocols algorithms and source code in C*. 2nd. ed. [S.l.]: John Wiley & Sons, 1996.

SHAMIR, A. *How to Share a Secret*. 1979.

SHANNON, C. Communication theory of secrecy systems. *Bell System Technical Journal*, v. 28, p. 656–715, 1949.

SNADER, J. C. *VPNs Illustrated: Tunnels, VPNs and IPSec*. [S.l.]: Addison-Wesley Professional, 2005.

SOCOLOFSKY, T. J.; KALE, C. J. *RFC 1180: A TCP/IP Tutorial*. [S.l.]: IETF, 1991.

SRISURESH, P.; HOLDREGE, M. *RFC 2663: IP Network Address Translator (NAT) Terminology and Considerations*. [S.l.]: IETF, 1999.

STALLINGS, W. *Criptografia e Segurança de Redes - Princípios e Práticas*. 4. ed. São Paulo: Pearson Prentice Hall, 2008.

STEVENS, W. R. *TCP/IP Illustrated: the protocols*. 1st. ed. [S.l.]: Pearson Addison Wesley, 1993.

SYVERSON, P. A taxonomy of replay attacks. In: *In Proceedings of the 7th IEEE Computer Security Foundations Workshop*. [S.l.]: Society Press, 1994. p. 187–191.

TARIQ, M.; BAIG, M. S.; SAEED, M. T. Associating the authentication and connection-establishment phases in passive authorization techniques. In: INTERNATIONAL ASSOCIATION OF ENGINEERS. *Proceedings of the World Congress on Engineering 2008 Vol I, WCE '08, July 2 - 4, 2008, London, U.K.* [S.l.]: Newswood Limited, 2008. (Lecture Notes in Engineering and Computer Science), p. 570–575. ISBN 978-988-98671-9-5.

ULBRICH, H. C.; VALLE, J. D. *Universidade H4ck3r*. 4. ed. [S.l.]: Digerati Books, 2004.

VAUDREUIL, G. M.; FREED, N.; BORENSTEIN, N. S. *RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. [S.l.]: IETF, 1996.