PAULO MANOEL MAFRA

COMUNICAÇÃO SEGURA NA COMPOSIÇÃO DE IDSs E SEUS CUSTOS

FLORIANÓPOLIS
2006

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

COMUNICAÇÃO SEGURA NA COMPOSIÇÃO DE IDSs E SEUS CUSTOS

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica.

Paulo Manoel Mafra

Florianópolis, Agosto de 2006.

COMUNICAÇÃO SEGURA NA COMPOSIÇÃO DE IDSs E SEUS CUSTOS

Paulo Manoel Mafra

'Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica, Área de Concentração em *Automação e Sistemas*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Joni da Silva Fraga, Dr.
Orientador

Nelson Sadowski, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Joni da Silva Fraga, Dr.
Presidente

Carlos Barros Montez, Dr.

Frank Siqueira, Dr.

Ricardo Felipe Custodio, Dr.

Dedico este trabalho à Ayrton Bento Mafra, que nunca mediu esforços para me proporcionar a melhor educação possível.

AGRADECIMENTOS

Agradeço aos meus pais Ayrton Bento Mafra e Maria Rosa Mosimann Mafra, pelo constante amor e carinho. Agradeço também aos meus irmãos Áurea Maria Mafra e Álvaro Luis Mafra pelo apoio e incentivo nos meus estudos.

Um agradecimento especial à Jerusa Marchi, que sempre me apoiou, incentivou, deu carinho e amor. Sem ela, o caminho percorrido seria muito mais difícil.

Agradeço ao meu orientador, professor Joni da Silva Fraga, que tanto me ensinou ao longo destes anos; aos demais professores do DAS, que de alguma forma contribuíram para a minha formação.

Gostaria de agradecer aos meus amigos: Luciano Rottava, que me ensinou muito do que eu sei sobre Unix; Rafael Obelheiro, pelas constantes dicas; José Brandão, pela ajuda com este trabalho; Patrícia Pena, pela ajuda na disciplina do Werner; Eder Gonçalves, meu companheiro de depósito. Aos demais amigos de laboratório: Tatiana Garcia, Emerson Mello, Marcos Vallim, Rodrigo Sumar, Wagner Dantas, Robson Costa, Rodrigo Raimundo (spyro), Ricardo Grutzmacher, Alberto Pavim, Karen Farfan (tratante), Fernando Passold, Carlos Brandão e Karina Barbosa - pela amizade.

Por fim, às pessoas que me ajudaram, mas que estou esquecendo de citá-las.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

COMUNICAÇÃO SEGURA NA COMPOSIÇÃO DE IDSs E SEUS CUSTOS

Paulo Manoel Mafra

Agosto/2006

Orientador: Joni da Silva Fraga, Dr.
Área de Concentração: Automação e Sistemas
Palavras-chave: Segurança Computacional, Sistemas Distribuídos
Número de Páginas: xx + 105

Os sistemas de detecção de intrusão geralmente são projetados para funcionar solitários em redes locais. Estes sistemas não prevêem a integração com outras ferramentas de detecção de intrusão, nem o uso da infra-estrutura da Internet para correlação de eventos, reduzindo o seu escopo da análise. Este trabalho apresenta um modelo para a comunicação segura entre os componentes de sistemas de detecção de intrusão em ambientes de larga escala utilizando os programas de detecção de intrusão convencionais, com o propósito de formar composições destes sistemas, dispersos geograficamente, e detectar uma quantidade maior de ataques, inclusive ataques distribuídos, através da correlação dos eventos. É introduzido um modelo de segurança fim a fim, utilizando padrões de segurança e a tecnologia de *Web Services*. São apresentados os principais esforços presentes na literatura sobre detecção de intrusão distribuída fazendo um comparativo entre as abordagens utilizadas e, então, o modelo de comunicação proposto é descrito. Também são descritos um protótipo desenvolvido, os resultados dos testes efetuados e algumas conclusões sobre o modelo proposto.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

SECURE COMMUNICATION FOR IDSs COMPOSITION AND COSTS

Paulo Manoel Mafra

August/2006

Advisor: Prof. Joni da Silva Fraga, Dr.
Area of Concentration: Automation and Systems
Key words: Computer Security, Distributed Systems
Number of Pages: xx + 105

The intrusion detection systems are usually designed to work in local area networks. These systems do not foresee the integration with other intrusion detection tools, neither use the Internet infrastructure to correlate events, by reducing the scope of the analysis. This work presents a model for secure communication between components of intrusion detection systems in large-scale environments, through the use of known intrusion detection tools to make intrusion detection system compositions, spread geographically, in which the main goal is to detect a higher number of attacks, including distributed attacks, through event's correlation. This work introduces a model for end-to-end security by using standards and the Web Services technology. The main efforts treated by distributed intrusion detection literature are presented, by making a comparison between the distinct approaches. The next chapters describe the proposed communication model and a developed prototype. The results of applied tests and some conclusions about the proposed model are also described.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

A Internet provê uma infra-estrutura de comunicação para ambientes distribuídos, possibilitando assim a integração de aplicações em ambientes fisicamente separados. A tecnologia usada em aplicações de comércio eletrônico (*e-business*), que fazem uso da Internet, está em constante evolução e já permite a integração de aplicações em ambientes distribuídos. Hoje em dia, estas tecnologias permitem o desenvolvimento de aplicações capazes de se inter-relacionar, através de mensagens, sem a interferência humana. Porém tais aplicações sofrem, a todo momento, ataques dos mais variados tipos. Estes ataques buscam explorar vulnerabilidades do sistema e obter vantagens como, por exemplo, o direito de acesso ao sistema para roubar informações sigilosas.

A arquitetura SOA (*Service Oriented Architecture*) provê um ambiente onde aplicações invocam serviços de outras aplicações, através da troca de mensagens (documentos), de maneira assíncrona, no formato padrão XML. Estas aplicações são pouco acopladas, deixando de lado a complexidade de sistemas fortemente acoplados como, por exemplo, o CORBA. Nesta arquitetura, os serviços são publicados e disponibilizados através do serviço de registro e pesquisa e podem ser acessados por quaisquer outros serviços. Esta arquitetura, apesar de recente, está sendo bastante utilizada por empresas para a disponibilização de seus serviços pela *Web*.

O aumento do número de ataques aos sistemas computacionais faz necessário o uso e desenvolvimento de sistemas de detecção de intrusão (*Intrusion Detection System - IDS*) capazes de identificar e prevenir violações a estes sistemas. No caso dos sistemas distribuídos, um ponto vulnerável é o sistema de comunicação que utiliza meios de comunicação de terceiros e inseguros. Para

prevenir violações em sistemas distribuídos, surgiram os sistemas de detecção de intrusão que atuam em ambientes distribuídos e buscam, além de identificar ataques locais, correlacionar eventos a fim de identificar ataques que aconteçam de maneira distribuída, partindo de diversos lugares. Estes ataques, geralmente, têm o objetivo de "atrapalhar" a comunicação e tornar o serviço indisponível.

Boa parte dos sistemas distribuídos atuais não utilizam modelos de comunicação segura ou utilizam modelos que permitem apenas a comunicação direta entre dois pontos, sem a possibilidade das mensagens serem transportadas passando por nós intermediários. Estes sistemas ainda apresentam dificuldades para transpor barreiras como *firewalls* e estabelecer a comunicação entre as partes comunicantes.

1.2 Objetivos

O presente trabalho consiste em desenvolver um modelo para comunicação segura e interoperável entre elementos de detecção de intrusão, seguindo o modelo de IDSs padrão recomendado pelo IDWG/IETF, para ambientes de larga escala. Uma característica importante do modelo é que o mesmo permite o uso de diversos sistemas de detecção de intrusão existentes, convertendo os alertas gerados por estes sistemas para um formato padrão de mensagens. Um requisito do modelo é que sejam utilizados padrões de segurança que garantam a autenticidade e integridade na troca de mensagens, até mesmo com a presença de nós intermediários.

Em seus objetivos iniciais, este trabalho se propôs a introduzir os conceitos de sistemas de detecção de intrusão, dando um enfoque especial para sistemas de detecção de intrusão distribuídos. O modelo que é objeto maior do trabalho, está fundamentado em *Web Services* e padrões IETF, sendo integrador e fornecendo suporte para ultrapassar barreiras entre domínios administrativos. O enfoque portanto está centrado sobre a comunicação.

Com a integração desses IDSs é possível reduzir o número de falsos positivos gerados pelos IDSs individualmente, detectar uma quantidade maior de ataques (inclusive ataques distribuídos) e evitar que todos os domínios administrativos presentes em uma composição sofram o mesmo tipo de ataque.

Este trabalho visa desenvolver um procedimento geral para implantação de composições de IDSs utilizando orquestração de *Web Services* e, principalmente, o desenvolvimento de um modelo de comunicação segura para integrar partes ou IDS completos já existentes. O resultado deste trabalho será usado em um projeto maior que tem o objetivo de criar composições dinâmicas de IDSs.

Este trabalho foi desenvolvido no Laboratório de Controle e Microinformática do DAS da Universidade Federal de Santa Catarina (LCMI-DAS). Este projeto tem como objetivo desenvolver um sistema seguro para comunicação entre elementos de detecção de intrusão distribuídos de larga escala.

1.3 Organização do Texto

Esta dissertação está estruturada em sete capítulos. Este capítulo inicial descreveu o contexto geral do trabalho, sua motivação e objetivos. Os demais capítulos encontram-se assim organizados:

O capítulo 2 apresenta os fundamentos sobre segurança computacional necessários ao compreendimento deste trabalho, as principais políticas e modelos de segurança presentes na literatura.

O capítulo 3 introduz a arquitetura orientada a serviços SOA, focando a tecnologia de *Web Services* utilizada para implementação dos serviços que compõem esta arquitetura.

O capítulo 4 discute alguns modelos de sistemas de detecção de intrusão, expondo as características de alguns sistemas de detecção distribuídos existentes e apresenta uma comparação destes sistemas discutidos.

O capítulo 5 introduz o modelo de composição de IDS usando *Web Services*, apresenta o procedimento geral desenvolvido para a criação e manutenção de composições de IDSs e descreve o modelo de comunicação segura entre elementos de IDS desenvolvido.

O capítulo 6 apresenta o protótipo do modelo desenvolvido, expondo as suas principais características, descrevendo os testes realizados e analisando os resultados obtidos com estes testes.

O capítulo 7 apresenta as conclusões sobre este trabalho, apresentando sugestões para trabalhos futuros.

Capítulo 2

Segurança Computacional

A preocupação com segurança em sistemas computacionais intensificou-se a partir da década de 70. Desde então foram desenvolvidos diversos modelos de segurança tanto para a comunicação quanto para os sistemas operacionais e suas aplicações. Neste capítulo abordamos os conceitos básicos de segurança computacional.

Com a evolução tecnológica, temos hoje milhares de computadores conectados à Internet. Um computador isolado, sem acesso a outros computadores, teria pouca utilidade no mundo atual. Os serviços de rede, tais como *e-mail*, *e-banking*, sítios de compras entre outros, que fazem uso da estrutura da Internet, são amplamente utilizados. Todos esses serviços precisam ser seguros para que possam garantir que não haja enganos ou erros nas transações e também evitar fraudes ou roubo de informações sigilosas.

O significado de segurança computacional tem sido modificado ao longo do tempo. Segundo o NIST[1] (Guttman, 1995), segurança computacional é a proteção de um sistema de informações automatizado a fim de alcançar os objetivos de preservação da integridade, disponibilidade e confidencialidade do sistema e seus recursos (hardware, software, firmware, dados e telecomunicações).

Segundo (Landwehr, 2001), um sistema computacional é seguro se ele estiver livre de preocupações e se ele for seguro contra ameaças. Landwehr define ainda a segurança computacional como a disciplina que nos ajuda a não nos preocuparmos com nossos sistemas computacionais.

Atualmente, dizer que um computador é seguro significa dizer que o sistema estará disponível para uso correto, e que ele irá prover as informações de maneira legítima, garantindo a confidencialidade, disponibilidade e integridade dessas informações.

[1] National Institute of Standards and Technology

2.1 Conceitos Básicos

A segurança de um sistema computacional está fundamentada na garantia de propriedades básicas (Landwehr, 2001):

- Confidencialidade: a informação não será revelada para indivíduos não autorizados.
- Disponibilidade: a informação estará sempre disponível, mesmo que ela venha a sofrer qualquer tipo de ataque.
- Integridade: a informação não será modificada ou destruída, ou seja, ela permanecerá completa, exata e consistente.

Além dessas três propriedades básicas, outras propriedades são usualmente indicadas:

- Autenticidade: a legitimidade de usuários, programas e informações deve ser garantida no sistema.
- Não-repúdio: a autoria de ações no sistema não pode ser negada.

Violação de Segurança

A violação de segurança é definida como a atividade que determina a não verificação de uma das propriedades citadas acima (Landwehr, 2001). Basicamente são três as violações identificadas na literatura:

- Revelação não autorizada de informação.
- Modificação não autorizada de informação.
- Negação de serviço.

As próximas seções definem conceitos amplamente usados em segurança computacional e que são aplicados ao longo deste texto.

2.2 Ameaças, Vulnerabilidades, Ataques e Riscos

Uma ameaça a um sistema computacional consiste em uma possível ação que, uma vez concretizada, produziria efeitos indesejáveis sobre os dados ou recursos do sistema, comprometendo as suas propriedades. Uma vulnerabilidade pode ser definida como um ponto fraco ou uma falha de concepção de um sistema computacional.

James P. Anderson em 1980 publicou um relatório "Computer Security Threat Monitoring and Surveillance" (Anderson, 1980) que define o termo vulnerabilidade como sendo um hardware ou software faltoso que expõe o sistema a uma possível invasão ou exposição acidental de informações. Portanto, essas falhas podem existir em uma parte (ou em um módulo) do software que executa em um modo privilegiado, uma senha fraca, uma regra de *firewall* mal configurada, um erro no projeto do hardware, etc.

As vulnerabilidades existem em sistemas operacionais, protocolos de rede e hardware. Diariamente surgem novos anúncios de vulnerabilidades. Existem diversos centros CSIRTs[2] que reportam e atendem aos usuários. A partir das vulnerabilidades descobertas, surgem os ataques.

Os ataques são ações que, usando vulnerabilidades do sistema, visam violações de segurança no mesmo. Todos os sistemas podem sofrer ataques, basta a máquina estar conectada à rede (Internet) para estar sujeita aos mesmos.

É de vital importância conhecer os tipos de ataques, as vulnerabilidades mais exploradas e os objetivos dos atacantes antes de tomar qualquer contra-medida. Os ataques mais freqüentes, segundo (Landwehr, 2001) são:

- Tentativas de negação de serviço: visam tirar servidores do ar por algum tempo. Essa técnica inunda o servidor alvo com milhares de requisições até que ele pare de responder.
- Varredura (*scan*) de portas: visa identificar os serviços de rede disponíveis, as versões dos programas, qual a versão do sistema operacional, se existe um *firewall* no caminho, entre outros.
- Uso de programas que exploram vulnerabilidades conhecidas em servidores desatualizados (geralmente estouro de *buffer*): efetuado geralmente por pessoas que não entendem de segurança e usam pequenos programas distribuídos nas salas virtuais de bate-papo.
- Cavalos de tróia: são programas que capturam dados digitados pelo usuário. São enviados por e-mail e dificilmente são detectados por anti-vírus.

[2]Computer Security Incident Response Team

- Sítios falsos: sítios que imitam os originais visando roubar dados bancários de usuários. O usuário recebe os caminhos para esses sítios por e-mail ou quando o servidor de nomes foi alterado.

- Ataques mais sofisticados que utilizam recursos distribuídos. Não existem técnicas para bloquear tais ataques.

Uma porcentagem grande do que foi relatado ao CERT[3] nos últimos anos envolveu vulnerabilidades de estouro de *buffer*. Assim como muitos outros tipos de ataques, o que explora o estouro de *buffer* tem evoluído em complexidade e sofisticação. No início, os ataques envolviam dados explicitamente lidos pelo programa alvo. Depois descobriu-se que variáveis usadas para guardar valores de variáveis de ambiente poderiam também ser usadas como vetores de ataques. Recentemente esses ataques são baseados no formato de *strings*.

Em 1992 surgiram os chamados *root kits* (Wahbe et al., 1993), programas desenvolvidos para explorar alguma vulnerabilidade e ganhar acesso ao sistema. Uma vez que o invasor obteve acesso privilegiado, ele pode modificar alguns programas (su, ssh por exemplo) para permitir acesso remoto ao sistema, facilitando um novo acesso ao sistema comprometido.

Tabela 2.1: *Vulnerabilidades reportadas ao CERT*

Ano	2000	2001	2002	2003	2004	2005
Vulnerabilidades	1.090	2.437	4.129	3.784	3.780	5.990

Tabela 2.2: *Incidentes reportados ao CERT.BR*

Ano	2000	2001	2002	2003	2004	2005
Incidentes	5.997	12.301	25.092	54.607	75.722	68.000

A Tabela 2.1 mostra a evolução do número de vulnerabilidades reportadas ao CERT. A Tabela 2.2 apresenta o número de incidentes de segurança reportados ao CERT.BR. As estatísticas vão de 2000 até 2005.

O número de vulnerabilidades encontradas por ano não tem variado muito, porém o número de incidentes tem sofrido um sensível aumento ao longo dos anos.

[3]Community Emergency Response Team

A maioria dos desenvolvedores de sistemas preocupa-se com desempenho e deixa de lado a segurança, o que aumenta o risco do sistema ter problemas de segurança. Algumas organizações utilizam a análise de risco (Stoneburner et al., 2001) na tentativa de determinar e quantificar potenciais ameaças. Essa análise pode ser usada tanto para desenvolver um plano de segurança da organização quanto pelos desenvolvedores de produtos. Com a análise de risco é possível identificar problemas para, posteriormente, adotar medidas visando sua redução, tanto na organização quanto em seus produtos.

2.3 Políticas e Modelos de Segurança

2.3.1 Políticas

Uma **política de segurança** pode ser definida como um conjunto de regras pré-definidas no sentido de manter propriedades de segurança no sistema (Landwehr, 2001). Para (Bishop, 2003), uma política de segurança forma a base para o estabelecimento do que é e o que não é permitido. As políticas de segurança podem seguir dois padrões de implementação distintos (Garfinkel et al., 2003):

1. Padrão de negação: especificar apenas o que é permitido e negar o resto.
2. Padrão de permissão: especificar somente o que é proibido e liberar o resto.

O padrão de negação é o mais prudente. Mesmo havendo um equívoco na sua definição, nenhum serviço ficará exposto. Já o padrão de permissão pode levar a uma violação.

As políticas de segurança de sistemas definem três níveis de segurança: segurança física, segurança gerencial e segurança lógica.

A segurança física é voltada à proteção dos recursos físicos do sistema. Nesta política são definidas medidas contra desastres físicos e medidas para proteger o acesso físico ao sistema, através da proibição do acesso de pessoas não autorizadas.

A política de segurança gerencial preocupa-se com o ponto de vista da organização, sendo responsável pela definição dos processos para criação e manutenção das próprias políticas de segurança.

A política de segurança lógica define quais usuários terão direito de acesso ao sistema e quais são estes direitos. Nesta política, são aplicados dois conceitos básicos: a autenticação, onde o usuário

precisa se identificar ao sistema para que possa obter acesso ao recurso; e a autorização, onde o usuário necessita provar que possui direitos sobre o recurso o qual deseja acessar.

Na literatura encontramos os termos **principal** e **intruso**. O termo **principal** é utilizado para identificar usuários, processos ou máquinas atuando em nome dos usuários de um sistema, que são considerados aptos (pela política lógica) para executar determinada ação no sistema. O termo **intruso**, em sentido contrário, é usado para identificar usuários, processos e máquinas não autorizados pela política estabelecida.

2.3.2 Modelos de Segurança

Os modelos de segurança são diferenciados de mecanismos de segurança. Modelos de segurança determinam como os acessos são controlados e mecanismos de segurança são as funções de *software* e *hardware* que são configuradas a partir de um modelo de segurança. A seção 2.5 trata destes mecanismos. Um modelo de segurança é uma expressão, muitas vezes, formal de uma classe de políticas, abstraindo detalhes e concentrando-se em um conjunto de comportamentos. Este modelo é utilizado como base para definição de políticas de segurança (Landwehr, 1981). Na literatura, os modelos são apresentados divididos em três tipos básicos: discricionários (*discretionary*), obrigatórios (*mandatory*) e os baseados em papéis (*roles*).

Modelos Discricionários

Nos modelos discricionários os controles de acesso são definidos segundo políticas construídas pelos proprietários dos objetos. Cada requisição de um usuário para acessar um objeto requer a verificação das autorizações previamente especificadas. Caso exista uma autorização permitindo que o usuário possa acessar o objeto, no modo especificado (ler, escrever, etc), o acesso é permitido do contrário é negado (Sandhu e Samarati, 1996).

Em (Lampson, 1974) foi introduzido o modelo de matriz de acesso, um modelo conceitual discricionário o qual especifica os direitos que cada sujeito possui sobre cada objeto do sistema. Os sujeitos (usuários, processos ou máquinas) representam o usuário (**principal**). Fazendo uma analogia com o controle de acesso de um sistema operacional, cada linha da matriz representa um sujeito do sistema, cada coluna da matriz representa um objeto deste sistema (arquivo, processo, etc) e as células indicadas pelas linhas e pelas colunas especificam os direitos de autorização do sujeito (linha) sobre o objeto (coluna).

Os modelos discricionários não impõem qualquer restrição na forma de uso da informação por um usuário, facilitando a transposição das restrições de acesso impostas pelas autorizações. Um usuário que possui permissão para ler dados pode passar esta permissão para outros principais não autorizados.

Modelos Obrigatórios

Modelos obrigatórios caracterizam políticas globais incontornáveis em um sistema. Estes modelos determinam controles que definem os fluxos permitidos de informação no sistema. Estes modelos são baseados na classificação de sujeitos e objetos no sistema. Cada sujeito e cada objeto no sistema é associado a um nível de segurança. O nível de segurança associado a um objeto reflete a sensibilidade da informação contida neste objeto, ou seja, quanto mais sensível for a informação maior será o nível de segurança deste objeto. O nível de segurança associado a um sujeito (*clearance*) reflete a confiança no sujeito em não revelar informações sensíveis para outros sujeitos (Sandhu e Samarati, 1996). Em um exemplo simples, o nível de segurança é um elemento de um conjunto (ultra-secreto (US), secreto (S), confidencial (C) e não classificado (NC)) ordenado hierarquicamente, onde US $>$ S $>$ C $>$ NC. Cada nível de segurança possui acesso ao seu próprio nível e todos os outros abaixo dele na hierarquia.

Além dos níveis de segurança hierárquicos, categorias também podem ser associadas com objetos e sujeitos. Neste caso, os rótulos de classificação de segurança são associados a sujeitos e objetos, formando um par composto por um nível de segurança e um conjunto de categorias. O conjunto de categorias associado com um usuário reflete as áreas específicas às quais o usuário pertence. O conjunto de categorias associadas a um objeto reflete a área na qual as informações contidas neste objeto são referenciadas. O uso de categorias provê um refinamento na classificação de segurança.

Na literatura existem diversas descrições de modelos obrigatórios, entre os quais podemos citar o modelo Bell-LaPadula (Bell e LaPadula, 1976), baseado nos procedimentos usuais de manipulação de informação em áreas ligadas à segurança nacional americana. Neste modelo, o sistema é descrito através de uma máquina de estados finitos onde as transições de estado obedecem a determinadas regras. Bell e LaPadula demonstraram por indução que a segurança do sistema é mantida partindo de um estado seguro e as únicas transições de estado permitidas são as que conduzem o modelo a um outro estado seguro.

Modelos Baseados em Papéis

Os modelos discricionários e obrigatórios não preenchem todos os requisitos necessários para a maioria das organizações. Desta forma, surgiram algumas alternativas para estes modelos, dentre os quais estão os modelos baseados em papéis (Ferraiolo e Kuhn, 1992; Sandhu et al., 1996).

Um papel pode ser definido como um conjunto de ações e responsabilidades associadas com uma atividade do sistema (Sandhu e Samarati, 1996). Ao invés de especificar todos os acessos que cada principal pode executar, as autorizações de acesso em objetos são especificadas através de papéis. O principal pode executar todos os acessos concedidos àquele papel. Em geral, um principal pode fazer uso de diferentes papéis em diferentes ocasiões e o mesmo papel pode ser usado por vários principais, algumas vezes simultaneamente. Um estudo (Ferraiolo et al., 1993) afirma que o uso de modelos baseados em papéis em organizações é uma abordagem que possui muitas vantagens. Algumas delas são discutidas a seguir:

- Gerência de autorização: a tarefa de especificar autorizações é dividida em duas partes, a associação de principais aos papéis e a associação de direitos de acesso aos papéis, simplificando o gerenciamento de segurança, por exemplo, no caso de um principal ter que desempenhar um novo papel na organização.
- Hierarquia de papéis: em muitas aplicações existe uma hierarquia de papéis natural, baseada nos princípios de generalização e especialização, permitindo, desta forma, que permissões sejam herdadas ou compartilhadas, simplificando a gerência de autorização.
- Privilégio mínimo: papéis permitem que um principal trabalhe com o privilégio mínimo necessário para uma determinada tarefa, minimizando o risco de danificar o sistema em uma ocorrência de erros ou em uma intrusão.
- Separação de tarefas: a separação de tarefas parte do princípio de que nenhum principal deveria obter privilégios a fim de que o mesmo possa utilizá-los de forma maliciosa, em benefício próprio. Por exemplo, o responsável pela autorização de pagamentos não deveria ser o responsável por efetuá-los.
- Classificação de objetos: o modelo baseado em papéis provê uma classificação de principais de acordo com a atividade que cada um executa no sistema. Analogamente, uma classificação deveria ser dada aos objetos do sistema. Desta forma, as autorizações seriam dadas a classes de objetos e não aos objetos específicos, facilitando a administração das autorizações e o seu controle.

2.4 Princípios de Segurança

Em (Zwicky e Cooper, 2000) são descritas propriedades e requisitos que aumentam a segurança:

- *Accountability*: ações dentro do sistema devem ser controladas por políticas de seguranca, registradas e autorizadas. Ações críticas de segurança precisam ser auditadas.
- *Least Privilege*: para cada processo em um sistema, devem ser garantidos somente aqueles privilégios necessários para que ele desempenhe a sua função. Aplicações que usam este princípio requerem que cada processo possa ler e escrever somente naqueles arquivos ou em partes particulares de arquivos, essencialmente para esse propósito. Por outro lado, hoje este princípio parece largamente esquecido ou ignorado. Cada aplicação que executa em um computador pessoal tem acesso completo ao sistema de arquivos e pode causar muitos danos, acidentalmente ou maliciosamente.
- *Defence in depth*: segue o princípio de usar vários mecanismos diferentes e mecanismos duplicados, assim alguns mecanismos podem falhar mas outros não. Com isso, ataques que exploram a falha de um mecanismo irão ser bloqueados por outro mecanismo diferente.
- *Fail safe*: em caso de falha, ativa o modo de segurança. Deixa o sistema indisponível porém mantém a integridade de seus dados.

Geralmente o esforço para manter um sistema seguro é proporcional ao seu tamanho e complexidade. Minimizar a variedade, espaço e complexidade dos componentes confiáveis no modelo do sistema pode reduzir custos e riscos. Com isso é mais fácil assegurar que tais sistemas sejam corretamente especificados e implementados. Se um modelo centraliza toda a verificação de segurança em um componente que executa em um domínio separado e protegido, ele será freqüentemente invocado e isso requer um grande número de operações. Os *kernels* centralizados sofrem desse problema (Landwehr, 1983).

2.5 Mecanismos de Segurança e Autenticação

Definidas as políticas de segurança, apresentamos alguns mecanismos utilizados para evitar ou prevenir violações de segurança. Um mecanismo de segurança pode ser um método, uma ferramenta ou um procedimento para viabilizar e implementar uma política de segurança (Bishop, 2003).

Em (Stallings, 2000) são definidos mecanismos de segurança como aqueles mecanismos que são desenvolvidos para detectar, previnir ou recuperar o sistema de um ataque de segurança. Algumas ferramentas e mecanismos utilizados na prevenção e detecção de ataques são definidos a seguir.

Domínios de Proteção

Um domínio é definido como uma coleção de dados e um conjunto de autorizações para manipulá-los, utilizando um sistema de computador (Lampson, 1974; Landwehr, 1984). Além dos mecanismos para criação de domínios, são necessários mecanismos para controlar a comunicação entre domínios. Domínios podem ser construídos usando características de *hardware*, usando somente *software* ou, como é o caso usual, alguma combinação de *software* e *hardware*.

Autenticação

O processo de autenticação em sistemas computacionais consiste de um conjunto de procedimentos e mecanismos que permitem que agentes externos (usuários, dispositivos, etc) sejam identificados como principais, autorizados segundo as políticas de segurança adotadas no sistema.

O nome de usuário e senha continua sendo a forma mais comum de identificação e autenticação (Landwehr, 2001). Entretanto, o uso de cartões (tipo *smart cards*) e biometria está crescendo. Estas formas de autenticação dependem de fatores como: alguma coisa que o usuário conhece (senha), alguma coisa que o usuário tem (uma chave) ou alguma coisa que o usuário é (impressão digital, íris).

Um elemento chave no processo de autenticação é a existência de um caminho de confiança entre a fonte da autenticação e o componente de decisão. Se um componente não confiável pode introduzir ou coletar informações que são cruciais ao processo de autenticação, esse processo pode ser clonado.

Controle de Acesso

O controle de acesso faz a mediação de requisições entre sujeitos e objetos em um sistema. Existe uma entidade, chamada de monitor de referência, que recebe as requisições de acesso dos sujeitos e autoriza ou nega o acesso de acordo com a política de segurança utilizada.

Mecanismos de autorização e de controle de acesso existem em uma variedade de níveis em um computador. O processo de *login* pode determinar o que um certo sujeito está autorizado a utilizar.

O sistema pode decidir se um sujeito está autorizado a acessar alguns arquivos em particular e uma aplicação como um sistema de gerência de base de dados pode decidir quais campos o sujeito pode acessar.

Detecção de Intrusão e Tolerância a Intrusão

Os sistemas de detecção de intrusão são sistemas que monitoram redes ou máquinas, com o propósito de encontrar violações de segurança. Tais sistemas podem ser centralizados, híbridos ou distribuídos. Para mais detalhes veja o capítulo 4.

Sistemas de tolerância a intrusão são sistemas de alto risco que, em geral, mesmo estando sob ataques devem continuar funcionando, ainda que com seu desempenho reduzido. Em (Fraga e Powell, 1985) são definidos sistemas tolerantes a intrusão como sistemas que toleram algumas intrusões ou ataques, mas continuam funcionando de maneira correta.

Auditoria

Auditoria é a análise de registros visando apresentar informações sobre o sistema de forma clara e compreensível (Bishop, 2003). Fazer auditoria em registros é extremamente útil para administradores de sistema, pois é possível identificar violações do sistema ou falhas em programas.

O mecanismo de auditoria deve ser capaz de resistir a possíveis ataques. É possível implementar mecanismos, que geram saídas em dispositívos que escrevem somente uma vez ou enviam os dados para uma máquina remota, dificultando a destruição dos registros do sistema por um intruso.

A auditoria feita no nível do sistema operacional pode ser consistente, mas será constituída de registros grandes de eventos pequenos. Tais registros podem ser usados para reconstruir a atividade de um intruso, depois que a invasão foi detectada.

Cada aplicação irá gerar seus próprios tipos de auditoria. Hoje, a auditoria de segurança em registros é provavelmente a maneira mais comum de verificar se houve violações, além de ser uma fonte de dados para os sistemas de detecção de intrusão (Landwehr, 2001).

Controles Criptográficos

Controles criptográficos são sistemas que transformam um texto inteligível numa forma ilegível e vice-versa (Clark e Limited, 1996). Estes sistemas são utilizados para proteger documentos de

pessoas não autorizadas e para assinatura digital de documentos.

Os algoritmos criptográficos podem utilizar uma chave ou um par de chaves. Se as chaves e os algoritmos para cifrar e decifrar são os mesmos, nós temos um sistema criptográfico simétrico (Berthold et al., 2000).

Se os algoritmos envolvem duas chaves diferentes, uma para cifrar e outra para decifrar, nós temos um sistema criptográfico assimétrico ou algoritmo de chave pública (Berthold et al., 2000). Esses algoritmos assimétricos têm a vantagem de que uma das chaves pode ser pública e distribuída livremente. As informações cifradas com a chave pública, podem ser decifradas somente com a chave privada. As informações cifradas com a chave privada podem ser decifradas por qualquer um, usando a chave pública.

Em geral, algoritmos de chave pública são computacionalmente menos eficientes do que algoritmos de chave secreta (simétricos), então eles são aplicados somente em "pequenos trechos" de informação. Algumas vezes usa-se um sistema assimétrico somente para cifrar uma chave e um algoritmo simétrico para fazer a cifragem do texto. Alguns sistemas operacionais oferecem a opção de utilizar sistemas de arquivos cifrados.

A criptografia é também usada para a criação de canais de comunicação seguros (*secure sockets layer - SSL* e *transport layer security - TLS*) (Garfinkel et al., 2003). Tais canais são usados por aplicações de comércio eletrônico, transações bancárias entre outras. O sistema de chaves públicas é usado para gerar e comunicar uma chave de sessão compartilhada, que é usada com uma chave secreta (sistema simétrico) para evitar o roubo de informações como, números de cartões de crédito ou senhas bancárias, por exemplo. A criptografia pode transformar o problema de comunicação segura em um problema de gerenciamento de chaves (Gollman, 1999).

2.6 Conclusões do Capítulo

A "explosão" do número de computadores interligados através da Internet nos últimos anos, aliada à falta de sistemas computacionais projetados com o quesito segurança abriu uma brecha para o grande número de *bugs* de segurança encontrados nestes sistemas.

Muitas falhas de segurança são ocasionadas por descuidos de programação dos *softwares*. Um dos problemas mais comuns é o estouro de *buffer*. Este problema é descrito com detalhes em (McHugh, 2001). A maioria das violações de segurança pode ser evitada ou minimizada através do uso de políticas de segurança adequadas, contudo, não existe um sistema computacional inviolável.

Este capítulo tratou dos conceitos básicos relacionados a segurança computacional, introduzindo tais conceitos e apresentando algumas das principais políticas e modelos de segurança presentes na literatura. O capítulo também definiu conceitos importantes como ameaças, vulnerabilidades, ataques e riscos que serão empregados ao longo deste texto, encerrando com a apresentação de mecanismos de segurança e autenticação, amplamente adotados em sistemas computacionais.

Capítulo 3

Arquitetura Orientada a Serviços

A necessidade de interoperabilidade entre serviços *Web* provocou uma evolução das tecnologias envolvidas. Esta evolução chegou a uma arquitetura orientada a serviços SOA (Service Oriented Architecture), também conhecida como terceira geração de aplicações *Web* (O'Neill, 2003). O conceito de SOA é recente em computação distribuída, no qual aplicações chamam serviços de outras aplicações em uma rede. Nesta arquitetura, os serviços são publicados (operação *publish*) e disponibilizados através de outras duas operações importantes: *discovery* (descoberta do serviço) e *binding* (habilidade para se conectar ao serviço).

Este capítulo trata essencialmente da tecnologia de *Web Services*, definindo primeiramente o que são *Web Services*. A seguir, descrevemos a sua arquitetura, explicando como funciona o empacotamento de mensagens, a linguagem de definição de *Web Services*, a UDDI e o sistema de notificações. Em seguida, apresentamos dois modelos para composição de *Web Services* chamados Coreografia e Orquestração. Finalizamos o capítulo citando alguns mecanismos para manter segurança em *Web Services* e em suas transações.

3.1 Definição

Os *Web Services* são compostos por objetos distribuídos para a integração de aplicações através da Internet. Estas aplicações, em geral desenvolvidas para o comércio eletrônico, provêem a troca de documentos de maneira independente (somente entre aplicações) ou com alguma interação humana, para busca de informações, produtos ou serviços (Booth et al., 2004), na rede mundial.

Os *Web Services* são baseados no grande potencial da combinação de XML (Extensible Markup

Language) (Bray et al., 2000), da infra-estrutura da *Web*, do protocolo SOAP (Gudgin et al., 2003) e das especificações WSDL (Web Service Description Language) (Chinnici et al., 2006). Um *Web Service* interage com outros *Web Services* trocando informações na forma de mensagens, usando uma variedade de padrões que os tornam independente de plataformas, linguagens e hardware. O padrão de mensagens requisição/resposta é a base de qualquer modelo de *middleware*.

Segundo a definição da W3C, um *Web Service* é um sistema de software identificado por uma URI, no qual as interfaces públicas são definidas e descritas usando o XML. Essa definição pode ser encontrada por outros sistemas de software que podem então interagir com o *Web Service* seguindo a maneira prescrita na mesma, usando mensagens baseadas em XML através dos protocolos de comunicação que formam a Internet.

O *Web Service* é composto basicamente por mensagens XML e agentes de software que processam essas mensagens (Booth et al., 2004). Esses agentes podem ser uma combinação qualquer de:

- Emissor - gera e envia dados no padrão XML na forma de uma mensagem.
- Receptor - aceita e processa dados XML recebidos como uma mensagem.
- Intermediário - uma combinação de emissor e receptor que aceita mensagens e processa certas funções restritas associadas à mensagem que está roteando ou outra operação não envolvendo o conteúdo XML e a sua função principal é o repasse da mensagem.

A seguir, apresentamos a arquitetura de *Web Services* e os serviços básicos que a compõem.

3.2 Arquitetura

Na arquitetura de *Web Services* são identificados os componentes principais do sistema e definidas as relações entre estes componentes, a fim de cobrir os requisitos do sistema. A arquitetura de *Web Services* descreve um sistema composto por tecnologias que trocam mensagens entre emissores e receptores, podendo incluir intermediários. Esta arquitetura utiliza padrões para a troca de mensagens. Estes padrões são estendidos com funcionalidades para assegurar privacidade, coordenação das transações, uso de múltiplas mensagens, etc. Os componentes da arquitetura de *Web Services* usam definições padrão em XML para o tipo dos dados e a estrutura das mensagens.

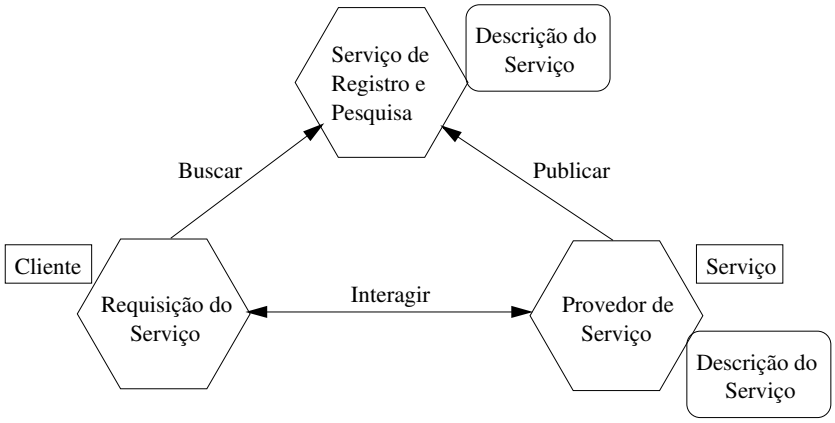

Figura 3.1: *Arquitetura básica de Web Services*

A arquitetura básica de *Web Services* é ilustrada na Figura 3.1, onde clientes e provedores interagem usando um ou mais padrões para troca de mensagens. As interações envolvem três atividades básicas:

- Publicação: no sentido de ser acessível, o provedor de um serviço precisa publicar suas descrições para que o cliente possa encontrá-las. O local da publicação pode variar (UDDI, FTP, etc).
- Busca: na operação de busca, o cliente recupera a descrição de um serviço diretamente ou requisita o registro para o tipo de serviço desejado. A busca pode ser utilizada em duas etapas do ciclo de vida do cliente: no desenvolvimento do programa cliente ou em tempo de execução, para encontrar o serviço desejado.
- Interação: na operação de interação o cliente invoca ou inicializa uma interação com o serviço, em tempo de execução, utilizando as informação sobre localização, contato e invocação do serviço. As interações podem ser: simples (*one way*), *broadcast* de um cliente para muitos serviços, de um serviço para múltiplos clientes ou a maneira convencional onde um cliente e um servidor trocam múltiplas mensagens. Qualquer destes tipos de interação podem ser apresentados na forma síncrona ou assíncrona.

Como já foi dito, a base do *Web Service* é a troca de mensagens. Clientes requerem a execução de um serviço nos seus provedores. Os provedores são responsáveis por publicar a descrição do serviço para que seus clientes sejam capazes de encontrar estas descrições e possam acessar estes serviços. Um serviço, ao ser invocado por um cliente, pode também funcionar como cliente, usando outros serviços em sua implementação.

A descrição do serviço contém os detalhes de sua interface e de sua implementação, incluíndo seus tipos de dados, operações e localização na rede. A descrição completa pode ser feita como um conjunto de documentos XML. Esta descrição do serviço pode ser publicada diretamente em um cliente ou através de um serviço de registro e pesquisa (SRP).

Após definidos os papéis e operações básicas da arquitetura de *Web Services*, citamos algumas características importantes desta tecnologia:

- Uso de mensagens síncronas ou assíncronas.
- O empacotamento de múltiplos documentos em uma mesma mensagem.
- Padrão de troca de mensagem (Message Exchange Patterns - MEP) - é uma forma especializada que descreve um padrão geral para troca de mensagens entre dois serviços.
- Autenticação de mensagens - é possível fazer a autenticação de mensagens através do HTTP *auth*, um módulo SOAP com usuário/senha, x.509, etc.
- Confidencialidade de mensagens - é possível utilizar um protocolo de transporte protegido por SSL ou TLS ou um módulo SOAP que provê a cifragem dos dados sensíveis da mensagem.
- Integridade de mensagens - uma solução é o uso de um módulo SOAP que usa uma assinatura digital sobre uma parte ou toda a mensagem.

Aprofundando um pouco a arquitetura básica definida, apresentamos *Web Services* como entidades que podem assumir múltiplos papéis simultaneamente. No cenário par a par, cada instância de *Web Service* tem o papel como cliente e como servidor. A Figura 3.2 ilustra este cenário. O cliente pode requisitar um serviço e prover outro. A publicação e busca da descrição dos serviços é feita no serviço de registro e pesquisa.

Nesta Figura 3.2, aparece o papel do intermediário, uma combinação de emissor e receptor que aceita mensagens e processa certas funções restritas associadas com a mensagem que está roteando ou outra operação não envolvendo o conteúdo XML e ao final repassa a mensagem. Em transações entre *Web Services* não é necessário a presença de intermediários.

A seguir descrevemos a estrutura do SOAP, o protocolo usado para empacotar mensagens trocadas entre *Web Services*.

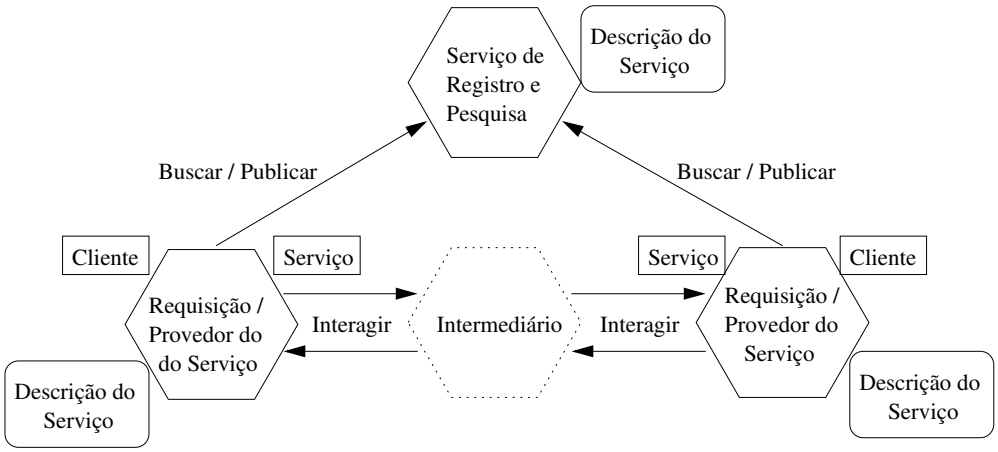

Figura 3.2: *Cenário para uma rede par a par*

3.2.1 SOAP

O SOAP (Gudgin et al., 2003) é um protocolo baseado em XML (Bray et al., 2000) para realizar trocas de mensagens XML entre aplicações. Atualmente, o SOAP é o principal protocolo para troca de mensagens, usado no ambiente dos *Web Services*. O foco inicial do SOAP é possibilitar o uso de RPC sobre o HTTP, porém o seu uso nos *Web Services* não se restringe simplesmente a isto, podendo ser utilizado em diversas formas para troca de mensagens, incluindo a troca de mensagens uni-direcional e bi-direcional.

O protocolo SOAP consiste de quatro componentes fundamentais: um envelope que define um modelo para descrição da estrutura da mensagem, um conjunto de regras para expressar instâncias de tipos de dados, um modelo para representar requisições e respostas, e um conjunto de regras para usá-lo com protocolos de transporte. O SOAP pode ser usado em combinação com uma variedade de protocolos de transporte, tais como HTTP, SMTP, FTP, RMI/IIOP ou outros protocolos proprietários, garantindo a interoperabilidade entre sistemas diversos.

Apesar desta independência de protocolo para transferência de suas mensagens, o HTTP é o meio de comunicação dominante para tal. Muitos acreditam que o HTTP é o protocolo nativo de *Web Services* porque ele foi desenvolvido para trabalhar com URIs que identificam recursos *Web*.

Mensagens SOAP

O SOAP, que está se tornando padrão para troca de mensagens entre aplicações e *Web Services*, provê um formato para os dados XML. O formato básico de mensagem XML é descrito na Figura

3.3, consistindo em:

- Envelope - é o elemento raiz no documento XML que representa a mensagem SOAP.
- Cabeçalho - especifica outros dados da mensagem e oferece mecanismos para estendê-la (é opcional). Aqui entram as especificações das extensões mencionadas.
- Corpo - contém o conteúdo da mensagem, a RPC e os demais dados (Ex. mensagem cifrada).

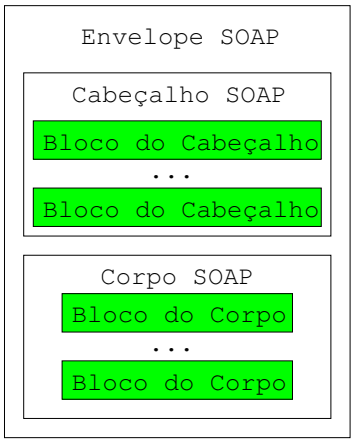

Figura 3.3: *Mensagem SOAP*

Outra característica do SOAP é o uso do MEP (Message Exchange Pattern), que estabelece um padrão para a troca de mensagens entre nodos SOAP (agentes de *software*). Exemplos de MEPs incluem requisição/resposta, *one-way*, par-a-par ou muitos para muitos.

O SOAP assume que uma mensagem originada em um emissor inicial é enviada para um receptor SOAP, podendo passar por um ou mais nós intermediários SOAP. Um nó SOAP pode ser o emissor inicial, um último receptor ou um intermediário SOAP. Um nó que recebe uma mensagem SOAP deve processá-la de acordo com o seu papel na comunicação.

Ambos emissor e receptor de uma mensagem SOAP devem ter acesso à mesma descrição do serviço. O emissor precisa da descrição do serviço para saber como formatar a mensagem corretamente e o receptor precisa da descrição do serviço para entender como receber a mensagem corretamente. Esta descrição do serviço é feita através da linguagem de descrição de *Web Services* (WSDL), descrita a seguir.

3.2.2 WSDL

A implementação por trás de um *Web Service* é transparente aos seus usuários, podendo variar a liguagem, a forma como os métodos foram implementados, etc. Ou seja, somente a representação dos serviços, definida através da linguagem de descrição de *Web Services* chamada WSDL (Web Service Definition Language) (Champion et al., 2004) é que é visível.

A linguagem de descrição de *Web Services* (WSDL) é usada para descrever os tipos de dados da mensagem, a definição da estrutura da mensagem, a definição do padrão de troca de mensagens e o endereço destino do receptor. Quando um emissor deseja enviar uma mensagem para um receptor, o emissor obtém a descrição do serviço e gera uma mensagem correspondente ao tipo de dado e estrutura da informação contida na descrição do serviço e envia a mensagem ao endereco destino identificado na descrição do serviço. O receptor espera por novas mensagens no endereço definido. Quando o receptor recebe uma nova mensagem, ele a valida usando o tipo de dados e informação de estrutura contidos na sua descrição do serviço.

Na WSDL, um *Web Service* é modelado como um elemento do serviço. Um elemento do serviço contém uma coleção de portas. Uma porta associa uma URI com um elemento da interface do serviço. WSDLs descrevem *Web Services* de uma forma estruturada. Uma descrição WSDL de um serviço nos mostra a interface para o serviço, os tipos de dados utilizados e onde o serviço está localizado, ou seja, uma WSDL define:

1. A funcionalidade abstrata de um serviço.
2. Detalhes concretos de associação com os protocolos SOAP, HTTP, MIME entre outros.

Definições de *Web Services* podem ser implementadas em qualquer linguagem, modelo de objeto ou sistema de mensagens. Extensões simples existentes na Internet podem implementar *Web Services* para interagir com navegadores ou diretamente com uma aplicação. A aplicação pode ser implementada usando COM, JMS, CORBA, COBOL ou qualquer outra solução proprietária.

A WSDL descreve *Web Services* começando com mensagens que são trocadas entre o provedor do serviço e o cliente. As mensagens são descritas de maneira abstrata, mas especificam um protocolo concreto de rede e o formato da mensagem. As trocas de mensagens entre o provedor do serviço e o cliente são descritas como operações.

Um serviço *Web* representa uma implementação de um serviço, descrito via documento WSDL, que contém um conjunto de portas, onde uma porta é um canal para interagir com este serviço.

Estes documentos WSDL, que contém as descrições de serviços, ficam armazenados em um tipo de repositório chamado UDDI, que suporta o registro, a pesquisa e a integração de serviços *Web*.

3.2.3 UDDI

Para publicar serviços, encontrar outros serviços e permitir a integração destes, os *Web Services* utilizam a UDDI (*Universal Description, Discovery and Integration specification*) (Clement et al., 2004), que é uma especificação que suporta a publicação e a consulta de documentos WSDL sobre serviços e por consequência para obtenção de suas URIs.

A UDDI está baseada no uso de padrões como XML, XML Schema e SOAP, e fornece uma infra-estrutura para procura de *Web Services* em ambientes de larga escala (abertos) ou restritos a um domínio administrativo. Através da replicação da UDDI é possível manter a escalabilidade necessária para atuar em ambientes de larga escala.

Na UDDI, existem estruturas do tipo *tModel* (*technical Model*) que definem tipos de *Web Services*, protocolos utilizados pelos mesmos ou categorias de sistemas. As interfaces do serviço, descritas em documento WSDL, são obtidas a partir das URLs disponíveis nos *tModels*. Cada instância de serviço é registrado na UDDI utilizando uma estrutura *businessService* cujas subestruturas *bindingTemplate* fornecem, por exemplo, as informações técnicas sobre o serviço e sua relação com os *tModels*.

Na sua especificação mais recente, a UDDI pode ser utilizada também para armazenar outros tipos de documentos, como por exemplo, um repositório de certificados digitais. No modelo de comunicação segura proposto, neste trabalho utilizaremos tal repositório.

A seguir, descrevemos um sistema de notificação de eventos, usado com freqüência em *Web Services* e que também faz parte do modelo de comunicação proposto nesta dissertação.

3.2.4 Sistema de Notificações

Em diversas situações de negócios envolvendo a tecnologia de *Web Services* como, por exemplo, um sistema de mensagens de empresas, é necessário o uso de um sistema de notificações. Este tipo de serviço caracteriza um modelo de comunicação desacoplada.

O *Pubscribe* é um sistema de notificações da família WSN (*Web Service Notification*), especificado pela OASIS (Graham et al., 2004). Estas especificações descrevem como implementar o padrão

publish/subscribe usando *Web Services* de forma escalável (Apache, 2005b). As especificações incluem:

- *WS-BaseNotification* - Define os participantes (produtores e consumidores) e as mensagens necessárias para suportar notificações básicas. Um consumidor pode inscrever-se em um ou mais tópicos de notificações que são suportados por um produtor. Quando uma notificação é publicada por um produtor, a mensagem SOAP correspondente é enviada a todos os consumidores que estão inscritos para aquele tópico de notificação.

- *WS-Topics* - Descreve uma hierarquia de tópicos de notificações e as expressões que serão usadas para referenciar um ou mais tópicos nesta hierarquia. Os tópicos funcionam como assuntos gerais em que as notificações devem ser publicadas. Uma notificação deve referenciar pelo menos um tópico.

- *WS-BrokeredNotification* - Define como o produtor de uma notificação deve proceder (quais operações devem ser executadas) para publicá-la, através de uma entidade chamada *broker*. Um *broker* agrega mensagens de muitos produtores e também possui a habilidade para prover diferentes canais de notificação através do uso de tópicos.

Na seção seguinte, apresentamos o conceito de composição de *Web Services* e os dois principais modelos de composição presentes na literatura: **coreografia** e **orquestração**.

3.3 Composição de Web Services

Tão logo a tecnologia de *Web Services* começou a ser usada, sentiu-se a necessidade de integrar aplicações utilizando padrões abertos baseados em XML (Peltz, 2003), formando o que é identificado como composições de serviços. Entre os padrões usados para estas composições estão o BPEL4WS (*Business Process Execution Language for Web Services*) e o WSCI (*Web Service Choreography Interface*). Estes padrões são usados respectivamente para representar a orquestração e a coreografia de *Web Services*. Estes termos (orquestração e coreografia) foram introduzidos recentemente na literatura e tanto a orquestração quanto a coreografia formam a composição que integra e permite a definição de interações de *Web Services*.

3.3.1 Coreografia

A coreografia descreve uma composição, determinando um conjunto de mensagens envolvendo múltiplas partes e múltiplas origens desta composição. A coreografia está associada com a troca pública de mensagens que ocorre entre quaisquer *Web Services*, ao invés de um processo de negócio específico que é executado por um conjunto específico de serviços (orquestração). A coreografia é mais aberta (colaborativa), nela cada parte envolvida na interação descreve a sua parte na negociação.

O WSCI é uma especificação da Sun, SAP, BEA e Intalio que define uma linguagem baseada em XML para a colaboração de *Web Services* (Arkin, 2002). Esta especificação define toda a coreografia descrevendo as mensagens entre *Web Services* que participam das trocas colaborativas. A especificação suporta correlação de mensagens, regras para seqüência, tratamento de exceções, transações e colaborações dinâmicas.

Um aspecto importante do WSCI é que este descreve apenas o comportamento visível entre *Web Services*. O WSCI não apresenta a definição de um processo de negócio executável como no caso de orquestração. Um simples documento WSCI descreve a participação de cada parceiro em trocas de mensagens.

A Figura 3.4 ilustra o modelo de coreografia WSCI, que pode incluir um conjunto de documentos, um para cada parceiro no conjunto de interações. No WSCI não existe um processo controlando a interação e sim uma colaboração entre *Web Services*.

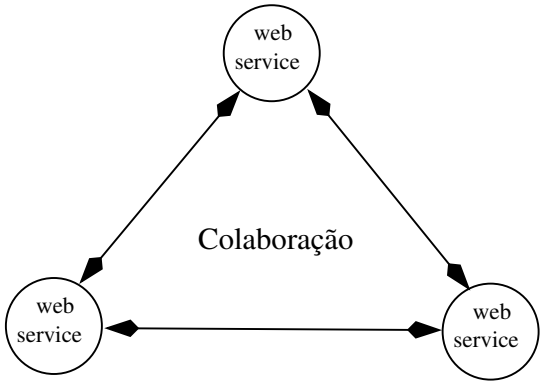

Figura 3.4: *Modelo de coreografia*

Cada ação no WSCI representa uma unidade de trabalho na qual é mapeada para uma operação WSDL específica. A WSDL descreve as entradas para cada serviço disponível e o WSCI descreve as interações proporcionadas por estas operações WSDL.

3.3.2 Orquestração

Web Services permitem a composição de serviços, mesmo em organizações diferentes. Para especificar como um conjunto de *Web Services* deve ser utilizado para realizar funções mais complexas, como os descritos em processos de negócio, foi introduzido o conceito de orquestração. Uma orquestração descreve como *Web Services* podem interagir entre si em nível de mensagens, incluindo a lógica do negócio e a ordem de execução das interações nesta composição. Estas interações podem interligar aplicações ou organizações resultando em um modelo de processo longo e de vários passos.

Na orquestração, um processo de negócio (*business process*) especifica a ordem de execução das operações em um conjunto de *Web Services*, os dados compartilhados entre esses serviços, quais parceiros estão envolvidos e como eles são envolvidos no processo, aumentando a consistência e a confiança nas aplicações (Leymann e Roller, 2002).

O BPEL4WS, ou simplesmente BPEL, é uma linguagem para especificar processos e protocolos para interações de negócios (Leymann e Roller, 2002). Esta linguagem evoluiu do XLANG (Tartanoglu et al., 2002) da Microsoft e do WSFL (Leymann, 2001) da IBM como um padrão para especificação de fluxos de processos para *Web Services*.

Para especificar a troca de mensagens entre parceiros envolvidos em um negócio, são usados protocolos de interação, que são chamados de processos abstratos. Estes processos não revelam o comportamento interno ou a implementação das partes envolvidas. As descrições feitas com BPEL são baseadas no modelo de descrição do serviço WSDL.

Processos de negócio são especificados via BPEL e descrevem a troca de mensagens entre *Web Services*. Estas mensagens são geradas na concretização de operações de um determinado serviço. O contexto de um processo de negócio é formado por uma coleção de mensagens chamadas *containers*, que representam dados que são importantes para a correta execução do processo de negócio.

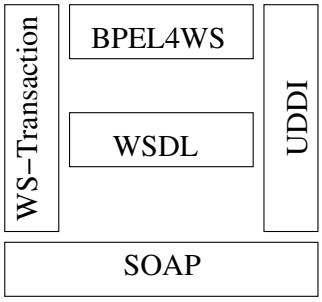

Figura 3.5: *Camadas do Web Service*

A Figura 3.5 situa o BPEL levando em conta as camadas do *Web Service*. O *WS-Transaction* especifica um protocolo para transações de longa duração definidos no BPEL assim como transações atômicas entre *Web Services* (Freund e Storey, 2002). Aplicações criadas com BPEL são chamadas de *process-based applications*. Este tipo de estrutura divide uma aplicação em duas camadas: a camada do topo, que contém o processo de negócio, é escrito em BPEL e representa o fluxo lógico da aplicação; e a camada de baixo, que contém o *Web Service* e representa a função lógica da aplicação (Leymann e Roller, 2002).

Esta estrutura tem muitas vantagens sobre as abordagens convencionais. Por exemplo, o processo de negócio assim como o *Web Service* invocado pode ser trocado sem qualquer impacto nos outros *Web Services* dentro da aplicação ou nos *Web Services* que o processo representa. Além disso, a aplicação pode ser desenvolvida e testada em dois estágios separados. O processo de negócio é desenvolvido e testado separadamente do desenvolvimento e teste de cada *Web Service* da composição. Esta abordagem permite grande flexibilidade na troca ou atualização de uma aplicação.

Aplicações criadas baseadas no BPEL são portáveis entre ambientes que suportam BPEL e *Web Services*. Os processos BPEL podem ser executados por qualquer motor BPEL, e durante sua execução o motor BPEL fará a interação com os *Web Services* que foram encontrados através do processo de busca na UDDI.

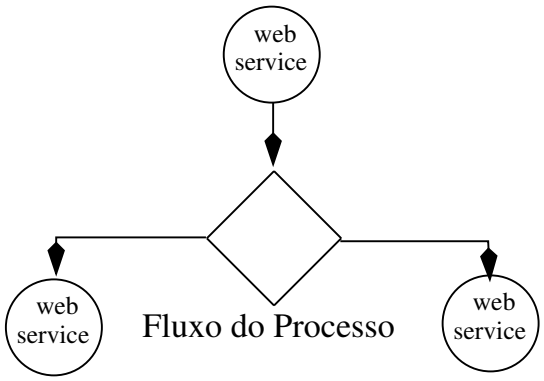

Figura 3.6: *Visão geral da orquestração*

A Figura 3.6 mostra uma visão geral de uma orquestração de *Web Services*, onde o fluxo do processo representa a seqüência lógica das operações e interações de um *Web Service* com os demais envolvidos. Neste fluxo de execução podem acontecer processos sendo executados em paralelo e desvios do fluxo normal de execução. Um processo sempre é controlado e construído baseado na perspectiva de um dos envolvidos. Na orquestração, está incluída a ordem de execução das interações entre os *Web Services*.

A Figura 3.7 apresenta uma visão mais detalhada do fluxo do processo introduzido na Figura 3.6 do ponto de vista de uma entidade envolvida, ou seja, como esta entidade deve interoperar com outros *Web Services* e quais operações devem ser executadas em cada passo do processo, incluindo a ordem dos passos (seqüencial ou em paralelo). Neste exemplo, no passo 1 a entidade recebe uma requisição de um *Web Service*, invoca outro *Web Service* e responde ao *Web Service* que fez a requisição. Na seqüência, é executado o passo 2 seguido pelos passos 3A e 3B simultaneamente. A representação desta orquestração é armazenada na forma de um documento BPEL.

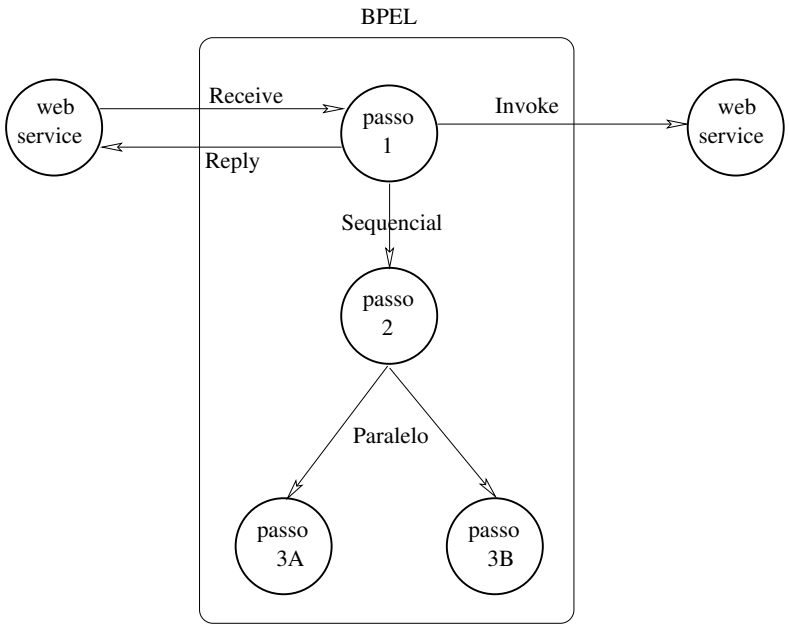

Figura 3.7: *Visão do fluxo do processo por uma entidade*

Os padrões para coreografia e orquestração vêm sendo usados em aplicações para criar e gerenciar composições de *Web Services*. Apesar de coreografia e orquestração serem maneiras complementares para formar composições, muitos autores adotam apenas um destes padrões de composição em seus trabalhos. Uma das razões para isto é que utilizando apenas um padrão, na maioria dos casos, é possível alcançar todos os pré-requisitos para a manutenção e o correto funcionamento da composição, onde é necessário suportar o ciclo de vida completo da composição, incluindo a pesquisa por serviços, o gerenciamento destes serviços, uma possível recomposição dos elementos envolvidos e o contrato de serviços.

A seção seguinte trata das questões de segurança envolvidas na tecnologia de *Web Services*, abordando os principais padrões encontrados na literatura.

3.4 Segurança em Web Services

Para prover segurança em *Web Services*, são necessários diversos mecanismos baseados em XML que garantam a autenticação, controle de acesso, políticas de segurança distribuída e uma camada de comunicação segura, mesmo na presença de intermediários. Atualmente não existe nenhuma solução para segurança em *Web Services* que seja amplamente adotada. A segurança pode ser ponto a ponto ou fim a fim, dependendo da implementação e das necessidades (Booth et al., 2004).

Tecnologias tradicionais de segurança em redes como Secure Socket Layer (SSL), Transport Layer Security (TLS), Virtual Private Networks (VPN), Internet Protocol Security (IPSec) e Secure Multipurpose Internet Mail Exchange (S/MIME) são tecnologias ponto a ponto. Embora essas tecnologias sejam usadas em *Web Services*, elas não são suficientes para prover segurança no contexto fim a fim. Em geral *Web Services* usam uma abordagem baseada em mensagens que trafegam por diversos domínios de segurança, passando por vários intermediários. A Figura 3.8 mostra essa situação.

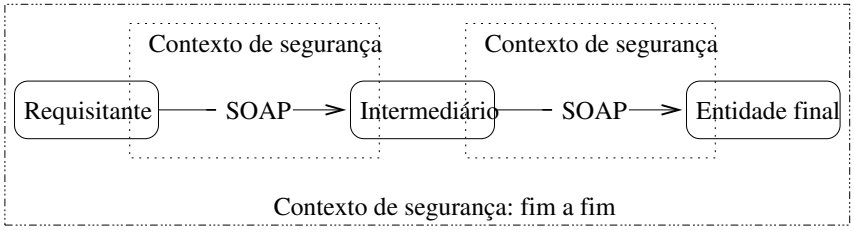

Figura 3.8: *Segurança fim a fim*

Existem diversas formas para um intruso violar mensagens ou parte de mensagens. Podemos citar:

- Quebra da confidencialidade - quando uma entidade não autorizada obtém acesso às informações de uma mensagem ou parte dela. Por exemplo quando um intermediário obtém acesso aos dados do cartão de crédito de uma entidade final.
- Ataque do homem do meio - neste tipo de ataque é possível comprometer um intermediário e então interceptar mensagens entre o *Web Service* e o último receptor. Essas mensagens podem ser modificadas. Técnicas de autenticação mútua podem ser usadas para aliviar este problema.
- *Spoofing* - é um ataque que explora relações de confiança. O intruso assume a identidade de uma entidade confiável para burlar a segurança da entidade alvo. O uso de técnicas de autenticação forte podem ajudar a se defender deste tipo de ataque.

- Negação de serviço - a violação de negação de serviço poder ser obtida pelo uso massivo de um recurso para torná-lo indisponível aos usuários legítimos. Este ataque é fácil de ser implementado e pode causar um estrago significante. Ele pode explorar fraquezas na arquitetura do sistema ou até a sobrecarga causada pelos mecanismos de segurança.

Requisitos de Segurança para Web Services

Assim como em qualquer sistema que atua de maneira distribuída, há muitos desafios quando se tenta construir segurança em *Web Services*. O objetivo principal é criar um ambiente onde as transações possam ser conduzidas de maneira segura, ou seja, existe a necessidade de garantir que as mensagens estejam seguras, com ou sem a presença de intermediários. Existe também a necessidade de garantir a segurança no armazenamento dos dados.

Os requisitos para prover segurança fim a fim em *Web Services* são:

- Mecanismos de Autenticação - a autenticação é necessária para verificar a identidade de clientes e servidores. Dependendo da política de segurança, pode-se autenticar o cliente, o servidor ou usar autenticação mútua.

- Autorização - após a autenticação, os mecanismos de autorização controlam o acesso aos sistemas e seus componentes. As políticas determinam os direitos de acesso para cada entidade do sistema.

- Integridade e Confidencialidade de Dados - a integridade de dados garante que a informação não foi alterada ou modificada durante a transmissão. A confidencialidade dos dados garante que a informação somente será acessada por entidades que tenham permissão.

- Não Repúdio - uma entidade não pode negar que ocorreu uma transação. Utilizam-se assinaturas digitais que garantem a autenticidade de uma mensagem ou transação.

- Auditoria - é um exame que verifica acessos e comportamentos de usuários, de maneira a garantir a integridade do sistema. Agentes podem ser utilizados para fazer essa auditoria. Estes agentes verificarão o sistema, de acordo com as políticas previamente estabelecidas, em busca de violações.

Algumas Considerações de Segurança

As organizações que implementam soluções utilizando *Web Services* devem levar em conta a preocupação com segurança. Estas organizações devem identificar e definir:

- Identidades além de um Domínio - clientes e servidores podem comunicar-se usando vários tipos de verificação de identidade, a partir de domínios de segurança diferentes. Muitos sistemas definem privilégios de acesso baseado em papéis. É importante suportar o mapeamento de identidades através de múltiplos domínios.
- Políticas Distribuídas - políticas de segurança que são associadas a clientes, serviços e mecanismos de busca definem os privilégios de acesso dos clientes. Estas políticas devem ser validadas e verificadas em tempo de execução.
- Políticas de Confiança - são políticas distribuídas que aplicam-se a ambientes multi-domínios. Uma entidade cliente precisa confiar no ambiente do serviço e a entidade provedora de serviços precisa confiar no ambiente do cliente. Políticas de confiança podem ser transitivas, ou seja, se *a* confia em *b*; e *c* confia em *a* então *c* confia em *b*. Mecanismos de confiança podem ser usados para formar delegações e federações. Estes mecanismos podem ser utilizados para facilitar interações seguras entre *Web Services*, em ambientes de larga escala.
- Mecanismo de Busca Seguro - no mecanismo de busca seguro, é necessário a apresentação de uma identidade do cliente para fazer a sua autenticação e estabelecer um canal de comunicação seguro entre o cliente e o serviço de registro e pesquisa (SRP). O mecanismo de busca, em alguns casos, pode ser anônimo como por exemplo em serviços de busca em redes par-a-par.
- Mensagens Seguras - o uso de mensagens seguras garante privacidade, confidencialidade e integridade. Técnicas de assinaturas digitais podem ser usadas para assegurar o não repúdio. As técnicas de cifragem e assinatura de mensagens podem ser usadas em roteamento e em trocas confiáveis de mensagens, como mostra a Figura 3.9.

3.4.1 Mecanismos e padrões usados na segurança de *Web Services*

O padrão *XML-Signature* (Eastlake et al., 2002; Reagle, 2000) foi desenvolvido pela W3C e pelo IETF (RFC 2807 e RFC 3275) para assinatura digital em documentos XML. São usadas assinaturas em XML, autenticação, integridade de dados e suporte para não repúdio dos dados assinados.

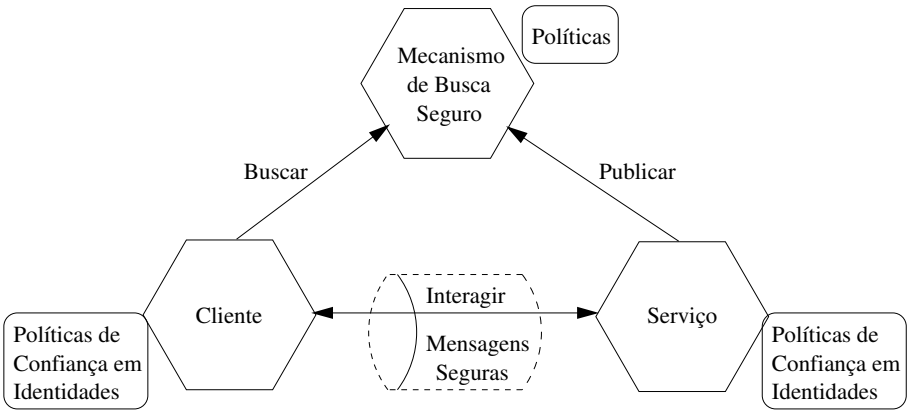

Figura 3.9: *Busca Segura*

O *XML-Signature* tem a habilidade para assinar somente partes específicas da árvore XML. Isto é importante quando um documento XML simples precisa ser assinado múltiplas vezes por uma entidade simples ou por múltiplas partes. Esta flexibilidade assegura a integridade de certas partes de um documento XML, enquanto deixa aberta a possibilidade para outras partes do documento ser modificada. A verificação da assinatura implica que o dado que estava assinado esteja acessível à parte que interessa na transação. A assinatura XML irá geralmente indicar a localização do objeto original assinado. O padrão *XML-Signature* não faz uso de nenhum algoritmo especial, ele utiliza algoritmos para assinatura digital conhecidos como o RSA e DSA.

O *XML-Encryption* (Reagle, 2002; Imamura et al., 2002) especifica um processo para cifragem de dados e diz como representar o resultado em XML. Os dados podem ser arbitrários, como um elemento XML ou o conteúdo de um elemento XML. O resultado da cifragem é um elemento *XML-Encryption* que contém ou referencia a informação cifrada.

Web Services Security

O *Web Services Security* (Nadalin et al., 2004) (WSS) foi desenvolvido pela OASIS[1] e define uma extensão SOAP para garantir proteção, utilizando as propriedade de integridade, confidencialidade e autenticação de mensagens. Os mecanismos do WSS podem ser usados para acomodar uma grande variedade de modelos de segurança e tecnologias de criptografia.

O padrão fornece um mecanismo geral para associação de segurança em mensagens SOAP. A especificação não requer um tipo específico de algoritmo criptográfico. O WSS descreve, entre outras

[1] Organization for the Advancement of Structured Information Standards (`http://www.oasis-open.org/`)

coisas, como codificar certificados X.509 e *tokens kerberos* (Booth et al., 2004).

A seguir apresentamos um exemplo simples de uma mensagem SOAP com o cabeçalho *WS-Security*. Este exemplo consiste em enviar uma credencial "paulo", sem qualquer tipo de proteção. Na linha 2, são informadas as URIs para os espaços de nomes XML do SOAP e da *WS-Security*. O elemento <wsse:Security> (linhas 5-9) pode expressar informações sobre a cifragem, assinatura e sobre as credenciais de segurança. As linhas 6-8 expressam detalhes sobre uma credencial de segurança. Contudo, estes elementos <wsse:Security> podem conter mais de uma credencial de segurança.

```
1  <soapenv:Envelope
2     xmlns:soapenv="..." xmlns:wsse="...">
3     <soapenv:Header>
4
5        <wsse:Security>
6           <wsse:UsernameToken wsu:Id="...">
7              <wsse:Username>paulo</wsse:Username>
8           </wsse:UsernameToken>
9        </wsse:Security>
10
11    </soapenv:Header>
12    <soapenv:Body>
13       ...
14    </soapenv:Body>
15 </soapenv:Envelope>
```

Exemplo de Mensagem SOAP com o WSS

A especificação WSS define uma estrutura de segurança fim a fim que provê suporte para processamento seguro, mesmo com a presença de intermediários. A integridade das mensagens é garantida usando *XML-Signature* para garantir que mensagens sejam transmitidas sem modificações. Os mecanismos de integridade suportam múltiplas assinaturas, por múltiplas entidades. As técnicas são extensíveis para que elas possam suportar formatos adicionais de assinaturas. A confidencialidade é garantida usando *XML-Encryption* para manter partes confidenciais da mensagem SOAP.

XML Key Management Specification (XKMS)

XKMS (Ford et al., 2001) é um conjunto de operações oferecidas para gerenciar uma infraestrutura de chaves públicas (PKI). Esta especificação é baseada em XML e é própria para um sistema que usa criptografia de chave pública na cifragem, assinatura, autorização e verificação de autenticidade de informação. Este serviço especifica protocolos para distribuição e registro de chaves públicas, sendo apropriado para uso em conjunto com os padrões *XML-Signature* e *XML-Encryption*.

O uso do XKMS permite que os desenvolvedores da aplicação passem a tarefa de registro e validação de chaves para uma entidade confiável, fora da lógica da aplicação. Isto simplifica a implementação da aplicação e o trabalho para manipular chaves públicas e privadas fica restrito a um serviço especializado: o XKMS.

Uma entidade confiável XKMS deve trabalhar com qualquer sistema PKI, onde a informação passa entre ele e o *Web Service*. Com esta entidade, a implementação do *Web Service* se mantém simples.

Security Assertion Markup Language (SAML)

O SAML (Cantor et al., 2005) é um padrão XML, desenvolvido pela OASIS, que suporta o uso de assinaturas simples. O SAML permite que um usuário efetue o login apenas uma vez com o *Web Service* que, por sua vez, conduz o negócio com seus parceiros (outros serviços). O SAML pode ser usado em transações negócio-negócio ou negócio-cliente.

Existem três componentes básicos SAML: asserção, protocolo e associação. A asserção pode ser de autenticação, de atributo ou de autorização. A asserção de autenticação valida a identidade do usuário. A asserção de atributo contém informações específicas sobre o usuário. A asserção de autorização identifica o que o usuário está autorizado a fazer. O protocolo define como o SAML envia e recebe asserções. Há diversos mecanismos de associação no SAML. Alguns definem como a troca de mensagens SAML devem ser mapeadas para SOAP, HTTP, SMTP e FTP entre outros.

eXtensible Access Control Markup Language (XACML)

O XACML (Godik e Moses, 2003) é um sistema de políticas de propósito geral, baseado em XML e desenvolvido pela OASIS para garantir a interoperabilidade entre os diversos sistemas. O XACML descreve uma linguagem para políticas de controle de acesso, seguindo o modelo requisição resposta. Esta linguagem é utilizada para definir quem possui direitos de acesso sobre o que. O formato de requisição resposta descreve como as consultas (pedidos) sobre o sistema de políticas deverão ser realizadas e como deverão ser as respostas.

WS-Trust

A WS-Trust (Anderson et al., 2005) é uma proposta que visa principalmente a troca de atributos de segurança, possibilitando a comunicação através de domínios de segurança diferentes. Esta pro-

posta está sendo desenvolvida por um conjunto de empresas, entre elas Microsoft e IBM. A WS-Trust baseia-se no caso em que os atributos de segurança, considerados válidos pelo provedor do serviço, já estejam contidos no pedido originado pelo cliente ao serviço. O serviço pode indicar, ao cliente, os mecanismos necessários para a implementação dos controles através da WS-Policy (Bajaj et al., 2004).

A WS-Trust define ainda o serviço de atributos de segurança (*Security Token Service - STS*) como a autoridade responsável por emitir, renovar e validar os atributos de segurança, sendo este a base do modelo de confiança. O STS consiste de um serviço *Web* que implementa uma interface WSDL, especificada pela WS-Trust, e processa mensagens SOAP seguras.

3.5 Conclusões do Capítulo

Este capítulo apresentou uma descrição da tecnologia de *Web Services*, sua arquitetura básica e os padrões de segurança adotados. Foram relatadas também as dificuldades para integração de aplicações *Web* e como é possível utilizar a tecnologia de *Web Services* para resolver tais problemas.

O SOAP 1.2 provê um modelo no qual características adicionais podem ser inseridas via cabeçalhos, porém não existe uma especificação de quais categorias de funcionalidades ou mesmo do formato específico que deve ser adotado.

Com o uso dos padrões para orquestração e coreografia em *Web Services*, é possível integrar diversas aplicações de maneira a estabelecer um fluxo de processo, ou seja, como estas aplicações devem interagir entre si.

A comunicação entre *Web Services* é feita em ambientes diversos, com ou sem a presença de nodos intermediários. Para garantir a segurança nas transações entre *Web Services* é necessário o uso de técnicas que garantam a segurança fim a fim das mensagens. O padrão *Web Services Security* especifica quais padrões de segurança devem ser adotados.

No próximo capítulo é apresentado o conceito de sistemas de detecção de intrusão, e também são introduzidos alguns dos sistemas de detecção de intrusão existentes.

Capítulo 4

Sistemas de Detecção de Intrusão

Nos últimos anos, observou-se um aumento do número de vulnerabilidades descobertas em sistemas computacionais e conseqüentemente um aumento do número de intrusões nestes sistemas. Diversos sistemas de detecção de intrusão (*Intrusion Detection Systems - IDS*) têm sido desenvolvidos para ajudar as equipes de administração de rede a detectar possíveis violações ou tentativas de violação dos sistemas computacionais e muitas vezes evitar tais violações.

Este capítulo trata essencialmente de sistemas de detecção de intrusão. Primeiramente introduzimos algumas definições de sistemas de detecção de intrusão presentes na literatura e o contexto histórico da evolução destes sistemas. A seguir, descrevemos os elementos básicos que formam um IDS. Em seguida, apresentamos algumas abordagens para coleta de dados. Finalizamos este capítulo com alguns exemplos de sistemas existentes, mostrando suas características.

4.1 Definições

A detecção de intrusão pode ser definida como uma tentativa de identificar as tentativas de intrusão em sistemas computacionais (McHugh, 2001). Segundo Sandhu (Sandhu e Samarati, 1996) detecção de intrusão corresponde à auditoria de sistema feita *on-line* (em tempo de execução). Sistemas de detecção de intrusão, conhecidos como IDSs, são programas de computador elaborados para detectar possíveis intrusões, fazendo a comparação do comportamento observado nas atividades computacionais contra padrões normais de comportamento, preferencialmente em tempo de execução (Janakiraman et al., 2003).

4.2 Evolução dos Sistemas de Detecção de Intrusão

Na metade da década de 1960, como o custo do tempo de processamento era elevado, tiveram início as primeiras auditorias em sistemas computacionais. Essas auditorias eram baseadas exclusivamente na perícia do olhar humano (McHugh, 2001). Em 1972, James Anderson (Anderson, 1972) publicou o *Computer Security Technology Planning* onde levantou questões sobre o que deve ser detectado, como fazer a análise e como proteger o sistema e seus dados contra ataques e violações.

Em 1980, Anderson publicou outro relatório (Anderson, 1980) para um cliente que processava grande quantidade de dados usando *mainframes*. A equipe de segurança fez um manual detalhado de dados para serem auditados, procurando informações que pudessem indicar violações de segurança. Porém, devido ao grande volume de dados processados, tal tarefa consumia bastante tempo, além de não ser trivial pelo fato de que algumas informações necessárias não eram capturadas e algumas informações eram capturadas repetidas vezes. Anderson sugeriu, então, formas para reduzir ou restringir a quantidade de dados a ser analisada, estabelecendo que dados estatísticos do comportamento de usuários e grupos fossem comparados com um perfil pré definido, para assim detectar comportamentos anormais.

Em 1984 Dorothy Denning (Denning, 1986) e Peter Neumann iniciam o projeto IDES (*Instrusion Detection Expert System*), que foi um dos mais significantes no início das pesquisas sobre IDSs. O modelo do IDES é baseado na suposição de que é possível estabelecer perfis para caracterizar a interação normal de usuários com objetos (arquivos, programas ou dispositivos). Os perfis são modelos estatísticos do comportamento de usuários e o sistema tenta detectar comportamentos anormais. Os perfis são complementados por um sistema que usa uma base de regras para descrever atividades que representam violações de segurança conhecidas. Com isso, evita-se que um usuário possa gradualmente treinar o sistema para aceitar comportamentos ilegais. Denning sugere um certo número de perfis em determinadas atividades que deve ser útil para detectar intrusões, tais como:

- Login e atividades da sessão: definem-se modelos para freqüência de logins, duração das sessões, local dos logins.
- Comandos ou comportamento de execução de programas: definem-se modelos para freqüência de execução de vários comandos, recursos usados pelos comandos, falhas de permissão e exaustão de recursos.
- Atividades de acesso a arquivos: definem-se modelos para freqüências de acesso, comportamento de leitura e escrita, falhas de acesso e exaustão de recursos.

No final da década de 1980, um número notável de sistemas foi desenvolvido, em sua maioria, uma combinação de abordagens estatísticas e sistemas especialistas. Num destes sistemas, denominado NADIR (Hochberg et al., 1993), os motores de análise incorporaram sistemas de gerenciamento de bases de dados comerciais tais como Oracle e Sybase, aproveitando a vantagem da habilidade que tais sistemas possuem para organizar os dados a serem auditados. O NADIR continua em uso e está em atividade de refinamento para acomodar novas ameaças (outros tipos de vulnerabilidades) e para adaptar-se aos novos sistemas alvos.

Outro destes sistemas, chamado MIDAS, foi desenvolvido pelo *National Computer Security Center* para monitorar seus sistemas Multics e Dockmaster (Sebring et al., 1988). É um sistema híbrido que usa a abordagem de análise estatística para examinar registros de dados do Multics, que controlava o login de usuários. O MIDAS foi um dos primeiros sistemas que trabalhou conectado à Internet e ficou em uso de 1989 até a metade da década de 1990.

Dentre os sistemas que surgiram na época, o NSM (*Network System Monitor*) inovou bastante. Ele monitorava o tráfego de rede diretamente no segmento ethernet e usou isto como sua fonte primária de dados para análise (Heberlein et al., 1990). Hoje a maioria dos IDSs monitoram a rede para coletar dados para uma primeira análise.

Atualmente, alguns problemas persistem nos sistemas de detecção de intrusão. Podemos citar a dificuldade para obter dados que estejam livres de intrusões para treinamento dos sistemas[1] e o alto índice de falsos alarmes (Jansen et al., 1999).

A seguir apresentamos os elementos que formam um IDS, seguindo alguns modelos presentes na literatura.

4.3 Modelo de IDSs

Todos os IDSs seguem um modelo padrão. Eles possuem módulos diferentes, mas os elementos (componentes) principais são os mesmos, independente da arquitetura ou do método de detecção utilizado. Nos últimos anos, um grande esforço vem sendo feito no sentido de criar um padrão de nomenclatura e padronizar as funções destes elementos para facilitar a integração de diferentes sistemas de detecção de intrusão. O CIDF[2] (*Common Intrusion Detection Framework*) (Staniford-Chen, 1998) define um modelo de elementos (componentes) de um IDS contendo:

[1] Testes de software para confirmar que o software está respondendo corretamente a determinada entrada
[2] http://www.isi.edu/gost/cidf/

- **geradores de eventos**: elemento responsável pela geração de eventos e pela padronização do formato de dados.
- **analisadores de eventos**: elemento responsável pela detecção de intrusão. Ele recebe os dados dos geradores de eventos e faz uma análise buscando padrões que caracterizam um possível ataque.
- **bases de dados de eventos**: elemento responsável pelo armazenamento dos eventos.
- **unidades de resposta**: elemento responsável pelas ações a serem tomadas como respostas à incidentes. Pode ser um bloqueio de conexão, um sinal para encerrar algum processo, etc.

Outro grupo que trabalha para a padronização é o IDWG (*Intrusion Detection Working Group*) (Wood, 2002). A Figura 5.1 mostra a disposição dos elementos de um IDS segundo o IDWG.

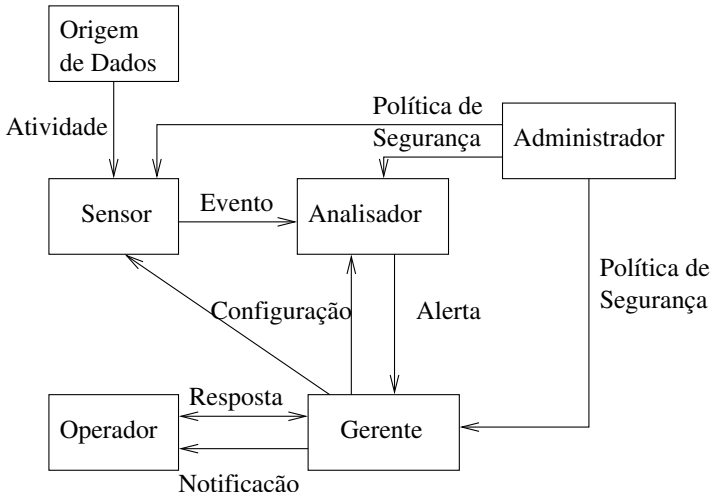

Figura 4.1: *Componentes de um IDS segundo o IDWG*

Onde:

- **origem de dados**: elemento onde ocorrem as atividades de sistema.
- **sensor**: elemento responsável pela geração de eventos para auditoria.
- **analisador de eventos**: elemento responsável pela detecção de intrusão. Ele recebe os dados dos sensores e faz uma análise, buscando anomalias ou padrões que caracterizam um possível ataque.
- **gerente**: elemento responsável pela configuração do sistema e por notificar os operadores.

- **operador**: pessoa responsável por analisar os alertas; ele pode responder ou não ao ataque.
- **administrador**: pessoa responsável por definir as políticas de segurança.

Neste modelo, observamos que o sensor, o analisador e o gerente devem estar presentes em todos os sistemas IDSs pois correspondem à coleta, análise dos dados e tomada de decisão, que são funções básicas de qualquer IDS. Tomaremos este modelo do IDWG como base para todos os modelos apresentados e para o modelo desenvolvido como resultado desta dissertação.

Em alguns sistemas esses elementos podem ser decompostos em vários sub-elementos, mas sempre preservando as suas funções básicas. Alguns modelos definem, além destes três, outros elementos como interfaces, geradores de alarmes, etc.

O uso de respostas ativas, ou seja, respostas do sistema ao ataque em tempo real como por exemplo reconfiguração de firewall ou encerramento de um processo, em sistemas de detecção de intrusão é perigoso. Se o sistema reage automaticamente recusando uma conexão de rede ou de uma máquina que acredita-se ser a responsável pelo ataque, os atacantes têm a oportunidade de criar ataques de negação de serviços, fazendo um *spoofing*[3] de uma fonte crítica.

Respostas onde o sistema faz, apenas, uma coleta de informações e identificação do sujeito, sem modificação (em tempo real) do sistema são bastante usadas e evitam problemas como os citados. Muitos IDSs optam pelas respostas passivas, ou seja, o sistema apenas envia um relatório (geralmente por email) informando os possíveis problemas detectados naquele período, deixando a responsabilidade de reação ao administrador da rede.

Na seção seguinte apresentamos as duas principais abordagens para coleta de dados: baseada em *hosts* e na rede. Em seguida apresentamos as abordagens para detecção de intrusão baseadas em anomalias e em assinaturas.

4.4 Abordagens para Coleta de Dados

Nos primeiros esforços na área de detecção de intrusão foram usadas técnicas de auditoria em sistemas operacionais, aplicadas aos dados a serem analisados. Com o desenvolvimento das pesquisas, definiu-se dois tipos distintos de abordagens para a coleta de dados, são eles: baseados em máquinas (*hosts*) e baseados na rede.

[3] Atividade em que o atacante forja o seu endereço IP, se fazendo passar por outra interface de rede

4.4.1 Coleta de dados baseada em *hosts*

As primeiras propostas para detecção de intrusão eram baseadas no uso de técnicas de auditoria de dados na máquina que seria monitorada. O TCSEC (Brand, 1985) (*Trusted Computer System Evaluation Criteria*) estabeleceu alguns critérios para auditoria, começando com uma listagem dos eventos que devem ser auditados. Isso inclui o uso de identificação e autenticação, introdução de objetos dentro do espaço de endereçamento dos programas (abertura de arquivos ou execuções de programas), remoção de objetos, ações administrativas e outros eventos relevantes à segurança. Podemos definir a coleta de dados baseada em *host* como sendo uma forma de coletar dados em máquinas, através da análise de seus registros de eventos, de seus arquivos ou de seus processos em execução.

A maioria dos sistemas de coleta baseados em *host* coletam dados de forma contínua, de acordo com o uso do sistema operacional. Porém, coletas periódicas de estados do sistema também podem prover dados para revelar mudanças inesperadas. Estes sistemas de coleta, muitas vezes modulares dentro do IDS, não devem expor os dados coletados e para isso necessitam de certa proteção. Eles também devem tomar cuidado para não expor os arquivos de registros de eventos.

As primeiras literaturas sobre detecção de intrusão presumem que alguns conjuntos de dados auditados são suficientes para detectar intrusões. Mesmo se a atividade intrusiva não se manifestar diretamente, é possível detectar alguma atividade suspeita. Por exemplo, se uma entrada para um processo (*daemon*) de rede foi formada de maneira errada, isto pode causar a morte daquele processo de maneira anormal. A auditoria deve descobrir o que causou a morte súbita daquele processo. Se o armazenamento dos eventos do sistema não for feito, não é possível determinar se houve erros na construção daquele processo ou de algum outro componente do sistema.

A escolha de itens a serem auditados e o nível de detalhes a ser registrado leva a uma polarização em um processo de detecção de intrusão. Se a técnica de análise envolve somente a procura por padrões específicos que indicam intrusão, em alguns casos será impossível detectar certas intrusões. Em sistemas que tentam detectar padrões não usuais, os efeitos podem ser mais sutis. Quanto mais detalhada for a coleta, maior será o impacto na performance do sistema e maior será o espaço necessário para guardar estes dados.

Análises posteriores (*offline*) também serão afetadas pela quantidade de dados coletados. Por outro lado, coletar poucos dados aumenta o risco de haver manifestações ou ataques e estes não serem detectados.

As coletas de dados em *hosts* podem ser feitas através de: busca de erros ou violações nos registros de eventos do sistema; identificação de alterações nos arquivos de programas do sistema

(por exemplo, a alteração do código de um programa do sistema ocasionando uma alteração na data de sua criação); verificação dos acessos de processos às bibliotecas do sistema.

4.4.2 Coleta de dados baseada na rede

Outra forma de coletar dados é observar o tráfego de rede e procurar por sinais de intrusão nestes dados. Esta abordagem tem a vantagem de que um simples sensor pode monitorar várias máquinas e desta forma, com maior facilidade, detectar ataques que buscam por brechas de segurança em tais máquinas.

Porém, existe a desvantagem de não poder detectar ataques que são feitos a partir de um console ligado ao *host*. Além disso, é muito difícil de detectar ataques a serviços que utilizam criptografia no canal de comunicação. Apesar desses problemas, o monitoramento de rede é o método escolhido pela maioria dos sistemas de detecção de intrusão. Existem algumas razões para isso. Podemos citar a facilidade de construir uma plataforma dedicada que combina sensores de rede com redução de dados e análise. É necessário apenas uma interface de rede para ”enxergar“ todo o tráfego na rede na qual está conectada e alguns mecanismos para capturar o tráfego ou uma parte dele. A biblioteca *libpcap* (Jacobson et al., 1994) é freqüentemente usada para construção de programas deste tipo.

Infelizmente, o tráfego nas redes está aumentando muito rápido, a capacidade de transmissão nas redes passou de 10 megabits/s para 100 megabits/s e muitas já utilizam 1000 megabits/s. Como a velocidade para análise dos dados (CPU dos computadores) não aumentou muito, uma análise detalhada do tráfego em uma rede gigabit causa um atraso considerável. Os sensores de rede também são alvos de ataques. É comum fazer a coleta e análise em uma única plataforma, tornando o IDS vulnerável.

O protocolo TCP/IP tem várias ambiguidades que são resolvidas de maneira diferente por sistemas operacionais distintos. Usando retransmissão, sobreposição e fragmentação de pacotes, uma análise de intrusão pode apresentar resultados diferentes para o IDS e para o sistema alvo.

A seguir apresentamos as abordagens para detecção de intrusão freqüentemente usadas, após a coleta de dados no *host* ou na rede.

4.5 Abordagens para Detecção de Intrusão

4.5.1 Detecção de Intrusão Baseada em Anomalias

A abordagem da detecção de intrusão baseada em anomalias assume que intrusões serão acompanhadas de manifestações que não são usuais e assim permitem a sua detecção. Entretanto, nem sempre isto ocorre. Segundo Anderson (Anderson, 1980) e Myers (Myers, 1980), é necessário estar ciente das altas habilidades que um intruso pode ter e que este intruso tentará entender o sistema e evitar a sua detecção.

Esta técnica de detecção por anomalias baseia-se no comportamento de sujeitos (principais) e programas, onde é estabelecido um padrão de comportamento com as atividades que sujeitos e programas costumam praticar, Qualquer atividade que ocorra fora deste padrão pré-estabelecido será considerado como atividade suspeita. Se for atingido um certo nível (porcentagem) de atividade suspeita é gerado um alerta. Desta forma é possível identificar novos padrões de comportamento malicioso (atividade maliciosa) sem que exista um conhecimento prévio deste tipo de atividade.

Os sistemas de detecção por anomalias podem ser divididos em dois tipos: os que realizam auto-aprendizagem e os que recebem uma programação prévia da base de conhecimento, suficiente para indicar uma violação de segurança, sem modificar esta base de conhecimento de forma automática. A auto-aprendizagem é capaz de construir sua própria base de conhecimento do comportamento "normal" de sujeitos e programas. Estes sistemas podem utilizar séries temporais (estatísticas de um conjunto discreto de estados ao longo do tempo), podem fazer a modelagem por regras (onde o sistema estuda, por exemplo, o tráfego e formula um número de regras que descrevem a operação normal deste sistema) ou ainda podem usar estatísticas descritivas (onde o sistema coleta estatísticas simples a partir de certos parâmetros para um perfil e constrói um vetor de distância entre o tráfego observado e o perfil criado). Os sistemas que usam séries temporais podem empregar técnicas como as de redes neurais e as técnicas inspiradas nos sistemas imunológicos.

A vantagem da abordagem de detecção por anomalias é que esta permite a detecção de intrusões através de vulnerabilidades até então desconhecidas. Por outro lado, é difícil definir e representar as atividades normais de um principal. Esta abordagem gera muitos falsos positivos (alarmes falsos) e consome muitos recursos computacionais para guardar e verificar as atividades no sistema que está sendo monitorado.

4.5.2 Detecção por Assinaturas

A detecção de intrusão por assinaturas utiliza uma base de assinaturas de atividades intrusivas (conhecidas). As informações que chegam ao sistema (por exemplo, o tráfego de rede) são postas contra esta base de assinaturas a fim de identificar (casar) a atividade atual com alguma assinatura. Para que um IDS baseado em assinaturas possa detectar intrusões, ele precisa de descrições (assinaturas) de intrusões, com os seus respectivos ataques. Estas descrições podem ser tão simples quanto um padrão específico, que alcança uma parte dos pacotes de rede ou tão complexo quanto um estado de uma máquina ou a descrição de uma rede neural, que mapeia múltiplos sensores para representações abstratas (McHugh, 2001).

Quando uma dada assinatura é associada com uma abstração conhecida de ataque, é relativamente fácil, para um detector baseado em assinaturas, ligar atividades a tipos conhecidos de ataques (ex. Ping da morte) (Axelsson, 2000). A família STAT de detectores (Vigna et al., 2000), utiliza uma representação abstrata de algumas classes de ataques na tentativa de reconhecer novos ataques e gerar suas respectivas assinaturas. Em geral, a base de assinaturas destes sistemas precisa ser atualizada quando novos ataques são descobertos. A maioria dos sistemas comerciais segue esta categoria e são freqüentemente atualizados.

Lee (Lee et al., 1999) desenvolveu modelos, utilizando a abordagem de *data-mining*, que são utilizados em alguns algoritmos para minimizar os dados a serem auditados. Essa abordagem funciona muito bem para detectar ataques conhecidos, mas falha para novas categorias de ataques. Essa abordagem não se adapta a novos ambientes, ou seja, a coleta feita em um ambiente não se adapta em outro ambiente.

Estes sistemas, baseados em assinaturas, sofrem de alarmes falsos quando as assinaturas são amplas, pertencendo tanto a intrusões como a atividades normais. Assinaturas podem ser desenvolvidas de várias formas, desde a tradução das manifestações de ataques até treino automático ou aprendizado usando informações dos sensores. A próxima seção apresenta as principais características de alguns sistemas de detecção de intrusão presentes no mercado.

4.6 Exemplos de Sistemas IDS

O **Real Secure** da ISS[4] é um IDS híbrido e sua arquitetura é dividida em três partes, consistindo de um motor de reconhecimento baseado na rede, um motor de reconhecimento baseado em *host* e

[4]http://www.iss.net

um módulo administrativo.

O motor de reconhecimento baseado em rede é executado em uma máquina dedicada para prover detecção de intrusão e respostas. A monitoração de um segmento de rede envolve a procura por pacotes que combinem com assinaturas de ataques. Quando um motor de reconhecimento de rede detecta atividade intrusiva, ele pode responder terminando a conexão, enviando alertas, salvando a sessão, reconfigurando o *firewall*, etc. Ele envia também um alarme ao módulo administrador ou ao console.

O motor de reconhecimento baseado em *hosts*, analisa os registros de eventos para reconhecer ataques. Cada *host* examina seus registros de sistema em busca de evidências de intrusões. Este motor pode prevenir a morte súbita de processos de usuários ou acabar suspendendo contas de usuários, podendo também tomar ações similares ao motor baseado em rede. Os vários motores de reconhecimento são gerenciados pelo módulo administrativo. Assim é possível configurar e administrar de um único lugar.

O **Tripwire**[5] é um programa para verificação dos efeitos de uma intrusão, onde é criada uma base de dados de informações críticas ao sistema no qual inclui tamanho de arquivos e os *checksums* de cada arquivo. Na detecção, o *tripwire* compara a informação corrente com a gerada anteriormente e identifica as mudanças nos arquivos. São reportados os arquivos que foram modificados, porém o administrador deve decidir quais as modificações devem ter sido causadas por intrusão.

O **Snort**[6] (Roesch, 1999) é um programa de detecção de intrusão de código aberto, leve, eficiente, funciona em qualquer sistema *Unix* e utiliza uma abordagem baseada em rede. Pode ser usado para análise de protocolos de rede e também para detectar uma variedade de ataques ou tentativas de ataques como estouro de buffer, listagem de portas, ataques de código CGI, tentativas de identificação do sistema operacional ou ataques aos serviços de rede. O *snort* tem a capacidade de enviar alertas em tempo real. Estes alertas podem ser registrados no servidor de registros *syslog*, em arquivo de alerta ou enviados ao administrador.

A comunidade de usuários e desenvolvedores contribui constantemente para a criação de ferramentas auxiliares e novas assinaturas. Como resultado, novos ataques são rapidamente descobertos e novas assinaturas são criadas para os mesmos. A comunidade de desenvolvedores do *snort* "abraçou" o esforço do IETF, através do grupo de pesquisa IDWG[7] para a especificação do Intrusion Detection Message Exchange Format (IDMEF) (Debar et al., 2006) para padronizar as mensagens de alerta.

[5] http://www.tripwire.com
[6] http://www.snort.org
[7] Intrusion Detection Working Group

Desta forma, os sensores do *snort* são capazes de comunicar-se facilmente com outros IDSs e sistemas de gerência que suportam este padrão.

Em seguida apresentamos os sistemas de detecção de intrusão que atuam de forma distribuída, também conhecidos como *Distributed Intrusion Detection Systems - (DIDS)*.

4.6.1 Projetos de IDSs Distribuídos - DIDS

Com o crescimento das redes nas organizações, surgiu a necessidade de um sistema de detecção de intrusão distribuído. Os IDSs comerciais geralmente concentram a detecção de intrusão somente dentro de uma organização. São raros os projetos onde é possível fazer a comunicação entre vários detectores, distribuídos em diversas organizações (ambientes heterogêneos e de larga escala). Podemos citar algumas características desejáveis nestes sistemas (Balasubramaniyan et al., 1998). São elas:

- O sistema deve executar continuamente sem a interferência humana.
- O sistema deve ser tolerante a faltas (capaz de sobreviver a *crashs* e reinicializações de componentes).
- O sistema deve ser resistente, capaz de fazer um monitoramento em si mesmo e detectar possíveis ataques.
- Deve causar um *overhead* mínimo no sistema onde o IDS está sendo executado.
- Deve ser capaz de se adaptar às políticas de segurança locais (onde ele está executando).
- Deve ser capaz de se adaptar às mudanças de comportamento do sistema e dos usuários.
- Deve ter escalabilidade para monitorar uma grande quantidade de *hosts*.
- Deve ter uma baixa degradação no mesmo, no caso de algum componente parar de funcionar.
- Deve permitir a habilidade de reconfiguração dinâmica sem a necessidade de reinicialização do sistema.

Nos últimos anos, surgiram novos projetos de pesquisa de IDSs distribuídos. Estes projetos procuram manter uma arquitetura distribuída, utilizando diversas técnicas (protocolos) para comunicação e buscando a duplicação de seus componentes para evitar a indisponibilidade do serviço. Podemos citar alguns projetos:

EMERALD

O EMERALD (*Event Monitoring Enabling Responses to Anomalous Live Disturbances*)[8] faz coleta de dados no *host* e na rede, utilizando um sistema baseado no comportamento de usuários e também um sistema baseado em padrões de assinaturas para a detecção de intrusão (Porras e Neumann, 1997). O EMERALD é um modelo de IDS distribuído muito citado na literatura. O objetivo principal neste projeto foi o de prover detecção de intrusão em redes empresariais grandes e pouco acopladas.

O EMERALD faz uso de componentes pequenos e distribuídos em diferentes domínios para prover monitoramento, detecção e resposta a incidentes de segurança. A estratégia do EMERALD é utilizar monitores distribuídos que possam analizar e responder a atividades maliciosas nos locais que elas acontecem e possam interoperar para formar uma hierarquia de análise. Esta hierarquia é dividida em três níveis (camadas de monitoramento) e fornece um *framework* para reconhecimento de ameaças globais, incluindo tentativas coordenadas para infiltrar ou destruir a conectividade em redes. Esta hierarquia de camadas para monitoramento de rede inclui a análise de serviços abrangendo desde o mal uso de componentes individuais e serviços de rede dentro de um domínio até o mal uso de múltiplos serviços e componentes entre múltiplos domínios.

É introduzido o conceito de monitores de serviços dinamicamente organizados e altamente distribuídos. Monitores de serviços são organizados em redes locais para análisar a sua infra-estrutura (roteadores, gateways) e os seus serviços. Monitores de serviços podem interagir com seus ambientes passivamente (verificando atividades em registros de eventos) ou ativamente (acessando e testando os serviços em busca de vulnerabilidades ou intrusões). Este monitoramento de serviços de rede dentro de domínios forma a camada mais baixa do esquema de monitoramento do EMERALD. A segunda camada do esquema de monitoramento do EMERALD é a análise em escala de domínio. O monitor de domínio é responsável pelo monitoramento de todas as partes daquele domínio. Os monitores de domínio correlacionam alertas enviados por monitores locais, fornecendo uma perspectiva da atividade (ou padrões de atividade) em nível do domínio. O monitor do domínio é responsável pela reconfiguração dos parâmetros do sistema, trocando informações com outros monitores de domínios e reportando ameaças ao administrador do domínio a que pertence.

A terceira camada é a análise em nível empresarial (em ambientes de larga escala), fornecendo uma abstração global das atividades nos domínios. Estes monitores correlacionam atividades que aconteceram nos domínios monitorados. Este monitor visa detectar ameaças em escala global, por

[8] http://www.csl.sri.com/projects/emerald/

exemplo um ataque de vírus na Internet ou ataques coordenados de muitos domínios contra um único domínio.

Informações correlacionadas por um monitor de serviços podem ser disseminadas para outros monitores através de um modelo de comunicação *publish/subscribe*, disseminando os resultados de forma assíncrona. Desta forma os monitores do EMERALD, distribuidos em um domínio e se comunicando de forma assíncrona, são capazes de disseminar relatórios de atividades maliciosas de maneira mais eficiente do que nos modelos síncronos. Estes monitores tem a mesma arquitetura básica nas três camadas sendo compostos por um conjunto de perfis para detecção por anomalias, um conjunto de assinaturas para a análise baseada em assinaturas e um componente para integrar o resultado gerado. Existe também a possibilidade de integração de módulos desenvolvidos por terceiros.

O EMERALD foi desenvolvido para atuar em ambientes de larga escala, agindo em tempo real e foi o primeiro a introduzir a idéia de usar monitores distribuídos para monitorar ambientes de larga escala de forma eficaz, possibilitando a identificação de ataques distribuídos através da correlação de pequenos eventos. Em desenvolvimento nos últimos vinte anos, o EMERALD reúne uma vasta experiência em detecção de intrusão distribuída, porém não faz uso do padrão IDMEF para troca de mensagens de alertas.

AAFID

O AAFID é um sistema de detecção de intrusão que utiliza agentes autônomos para tentar resolver os problemas de configuração, escalabilidade e eficiência, encontrados em outros IDS (Balasubramaniyan et al., 1998). A arquitetura do AAFID possui três componentes: agentes, *transceivers* e monitores. Fazendo uma comparação com o modelo de IDS desenvolvido pelo IETF, os agentes seriam os sensores, os *tranceivers* seriam os analisadores e os monitores seriam os gerentes.

Os agentes podem prover mecanismos para auto reconfiguração sem precisar reiniciá-los e também podem ser testados antes de serem introduzidos em um ambiente mais complexo. Estes agentes fazem a coleta de dados na rede e a análise é feita, baseada em assinaturas, pelos *transceivers*. Se um agente pára de funcionar, podem acontecer duas coisas:

- Se o agente for independente, apenas os seus dados coletados serão perdidos.
- Se os dados deste agente forem usados por outros agentes, pode haver problemas com esses outros agentes também.

Os agentes executam de maneira independente e fazem o monitoramento dos *hosts*, reportando anormalidades apenas aos *transceivers*, não podendo comunicar-se com outro agente. Estes agentes podem aprender, migrar de máquina e serem escritos em qualquer linguagem de programação.

Os *transceivers* formam a interface externa de comunicação de cada *host*. Eles tem a função de controle e processamento dos dados. Eles podem inicializar ou desativar agentes, receber relatórios gerados pelos agentes e distribuir informações para outros agentes ou para um monitor, podendo também reportar para mais de um monitor, a fim de prover redundância e resistência a falhas.

Os monitores recebem dados de agentes e *transceivers* e fazem a correlação de eventos suspeitos, podendo detectar intrusões que, por algum motivo, não foram reportadas por alguns *transceivers*. Eles podem controlar entidades que estão executando em outros *hosts*. Se dois monitores estão controlando o mesmo *transceiver*, é necessário um mecanismo para resolver o problema de consistência das informações e o comportamento adotado por ele. O AAFID não especifica nenhum mecanismo a ser adotado nestes casos. As Figuras 4.2 e 4.3 mostram respectivamente a organização física e lógica dos componentes do AAFID.

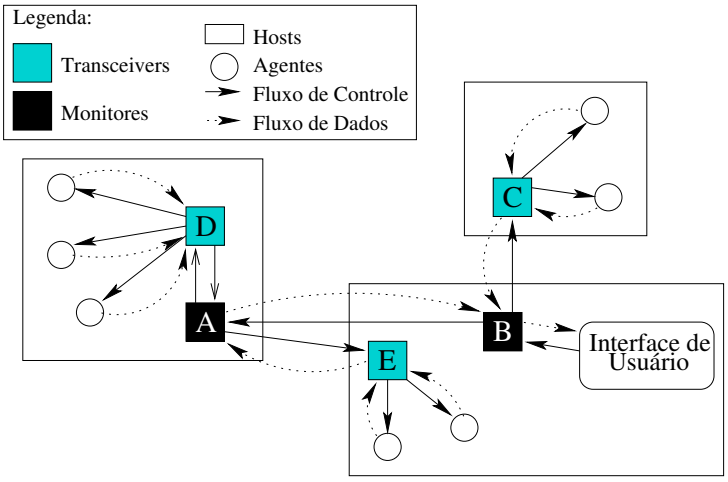

Figura 4.2: *Estrutura física dos componentes do AAFID*

A arquitetura do AAFID possui a vantagem de conseguir monitorar redes heterogêneas e de larga escala com o uso de seus agentes. Isto se deve ao fato do AAFID ter sido implementado na linguagem de programação Perl, que é interpretada e pode ser executada em diversas plataformas, possibilitando a execução de seus agentes em ambientes distribuídos e heterogêneos. Algumas desvantagens da arquitetura do AAFID são:

- O monitor no maior nível hierárquico é um ponto único de falha.

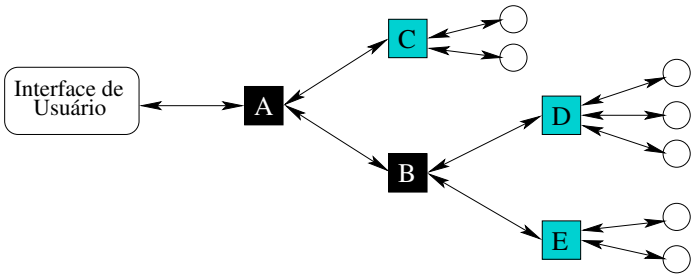

Figura 4.3: *Estrutura lógica dos componentes do AAFID*

- Se for utilizado mais de um monitor controlando o mesmo *tranceiver*, é necessário um mecanismo para manter a consistência dos dados neste *tranceiver*.
- A arquitetura não especifica mecanismos para controle de acesso, para permitir que diferentes usuários possam ter diferentes níveis de acesso ao IDS.

DOMINO

O DOMINO (*Distributed Overlay for Monitoring InterNet Outbreaks*) é um sistema de detecção de intrusão heterogêneo, escalar e robusto contra ataques e falhas. Uma rede DOMINO é uma infraestrutura dinâmica composta por uma coleção de diversos nós localizados através da Internet em um sistema distribuído em várias organizações ou em uma organização virtual. O DOMINO é um sistema que faz uso de uma rede *overlay* para detecção de intrusão de forma cooperativa, organizado em duas camadas: um pequeno centro de nós confiáveis e uma grande coleção de nós conectados ao centro. Este IDS faz a detecção de intrusos que utilizam endereços IP clonados e tenta reduzir o número de falsos positivos utilizando listas negras para controle de acesso ao sistema que ele monitora.

O objetivo do DOMINO é disponibilizar um *framework* para compartilhamento de informação a fim de prover a capacidade de detecção de intrusão para todos os participantes (Yegneswaran et al., 2004). Este *framework* possui as seguintes características: disponibilidade, descentralização, heterogeneidade e privacidade. A Figura 4.4 mostra a organização dos nós em uma rede overlay *peer-to-peer*.

Esta rede é composta por três conjuntos de participantes: nós concentradores, comunidades satélites e nós terrestres.

- Nós concentradores - são os componentes centrais da arquitetura do DOMINO. Eles são responsáveis pelo cruzamento e correlação dos dados. Eles se comunicam entre si de maneira

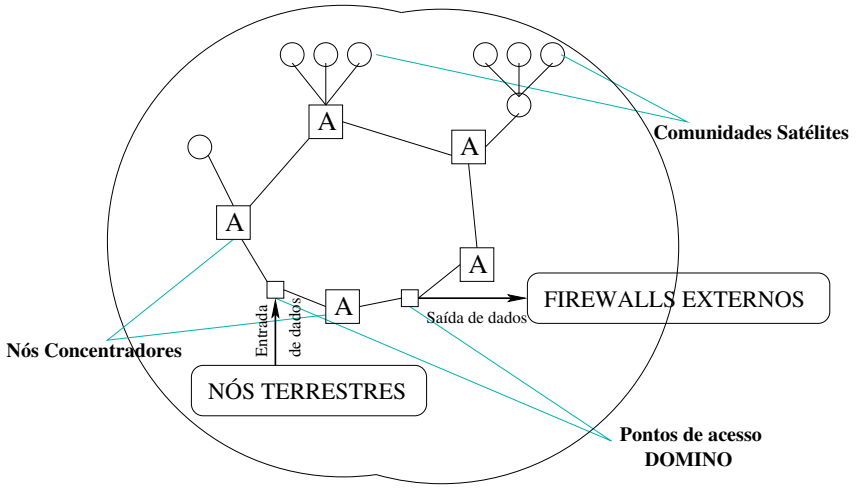

Figura 4.4: *Organização dos nós do DOMINO*

segura.

- Comunidades satélites - são pequenas redes de nós satélites que implementam uma versão local do protocolo DOMINO. Os nós satélites estão organizados em uma hierarquia de sensores. Cada nó possui comunicação com seu pai. Eles simulam máquinas virtuais para atrair/monitorar o tráfego da rede.

- Nós terrestres - são os mais confiáveis e possuem uma grande quantidade de dados. Estes nós não implementam o protocolo DOMINO, eles servem para expansão da cobertura, incluindo conjuntos de dados sobre intrusões para fora da infra-estrutura.

- Mensagens DOMINO - o protocolo de mensagens do DOMINO é representado em XML.

O DOMINO usa uma abordagem de coordenação centralizada onde um centro de nós confiáveis coordenam o resto dos nós distribuídos em diversas redes. Desta forma é possível gerar alarmes rapidamente quando ataques de larga escala acontecem, além de reduzir o número de alarmes falsos. Por outro lado, com esta centralização pode haver uma inundação destes nós centrais, no caso de ataques por vírus ou outras pragas digitais que atuam de maneira uniforme gerando muitos alertas para estes nós centrais processarem ou até mesmo em ataques distribuídos direcionados aos nós centrais. Outro ponto a ser destacado é que o DOMINO não produz respostas aos ataques, apenas detecta e gera os alertas, deixando aos administradores a tarefa de conter ou eliminar os ataques. O DOMINO é bastante usado em *honeypots* distribuídos que buscam identificar possíveis ataques distribuídos contra determinadas redes ou apenas gerando estatísticas da atividade maliciosa em um determinado período.

INDRA

O INDRA *Intrusion Detection and Rapid Action* é um sistema IDS distribuído, baseado no compartilhamento de informações entre pares confiáveis em uma rede, para proteger a rede como um todo contra tentativas de intrusão. Este sistema faz coleta na rede e utiliza uma base de assinaturas para detecção de intrusão. O objetivo principal do INDRA é distribuir as informações sobre tentativas de intrusão em uma rede P2P (*peer-to-peer*) e atuar de forma colaborativa nesta rede (Janakiraman et al., 2003). Isso permite que o sistema reaja, aplicando correções, desconectando temporariamente serviços ou desconectando máquinas que possam estar comprometidas, para limitar os danos causados.

Os sistemas *peer-to-peer* são rápidos para busca e difusão de informações. A Figura 4.5 mostra como funciona a subscrição dos nós. Neste exemplo, os nós listrados fazem a subscrição para receber informações sobre vulnerabilidades do SSH e os nós em preto para receber informações sobre ataques de negação de serviço.

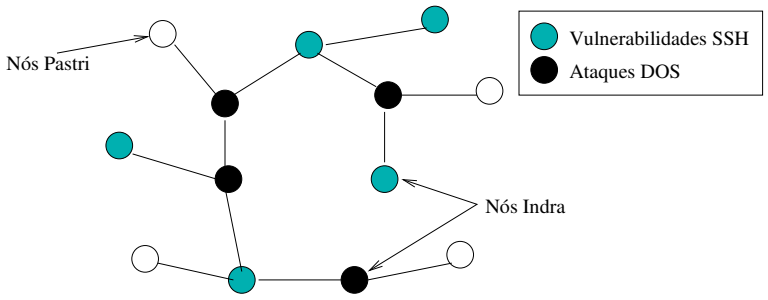

Figura 4.5: *Subscrição de nós no INDRA*

Uma vantagem significativa dos sistemas distribuídos como o INDRA é que eles podem ser usados para balanceamento de carga, ou seja, distribuindo as informações de alertas em vários *hosts* da rede. O funcionamento do sistema é fundamentado na posse de certificados. O sistema possui uma relação de confiança com servidores chaves de onde o INDRA obtém os certificados para seus pares. As redes de confiança são montadas fazendo uso de certificados (Stallings, 1994). A Figura 4.6 ilustra o funcionamento da rede de confiança no INDRA.

Os nós estão conectados por relações de confiança. Cada relação possui um valor que indica o grau de confiança.

O INDRA possui um conjunto de *daemons* (processos) que fornecem toda a funcionalidade para o sistema, sendo que estes processos estão presentes em cada nó da rede *peer-to-peer*. Estes

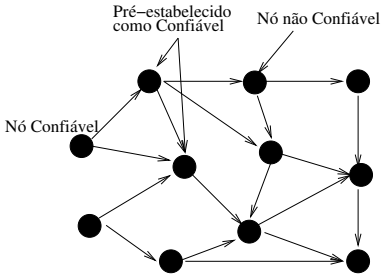

Figura 4.6: *Rede de confiança*

processos são responsáveis pela troca de informações entre os pares da rede *peer-to-peer* e também por atualizar a base de controle de acesso na sua memória. Para isso, esta rede *peer-to-peer* precisa ser de confiança. A Figura 4.7 mostra os *daemons* do INDRA:

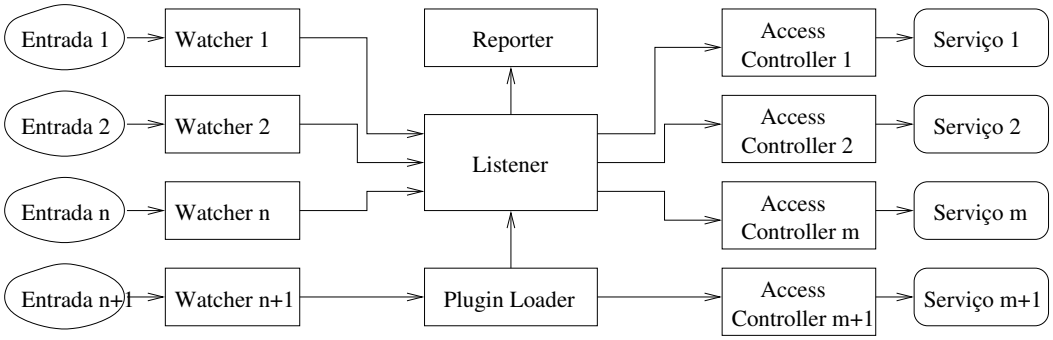

Figura 4.7: *Daemons do INDRA*

- *Watchers* - estão no primeiro nível, na vigilância das atividades suspeitas. Eles atuam tanto no sistema local como na rede.
- *Access Controlers* - esses *daemons* provêem o controle de acesso aos recursos.
- *Listeners* - eles funcionam como filtros, recebendo e agregando os alertas gerados pelos *watchers* e então, dependendo da política de segurança definida pelo administrador, eles enviam as notificações aos *access controlers*.
- *Reporters* - são responsáveis pela comunicação com outros *hosts*. Eles agregam as notificações e repassam para outros *hosts*.
- *Plugin loader* - faz a carga local de *daemons* adicionais. Este processo pode ser executado em qualquer momento.

O INDRA é um IDS ainda em desenvolvimento que traz a idéia nova de utilizar redes *peer-to-peer* confiáveis para a troca de informações entre seus pares. Esta comunicação pode acontecer entre redes heterogêneas, porém esta troca de mensagens não segue um padrão, de forma que outros softwares de segurança não podem fazer parte desta rede. O INDRA age de forma ativa, ou seja, o sistema automaticamente toma providências para conter ou eliminar uma ameaça, reduzindo o tempo de resposta, porém corre-se o risco de errar e acabar bloqueando algum acesso legítimo. O INDRA foi implementado na linguagem de programação Java, podendo ser executado em redes que possuem diversos sistemas operacionais (heterogêneas).

Intrusion Detection Force (IDF)

O IDF é uma infra-estrutura virtual que permite a detecção de intrusão na escala da Internet. O objetivo principal do IDF é defender organizações e proteger a Internet como um todo (Teo et al., 2003), através do compartilhamento de informações inter-organizações de forma segura, provendo inteligência na resposta e análise de dados.

O IDF define quatro propriedades básicas: compartilhamento da informação, segurança, escalabilidade e sobrevivência. Apresenta ainda outras propriedades como interoperabilidade, extensibilidade, integração do IDF com outros sistemas, prover uma plataforma segura de desenvolvimento. A infra-estrutura do IDF tem entidades e grupos de entidades distribuídas que trabalham em diferentes áreas e sistemas de um ambiente de larga escala. A Figura 4.8 apresenta as entidades na arquitetura do IDF.

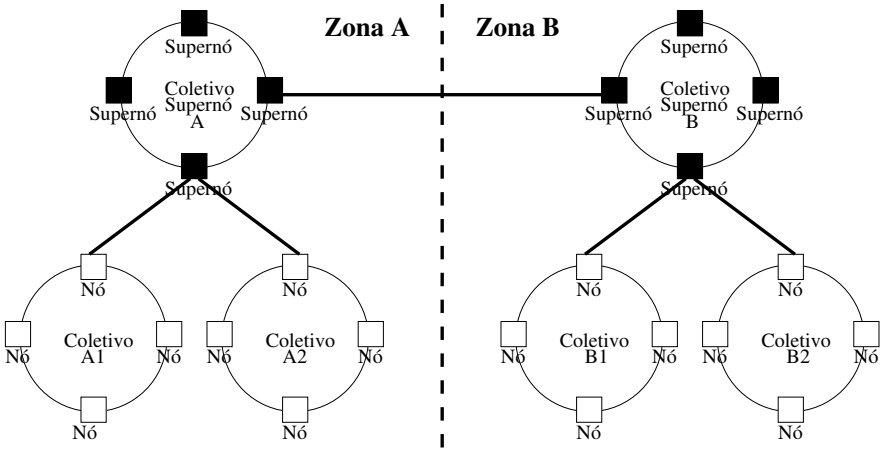

Figura 4.8: *Entidades do IDF*

- **Nó**: um nó é um *host* no qual está sendo executado um agente de software IDF. Um nó compartilha informações com outros nós.
- **Coletivo** - é uma coleção de nós.
- **SuperNó** - é um nó especial que provê serviços de alto nível para o coletivo, tal como operações de uso intensivo de CPU.
- **SuperColetivo** - uma coleção de supernós, atuando da mesma maneira para alcançar flexibilidade entre supernós.
- **Zonas** - uma área da rede ou da Internet sobre a autoridade de um supercoletivo. Facilita a gerência e administração a fim de alcançar escalabilidade.

A Figura 4.9 mostra todos os componentes de um nó do IDF.

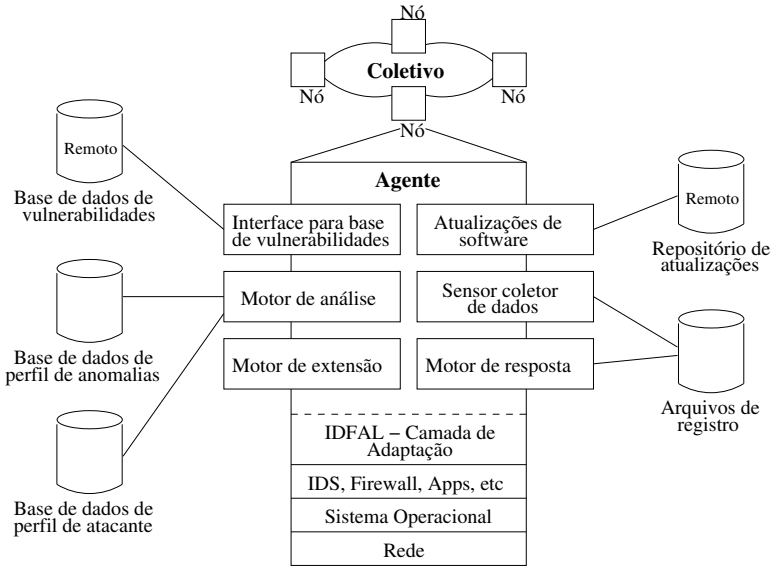

Figura 4.9: *Componentes do nó IDF*

Em cada nó do sistema, existem componentes responsáveis por todas as tarefas do IDF. Esses componentes são:

- IDFAL (*IDF Adaptation Layer*) - faz a interface entre diversos IDSs, aplicações e o sistema operacional. Provê uma plataforma genérica e uma interface independente de rede para os componentes do IDF de mais alto nível. Concentra modularidade e independência de plataforma.

- Sensor Coletor de Dados - faz a coleta dos eventos do host e da rede. Tem a função de registrar (*logs*) eventos de um *host* específico e repassar os dados ao supernó em intervalos de tempo.

- Motor de Análise - ao nível do nó faz o processamento normal de um IDS. Ao nível de supernó, existe o processamento de alto nível que inclui a detecção de comportamentos suspeitos encontrados na sua zona ou em outras zonas.

- Motor de Resposta - pode atuar de maneira ativa ou passiva. Quando ocorreu alguma violação, o motor de resposta envia os resultados ao administrador da rede.

- Interface para Base de Dados de Vulnerabilidades - provê informações sobre atualizações para os outros componentes tal como o motor de análise e o motor de atualização de software.

- Motor de Atualizações de Software - é responsável por propagar atualizações de software para os nós do IDF de maneira rápida. Assegura que as atualizações foram entregues de maneira segura. As atualizações devem ser autenticadas antes de serem aplicadas.

- Motor de Extensão - permite o desenvolvimento de novas funcionalidades ao IDF. As extensões devem ser autenticadas e certificadas antes de serem usadas.

O IDF é um IDS distribuído bastante flexível, faz coleta e processamento de dados de forma distribuída, utiliza coleta de dados no *host* e na rede, faz a análise baseada em assinaturas de vulnerabilidades conhecidas e sua resposta pode ser ativa ou passiva. Uma vantagem deste sistema é que ele atua em tempo real, possibilitando respostas aos ataques no momento que eles acontecem. Desta forma é possível evitar que invasões ou danos maiores ocorram no sistema atacado. O IDF, assim como a maioria dos IDSs, utiliza um formato próprio para a troca de mensagens. Desta forma não é possível a integração de outras ferramentas de segurança e dificulta a sua integração com outras ferramentas, uma vez que seria necessário um tradutor de mensagens específico.

Prelude

O Prelude é um IDS híbrido, ou seja, faz análise na máquina e na rede. Sistemas híbridos agregam as vantagens de ambos sistemas, possibilitando um índice de acerto maior e uma melhor identificação de incidentes de segurança (Zaraska, 2003). O Prelude possui mecanismos de detecção de intrusão que usam assinaturas e também mecanismos que não usam assinaturas. O sistema baseado em assinaturas utiliza uma base de dados de assinaturas de vulnerabilidades conhecidas e permite a utilização de assinaturas desenvolvidas para outros IDSs (por exemplo, é possível utilizar a base

de assinaturas do Snort). Essas assinaturas são regras que identificam determinado tipo de ataque conhecido.

O Prelude permite o uso de diversas aplicações de segurança (analisador de registros, IDS de rede, analisadores de aplicações) que reportam eventos detectados para o seu gerente. O Prelude funciona de maneira distribuída, é rápido e eficiente (difícil de enganar) no processo de análise de dados. Permite o uso de vários sensores como o *snort, honeyd, nessus, samhain*, mais de 30 tipos de sistemas de *logs*. Ele funciona em qualquer sistema POSIX (Unix).

O Prelude utiliza um sistema gerente centralizado e segue o modelo hierárquico, onde sensores fazem a coleta e análise de dados e enviam para seus gerentes. Estes gerentes podem ser replicados, evitando assim a ocorrência de faltas. Para a comunicação de eventos, ele utiliza o IDMEF [9], um padrão IETF que permite que diferentes tipos de sensores gerem eventos, utilizando uma mesma linguagem de comunicação.

O Prelude foi implementado seguindo os objetivos:

- Modularidade - a funcionalidade é dividida em módulos, cada módulo executa uma função e então é possível alterar o sistema apenas incluíndo ou retirando um módulo.
- Suporte ao padrão IDMEF - o uso do IDMEF permite a integração de diversas ferramentas de segurança.
- Seguir um modelo híbrido de detecção - utilizar sensores de rede e sensores de *host* para a coleta de informações.
- Confiabilidade - garantia de entrega de mensagens.
- Extensibilidade - garantir o uso de qualquer outra aplicação de segurança para servir ao Prelude como um sensor.

A Figura 4.10 ilustra a arquitetura básica do Prelude.

A arquitetura básica do Prelude é dividida em alguns componentes: diversos sensores, analisadores, um ou mais gerentes, um repositório e uma aplicação que fornece uma interface para o sistema. Estes componentes são descritos a seguir.

O Prelude NIDS é um sensor baseado em rede do Prelude. Ele monitora a rede em busca de padrões de ataques. Ele usa um mecanismo baseado em uma base de assinaturas, suporta fragmentação

[9] Intrusion Detection Message Exchange Format

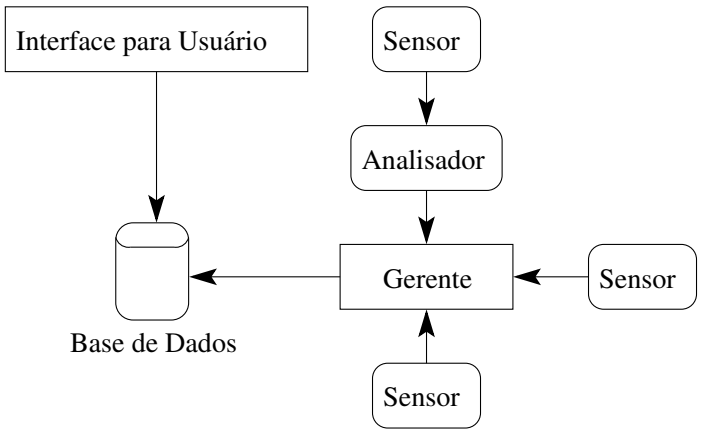

Figura 4.10: *Arquitetura Básica do Prelude*

IP e segmentação TCP para rastrear as conexões. Esse sensor é responsável pela detecção de intrusão e geração de um relatório de eventos, que é enviado para um servidor centralizado (gerente), utilizando SSL[10]. O cliente, onde está sendo executado o sensor, precisa estar registrado no gerente para poder enviar os dados.

O Prelude Manager (gerente) é um servidor que faz a coleta e normalização das informações recebidas dos sensores. Ele é o programa principal do Prelude, recebe alertas de diversos sensores e analisadores, guarda os alertas em uma base de dados ou qualquer formato suportado por um *plugin*, faz a análise centralizada e provê replicação. Durante a filtragem dos eventos recebidos, ele permite que uma ação específica, para um evento específico, seja tomada. Os alertas são formatados no padrão IDMEF.

A Figura 4.11 apresenta a estrutura de funcionamento dos sensores, mostrando como estes sensores se comunicam com o gerente (Prelude Manager).

O Prelude Import (Importador de dados) é uma ferramenta com a função de importar dados de aplicações que reportam eventos em 3 formatos específicos:

- IDMEF-XML - importa dados no formato IDMEF-XML e converte para o formato IDMEF binário, nativo do Prelude
- Nessus XML - importa vulnerabilidades reportadas pelo Nessus
- IDMEF Object - um formato IDMEF específico do Prelude

[10]Secure Socket Layer

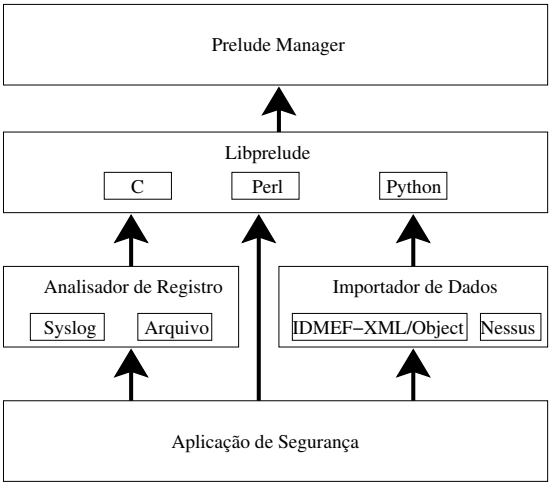

Figura 4.11: *Estrutura Básica do Prelude*

O Prelude LML é um analisador baseado na máquina, utiliza uma base de assinaturas para analisar os registros do sistema e mensagens recebidas pelo sistema de registros *syslog*. Ele trabalha com diversos programas de segurança, incluíndo *Clamav, Honeyd, ipchains, Netfilter, pf, Nagios, ssh*, etc. O Prelude LML possui como função principal a análise de registros. Ele pode ser usado em sistemas *unix*, *switches*, roteadores, *firewalls*, etc, operando de duas maneiras: verificando arquivos de registro na máquina onde está executando ou recebendo mensagens UDP de sistemas de registro *syslogs* de outras máquinas na rede. Ele suporta o formato do *syslog* dos *BSD e outros formatos utilizando a biblioteca *pcre*. Este sistema envia um alerta ao Prelude Manager quando uma atividade suspeita é detectada.

O PreludeDB Library é uma biblioteca para acesso ao banco de dados onde são guardados os registros do Prelude. Essa biblioteca possui uma abstração dos comandos SQL.

A Figura 4.12 ilustra a arquitetura completa do Prelude. No caso de falha do gerente local, o sensor pode se comunicar diretamente com o gerente centralizador. Outros elementos podem ser adicionados a esta arquitetura.

A integração do Prelude com outros programas acontece com o auxílio da biblioteca *libprelude*, que também provê suporte às operações internas do Prelude. Toda a comunicação entre os módulos do prelude utiliza essa biblioteca, que provê recuperação de falhas, utiliza eventos assíncronos, interfaces temporizadas e um *plugin* genérico que torna possível utilizar outros programas como sensores do Prelude. A comunicação entre elementos do Prelude acontece de maneira segura.

O Prelude é um sistema tolerante a falhas, ou seja, caso ele não consiga chegar ao gerente ou

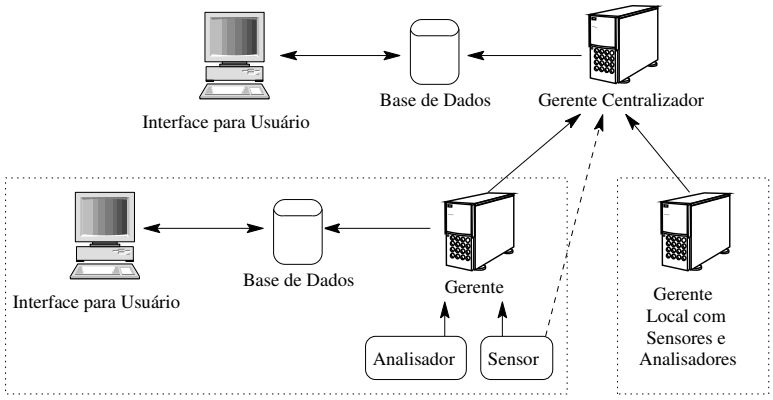

Figura 4.12: *Arquitetura Completa do Prelude*

por problemas na rede ou por problemas do próprio gerente ele irá gravar os dados em um arquivo local e tentar retransmitir mais tarde.

4.6.2 Comparação Entre as Abordagens Apresentadas

A comparação entre sistemas de detecção de intrusão não pode ser feita baseada nos detalhes de cada IDS, mas sim nas suas principais características como: formas para coleta de dados, mecanismos para análise e detecção de intrusão, modelos de comunicação para IDSs distribuídos, etc. A Tabela 4.1 apresenta uma comparação entre as principais funcionalidades e características relevantes, para este trabalho, dos sistemas de detecção de intrusão abordados neste capítulo.

Tabela 4.1: *Comparação entre IDSs*

IDS	Coleta de Dados	Método de Detecção	Plataforma	Suporta Padrão IDMEF	Executa em Ambiente Distribuído
Real Secure	rede	assinaturas	Windows/Unix	não	não
Tripwire	host	anomalias	Windows/Unix	não	não
Snort	rede	assinaturas	Unix	sim	não
EMERALD	rede/host	anomalias/assinaturas	Unix	não	sim
AAFID	rede	assinaturas	Windows/Unix	não	sim
DOMINO	rede	assinaturas	Unix	não	sim
INDRA	rede	assinaturas	Windows/Unix	não	sim
IDF	rede	assinaturas	Unix	não	sim
Prelude	rede/host	assinaturas	Unix	sim	sim

Nesta tabela observamos a forma como é feita a coleta de dados, em quais sistemas operacionais (plataformas) este IDS funciona, se o formato IDMEF para troca de mensagens é suportado, e se o IDS atua em ambientes distribuídos e heterogêneos. Em geral, os sistemas de detecção de intrusão

que fazem coleta de dados na rede possuem dois problemas: altas taxas de falsos positivos e uma visão limitada de redes grandes e heterogêneas, restringindo a habilidade para detectar ataques. Da mesma forma, sistemas que coletam dados apenas no *host* ficam limitados às informações contidas naquele *host*, deixando de identificar possíveis ataques ou tentativas de ataques através da rede ou em softwares que não reportam tais tentativas. O uso de sistemas de detecção de intrusão híbridos (que fazem a coleta de dados tanto na rede quanto no *host*) têm se mostrado mais eficazes na detecção de intrusão ou tentativas de intrusão tanto para redes pequenas quanto para redes maiores.

A detecção de intrusão baseada em assinaturas é amplamente usada pelos IDSs, por ser mais flexível e fácil de usar em qualquer tipo de rede (grande ou pequena). A dificuldade e o custo computacional para gerar bases de comportamento individuais para detecção por anomalias é grande, impossibilitando o uso de tais técnicas em IDSs que operam em redes de larga escala.

A maioria dos IDSs analisados neste capítulo são propostas que foram desenvolvidas no meio acadêmico, empregando o uso de sistemas operacionais Unix e com o seu código fonte aberto. Contudo, as técnicas para coleta de dados, detecção de intrusões e comunicação de alertas introduzidas por estes sistemas, podem ser empregadas em sistemas comerciais no futuro. Estes sistemas são desenvolvidos com suporte a outras plataformas como o Windows, amplamente usado em ambientes empresariais.

4.7 Conclusões do Capítulo

Analisando o histórico dos IDSs, podemos observar que houve um aumento da pesquisa e desenvolvimento destes sistemas na última década. Tal crescimento vem acompanhado de um aumento expressivo do número de incidentes reportados aos centros de resposta a incidentes de segurança, o que nos induz a acreditar que atualmente o uso de IDSs integrados a outros *softwares* de segurança é indispensável para manter uma rede sem a presença de intrusos.

Os diversos IDSs utilizam métodos (heurísticas) diferentes para a análise e detecção de intrusos. Para fazer uma boa análise de dados, em sistemas de detecção de intrusão, é importante o uso de mais de um método de análise. Diferentes métodos produzem diferentes respostas e se for possível fazer a união de vários métodos, pode-se chegar a um resultado mais preciso e confiável, reduzindo bastante o número de falsos positivos gerados.

Os sistemas de detecção baseados em análise de comportamento conseguem identificar intrusões inéditas ou desconhecidas pelo sistema. Já os sistemas de detecção baseados em assinaturas

precisam de atualização constante da base de assinaturas e são dependentes de entidades que disponibilizam tais assinaturas.

No próximo capítulo é apresentado um modelo para composição de IDSs usando a tecnologia de *Web Services* e o modelo de comunicação segura e interoperável desenvolvido para a comunicação e integração de elementos de IDS em ambientes de larga escala.

Capítulo 5

Modelo de Comunicação para Composição de IDSs

A comunicação entre sistemas de detecção de intrusão (IDS) e outras ferramentas de segurança é um processo complexo e muitas vezes difícil de se concretizar. As diversas ferramentas existentes no mercado utilizam protocolos proprietários para comunicação interna entre seus componentes e no envio de registros de alertas. Geralmente a comunicação interna destas ferramentas é feita com os registros gerados no formato de texto puro, mas nem sempre seguindo um padrão.

Na interação entre IDSs para o compartilhamento de registros ou na interação com outras ferramentas (por exemplo *chkrootkit*[1]) necessita-se que seja estabelecida uma mesma "linguagem de comunicação", ou seja, que utilizem um protocolo padrão para a comunicação. Este protocolo está em processo de padronização pelo IETF com o nome de IDMEF[2].

Da literatura examinada neste trabalho, pouco se viu sobre IDSs transpondo os limites de um domínio administrativo. O avanço tecnológico envolvendo o uso da Internet trouxe novas formas de operações, aproximando empresas e propondo novas abordagens de *e-business*; são as organizações virtuais (OVs) que começam a se fazer presentes em grandes aplicações através da Internet. Estas associações virtuais empregam muitas vezes a composição envolvendo recursos de várias empresas, formando aplicações que são mantidas através de recursos dispostos em vários domínios administrativos. Foi com base nestes novos contextos de empresas virtuais que em (Brandão et al., 2005) foram introduzidos IDSs distribuídos que fazem uso de tecnologias como *Web Services*, que são próprios para estas organizações virtuais.

[1] http://www.chkrootkit.org/

[2] Intrusion Detection Message Exchange Format

Neste capítulo apresentamos o modelo que dá suporte à composição de IDSs através de *Web Services*. Este modelo envolve a comunicação segura entre elementos de detecção de intrusão em ambientes de larga escala. O modelo é apropriado no trato de sistemas abertos e faz uso de padrões de segurança e da tecnologia de Web Services (Booth et al., 2004) para conseguir a interoperabilidade na integração de IDSs.

A aplicação do modelo de comunicação segura permite a integração de elementos de detecção de intrusão distribuídos entre sub-redes de uma mesma organização ou entre organizações distintas para formar IDSs de larga escala. Tais sistemas de larga escala são formados através da composição de IDSs completos ou elementos de IDSs, atuando como sensores e analisadores.

Neste capítulo, primeiramente introduzimos o modelo de composição de IDSs usando *Web Services*. Descrevemos também o procedimento geral desenvolvido para a criação e a manutenção de composições de IDSs. Em seguida apresentamos o modelo para a comunicação segura entre os elementos de IDS.

5.1 Composição de IDSs usando Web Services

A composição de IDSs é uma nova abordagem para construção de sistemas de detecção de intrusão de larga escala, onde IDSs completos ou elementos de detecção de intrusão interagem entre si para compor um sistema de monitoramento muito mais amplo e diversificado do que o obtido com o emprego dos IDSs tradicionais (distribuídos ou não). As composições podem ser criadas para atender às necessidades de segurança de uma única organização ou se estender por diferentes organizações que desejam compartilhar informações e alertas de segurança.

São adotados como elementos básicos de um IDS, aqueles definidos no modelo de detecção de intrusão do IDWG/IETF (Wood, 2002): sensores, analisadores e gerentes, como ilustrados na Figura **??**. No modelo proposto, estes elementos básicos podem estar disponíveis para participar de uma ou mais composições de sistemas de detecção de intrusão. IDSs completos podem agir também como sensores ao enviarem alertas a outros elementos da composição. Também podem agir como analisadores ao receberem e estabelecerem correlação entre alertas provenientes de elementos distribuídos através de redes de larga escala. Como o conceito de Composição de IDSs comporta que um mesmo elemento de IDS possa ser integrado a mais de uma composição, o papel de cada elemento irá depender da forma como os mesmos são organizados em cada composição.

O modelo definido permite a construção de composições de IDSs para coletar dados de determinados sensores, pesquisar diversas bases de dados de eventos ou para compartilhar informações

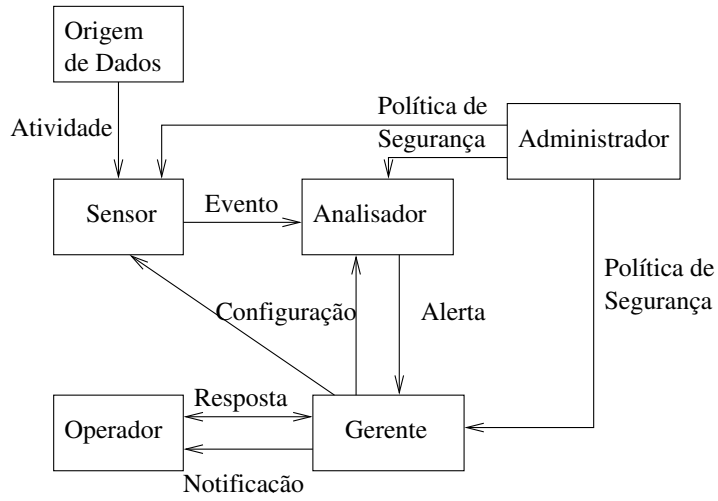

Figura 5.1: *Componentes de um IDS segundo o IDWG*

sobre ataques em andamento. Estas composições são dinâmicas e podem ser facilmente modificadas, permitindo a adaptação a novas situações em ambientes distribuídos de larga escala.

Para permitir a interação entre os elementos de uma composição de IDSs foi criada uma infra-estrutura de serviços (Brandão et al., 2006), ilustrada na Figura 5.2, seguindo a arquitetura orientada a serviços suportada pela tecnologia de *Web Services* (Booth et al., 2004), com o amplo emprego de XML (Bray et al., 2000).

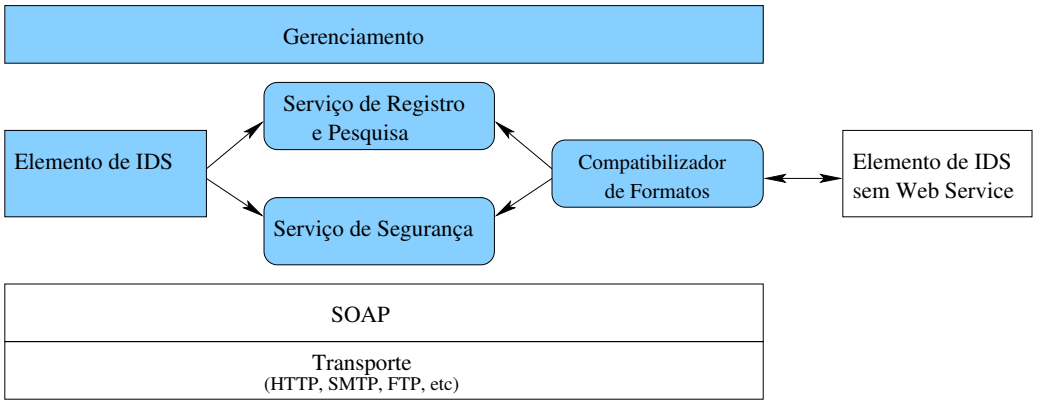

Figura 5.2: *Infra-estrutura para Composição de IDSs*

No modelo, os elementos de detecção de intrusão são apresentados, para composição, como *Web Services*. O gerenciamento, apresentado na Figura 5.2, tem a função de configurar e monitorar os serviços e elementos de IDS da composição. Este gerenciamento é feito através de interfaces em *Web Services* e por esta razão só pode ser feito em elementos encapsulados na forma de *Web Services*.

Através destas interfaces de gerenciamento é possível reconfigurar a composição de IDSs.

Para que uma composição seja criada são necessárias diversas informações sobre os elementos envolvidos, como suas características, localização, protocolos de comunicação e interfaces. Tais informações são apresentadas e acessadas através do Serviço de Registro e Pesquisa (SRP), que junto com o "Compatibilizador de Formatos" e o Serviço de Segurança fazem parte da infra-estrutura de serviços para a composição de IDSs.

5.1.1 Serviço de Registro e Pesquisa

O SRP está fundamentado na especificação UDDI (*Universal Description, Discovery and Integration specification*) (Clement et al., 2004), usando mecanismos para a criação e registro de domínios administrativos com o objetivo de armazenar informações nestes domínios. A especificação define também mecanismos que permitem o uso de diversos servidores UDDI. Estes servidores podem ser replicados, mantendo a mesma informação em mais de um servidor, provendo escalabilidade e disponibilidade necessárias ao serviço. Este SRP utiliza servidores UDDI exclusivamente para a formação de composições de IDSs.

Uma composição de IDSs é efetivada a partir de sua representação que é dada através de seu registro no SRP, que é feito na forma de um *tModel*. As interfaces de cada *Web Service* são descritas em um formato processável, fornecido pela linguagem WSDL (*Web Services Description Language*) (Chinnici et al., 2006) e a localização desta descrição também é mantida no SRP.

5.1.2 Serviço de Segurança

O Serviço de Segurança trata da autenticação e controle de acesso dos elementos da composição. Os mecanismos de segurança disponibilizados na especificação da UDDI são usados para autenticar administradores, operadores e elementos de IDS em suas interações com o SRP, e também provêem confidencialidade e integridade às informações sobre os elementos de IDS contidas no SRP.

As chaves públicas e os contextos de segurança usados nas comunicações entre os elementos da composição seguem a especificação *WS-Security* e são obtidos destes elementos registrados na UDDI através do Serviço de Segurança. Em um registro de elemento, uma estrutura *bindingTemplate* armazena uma chave pública ou indica a sua localização. As chaves pública e privada são representadas seguindo o modelo X.509 (ITU-T, 1993).

5.1.3 Compatibilizador de Formatos

O modelo de composição prevê o uso de diversos IDSs convencionais ou partes dos mesmos, ainda que estes não possuam suporte à tecnologia de *Web Services*. Para definir elementos "serviços" a partir destes elementos de IDS, é necessário introduzir um nível de funções que formam o que chamamos de "Compatibilizador de Formatos", cujo papel principal é tratar com os formatos usados nas trocas de mensagens em composições de serviços. Estas funções, por exemplo, atuam também na intermediação da comunicação entre partes usadas de IDSs convencionais, na forma de *Web Services*, disponibilizando-as para composições de IDSs.

O Compatibilizador de Formatos integra-se aos IDSs convencionais, atuando na comunicação entre estes IDSs e o gerente ou outros elementos de IDS da composição, através de interfaces para Serviços *Web*, que convertem as mensagens geradas por estes IDSs para o formato padrão IDMEF em XML e as disponibiliza para a composição. Abaixo podemos observar um exemplo resumido de uma mensagem de alerta no formato IDMEF em XML, gerada pelo IDS *Prelude* alterado, onde foi detectado um ataque do tipo *Scan* na porta 1900, de baixa severidade, contendo um datagrama *udp* com tamanho de 109 bytes. Nesta detecção foi utilizada uma assinatura do IDS *Snort*, importada pelo IDS *Prelude*.

```
1  <IDMEF-Message><Alert ident="4"><Analyzer analyzerid="1047155569558864375"
2  version="0.8.6-0-8-20050509" class="NIDS" ostype="Linux" osversion="2.6.12-9-386">
3  <Process><name>prelude-nids</name><pid>11177</pid>
4  ...
5  <DetectTime ntpstamp="0xc7cd7fbb.0x66f6700">2006-03-23 17:12:43.402-0300</DetectTime>
6  <address>150.162.xx.xx</address></Address></Node><Service><port>8008</port>
7  <protocol>udp</protocol></Service></Source><Target decoy="unknown">
8  <Classification origin="unknown"><name>SCAN UPnP service discover attempt</name>
9  <Impact severity="low" completion="NULL" type="recon">Detection of a Network Scan</Impact></Assessment>
10 ...
11 <AdditionalData type="string" meaning="Udp header">8008 -&gt; 1900 [len=109]</AdditionalData>
12 <AdditionalData type="string" meaning="Payload header">size=101 bytes</AdditionalData>
13 <AdditionalData type="string" meaning="Payload Hexadecimal Dump">4d 2d 53 45  41 52 43 48
14 20 2a 20 48  54 54 50 2f M-SEARCH * HTTP/
15 ...
16 <AdditionalData type="string" meaning="Detection Plugin Name">SnortRules</AdditionalData>
17 <AdditionalData type="string" meaning="Detection Plugin Contact">prelude-devel@prelude-ids.org</AdditionalData>
18 <AdditionalData type="string" meaning="Detection Plugin Description">Snort signature parser.</AdditionalData>
19 <AdditionalData type="integer" meaning="Snort rule ID">1917</AdditionalData>
20 <AdditionalData type="integer" meaning="Snort rule revision">4</AdditionalData></Alert></IDMEF-Message>
```

Exemplo de Mensagem IDMEF

A mensagem IDMEF apresenta ainda a versão do IDS, o sistema operacional no qual ele está sendo executado, o número do processo, a data e o horário do ataque, o endereço IP, a porta de origem e o protocolo utilizado. Outras informações foram suprimidas deste exemplo a fim de simplificá-lo

e facilitar sua visualização. Estas mensagens IDMEF podem, através do protótipo desenvolvido, ser distribuídas entre os elementos gerentes que fazem parte da composição que por sua vez repassam para os Operadores locais daquele domínio administrativo.

Estas mensagens são disponibilizadas à composição através do Compatibilizador de Formatos, que atua nos dois sentidos: enviando alertas de seus elementos de IDS e recebendo alertas. Para isto, utilizam-se dois módulos, um que recebe as mensagens no formato nativo do IDS, gera uma mensagem no formato IDMEF, encapsula em uma mensagem SOAP e então envia esta mensagem SOAP a um serviço ou elemento da composição. O outro faz o processo inverso, recebendo uma mensagem SOAP, convertendo para o formato nativo utilizado pelo elemento de IDS e enviando ao mesmo. Este Compatibilizador de Formatos não identifica os formatos de forma automática, sendo necessário que cada Compatibilizador de formatos seja previamente configurado.

O Compatibilizador de Formatos faz parte do modelo de comunicação desenvolvido e será tratado com detalhes na seção 5.2. A seguir apresentamos o procedimento geral desenvolvido para a criação e a manutenção de composições de IDSs utilizando orquestração de *Web Services*.

5.1.4 Procedimento Geral para Implantação de Composições

Um procedimento geral para a criação e a manutenção dinâmica de composições de IDSs foi desenvolvido. Este procedimento foi criado para especificar como os elementos de IDS (na forma de *Web Services*) devem se relacionar para estabelecer uma composição.

O procedimento foi desenvolvido de acordo com o conceito de orquestração de *Web Services*, onde esta orquestração trata da interação dos elementos, refletindo a visão do administrador da composição. Escolhemos usar orquestração por causa da sua flexibilidade, necessária no trato de composições dinâmicas. Uma orquestração permite que o administrador da composição defina um único fluxo de processo geral que pode ser usado para formar uma composição e que permanece inalterado quando elementos de IDS são adicionados ou removidos. Este fluxo de processo, que é o procedimento desenvolvido, conta com um conjunto de operações básicas ilustradas na Figura 5.3. Esta figura foi extraída do software (*Weblogic Workshop*) através da interface resultante da implementação deste procedimento.

Em princípio, a localização dos serviços não é conhecida, apenas as funcionalidades desejadas dos mesmos. Na figura, o fluxo indica a ordem em que as operações devem ser efetuadas. A operação *ClientRequestDiscovery* (Figura 5.3a) é responsável por executar a busca por serviços na UDDI utilizando o Serviço de Registro e Pesquisa, de acordo com parâmetros estabelecidos na operação, como

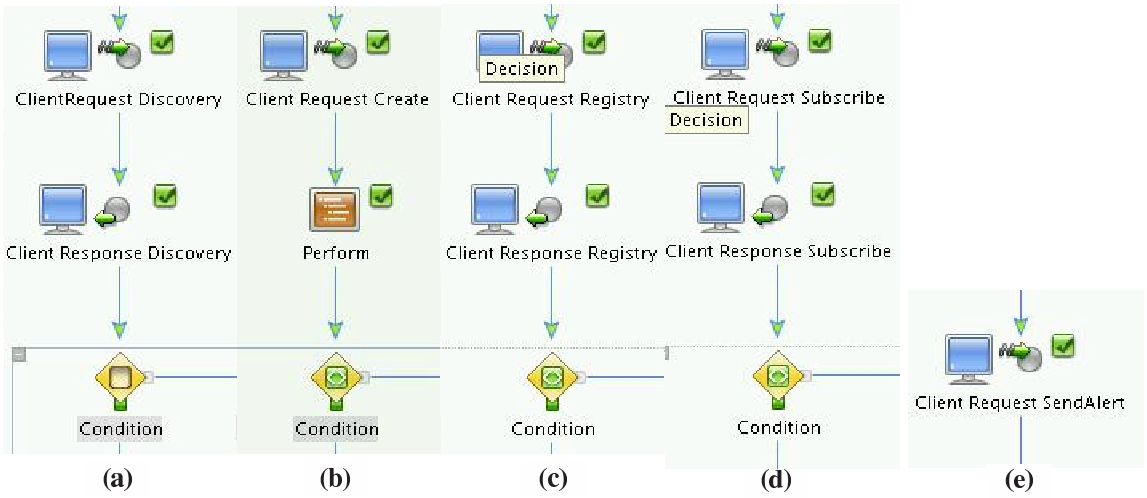

Figura 5.3: *Conjunto de Operações para formar uma Composição*

por exemplo, o endereço de rede em que o elemento está operando.

A operação *ClientRequestCreate* (Figura 5.3b) é responsável pela requisição para a criação do registro da composição. A operação *Perform* faz a verificação dos dados e a inserção do registro da composição na UDDI. As operações *ClientRequestRegistry* e *ClientResponseRegistry* (Figura 5.3c) são responsáveis respectivamente pela requisição e resposta da publicação de um serviço na UDDI. Os registros destes serviços podem ou não ser inseridos na UDDI, seguindo políticas de segurança pré-estabelecidas.

As operações *ClientRequestSubscribe* e *ClientResponseSubscribe* (Figura 5.3d), por sua vez, fazem a subscrição de serviços sensores e analisadores em gerentes. Nestas operações são trocadas as chaves públicas de cada elemento. Por último, a operação *ClientRequestSendAlert* (Figura 5.3e) é utilizada pelos serviços já registrados na composição para enviar alertas à composição. Esta operação faz uso do serviço de comunicação apresentado com mais detalhes na seção 5.2.

A orquestração de composições segue uma seqüência lógica e cronológica dos eventos representados na Figura 5.3. A partir de cada evento é possível continuar ou voltar para o evento anterior do fluxo de execução (representado pelo *Condition*). Para executar a operação *ClientRequestSendAlert*, por exemplo, não é necessário executar os eventos anteriores, desde que eles já tenham sido executados pelo menos uma vez.

Após a criação da orquestração da composição, foram gerados os documentos WSDL e BPEL que representam, respectivamente, as interfaces da composição e a orquestração que contém o seu fluxo de execução. A aplicação dos passos do método descrito na criação das composições apresen-

tadas na Figura 5.3 são apresentados na Tabela 5.1, descrevendo os passos necessários para a criação de qualquer composição de IDSs utilizando a infra-estrutura desenvolvida.

Tabela 5.1: *Descrição dos passos para criar as composições*

Passo	Operação
1	Localizar na UDDI o Serviço Gerenciador, de acordo com os parâmetros de busca especificados utilizando a operação *ClientRequestDiscovery*
2	Executar a operação ClientRequestCreate para criar um registro da composição na UDDI.
3	Inicializar o Serviço Gerenciador (*Invoke*).
4	Associar o Serviço Gerenciador à composição, usando a operação ClientRequestRegistry.
5	Localizar na UDDI o Serviço Analisador, de acordo com os parâmetros de busca especificados utilizando a operação ClientRequestDiscovery.
6	Executar a operação ClientRequestSubscribe no Serviço Analisador para subscrição dos alertas, enviando-os ao Serviço Gerenciador.
7	Associar o Serviço Analisador à composição, usando a operação ClientRequestRegistry.
8	Repetir os passos 5, 6 e 7 para cada um dos demais elementos envolvidos na composição (Sensores 1, 2 e 3).

Na seção seguinte, descrevemos o modelo de comunicação segura desenvolvido para uso em composições de IDSs. Discutimos também o "Compatibilizador de Formatos" e a sua aplicação para prover segurança na comunicação e interoperabilidade necessárias aos elementos da composição de IDSs.

5.2 Modelo para comunicação segura e interoperável

Para prover um serviço de comunicação seguro nas interações de uma composição de IDSs, fazemos uso de um conjunto de serviços que estão fundamentados em padrões e conceitos da tecnologia de *Web Services*. O uso desta tecnologia é viável em ambientes de larga escala e permite que se consiga a interoperabilidade e a integração de IDSs.

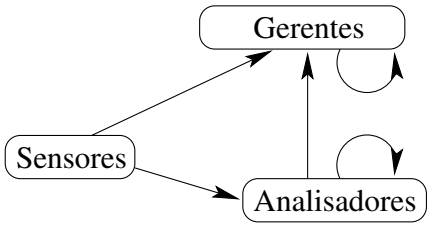

Figura 5.4: *Comunicação entre elementos de IDSs*

Nestes ambientes, a comunicação entre domínios administrativos pode acontecer: entre sensores e analisadores, entre sensores e gerentes, entre analisadores e gerentes, somente entre analisadores

ou somente entre gerentes conforme é apresentado na Figura 5.4. Em todas estas comunicações são adotados mecanismos e padrões de segurança.

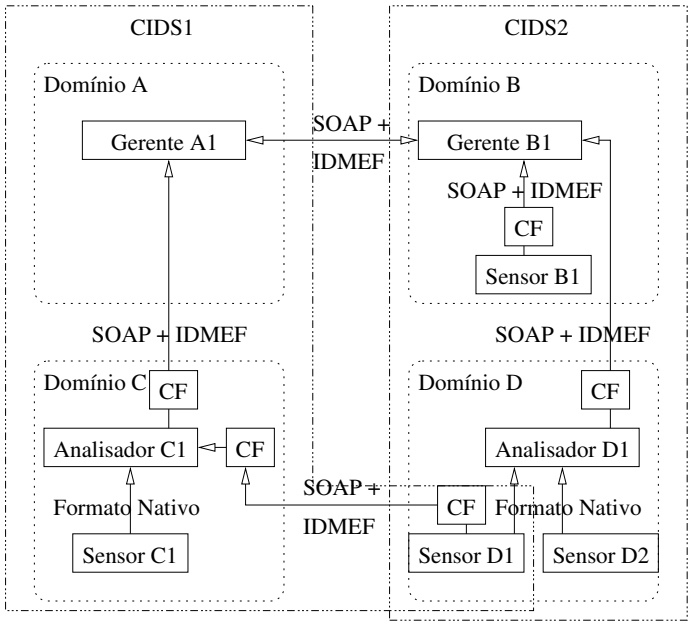

Figura 5.5: *Exemplo de comunicação em composições de IDSs*

A Figura 5.5 apresenta um exemplo de composição de IDSs em ambientes de larga escala. Este exemplo ilustra duas composições de IDSs (*CIDS1* e *CIDS2*) utilizando quatro domínios administrativos distintos (*Domínio A*, *Domínio B*, *Domínio C* e *Domínio D*). A composição *CIDS1* é composta pelo gerente A1 do domínio A, pelo analisador C1 e o sensor C1 do domínio C e pelo sensor D1 do domínio D. Este último sensor também faz parte da composição *CIDS2* que contém ainda: o sensor D2 e o analisador D1 do domínio D, o sensor B1 e o gerente B1 do domínio B. As setas, na figura, indicam o sentido das mensagens. Estas duas composições também trocam entre si mensagens SOAP através de seus gerentes.

No modelo proposto, mensagens que envolvem comunicações entre componentes de domínios diferentes, pertencentes ou não à mesma composição, utilizam-se sempre de alertas no formato IDMEF encapsulados em mensagens SOAP. Estas mensagens são montadas a partir do que chamamos "Compatibilizador de Formatos", representado pela sigla *CF* na figura acima. A comunicação dentro de um domínio pode acontecer de duas maneiras: utilizando o formato nativo[3] ou através do Compatibilizador de Formatos. Na Figura 5.5, por exemplo, o sensor D1 pode comunicar-se com o analisador D1 utilizando o formato nativo e com outro analisador externo ao seu domínio (analisador

[3]Formato próprio, adotado pelo sensor ou analisador de um IDS

C1) fazendo uso do seu *CF*.

Estes "Compatibilizadores de formatos" (CFs) fazem a tradução das mensagens do formato nativo de qualquer sensor ou analisador para o formato IDMEF/XML[4] (Debar et al., 2006) encapsulado em SOAP. O *CF* permite a integração de diversos elementos de IDSs existentes através da tecnologia de *Web Services*, produzindo uma cápsula de *Web Service* para estes elementos. Estes elementos (sensores ou analisadores) ficam disponíveis à composição como Serviços *Web* através de seus *CFs*. Os gerentes das composições são admitidos como sendo Serviços *Web* no modelo proposto e, portanto, "entendem" o formato IDMEF em XML encapsulado em SOAP e não precisam fazer uso de *CFs*. Porém comunicações envolvendo outros componentes de IDSs em outros níveis podem envolver *CFs* nos dois extremos de uma comunicação.

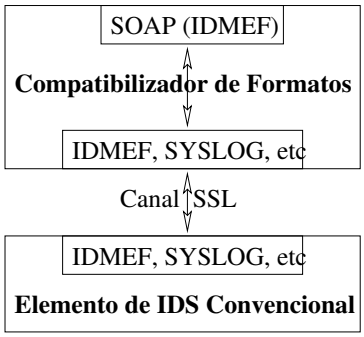

Figura 5.6: *Comunicação entre elemento de IDS e CF*

Na comunicação entre os elementos de detecção de intrusão convencionais e o *CF* podem ser usadas mensagens nos formatos IDMEF, SYSLOG[5] ou outro formato (nativo do elemento) e suportado pelo *CF*, como mostra a Figura 5.6. Como esta comunicação acontecerá na mesma máquina ou na rede local, é utilizado um canal de comunicação seguro SSL (Secure Socket Layer).

Neste modelo proposto, seguimos o padrão *WS-Security* (Nadalin et al., 2004) para assinatura (*XML Signature* (Reagle, 2000; Eastlake et al., 2002)) e cifragem (*XML Encryption* (Reagle, 2002; Imamura et al., 2002)) de mensagens encapsuladas em SOAP. Contudo, é necessário o uso de um repositório de certificados para conter as chaves públicas dos elementos de IDSs. Este repositório, no nosso modelo, localiza-se no Serviço de Registro e Pesquisa (SRP), descrito na seção 5.1.

A Figura 5.7 especifica, com alguns detalhes, o modelo de comunicação proposto para as interações em composições de IDSs. Nesta figura, é ilustrado um domínio A com componentes de

[4]O IDMEF em XML é o formato padrão, segundo o IDWG
[5]http://www.ietf.org/html.charters/syslog-charter.html

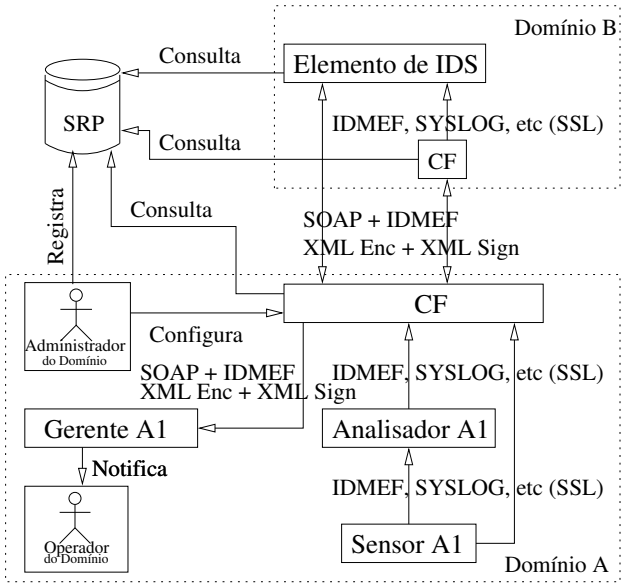

Figura 5.7: *Detalhes do modelo de comunicação segura*

uma composição de IDSs onde um sensor e um analisador fazem uso de um *CF* para se comunicarem com o elemento de IDS no domínio B. A comunicação entre o sensor e o analisador no domínio A pode ser realizada usando o formato nativo ou o IDMEF, desde que seja suportado por ambos sensor e analisador.

Comunicações entre elementos de IDSs, em ambientes locais (mesmo domínio), sem a necessidade de transpor *firewalls* ou outras barreiras, podem ser estabelecidas diretamente (sem o uso do *CF*) desde que estes elementos utilizem o mesmo formato de mensagens e façam uso de canais seguros SSL para garantir a segurança destas mensagens. O custo computacional usando o formato nativo é menor do que o custo computacional usando o *CF*, como pode ser comprovado nos testes realizados (Capítulo 6).

No domínio A da Figura 5.7, aparece o papel do Administrador de Domínio que possui, entre outras, a função de registrar (operação Registra) no SRP, na forma de Serviços *Web*, os seus elementos de detecção de intrusão através de seus respectivos *CFs*. Uma vez registrados, estes elementos e seus *CFs* ficarão disponíveis para o uso em composições de IDSs. O serviço de registro vê estes elementos e seus *CFs* como Serviços *Web*. Outra função do administrador do domínio é configurar (operação Configura) o Compatibilizador de Formatos (*CF*), informando o formato no qual o *CF* deve receber mensagens e a quem (qual SRP) este *CF* deve fazer as consultas para obter o endereço destino das mensagens.

Durante a construção de uma composição é necessário que se estabeleçam conexões entre *CFs* de elementos de IDSs de domínios diferentes. Para isto um *CF* deve fazer consultas ao SRP, para obter os certificados necessários para que se crie uma conexão segura entre os pares comunicantes. No caso da Figura 5.7, o *CF* do domínio A faz consultas (operação Consulta) ao SRP para obter o certificado necessário que permita o estabelecimento de um canal seguro entre *CFs*, permitindo que mensagens SOAP, assinadas e cifradas, sejam enviadas do domínio A para outro domínio (Domínio B, no caso da Figura 5.7).

No exemplo da Figura 5.7, é mostrada a comunicação entre o sensor A1 do domínio A e o *Elemento de IDS* no domínio B que na estrutura de um IDS pode ser um analisador ou um gerente. Se o elemento em B for um gerente, a comunicação do sensor A1 com este elemento deve fazer uso de um *CF* no domínio A. No outro lado da comunicação, no domínio B, se o elemento for um gerente, não será necessária a recuperação de padrões nativos a partir da mensagem SOAP encapsulando o IDMEF. Todo gerente sendo um *Web Service* permite a comunicação direta, recebendo mensagens SOAP, sem a necessidade de um *CF* para a recuperação de padrões nativos. Para se habilitar ao canal seguro, o gerente deverá também consultar ao SRP (operação Consulta) para obter a chave pública do emissor da mensagem e, então, poderá receber, decifrar, verificar a assinatura e processar as mensagens recebidas.

Se o Elemento de IDS da figura for um analisador, um *CF* no domínio B deve fazer a consulta (operação Consulta) ao SRP para obter a chave pública do emissor, decifrar, verificar sua assinatura e transformar mensagens recebidas em IDMEF, SYSLOG ou qualquer outro formato nativo suportado pelo analisador.

Ainda na Figura 5.7 temos o papel do Operador do domínio, que tem a função de receber notificações de gerentes e tomar possíveis ações no sentido, por exemplo, de conter um ataque em andamento. Estas notificações seguem o padrão *publish/subscribe* e foram baseadas nas especificações *Web Services Notification* (WSN) (Graham et al., 2004). Cada Operador dos domínios administrativos pertencentes à composição deve fazer a sua subscrição nos gerentes para poder receber as notificações.

No processo de subscrição, o Operador recebe uma chave, que o identifica e o autoriza perante o sistema de notificações. Após subscrito, o Operador estará habilitado para receber notificações. Se algum gerente publicar alguma mensagem, esta será recebida pelo Operador. A seguir é apresentado um exemplo de mensagem utilizando o servidor de notificações WSN.

Neste exemplo, é apresentado uma mensagem SOAP gerada pelo sistema de notificações WSN. Esta mensagem possui um cabeçalho (*Header*, linhas 1 a 10), que informa qual o modelo de esquema

XML usado e quais os esquemas para composição do envelope SOAP, e um corpo (*Body*, linhas 11 a 24) onde faz-se uso da operação *GetCurrentMessageResponse* (linhas 12 a 23) para obter as mensagens associadas ao tópico *AlertaOperador*, definido para mensagens destinadas aos Operadores. Os campos *OldValue* (linha 16) e *NewValue* (linhas 17 a 21) apresentam as mensagens publicadas recentemente onde, neste exemplo, existe apenas uma mensagem nova (*NewValue*) de teste do sistema.

```
1  <?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope
2  /" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3    <soapenv:Header>
4      <wsa:Action soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next" soapenv:mustUnderstand="0" xmlns:wsa
5    ="http://schemas.xmlsoap.org/ws/2004/03/addressing">http://schemas.xmlsoap.org/ws/2004/03/addressing/anonymous<
6    /wsa:Action>
7      <wsa:To soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next" soapenv:mustUnderstand="0" xmlns:wsa="ht
8    tp://schemas.xmlsoap.org/ws/2004/03/addressing">http://schemas.xmlsoap.org/ws/2004/03/addressing/anonymous</wsa
9    :To>
10   </soapenv:Header>
11   <soapenv:Body>
12     <wsn:GetCurrentMessageResponse xmlns:wsn="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.
13   2-draft-01.xsd">
14       <wsrf:ResourcePropertyValueChangeNotification xmlns:wsrf="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS
15   -ResourceProperties-1.2-draft-01.xsd">
16         <wsrf:OldValue/>
17         <wsrf:NewValue>
18           <fs:AlertaOperador xmlns:fs="http://abc.com/my/wsdl/namespace" xmlns:wsrp="http://docs.oasis-open.org
19   /wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd">Mensagem teste do servidor de notificações WSN</fs:A
20   lertaOperador>
21         </wsrf:NewValue>
22       </wsrf:ResourcePropertyValueChangeNotification>
23     </wsn:GetCurrentMessageResponse>
24   </soapenv:Body>
25 </soapenv:Envelope>
```

Exemplo de Mensagem do Servidor WSN

5.3 Considerações

O padrão IDMEF, embora ainda não seja amplamente usado pelos IDSs existentes, vem sendo abordado com freqüência na literatura. Contudo, a idéia de integrar IDSs como Serviços *Web* nesta proposta é inédita na literatura. O Prelude é o IDS que mais se aproxima do modelo proposto, utiliza o formato IDMEF, faz a integração com outras ferramentas, porém não consegue transpor *firewalls* ou outras barreiras sem uma configuração específica destas ferramentas.

O nosso modelo de comunicação possibilita a formação de composições dinâmicas de sistemas de detecção de intrusão, através do uso da tecnologia de *Web Services* e do uso de ferramentas de detecção de intrusão já existentes, mesmo que estas ferramentas não possuam suporte ao formato IDMEF, formando os chamados elementos de detecção de intrusão do modelo do IETF.

Com o emprego da tecnologia de *Web Services*, em conjunto com a linguagem de programação Java, o modelo desenvolvido pode atuar tanto em pequenos ambientes como em ambientes distribuídos, de larga escala e heterogêneos. A grande vantagem do uso dos padrões *WS-Security* sobre o SSL ou outros métodos para comunicação segura é que com o *WS-Security* é possível fazer a assinatura e cifragem de apenas partes da mensagem, possibilitando o tráfego desta mensagem através de diversos *Web Services* e promovendo a autenticação e a segurança fim a fim destas mensagens. Já com o SSL ou outro método semelhante, é necessário criar um caminho cifrado, ponto a ponto entre o emissor e o receptor da mensagem, sem a presença de nós intermediários. A vantagem do SSL é que, depois de estabelecido o caminho, podem ser enviadas diversas mensagens pelo mesmo, enquanto no *WS-Security* é necessário o estabelecimento de uma nova conexão para cada mensagem a ser enviada.

O *Web Service Notification*, por usar mensagens assíncronas (comunicação desacoplada), fornece uma maneira simples, rápida e confiável para difusão de mensagens de alerta entre os membros da composição. Outros padrões que poderiam ser adotados são o *Keryx Internet Notification System* (Brandt e Kristensen, 1997) ou o *Corba Notification Service* (OMG, 1999), porém ambos não possuem suporte nativo a *Web Services*.

Neste capítulo explicitamos as operações envolvidas no procedimento para criação e manutenção de composições e as características do modelo de comunicação desenvolvido. O modelo de comunicação desenvolvido visa manter a segurança das mensagens, utilizando padrões e garantindo a segurança fim a fim das mesmas, permitindo a interoperabilidade entre os diversos sensores, analisadores e gerentes da composição de IDSs.

O modelo de comunicação desenvolvido possibilita a integração de IDSs, permite a troca de informações de segurança entre organizações e permite a criação de sistemas distribuídos em ambientes de larga escala, o que não é possível utilizando apenas os sistemas de detecção de intrusão nativos.

Existem algumas aplicações e cenários possíveis para o uso do modelo de comunicação desenvolvido. Os cenários possíveis incluem tanto o seu uso em redes locais quanto em ambientes de larga escala. Podemos citar como exemplo o próprio modelo de composição de IDSs que está em desenvolvimento ou um sistema para troca de informações de segurança entre CSIRTs (*Computer Security Incident Response Team*).

O modelo apresentado foi testado na forma de um protótipo que é descrito no capítulo seguinte.

5.4 Conclusões do Capítulo

Este capítulo descreveu uma infra-estrutura de serviços necessários na integração de IDSs em ambientes de larga escala. Como parte de um modelo de composição de IDSs, desenvolveu-se um procedimento geral para a criação e a manutenção dinâmica de composições de IDSs e um modelo de comunicação interoperável e seguro.

A segurança dos IDSs em ambientes abertos é um ponto crítico para qualquer sistema de monitoramento. Para tal, faz-se necessário o uso de mecanismos e padrões que possam garantir as propriedades de segurança das próprias informações trocadas ou manipuladas nestas composições distribuídas de IDSs.

Os sistemas de detecção de intrusão distribuídos existentes, em geral, não se preocupam com a questão de segurança em suas comunicações. Os poucos que utilizam métodos seguros, possuem dificuldades para comunicação em ambientes de larga escala pois os seus modelos de comunicação provêem segurança ponto a ponto e não fim a fim.

A composição de IDS é uma abordagem nova que junto com o modelo de comunicação apresentado, consegue cumprir os principais requisitos desejáveis em sistemas distribuídos, vistos na seção 4.6.1, garantindo a sua execução contínua, sendo tolerante a faltas, resistente a ataques, escalável, permitindo a sua reconfiguração dinâmica e conseguindo adaptar-se às mudanças de comportamento do sistema e dos usuários.

Capítulo 6

Implementação e Resultados Obtidos

Após a definição do modelo de comunicação segura e interoperável entre elementos de IDS, foi desenvolvido um protótipo deste modelo de comunicação e foram realizados testes sobre este protótipo para avaliar os custos envolvidos na comunicação entre estes elementos de detecção de intrusão. Nestes testes fazemos um comparativo entre o modelo de comunicação proposto e o sistema de comunicação nativo.

Neste capítulo descrevemos ainda os cenários onde os testes com o protótipo foram realizados, apresentamos os resultados obtidos com estes testes e fazemos uma análise destes resultados.

6.1 Protótipo

Foi desenvolvido um protótipo seguindo o modelo proposto para a comunicação segura e interoperável que permite a integração de IDSs em ambientes de larga escala. A Figura 6.1 apresenta este protótipo. O modelo está centrado no Compatibilizador de Formatos (*CF*). Neste protótipo foi alterado o código fonte do IDS *Prelude* para integrá-lo aos serviços do *CF*, formando "Serviços-Sensores". O *CF* foi implementado fazendo uso da linguagem de programação Java, onde os métodos para assinatura e cifragem de documentos XML são implementados seguindo os padrões *XML-Signature* e *XML-Encryption*. A implementação de gerentes (WS-Gerente na figura 6.1) neste protótipo segue a arquitetura orientada a serviços, baseada na tecnologia de *Web Services* e na aplicação dos padrões de segurança *WS-Security* (Nadalin et al., 2004).

A integração do *CF* com diversas ferramentas de segurança que não possuem suporte ao IDMEF (*Clamav, Honeyd, ipchains, Netfilter, pf, Nagios, sshd, etc*), foi realizada utilizando parte do

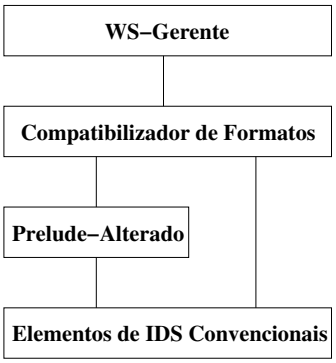

Figura 6.1: *Protótipo desenvolvido*

código fonte do *Prelude* (com algumas alterações). Ou seja, através da biblioteca *libprelude* alterada é feita a conversão destes alertas para o formato IDMEF nativo do *Prelude*. Este formato nativo do *Prelude* é um formato IDMEF que utiliza estruturas de dados próprias para reduzir seu tamanho e facilitar a comunicação entre seus componentes. Foram feitas modificações no código fonte do *Prelude* para que este envie os alertas, através de uma conexão SSL, utilizando o formato IDMEF em XML, padrão do IETF, ao invés do formato IDMEF nativo do *Prelude*. Para as alterações no código fonte do *Prelude* foi utilizada a linguagem de programação C++.

A comunicação entre o *Prelude* e o *CF* (o elemento de IDS e o seu tradutor com a interface para a composição) deve ser realizada em ambiente local, sem a presença de *firewalls* ou outras barreiras no caminho. Neste tipo de ambiente, o uso do SSL supre todas as necessidades de segurança, além de ter um custo computacional baixo e de fácil implementação. Por outro lado, a comunicação entre o *CF* e o gerente (comunicação entre elementos da composição) pode ocorrer em ambientes abertos, sendo necessário o uso de mensagens SOAP com os padrões de segurança *WS-Security*. A seguir apresentamos dois exemplos destas mensagens SOAP, primeiro com a aplicação do *WS-Signature* e em seguida com a aplicação do *WS-Encryption*.

Neste exemplo, o código *SOAP-ENV:Header* (linha 3) representa o início do cabeçalho da mensagem SOAP que se estende até a linha 31. Neste cabeçalho estão presentes os padrões usados para formar a mensagem e seus respectivos valores. Estes padrões devem ser lidos e interpretados pelo receptor desta mensagem. Alguns campos desta mensagem que merecem destaque são: a linha 4 apresenta o início do ambiente de segurança, a linha 6 apresenta o padrão para assinatura adotado, a linha 9 apresenta o algoritmo usado. O campo *SignatureValue* (linha 18) apresenta o *hash* cifrado do documento. Entre as linhas 20 e 28 são apresentadas informações sobre a chave pública usada no processo de assinatura. A linha 30 termina o ambiente de segurança. Entre as linhas 32 e 39 está

o corpo da mensagem, onde foi criado o campo *Alerta* (linhas 34 a 38) para inclusão da mensagem IDMEF em XML, que foi excluída deste exemplo. Um exemplo de mensagem IDMEF em XML é apresentado na seção 5.1.3.

```
1  <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope" xmlns:xsd="http://www.w3.org/2001/X
2  MLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3    <SOAP-ENV:Header>
4      <wsse:Security SOAP-ENV:mustUnderstand="true"
5        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
6        <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
7         <ds:SignedInfo>
8          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
9          <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
10         <ds:Reference URI="#id-27742346">
11       <ds:Transforms>
12         <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
13       </ds:Transforms>
14          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
15          <ds:DigestValue>Hpf4f+VRKWJzSwCGrIKddmUEDW0=</ds:DigestValue>
16      </ds:Reference>
17     </ds:SignedInfo>
18       <ds:SignatureValue>wn9M/8Q+zaUMGoMM5uFPtuQxHisgH+zEXREgzjOx6t9DzaU6MDfNNgkxR/kVCK1PZIfD6KeKKAbfvD8mqVOtEA
19  ==</ds:SignatureValue>
20        <ds:KeyInfo Id="KeyId-28235257">
21         <wsse:SecurityTokenReference wsu:Id="STRId-23667197" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
22  oasis-200401-wss-wssecurity-utility-1.0.xsd">
23       <ds:X509IssuerSerial>
24         <ds:X509IssuerName>CN=dims</ds:X509IssuerName>
25         <ds:X509SerialNumber>443697782562172243709849148479920226l3</ds:X509SerialNumber>
26       </ds:X509IssuerSerial>
27         </wsse:SecurityTokenReference>
28     </ds:KeyInfo>
29    </ds:Signature>
30     </wsse:Security>
31   </SOAP-ENV:Header>
32   <SOAP-ENV:Body wsu:Id="id-27742346" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecu
33  rity-utility-1.0.xsd">
34      <Alerta xmlns="http://www.das.ufsc.br/sic">
35         <IDMEF-Message>
36          ......Mensagem IDMEF em XML......
37         </IDMEF-Message>
38      </Alerta>
39   </SOAP-ENV:Body>
40  </SOAP-ENV:Envelope>
```

Exemplo de Mensagem do SOAP com WS-Signature

O exemplo a seguir apresenta a mesma mensagem SOAP após o processo de cifragem. Neste exemplo, o código do cabeçalho permanece o mesmo (linhas 3 a 31). No corpo da mensagem (linhas 32 a 43), estão presentes o padrão usado para cifragem (linha 34), o algoritmo usado (linha 36) e a respectiva mensagem IDMEF cifrada (linhas 38 a 40). O bloco onde aparece a mensagem cifrada foi suprimido deste exemplo por ser ilegível.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope" xmlns:xsd="http://www.w3.org/2001/X
MLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Header>
    <wsse:Security SOAP-ENV:mustUnderstand="true" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-2004
01-wss-wssecurity-secext-1.0.xsd">
        <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
         <ds:SignedInfo>
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <ds:Reference URI="#id-27742346">
           <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      </ds:Transforms>
           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
           <ds:DigestValue>Hpf4f+VRKWJzSwCGrIKddmUEDW0=</ds:DigestValue>
     </ds:Reference>
    </ds:SignedInfo>
      <ds:SignatureValue>wn9M/8Q+zaUMGoMM5uFPtuQxHisgH+zEXREgzjOx6t9DzaU6MDfNNgkxR/kVCK1PZIfD6KeKKAbfvD8mqVOtEA
==</ds:SignatureValue>
      <ds:KeyInfo Id="KeyId-28235257">
        <wsse:SecurityTokenReference wsu:Id="STRId-23667197" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd">
      <ds:X509IssuerSerial>
        <ds:X509IssuerName>CN=dims</ds:X509IssuerName>
        <ds:X509SerialNumber>443697782562172243709849148479920226l3</ds:X509SerialNumber>
      </ds:X509IssuerSerial>
        </wsse:SecurityTokenReference>
    </ds:KeyInfo>
   </ds:Signature>
    </wsse:Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body wsu:Id="id-27742346" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecu
rity-utility-1.0.xsd">
    <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Content" xmlns:xenc="http://www.w3.org/2001/04/x
mlenc#">
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
      <xenc:CipherData>
        <xenc:CipherValue>
        ......Bloco Cifrado - Mensagem IDMEF em XML cifrada......
        </xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemplo de Mensagem do SOAP com WS-Encryption

Para o desenvolvimento do gerente (WS-Gerente do protótipo) foi utilizada a tecnologia de *Web Services*, fazendo uso do servidor *Web Apache Tomcat* versão 5.5.12 integrado com o servidor de *Web Services* (*Axis*) e com a implementação dos padrões WSN da *Apache* (*Pubscribe*) (Apache, 2005a), além da ferramenta *BEA Weblogic Workshop*[1] versão 8.1 que provê uma plataforma para o desenvolvimento deste gerente.

O *Pubscribe* é uma implementação da família WSN (*Web Service Notification*) de especificações

[1] http://www.bea.com/framework.jsp?CNT=index.htmFP=/content/products/Weblogic/workshop

e foi usado para a implementação do sistema de notificações dos operadores dos domínios. O sistema foi implementado baseado em quatro operações básicas: **inscrever**, **gerar eventos**, **consumir eventos** e **remover registro** descritas a seguir.

- **Inscrever** - Para que um Operador[2] (consumidor) possa utilizar o sistema, ele precisa estar inscrito em um ou mais gerentes (produtor). Esta operação de **inscrever** está baseada no *Subscribe* do WSN. No final desta operação, o consumidor recebe uma chave para ser usada em transações futuras.

- **Gerar Eventos** - O gerente (produtor) gera eventos baseado nas informações de alertas recebidos pela composição de IDSs. Para gerar tais eventos este modelo executa a operação *SetCurrentMessage* do WSN, publicando a mensagem de alerta.

- **Consumir Eventos** - Após a inscrição do Operador (consumidor), este está então habilitado para receber notificações. Se algum gerente (produtor) já publicou alguma mensagem, esta pode ser recebida pelo consumidor. Este processo executa a operação *GetCurrentMessage* para obter a mensagem de alerta, sendo necessário a apresentação da chave de registro do Operador (*ResourceIdentifier*).

- **Remover Registro** - Esta operação é utilizada para remover um Operador da lista de consumidores de um determinado gerente. Para realizar este processo, é executada a operação *Destroy*, sendo necessária a apresentação da chave de registro do Operador (*ResourceIdentifier*).

Tabela 6.1: *Seqüência de ações*

Passo	Tipo de Ação
1	O *CF* recebe alertas do *Prelude* alterado através de uma conexão segura (SSL) em um ambiente local.
2	O *CF* gera uma mensagem SOAP com este alerta.
3	O *CF* realiza uma consulta ao SRP e recebe o certificado X.509 do gerente. Este certificado é armazenado em *cache* pelo *CF* a fim de evitar consultas desnecessárias. Periodicamente o *CF* verifica no SRP se o certificado ainda é válido.
4	A mensagem SOAP é assinada através do *XML-Signature*.
5	A mensagem SOAP é cifrada através do *XML-Encryption*, utilizando o certificado do gerente.
6	A mensagem é então enviada ao gerente (*Web Service*) utilizando o protocolo de transporte *HTTP*.
7	O gerente recebe a mensagem SOAP, realiza uma consulta ao SRP e recebe o certificado X.509 do emissor da mensagem, decifra a mensagem, verifica a assinatura e processa o alerta.

[2]Pessoa responsável por receber os alertas de segurança em determinada organização e tomar as devidas providências

Para implementar o Serviço de Registro e Pesquisa (SRP) utilizamos a ferramenta *BEA WebLogic Server UDDI Registry*[3] que contém a implementação da UDDI versão 3. O repositório de chaves públicas dos elementos de IDS fica armazenado neste SRP.

A Tabela 6.1 mostra a seqüência de ações executadas para a comunicação segura e interoperável entre um elemento de IDS (sensor ou analisador) e um serviço *Web* (gerente). A Figura 6.2 ilustra estas trocas para o estabelecimento do canal seguro, envio da mensagem e recuperação do conteúdo pelo receptor. O elemento central desta comunicação é o *CF* que envia mensagens SOAP/IDMEF para o gerente.

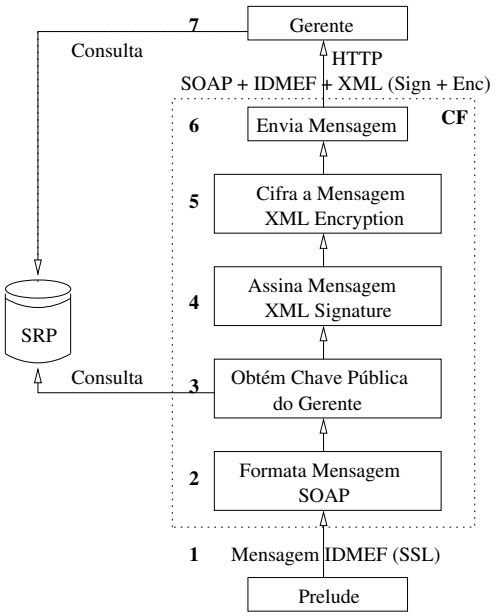

Figura 6.2: *Seqüência de eventos*

Visando verificar a eficiência e os custos computacionais do modelo de comunicação interoperável e segura, foram realizados testes comparativos entre o modelo proposto, o modelo de comunicação utilizando IDMEF em XML e o modelo de comunicação nativo do *Prelude*. As próximas seções apresentam estes testes e os resultados obtidos.

6.2 Testes em um Ambiente Local

A Figura 6.3 mostra a disposição das máquinas para os testes comparativos realizados, utilizando o protótipo descrito na seção 6.1. No ambiente proposto para os testes foram caracterizados

[3]http://www.bea.com/framework.jsp?CNT=index.htmFP=/content/products/server

dois domínios administrativos através de duas sub-redes (rede 1 e rede 2, na Figura 6.3). Para efetuar os ataques, utilizou-se a ferramenta *Idswakeup*[4], que utiliza a base de assinaturas do *Snort* para gerar ataques que serão identificados pelo IDS.

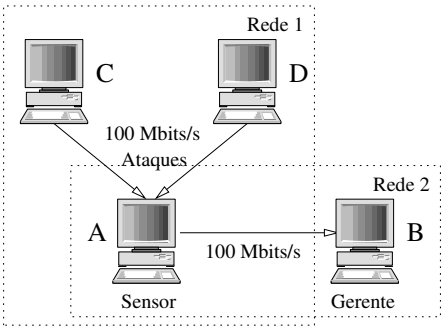

Figura 6.3: *Disposição das máquinas*

Na concretização do ambiente de testes foram ainda usados quatro computadores modelo AMD Athlon XP 2600+ contendo 512MB de memória RAM e placas de rede de 100Mbits/s. Foi instalado o sistema operacional Ubuntu Gnu/Linux versão 5.10.

Cenários de Testes

Na avaliação dos resultados são descritos três cenários de testes que permitiram obter as comparações desejadas. Estes cenários não refletem um ambiente realmente de larga escala por dois motivos:

1. O sistema de comunicação nativo do IDS (cenário 1), por fazer uso do SSL, tem dificuldades para comunicação em tais ambientes, sendo necessário a liberação de portas específicas em *firewalls*, dificultando a realização dos testes em ambientes de larga escala;

2. Os resultados dos testes (tempos de transmissão, por exemplo), com o modelo proposto, seriam prejudicados por causa da alta latência das redes nestes ambientes. Por exemplo, em determinado momento poderiam ser transmitidos os dados em um tempo x e em seguida, a mesma quantidade de dados em tempo $3x$, impossibilitando a comparação destes tempos para análise de custos reais.

Contudo, os testes realizados são válidos para avaliar os custos de comunicação entre as tecnologias envolvidas em um ambiente considerado "ideal", desconsiderando a latência existente em ambientes de larga escala.

[4]http://www.hsc.fr/ressources/outils/idswakeup

- **Cenário 1** (testes com o sistema de comunicação nativo): Um computador A executa o sensor de rede do *Prelude* que está conectado nas duas redes (1 e 2). Um outro computador B presente na rede 2 executa o gerente. Este gerente receberá os alertas gerados pelo sensor através de um canal SSL. Outros dois computadores C e D disponíveis na rede 1 executam ataques ao computador A.

- **Cenário 2** (testes com o sistema de comunicação com IDMEF em XML): O computador A executa o sensor de rede alterado do *Prelude*. O computador B na rede 2 executa o *CF*. Os alertas são gerados pelo sensor do computador A, convertidos para o formato IDMEF em XML e transmitidos ao *CF* através de um canal de comunicação seguro criado no SSL.

- **Cenário 3** (testes com o sistema de comunicação segura e interoperável): Nos testes com o sistema seguindo o modelo proposto, o computador A executa o sensor de rede do *Prelude* e o respectivo *CF*. O computador B na rede 2 executa o gerente. Os alertas são gerados pelo sensor do computador A, transmitidos ao *CF* através de um canal de comunicação seguro e enviados ao gerente através de mensagens SOAP protegidas segundo os padrões *WS-Security*.

A seção seguinte apresenta os resultados dos testes aplicados ao modelo e faz uma análise destes resultados.

6.2.1 Resultados

Os objetivos dos testes foram determinar o tamanho médio das mensagens geradas por cada sistema de comunicação, testar o tempo e a capacidade de transmissão de alertas nos três cenários descritos acima além de verificar, em diferentes situações de ataque, os tempos para processar e gerar alertas nos três formatos.

Os testes foram realizados variando o número de ataques efetuados ao computador A. Foi utilizado o software *Tcpdump* (Jacobson et al., 2005) para coleta de dados de entrada e saída das interfaces de rede das máquinas. Através da análise destes dados, foram obtidos o tamanho médio das mensagens[5], o tempo médio de transmissão das mensagens e o tempo médio de processamento dos alertas[6].

A Figura 6.4 mostra um gráfico com o tamanho médio das mensagens utilizando os três formatos para a comunicação com o gerente: IDMEF nativo com SSL (*IDMEF Nativo + SSL*), IDMEF

[5]As mensagens seguem os três formatos introduzidos: IDMEF nativo com SSL, IDMEF em XML com SSL e IDMEF em XML com SOAP

[6]Tempo entre o início do ataque e o instante em que o alerta é transmitido para o gerente

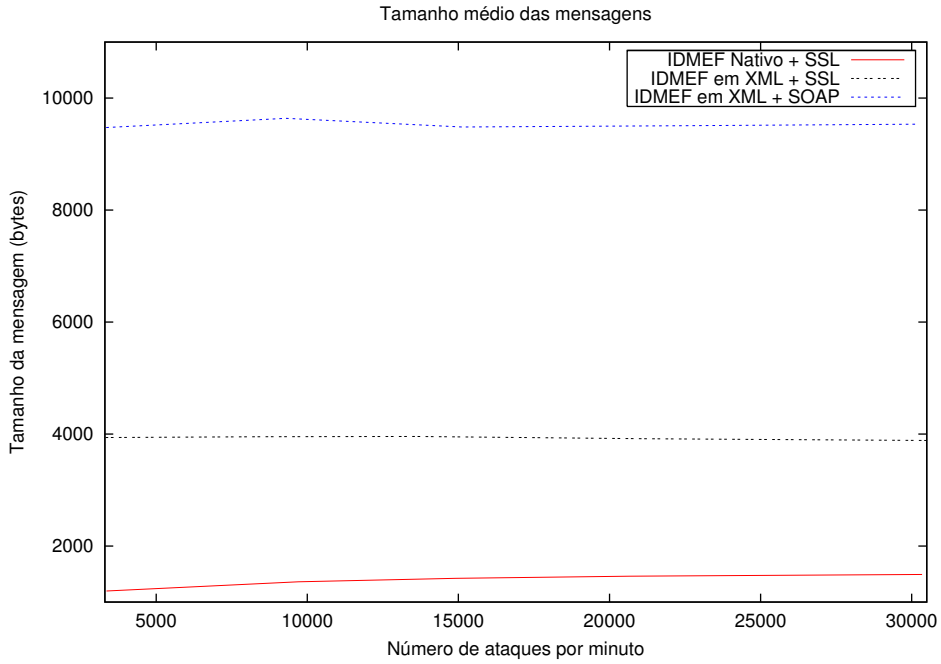

Figura 6.4: *Tamanho médio das mensagens*

em XML com SSL (*IDMEF em XML + SSL*) e IDMEF em XML com SOAP (*IDMEF em XML + SOAP*). Neste gráfico observou-se que o tamanho das mensagens de alerta permanece relativamente constante em 9500 bytes para mensagens no formato IDMEF em XML com SOAP, 4000 bytes para mensagens no formato IDMEF em XML com SSL e 1500 bytes para mensagens no formato IDMEF nativo com SSL.

Em outra análise observou-se o tempo de transmissão entre o computador A (Sensor) sob ataque e o computador B (Gerenciador) que recebe os alertas correspondentes. Como esses dois computadores foram interligados por uma rede independente (rede 2), esta comunicação não sofreu interferências e pode-se observar que o tempo de transmissão permaneceu constante. O *link* de 100Mbits/seg não ficou saturado mesmo quando o sensor sofreu muitos ataques. A Figura 6.5 ilustra o gráfico com os tempos de transmissão.

Em uma terceira análise, observou-se o tempo de processamento e resposta do sistema, ou seja, o tempo entre o início do ataque e o instante em que a mensagem de alerta é enviada. A Figura 6.6 mostra o gráfico desta análise. Este tempo de processamento depende do número de ataques que chegam e são detectados e do poder de processamento do sistema. Observou-se que o tempo é maior com poucos ataques acontecendo ou quando o sistema está saturado (recebe mais do que sua capacidade de processamento), ou seja, se para um caso A o sistema receber x ataques em y tempo

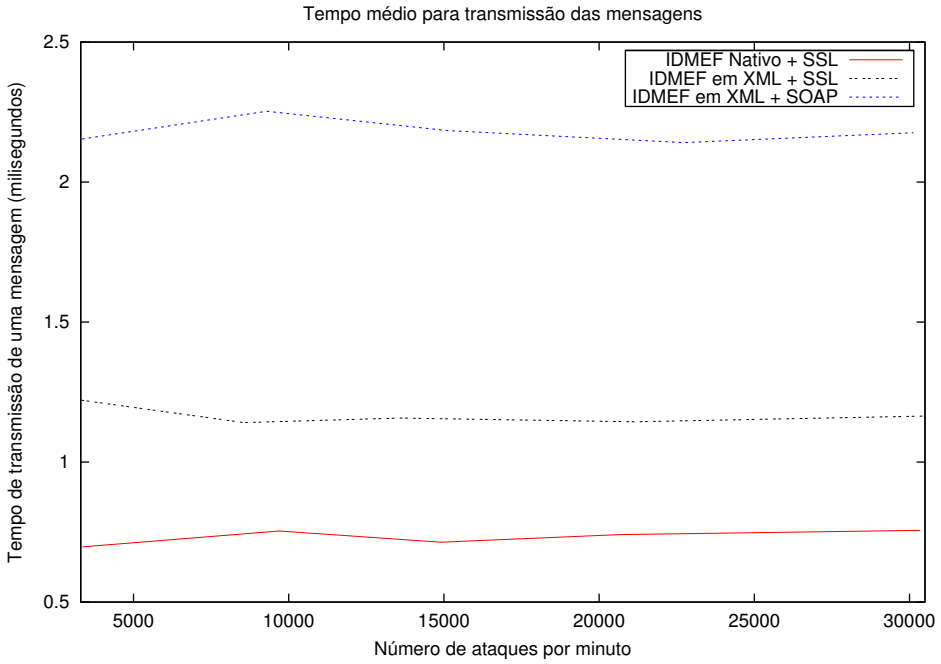

Figura 6.5: *Tempo médio para transmissão de mensagens*

e no caso B o sistema receber $2x$ ataques em y tempo e conseguir processá-los sem atingir o ponto de saturação, o tempo médio para processar um alerta no caso B será menor. No caso do sistema desenvolvido que utiliza o padrão *WS-Security* para atender os aspectos de segurança em ambientes abertos, o ponto de saturação ficou em 15000 ataques por minuto. A partir deste valor, o número de ataques foi maior do que este sistema pode processar. Ao chegar em 30000 ataques por minuto este sistema entrou em colapso e parou.

A performance do sistema utilizando o protocolo nativo ou o IDMEF em XML foi melhor. Mesmo com 30000 ataques por minuto o sistema não chegou a uma saturação. Uma observação interessante é que o sistema nativo faz *cache* em disco, adiando o seu ponto de saturação. Uma das razões para esta performance inferior do protótipo desenvolvido está no uso do protocolo HTTP. Para cada mensagem SOAP a ser enviada é estabelecida uma nova conexão, enquanto no sistema nativo com o uso do SSL, uma única conexão é usada para o transporte de todas as mensagens entre os pares comunicantes.

O tamanho das mensagens SOAP é maior e os custos das conversões de formatos também se fazem sentir. Primeiro o alerta é convertido do IDMEF binário usado pelo *Prelude* para o formato padronizado IDMEF em XML. Depois esta informação passa pelo encapsulamento da mensagem SOAP, com a inclusão da respectiva assinatura. Com isto, o tempo para transmissão de uma mensa-

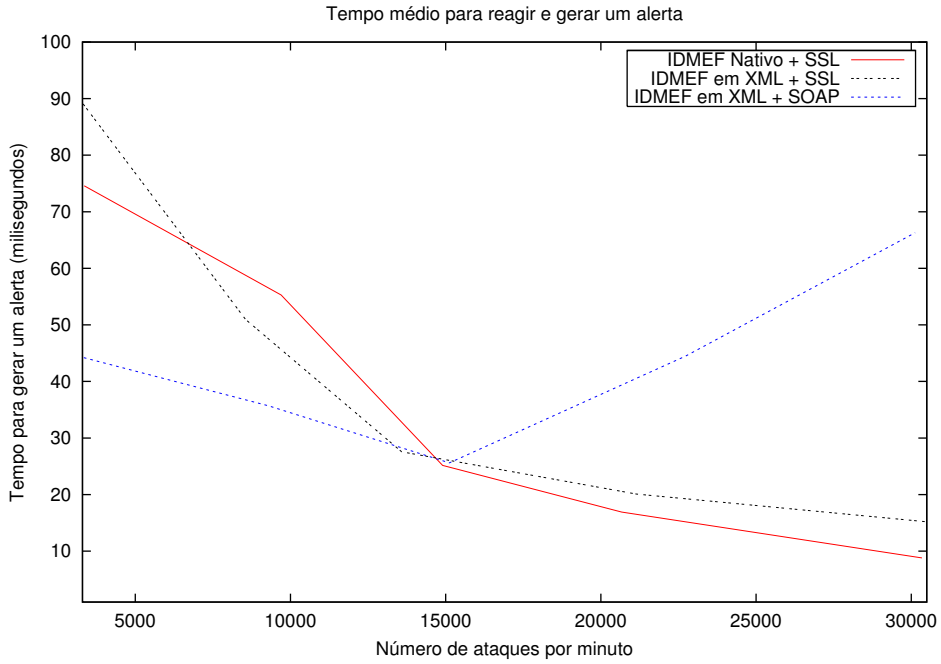

Figura 6.6: *Tempo médio para gerar um alerta*

gem utilizando o modelo proposto é maior pois, além da mensagem ser maior, existe a necessidade de uma nova conexão TCP para cada mensagem de alerta a ser transmitida.

O sistema proposto satura mais rápido pois o processamento para converter as mensagens, assiná-las e cifrá-las é maior. Contudo, o uso de *Web Services*, IDMEF e padrões de segurança permite a integração de diversos sistemas de segurança em redes distintas, além da transposição de *firewalls* e barreiras que existem entre domínios administrativos diferentes.

6.3 Testes em um Ambiente de Larga Escala

Foram aplicados testes com o protótipo desenvolvido em um ambiente de larga escala, visando comprovar o funcionamento do modelo nestes ambientes. Nestes testes foram utilizadas as redes do DAS (Departamento de Automação e Sistemas - UFSC) e do IPEA (Instituto de Pesquisa Econômica Aplicada - DF). A figura 6.7 apresenta a disposição das máquinas nestas redes.

Nestes testes, foram utilizados um computador A (sensor), um computador C (gerente) e dois outros computadores B e D que geram ataques ao computador A. Este computador A (sensor) recebe ataques dos computadores B, localizado na rede do IPEA e D, localizado na rede do DAS. O compu-

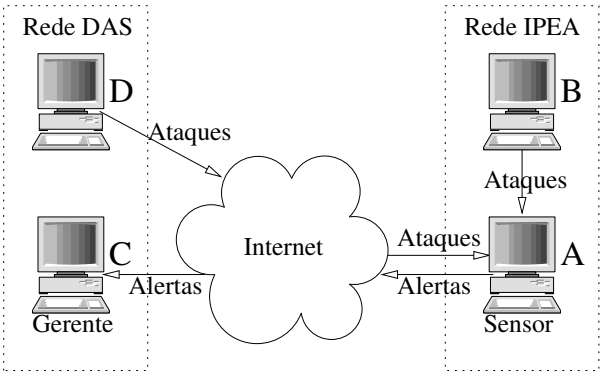

Figura 6.7: *Disposição das máquinas*

tador C (gerente), localizado na rede do DAS, recebe e processa os alertas gerados pelo computador A.

O sensor A recebeu, identificou e gerou alertas de ataques originados em ambas as máquinas B e D. O tempo médio de transmissão das mensagens de alerta foi de 4,403 milisegundos. Fazendo uma comparação com os resultados obtidos anteriormente, observamos um acréscimo de 100% no tempo de transmissão de uma mensagem, o que é bastante razoável para um ambiente como a Internet. Os outros quesitos analisados (tamanho médio das mensagens e tempo médio para gerar um alerta) não apresentaram variações quando comparados aos testes anteriores pois não dependem do ambiente de comunicação. O resultado destes testes comprovaram o funcionamento deste modelo em ambientes distribuídos e de larga escala, fazendo uso da Internet. Os outros modelos analisados previamente não conseguem se comunicar nestes ambientes e portanto não foram testados.

6.4 Considerações

Fazendo uma comparação com a literatura relacionada, a proposição preenche lacunas encontradas nos modelos de IDS analisados na seção 4.6.2. Os sistemas são monolíticos, ou restritos a um ambiente único de rede. O modelo desenvolvido consegue formar IDSs mais apropriados para atuar com os novos conceitos que estão surgindo para aplicações na Internet. Ou seja, o nosso IDS cumpre os requisitos básicos para funcionar em organizações virtuais.

Na comunicação, o modelo é altamente interoperável com outros modelos de comunicação usados por diferentes IDSs. Cabe ainda destacar a flexíbilidade do modelo proposto com relação à segurança, pois este permite que um ou vários receptores autorizados leiam uma mesma mensagem,

mesmo que esta mensagem trafegue por nós terceiros.

O protótipo desenvolvido possui algumas limitações, como o processamento mais lento das mensagens de alerta do que o modelo nativo com SSL e a sua saturação é mais rápida. A linguagem de programação Java, usada na implementação do protótipo, também apresentou alguns problemas relacionados a estouro de memória quando o sistema está sob intenso ataque. Esse estouro de memória ocorre porque o sistema não consegue processar um volume tão grande de mensagens e acaba armazenando essas informações em memória, até que ela se esgote, causando uma excessão.

6.5 Conclusões do Capítulo

Neste capítulo descrevemos um protótipo do modelo de comunicação proposto, apresentamos cenários para execução de testes visando medir os custos na comunicação das partes envolvidas e discutimos os resultados destes testes.

A tecnologia de *Web Services* e os protocolos padrões para troca de mensagens e segurança torna possível a integração de diferentes aplicações de segurança em ambientes de larga escala formando composições.

O desenvolvimento, pelo IETF, do formato para mensagens de alertas IDMEF é de fundamental importância para este trabalho pois possibilitou a integração de diversos sistemas de detecção de intrusão e, em breve, deve ser adotado pelos principais IDSs existentes no mercado, assim como já fizeram o *Snort* e o *Prelude*.

Os custos para o estabelecimento de comunicações seguras em ambientes de larga escala mostraram-se altos, porém a integração de diversos sistemas de detecção possibilita a difusão de alertas de maneira rápida e segura e permite a correlação de eventos originados em redes distintas, possibilitando uma redução do número de falsos positivos desta composição de sistemas.

Capítulo 7

Conclusões e Trabalhos Futuros

Sistemas de detecção de intrusão desempenham um papel fundamental no arcabouço de programas usados para garantir a segurança em sistemas computacionais. O uso crescente de arquiteturas distribuídas baseadas em redes de larga escala, como a Internet, aliado ao aumento do número de incidentes de segurança reportados nos últimos anos, aumenta a necessidade do uso de sistemas de detecção de intrusão distribuídos.

Esta dissertação fez uma descrição dos sistemas de detecção de intrusão presentes na literatura, mostrando as principais características e fazendo uma análise comparativa destes sistemas. Foram mostradas diferentes abordagens para tratar da coleta e análise de dados, além das diferentes abordagens para comunicação destes sistemas em ambientes de larga escala.

Este trabalho mostrou os esforços no desenvolvimento de um formato padrão para mensagens de alertas, bem como o desenvolvimento de um modelo de comunicação segura e interoperável entre diversos sistemas de detecção de intrusão existentes, valendo-se de padrões como o IDMEF e a tecnologia de *Web Services* para formar uma composição destes sistemas. O protótipo implementado enfatiza a viabilidade do modelo proposto e mostra os custos de comunicação deste modelo.

As principais contribuições deste trabalho foram: o desenvolvimento de um procedimento geral para a implantação de composições de IDSs, utilizando partes de IDSs já existentes (ou IDSs completos), através da orquestração de *Web Services* e a criação de um modelo de comunicação segura e interoperável que integra estes diversos elementos de IDS.

Comparando o modelo desenvolvido com o Prelude, que mais se aproxima desse modelo, podemos destacar que o modelo desenvolvido consegue transpor barreiras e funcionar em ambientes

de larga escala como a Internet. O Prelude, por usar SSL, é ideal para uso em redes locais, que não possuem mecanismos de bloqueio da comunicação.

Este trabalho é parte de um projeto que visa a formação de uma composição de IDSs em ambientes abertos. Existem diversas possibilidades da continuidade deste trabalho, onde a mais visível e imediata é o aprimoramento do protótipo, com o desenvolvimento do serviço de registro e pesquisa utilizando mecanismos que garantam a segurança nas operações deste serviço e a inclusão de melhorias no modelo geral para criação e manutenção de composições, como por exemplo, a replicação deste serviço.

Uma outra possibilidade é fazer um estudo do uso de outros protocolos de transporte de mensagens SOAP em *Web Services*, verificando a possibilidade de melhorar a questão de desempenho sem prejudicar a interoperabilidade dos sistemas. Em (Pereira et al., 2005) é apresentada uma comparação entre o uso do SNMP e o SMTP com SOAP para a notificação de eventos de gerenciamento.

Este trabalho que foi desenvolvido no contexto de uma dissertação, procurou responder a questão sobre a viabilidade da interoperação entre IDSs e elementos de IDSs em ambientes de larga escala. Esperamos que os esforços apresentados de alguma forma contribuam no sentido de garantir e permitir estas interoperabilidades.

Referências Bibliográficas

Anderson, J. P. (1972). Computer Security Technology Planning Study. Em *Technical Report ESC-TR-73-51*. URL http://seclab.cs.ucdavis.edu/projects/history/CD/ande72b.pdf.

Anderson, J. P. (1980). Computer Security Threat Monitoring and Surveillance. Em *Technical Report*. URL http://seclab.cs.ucdavis.edu/projects/history/CD/ande80.pdf.

Anderson, S., Bohren, J., Boubez, T., Chanliau, M., e Della-Libera, G. (2005). Web Services Trust Language (WS-Trust). URL http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf.

Apache (2005a). Apache Webservices - pubscribe. URL http://ws.apache.org/pubscribe.

Apache (2005b). Pubscribe developer guide. URL http://ws.apache.org/pubscribe/dev_guide/index.html.

Arkin, A. (2002). Web Service Choreography Interface 1.0. URL http://www.sun.com/software/xml/developers/wsci/wsci-spec-10.pdf.

Axelsson, S. (2000). Intrusion Detection Systems: A Survey and Taxonomy. Relatório Técnico 99-15, Chalmers Univ. URL http://citeseer.nj.nec.com/axelsson00intrusion.html.

Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., e Halam-Baker, P. (2004). Web Service Policy Framework. URL http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf.

Balasubramaniyan, J. S., Fernandes, J. G., Isacoff, D., Spafford, E., e Zamboni, D. (1998). An Architeture for Intrusion Detection using Autonomous Agents. Em *14th IEEE Computer Security Applications Conference*, páginas 13–24.

Bell, E. D. e LaPadula, J. L. (1976). Secure Computer Systems: Unified Exposition and Multics Interpretation. Relatório técnico, MITRE Corporation, Bedford, MA.

Berthold, O., Pfitzman, A., e Standtke, R. (2000). The disvantages of free MIX routes and how to overcome them. Em *Proceedings of Workshop on Design Issues in Anonymity and Unobservability*, páginas 29–44.

Bishop, M. (2003). *Computer Security: Art and Science*. Pearson Education, Boston, MA.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., e Orchard, D. (2004). Web Services Arquiteture. URL `http://www.w3.org/TR/ws-arch/`.

Brand, S. (1985). *Trusted Computer System Evaluation Criteria DoD 5200.28-STD*. US Department of Defense. URL `http://seclab.cs.ucdavis.edu/projects/history/CD/dod85.pdf`.

Brandt, S. e Kristensen, A. (1997). Web push as an internet notification service. Em *Proceedings of the W3C Workshop on Push Technology*, Boston, MA.

Brandão, J. E., da Silva Fraga, J., e Mafra, P. M. (2005). Composição de IDSs usando web services. Em *Proceedings of the 2005 Simpósio Brasileiro em Segurança da Informação e de Sistemas (SBSeg)*, páginas 339–342.

Brandão, J. E., da Silva Fraga, J., e Mafra, P. M. (2006). A New Approach for IDS Composition. Em *Proceedings of the 2006 IEEE International Conference on Communications (ICC), Istambul. NO PRELO*.

Bray, T., Paoli, J., Sperberg-McQueen, C., e Maler, E. (2000). eXtensible Markup Language (XML). URL `http://www.w3.org/TR/REC-xml`.

Cantor, S., Kemp, J., Philpott, R., e Maler, E. (2005). Assertions and Protocols for the OASIS Security Assertion Markup Language 2.0. URL `http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip`.

Champion, M., Ferris, C., Newcomer, E., e Orchard, D. (2004). Web Service Arquiteture. URL `http://www.w3.org/TR/ws-arch/`.

Chinnici, R., Moreau, J.-J., Ryman, A., e Weerawarana, S. (2006). Web Services Description Language (WSDL) Version 2.0 part 1: Core Language. Relatório técnico, W3C. URL `http://www.w3.org/TR/wsdl20/`.

Clark, A. e Limited, S. (1996). Cryptographic controls the eternal triangle. Em *Proceedings of the COMPSEC World Conference on Computer Security*.

Clement, L., Hately, A., von Riegen, C., e Rogers, T. (2004). UDDI version 3.0.2. OASIS UDDI Spec Technical Committee Draft. URL `http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf`.

Debar, H., Curry, D., e Feinstein, B. (2006). The Intrusion Detection Message Exchange Format. technical report draft-ietf-idwg-idmef-xml-16. URL `http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-16.txt`.

Denning, D. (1986). An Intrusion Detection Model. Em *Proceedings of the 1986 IEEE Symposium on Security and Privacy*.

Eastlake, D., Reagle, J., e Solo, D. (2002). XML-Signature Syntax and Processing. request for comments 3275. Relatório técnico, National Institute of Standards and Technology.

Ferraiolo, D. e Kuhn, R. (1992). Role-based Access Controls. Em *NIST-NCSC National Computer Security Conference*, páginas 554–563.

Ferraiolo, D. F., Gilberto, D. M., e Lynch, N. (1993). An Examination of Federal and Commercial Access Control Policy Needs. Em *NIST-NCSC National Computer Security Conference*, páginas 107–116.

Ford, W., Hallam-Baker, P., Fox, B., Dillaway, B., LaMacchia, B., Epstein, J., e Lapp, J. (2001). XML Key Management Specification (XKMS). URL `http://www.w3.org/TR/xkms/`.

Fraga, J. e Powell, D. (1985). A Fault and Intrusion-Tolerant File System. Em *Proceedings of IFIP 3rd Int. Conf. on Computer Security, Dublin Ireland*, páginas 203–218. Elsevier Science Publishers.

Freund, T. e Storey, T. (2002). Transactions in the World of Web Services. URL `http://www-128.ibm.com/developerworks/webservices/library/ws-wstx2/`.

Garfinkel, S., Spafford, G., e Schwartz, A. (2003). *Practical Unix and Internet Security*. O'Reilly.

Godik, S. e Moses, T. (2003). eXtensible Access Control Markup Language 1.0. URL `http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.%pdf`.

Gollman, D. (1999). *Computer Security*. Wiley.

Graham, S., Niblet, P., Chappell, D., Lewis, A., Nagaratnam, N., Parikh, J., Patil, S., Samdarshi, S., Tuecke, S., Vambenepe, W., e Weihl, B. (2004). Web Services Notification. URL `http://www-128.ibm.com/developerworks/library/specification/ws-notifica%tion`.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., e Nielsen, H. F. (2003). Simple Object Access Protocol (SOAP). URL `http://www.w3.org/TR/soap/`.

Guttman, B. (1995). *An Introdution to Computer Security: The NIST Handbook*. U.S. Dept. of Commerce, Technology Administration, National Institute of Standards and Technology.

Heberlein, T., Dias, G., Levitt, K., Mukherjee, B., Wood, J., e Wolber, D. (1990). A Network Security Monitor. Em *Proceedings of IEEE Symposium on Security and Privacy, Oakland*, páginas 296–304. IEEE Computer Society Press.

Hochberg, J., Jackson, K., Stallings, C., McClary, J., DuBois, D., e Ford, J. (1993). NADIR, an automated system for detecting network intrusion and misuse. Em *8th National Information Systems Security Conference*. Computer Security 12(3):235-248.

Imamura, T., Dillaway, B., e Simon, E. (2002). Xml Encryption Syntax and Processing. Relatório técnico, Department of the Navy XML Registry.

ITU-T (1993). ITU-T recommendation X.509.

Jacobson, V., Leres, C., e McCanne, S. (1994). Libpcap. URL `http://www-nrg.ee.lbl.gov`.

Jacobson, V., Leres, C., e McCanne, S. (2005). Tcpdump. URL `http://www.tcpdump.org/`.

Janakiraman, R., Waldvogel, M., e Zhang, Q. (2003). Indra: A Peer-to-Peer Approach to Network Intrusion Detection and Prevention. Em *Proceedings of IEEE WETICE 2003*. URL `http://citeseer.ist.psu.edu/article/janakiraman03indra.html`.

Jansen, W., Mell, P., Karygiannis, T., e Marks, D. (1999). Applying Mobile Agents to Intrusion Detection and Response. Relatório Técnico 6416, National Institute of Standards and Technology. URL `http://citeseer.ist.psu.edu/jansen99applying.html`.

Lampson, B. (1974). Protection. Em *Proceedings of 5th Princeton Symp. on Information Science and Systems*, páginas 437–443. ACM.

Landwehr, C. E. (1981). Formal Models for Computer Security. Em *ACM Comput. Surv.*, páginas 247–278.

Landwehr, C. E. (1983). Best available technologies for computer security. Em *Symposium on Security and Privacy*, páginas 34–40. IEEE Computer.

Landwehr, C. E. (1984). Hardware requirements for secure computer systems. Em *Proceedings of IEEE Symposium on Security and Privacy*, páginas 34–40.

Landwehr, C. E. (2001). Computer Security. *International Journal of Information Security*.

Lee, W., Stolfo, S., e Mok, K. (1999). A data mining framework dor building intrusion detection models. Em *Proceedings of the 1999 IEEE Symposim on Security and Privacy, IEEE Press, Oakland*, páginas 120–132.

Leymann, F. (2001). Web Service Flow Language (WSFL 1.0). URL `http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`.

Leymann, F. e Roller, D. (2002). Business Processes in a Web Services World. URL `http://www-128.ibm.com/developerworks/library/ws-bpelwp`.

McHugh, J. (2001). Intrusion and Intrusion Detection. Relatório técnico, Carnegie Mellon University, Pittsburg, PA.

Myers, P. (1980). The neglected aspect of computer security. Dissertação de Mestrado, Naval Postgraduate School, Monterey. URL `http://seclab.cs.ucdavis.edu/projects/history/CD/myer80.pdf`.

Nadalin, A., Kaler, C., Hallam-Baker, P., e Monzillo, R. (2004). Web Services Security: SOAP Message Security 1.0. URL `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-se%curity-1.0.pdf`.

OMG, O. M. G. (1999). Notification service specification. Relatório Técnico Technical Report telecom/99-07-01, Object Management Group, Famingham, MA.

O'Neill, M. (2003). *Web Services Security*. McGraw Hill.

Peltz, C. (2003). Web Services Orchestration. Relatório técnico, Hewlett Packard Company.

Pereira, E. D., Granville, L. Z., Almeida, M. J., e Tarouco, L. M. (2005). Avaliação de Suporte de Notificações utilizando SNMP e Web Services em uma Arquitetura de Correlação de Eventos Distribuída. Em *Anais do 23o Simpósio Brasileiro de Redes de Computedores (SBRC)*, páginas 1–14. SBC.

Porras, P. A. e Neumann, P. G. (1997). EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. Em *1997 National Information Systems Security Conference*. URL `http://www.sdl.sri.com/papers/emerald-niss97/`.

Reagle, J. (2000). XML Signature requirements. request for comments 2807. Relatório técnico, Massachusetts Institute of Technology Laboratory for Computer Science.

Reagle, J. (2002). XML Encryption requirements. note 04. Relatório técnico, Massachusetts Institute of Technology Laboratory for Computer Science.

Roesch, M. (1999). Snort - Lightweight Intrusion Detection for Networks. Em *Proceedings of USENIX LISA, Berkeley*, páginas 229–238.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., e Youman, C. E. (1996). Role-based Access Control Models. Em *IEEE Computer*, páginas 38–47.

Sandhu, R. S. e Samarati, P. (1996). Authentication, Access Control and Intrusion Detection. Em *ACM Computing Survey, Vol. 28*, páginas 241–273.

Sebring, M., Shellhouse, E., Hanna, M., e Whitehurst, R. (1988). Expert Systems in Intrusion Detection: A case study. Em *Proceedings of the Eleventh National Computer Security Conference, Washington*, páginas 74–81.

Stallings, W. (1994). Pretty Good Privacy. Em *ConneXions, Vol 8, Issue 12*, páginas 2–11.

Stallings, W. (2000). *Network Security Essentials*. Prentice Hall.

Staniford-Chen, S. (1998). Common Intrusion Detection Framework (CIDF). URL `http://seclab.cs.ucdavis.edu/cidf/`.

Stoneburner, G., Goguen, A., e Ferringa, A. (2001). *Risk Management Guide*. NIST Special Publication 800-30.

Tartanoglu, F., Issarny, V., Romanovsky, A., e Levy, N. (2002). Dependability in the Web Service Architecture. Em *Proceedings of the ICSE Workshop on Architecting Dependable Systems*, páginas 89–108.

Teo, L., Zheng, Y., e Ahn, G. (2003). Intrusion Detection Force: An infrastructure for internet-scale intrusion detection. Em *Proceedings of the first IEEE International Workshop on Information Assurance*.

Vigna, G., Eckmann, S., e Kemmerer, R. (2000). The STAT tool suite. Em *Proceedings of DARPA Information Survivability Conference and Exposition, IEEE Press, Los Alamitos*, páginas 46–55.

Wahbe, R., Lucco, S., Anderson, T., e Graham, S. (1993). Efficient Software-Based Fault Isolation. Em *Proceedings 14th ACM Symposium on Operating Systems Principles*, páginas 203–216. ACM.

Wood, M. (2002). Intrusion Detection Message Exchange Requirements. URL `http://www.ietf.org/internet-drafts/draft-ietf-idwg-requirements-10.txt%`.

Yegneswaran, V., Barford, P., e Jha, S. (2004). Global Intrusion Detection in the DOMINO Overlay System. Em *Network and Distributed System Security Simposium*.

Zaraska, K. (2003). Prelude IDS: Current state and development perspectives. URL `http://www.prelude-ids.org/download/misc/pingwinaria/2003/paper.pdf`.

Zwicky, E. D. e Cooper, S. (2000). *Building Internet Firewalls*. O'Reilly.